

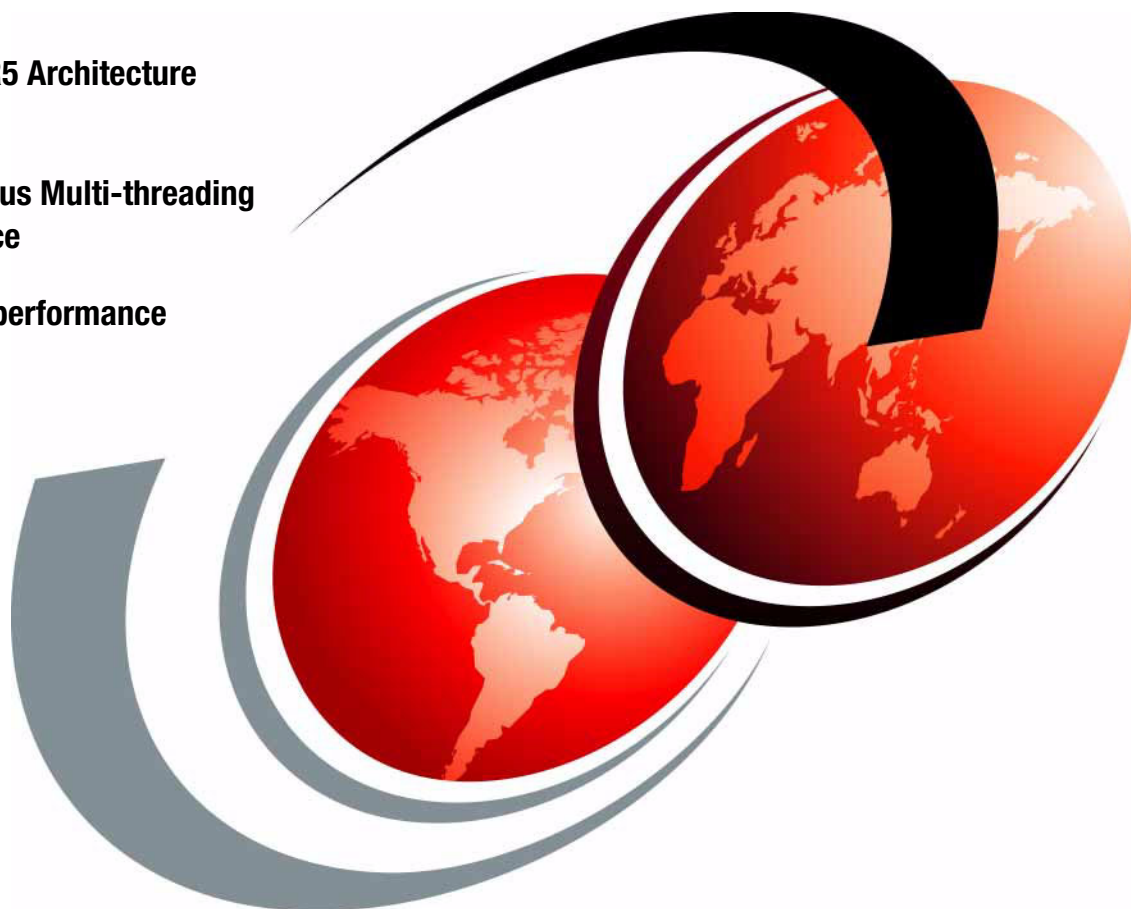
Advanced POWER Virtualization on IBM *e*server p5 Servers

Architecture and Performance Considerations

The POWER5 Architecture

Simultaneous Multi-threading
Performance

Virtual I/O performance





International Technical Support Organization

**Architecture and Performance Considerations in
POWER5 Virtualization**

December 2004

Note: Before using this information and the product it supports, read the information in “Notices” on page xv.

First Edition (December 2004)

This edition applies to the POWER5 architecture, the IBM @server p5 systems and Version 5, Release 3, of the AIX operating system.

This document created or updated on December 10, 2004.

© Copyright International Business Machines Corporation 2004. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

| | |
|---|-------|
| Figures | ix |
| Tables | xiii |
| Notices | xv |
| Trademarks | xvi |
| Preface | xvii |
| The specialists that wrote this redbook | xviii |
| Become a published author | xxii |
| Comments welcome | xxii |
| Part 1. General Concepts and Terminology | 1 |
| Chapter 1. Introduction | 1 |
| 1.1 Understanding performance | 2 |
| 1.2 System performance | 3 |
| 1.3 Applications performance | 5 |
| 1.4 Performance on a system with virtual features | 5 |
| Part 2. IBM @server p5 Architecture | 7 |
| Chapter 2. POWER5 Architecture | 9 |
| 2.1 Introduction | 10 |
| 2.1.1 Chip design | 12 |
| 2.1.2 POWER5 instruction pipelines | 13 |
| 2.1.3 L1 caches | 19 |
| 2.1.4 L2 cache | 20 |
| 2.1.5 L3 cache | 22 |
| 2.1.6 Summary of caches on POWER5 | 25 |
| 2.1.7 Address translation resources | 25 |
| 2.2 Enhanced SMT features | 27 |
| 2.2.1 Dynamic resource balancing (DRB) | 27 |
| 2.2.2 Adjustable thread priorities | 28 |
| 2.3 Dynamic power management | 31 |
| 2.4 Large POWER5 SMPs | 32 |
| Chapter 3. POWER Hypervisor | 39 |
| 3.0.1 POWER Hypervisor Support | 41 |
| 3.0.2 Hypervisor Call Functions | 43 |

| | | |
|-------------------|---|-----------|
| 3.0.3 | Micro-Partitioning Logical Partition Hypervisor Extensions | 49 |
| 3.0.4 | POWER Hypervisor Design | 52 |
| 3.0.5 | Performance Considerations | 55 |
| Chapter 4. | Operating system support | 59 |
| 4.1 | AIX 5L Version 5.3 | 60 |
| 4.1.1 | Introduction | 60 |
| 4.1.2 | Processors | 60 |
| 4.1.3 | Dynamic re-configuration | 66 |
| 4.1.4 | Existing performance commands enhancement | 67 |
| 4.1.5 | New performance commands | 75 |
| 4.1.6 | Paging space | 78 |
| 4.1.7 | Logical Volume Manager (LVM) | 79 |
| 4.1.8 | Virtual Local Area Network (VLAN) | 82 |
| 4.1.9 | EtherChannel | 83 |
| 4.1.10 | Partition Load Manager (PLM) | 83 |
| 4.2 | Linux on POWER | 85 |
| 4.3 | Sample level 2 “n.n” chapter heading (Head 1) new page | 86 |
| 4.4 | Sample level 2 “n.n” chapter heading (Head 2) | 86 |
| 4.4.1 | Sample level 3 “n.n.n” chapter heading (Head 3) | 86 |
| Part 3. | Features and capabilities | 87 |
| Chapter 5. | Simultaneous Multi-Threading (SMT) | 89 |
| 5.1 | What is SMT? | 90 |
| 5.2 | POWER5 SMT implementation | 92 |
| 5.2.1 | Resources for SMT | 93 |
| 5.2.2 | Dynamic resource balancing | 93 |
| 5.2.3 | Adjustable thread priorities | 94 |
| 5.3 | Software considerations for SMT | 98 |
| 5.3.1 | Snooze and snooze delay | 98 |
| 5.3.2 | Process accounting | 99 |
| 5.3.3 | Processor utilization | 100 |
| 5.3.4 | SMT aware scheduling | 100 |
| 5.3.5 | Interrupts | 101 |
| 5.3.6 | Effective use of adjustable thread priorities | 101 |
| 5.3.7 | Thread priorities on AIX | 102 |
| 5.3.8 | Thread priorities on Linux | 103 |
| 5.4 | Performance considerations | 103 |
| 5.4.1 | Cache effects | 106 |
| 5.4.2 | Exploitation of SMT | 107 |
| 5.5 | Exploitation of SMT using scientific and engineering applications | 107 |
| 5.5.1 | Life sciences applications | 108 |
| 5.5.2 | CAE applications | 110 |

| | |
|---|------------|
| 5.5.3 SMT benchmarks for sizing and capacity planing | 110 |
| 5.5.4 SMT exploitation via multi-processor benchmarks | 110 |
| 5.5.5 SMT exploitation via throughput benchmarks | 115 |
| 5.5.6 SMT exploitation on production system | 122 |
| Chapter 6. Micro-Partitioning | 125 |
| 6.1 Partitioning on the IBM @server p5. | 126 |
| 6.2 Micro-Partitioning Implementation. | 128 |
| 6.2.1 Implementation of Virtual Processor Dispatch | 129 |
| 6.2.2 Types of Micro-Partitioning | 136 |
| 6.2.3 Phantom Interrupt | 139 |
| 6.2.4 Configuring Micro-Partition on HMC | 142 |
| 6.2.5 Typical usage of Micro-Partitions | 144 |
| 6.3 Performance considerations | 145 |
| 6.3.1 Micro-Partitioning considerations | 145 |
| 6.3.2 Simultaneous Multithreading and Micro-Partitioning. | 146 |
| 6.3.3 Cache architecture and number of virtual processors. | 148 |
| 6.3.4 SMP locking and number of virtual processors. | 152 |
| 6.3.5 Memory affinity considerations | 158 |
| 6.3.6 Idle partition performance impact | 158 |
| 6.3.7 Application considerations for Micro-Partitions | 160 |
| 6.3.8 Guidelines for planning Micro-Partitions | 164 |
| Chapter 7. Virtual I/O | 173 |
| 7.1 Introduction | 174 |
| 7.1.1 Hypervisor support to Virtual I/O. | 174 |
| 7.1.2 Virtual I/O infrastructure | 176 |
| 7.1.3 Types of Connections | 178 |
| 7.1.4 The Virtual I/O Server | 180 |
| 7.2 Virtual Serial adapter. | 181 |
| 7.3 Virtual Ethernet | 181 |
| 7.3.1 Introduction | 182 |
| 7.3.2 General Concepts | 183 |
| 7.3.3 Hypervisor switch implementation. | 185 |
| 7.3.4 Performance Considerations and Measurements. | 188 |
| 7.3.5 Virtual LAN throughput at different processor entitlements. | 190 |
| 7.3.6 Comparing throughput of Virtual LAN to Gb ethernet. | 193 |
| 7.3.7 Virtual Ethernet in single-threaded and SMT mode | 199 |
| 7.3.8 Virtual Ethernet implementation guidelines. | 202 |
| 7.4 Shared Ethernet adapter functionality. | 203 |
| 7.4.1 Introduction | 203 |
| 7.4.2 Performance measurements with the Virtual I/O Server. | 205 |
| 7.4.3 Throughput and processor utilization | 205 |

| | |
|--|------------|
| 7.4.4 Virtual I/O Server request/response time and latency. | 209 |
| 7.4.5 Implementation Rules of Thumb | 213 |
| 7.4.6 Basis for finer Shared Ethernet Adapter I/O server sizing | 215 |
| 7.4.7 Turning threading of the shared Ethernet adapter on or off | 223 |
| 7.5 Virtual SCSI. | 225 |
| 7.5.1 Virtual SCSI communication basic concepts. | 226 |
| 7.5.2 Server Partition | 229 |
| 7.5.3 Client Partition. | 229 |
| 7.5.4 Emulated DASD | 230 |
| 7.5.5 Interpartition Communication | 231 |
| 7.5.6 Disks considerations | 236 |
| 7.5.7 Redundant Configurations | 236 |
| 7.5.8 Performance Considerations. | 238 |
| 7.5.9 VSCSI Performance Characteristics. | 240 |
| 7.5.10 VSCSI Server Sizing. | 244 |
| 7.6 Virtual SCSI. | 249 |
| 7.6.1 Virtual SCSI Structure and Concepts | 250 |
| 7.6.2 Virtual SCSI Model Overview | 255 |
| 7.6.3 Performance Considerations. | 261 |
| 7.6.4 VSCSI Performance Characteristics. | 263 |
| Part 4. Performance monitoring and management | 271 |
| Chapter 8. POWER5 system performance | 273 |
| 8.1 Performance commands. | 274 |
| 8.1.1 lparstat command | 274 |
| 8.1.2 mpstat command. | 281 |
| 8.1.3 vmstat command. | 285 |
| 8.1.4 iostat command. | 287 |
| 8.1.5 sar command | 289 |
| 8.1.6 topas command. | 292 |
| 8.1.7 xmperf command | 295 |
| 8.2 Performance tuning approach. | 300 |
| 8.2.1 Global performance analysis. | 300 |
| 8.2.2 CPU analysis. | 306 |
| 8.2.3 Memory analysis | 311 |
| 8.2.4 Disk I/O analysis | 313 |
| 8.2.5 Network I/O analysis | 322 |
| Chapter 9. Partition Load Manager (PLM). | 329 |
| 9.1 When and how should I use PLM | 330 |
| 9.1.1 PLM and other load balancing tools. | 330 |
| 9.1.2 When to use PLM | 332 |
| 9.1.3 How to deploy PLM. | 337 |

| | |
|---|------------|
| 9.2 More about PLM installation and setup | 339 |
| 9.2.1 Overview of PLM behavior | 339 |
| 9.2.2 Management versus monitoring modes | 340 |
| 9.2.3 Configuration file and tunables | 341 |
| 9.3 Managing and monitoring with PLM | 346 |
| 9.3.1 Managing multiple partitions CEC from the same PLM manager. | 346 |
| 9.3.2 Extra tips about the xlplm command. | 348 |
| 9.3.3 Examples of PLM commands output | 349 |
| 9.4 PLM performance impact | 352 |
| 9.4.1 PLM resource requirements | 352 |
| 9.4.2 PLM impact on managed partitions. | 353 |
| Chapter 10. Applications Tuning | 357 |
| 10.1 Performance bottlenecks identification | 358 |
| 10.1.1 Time commands, time utilities and time routines | 360 |
| 10.2 Tuning applications using only the compiler | 364 |
| 10.2.1 Compiler brief overview. | 364 |
| 10.2.2 Most commonly used flags | 367 |
| 10.2.3 Compiler directives for performance | 373 |
| 10.2.4 POWER5 compiler features | 380 |
| 10.3 Profiling applications | 383 |
| 10.3.1 Hardware performance monitor | 384 |
| 10.3.2 Profiling utilities | 390 |
| 10.4 Memory management | 400 |
| 10.4.1 Memory Hierarchy. | 401 |
| 10.4.2 L1, L2, and L3 caches. | 401 |
| 10.4.3 Translation Lookaside Buffer (TLB) | 401 |
| 10.5 Optimization of critical sections in the code | 402 |
| 10.5.1 Array Optimization. | 403 |
| 10.5.2 Loop Optimization | 404 |
| 10.5.3 Tuning for Arithmetic Operations | 411 |
| 10.5.4 Multiple Loads and Prefetch | 411 |
| 10.5.5 Divides | 411 |
| 10.5.6 Floating Point-to-Integer Conversion | 412 |
| 10.6 Optimized Libraries | 412 |
| 10.6.1 MASS Library | 412 |
| 10.6.2 ESSL Library | 415 |
| 10.7 Parallel Programming for Performance | 418 |
| 10.7.1 General Concepts | 418 |
| 10.7.2 Shared-Memory Parallel Performance | 419 |
| 10.7.3 Distributed-Memory Parallel Performance | 419 |
| Chapter 11. Sizing and Capacity Planning | 421 |

- 11.1 Sample level 2 “n.n” chapter heading (Head 1) new page 422
 - 11.2 Sample level 2 “n.n” chapter heading (Head 2). 422
 - 11.2.1 Sample level 3 “n.n.n” chapter heading (Head 3) 422
- Part 5. Appendixes 423**
 - Related publications 425**
 - IBM Redbooks 425
 - Other publications 426
 - Online resources 428
 - New AIX/pSeries doc in eserver information center 429
 - How to get IBM Redbooks 429
 - Help from IBM 429
 - Index 431**

Figures

| | | |
|------|--|-----|
| 2-1 | POWER5 processor chip | 11 |
| 2-2 | High level structure of POWER5 | 12 |
| 2-3 | POWER5 instruction pipeline. | 14 |
| 2-4 | POWER5 instruction and data flow | 18 |
| 2-5 | Cache indexing bits | 21 |
| 2-6 | L2 cache organization | 21 |
| 2-7 | L3 cache high-level design | 22 |
| 2-8 | L3 cache organization | 23 |
| 2-9 | Comparison between POWER4 and POWER5. | 24 |
| 2-10 | Thread priority pairs vs. instructions executed per second | 30 |
| 2-11 | POWER5 photos using thermal sensitive camera. | 32 |
| 2-12 | POWER5 DCM | 33 |
| 2-13 | DCM interconnection for a 16 way SMP | 34 |
| 2-14 | Picture of a POWER5 DCM | 34 |
| 2-15 | Logical view of the POWER5 MCM | 35 |
| 2-16 | POWER5 MCM | 36 |
| 2-17 | POWER5 book. | 37 |
| 2-18 | MCMs interconnected to make a 64 way SMP | 37 |
| 3-1 | POWER Hypervisor | 40 |
| 3-2 | POWER Hypervisor on AIX 5L and Linux | 41 |
| 3-3 | lparstat -H command output. | 49 |
| 3-4 | Current memory available for partition usage using HMC | 51 |
| 3-5 | lparstat -h 1 16 command output | 56 |
| 3-6 | lparstat -i command output | 57 |
| 4-1 | Logical versus physical processors | 62 |
| 4-2 | 3dmon monitoring two LPARs | 73 |
| 4-3 | xmperf Mini Monitor | 74 |
| 4-4 | jtopas default display | 75 |
| 4-5 | procmon partition wide metrics | 77 |
| 4-6 | procmon sorted list of processes | 78 |
| 5-1 | Different Multi-Threading models. | 92 |
| 5-2 | Thread priority pairs vs. instructions executed per second | 97 |
| 5-3 | SMT gains for various workloads. | 107 |
| 5-4 | Gaussian03 parallel runs using ST and SMT modes. | 112 |
| 5-5 | AMBER7 paprallel runs using ST and SMT modes. | 113 |
| 5-6 | BLAST paprallel runs using ST and SMT modes. | 114 |
| 5-7 | Gaussian03 Series of throughput benchmarks using ST and SMT modes. | 116 |

| | | |
|------|---|-----|
| 5-8 | Performance advantage of the SMT mode over the ST mode.. ST was used as the baseline for all the measurements and it was set to zero. | 117 |
| 5-9 | AMBER7 Series of throughput benchmarks using ST and SMT modes. | 118 |
| 5-10 | Performance advantage of the SMT mode over the ST mode. ST was used as the baseline for all the measurements and it was set to zero. | 119 |
| 5-11 | BLAST series of throughput benchmarks using ST and SMT modes.. | 120 |
| 5-12 | Performance advantage of the SMT mode over the ST mode. ST was used as the baseline for all the measurements and it was set to zero. | 121 |
| 5-13 | A balance throughput benchmark. | 123 |
| 6-1 | System with dedicated processor partition and Micro-Partitions | 127 |
| 6-2 | Dispatch wheel. | 130 |
| 6-3 | Dispatch wheel for SMT-enabled processors | 131 |
| 6-4 | Example on Virtual Processor Dispatch. | 136 |
| 6-5 | Capped partition. | 137 |
| 6-6 | Uncapped partition. | 138 |
| 6-7 | Uncapped partition example | 139 |
| 6-8 | Interrupt Processing on POWER5 | 141 |
| 6-9 | Effect of an SMT idle thread on Micro-Partitions | 148 |
| 6-10 | Affinity between virtual and physical processors | 150 |
| 6-11 | Impact on cache due to number of virtual processors | 151 |
| 6-12 | Measurements of cache effects in different partitions | 152 |
| 6-13 | The effect of multiple virtual processors in overall performance | 158 |
| 6-14 | Uncapped partition performance example | 159 |
| 6-15 | Dispatch latencies for virtual processors | 160 |
| 6-16 | User distribution during the day in an ERP application server. | 162 |
| 6-17 | Processor utilization by the ERP application server | 162 |
| 6-18 | Processor utilization between five partitions with different workloads . | 163 |
| 6-19 | Processing resource consumption for three different applications . . | 170 |
| 6-20 | Application resource consumption example | 171 |
| 7-1 | Virtual I/O provided by Hypervisor | 175 |
| 7-2 | Hypervisor simulated adapters | 176 |
| 7-3 | Virtual Ethernet | 182 |
| 7-4 | Virtual and local adapters on one partition. | 183 |
| 7-5 | TCP/IP Suite of protocols. | 184 |
| 7-6 | Example of two VLANs in a Virtual Ethernet Environment | 185 |
| 7-7 | Flow chart of Virtual Ethernet. | 187 |
| 7-8 | Processor entitlements and MTU sizes | 190 |
| 7-9 | Throughput vs. CPU entitlements, MTU size=1500. | 191 |
| 7-10 | Throughput vs. CPU entitlements, MTU size=9000. | 191 |
| 7-11 | Throughput vs. CPU entitlements, MTU size=65394. | 192 |
| 7-12 | Throughput normalized to 0.1 Entitlement. | 193 |
| 7-13 | VLAN to VLAN performance | 194 |
| 7-14 | Gb Ethernet to Gb Ethernet | 194 |

| | | |
|------|--|-----|
| 7-15 | Throughput of Virtual LAN and Gb ethernet with TCP_STREAM | 195 |
| 7-16 | Processor consumption with TCP_STREAM, simplex mode. | 196 |
| 7-17 | Processor consumption with TCP_STREAM, duplex mode | 197 |
| 7-18 | Transaction rate at different MTU sizes and 1/20 sessions. | 198 |
| 7-19 | Latency at different MTU sizes and 1/20 sessions | 198 |
| 7-20 | Performance gain with SMT, TCP_STREAM | 200 |
| 7-21 | Performance gain with SMT, TCP_RR | 201 |
| 7-22 | VIO Server bridging between an external network and internal VLANs | 203 |
| 7-23 | Sharing a (physical) ethernet adapter on OSI-Layers | 204 |
| 7-24 | Setup for I/O Server performance measurements | 206 |
| 7-25 | Throughput of the Virtual I/O Server | 207 |
| 7-26 | Processor utilization of the Virtual I/O Server | 208 |
| 7-27 | Dedicated connection between a partition and an external server. . | 209 |
| 7-28 | Transaction rates, TCP_RR, 1 session | 210 |
| 7-29 | Transaction rates, TCP_RR, 20 sessions | 211 |
| 7-30 | Latencies, TCP_RR, 1 session | 212 |
| 7-31 | Latency, TCP_RR, 20 sessions | 212 |
| 7-32 | Number of processor needed to support network traffic | 222 |
| 7-33 | Virtual I/O Server and Client Partitions | 226 |
| 7-34 | Server SCSI adapter possibly connecting to 2 client SCSI adapters. . | 228 |
| 7-35 | Reliable Command / Response Transport and LRDMA | 229 |
| 7-36 | Volume group on Virtual I/O Server | 231 |
| 7-37 | Logical Remote Direct Memory Access | 233 |
| 7-38 | Using LVM mirroring for virtual SCSI | 237 |
| 7-39 | Additional Latency per I/O operation | 241 |
| 7-40 | Native I/O average response time | 242 |
| 7-41 | Native to VSCSI I/O comparison | 243 |
| 7-42 | Native/VSCSI bandwidth scaling | 244 |
| 7-43 | Comparison of native I/O to VSCSI | 246 |
| 7-44 | AIX 5L Server and Client Partitions | 250 |
| 7-45 | Reliable Command / Response Transport and LRDMA | 251 |
| 7-46 | Logical Remote Direct Memory Access | 253 |
| 7-47 | Volume group on Virtual I/O Server | 258 |
| 7-48 | Using LVM mirroring for virtual SCSI | 260 |
| 7-49 | Latency overhead per I/O operation | 264 |
| 7-50 | Native I/O average response time | 265 |
| 7-51 | Native to VSCSI I/O comparison | 266 |
| 7-52 | Native/VSCSI bandwidth scaling | 266 |
| 7-53 | Comparison of native I/O to VSCSI | 268 |
| 8-1 | “shared processor utilization authority” activation. | 278 |
| 8-2 | xmperf - Local System Monitor | 296 |
| 8-3 | Physical and logical processors usage | 298 |
| 8-4 | xmperf - process redispatch. | 299 |

| | | |
|-------|---|------------|
| 8-5 | Global performance diagram | 301 |
| 8-6 | CPU analysis diagram. | 306 |
| 8-7 | System properties - Processors. | 308 |
| 8-8 | Allow idle processors to be shared | 311 |
| 8-9 | System properties - Memory | 313 |
| 8-10 | Disk I/O analysis diagram. | 314 |
| 8-11 | Virtual I/O adapters | 319 |
| 8-12 | Virtual SCSI Adapter Properties. | 320 |
| 8-13 | Network I/O analysis diagram. | 323 |
| 9-1 | PLM management using WebSM | 351 |
| 9-2 | PLM processor statistics using WebSM. | 351 |
| 9-3 | PLM memory statistics using WebSM | 352 |
| 9-4 | PLM impact on number of virtual processors. | 355 |
| 10-1 | This flowchart illustrates the first step in applications tuning. | 360 |
| 10-2 | IBM compiler architecture. | 365 |
| 10-3 | Inside TPO compile time. | 366 |
| 10-4 | Loop optimization overview in TPO. | 367 |
| 10-5 | xprofiler calling tree for the sander module. | 398 |
| 10-6 | xprofiler view of the time consuming routines. | 399 |
| 10-7 | Flat profile produced using xprofiler | 400 |
| 10-8 | Memory heirarchy | 401 |
| 10-9 | Blocking matrix diagram. | 406 |
| 10-10 | Graphical representation of the different techniques against peak performance.408 | |
| 10-11 | Matrix multiplication as a function of block size. | 410 |
| 10-12 | Large pages effect on performance for matrix multiplication. | 411 |
| 10-13 | CPU-bound code might benefit from optimized libraries.. . . . | 413 |
| 10-14 | CPU-bound code might benefit from optimized libraries.. . . . | 416 |
| 10-15 | DGEMM optimized routine in ESSL versus Fortran version. | 417 |

Tables

| | | |
|------|---|-----|
| 2-1 | Rename resources in the POWER5 core | 16 |
| 2-2 | Cache characteristics of the POWER5 processor | 25 |
| 2-3 | POWER5 thread priority levels | 28 |
| 2-4 | Effect of thread priorities on execution resource sharing. | 29 |
| 3-1 | Hypervisor calls | 45 |
| 4-1 | Maximum values for volume groups | 80 |
| 4-2 | LTG and stripe sizes | 80 |
| 5-1 | POWER5 thread priority levels | 95 |
| 5-2 | Effect of thread priorities on execution resource sharing. | 95 |
| 5-3 | Differences between POWER4 required to accomodate SMT on POWER5 104 | |
| 5-4 | Performance of parallel FLUENT testcase: FL5M3. | 115 |
| 5-5 | Throughput performance of serial FLUENT for testcase: FL5M3. | 122 |
| 6-1 | Micro-Partition definition: | 135 |
| 6-2 | An example of an ERP system requirements | 166 |
| 6-3 | Implementation with separate servers | 167 |
| 6-4 | Dedicated processor partitioning with 1.45GHz middle range server | 167 |
| 6-5 | Implementation with Micro-Partitioning | 168 |
| 6-6 | Peak consumption per partition | 171 |
| 7-1 | Required attributes of the /vdevice node | 177 |
| 7-2 | Comparing TCE and RTCE | 179 |
| 7-3 | Network streaming rates | 213 |
| 7-4 | Single direction TCP Streaming rates | 217 |
| 7-5 | Two direction (duplex) TCP Streaming rates | 218 |
| 7-6 | Machine Cycles per Byte for TCP Streaming workload. | 220 |
| 7-7 | Machine Cycles per Byte for TCP Streaming workload. | 221 |
| 7-8 | Shared Ethernet with Threading option enabled | 221 |
| 7-9 | Shared Ethernet with Threading option disabled | 221 |
| 7-10 | I/O CPU cycles required for various block sizes | 245 |
| 7-11 | I/O CPU cycles required for various block sizes | 267 |
| 8-1 | Commands summary. | 274 |
| 8-2 | lparstat command summary. | 274 |
| 8-3 | mpstat command summary | 281 |
| 8-4 | vmstat command summary | 285 |
| 8-5 | iostat command summary | 287 |
| 8-6 | sar command summary | 289 |
| 8-7 | topas command summary | 292 |
| 8-8 | xmperf command summary | 295 |

9-1 Partitions resources definition on the HMC 333

9-2 Resources allocations for time period 1 333

10-1 time description fields. 361

10-2 -qipa suboptions. 371

10-3 New or changed options and suboptions. 372

10-4 372

10-5 Assertive, loop optimzation and hardware-specific directives. 374

10-6 hpmcount flags and description. 385

10-7 Selected set of groups for applications tuning. 386

10-8 Optimal unrolling levels for some IBM POWER Series machines . . . 406

10-9 Performance of matrices with different unrolling levels. 407

10-10 Blocking and large pages effect on matrix multiplication. 408

10-11 AMBER7 performance with the **sqr**t vector MASS routine. 415

10-12 DGEMM routine optimized in the ESSL library. 417

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:


This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX 5L™
AIX®
@server®
@server®
HyperSockets™
ibm.com®
IBM®
iSeries™

Micro-Partitioning™
OS/400®
Perform®
POWER4™
POWER5™
PowerPC®
PR/SM™
pSeries®

Redbooks®
Redbooks (logo) ™
RS/6000®
Tivoli®
TotalStorage®
Virtualization Engine™
zSeries®

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Preface

This redbook provides an insight into the performance considerations of Advanced Virtualization on the IBM eserver p5 platforms. It discusses the major hardware, software, benchmarks, and various tools that are available.

This redbook is suitable for professionals who want to acquire a better understanding of the POWER5 architecture and micro-partitioning that is supported by the IBM eserver p5 platforms. It targets clients, sales and marketing professionals, technical support professionals, and IBM Business Partners.

Inside this redbook, you will find:

- ▶ A description of the POWER5 microprocessor architecture.
- ▶ A description of the POWER Hypervisor.
- ▶ An informative review and performance aspects of micro-partitioning.
- ▶ A discussion of AIX and Linux support.
- ▶ A review of changes to existing performance tools and a look at some new tools.
- ▶ A description and performance aspects of Simultaneous Multi-Threading (SMT).
- ▶ A description and performance issues related to virtualization of Ethernet and SCSI.

This redbook is intended as an additional source of information that, together with existing sources referenced throughout this document, enhances your knowledge of IBM solutions for the UNIX marketplace. It does not replace the latest marketing materials and tools.

The specialists that wrote this redbook

This redbook was the result of two separate residencies and was made up of specialists from around the world working at the International Technical Support Organization, Austin Center.

Ben Gibbs is a Senior Consulting Engineer with Technonics, Inc. (<http://www.technonics.com>) in Austin, Texas. He has over 20 years of experience with UNIX-based operating systems. He started working with the AIX operating system in November of 1989. His areas of expertise include performance analysis and tuning, operating system internals, and device driver development for the AIX operating system. He was the project leader for this IBM Redbook.

Dr. Balaji Atyam is a Senior Software Engineer in the Systems and Technology Group since 2000. His responsibilities are porting, benchmarking, performance tuning, parallel programming and technical consulting services to key Independent Software Vendors (ISV) in the area of High Performance Computing on IBM eServers. He received his Ph.D. in Applied Mathematics from Indian Institute of Technology, Roorkee, India. He was a Scientist/Engineer in Indian Space Research Organization (ISRO) prior to join IBM.

Frank Berres is a Senior Architect with SerCon GmbH in Germany. SerCon is an IBM Company, that is assigned to IBM Business Consulting Services (BCS). Frank has over 5 years of experience in IT consulting and support on AIX-based systems. He holds a degree in Electrical Engineering from the University of Applied Sciences, Bingen, Germany.

Bruno Blanchard is a Certified IT Specialist working for IBM France in the IGS Strategy Design and Authority team in La Gaude. He has been with IBM since 1983, starting as a System Engineer for VM. He has started to work with AIX in 1988, using AIX on the IBM RT/PC, the PS/2, the RS/6000, the SP/2 and the pSeries. He has written many redbooks, and is currently working as an Architect in projects deploying eServer cluster 1600 and pSeries servers infrastructures, for server consolidation and on-Demand environments.

Lancelot Castillo is an IBM Certified Advanced Technical Expert – pSeries and AIX 5L. He works as a pSeries Product Manager at Questronix Corporation, an IBM Business Partner in Philippines. He has over six years of experience in AIX and pSeries Servers. He holds a Bachelor's degree in Electronics and Communications Engineering from Mapua Institute of Technology. His areas of expertise include AIX performance tuning and sizing, RS/6000 SP and HACMP.

Pedro Coelho is an IT Specialist with IBM Global Services in Portugal. He has five years of experience in AIX and Linux in the area of post-sales support and services. He holds a degree in Computer Science from COCITE, Lisbon,

Portugal. His areas of expertise include HACMP and performance analysis and tuning. He is also working with IBM Learning Services teaching beginners and advanced classes on AIX and Linux.

Nicolas Guerin is an IT Specialist working for IBM France in La Gaude. He has eight years of experience in the Information Technology field. His areas of expertise include AIX, system performance and tuning, HACMP, pSeries, SP, ESS and SAN. He has been working for IBM for ten years. He is an IBM Certified Advanced Technical Expert - pSeries and AIX 5L. This is his second redbook.

Lei Liu is a Senior IT Specialist working for IBM China in Technical Sales Support Center in Beijing. She has over 13 years of working experiences on UNIX systems. As ATS she is responsible for large account support for telecom customers, including both pre-sale and post-sale technical support. She joined IBM in 1998, her area of expertise includes AIX, system performance analysis and tuning and HACMP. She is an IBM Certified Advanced Technical Expert - pSeries and AIX 5L.

Cesar Diniz Maciel is a Certified IT Specialist with the pSeries division in IBM Brazil. He has 9 years of experience on AIX and pSeries systems. He holds a degree in Electrical Engineering from UFMG, Belo Horizonte. He works as a pre-sales technical support in Brazil for pSeries, AIX and Linux on pSeries. He is also a Regional Designated Specialist for Latin America for High End systems and Linux on pSeries. He works for IBM since 1996.

Ravikiran Thirumalai is a Software Engineer at IBM India Software Labs. He has been in the IT industry for over 6 years. He holds a Bachelor's degree in Electrical and Electronics engineering from Bangalore University and a MS in Software Systems from BITS Pilani. He works for the IBM Linux Technology Center as a kernel developer for the baseos team. His main areas of interest in the kernel are SMP scalability, locking algorithms, lockfree techniques and the VFS.

Dr. Carlos Sosa is a Senior Technical Staff Member in the Systems and Technology Group, where he has been a member of IBM's Chemistry and Life Sciences high-performance effort since 2001. For the last 18 years, he has focused on scientific applications with emphasis in Life Sciences, parallel programming, benchmarking, and performance tuning. He received a Ph.D. degree in Physical Chemistry from Wayne State University. He completed his post-doctoral work at the Pacific Northwest National Laboratory. His area of interest is in future POWER series architectures and cellular molecular biology.

Thanks to the following people for their contributions to this project:

Dr. Joel Tendler
IBM Austin

Bret Olszewski
IBM Austin

David Chisholm
IBM Mount Laurel

Harry Mathis

IBM Austin

Frank O'Connell

IBM Austin

Sujatha Kashyap

IBM Austin

Arie Tal

IBM Toronto

David Klepacki

IBM Watson

David Wootton

IBM Poughkeepsie

Bruce D. Hurley

IBM Austin

Matthew Cali

IBM Raleigh

Hari Reddy

IBM Dallas

Suresh Warriar

IBM Austin

Jorge D Rodriguez
IBM Austin

Luc Smolders
IBM Austin

Luke Browning
IBM Austin

Octavian F. Herescu
IBM Austin

Kiet H. Lam
IBM Austin

Herman D. Dierks
IBM Austin

Sergio Reyes
IBM Austin

Tommy Todd
IBM Atlanta

Stephen Nasypany
IBM Austin

Tony Ramirez
IBM Austin

Larry Brenner
IBM Austin

Sujatha Kashyap
IBM Austin

Bob Kovacz
IBM Austin

Mysore Srinivas
IBM Austin

Claudio Garrido
IBM Brasil

Leonardo Vidal
IBM Brasil

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an Internet note to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. JN9B Building 003 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493



Part 1

General Concepts and Terminology

Note to Author: Optionally, describe the book part here. If you are not using Part files, you need to restart the page numbering in the first chapter file of your book:

- ▶ In FrameMaker 6.0: **Open .book > select first chapter> Format > Document > Numbering > Page “tab” > select First Page # radio button and set Page # to 1 > Set**
 - Now delete the three Part files (p01, p02 and p03) from your book:
Open .book > select a part file > Edit > Delete File from Book

In this part we introduce/provide/describe/discuss...



Introduction

The IBM *@server* p5 family of servers include powerful new capabilities such as the partitioning of processors to 1/10th of a processor, share processor resources in a pool to drive up utilization, share physical disk storage and communications adapters between partitions and take advantage of cross partition workload management, to mention a few.

In the subsequent chapters you will find an in-depth discussion about each component that makes up the new IBM capabilities available on POWER5 family of systems, such as, POWER Hypervisor, Simultaneous Multi-Threat (SMT), Micro-Partitioning, Virtual LAN, and Virtual I/O.

With the advent of this new technology or new functionality, our traditional concepts of system and applications performance need to accommodate this new virtual dimension. In addition to defining and explaining these concepts, this book also covers traditional performance issues as well as performance as a function of a system environment with virtual capabilities.

The following reference is recommended for the reader that is looking for introductory material on IBM concepts on virtualization: Advanced POWER Virtualization on IBM *@server* p5 Servers Introduction and Basic Configuration, SG24-7940.

1.1 Understanding performance

Technological improvements in microprocessors, disks, and networking equipment have dramatically changed the landscape of server computing. While those improvements have more often than not reduced the incidence of performance problems in customer environments, they have also increased the capabilities of systems such that more complex problems may be solved. Thus, performance tuning has tended to change in nature from simple hardware and software bottleneck analysis toward evaluation of more complex interactions. Performance evaluation and tuning of complex systems requires discipline and exactness. Frequently, the solution to a problem is not obvious to the casual observer. Often, the steps along the journey toward the solution may seem inconclusive or even counter-productive. But, with a systematic rigor, nearly every bottleneck can be alleviated in some way. To help the reader achieve the goal of making system tuning rewarding and beneficial, we have dedicated one chapter to provide you with tools to help in the effort.

In addition, it is important to note that performance tuning can be subdivided into: *system tuning* and *applications tuning*. In the following two sections we describe both techniques. However, here we mention that the objectives of these two tuning efforts are very different, one relies on system parameters that can actually be modified to provide faster throughput measurement; where throughput consists of the amount of work performed over a period of time. This normally corresponds to a series of identical or different jobs running simultaneously and competing for the same system resources. The other one looks at the source code for a particular application and requires most-likely tailoring of certain subroutines to a particular architecture or common architectural features.

An IBM @server p5 system is subjected to various types of loads. The load can vary widely depending on the number of applications used and the type of applications being run. Obviously the number of loads and the type of applications being run will vary widely over the period of the server's working life. Consequently, changes have to be made to the server's hardware and software setup to accommodate these changing conditions. Applications require tuning as well.

System administrators often refer to any degradation of service as a *bottleneck* in the server system, but users who are less aware might simply consider the server to be running slow or that something is wrong. Bottlenecks need to be understood and compensated for, if the system administrators are to keep the users satisfied with performance. Similarly, programmers need to identify bottlenecks within the source code of certain applications that form part of the system load.

1.2 System performance

Within a server there are limited resources that can affect the performance of a given system. Each of these resources work together hand-in-hand and are capable of influencing the behavior of one another. If performance modifications are not carefully administered then the overall effect could be a deterioration of server performance.

Here we distinguish two types of resources:

- ▶ Physical resources
- ▶ Logical resources

In this book we refer to physical resources what comprises the fundamental subsystems found within a server:

- ▶ Processors
- ▶ Memory
- ▶ Disk I/O
- ▶ Networking

On the other hand, logical resources are software-oriented, such as operating system locks, resource limits (e.g., mbufs), applications locks, application resource limits, etc. Tuning tends to go from the simplest solution (e.g., parameter tweaking), toward more complex solutions (applications tuning, recompiling, OS tuning, changing hardware, etc.)

Efficiency of the operating system will maximize the hardware performance. The operating system in this case is AIX Version 5.3. As server performance is distributed throughout each server component, it is essential to identify the most important factors or bottlenecks that will affect the performance for a particular activity.

Detecting the bottleneck within a server system depends on a range of factors such as:

- ▶ Configuration of the server hardware
- ▶ Software application(s) workload
- ▶ Configuration parameters of the operating system
- ▶ Network configuration and topology

File servers need fast network adapters and fast disk subsystems. In contrast, database server environments typically produce high processor and disk utilization, requiring fast processors or multiple processors and fast disk

subsystems. Both file and database servers require large amounts of memory for caching by the operating system.

Traditionally there was a simplified approach to performance tuning. If the performance bottleneck is the processor then either a faster processor or more processors could be installed. An alternative to processor upgrade is to off-load processing requirements by using workload management techniques. If the bottleneck is memory then additional memory could be installed. Memory bottlenecks often resulted in excessive disk I/O as a result of paging (swapping) between paging space and memory. If the bottleneck is the disk subsystem then either additional disks and/or disk adapters can be installed. In addition, a specialized high performance disk subsystem could also be used. If the bottleneck is the network adapter then a faster network interface could be installed. Another optimization technique that can be employed is to utilize multiple network adapters in the server increasing throughput onto one or multiple segments.

Before any tuning is actually performed it is worth understanding the framework within which performance testing is done. A set of simple guidelines needed only be followed to assist in any type of performance analysis.

There are many trade-offs related to performance tuning that have to be considered. In order to choose the best set of options it is vital to ensure that there is a balance between them. The trade-offs are:

- ▶ Cost versus performance. There are situations where the only way performance can be improved is by using more or faster hardware while keeping in mind, “Does the additional cost result in a proportional increase in performance?”
- ▶ Conflicting performance requirements. When more than one application is used simultaneously, there may be conflicting performance requirements.
- ▶ Speed versus functionality. Here, for example, resources may be increased to improve a particular section, but serve as an overall detriment to the system. Using a methodical approach you can obtain improved server performance, such as by:
 - Understanding the factors which can affect server performance, for the specific server functional requirements and for the characteristics of the particular system
 - Measuring the current performance of the server
 - Identifying a performance bottleneck
 - Upgrading the component which is causing the bottleneck
 - Measuring the new performance of the server to check for improvement

Although we cover this material in this book, additional information may be found in the following reference: AIX 5L Performance Tools Handbook, SG24-6039.

1.3 Applications performance

Applications tuning or applications optimization requires careful analysis of the source code to try to tailor it to a particular hardware architecture. In other words, it is the goal of the programmer to make the application aware of the hardware features that are accessible to the application. For instance, we shall see in the applications tuning chapter how to re-write do loops to better take advantage of the L2 cache on pSeries architectures.

We shall see that in general, any tuning that we carry out at the application level will leverage systems with and without virtual environments. The following references cover this subject on POWER3 and POWER4:

- ▶ RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide, SG24-5155
- ▶ The POWER4 Processor Introduction and Tuning Guide, SG24-7041

1.4 Performance on a system with virtual features

The goal of virtualization is to allow the deployment of resources in a flexible manner. This flexibility allows best use of resources and, when correctly used, should improve performance and the end-user experience. But, virtualization alters the way we look at system performance. We still follow the same rules with respect to identifying existing or potential bottlenecks, but the remedy can be different and difficult to obtain. Virtualization is a flexible, resource model for the on demand world. The focus here is more on increasing resource utilization and to respond to changing workloads. Resources are dynamically allocated, including fractional, on an as needed basis. Capacity-on-Demand allows the allocation of additional resources as needed and Workload Management (WLM) allows the optimization of resources to respond to changing workloads.

In this book we cover the subject of virtualization with emphasis on performance. To address this topic and in an effort to make it useful to a wide number of audiences with different technical backgrounds, we start by providing the reader with the most fundamental building block, that is, an in-depth introduction to the POWER5 architecture. This is presented in chapter 2. Chapters 3 and 4 extend the notion of the IBM @server POWER5 architecture by introducing the POWER Hypervisor and the AIX and Linux operating systems.

As part of features and capabilities in the POWER5 family of systems, chapters 5, 6, and 7 cover Simultaneous Multi-Threading, Micro-Partitioning, and Virtual I/O, respectively. In chapter 8 we cover system performance and describe some of the tools available to monitor the system. Chapter 9 covers Partition Load Manager (PLM). Chapter 10 corresponds to a chapter that is devoted to looking at performance tuning but at the applications level. The last chapter in the book is a very important chapter for those of you that are involved in benchmarks and providing with solutions enablement.



Part 2

IBM @server p5 Architecture

Note to Author: Optionally, describe the book part here. If you are not using Part files, you need to restart the page numbering in the first chapter file of your book:

- ▶ In FrameMaker 6.0: **Open .book > select first chapter> Format > Document > Numbering > Page “tab” > select First Page # radio button and set Page # to 1 > Set**
 - Now delete the three Part files (p01, p02 and p03) from your book:
Open .book > select a part file > Edit > Delete File from Book

In this part we introduce/provide/describe/discuss...



POWER5 Architecture

This chapter describes the components that constitute a Micro-Partitioning environment. We briefly discuss the hardware, firmware and software players that make Micro-Partitioning possible. The following major topics are discussed in this chapter:

- ▶ The POWER5 processor
 - Instruction pipelines
 - L1, L2 and L3 caches

2.1 Introduction

The POWER5 processor is IBM's latest 64-bit implementation of the PowerPC AS architecture (Version 2.02). This dual core processor with *multithreading* technology, is fabricated using silicon-on-insulator (SOI) devices and copper interconnect. SOI technology is used to reduce the device capacitance and increase transistor performance. Wire resistance is lower in copper interconnects and results in reduced delays in wire dominated chip timing paths. The chip is implemented using 130 nm lithography with eight metal layers and a die that measures 389 mm². The chip is made up of 276 million transistors.

The POWER5 processor is binary compatible with all PowerPC and PowerPC AS application level code. The POWER5 has been designed for very high frequency operations with operating frequencies of up to 2.0 GHz. POWER5 consists of a deeply pipelined design with 16 stages for fixed-point register-to-register operations, 18 stages for most load and store operations (with L1 data cache hits), and 21 stages for most floating point operations.

The processor exhibits a speculative superscalar inner core organization with aggressive branch prediction, out-of-order issues, register renaming, large number of instructions in flight and fast selective flush of incorrect speculative fetched instructions and results. There has been a specific focus on storage latency management where the core can issue out-of-order load operations with support for up to eight outstanding L1 data cache line misses. There is hardware or software initiated instruction prefetching from the L2, L3 and memory along with hardware initiated data stream prefetching, and software instruction prefetching based on branch prediction hints.

The POWER5 architecture is an enhancement over the POWER4 architecture, but maintains binary and structural compatibility. The identical pipeline structure lets compiler optimizations targeted for POWER4 to work equally well on POWER5 based systems.

Each POWER5 processor core is designed to support both *Simultaneous Multithreading* (SMT) and single-threaded modes. Software (operating system using Hypervisor calls) can switch the processor from SMT mode to single-threaded mode.

The layout of the POWER5 processor is found in Figure 2-1 on page 11.

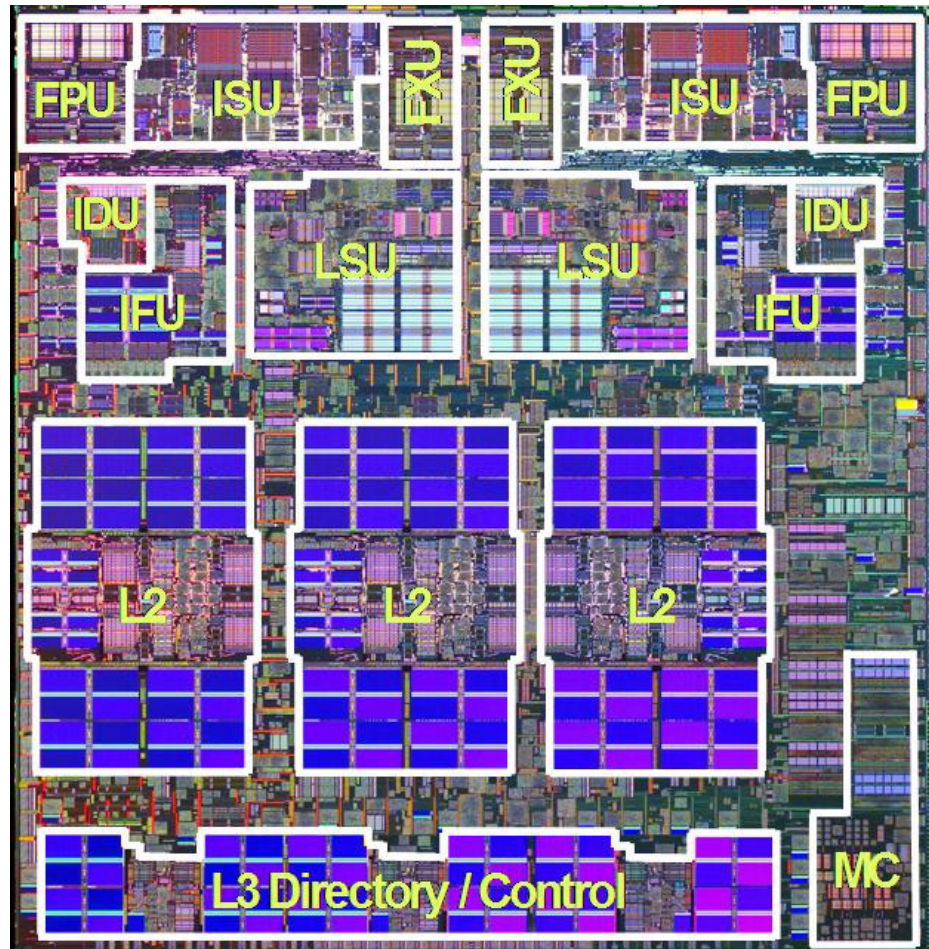


Figure 2-1 POWER5 processor chip

FXU - Fixed Point (Integer) Unit
FPU - Floating Point Unit
ISU - Instruction Sequencing Unit
IDU - Instruction Decoding Unit
LSU - Load Store Unit
IFU - Instruction Fetch Unit
L2 - Level 2 Cache
L3 - Level 3 Cache
MC - Memory Controller

2.1.1 Chip design

Two identical processor cores are found in a single POWER5 chip. Figure 2-2 shows the high-level layout of a POWER5 processor including the L3 cache and memory. Because of the dual core design and support for two hardware threads per core (SMT), a single POWER5 chip appears as a 4-way microprocessor system to the operating system.

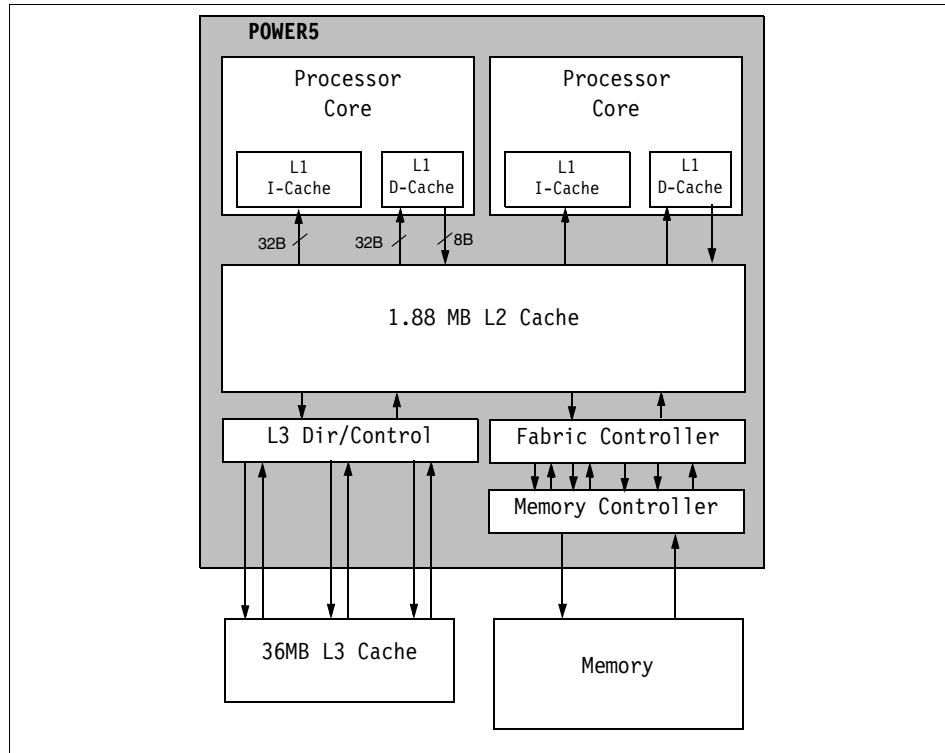


Figure 2-2 High level structure of POWER5

SMT is a hardware multithreading¹ technology which can greatly improve the utilization of the processor's hardware resources, resulting in better system performance. While superscalar processors can issue multiple instructions in a single cycle from a single code path (hardware thread), SMT processors can issue multiple instructions from multiple code paths (hardware threads) in a single cycle. POWER5 provides for two hardware threads per processor core. Hence, multiple instructions from both the hardware threads can be issued in a single processor cycle on the POWER5. For more information on Simultaneous

¹ The terminology 'multithreading' used here refers to the hardware execution of threads provided on a processor core as used in the computer architecture community. It is not same as the software use of the term.

Multithreading (SMT) see Chapter 5., “Simultaneous Multi-Threading (SMT)” on page 89.

2.1.2 POWER5 instruction pipelines

The POWER5 instruction pipeline can be subdivided into a master pipeline and several different execution pipelines. Figure 2-3 depicts the POWER5 instruction pipeline. Each box in the diagram represents a pipeline stage. The POWER5 pipeline structure is very similar to the POWER4 pipeline structure. Even the pipeline latencies including penalties for mispredicted branches and load-to-use latencies for L1 data cache hits remain the same. This design lets the compiler optimizations designed for POWER4 to work equally well on POWER5.

The master pipeline presents speculative in-order instructions to the mapping, sequencing and dispatch functions, and ensures an orderly completion of the real execution path. The master pipeline (in-order processing) throws away any potential speculative results associated with mispredicted branches. The execution pipelines allow out-of-order issuing of speculative and non-speculative instructions. The execution unit pipelines progress independently from the master pipeline and each other.

Instruction Fetching

In SMT mode, the POWER5 core uses two separate *Instruction Fetch Address Registers* (IFAR) to store the program counter for the two threads. Instructions are fetched every alternate cycle for each hardware thread (IF - instruction fetch stage see Figure 2-3 on page 14). In single-threaded mode, instructions are fetched from the active thread every cycle, and the program counter corresponding to that hardware thread is used. The POWER5 core can fetch an eight word (32 byte) aligned block of eight instructions per cycle. Keep in mind that all instructions in POWER and PowerPC are 32-bits (one word). The two threads share the instruction cache and the instruction address translation facility (L1 I-cache and I-ERAT). POWER5 also provides a four-entry 128B instruction prefetch queue above the I-cache for hardware initiated prefetching. The first two entries of the instruction prefetch queue are dedicated for thread 0 and the remaining two entries for thread 1 regardless of whether the core is running in SMT or single-threaded mode.

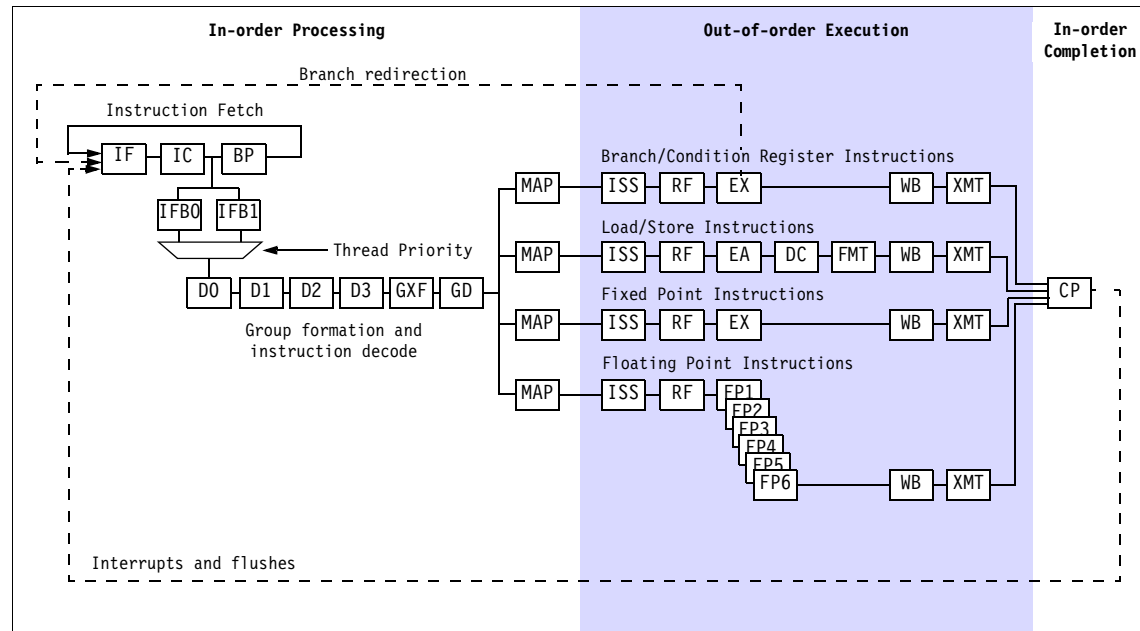


Figure 2-3 POWER5 instruction pipeline.

| | |
|-----------------------------------|---|
| IF - Instruction Fetch | IC - Instruction Cache Access |
| BP - Branch Prediction | IFB - Instruction Fetch Buffers |
| D0-D3 - Decode Stages | GXF - Group Transfer |
| GD - Group Dispatch | MAP - Register Mapping |
| ISS - Instruction Issue | RF - Register File Access |
| EX - Execution | EA - Effective Address Generation |
| DC - Data Cache Access | FMT - Data Formatting |
| WB - Write Back to Register | FP1 - Floating Point Alignment and Multiply |
| FP2 - Floating Point Multiply | FP3 - Floating Point Add |
| FP4 - Floating Point Add | FP5 - Floating Point Normalize Result |
| FP6 - Floating Point Round Result | XMT - Finish and Transmit |
| CP - Group Completion | |

Branch prediction

The eight fetched instructions are scanned for branch instructions each cycle (BP stage shown above). If branch instructions are found, the branch direction is predicted using three *Branch History Tables* (BHT). The tables are shared by the two threads and two of the tables use bimodal and path-correlated branch prediction mechanisms to predict branches. The third table is used as a selector designed to predict which of these prediction mechanisms is more likely to predict the right instruction path. The BP stage can predict all the branches at the same time in the fetched instruction group. If the instructions fetched contain multiple branches, the core logic has the capability to track up to eight

outstanding branches per thread in SMT mode, and 16 outstanding branches in single-threaded mode. The core logic also predicts the target of a taken branch in the current cycle's eight instruction group. The target address of most branches are calculated from the instruction's address plus an offset as described by the POWER and PowerPC architecture. For predicting targets of subroutine returns, the core logic uses a per thread eight-entry *Link Stack* (return stack). For predicting targets of the **bcctr** (branch conditional to address in the Count Register) instruction, a 32-entry target cache shared by both the threads is used. If a branch is taken, the core logic loads the program counter with the target address of the branch. If the branch is not predicted as taken, the address of the next sequential instruction (current instruction address + 4) is loaded into the program counter.

Instruction decoding and preprocessing

Instructions in the predicted path from BP stage are placed in the per thread *Instruction Fetch Buffers* (IFBs). This happens in the D0 stage (see Figure 2-3 on page 14). The core has two six-entry IFBs, one for each thread. Each IFB entry can hold four instructions. Up to eight instructions can be placed in one of the two IFBs every cycle. Up to 5 instructions can be taken out from either of the two IFBs every cycle. Based on the thread priorities, instructions from one of the IFBs are selected, split up into internal instructions in some cases (*instruction cracking*) and an *instruction group* is formed. This corresponds to the D1 to D3 stage. As instructions are later executed out of order, it is necessary to remember the program order of all instructions in flight. Instruction groups are formed in order to minimize the logic for tracking large number of instructions in flight. Groups of these instructions are tracked instead. Care is taken during group formation so that internal instructions that resulted from the cracking of an instruction do not end up in different groups. All instructions in a group come from the same thread and are decoded in parallel. Each group can have a maximum of five instructions.

Group dispatch

The process of moving the instructions belonging to a group formed in the D0 to D3 stages into the *Issue Queues* is known as *Group Dispatch* (GD). Before a group can be dispatched, the processor must ensure that resources required by the instructions in the group are available. Each instruction in the group needs an available entry in an appropriate issue queue, each load and store instruction needs an entry in the *Load Reorder Queue* and *Store Reorder Queue* respectively to be able to detect out-of-order execution hazards, each dispatched group needs an available entry in the *Global Completion Table* (GCT). The GCT is used to track the groups of five instructions formed in the D0-D3 stage. The core logic allocates GCT entries in program order for each thread. When all the necessary resources are available for the group, the group is dispatched (GD).

stage). Note that the instruction flow from the IF stage to the GD stage happens in program order.

Register renaming

To facilitate out-of-order and parallel execution of instructions in a group, the architected registers (the ones specified in the instruction) are renamed by utilizing a large physical register file provided in the core. Each register that is renamed needs to have a corresponding physical register. The *Rename Mapper* serves this purpose and renaming takes place in the MAP stage of the instruction pipeline. Table 2-1 summarizes the rename resources available to the POWER5 core. In SMT mode, both the threads can dynamically share the physical register files (rename resources). Instruction-level parallelism exploited for each thread is limited by the physical registers available for each thread. Certain workloads such as scientific applications exhibit high instruction level parallelism. To exploit instruction level parallelism of such applications, the POWER5 makes all the physical registers available to a single thread in single-threaded mode, allowing higher instruction level parallelism.

Table 2-1 *Rename resources in the POWER5 core*

| Resource type | Logical size per thread | Physical size |
|---------------|----------------------------------|---------------|
| GPRs | 32 (36 ^a) | 120 |
| FPRs | 32 | 120 |
| XER | 4 fields ^b | 32 |
| LR/CTR | 2 | 16 |
| CR | 8 (9 ^c) 4-bit fields | 40 |
| FPSCR | 1 | 20 |

- a. The POWER5 architecture uses 4 extra scratch registers known as eGPRs and one additional 4 bit CR field known as eCR, for instruction cracking and grouping. These are not the architected registers and are not available for the programming environment
- b. The XER is broken into four mappable fields and one non-mappable field per thread
- c. Eight CR fields plus one non-architected eCR field for instruction racking and grouping

Instruction execution

After the MAP stage, instructions enter the issue queues shared by the two threads. These issue queues feed the execution pipelines.

Each POWER5 processor core contains:

- Two fixed point (integer) execution pipelines²
 - Both capable of basic arithmetic, logical and shifting operations
 - Both capable of multiplies
 - one capable of divides and the other capable of Special Purpose Register (SPR) operations
- Two six stage load/store execution pipelines
- Two nine stage floating point execution pipeline (6-stage execution)
 - Both capable of the full set of floating-point instructions
 - All data formats supported in hardware including IEEE 754.
- One branch execution pipeline
- One condition register logical pipeline

The following instruction issue queues are built into the POWER5 core:

- Combined, two 18-entry issue queues to feed the fixed-point and load/store execution pipelines
- Two 12-entry issue queues to feed the floating point execution pipelines
- One 12-entry issue queue for branch execution pipeline
- One 10-entry issue queue for condition register logical execution pipeline

In summary, each POWER5 processor core has eight execution units, each of which can issue instructions out-of-order, with bias towards the oldest instructions first. Each execution unit can issue an instruction each cycle. Each execution unit can complete an instruction every cycle. Keep in mind that the total latency of an instruction depends upon the number and nature of each pipeline stage of execution. Instructions in the issue queue become eligible for issue when all the input operands for that instruction become available. The issue logic selects an eligible instruction from the issue queue and issues it (ISS stage). The issue logic does not differentiate between instructions from the two threads. Therefore, instructions from either of the threads can be issued at any given time, simultaneously to the execution units, thus making the core truly SMT. Upon issue of an instruction, the input physical registers for that instruction is read (RF stage), executed on the proper execution unit (EX stage), and results written

² Figure 2-3 on page 14 does not illustrate the number of execution units. See Figure 2-4 on page 18 instead.

back to the output physical register (WB stage). In each Load/Store Unit (LSU), an adder is used to compute the effective address to read from (load) or write to (store) in the EA stage and the data cache is subsequently accessed in DC stage. For load instructions, when data is returned from the data cache, a formatter selects the correct bytes from the cache line (FMT stage) and writes them to the register (WB stage).

When all the instructions in a group have executed without generating an exception and the group is the oldest of a given thread, the group is committed (CP stage). The processor core can commit two groups per cycle, one from each thread. The GCT entry allocated to the group during the GD stage is deallocated once the group is committed. Each POWER5 processor core has a 20-entry GCT shared by the two threads.

Figure 2-4 shown below provides some additional detail.

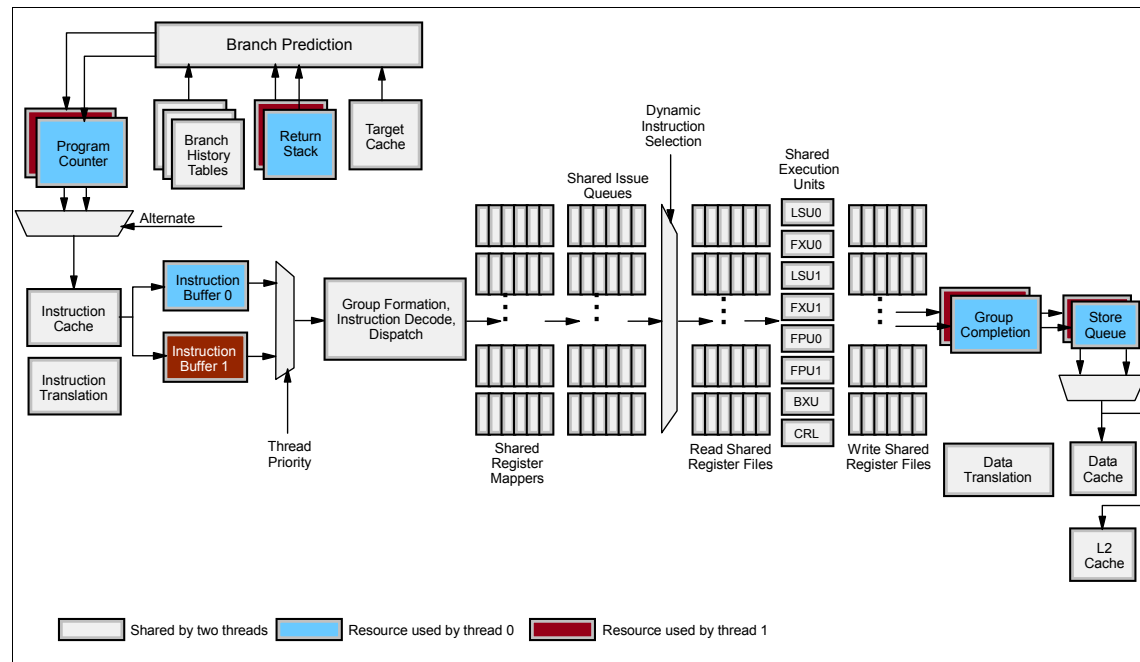


Figure 2-4 POWER5 instruction and data flow

2.1.3 L1 caches

Overview

The Level 1 (L1) caches are private to each processor core. Each processor core contains a 64 KB instruction cache and a 32 KB data cache. The cache line size for both caches is 128 bytes. In SMT mode, these caches are shared by the two hardware threads running in the core. Both the processor cores in a chip share a 1.88 MB unified L2. The processor chip houses a L3 cache controller which provides for a L3 cache directory on the chip. However, The L3 cache itself is on a separate Merged Logic DRAM (MLD) cache chip. The L3 is a 36 MB victim cache of the L2 cache. The L3 cache is shared by both the processor cores of the POWER5 chip. Needless to say, the L2 and L3 caches are shared by all the hardware threads of both processor cores on the chip. Table 2-2 on page 25 lists the cache characteristics of the POWER5 processor architecture.

L1 instruction cache

The L1 instruction cache is two-way set associative with *Least-Recently-Used* (LRU) replacement policy. The instruction cache is indexed using the effective address bits. The instruction cache is physically a single-ported array capable of either a read or a write in each cycle. On a cache miss, instructions are returned on the L2 cache interface in four 32-byte beats (typically, two beats on successive cycles, followed by several unrelated cycles, followed by the final two beats on successive cycles). The L2 always returns the “critical sector” (the sector containing the specific instruction address that references the cache line) in the first two beats, and the instruction fetching logic bypasses these demand-oriented instructions into the pipeline as quickly as possible. In addition, the cache line is written into one entry of the instruction prefetch queue so that the instruction cache itself is available for successive instruction fetches (which may be to cache lines other than the one that has just been received from the L2 cache). The instructions are written to the instruction cache during cycles when the instruction fetching is not using the instruction cache (for example, when the pipeline has backed up or when another instruction cache miss occurs). In this way, the writes to the instruction cache are hidden and do not interfere with normal instruction fetches.

L1 data cache

The L1 data cache is four-way set associative with *Least-Recently-Used* (LRU) replacement policy. On cache accesses, effective address bits are used to index into the L1 data cache. The data cache is a write-through design, so that it never holds modified data because data is written into both the L1 data cache and the L2 cache. The L1 Dcache provides two read ports and one write port to the core. On a cache miss, data is returned on the L2 cache interface in four 32-byte beats. Like instruction cache misses, the L2 always returns the “critical sector” (the sector containing the specific data address that reference the cache line) in

the first beat and the load miss queue (LMQ) forwards these demand-oriented loads into the pipeline as quickly as possible, critical data forwarding (CDF). The other half of the 64-byte L2 read is returned in the second beat. As each 32-byte beat is received it is written to the cache. When all four 32-byte beats are received the data cache directory is updated. Since the L1 data cache is a write-through cache an eight byte (64-bit) data bus is used to update the L2 cache when core executes a store instruction.

2.1.4 L2 cache

The L2 is a unified cache accessed by both cores on the POWER5 chip. In addition, it maintains full hardware coherence within the system and can supply intervention data to the cores on other POWER5 processors. Logically, the L2 is an in-line cache. Unlike the L1 caches, which are write-through, it is a copy-back (store-in) cache. It is fully inclusive of the two L1 instruction caches and the two L1 data caches located in the two processor cores on one POWER5 chip.

The L2 is a total of 1.88 MB and is physically partitioned into three symmetrical *slices* with each slice holding 640 KB of instructions or data. As shown in Figure 2-6 on page 21, each slice is comprised of 512 associative sets called *congruence classes*, each congruence class contains ten 128-byte cache lines. Each of the slices have separate L2 cache controllers. Either processor core of the chip can independently access each L2 controller. The correct slice is determined by a hashing algorithm involving bits 36:55 of the physical address.

Once the slice has been determined, the indexing of the cache by the L2 controller is performed using the address bits as shown in Figure 2-5 on page 21. Using the address of either the requested instruction or data, bits 57:63 are used to represent the byte offset within the cache line. Address bits 48:56 are used to select the congruence class. A physical tag comparison (i.e. real address bits 14-47) is used to determine if the desired cache line is resident within one of the ten ways for that congruence class.

The L2 provides 64 bytes of access per two processor clock cycles per slice. There is also a 32 byte cache reload bus that operates at core frequency. Each slice has a castout/intervention/push bus (16 bytes wide) to the fabric controller and operates at half the core frequency. *Error Correction Control* (ECC) provides single bit error recovery. To aid in performance, eight, 64-byte wide store queues are provided per slice supporting SMT. To minimize bus contention, store gathering is also supported.

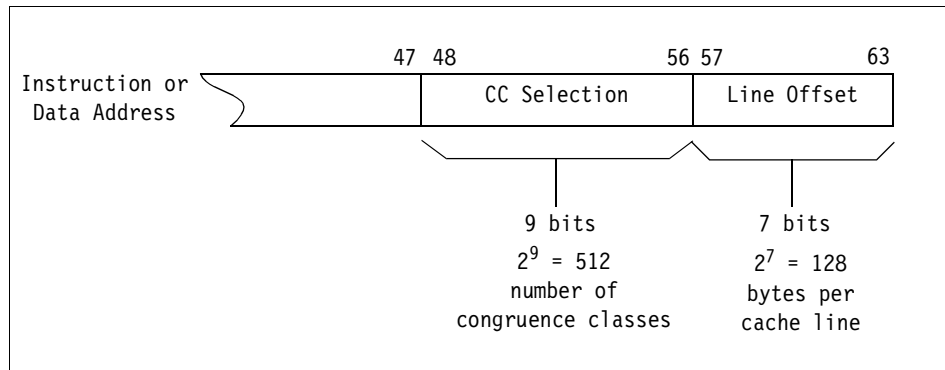


Figure 2-5 Cache indexing bits

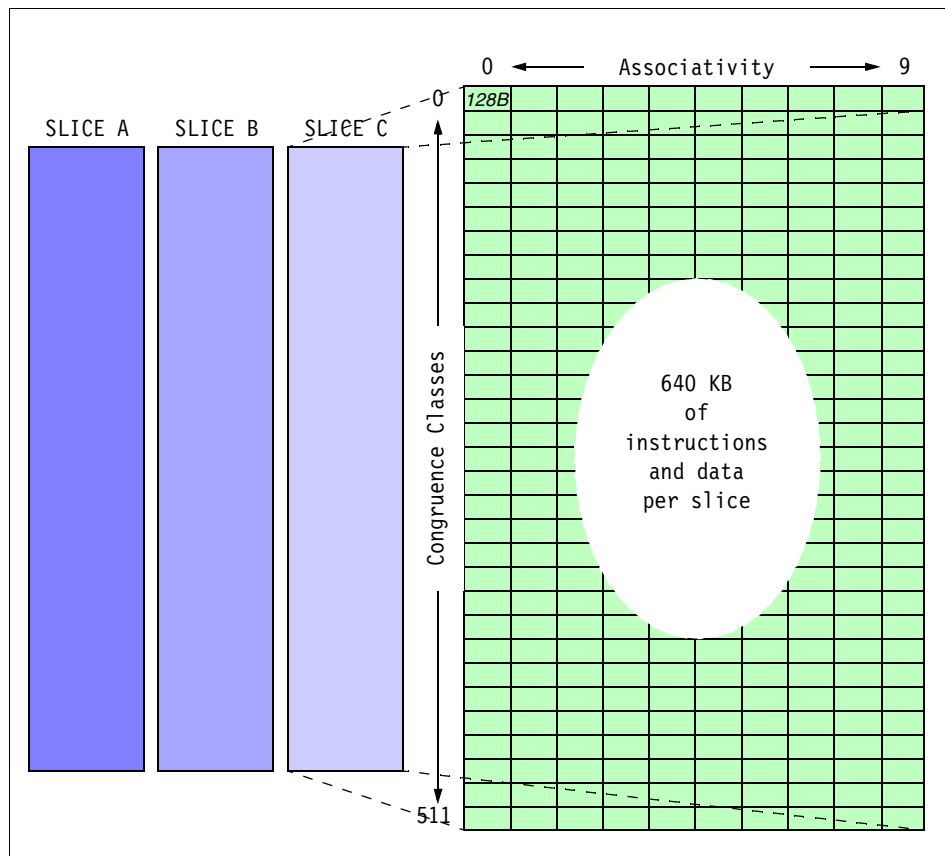


Figure 2-6 L2 cache organization

2.1.5 L3 cache

The L3 cache is a unified 36 MB cache accessed by both cores on the POWER5 processor chip. It maintains full memory coherency with the system and can supply intervention data to cores on other POWER5 processor chips. The L3 is a *victim cache* and is not inclusive of the L2. This means that the same cache line will never reside in both caches simultaneously and a valid, modified cache line cast-out from the L2 due to being least-recently-used in the associated congruence class is placed into the L3 cache.

This cache is implemented off-chip as a separate *Merged Logic DRAM* (MLD) cache chip. However, the L3 cache directory and control is on the POWER5 processor chip itself. Having the L3 directory on the processor chip itself helps the processor check the directory after an L2 miss without experiencing off-chip delays. Figure 2-7 shows a high level diagram of this design.

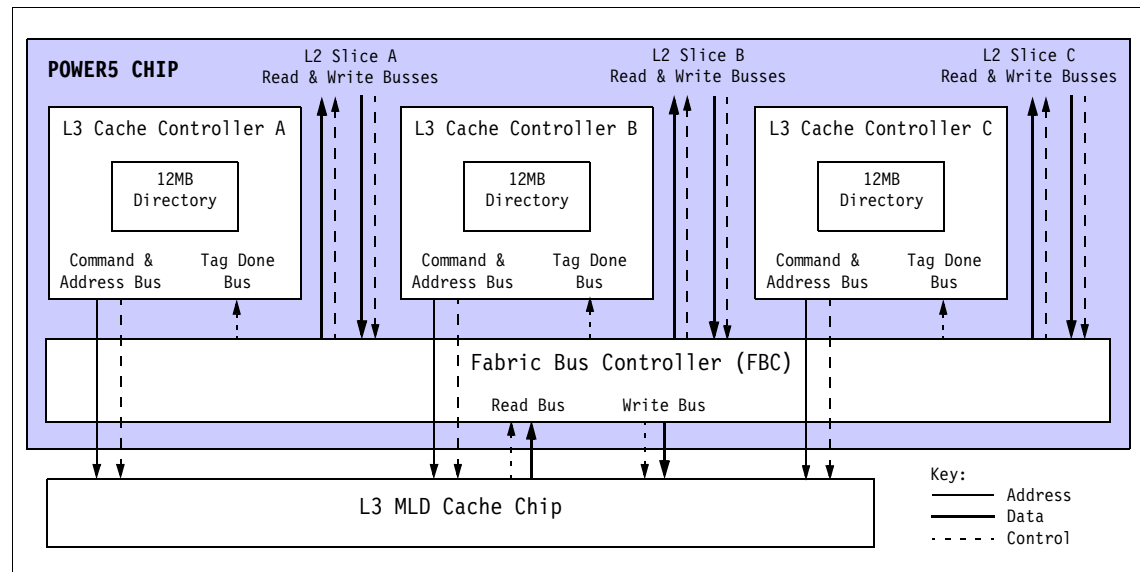


Figure 2-7 L3 cache high-level design

The cache is split into three identical 12 MB *slices* on the cache chip. The same hashing algorithm used to select the L2 slices is used to select the L3 slices for a given physical address. A slice is comprised of 12-way set-associative congruence classes (associative sets). There are 4096 congruence classes which are 2-way sectored (which means the directory manages two 128B cache lines per entry). Each of the 12 MB slices can be accessed concurrently. Figure 2-8 on page 23 depicts the L3 cache organization graphically.

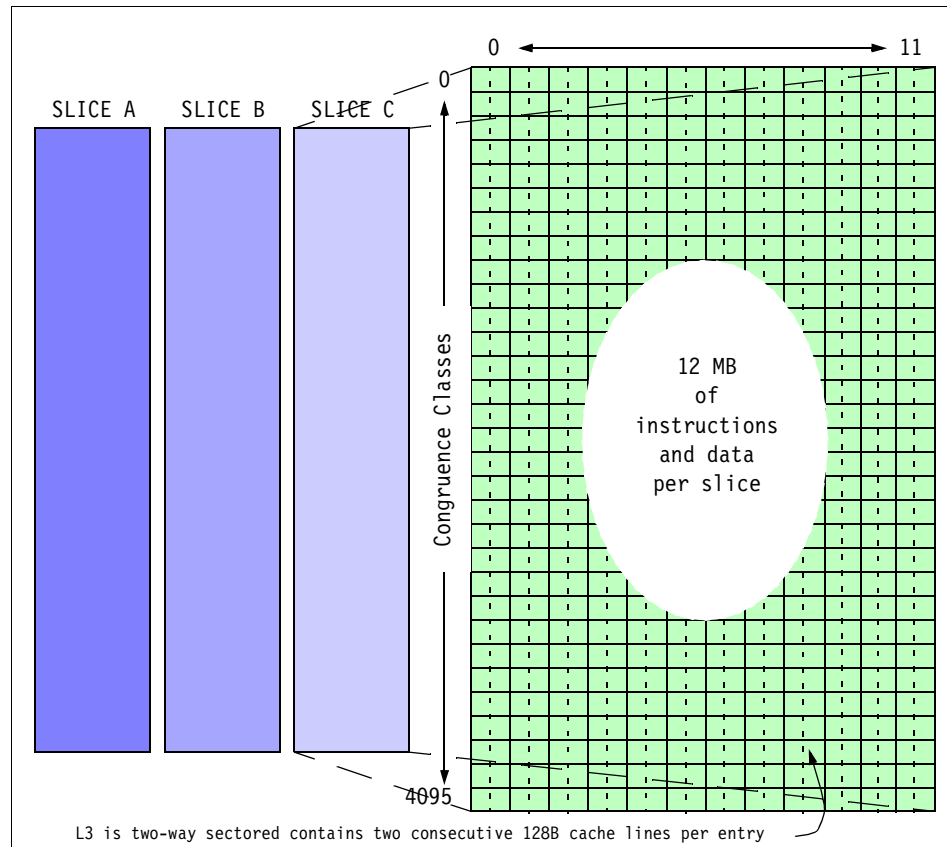


Figure 2-8 L3 cache organization

The L3 cache is on the processor side and not on the memory side of the fabric as in POWER4. This is depicted in Figure 2-9 on page 24. This design lets the POWER5 satisfy L2 cache misses more efficiently with hits on the off-chip cache, thus avoiding traffic on the interchip fabric. References to data not on the L2 cause the system to check the L3 cache before sending requests onto the interchip fabric. The L3 operates as a back door with separate buses for reads and writes that operate at half the processor speed. Because of higher transistor density of the POWER5 fabrication technology, the memory controller has now been moved on the chip, eliminating the need for a separate memory controller chip as in POWER4 systems.

These architectural changes to the POWER5 have the significant benefits of reducing latency to the L3 and main memory as well as the number of chips necessary to build a system. The result is higher level of SMP scaling. Initial POWER5 systems support 64 physical processors.

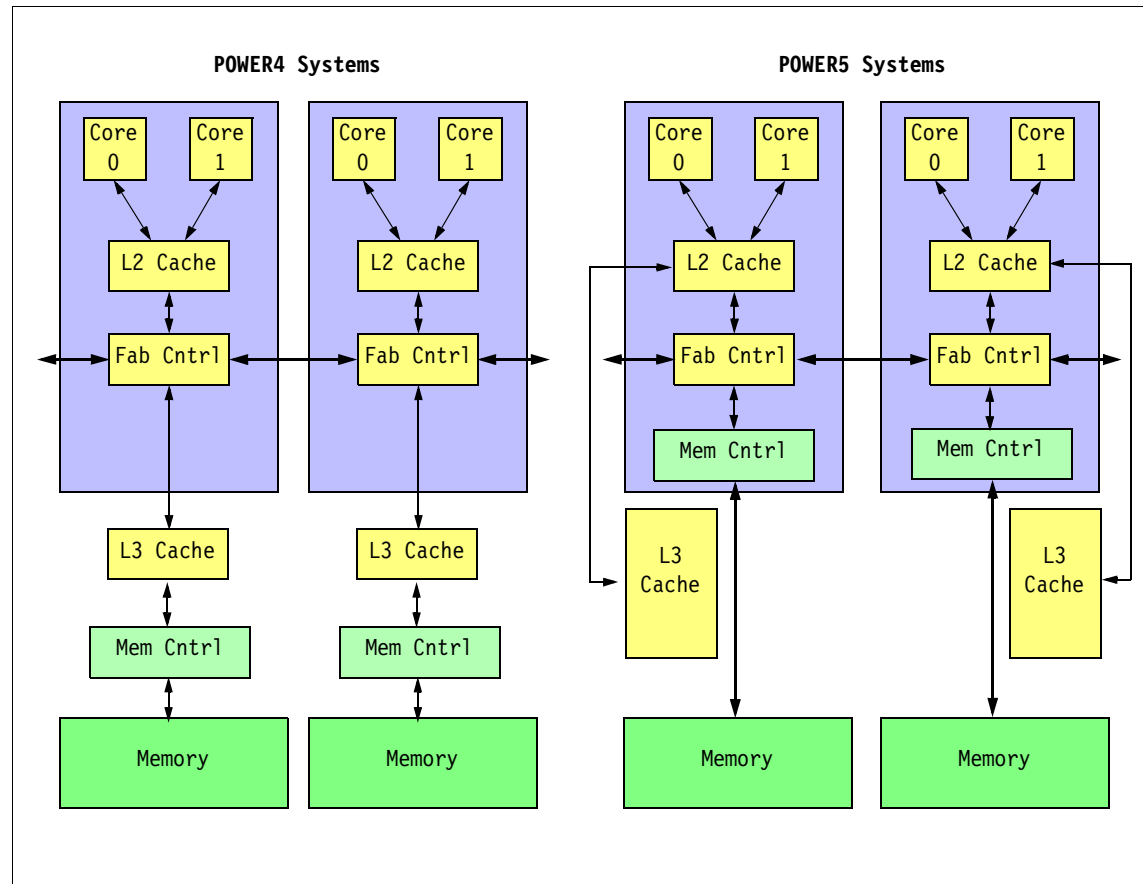


Figure 2-9 Comparison between POWER4 and POWER5

2.1.6 Summary of caches on POWER5

Table 2-2 Cache characteristics of the POWER5 processor

| Cache Characteristics | L1 Instruction Cache | L1 Data Cache | L2 Cache | L3 Cache |
|-----------------------|----------------------|--|---|---|
| Contents | Instructions only | Data only | Instructions and Data | Instructions and Data |
| Size | 64 KB | 32 KB | 1.88 MB | 36 MB |
| Associativity | 2-way | 4-way | 10-way | 12-way |
| Replacement Policy | LRU | LRU | 14-bit LRU | 15-bit LRU |
| Line size | 128 B | 128 B | 128 B | 256 B |
| Indexed by | Effective Address | Effective Address | Physical Address | Physical Address |
| Tags | Physical Address | Physical Address | Physical Address | Physical Address |
| Inclusivity | N/A | N/A | Inclusive of L1 instruction and data caches | <i>Not</i> inclusive of L2 cache (victim cache of L2) |
| Hardware Coherency | Yes | Yes | Yes (separate snoop ports) | Yes (separate snoop ports) |
| Store policy | N/A | Write-through No allocate on store miss | Copy-back Allocate on store miss | Copy-back |

2.1.7 Address translation resources

The POWER5 chip supports translation from a 64 bit *effective address* (EA) to a 65 bit *virtual address* (VA) and then to a 50 bit *real address* (RA). The processor architecture specifies a *Translation Lookaside Buffer* (TLB) and a *Segment Lookaside Buffer* (SLB) to translate the effective address used by software to a real address (physical address) used by the hardware. Each processor core contains a unified, 1024 entry, four-way set associative TLB. The TLB is a cache of recently accessed page table entries that describe the pages of memory.

There are two Effective-to-Real Address Translation (ERATs) caches. They are called the I-ERAT and the D-ERAT for instruction address translation and data address translation, respectively.

The I-ERAT is a 128 entry, 2-way set associative translation cache which uses a FIFO-based replacement algorithm. In this algorithm, one bit is kept per congruence class and is used to indicate which of the two entries was loaded first. As the name implies, the first entry loaded is the first entry targeted for replacement when a new entry needs to be loaded into that congruence class.

Each entry in the I-ERAT provides translation for a 4KB block of storage. In the event that a particular section of storage is actually mapped by a large page TLB entry (16 MB), each referenced 4KB block of that large page will occupy an entry in the I-ERAT (i.e., large page translation is not directly supported in the I-ERAT).

The D-ERAT is a 128 entry, fully associative translation cache which uses a binary LRU replacement algorithm. Like the I-ERAT, the D-ERAT provides address translations for 4KB and 16MB pages of storage.

2.2 Enhanced SMT features

All the resources on the POWER5 have been tuned for optimum performance within the area and power budget considerations. The optimal number of rename resources such as number of physical GPRs to be put on the core, have been arrived at by experimenting with workloads and varying the number of GPRs for maximum instructions executed per cycle during the course of chip design. To enhance SMT capabilities and better utilize processor resources, POWER5 provides *dynamic resource balancing and adjustable thread priorities*.

<Add info on PURR register here>

2.2.1 Dynamic resource balancing (DRB)

The purpose of this feature is to ensure smooth flow of both threads through the processor. If either of the hardware threads start dominating the processor resources and depriving the other thread, the dynamic resource balancing logic throttles down the dominating thread so that the other thread can flow smoothly without stalling. For example, if one of the hardware threads experiences multiple L2 data cache misses, the dependant instructions can block in the issue queue slots preventing the other thread from dispatching instructions (refer to the processor pipeline discussion earlier in Section 2.1.2, “POWER5 instruction pipelines” on page 13. To prevent such stalls, the DRB logic monitors the miss queues, and if a particular thread reaches a threshold for L2 cache misses, it throttles that thread down so that the other thread can progress smoothly. Similarly, one thread could start using too many *Global Completion Table* (GCT) entries, preventing the other thread from dispatching instructions. DRB logic then detects this condition and throttles down the thread dominating the GCT.

POWER5 DRB could throttle down a thread in three different ways. Choice of the throttling mechanism is made depending on the situation. The throttling mechanisms are:

1. Reducing the thread's priority
 - Used in situations where a thread has used more than a predetermined number of GCT entries.
2. Holding the thread from decoding instructions until resource congestion is cleared
 - When number of L2 misses incurred by a thread reach a threshold
3. Flushing all the dominating thread's instructions waiting for dispatch and holding the thread's decoding unit until congestion clears
 - Used if a long latency instruction such as memory ordering instructions (e.g., **sync**) causes dominating of the issue queues.

Studies have shown that higher performance is realized when resources are balanced across the threads using DRB.

2.2.2 Adjustable thread priorities

The dynamic resource balancing logic is built into the hardware to ensure balanced resource utilization by the threads. There are scenarios when software could know that a process running on a hardware thread might not be doing any useful work, such as spinning for a lock, executing the operating system's idle loop for example. Sometimes, software might also want to quickly execute a process, such as a process holding a critical spinlock. For better utilization of processor resources under such scenarios, the POWER5 features adjustable thread priorities, where in software it can be specified if a hardware thread running the process can have more or less execution resources.

The POWER5 supports eight levels of thread priorities (0-7). The thread's priority is stored in a *Thread Status Register* (TSR). A three bit field is used to indicate the thread priority). The POWER5 supports three processor states, *Hypervisor Mode*, *Supervisor Mode* (AIX or Linux kernel code), and *User Mode* (application program)

Table 2-3 POWER5 thread priority levels

| TSR Thread priority value | Priority Level | Privilege level for software to set this priority ^a | Equivalent no-op instruction |
|---------------------------|-----------------|--|------------------------------|
| 0 | Thread shut off | Hypervisor Mode ^b | - |
| 1 | Very low | Supervisor Mode | or 31,31,31 |
| 2 | low | User Mode | or 1,1,1 |
| 3 | Medium low | User Mode | or 6,6,6 |
| 4 | Normal | User Mode | or 2,2,2 |
| 5 | Medium high | Supervisor Mode | or 5,5,5 |
| 6 | high | Supervisor Mode | or 3,3,3, |
| 7 | Extra high | Hypervisor Mode | or 7,7,7 |

a. Certain fields in a thread control register (TCR) affect the privilege level. This column assumes recommended setting and setups, which is usually the case with well behaved software.

b. Hypervisor will be discussed later in subsequent sections. Hypervisor can be considered highest privilege level followed by supervisor (usually the O/S) and user applications.

Ratio of decode slots allocated to the threads depend on the two thread's priority as in $1/2^{(|x-y|+1)}$ of the decode slots is given to the lower priority thread, where x and y are thread priorities, and $x > 1$ and $y > 1$. So if thread 0 has a priority of 4 and thread 1 has a priority of 2, then thread 1 gets 1/8 of the total decoding slots and thread 0 gets 7/8 of the decoding slots. Table 2-4 on page 29 summarizes decode unit division among threads under different thread priority scenarios.

Table 2-4 Effect of thread priorities on execution resource sharing

| Thread 0 priority | Thread 1 priority | Decode slots status |
|-------------------|-------------------|---|
| 0 | 0 | Stopped |
| 0 | 1 | 1 /32 of decode slots given to thread 1 for power savings. Thread 0 is stopped |
| 0 | >1 | All of the decode slots go to thread 1 |
| 1 | 1 | Both the threads are given 1/32 of the total decode slots for power savings |
| 1 | >1 | Thread 1 gets all the execution resources and thread 0 gets the leftovers. |
| >1 | >1 | 1 of $2^{(x-y +1)}$ decode units for the lower priority thread and the rest to the other thread. |

Figure 2-10 on page 30 depicts the effects of thread priorities on instructions executed per cycle. The x-axis entries with commas represent actual thread priority pairs, e.g., 0,7 implies thread 0 has been stopped and thread 1 has a priority of 7). The numbers without commas represent the value of $(x-y)$ where x is the priority of thread0 and y is the priority of thread1. So a value of 5 on the x-axis would mean (7,2) or (6,1) for x and y .

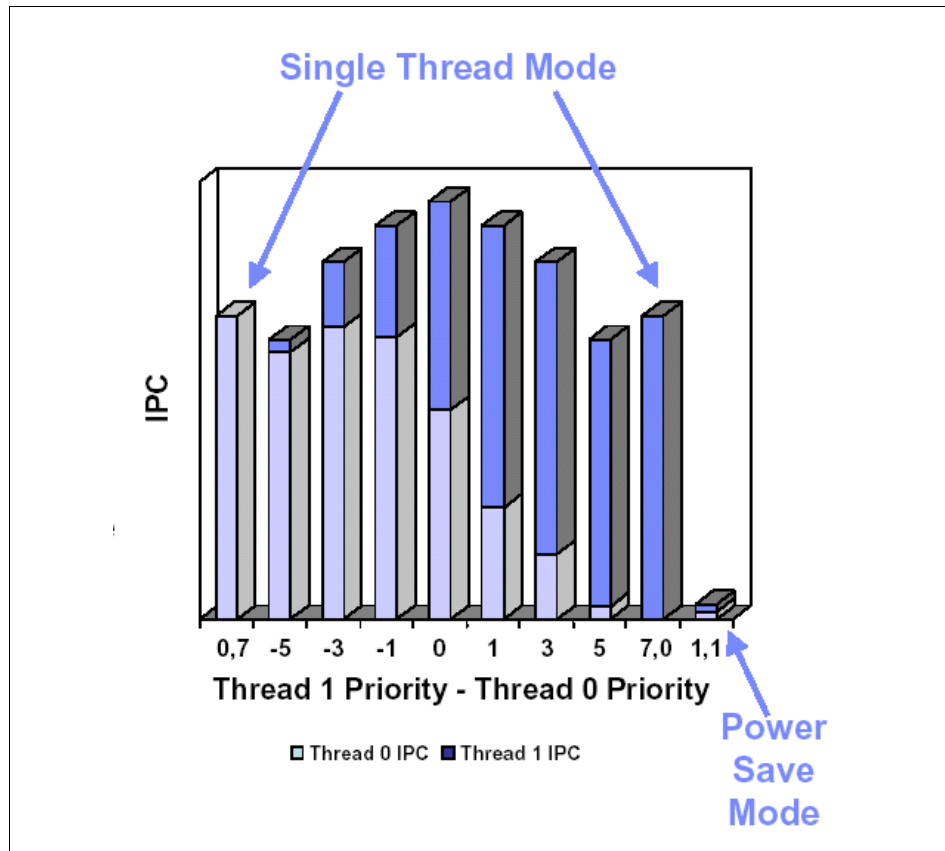


Figure 2-10 Thread priority pairs vs. instructions executed per second

Idea behind SMT is to increase overall throughput of the system by executing two threads which may not utilize the processor execution resources to the desired level, if run individually on the processor (in single-threaded mode). SMT does not speed up individual threads of execution, but the work done collectively, or overall throughput goes up. If applications care about real time responses rather than overall system performance, they are better off running in single-threaded mode. Some workloads are limited by the processor execution resources (such as technical workloads that exhibit high instruction level parallelism and consume large number of rename resources like FPRs), SMT will not help much. The POWER5 provides facilities for the operating systems to dynamically switch from SMT to single-threaded for such applications and workloads.

2.3 Dynamic power management

Chip power is a very important and limiting factor in modern processor designs. With SMT, the number of instructions executed per cycle goes up thus increasing the chip's total switching power. To reduce switching power, POWER5 chips use a fine grained dynamic clock-gating mechanism to gate off clocks to a local clock buffer, if the dynamic power management logic knows that the set of latches driven by that clock buffer will not be used in the next cycle. For example, if the FPRs will not be read on the next cycle, the dynamic power management logic detects it and turns off the clocks to the FPR read ports. A minimum amount of logic implements the clock gating function. Special care has been taken to ensure clock gating logic does not cause no performance loss and does not create a critical timing path for the chip.

In addition to switching power, leakage power has become a performance limiter. POWER5 uses transistors with low threshold voltage only in critical paths such as FPR read path.

POWER5 also provides for the software to hint low power modes, when the thread priorities run at priority one as shown in Table 2-4 on page 29, the POWER5 dispatches instruction utmost once in every 32 cycles saving on power. Figure 2-11 on page 32 shows the photos taken with thermal sensitive cameras with and without dynamic power management on prototype POWER5 chips. From the picture, it is evident that dynamic power management reduces power consumption below standard single-threaded level without power management.

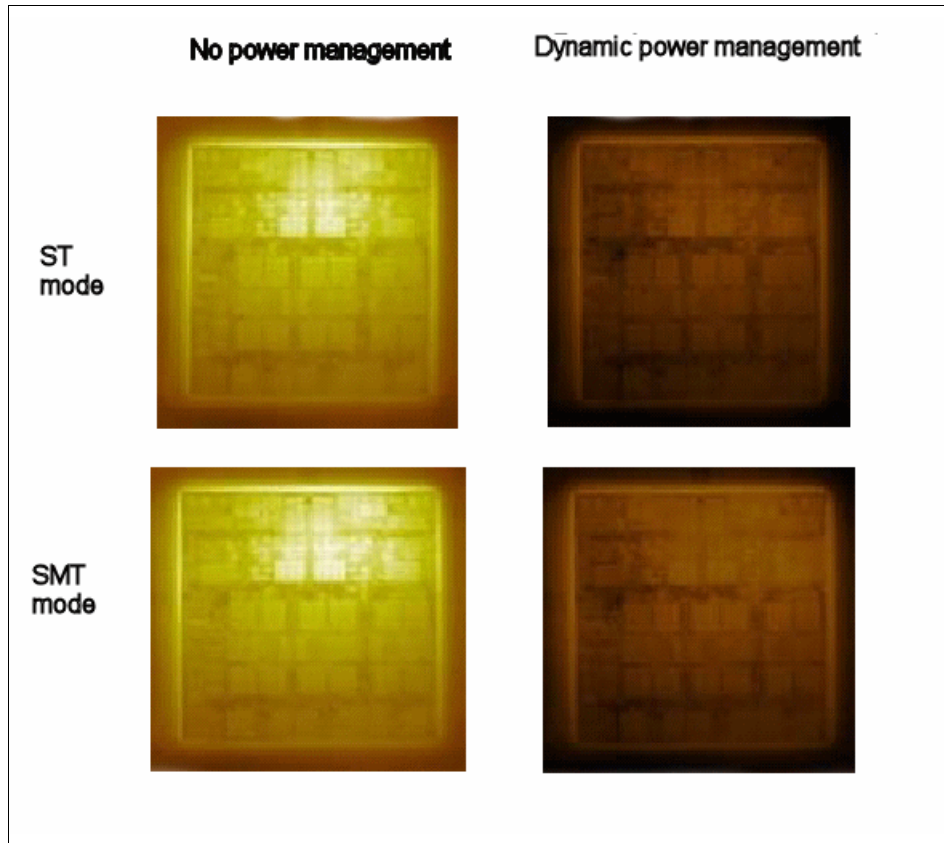


Figure 2-11 POWER5 photos using thermal sensitive camera

2.4 Large POWER5 SMPs

Somewhat like the POWER4, the POWER5 uses *Dual Chip Modules* (DCMs)³ and *Multi-Chip Modules* (MCMs) as the basic building blocks for low/midrange and high end servers respectively.

Figure 2-12 on page 33 depicts a POWER5 DCM. This basic building block can be put together to form a 16 way system as shown in Figure 2-13 on page 34. The pink boxes represent the dual core POWER5 chip. The light blue boxes represent the DCM module which houses the POWER5 chip and L3 (purple box). The gray box represents a drawer. The DCMs can be interconnected to

³ DCM has one POWER5 chip and one L3 MLD cache chip, hence the name 'dual' chip module. The DCM has only one POWER5 chip with two cores.

form a two way, four way, eight way, 12-way and 16-way smps with 1,2,4,6 and 8 DCMs respectively. The dark blue boxes represent memory.

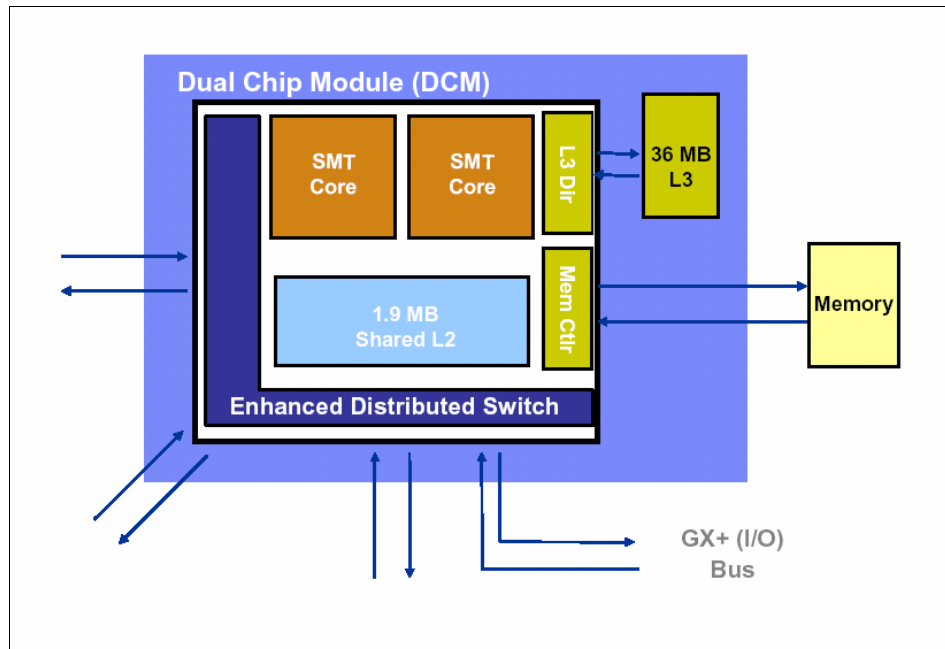


Figure 2-12 POWER5 DCM

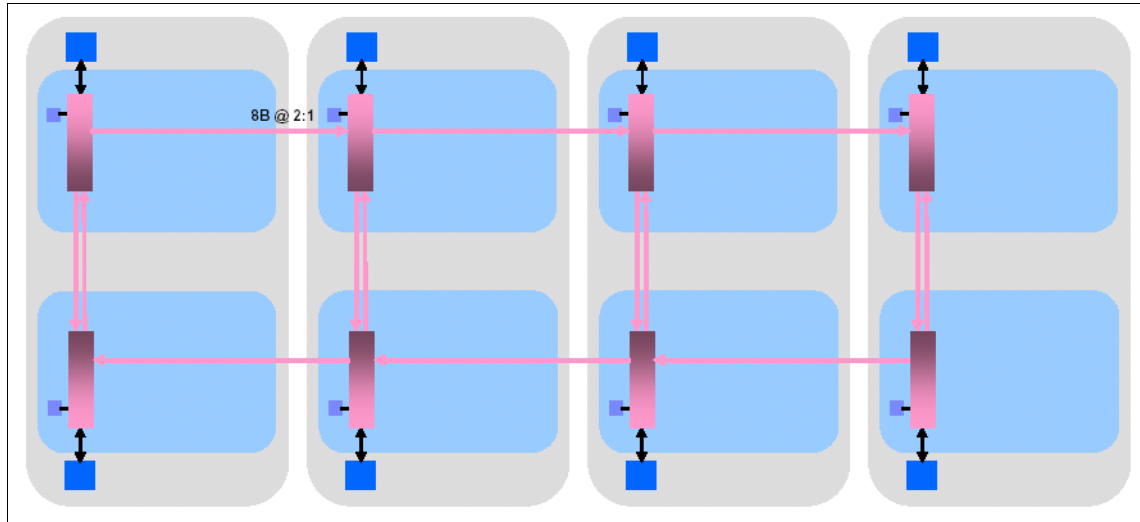


Figure 2-13 DCM interconnection for a 16 way SMP

Figure 2-14 shows a picture of the POWER5 DCM. The chip to the left of the reader is the L3 MLD cache chip, and the larger blue chip onto the right is the POWER5 chip.

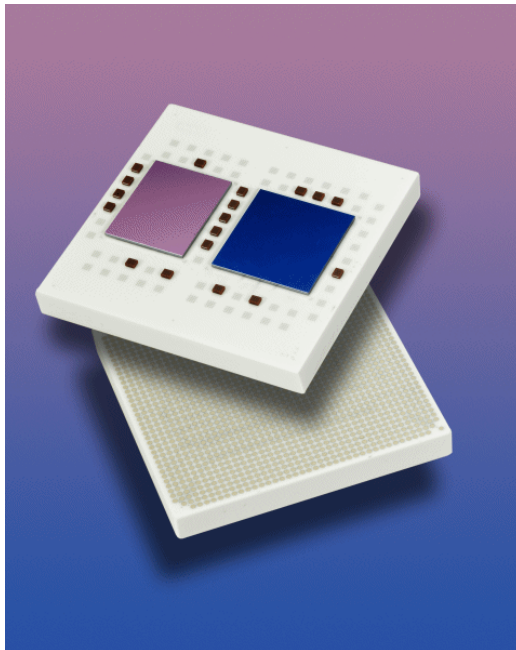


Figure 2-14 Picture of a POWER5 DCM

Like the POWER4, POWER5 exploits the enhanced distributed switch for interconnects. All chip interconnects operate at half the processor frequency and scale with processor frequency.

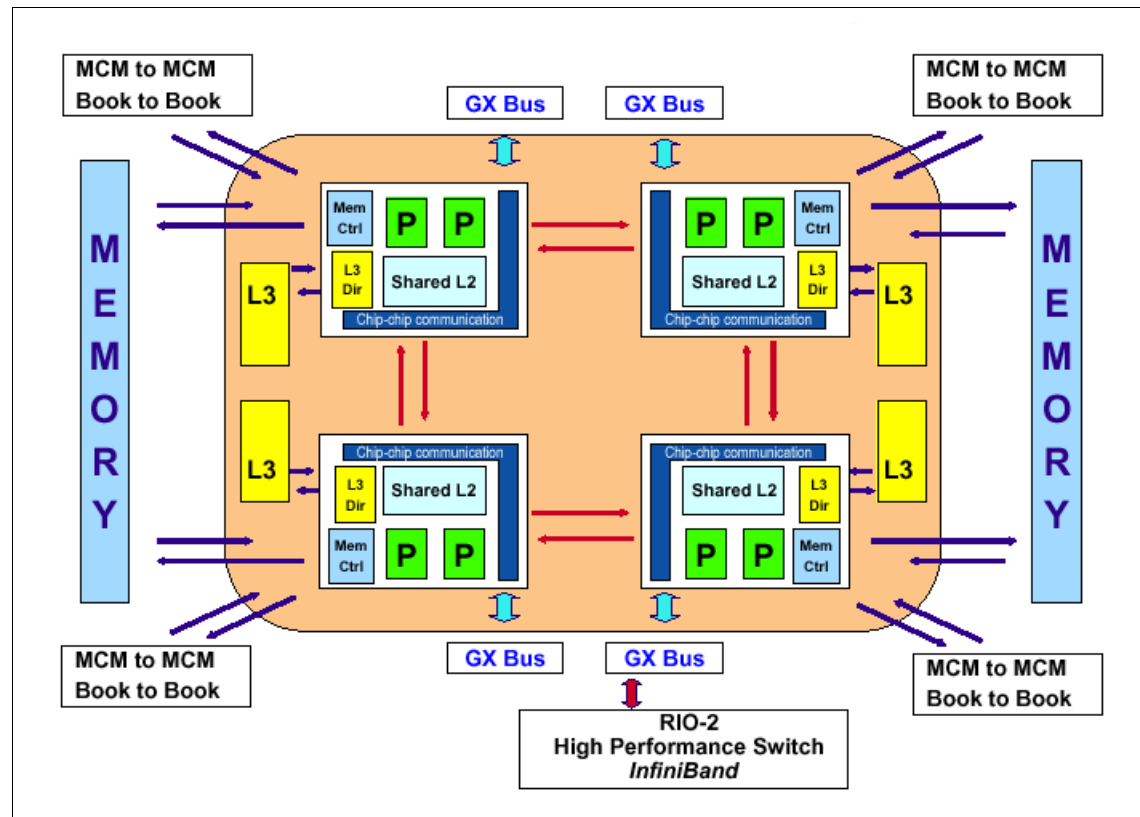


Figure 2-15 Logical view of the POWER5 MCM

Figure 2-15 on page 35 depicts the logical view of a POWER5 MCM. MCMs are used as basic building blocks on high end SMPs. MCMs have four POWER5 chips and four L3 cache chips each. Each MCM is a eight-way building block. Figure 2-16 on page 36 shows a picture of a POWER5 MCM.

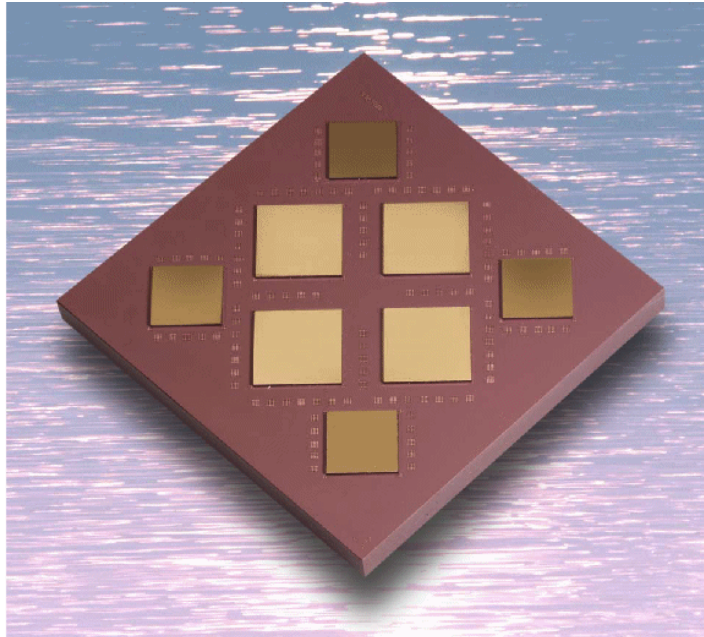


Figure 2-16 POWER5 MCM

Two POWER5 MCMs can be tightly coupled to form a book as shown in Figure 2-17 on page 37. These books are interconnected together again to form larger SMPs up to 64 ways as shown in Figure 2-18 on page 37. In the figure the light blue box represents a MCM. The MCMs and books can be interconnected to form a 8-way, 16-way, 32-way, 48-way and 64-way SMPs with 1,2,4,6 and 8 MCMs respectively.

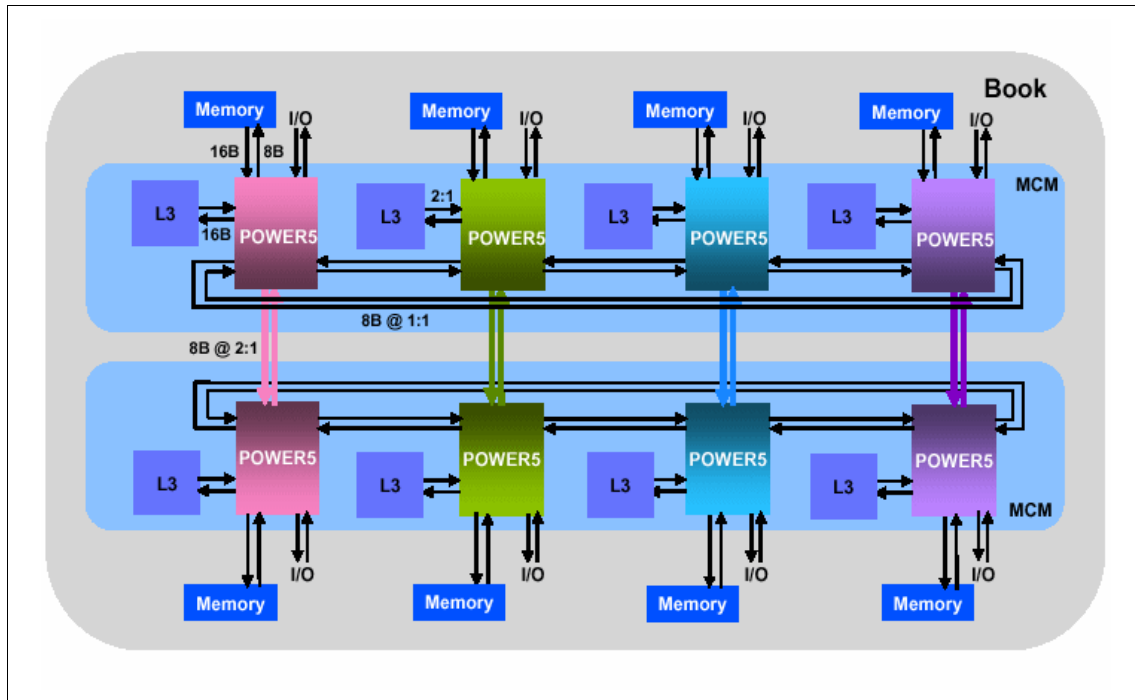


Figure 2-17 POWER5 book

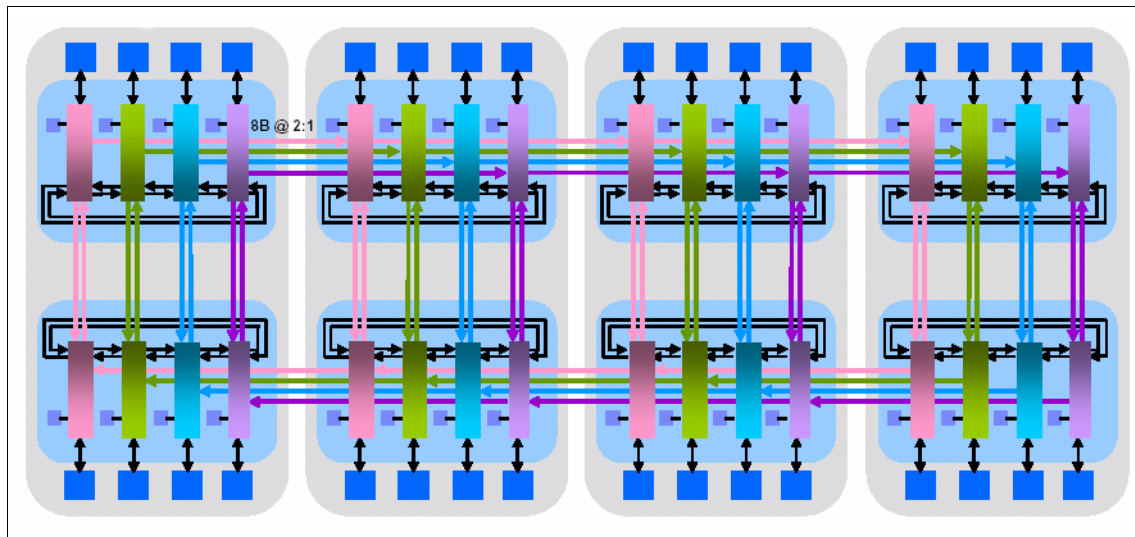


Figure 2-18 MCMs interconnected to make a 64 way SMP



POWER Hypervisor

The technology behind the shared processor on the POWER5 is provided by a piece of firmware known as the POWER Hypervisor™. The enhanced layered code structure of Hypervisor resides in flash memory on the Service Processor. This firmware performs the initialization and configuration of the POWER5 processor, as well as the virtualization support required to run up to 254 partitions concurrently on the IBM @serverp5 servers.

The Hypervisor supports many advanced functions when compared to the previous version of the Hypervisor, including sharing of processors, virtual I/O, high-speed communications between partitions using Virtual LAN, concurrent maintenance and allows for multiple operating systems to run on the single system. AIX 5L™, Linux and i5/OS™ are supported.

With support for dynamic resource movement across multiple environments, customers can move processors, memory and I/O between partitions on the system as they move workloads between the three environments.

The Hypervisor is the underlying control mechanism that resides below the operating systems, but above the hardware layer, see Figure 3-1 on page 40. The Hypervisor owns all system resources and creates partitions by allocating these resources and sharing them.

The layers above the Hypervisor are different for each supported operating system.

For the AIX 5L and Linux operating systems, the layer above the Hypervisor are similar but the contents are characterized by each operating system. The layers of code supporting AIX 5L and Linux consist of System Firmware and Run-Time Abstraction Services (RTAS).

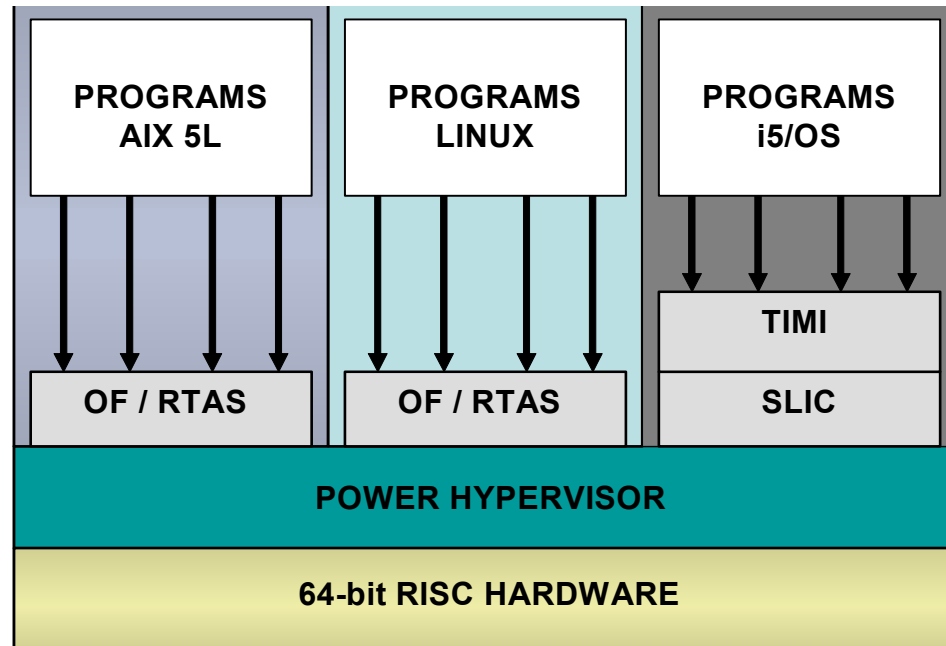


Figure 3-1 POWER Hypervisor

System firmware is composed of *low level firmware* that is code that performs server unique input/output (I/O) configurations and the Open Firmware which contains the boot time drivers, boot manager, and the device drivers required to initialize the PCI adapters and attached devices. Run-Time Abstraction Services (RTAS) consist of code that supplies platform dependent accesses and can be called from the operating system. These calls are passed to the Hypervisor that handles all I/O interrupts.

The role of RTAS versus Open Firmware is very important to understand. Open Firmware and RTAS are both platform-specific firmware and both are tailored by the platform developer to manipulate the specific platform hardware. However, RTAS is intended to present to access platform hardware features on behalf of the operating system, whereas Open Firmware need not be present when the operating system, is running. This frees Open Firmware's memory to be used by applications. RTAS is small enough to painlessly coexist with the operating system and applications.

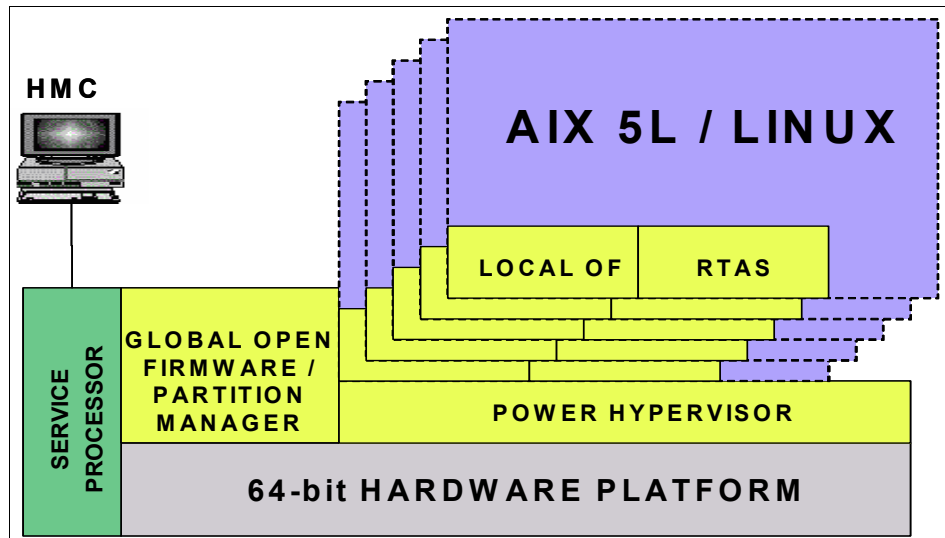


Figure 3-2 POWER Hypervisor on AIX 5L and Linux

For i5/OS, Technology Independent Machine Interface (TIMI) and the layers above the Hypervisor are still in place. System Licensed Internal Code (SLIC), however, is changed and enabled for interfacing with the Hypervisor. The Hypervisor code is based on the iSeries Partition Licensed Internal Code (PLIC) code that is enhanced for use with the IBM @server i5 hardware. The PLIC is now part of the Hypervisor.

Attention: All POWER5-based servers require the use of the Hypervisor. A system administrator can configure the system as a single, large LPAR “Full System Partition mode” that includes all the resources on the system, but cannot run in SMP mode without the Hypervisor as they could with POWER4 systems. A POWER5-based server is always in LPAR mode

3.0.1 POWER Hypervisor Support

The POWER5 processor supports a special type of instructions. These instructions are exclusively used by the Hypervisor. If an operating system instance in a partition requires access to hardware, it first invokes the Hypervisor by using Hypervisor calls. The Hypervisor allows privileged access to the operating system for dedicated hardware facilities and includes protection for those facilities in the processor and memory locations.

Architecturally, the Hypervisor, a component of global firmware, owns the partitioning model and the resource abstractions that are required to support that

model. Each partition is presented with the resource abstraction for its partition and other required information through the Open Firmware Device Tree, which is created by firmware and copied into the partition before the operating system is started. In this way, operating systems receive resource abstractions. They also participate in the partitioning model by making Hypervisor calls at key points in their execution as defined by the model.

The introduction of shared processors did not fundamentally change this model. New virtual processor objects and Hypervisor calls have been added to support shared processor partitions. Actually, the existing physical processor objects have just been refined, so as not to include physical characteristics of the processor, since there is not fixed relationship between a virtual processor and the physical processor that actualizes it. These new Hypervisor calls are intended to support the scheduling heuristic of minimizing idle time.

The Hypervisor is entered by the way of three interrupts:

- **System Reset Interrupt**

The Hypervisor code saves all processor state by saving the contents of the processor's registers (multiplexing the use of this resource with the operating system). The processor's stack and data are found by processing the Processor Identification Register (PIR). The PIR is a read only register. During power-on reset, PIR is set to a unique value for each processor in a multi-processor system.

- **Machine Check Interrupt**

Hypervisor code saves all processor state by saving the contents of the processor's registers (multiplexing the use of this resource with the operating system). The processor's stack and data are found by processing the Processor Identification Register (PIR).

The Hypervisor investigates the cause of the machine check. The cause may either be a recoverable event on the current processor or one of the other processors in the logical partition. Also the Hypervisor must determine if the machine check has corrupted its own internal state (by looking at the footprints, if any, that were left in the per processor data area of the errant processor).

- **System (Hypervisor) Call Interrupt**

The Hypervisor call (**hca11**) interrupt is a special variety of the **sc** (system call) instruction. The parameters to the **hca11()** are passed in registers using the POWERPC Application Binary Interface (ABI) definitions. This ABI specifies an interface for compiled application programs to system software. In contrast to the PowerPC ABI, pass by reference parameters are avoided to or from **hca11()**. This minimizes the address translation problem parameters passed by reference would cause because address

translation is disabled automatically when interrupts are invoked. Input parameters may be indexes. Output parameters may be passed in the registers and require special in-line assembler code on the part of the caller. The first parameter in the Hypervisor call function table to **hcall()** is the function token. The assignment of function token is designed such that a single mask operation can be used to validate the value to be within the range of a reasonable size branch table. Entries within the branch table can handle unimplemented code points. And some of the **hcall()** functions indicate if the system is in LPAR mode and which ones are available. The Open Firmware property is provided in the */rtas* node of the partition's device tree. The property is present if the system is in LPAR mode while its value specifies which function sets are implemented by a given implementation. If the system implements any **hcall()** of a function set it implements the entire function set. Additionally, certain values of the Open Firmware property indicate that the system supports a given architecture extension to a standard **hcall()**.

The Hypervisor routines are optimized for execution speed. In some rare cases, locks will have to be taken, and short wait loops will be required due to specific hardware designs. However, if a needed resource is truly busy, or processing is required by an agent, the Hypervisor returns to the caller, either to have the function retried or continued at a later. The performance class establishes specific performance against specific **hcall()** function.

3.0.2 Hypervisor Call Functions

The Hypervisor provides the following functions:

- **Page Frame Table**

Page Frame Table (PFT) access is called using 64-bit linkage conventions. The Hypervisor PFT access functions carefully update a Page Table Entry (PTE) with at least 64-bit store operations since an invalid update sequence could result in machine check. The Hypervisor protect from checkstop condition by allocating certain PTE bits for PTE locks and reserved for operating system is to assume that the PTE is in use.

For logical addressing, an additional level of virtual addresses translation is managed by the Hypervisor. The operating system is not allowed to use the physical address for its memory this includes main storage, memory-mapped I/O (MMIO) space, and NVRAM. The operating system sees main storage as regions of contiguous logical memory. Each logical region is mapped by the Hypervisor into a corresponding block of contiguous physical memory on a specific node. All regions on a specific system are the same size though different systems with different amount of memory may have different region sizes since they are the quantum of

memory allocation to partitions. That is, partitions are granted memory in region size chunks and if a partition's operating system gives up memory, it is in units of a full region.

- **Translation Control Entry**

Translation Control Entry (TCE) access `hca11()` and take as a parameters in the Logical I/O Bus Number (LIOBN) which is the logical bus number value derived from the property that are associated with the particular I/O adapter. TCE is responsible for the I/O address to memory address translation in order to perform direct memory access (DMA) transfers between memory and PCI adapters. The TCE tables are allocated in the physical memory.

- **Processor Register Hypervisor Resource Access**

Processor Register Hypervisor Resource Access provides controlled in the write access services.

- **Debugger Support**

Debugger support provides the capability for the real mode debugger to be able to get to its async port and beyond the real mode limit register without turning on virtual address translation.

- **Virtual Terminal Support**

The Hypervisor provides console access to every logical partition without a physical device assigned. The console emulates a vt320 terminal that can be used to access partition system using the Hardware Management Console. Some functions are limited, and the performance cannot be guaranteed because of the limited bandwidth of the connection between the HMC and the managed system. A partition's device tree that contains one or more nodes notifying that is has been assigned to one or more virtual terminal client adapters. The unit address of then node is used by the partition to map the virtual device(s) to the operating system's corresponding logical representations and notify the partition that the virtual adapter is a Vterm client adapter. The node's interrupts property specifies the interrupt source number that has been assigned to the client Vterm I/O adapter for receive data.

- **Dump Support**

This allow the operating system to dump Hypervisor data areas in support of field problem diagnostic the `hcall-dump` function set contains the `H_HYPERVISOR_DATA hca11()`. This `hca11()` is enabled or disabled (default disabled) via the Hardware Management Console.

- **Memory Migration Support**

The Memory Migration Support `hca11()` was provided to assist the operating system in the memory migration process. It is the responsibility

of the operating system not to change the DMA mappings referenced by the translation buffer. Failure of the operating system to serialize relative to the logical bus numbers may result DMA data corruption within the caller's partition.

– Performance Monitor Support

The performance registers will be saved when a virtual processor yields or is preempted. They will be restored when the state of the virtual processor is restored on the hardware. A bit in one of the performance monitor registers will enable the partition to specify whether the performance monitor registers count when a Hypervisor call (except yield) is made (MSR[HV]=1). When a virtual processor yields or is preempted, the performance monitor registers will not count. This will allow a partition to query the Hypervisor to appropriate information regarding Hypervisor code and data addresses.

Table 3-1 provides the list of Hypervisor calls:

Table 3-1 Hypervisor calls

| Hypervisor Call | Definition |
|-----------------|---|
| H_REGISTER_VPA | This hcall() provides a data area registered with the Hypervisor by the operating system for each virtual processor. The VPA is the control area which contains information used by Hypervisor and the operating system in cooperation with each other. |
| H_CEDE | This hcall() is to have the virtual processor, which has no useful work to do, enter a wait state ceding its processor capacity to other virtual processor until some useful work appears, signaled either through an interrupt or a H_PROD hcall() . |
| H_CONFER | This hcall() allows a virtual processor to give its cycles to one or all other virtual processors in its partition. |
| H_PROD | This hcall() makes the specific virtual processor “runnable”. |
| H_ENTER | This hcall() adds an entry into the page frame table. PTE high and low order bytes of the page table contains the new entry. |

| Hypervisor Call | Definition |
|-----------------|---|
| H_PUT_TCE | This hcall() provides mapping of a single 4096 byte page into the specified TCE. |
| H_READ | This hcall() returns the contents of a specific PTE in GPR4 and GPR5. |
| H_REMOVE | This hcall() is for invalidating an entry in the page table. |
| H_BULK_REMOVE | This hcall() is for invalidating up to four entries in the page table. |
| H_GET_PPP | This hcall() returns the partition's performance parameters. |
| H_SET_PPP | This hcall() allows the partition to modify its entitled processor capacity percentage and variable processor capacity weight within limits. |
| H_CLEAR_MODE | This hcall() clears the modified bit in the specific PTE. The second double word of the old PTE is returned in GPR4. |
| H_CLEAR_REF | This hcall() clears the reference bit in the specific PTE from the partition's node Page Frame Table. |
| H_PROTECT | This hcall() sets the page protects bits in the specific PTE. |
| H_EOI | This hcall() incorporates the interrupt reset function when specifying an interrupt source number associated with an interpartition logical I/O adapter. |
| H_IPI | This hcall() generates an interprocessor interrupt. |
| H_CPPR | This hcall() sets the processor's current interrupt priority. |
| H_MIGRATE_DMA | This hcall() is extended to serialize the sending of a logical LAN message to allow for migration of TCE mapped DMA pages. |

| Hypervisor Call | Definition |
|-----------------------|--|
| H_PUT_RTCE | This hca11() maps the number of contiguous TCEs in an RTCE to the same number of contiguous I/O adapter TCEs. |
| H_PAGE_INIT | This hca11() initializes pages in real mode either to zero or to the copied contents of another page. |
| H_GET_TCE | This standard hca11() s used to manage the interpartition logical LAN adapters's I/O translations. |
| H_COPY_RDMA | This hca11() copies data from an RTCE table mapped buffer in one partition to an RTCE table mapped buffer in another partition, with the length of the transfer being specified by the transfer length parameter in the hca11() . |
| H_SEND_CRQ | This hca11() sends one 16 byte message to the partner partition's registered Command / Response Queue (CRQ). The CRQ facility provides ordered delivery of messages between authorized partitions. |
| H_SEND_LOGICAL_LAN | This hca11() sends a logical LAN message. |
| H_ADD_LOGICAL_LAN_BUF | This hca11() adds receive buffers to the logical LAN receive buffer pool. |
| H_PIC | This hca11() returns the summation of the physical processor pool's idle cycles. |
| H_XIRR | This hca11() is extended to report the virtual interrupt source number associated with virtual interrupts associated with an interpartition logical LAN I/O adapter. |
| H_POLL_PENDING | This hca11() provides the operating system with the ability to perform background administrative functions and the implementation with indication of pending work so that it may more intelligently manage the use of hardware resources. |

| Hypervisor Call | Definition |
|---------------------|--|
| H_PURR ^a | This hca11() is a new resource provided for Micro-Partitioning and SMT. It provides an actual count of ticks that the shared resource has used on a per virtual processor or per SMT thread basis. In the case of Micro-Partitioning, the virtual processor's Processor Utilization Resource Register (PURR) begins incrementing when the virtual processor is dispatched onto a physical processor. Therefore, comparisons of elapsed PURR with elapsed Timebase provides an indication of how much of the physical processor a virtual processor is getting. The PURR will also count Hypervisor calls made by the partition, with the exception of H_CEDE and H_CONFER . For improved accuracy, the existing hca11() time stamping should be converted to use PURR instead of timebase. |

a. For a description of the Performance Utilization Resource Register, see "Processor Utilization Resource Register" on page 132.

The *lparstat* command in AIX 5L Version 5.3 with -H flag will display the partition data with detailed breakdown of Hypervisor time by call type as shown in Figure 3-3 on page 49.

System configuration: type=Shared mode=Capped smt=On lcpu=2 mem=512 psize=- ent=0.30

Detailed information on Hypervisor Calls

| Hypervisor Call | Number of Calls | %Total Time Spent | %Hypervisor Time Spent | Avg Call Time(ns) | Max Call Time(ns) |
|---------------------|-----------------|-------------------|------------------------|-------------------|-------------------|
| remove | 11488 | 0.0 | 12.0 | 403 | 3613 |
| read | 81 | 0.0 | 0.1 | 293 | 632 |
| nclear_mod | 0 | 0.0 | 0.0 | 1 | 0 |
| page_init | 228 | 0.0 | 0.5 | 898 | 2797 |
| clear_ref | 0 | 0.0 | 0.0 | 1 | 0 |
| protect | 0 | 0.0 | 0.0 | 1 | 0 |
| put_tce | 7851 | 0.0 | 11.8 | 576 | 1256 |
| xirr | 99 | 0.0 | 0.2 | 616 | 1859 |
| eoi | 95 | 0.0 | 0.1 | 437 | 589 |
| ipi | 0 | 0.0 | 0.0 | 1 | 0 |
| cppr | 94 | 0.0 | 0.1 | 328 | 478 |
| asr | 0 | 0.0 | 0.0 | 1 | 0 |
| others | 0 | 0.0 | 0.0 | 1 | 0 |
| enter | 13251 | 0.0 | 12.7 | 368 | 3130 |
| cede | 1076 | 0.0 | 59.4 | 21248 | 3507400 |
| migrate_dma | 0 | 0.0 | 0.0 | 1 | 0 |
| put_rtce | 0 | 0.0 | 0.0 | 1 | 0 |
| confer | 0 | 0.0 | 0.0 | 1 | 0 |
| prod | 858 | 0.0 | 1.0 | 439 | 772 |
| get_ppp | 9 | 0.0 | 0.0 | 1791 | 3135 |
| set_ppp | 0 | 0.0 | 0.0 | 1 | 0 |
| purr | 0 | 0.0 | 0.0 | 1 | 0 |
| pic | 9 | 0.0 | 0.0 | 391 | 618 |
| bulk_remove | 507 | 0.0 | 1.7 | 1285 | 2241 |
| send_crq | 125 | 0.0 | 0.4 | 1295 | 1584 |
| copy_rdna | 0 | 0.0 | 0.0 | 1 | 0 |
| get_tce | 0 | 0.0 | 0.0 | 1 | 0 |
| send_logical_lan | 0 | 0.0 | 0.0 | 1 | 0 |
| add_logical_lan_buf | 0 | 0.0 | 0.0 | 1 | 0 |

Figure 3-3 lparstat -H command output

3.0.3 Micro-Partitioning Logical Partition Hypervisor Extensions

A new virtual processor is dispatched on a physical processor when one of the following conditions happens:

- The physical processor is idle and a virtual processor was made ready to run (interrupt or process).
- The old virtual processor exhausted its time slice (HDERC interrupt).
- The old virtual processor ceded or conferred its cycles.

When one of the above conditions occurs, the Hypervisor, by default, records all the virtual processor architected state including the Time Base and Decrementer values and sets the Hypervisor timer services to wake the virtual processor per

the setting of the decremter. The virtual processor's Processor Utilization Resource Register (PURR) value for this dispatch is computed. The Virtual Processor Area (VPA) dispatch count is incremented (such that the result is odd). Then the Hypervisor selects a new virtual processor to dispatch on the physical processor using an implemented dependent algorithm having the following characteristics given in priority order:

1. The virtual processor is "ready to run" (has not ceded or conferred its cycles or exhausted its time slice).
2. Ready-to-run virtual processors are dispatched prior to waiting in excess of their maximum specified latency.
3. Of the non-latency critical virtual processors ready to run, select the virtual processor that is most likely to have its working set in the physical processor's cache or for other reasons will run most efficiently on the physical processor.

If no virtual processor is "ready to run" at this time, start accumulating the Pool Idle Count (PIC) of the total number of idle processor cycles in the physical processor pool.

Virtual Input/Output

Virtual I/O support is one of the advanced features of the new Hypervisor. Virtual I/O provides a given partition with the appearance of I/O adapters that do not necessarily have direct correspondence with a physical adapter. Virtual I/O is covered in detail in section 7.1.1, "Hypervisor support to Virtual I/O" on page 174

Memory Considerations

POWER5 processors use memory to temporarily hold information. Memory requirements for partitions depend on partition configuration, I/O resources assigned, and applications used. Memory can be assigned in increments of 16MB.

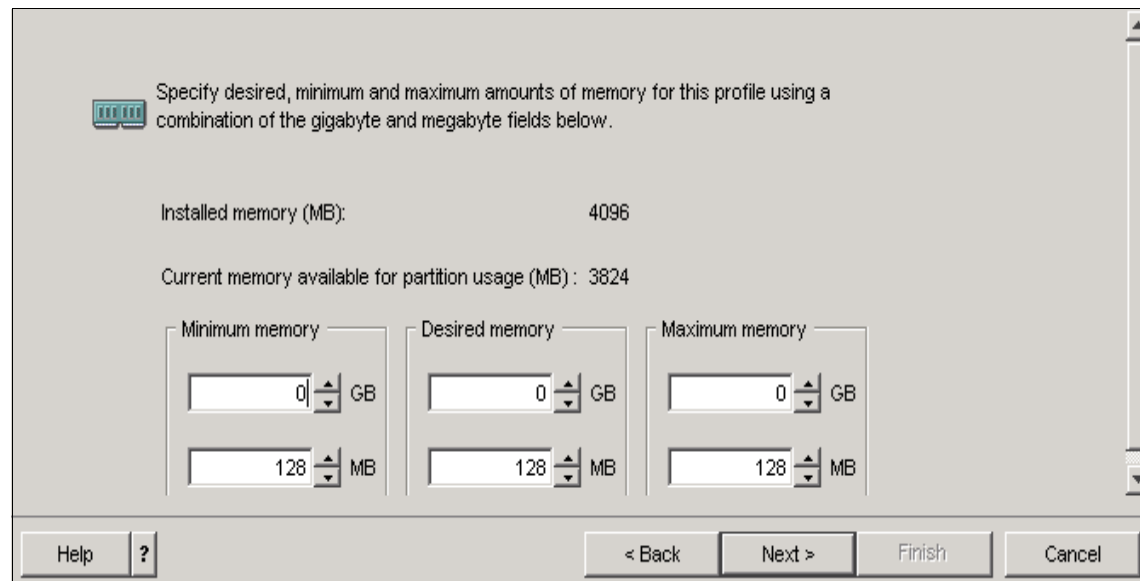
Depending on the overall memory in your system and the maximum memory values you choose for each partition, the server firmware must have enough memory to perform logical partition tasks. Each partition has a Hardware Page Table (HPT). The size of the HPT is based on an HPT ratio and determined by the maximum memory values you establish for each partition. The HPT ratio is 1/64.

When selecting the maximum memory values for each partition, consider the following:

- Maximum values affect the HPT size for each partition.
- The logical memory map size of each partition.

When you create a logical partition on your managed system, the managed system reserved an amount of memory to manage the logical partition. Some of this physical partition is used for Hypervisor page table translation support. The current memory available for partition usage in the HMC is the amount of memory that is currently available to the logical partitions on the managed system, see Figure 3-4. This is the amount of active memory on your managed system minus the estimated memory needed by the managed system to manage the logical partitions currently defined on your system. Therefore, the amount in this field decreases for each additional logical partition you create.

When you are assessing changing performance conditions across system reboots, it is important to know that memory allocations might change based on the availability of the underlying resources. Memory is allocated by the system across the system. Applications in partitions cannot determine where memory has been physically allocated.



The screenshot shows a window titled "Specify desired, minimum and maximum amounts of memory for this profile using a combination of the gigabyte and megabyte fields below." The window displays the following information:

- Installed memory (MB): 4096
- Current memory available for partition usage (MB) : 3824

Below this information are three columns for configuring memory settings:

| Minimum memory | Desired memory | Maximum memory |
|-------------------------------------|-------------------------------------|-------------------------------------|
| <input type="text" value="0"/> GB | <input type="text" value="0"/> GB | <input type="text" value="0"/> GB |
| <input type="text" value="128"/> MB | <input type="text" value="128"/> MB | <input type="text" value="128"/> MB |

At the bottom of the window are buttons for "Help", "?", "< Back", "Next >", "Finish", and "Cancel".

Figure 3-4 Current memory available for partition usage using HMC

3.0.4 POWER Hypervisor Design

The Hypervisor is primarily responsible for affinity in a Micro-Partitioning system. The pools of shared physical processors are to be grouped within natural hardware boundaries, such that all processors within the pool have the same affinity characteristics and the partition is guaranteed to only execute on that pool of processors, barring events such as a processor being GUARD'ed off due to predictive failures, and possibly replaced with a spare processor from another affinity domain. See Figure 6-10 on page 150 for the relationship between virtual and physical processors.

The Hypervisor will continue to provide affinity domain information in the device tree for processors, which are actually virtual processors in a Micro-Partitioning configuration. The side effect if Micro-Partitioning might be limits to the depth of hierarchy of affinity domain information that can be provided, i.e., instead of going down to the physical processor it might stop at the lowest common layer of all processors in the shared pool. The Hypervisor will do its best to re-dispatch a partition to the same physical processor that ran on previously, in order to maximize cache affinity. If a physical processor is available, the Hypervisor will dispatch a virtual processor to the last processor, the last MCM, and so on.

Saved/Restored Registers

The Hypervisor will save the following registers when a state is saved for a virtual processor: GPRs, FPRs, CR, XER, LR, CTR, ACCR, SPRG0, SPRG1, SPRG2, SPRG3, ASR, SLB state, DAR, DEC, DSISR, SRR0, SRR1, PMCs, MMCR0/1/A, SDAR, DABR and SDR1.

The Instruction Match Cam (IMC) facility on the POWER4™ processors (in part used for setting performance monitoring modes) is not a Hypervisor resource. It does not lend itself to easy virtualization as software cannot read what was written to the IMC. Hypervisor call support is required for proper functioning in shared processor environment.

Preemption of a Virtual Processor

The Hypervisor is responsible for time slicing and managing the dispatching of the partitions across the physical processors. One of the features of the POWER4+ and POWER5 which make this possible is the Hypervisor decremter (HDECR). This is a clock interrupt source utilized by the Hypervisor to preempt a dispatched partition and regain control of the physical processors. This interrupt will occur even if external interrupts are disabled and can not be masked by the partition. The Hypervisor utilizes this **HDECR** to drive its partition dispatcher. So in reality, the Hypervisor is managing the execution of multiple partition images across the same physical resources, just as an operating

system manages the execution of multiple processes / threads within its partition instance.

The POWER4+ processor does not have support for the Hypervisor decremter. The SRR0 and SRR1 registers are used to present a HDECR interrupt to the processor. To avoid loss of partition state, a pending HDECR interrupt will be held off for N (programmable hardware value) cycles if $MSR[RI]=0$. The number of cycles (N) has to be large enough to allow a partition to safely execute instructions until SRR0 and SRR1 are saved and indicated by the setting of $MSR[RI]=1$. If not, taking the HDECR interrupt would result in the corresponding loss of state because these registers are updated when an interrupt/exception occurs.

Note: For those not familiar with the POWER and PowerPC architecture, at the time of an exception or interrupt, SRR0 is loaded with either the address of the instruction that caused the exception, or the address of the instruction that would have been dispatched had the interrupt not occurred. SRR1 contains the Machine State Register (MSR) contents at the time of the exception or interrupt. For example, the thread being preempted may have been in user mode ($MSR[PR]=1$). In order for the interrupt to be serviced, the processor must be in supervisory (system) mode and this bit has to be cleared (0). If interrupts are not masked or held off, then the processor automatically saves off the current MSR into SRR1 and produces a new MSR value with appropriate bit settings is placed into the MSR. Therefore, if these registers are not saved, recovery may be impossible. The $MSR[RI]$ bit is not affected by exceptions or interrupts and can be used by the operating system to indicate recovery.

This places the requirement on the operating system to use the $MSR[RI]$ bit to avoid fatal failures that could occur because of Hypervisor preemption of a virtual processor.

POWER5-based server provides complete HDECR support that allow preemption with an unsaved SRR0 and SRR1.

The Hypervisor issues a **sync** instruction on the processor when it preempts a virtual processor. This ensures that a storage access sequence (in particular an Memory Mapped I/O sequence) by the preempted virtual processor is seen by the devices on the system in the order it was intended. The Hypervisor will also do the equivalent of a dummy **stwcx** instruction to cancel a reservation that may be held by the yielding or preempted virtual processor.

Cache Invalidation

The *Segment Lookaside Buffer* (SLB) which was saved when the virtual processor yielded or was preempted is restored on each dispatch of a virtual processor. There is one SLB per thread (two per processor core). Information derived from the SLB may also be cached in the instruction and / or data *Effective to Real Address Translation* (ERAT) along with information from the Translation Lookaside Buffer (TLB).

The TLB of a processor is invalidated every time the partition ID of a virtual processor switched in on a processor is different from the partition ID of the virtual processor that last ran on it. The POWER4™ family of processors provides an instruction to flush the TLB of a processor avoiding the need for a broadcast of TLB invalidations.

Since the number of partitions exceeds the number of hardware partition IDs, shared processor partitions may share a hardware partition ID. This can lead to false invalidations of TLB entries. Since the TLB is flushed in many instances on a dispatch of a virtual processor dispatch, the false invalidations are not a concern.

When a partition is IPLed (rebooted) in the shared pool, all processors in the pool flush their instruction cache prior to switching in a virtual processor from the partition being IPLed.

Hypervisor Dispatching Algorithm

Each shared pool has its own instantiation of the Hypervisor dispatcher. The Hypervisor will use a fixed scheduling window size of T time unit (=10 msec) to allocate processor cycles and guarantee that each virtual processor receives its share of the entitlement in timely fashion. If a partition does not use its allocation of cycles in a scheduling window, it will lose the unused cycles. The minimum allocation of resource is one msec per processor; the Hypervisor will calculate number of msec using the capacity entitlement and the number of virtual processors for each shared pool. Once capped shared processor has received its capacity entitlement within a dispatch interval, it becomes not-runnable. An uncapped partition may get more than its allocation of cycles in a scheduling window. Virtual processors are time sliced through the use of the hardware Decrementer much like the operating system time slices threads. The Hypervisor decrementer and time base will be used by the Hypervisor dispatcher for virtual processor accounting.

The physical processor resource in a shared pool may become over committed (with respect to uncapped partitions). A suitable variation of the TFHS (Time Function History Scheduling) algorithm will be used for making dispatch decisions when the pool is over committed. The algorithm need some notion of priority when making scheduling decisions.

Dispatching and Interrupt Latencies

Virtual processors have dispatch latency, since they are scheduled. When a virtual processor is made runnable, it is placed on a run queue by the Hypervisor, where it sits until it is dispatched. The time between these two events is referred to as dispatch latency.

The dispatch latency of a virtual processor is a function of the partition entitlement and the number of virtual processors that are online in the partition. Entitlement is equally divided amongst these online virtual processors, so the number of online virtual processors impacts the length of each virtual processor's dispatch. The smaller the dispatch cycle the greater the dispatch latency.

Timers have latency issues also. The hardware Decrementer is virtualized by the Hypervisor at the virtual processor level, so that timers will interrupt the initiating virtual processor at the designated time. If a virtual processor is not running, then the timer interrupt has to be queued with the virtual processor, since it is delivered in the context of the running virtual processor.

External interrupts have latency issues also. External interrupts are routed directly to a partition. When the operating system makes the accept-pending-interrupt Hypervisor call, the Hypervisor, if necessary, dispatches a virtual processor of the target partition to process the interrupt. The Hypervisor provides a mechanism for queuing up external interrupts that is also associated with virtual processors. Whenever this queuing mechanism is used, latencies are introduced.

These latency issues are not expected to cause functional problems, but they may present performance problems for real time applications. To quantify matters, the worst case virtual processor dispatch latency is 18 msec, since the minimum dispatch cycle that is supported at the virtual processor level is one msec. This figure is based on the Hypervisor dispatch wheel. It can be easily visualized by imagining that a virtual processor is scheduled in the first and last portions of two 10 msec intervals. In general, if these latencies are too great, then clients may increase entitlement, minimize the number of online virtual processors without reducing entitlement, or use dedicated processor partitions.

3.0.5 Performance Considerations

The Hypervisor does use a small percentage of the processor and memory resources. The processor and memory resources are used for the Hypervisor dispatcher, virtual processor data structures (including save areas for virtual processor), virtual memory management and for queuing up of interrupts. This should be minor for most workloads, but the impact increases with extensive amounts of page-mapping activity. Partitioning may actually help performance in some cases for applications that do not scale well on large SMP systems by

enforcing strong separation between workloads running in the separate partitions.

Specifically, the Hypervisor needs processor and memory resources to:

- Dispatch threads to virtual processors (i.e. saving and restoring state).
- Translation Lookaside Buffer (TLB) management. The codepath necessary to map the address space for the dispatched thread.
- Cache management.

The output of **lparstat** with **-h** flag will display the percentage spent in Hypervisor (%hypv) and the number of **hcalls**. Notice from the example output shown in Figure 3-5 on page 56, the %hypv in relation to entitlement capacity is only around 1% of the system resources. This shows that the Hypervisor consumes a small amount of the processor during this sample.

```
# lparstat -h 1 16
```

System configuration: type=Shared mode=Capped smt=On lcpu=2 mem=512 psize=- ent=0.30

| %user | %sys | %wait | %idle | physc | %entc | lbusy | app | vcsw | phint | %hypv | hcalls |
|-------|------|-------|-------|-------|-------|-------|-----|------|-------|-------|--------|
| 15.6 | 76.6 | 0.0 | 7.9 | 0.30 | 100.3 | 90.5 | - | 298 | 2 | 1.4 | 230 |
| 15.5 | 76.8 | 0.0 | 7.7 | 0.30 | 100.0 | 90.5 | - | 321 | 1 | 0.9 | 259 |
| 15.6 | 76.3 | 0.0 | 8.1 | 0.30 | 100.0 | 85.0 | - | 295 | 1 | 1.9 | 224 |
| 15.6 | 76.6 | 0.0 | 7.9 | 0.30 | 100.0 | 89.5 | - | 310 | 5 | 1.3 | 246 |
| 15.6 | 76.7 | 0.0 | 7.7 | 0.30 | 100.0 | 91.5 | - | 299 | 2 | 1.0 | 220 |
| 15.6 | 76.6 | 0.0 | 7.8 | 0.30 | 100.0 | 90.0 | - | 315 | 1 | 1.2 | 249 |
| 15.4 | 75.9 | 0.0 | 8.7 | 0.30 | 99.1 | 91.6 | - | 315 | 2 | 1.3 | 249 |
| 10.0 | 49.6 | 0.0 | 40.4 | 0.19 | 64.9 | 58.5 | - | 442 | 1 | 1.0 | 507 |
| 0.0 | 0.4 | 0.0 | 99.6 | 0.00 | 1.1 | 0.0 | - | 383 | 1 | 0.8 | 456 |
| 0.0 | 0.3 | 0.0 | 99.6 | 0.00 | 1.0 | 0.0 | - | 332 | 0 | 0.7 | 397 |
| 0.0 | 0.4 | 0.0 | 99.6 | 0.00 | 1.0 | 0.0 | - | 334 | 0 | 0.8 | 399 |
| 0.0 | 0.3 | 0.0 | 99.7 | 0.00 | 0.9 | 0.0 | - | 330 | 0 | 0.7 | 391 |
| 0.0 | 0.3 | 0.0 | 99.6 | 0.00 | 0.9 | 0.0 | - | 330 | 0 | 0.7 | 391 |
| 0.0 | 0.3 | 0.0 | 99.6 | 0.00 | 1.0 | 0.0 | - | 335 | 0 | 0.7 | 409 |
| 0.0 | 0.3 | 0.0 | 99.6 | 0.00 | 1.0 | 4.9 | - | 356 | 0 | 0.7 | 425 |
| 0.0 | 0.3 | 0.0 | 99.7 | 0.00 | 0.9 | 0.0 | - | 324 | 0 | 0.7 | 390 |

#

Figure 3-5 *lparstat -h 1 16* command output

To provide input to the capacity planning and quality of service tools, the Hypervisor reports to an operating system certain statistics, these include the number of virtual processor that are online, minimum processor capacity that the operating system can expect (the operating system may cede any unused capacity back to the system), the maximum processor capacity that the partition will grant to the operating system, the portion of spare capacity (up to the maximum) that the operating system will be granted, variable capacity weight, and the latency to a dispatch via an **hcall** (). The output of the **lparstat**

command with the `-i` flag, shown in Figure 3-6 on page 57, will report the logical partition related information.

```
squadron:root[/]lparstat -i
Node Name                : sq1test1
Partition Name           : Test1_AIX_0425A
Partition Number         : 2
Type                     : Shared-SMT
Mode                     : Capped
Entitled Capacity        : 30
Partition Group-ID       : 32770
Shared Pool ID           : 0
Online Virtual CPUs      : 1
Maximum Virtual CPUs     : 5
Minimum Virtual CPUs     : 1
Online Memory            : 512 MB
Maximum Memory           : 1024 MB
Minimum Memory           : 128 MB
Variable Capacity Weight : 0
Minimum Capacity         : 10
Maximum Capacity         : 50
Capacity Increment       : 1
Maximum Dispatch Latency : 13999999
Maximum Physical CPUs in system : 5
Active Physical CPUs in system : 2
Active CPUs in Pool      : -
Unallocated Capacity     : 0
Physical CPU Percentage   : 30.00%
Unallocated Weight       : 0
Minimum Virtual Processor Required Capacity: 10
squadron:root[/]
```

Figure 3-6 `lparstat -i` command output

4

Operating system support

In this chapter we will discuss what is new in AIX 5L Version 5.3 from a performance and POWER5 point of view in the following sections:

- ▶ Physical and logical processors
- ▶ Simultaneous multithreading
- ▶ Metrics problems
- ▶ Updated and new performance commands
- ▶ Logical Volume Manager
- ▶ Paging space
- ▶ Physical and virtual networks

4.1 AIX 5L Version 5.3

The appropriate version of AIX for POWER5 is AIX 5L Version 5.3. This means that this version has modifications to acknowledge the new functionalities of the POWER5 processor. AIX 5L Version 5.2 is also supported but will not be able to use the new features of POWER5. Versions prior to AIX 5L Version 5.2 are not supported.

4.1.1 Introduction

The implementation of the virtual processor abstraction is in the hardware and in the Hypervisor. From an operating system perspective, a virtual processor is indistinguishable from a physical processor, unless there is an enhancement to the operating system to make it aware of the difference.

4.1.2 Processors

Logical processors

Before AIX 5L v5.3 a partition was allocated to dedicated processors, which were entire physical processors. With virtualization the concept of a logical processor is used.

Author Comment: XXXRef Add reference to logical processors in the Hypervisor

It is the maximum number of operations the operating system can run at a time. A logical processor is seen by the operating system as a usual processor but is a hardware thread dispatched on a physical processor. Changes to ODM have been made to reflect the new type of processors, Example 4-1 on page 60 shows attributes of a POWER4 processor and Example 4-2 on page 61 shows two new attributes for a POWER5 processor. The attributes of processors are exactly the same if the operating system is running with dedicated processor or if it is running in Micro-Partitioning.

Example 4-1 Attributes of POWER4 processor

| | | | |
|------------------|----------------|-----------------|-------|
| <hr/> | | | |
| lsattr -El proc3 | | | |
| frequency | 1499960128 | Processor Speed | False |
| state | enable | Processor state | False |
| type | PowerPC_POWER4 | Processor type | False |
| <hr/> | | | |

Example 4-2 Attributes of POWER5 processor

| | | | |
|--------------------|----------------|-----------------------|-------|
| # lsattr -El proc0 | | | |
| frequency | 1656424000 | Processor Speed | False |
| smt_enabled | false | Processor SMT enabled | False |
| smt_threads | 2 | Processor SMT threads | False |
| state | enable | Processor state | False |
| type | PowerPC_POWER5 | Processor type | False |

Optimizations

For the most part, AIX 5L should be able to run and function on a Micro-Partitioning system with no changes. However, in order to optimize the OS performance as well as the collective performance of all shared partitions, it is important for the OS to add some specific Micro-Partitioning optimizations. These optimizations involve giving up the processor in the IDLE process so that another virtual processor within our partition might use it, or even so that another partition could use it.

We can call two Hypervisor functions to control those optimizations:

- H_CEDE** Used to give processor cycles to the pool.
- H_PROD** Used to restore processor cycles to the processor that has ceded them.

Simultaneous Multithreading (SMT)

On AIX 5L v5.3 with SMT enabled, each hardware thread is supported as a separate logical processor. A dedicated partition with one physical processor is seen under AIX as a partition with two logical processors as shown in Figure 4-1 on page 62. The same applies to Micro-Partitioning, a three virtual processor partition is configured by AIX as a logical 6-way partition. The two hardware threads are also called sibling threads.

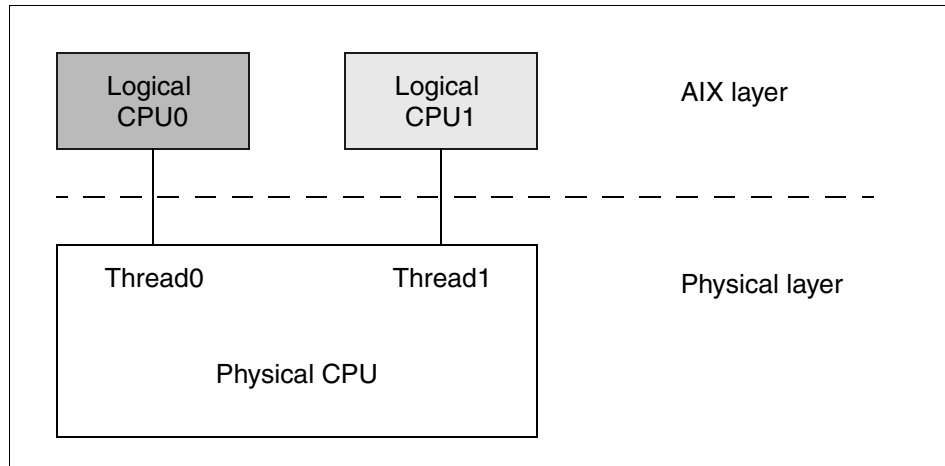


Figure 4-1 Logical versus physical processors

SMT mode can be enabled or disabled dynamically with **smtctl** command. The change can also be made at next boot and persists across system boots. By default SMT mode is enabled. The syntax of **smtctl** command is:

```
smtctl [ -m off | on [ -w boot | now]]
```

When you enter the command without any flags it returns information on the status of SMT on your system as shown in Example 4-3 on page 62.

Example 4-3 smtctl command example

```
# smtctl
```

```
This system is SMT capable.
```

```
SMT is currently enabled.
```

```
SMT boot mode is not set.
```

```
Processor 0 has 2 SMT threads
SMT thread 0 is bound with processor 0
SMT thread 1 is bound with processor 0
```

The boot image includes an indicator for SMT mode, if SMT mode is changed, boot image must be remade, otherwise at the next reboot the SMT mode will be the same as the previous boot.

Normally, AIX 5L maintains sibling threads at the same priority but will boost or lower thread priorities in a few key places to optimize performance. AIX 5L lowers

thread priorities, when the thread is doing non-productive work spinning in the idle loop or on a kernel lock. When a thread is holding a critical kernel lock, AIX 5L boosts the thread priorities. These priority adjustments do not persist into user mode. AIX 5L does not consider a software thread is dispatching priority, when choosing its hardware thread priority.

There where also made several scheduling enhancements to exploit SMT. For example, work will be distributed across all primary threads before work is dispatched to secondary threads. The reason for this enhancement is that the performance of a thread is best when its sibling thread is idle. AIX 5L also considers thread affinity in idle stealing and periodic run queue load balancing.

For detailed information on SMT refer to Chapter 5, “Simultaneous Multi-Threading (SMT)” on page 89.

Metrics Problems

A dedicated partition that is created with one real processor is configured by AIX 5L as a logical 2-way by default. This is independent of the partition type, so a shared partition with two virtual processors is configured by AIX 5L as a logical four-way by default. Logically, the only supported kernel in a SMT environment is the multiprocessor (MP).

In traditional processor utilization, data collection is sample based. There are 100 samples per second sorted into four categories:

| | |
|---------------|---|
| user | Interrupted code outside AIX 5L kernel. |
| sys | Interrupted code inside AIX 5L kernel and currently running thread is not waitproc. |
| iowait | Currently running thread is waitproc and there is an I/O pending. |
| idle | Currently running thread is waitproc and there is no I/O pending. |

Each sample corresponds to a 10 ms (1/100 sec.) clock tick. These are recorded in the **sysinfo** (system-wide) and **cpuinfo** (per-processor) kernel data structures and in order to preserve binary compatibility, this stays unchanged with AIX 5L v5.3.

Note: Performance tools like **vmstat**, **iostat** or **sar**, convert tick counts from the **sysinfo** structure into utilization percentages for machine/partition. Other tools like **sar -P ALL** and the **topas** “hot cpu” section there is a conversion of tick counts from **cpuinfo** into utilization percentages for a processor/thread.

This of course affects greatly on the metrics. Traditional utilization metrics are misleading because the tools believe there are two physical processors when in fact we only have one. As an example, one thread 100% busy and one thread idle would result in 50% utilization but the physical processor is really 100% busy. This is similar to what happened with Hardware Multi-Threading (HMT) and the same problem exists with *hyperthreading*.

Processor Utilization Resource Register (PURR)

<Move to POWER5 chapter>

A mechanism was added to track physical processor utilization in micro-partitioning and SMT mode called the Processor Utilization Resource Register (PURR).

Author Comment: Move to POWER5 chapter

Each hardware thread has its own PURR and only one of the two PURR gets incremented at each cycle. The units are the same as the timebase register and the sum of the PURR values for both threads is equal to timebase register.

The PURR increments measure instruction dispatch cycles. There is a different way to collect the data.

Each thread collects 100 utilization samples per second, which will still be collected in per-logical processor cpuinfo structures (for binary compatibility), but additional state-based PURR-based metrics will be collected in new structures and sorted in the same four categories. This way at each cycle, only one of the two PURRs gets incremented.

Author Comment: Add XXXref to chapter and section talking about PURR register

The displayed %user, %sys, %idle, %wait will now be calculated using the PURR-based metrics. Using the previous example where one thread is 100% busy and the other is idle then reported utilization would no longer be 50% but the correct 100%. This is because one thread would receive (almost) all the PURR increments, the other (practically) none, meaning 100% of PURR increments would go into the %user and %sys buckets. This is a more

reasonable indicator of the split of the work between the two threads. Unfortunately, this hides the SMT gain.

New metrics

We now show the new metrics on AIX 5L v5.3 with SMT. We have two different times to measure: the thread's processor time and the elapsed time. For the first, we use thread's PURRs, which are now virtualized. To measure the elapsed time we still use the Timebase register (TB).

For the physical resource utilization metric for a logical processor we use **(delta PURR/delta TB)** which represents the fraction of the physical processor consumed by a logical processor and **((delta PURR/delta TB)*100)** over an interval to represent the percentage of dispatch cycles given to a logical processor.

Using PURR-based samples and entitlement, we calculate the "physical" processor utilization metrics. As an example we have:

$$\%sys = (\text{delta PURR in system mode} / \text{entitled PURR}) * 100$$

where **entitled PURR = (ENT*delta TB)** and ENT is entitlement in number of processors (entitlement/100).

When we need to know how much physical processor is being consumed (PPC) we use **sum(delta PURR/delta TB)** for each logical processor in a partition. The result is in decimal number of processors.

We also may need the percentage of entitlement consumed: **(PPC/ENT)*100**.

Another useful metric is the available pool of processors. Taking the *Pool Idle Count* (PIC), which represents clock ticks where the Hypervisor was idle, that is, all partition entitlements are satisfied and there is no partition to dispatch, then we have **(delta PIC/delta TB)**.

This, also, results in decimal number of processors.

Logical Processor Utilization is useful to figure out if we should add more virtual processors to a partition and we calculate it by summing the old 10 ms tick-based %sys and %user.

There are two other usages for the PURR. The first is the measurement of the relative SMT split between threads and is just the ratio **purrr0/purrr1**. To know the fraction of time partition1 ran on a physical processor, i.e. the relative amount of processing units consumed use **(purrr0+purrr1)/timebase0**.

Binary compatibility

As with every release of AIX, the maintenance of the binary compatibility is a requirement. In a Micro-Partitioning LPAR, things like **bindprocessor** command continue to work, albeit binding to the virtual processor and not a physical processor. This aspect could possibly cause problems for an application or kernel extension, which is dependent on executing on a specific physical processor. For example, the AIX Floating Point Diagnostic Test unit relied on the ability to bind itself to and execute the FP test unit to completion on each physical processor in the system. Another example is the **bindintcpu** command, which allows an administrator to bind bus interrupt levels to specific processors. In Micro-Partitioning, AIX 5L v5.3 supports it, and will bound interrupts to virtual processors, however it will have no effect on the original intent of this command, which was to control the physical distribution of interrupts. The impact will be no absolute control over the routing of interrupts to physical processors when running in Micro-Partitioning mode. We do not expect to be a significant risk since that type of physical resource management does not make sense in a Micro-Partitioning environment, and workloads that require specific distribution of interrupts would probably not be candidates for running in a Micro-Partitioning environment.

There should also be an impact on third party performance tools due to resulting inconsistent or erroneous statistics unless those tools become Micro-Partitioning aware.

4.1.3 Dynamic re-configuration

Dynamic operations allows the addition or removal of resources from a logical partition (LPAR) without rebooting.

A dynamic remove operation on a CPU may fail for some reasons, the most common reason for a removal failure is because a process is bound to a processor. In order to provide more information to the user the **cpupstat** command was added. It helps to identify processes bound to logical processors. The Example 4-4 on page 66 shows that the **cpupstat** command first check WLM classes, then rset attachments and finally the logical processor number 2. If a process is bound to a processor, it can be unbound with the **bindprocessor** command. The highest bind ID is removed if processor dynamic removal operation succeeds.

Example 4-4 cpupstat output

```
# cpupstat -i 2
0 WLM classes have single CPU rsets with CPU ID 2.
0 processes have single CPU rset attachments with CPU ID 2.
0 processes are bound to bind ID 2.
```

4.1.4 Existing performance commands enhancement

Due to SMT, Micro-Partitioning and the ability to dynamically change some parameters it was necessary to make some changes to the old tools.

If SMT mode is active or in a Micro-Partitioning environment **vmstat**, **iostat** and **sar** commands automatically use the new PURR-based data and formula for %user, %sys, %wait and %idle.

In Micro-Partitioning mode new metrics are displayed, Example 4-5 on page 67 shows the traditional output of **vmstat** command run on a dedicated partition, Example 4-6 on page 67 shows the same **vmstat** command run that time on a Micro-Partition, the columns *pc* (physical processor consumed) and *ec* (entitled capacity consumed) are added.

Example 4-5 vmstat on a dedicated partition.

```
# vmstat 2
```

System configuration: 1cpu=2 mem=1024MB

| kthr | | memory | | | page | | | | faults | | | | cpu | | | |
|------|---|--------|--------|----|------|----|----|----|--------|----|----|-----|-----|----|----|----|
| r | b | avm | fre | re | pi | po | fr | sr | cy | in | sy | cs | us | sy | id | wa |
| 0 | 0 | 56057 | 196194 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 68 | 191 | 0 | 0 | 99 | 0 |
| 0 | 0 | 56058 | 196192 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 6 | 183 | 0 | 0 | 99 | 0 |

Example 4-6 vmstat on Micro-Partition.

```
# vmstat 2
```

System configuration: 1cpu=6 mem=512MB ent=0.3

| kthr | | memory | | | page | | | | faults | | | | cpu | | | | | |
|------|---|--------|-------|----|------|----|----|----|--------|----|----|-----|-----|----|----|----|------|-----|
| r | b | avm | fre | re | pi | po | fr | sr | cy | in | sy | cs | us | sy | id | wa | pc | ec |
| 0 | 0 | 46569 | 73370 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 73 | 149 | 0 | 1 | 99 | 0 | 0.00 | 1.5 |
| 0 | 0 | 46577 | 73360 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 147 | 0 | 0 | 99 | 1 | 0.00 | 1.1 |

The following list gives the new metrics for each commands:

► **vmstat**

- Number of physical processor consumed.
- Percentage of entitled capacity consumed.

► **iostat**

- Percentage of entitled capacity consumed.

- Percentage of physical processor consumed.
- **sar**
 - Number of physical processors consumed.
 - Percentage of entitled capacity consumed.
- **topas**
 - Number of physical processors consumed.
 - percentage entitlement consumed.
 - New display dedicated to logical processors.

All of these tools have a new feature called dynamic configuration support. They need it because we no longer work in a static environment with a fixed number of processors and memory. This way the tools start by a new pre-header with the configuration but if the configuration changes then there is a warning. After it, the tool prints the current iteration line, followed by summary line in **sar** case. The tool shows a new configuration pre-header and the regular header for the tool and continues. Obviously, each tool is monitoring a different set of configuration parameters but when running in a shared partition, they all monitor the entitlement. In Example 4-7 on page 68 while **vmstat** is running on a one logical processor partition, a configuration change occurred. The warning message is displayed and then the new configuration shows that a processor has been added.

Example 4-7 **vmstat** pre-header

| | | | | | | | | | | | | | | | | | |
|--|---|--------|--------|----|----|-------|----|----|----|--------|-----|----|----|-------|----|----|--|
| # vmstat 2 | | | | | | | | | | | | | | | | | |
| System configuration: 1cpu=1 mem=1024MB | | | | | | | | | | | | | | | | | |
| <hr/> | | | | | | | | | | | | | | | | | |
| kthr | | memory | | | | page | | | | faults | | | | cpu | | | |
| ----- | | ----- | | | | ----- | | | | ----- | | | | ----- | | | |
| r | b | avm | fre | re | pi | po | fr | sr | cy | in | sy | cs | us | sy | id | wa | |
| 0 | 0 | 59481 | 193172 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 67 | 95 | 0 | 0 | 99 | 0 | |
| 0 | 0 | 59481 | 193172 | 0 | 0 | 0 | 0 | 0 | 0 | 4 | 12 | 94 | 0 | 0 | 99 | 0 | |
| 1 | 0 | 59481 | 193172 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 17 | 97 | 0 | 0 | 99 | 0 | |
| System configuration changed. The current iteration values may be inaccurate. | | | | | | | | | | | | | | | | | |
| 8 | 0 | 59741 | 192881 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 443 | 83 | 0 | 30 | 69 | 0 | |
| System configuration: 1cpu=2 mem=1024MB | | | | | | | | | | | | | | | | | |
| <hr/> | | | | | | | | | | | | | | | | | |
| kthr | | memory | | | | page | | | | faults | | | | cpu | | | |
| ----- | | ----- | | | | ----- | | | | ----- | | | | ----- | | | |
| r | b | avm | fre | re | pi | po | fr | sr | cy | in | sy | cs | us | sy | id | wa | |
| 0 | 0 | 59773 | 192849 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 16 | 87 | 0 | 0 | 99 | 0 | |
| 0 | 0 | 59773 | 192849 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 90 | 0 | 0 | 99 | 0 | |

The trace base tools **filemon**, **netpmn**, **curt** and **splat** commands have been updated to give accurate information about the processor usage.

Another tool that needed modification was the **trace/trcrpt**. On a SMT environment, **trace** can optionally collect PURR register values at each trace hook and **trcrpt** can display elapsed PURR. **Trace** has also new trace hook marks phantom interrupts and new preemption hooks mark undispatched time to support the Micro-Partitioning environment. All trace based tools will adjust processor times using preemption hook. In addition, most hcalls are traceable. This means they will appear in **trcrpt** output.

Reporting tools **curt** and **splat** can optionally use the PURR values to calculate processor times on a SMT environment. For **splat** the -p option specifies the use of the PURR register. **curt** shows physical affinity and phantom interrupt statistics when in a Micro-Partitioning environment. It also shows the Hypervisor calls *hcall* summary reports similar to system calls reports, the number of preemptions, and the number of **H_CEDE** and **H_CONFER** Hypervisor calls for each individual virtual processor as shown in Example 4-8 on page 69.

Example 4-8 *curt* output - preemptions, H_CEDE and H_CONFER

| Processor Summary processor number 0 | | | |
|--------------------------------------|---------------------------------------|--------------------------------------|----------------------------------|
| ----- | | | |
| processing total time (msec) | percent total time (incl. idle) | percent busy time (excl. idle) | processing category |
| ===== | ===== | ===== | ===== |
| 0.02 | 1.59 | 1.64 | APPLICATION |
| 0.08 | 5.26 | 5.43 | SYSCALL |
| 7.03 | 471.38 | 486.24 | HCALL |
| 0.19 | 12.43 | 12.82 | KPROC (excluding IDLE and NFS) |
| 0.00 | 0.00 | 0.00 | NFS |
| 1.10 | 73.50 | 75.81 | FLIH |
| 0.04 | 2.77 | 2.86 | SLIH |
| 0.02 | 1.39 | 1.44 | DISPATCH (all procs. incl. IDLE) |
| 0.01 | 0.44 | 0.45 | IDLE DISPATCH (only IDLE proc.) |
| ----- | ----- | ----- | |
| 1.45 | 96.94 | 100.00 | CPU(s) busy time |
| 0.05 | 3.06 | | IDLE |
| ----- | ----- | | |
| 1.49 | | | TOTAL |

Avg. Thread Affinity = 1.00

Total number of process dispatches = 5
Total number of idle dispatches = 5

Total Physical CPU time (msec) = 8.64

Physical CPU percentage = 0.60
Physical processor affinity = 0.997590
Dispatch Histogram for processor (PHYSICAL CPUid : times_dispatched).
PHYSICAL CPU 0 : 415

Total number of preemptions = 415
Total number of H_CEDE = 415 with preemption = 414
Total number of H_CONFER = 0 with preemption = 0

Processor Summary processor number 1

| ----- | | | |
|------------|--------------|--------------|----------------------------------|
| processing | percent | percent | processing category |
| total time | total time | busy time | |
| (msec) | (incl. idle) | (excl. idle) | |
| ===== | ===== | ===== | ===== |
| 243.43 | 98.65 | 98.65 | APPLICATION |
| 2.26 | 0.91 | 0.91 | SYSCALL |
| 0.03 | 0.01 | 0.01 | HCALL |
| 0.00 | 0.00 | 0.00 | KPROC (excluding IDLE and NFS) |
| 0.00 | 0.00 | 0.00 | NFS |
| 1.07 | 0.43 | 0.43 | FLIH |
| 0.00 | 0.00 | 0.00 | SLIH |
| 0.01 | 0.00 | 0.00 | DISPATCH (all procs. incl. IDLE) |
| 0.01 | 0.00 | 0.00 | IDLE DISPATCH (only IDLE proc.) |
| ----- | ----- | ----- | |
| 246.77 | 100.00 | 100.00 | CPU(s) busy time |
| 0.00 | 0.00 | | IDLE |
| ----- | ----- | | |
| 246.77 | | | TOTAL |

Avg. Thread Affinity = 1.00

Total number of process dispatches = 5
Total number of idle dispatches = 2

Total Physical CPU time (msec) = 246.80
Physical CPU percentage = 19.74
Physical processor affinity = 0.997126
Dispatch Histogram for processor (PHYSICAL CPUid : times_dispatched).
PHYSICAL CPU 1 : 348

Total number of preemptions = 348
Total number of H_CEDE = 2 with preemption = 2
Total number of H_CONFER = 0 with preemption = 0

MAPI

With the new POWER5 processors, it was necessary to update the PMAPI.

There is a new API for POWER5 processor called **pm_initialize**, which you must use instead of the old **pm_init** API. With the updated PMAPI, there is a new way to return event status and characteristic. You can get it by bit array instead of char and there is a new “shared” characteristic, for processors supporting SMT. A shared event, is controlled by a signal not specific to a particular thread's activity and sent simultaneously to both sets (one for each thread) of hardware counters. There should be an average of counts across sibling threads. The added processor features bit array in the **pm_initialize** has two bits currently defined, the Hypervisor mode and runlatch mode. Moreover, **pm_initialize** can also retrieve event table for another processor instead of the old way in which we could only retrieve the tables for the current processor.

The new PMAPI now supports M:N threading model as opposed to the previous 1:1 model. This new model allows to map M user threads to N kernel threads and M is much bigger than N. There is a new set of APIs for third party calls (debugger) generically called **pm_*_thread** which differs from the old **pm_*_thread** interfaces in an additional argument to specify ptid. In 1:1 mode, there is no need to specify the ptid, but if you specify it, the library will verify that the specified pthread runs on the specified kernel thread. On the other hand, to use the M:N mode the ptid must always be specified. if ptid is not specified then there is the assumption that the pthread is currently undispatched. Regarding all other APIs, they are unchanged but now work in M:N mode.

With this new API come some new commands, **pm1ist** and **pmcycles** for example. The **pm1ist** is a utility to dump and search processors event and group tables. It currently supports text and spreadsheet output formats. The **pmcycles** command uses the Performance Monitor cycle counter and the processor real-time clock to measure the actual processor clock speed in MHz as shown in Example 4-9 on page 71.

Example 4-9 pmcycles output

```
# pmcycles -m
Cpu 0 runs at 1656 MHz
Cpu 1 runs at 1656 MHz
Cpu 2 runs at 1656 MHz
Cpu 3 runs at 1656 MHz
Cpu 4 runs at 1656 MHz
Cpu 5 runs at 1656 MHz
```

GPROF

The new environment variable GPROF controls the **gprof**'s new mode that supports multi-threaded applications.

```
GPROF=[profile:{process|thread}][,][scale:<scaling_factor>][,][file:{one|multi|
multithread}]
```

Where:

| | |
|-----------------------|--|
| profile | Indicates whether it will do a thread or process level profiling. |
| scaling_factor | Represents the granularity of the profiling data collected. |
| file | Indicates whether it will generate a single or multiple gmon.out file(s). |
| multi | Creates a file for each process (for each fork or exec) gmon.out.<progname>.<pid>. |
| multithread | Creates a file for each pthread gmon.out.<progname>.<pid>.Pthread<ptid> which can be used to look at one pthread at a time with gprof or xprofiler . |

The default values for **gprof** are process for the profile option, a scaling factor of 2 for process level and 8 for thread level (the thread level profiling consumes considerably more memory) and one file for the output. Several flags allow to optionally separate output into multiple files:

| | |
|--------------------|---|
| -g filename | Writes the call graph information to the specified output filename. It suppresses the profile information unless -p is used. |
| -p filename | Writes flat profile information to the specified output filename. It suppresses the call graph information unless -g is used. |
| -i filename | Writes the routine index table to the specified output filename. If this flag is not used, the index table goes either at the end of the standard output, or at the bottom of the filename(s) specified with -p and -g. |

The format of data itself is unchanged but now it can be presented in multiple sets in which the first set has cumulative data and the following sets have the data per thread.

Graphical tools

As with text based tools, the processor accounting for graphical tools has to be changed to use the new metrics regarding the shared mode environment and SMT. Graphics tools like PTX 3dmon, PTX xmperf and PTX jtopas have also been updated.

PTX 3dmon and xmperf

The most complete graphical tool, PTX, now for Micro-Partitioning uses PURR-based utilization metrics and entitlement utilization. An example of **3dmon**

display is shown in Figure 4-2 on page 73 and **xmperf** Mini Monitor is shown in Figure 4-3 on page 74.

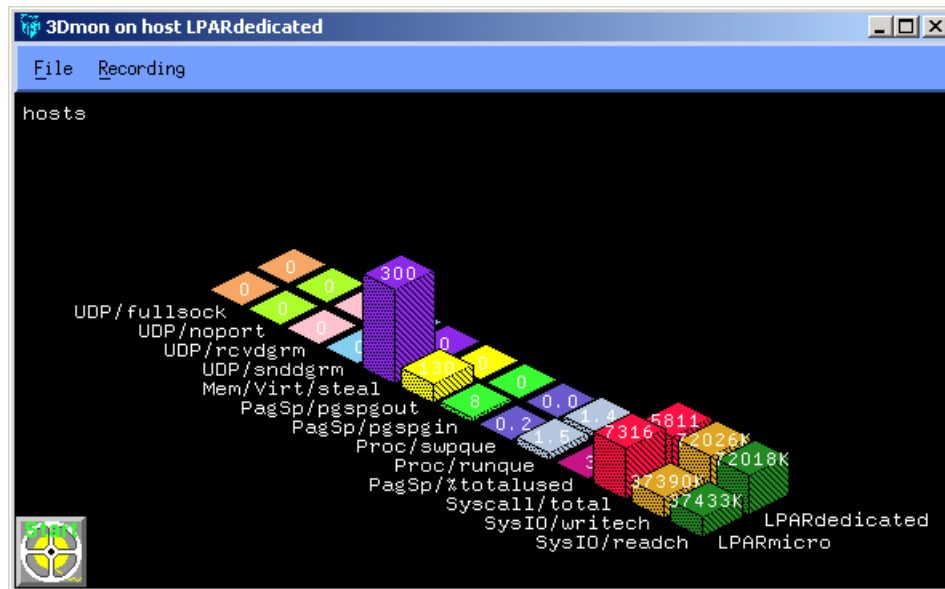


Figure 4-2 3dmon monitoring two LPARs

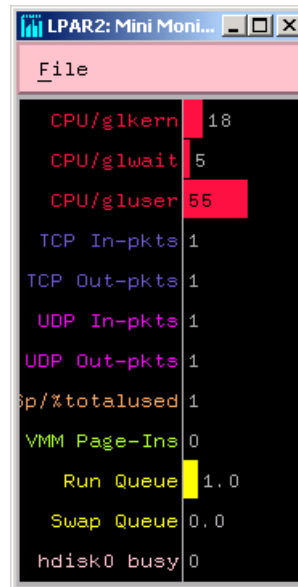


Figure 4-3 xmp perf Mini Monitor

PTX jtopas

Shipping with PTX since May 2003, the graphical tool **jtopas** is a hot resource monitoring tool, sibling of **topas**. **jtopas** starts with a pre-defined (no setup needed) Swing GUI. In the main screen, it shows a set of system metrics and hot resource summaries similar to **topas**, with access to more detailed information for each area. This is a generalization of the P, W and L commands of **topas**, which provide process, partition and WLM detail reports. **jtopas** works locally or remotely and it is able to generate dynamic reports with up to 7 days playback. It keeps data automatically for a week in 7 rotating daily files enabling **jtopas** to generate reports by hour or by day. You can save these reports in HTML format or in spreadsheet format. You can have a week by days report and a day by hours report. **jtopas** is a Swing GUI enabled application which means you can minimize or move each window and all resources are always available using scroll bar. **jtopas** uses **xmtrend** daemon.

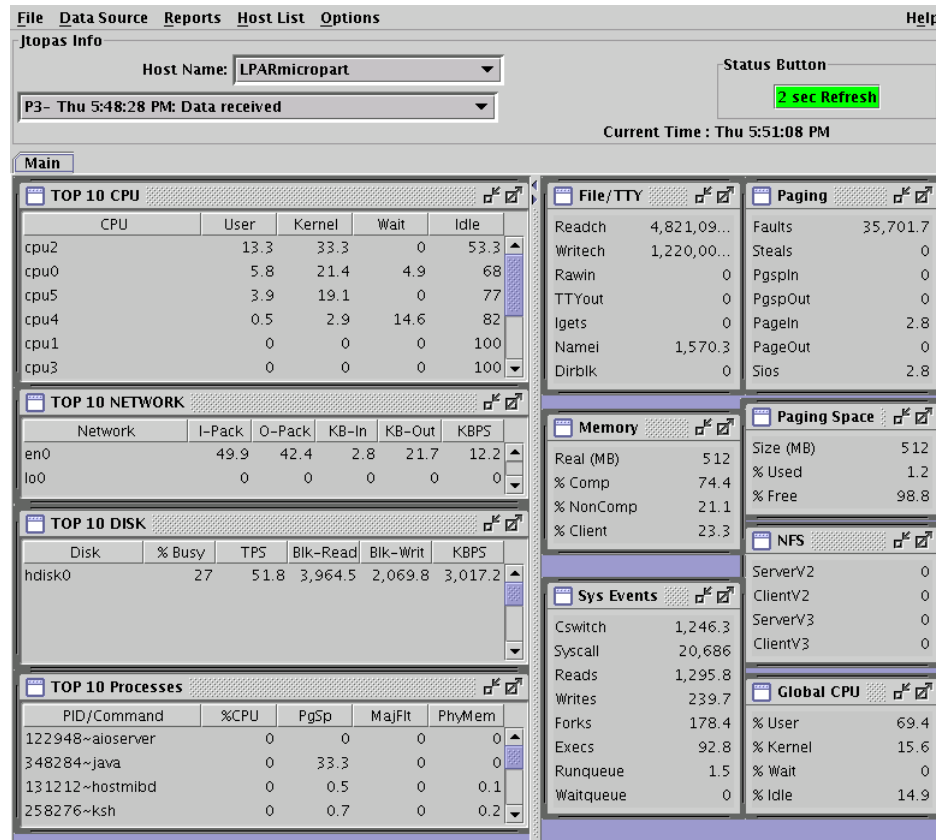


Figure 4-4 jtopas default display

For more details on performance commands refer to section 8.1, “Performance commands” on page 274.

4.1.5 New performance commands

Some new tools have been added for performance and tuning in SMT or micro-partitioning environment.

The following list give the main features of **lparstat**, **mpstat** and **perfwb** commands:

► **lparstat**

- Information and statistics about the partition.
- Details on the configuration of the partition,

- Summary of Hypervisor statistics and detailed Hypervisor information.
- The **lparstat** command output changes depending on the partition mode, as shown in Example 4-10 on page 76 and Example 4-11 on page 76.

In Micro-Partitioning, the real resource consumed in for the user is the percentage of CPU user (%user) times percentage of entitlement consumed (entc%) times the entitlement (ent). In this case 18.2% of 42,4% of 0.30 gives 2.3% of CPU consumed by the user.

The physical processor consumed (physc) is equal to the percentage of entitlement (%entc) times the entitlement (ent).

Example 4-10 **lparstat** output in a dedicated partition

lparstat

System configuration: type=Dedicated mode=Capped smt=On lcpu=2 mem=1024

| %user | %sys | %wait | %idle |
|-------|-------|-------|-------|
| ----- | ----- | ----- | ----- |
| 67.6 | 31.8 | 0.0 | 0.6 |

Example 4-11 **lparstat** output in Micro-Partitioning

lparstat

System configuration: type=Shared mode=Uncapped smt=On lcpu=6 mem=512 **ent**=0.30

| %user | %sys | %wait | %idle | physc | %entc | lbusy | app | vcs | phnt |
|-------|-------|-------|-------|-------|-------|-------|-----|------|-------|
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | --- | --- | ----- |
| 18.2 | 12.5 | 0.9 | 68.4 | 0.13 | 42.4 | 4.3 | - | 2747 | 3 |

► **mpstat**

- Basic utilization metrics.
- Logical and physical processor metrics (in SMT mode).
- Interrupt metrics.
- Logical processor affinity metrics.
- The **mpstat** command output changes depending on the partition mode.

► **perfbw**

- Dynamic process monitoring.
- Partition wide metrics about processor and memory activity as shown in Figure 4-5 on page 77.
- Sorted list of processes as shown in Figure 4-6 on page 78.

- Columns can be added or removed, sorted in ascending or descending order. Actions can be done on listed processes like kill, renice, run performance commands or get some information.
- Part of bos.perf.tools fileset, start “Performance Workbench” with **perfwb** command to launch Procmon tool.

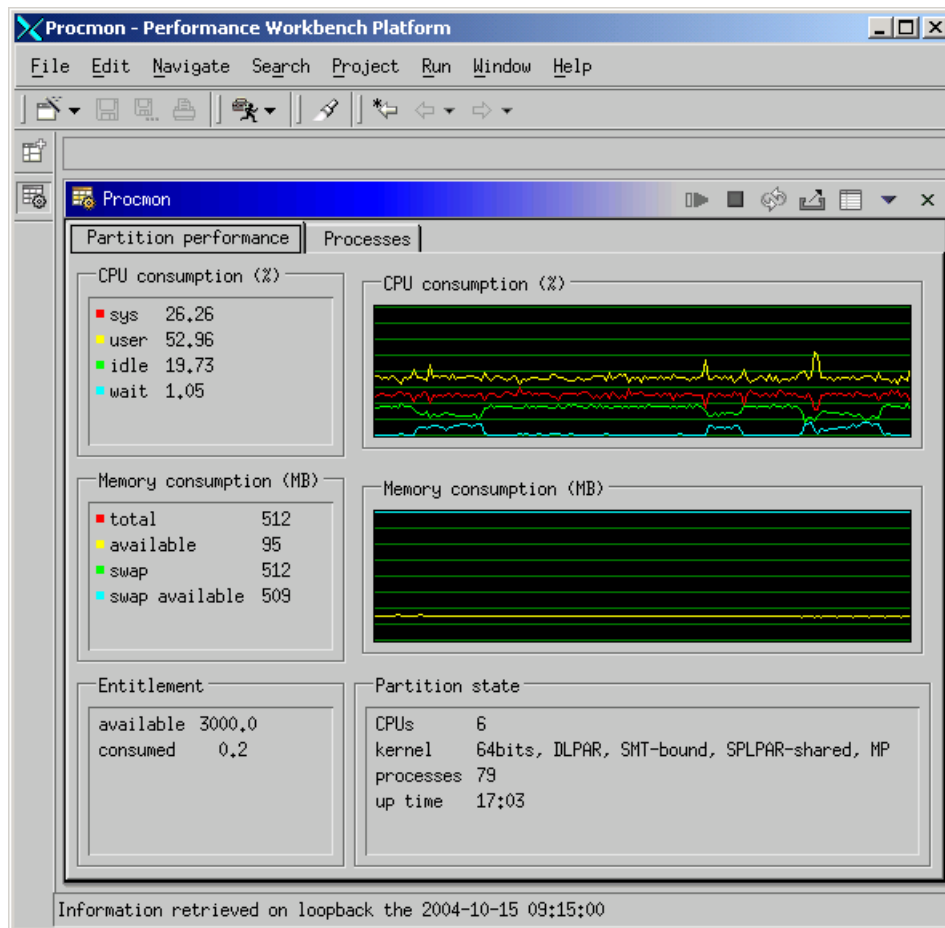


Figure 4-5 *procmon partition wide metrics*

| PID | % CPU | %... | User time | Command | Eff. login |
|--------|-------|------|-----------|-----------------|------------|
| 262384 | 0 | 9 | 0:24,33 | java | root |
| 274568 | 0 | 2 | 0:00,10 | IBM.ERmd | root |
| 372794 | 0 | 1 | 0:00,00 | lmigratepp | root |
| 241804 | 0 | 1 | 0:35,58 | IBM.CSMAGENTRMD | root |
| 20490 | 0 | 0 | 0:00,00 | wait | root |
| 24588 | 0 | 0 | 0:00,00 | wait | root |
| 28686 | 0 | 0 | 0:00,00 | wait | root |
| 32784 | 0 | 0 | 0:00,00 | wait | root |
| 36882 | 0 | 0 | 0:00,00 | wait | root |
| 40980 | 0 | 0 | 0:00,00 | wait | root |
| 45078 | 0 | 0 | 0:00,00 | wait | root |
| 49176 | 0 | 0 | 0:00,00 | reaper | root |
| 53274 | 0 | 0 | 0:00,00 | lrud | root |
| 57372 | 0 | 0 | 0:00,00 | vmptacrt | root |
| 61470 | 0 | 0 | 0:00,00 | xmfreed | root |
| 65568 | 0 | 0 | 0:00,00 | memgrdd | root |
| 69666 | 0 | 0 | 0:00,00 | psgc | root |
| 77760 | 0 | 0 | 0:00,00 | ... | ... |
| // | 0 | 13 | // | // | // |

Information retrieved on loopback the 2004-10-15 09:19:14

Figure 4-6 *procmon sorted list of processes*

For more details on performance commands refer to section 8.1, “Performance commands” on page 274.

4.1.6 Paging space

AIX 5L v5.3 introduces enhanced paging space management algorithms to collect paging space as needed. They apply only to deferred page space allocation policy. This management may be useful when paging space is almost full due to dynamic memory removal for example. In that case if the memory is added back to the partition, the paging space will not be freed, this adds some constraints on paging space although there is free real memory in the partition.

Garbage collect paging space on re-pagein

Applies only to deferred page space allocation policy. New mechanism to free or not a disk block after a pagein operation depending on the free space remaining in the paging space.

Garbage collect paging space scrubbing for in-memory frames

This mechanism tries to reclaim paging space disk blocs for pages that are already in memory, if the free space available in the paging space decreases under a tunable limit.

Tuning parameters for paging space garbage collection

Tuning on paging space parameters are done with **vmo** command. Here is a list of new ones:

| | |
|--------------------|--|
| npsrpgmin | Low paging space threshold for re-pagein garbage collector to start. |
| npsrpgmax | High paging space threshold for re-pagein garbage collector to stop. |
| rpgclean | Configures re-pagein garbage collector to be active when a page is read or when a page is read or write. |
| rpgcontrol | Enables or disables re-pagein garbage collector. |
| npssrubmin | Low paging space threshold for garbage collector scrubbing to start. |
| npsscrubnax | High paging space threshold for garbage collector scrubbing to stop. |
| scrubclean | Configures garbage collector scrubbing to be active when a page is read or when a page is read or write. |
| scrub | Enables or disables garbage collector scrubbing. |

4.1.7 Logical Volume Manager (LVM)

There have been several improvements to the AIX 5L v5.3 Logical Volume Manager that concern performance.

Scalable Volume Group (SVG)

The new SVG supports up to 1024 disks. This expands the capacity of the volume groups but needs a substantially larger volume group descriptor area (VGDA) and Volume Group Status Area (VGSA). Increasing maximum logical volumes or maximum physical partition per volume group from the defaults towards the limits increases the amount of metadata (VGDA / VGSA) that must be read or written during LVM operations. Every VGDA update operation

(creating a logical volume, changing a logical volume, adding a physical volume, and so on) might be longer to run as LVM keeps a copy of metadata on each physical volume. In the previous releases the maximum number of PPs was defined per disk, it is now defined per volume group. The limits for each type of volume group are listed in Table 4-1 on page 80.

Table 4-1 Maximum values for volume groups

| VG type | PVs | LVs | PPs | PP size |
|-------------|------|------|----------------|---------|
| Normal VG | 32 | 256 | 1016 per disk | 1 GB |
| Big VG | 128 | 512 | 1016 per disk | 1 GB |
| Scalable VG | 1024 | 4096 | 2097152 per VG | 128 GB |

Variable Logical Track Group (LTG)

The LVM device driver breaks I/O down into LTG size chunks before passing the I/O down to the device driver of the underlying disks. The LTG size is an attribute of the volume group. In the previous release, LTG size was defined at volume group creation or update, now it is determined at vary on time and will be dynamically updated if a physical volume is added or removed in the volume group. AIX 5L v5.3 allows the stripe size of an logical volume to be larger than the LTG size of the volume group, which was not allowed in previous versions. In addition, AIX 5L v5.3 now supports larger LTG sizes and stripe sizes. Valid LTG and stripe sizes are listed in Table 4-2 on page 80.

Table 4-2 LTG and stripe sizes

| AIX release | Valid LTG sizes | Valid stripe sizes |
|--------------------------|--|--|
| AIX 5L v5.2 and previous | 128 KB, 256 KB, 512 KB, 1 MB | 4 KB, 8 KB, 16 KB, 32 KB, 64 KB, 128 KB, 256 KB, 512 KB, 1 MB |
| AIX 5L v5.3 | adds support for 2 MB, 4 MB, 8 MB, 16 MB | adds support for 2 MB, 4 MB, 8 MB, 16 MB, 32 MB, 64 MB, 128 MB |

Performance improvements

LVM metadata (VGDA / VGSA) need to be stored in every single disk in a volume group, in order to improve performance AIX 5L v5.3 writes all metadata in parallel. There is one thread for each disk in the volume group. In addition, some commands that would read data, utilize a small piece, then read again, utilize a small piece, now read the metadata once and keep the metadata accessible throughout the life of the command. Special focus was made on the following

commands: **extendvg**, **importvg**, **mkvg**, **varyonvg**, **chlvcopy**, **mklvcopy**, **lslv** and **lspv**.

Striped column

Prior to AIX 5L v5.3 there was no good way to extend a striped logical volume if one of the disks was full. The work around was to backup the data, delete the striped logical volume, remake the logical volume with a larger stripe width, and then restore the data. Now, we can extend a striped logical volume even if one of the disks is full. We do this by modifying the maximum number of physical volumes for the new allocation, the upper bound. Prior to AIX 5L v5.3, the stripe width and upper bound were required to be equal. In AIX 5L v5.3, the upper bound can be a multiple of stripe width, where you can think of each stripe as a "column". You can use **extendlv** command to extend a striped logical volume into the next column. If you use **extendlv -u** then you can raise the upper bound and extend the logical volume all in one operation (like an **extendlv** and a **chlv -u** all in one).

Volume group pbuf pools

The LVM uses a structure called pbuf to handle disk I/O. In previous versions pbuf pool was a system wide resource, now each volume group gets its own pbuf pool. To manage pbuf we use the **lvmo** command which displays and tunes several volume group specific items:

| | |
|-------------------------------|---|
| pv_pbuf_count | Number of pbufs added when a physical volume is added to the volume group. It is tunable with lvmo command, it takes effect immediately. |
| total_vg_pbufs | Number of pbufs currently available for the volume group. It is tunable with lvmo command, takes effect at varyonvg time. |
| max_vg_pbuf_count | Maximum number of pbufs for this volume group. It is tunable with lvmo command, takes effect at varyonvg time. |
| pervg_blocked_io_count | Number of I/Os that were blocked due to lack of free pbufs for this volume group. Can only be displayed, not tunable. |

The **lvmo** command also displays the following system wide items:

| | |
|--------------------------|--|
| global_pbuf_count | Minimum number of pbufs that are added when a physical volume is added to any volume group. It is tunable with ioo command. It takes effect at varyonvg time. |
|--------------------------|--|

global_blocked_io_count

System wide number I/Os that were blocked due to lack of free pbufs.

For further information about LVM, refer to Section 3.1 in *AIX 5L Differences Guide Version 5.3 Edition*, SG24-7463

4.1.8 Virtual Local Area Network (VLAN)

Virtual Local Area Network (VLAN) is a method to logically segment a physical network, that means only adapters belonging to a same VLAN can communicate together. AIX 5L v5.3 supports *virtual ethernet* technology which allows communications between logical partitions on the same system using a VLAN.

Shared Ethernet Adapter (SEA) technology enables the logical partitions to communicate with machines that are outside the system without any *physical ethernet* slots assign to the logical partition. SEA creates a relation between virtual ethernet adapters and a real network adapter. The Shared Ethernet Adapter is part of the optional virtual I/O server.

In a dedicated or Micro-Partition we can configure physical and virtual adapters at the same time. Example 4-12 on page 82 shows two types of adapters, the physical *ent0* and the virtual *ent1*. Both have network address. The *Device Specific.(YL)* field contains in one case (ent0) a real physical location code and in the other case (ent1) a logical location code given by the virtual I/O server.

Example 4-12 Ethernet adapters

```
# lsdev -Ccadapter
ent0   Available 02-08 10/100 Mbps Ethernet PCI Adapter II (1410ff01)
ent1   Available      Virtual I/O Ethernet Adapter (1-lan)

# lspcfg -vl ent0
ent0 U787A.001.DNZ00XY-P1-C2-T1 10/100 Mbps Ethernet PCI Adapter II (1410ff01)

10/100 Mbps Ethernet PCI Adapter II:
Part Number.....09P5023
FRU Number.....09P5023
EC Level.....H10971A
Manufacture ID.....YL1021
Network Address.....000D600A58A4
ROM Level.(alterable).....SCU015
Product Specific.(Z0).....A5204209
Device Specific.(YL).....U787A.001.DNZ00XY-P1-C2-T1

# lspcfg -vl ent1
ent1 U9111.520.10DDEDC-V2-C10-T1 Virtual I/O Ethernet Adapter (1-lan)
```

```
Network Address.....C6BB3000200A
Displayable Message.....Virtual I/O Ethernet Adapter (1-lan)
Device Specific.(YL).....U9111.520.10DDEDC-V2-C10-T1
```

For more detail on VLAN refer to section 7.3, “Virtual Ethernet” on page 181.

4.1.9 EtherChannel

The EtherChannel technology is based on port aggregation, that means ethernet adapters are aggregated together and belongs to the same network. They share the same IP address and the same hardware address. The bandwidth of the EtherChannel adapter is increased due to the aggregation of physical ethernet adapters.

Prior to AIX 5L v 5.3, addition or removal operations of a physical adapter member of an EtherChannel was only possible if the interface was detached or not configured. The interface has also to be detached in order to modify EtherChannel attributes.

With AIX 5L v 5.3, the Dynamic Adapter Membership (DAM) allows addition, removal or update operations at runtime. A failed ethernet adapter can be replaced without IP disruption.

A fail over can be manually forced on condition that the EtherChannel has a working backup adapter, this is useful for recovering from a fail over due to a failure. The recovery time to primary has been improved.

4.1.10 Partition Load Manager (PLM)

Partition Load Manager (PLM) for AIX 5L is a load manager that balances resources (processor and memory) between partitions executing within the same physical server.

To benefit from PLM, the managed partitions must be running AIX 5L v5.2 or AIX 5L v5.3. PLM works with both dedicated partitions and micro-partitions.

PLM allocates resources to partitions according to rules defined by the system administrators. In PLM terminology, these rules are called policies. The policies define how PLM assigns unused resources or resources from partitions with low usage to partitions with a higher demand, improving the overall resource utilization of the system.

PLM is implemented using a client-server model. The server part of PLM is packaged as part of the Advanced Power Virtualization feature of pSeries

servers. There is no special code to install on a client partitions that will be managed by PLM.

The PLM client-server model is event based, not polling based. The PLM server receives events each time one of its managed partitions needs extra resources.

When the PLM server starts, it registers several events on each managed partition. In order for PLM to get system information and dynamically reconfigure resources, it requires an SSH network connection from the PLM server to the HMC. The Resource Management and Control (RMC) services are responsible to gather all the status information on the managed nodes. The RMC daemon exports system status attributes and processes the re configuration requests from HMC. With this data and in conjunction with the user-defined resource management policy, the PLM server decides what to do each time a partition exceeds one of the threshold defined in the PLM policies.

PLM is presented in more details in Chapter 9, "Partition Load Manager (PLM)" on page 329.

4.2 Linux on POWER

In this chapter we introduce/provide/describe/discuss ...

In this chapter:

In this chapter, the following topics are discussed/described:

This chapter provides/describes/discusses/contains the following:

- ▶ ...
- ▶ ...
- ▶ ...
- ▶ Sample level 2 “n.n” chapter heading
(created by **Special > Cross-Reference > Format: Head > Insert**)
- ▶ Sample next level 2 heading

4.3 Sample level 2 “n.n” chapter heading (Head 1) new page

Note to Author: The first level 2 “n.n” heading in a chapter should be the (Head 1) tag, skip to new page.

Add text here (Body0).

4.4 Sample level 2 “n.n” chapter heading (Head 2)

Add text here (Body0).

4.4.1 Sample level 3 “n.n.n” chapter heading (Head 3)

Add text here (Body0).

Sample level 4 heading (Head 4)

Add text here (Body0).

Sample level 5 heading (Head 5)

Add text here (Body0)



Part 3

Features and capabilities

5



Simultaneous Multi-Threading (SMT)

This chapter provides detailed information about Simultaneous Multi-Threading (SMT) as part of one of the new features in POWER5 and AIX 5.3

The following major topics are covered here:

- ▶ Understanding SMT and its implementation on POWER5
- ▶ Resources for SMT
- ▶ Software considerations for SMT
- ▶ Performance considerations when running with SMT
- ▶ Case Studies; performance of selected scientific applications running on single-threaded (ST) and SMT

5.1 What is SMT?

In general, Multi-Threading evolution can be broadly divided into:

- ▶ Single threading
- ▶ Coarse grain threading
- ▶ Fine grain threading
- ▶ Simultaneous Multi-Threading

The obvious case corresponds to a single thread. This is summarized in Figure 5-1 on page 92 along with the other three cases. In single-threaded mode we see a thread issuing two instructions per cycle. Note that in the single-threaded mode, just two slots are utilized in the first cycle (vertical column) and execution slot utilization is dependent on instruction level parallelism exhibited by the workload.

In coarse grain Threading (see Evaluation of Multithreading Uniprocessors for Commercial Application Environments, 23rd Annual International Symposium on Computer Architecture for a more extensive discussion), one thread known as the active thread; active and dormant threads are explained in this chapter, executes on the processor at one time while the other thread is dormant. If the active thread experiences a long latency event such as a cache miss, the processor puts the active thread into the dormant state and switches over to the dormant thread. This provides a mechanism to overlap cache misses between the two threads to improve throughput. For such threading mechanism to work efficiently, the thread switch time (time taken to switch executing from the active thread to the dormant thread) should be shorter than the latency of the event that caused the switch. But, switching the thread efficiently becomes difficult as the processor pipeline depth increases.

In fine grain Threading, processor issues multiple instructions per cycle from one thread, alternating between threads every cycle. While fine grain threaded processors tolerate long latency operations better and utilize execution units better, not all issue slots of the execution units are always utilized. Thus, efficiency of fine grained Threaded processors are also limited by the instruction level parallelism of the executing thread.

In a Simultaneous Multi-Threaded processor, the processor can issue multiple instructions per cycle from any of the hardware contexts on the processor. Since instructions from any of the threads can be issued by the processor in a given cycle, the processor is no longer limited by the instruction level parallelism of the individual threads. Thus, SMT provides a threading model which can exploit instruction level parallelism as well as thread level parallelism (multiprogramming) in workloads. In other words, all threads are simultaneously

active and in comparison with Threading (coarse or fine grain) there is no thread switch event. For instance, when one of the threads has a cache miss, the second thread can continue to execute.

More general information about SMT may be found in the following references:

- ▶ Simultaneous Multithreading: Maximizing On-Chip Parallelism, Proceedings 22nd International Symposium on Computer Architecture
- ▶ Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor, 23rd Annual International Symposium on Computer Architecture.
- ▶ Initial Observations of the Simultaneous Multithreading Pentium 4 Processor.

In this book we use the following definition for SMT:

SMT The ability of a single physical processor to simultaneously dispatch instructions from more than one hardware thread context.

In other words, SMT is a hardware design enhancement in POWER5 that allows two separate instruction streams (threads) to execute simultaneously on the processor.

SMT is found to improve performance for a variety of workloads. Thus, SMT combines the advantages of wide-issue superscalar processors and latency tolerant features of Multi-Threaded processors for enhanced performance. With SMT, since multiple programs share the execution resources (in a multiprogramming environment), the overall throughput of the system will be improved although the individual programs may run slower than they would if they ran in a single-threaded mode. The performance benefit is in being able to execute more instructions from multiple programs in a given amount of time.

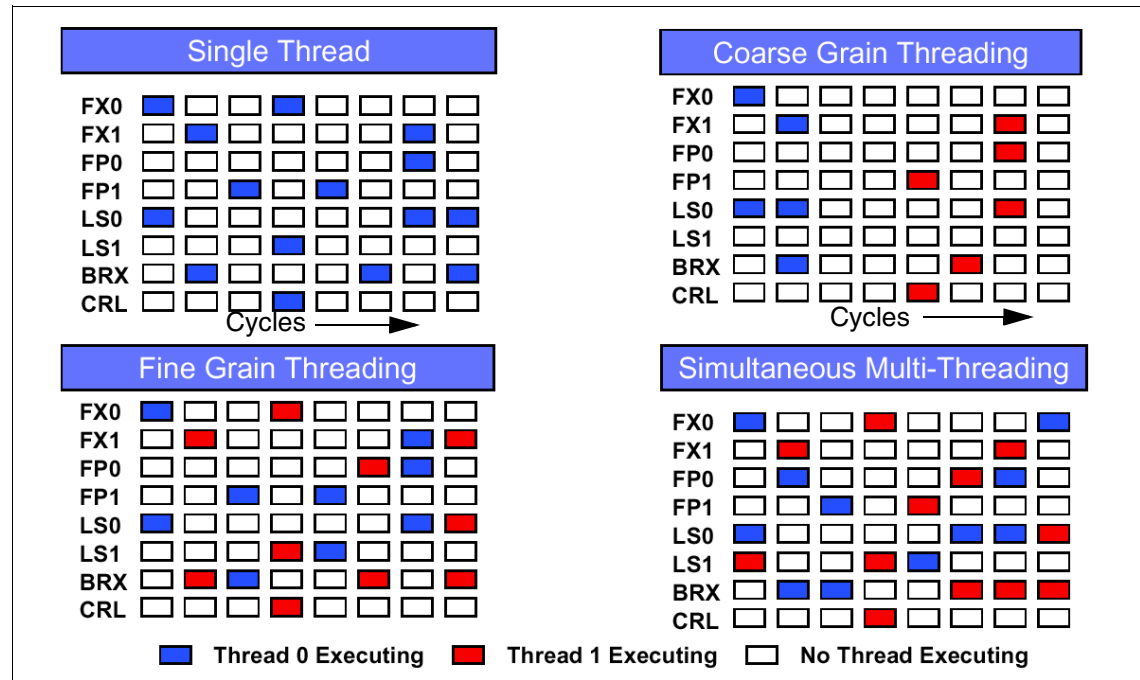


Figure 5-1 Different Multi-Threading models.

Figure 5-1 illustrates processor execution resource utilization under different environments every cycle. The first column on the left corresponds to the 8 execution units (1 branch conditional unit, 1 condition register unit, 2 fix-point execution units, 2 floating-point execution units for a total of 8 independent execution units). The rectangular blocks represent issue slot utilization. Each row represents utilization of the issue slot of the labelled execution unit every cycle. A column represents the utilization of issue slots on the same cycle. An empty box (white) represents an unused slot. A colored box means that the issue slot is being used.

As can be seen from the figure, utilization levels are not high in the single-threaded mode with low workloads. Utilization levels are not high on the coarse and fine grain threading modes either. Utilization levels are much better in SMT model.

5.2 POWER5 SMT implementation

The POWER5 SMT implementation is a natural extension to the eight instruction wide issue superscalar POWER4 design. POWER5 supports two way SMT by

providing two logical processors (hardware contexts) per processor core. *Each POWER5 processor core appears as a two processor Symmetric Multi-Processor (SMP) to the operating system.* Instructions from either thread can use the eight instruction wide issue slots in a given cycle. The POWER5 also features Dynamic resource balancing (DRB) and adjustable thread priorities for efficient resource utilization of the resources shared by both threads.

The POWER5 provides for the software to dynamically switch from SMT mode to single-threaded mode and vice versa. There are instances when this could be useful, like for real time applications where the program execution speed is more important than overall throughput, or scientific applications which are limited by execution resources (sharing of execution resources will then prove counterproductive), so running in single-threaded mode for such applications will prove useful. There might also be instances when there are not enough programs ready to run on all the hardware threads. In such cases, one thread of a processor core could be running the operating system's idle loop while the other is running a useful application process. Since, even an SMT thread executing the operating system's idle loop needs at least the architected registers from the rename resource pool, the performance difference of a task running in single-threaded mode compared to running in SMT mode with the other hardware thread running the operating system's idle loop could introduce a slight negative affect on performance due to the execution resources consumed by the idle thread.

5.2.1 Resources for SMT

All the SMT resources on the POWER5 have been tuned for optimum performance. The optimal number rename resources such as the number of physical general purpose registers (GPRs) to be put on the core, have been optimized with workloads and varying the number of GPRs for maximum instructions executed per cycle during the course of the chip design.

Although we have already introduced the idea of SMT in previous sections, here we provide a more detailed discussion. In particular, we look at DRB and adjustable thread priorities.

5.2.2 Dynamic resource balancing

The purpose of this feature is to ensure smooth flow of both threads through the processor. If either of the hardware threads start dominating the processor resources and depriving the other thread, the dynamic resource balancing logic throttles down the dominating thread so that the other thread can flow smoothly without stalling. For example, if one of the hardware threads experiences multiple L2 data cache misses, the dependent instructions can block in the issue queue slots preventing the other thread from dispatching instructions. To prevent such

stalls, the DRB logic monitors the miss queues, and if a particular thread reaches a threshold for L2 cache misses, it throttles that thread down so that the other thread can progress smoothly. Similarly, one thread could start using too many global completion tables (GCTs) entries, preventing the other thread from dispatching instructions. DRB logic then detects this condition and throttles down the thread dominating the GCT.

POWER5 DRB could throttle down a thread in three different ways. Choice of the throttling mechanism is made depending on the situation. The throttling mechanisms are:

1. Reducing the thread's priority
 - Used in situations where a thread has used more than a predetermined number of GCT entries.
2. Holding the thread from decoding instructions until resource congestion is cleared
 - When number of L2 misses incurred by a thread reach a threshold
3. Flushing all the dominating thread's instructions waiting for dispatch and holding the thread's decoding unit until congestion clears
 - Used if a long latency instruction such as memory ordering instructions (e.g., **sync**) causes dominating of the issue queues.

Studies have shown that higher performance is realized when resources are balanced across the threads using DRB.

Note: It is important to note that DRB is done at the hardware level, it is not controllable by programmers applications

5.2.3 Adjustable thread priorities

The DRB logic is built into the hardware to ensure balanced resource utilization by the threads. There are scenarios when software could know that a process running on a hardware thread might not be doing any useful work, such as spinning for a lock, executing the operating system's idle loop for example. Sometimes, software might also want to quickly execute a process, such as a process holding a critical spinlock. For better utilization of processor resources under such scenarios, the POWER5 features adjustable thread priorities, where in software it can be specified if a hardware thread running the process can have more or less execution resources.

Table 5-1 shows that POWER5 supports eight levels of thread priorities (0-7). The priority of the thread is stored in a *Thread Status Register* (TSR). A three bit field is used to indicate the thread priority). The POWER5 supports three processor states, *POWER Hypervisor Mode*, *Supervisor Mode* (AIX or Linux kernel code), and *User Mode* (application program)

Table 5-1 POWER5 thread priority levels

| TSR Thread priority value | Priority Level | Privilege level for software to set this priority ^a | Equivalent no-op instruction ^b |
|---------------------------|-----------------|--|---|
| 0 | Thread shut off | Hypervisor Mode ^c | - |
| 1 | Very low | Supervisor Mode | or 31,31,31 |
| 2 | low | User Mode | or 1,1,1 |
| 3 | Medium low | User Mode | or 6,6,6 |
| 4 | Normal | User Mode | or 2,2,2 |
| 5 | Medium high | Supervisor Mode | or 5,5,5 |
| 6 | high | Supervisor Mode | or 3,3,3, |
| 7 | Extra high | Hypervisor Mode | or 7,7,7 |

- a. Certain fields in a thread control register (TCR) affect the privilege level. This column assumes recommended setting and setups, which is usually the case with well behaved software.
- b. no-op executes an **or** instruction to change priorities.
- c. Hypervisor will be discussed later in subsequent sections. Hypervisor can be considered highest privilege level followed by supervisor (usually the O/S) and user applications.

Ratio of decode slots allocated to the threads depend on the two thread's priority as in $1/2^{**}(|x-y| + 1)$ of the decode slots is given to the lower priority thread, where x and y are thread priorities, and $x > 1$ and $y > 1$. So if thread 0 has a priority of 4 and thread 1 has a priority of 2, then thread 1 gets 1/8 of the total decoding slots and thread 0 gets 7/8 of the decoding slots. Table 5-2 on page 95 summarizes decode unit division among threads under different thread priority scenarios.

Table 5-2 Effect of thread priorities on execution resource sharing

| Thread 0 priority | Thread 1 priority | Decode slots status |
|-------------------|-------------------|--|
| 0 | 0 | Stopped |
| 0 | 1 | 1 /32 of decode slots given to thread 1 for power savings. Thread 0 is stopped |

| Thread 0 priority | Thread 1 priority | Decode slots status |
|-------------------|-------------------|---|
| 0 | >1 | All of the decode slots go to thread 1 |
| 1 | 1 | Both the threads are given 1/32 of the total decode slots for power savings |
| 1 | >1 | Thread 1 gets all the execution resources ad thread 0 gets the leftovers. |
| >1 | >1 | 1 of 2** (lx-yl+1) decode units for the lower priority thread and the rest to the other thread. |

Thread priorities via user-level applications (e.g., C code) is to use **#pragama** compiler directives. Normally, at the top of the C code (after all the **#includes**) function declarations can be specified, following this declarations the **pragma** directives can be defined. Example XYZ provides a template how do to this from a C code.

Example 5-1 C template to change thread priorities.

```
void    smt_verylow_priority(void);
void    hmt_low_priority(void);
void    hmt_medium_priority(void);
void    smt_mediumhigh_priority(void);

#pragma mc_func smt_low_priority { "7c210b78" } /* or r1, r1, r1 */
#pragma mc_func smt_normal_priority { "7c421378" } /* or r2, r2, r2 */
#pragma mc_func smt_verylow_priority { "7ffffb78" } /* or r31, r1,r31*/
#pragma mc_func mt_mediumhigh_priority { "7ca52b78" } /* or r5, r5,r5*/

/*
    These function calls illustrate how to change thread priority;
    appropriate functions should be invoked
*/

main(){

.
.
.
smt_low_priority();
.
.
.
smt_normal_priority();
}
```

Figure 5-2 on page 104 depicts the effects of thread priorities on instructions executed per cycle. The x-axis entries with commas represent actual thread priority pairs, e.g., 0,7 implies 0 has been stopped and thread 1 has a priority of 7. The numbers without commas represent the value of (x-y) where x is the priority of thread0 and y is the priority of thread1. So a value of 5 on the x-axis would mean (7,2) or (6,1) for x and y.

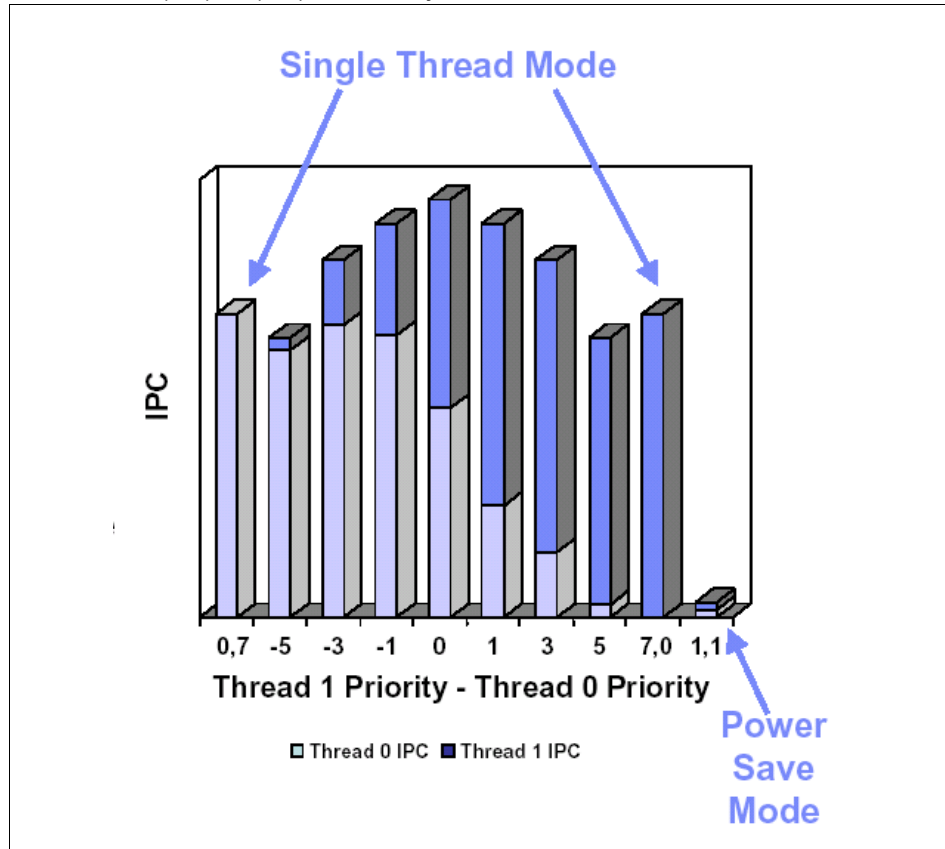


Figure 5-2 Thread priority pairs vs. instructions executed per second

The idea behind SMT is to increase overall throughput of the system by executing two threads which when run individually on the processor (in ST mode) may not utilize the processor execution resources to the desired level. SMT performance will depend of the type of application, however, for most cases some general rules may be introduced:

- ▶ SMT does not speed up individual threads of execution, but the work done collectively, or overall throughput should improve.
- ▶ If applications care about real time responses rather than overall system performance, they are better off running in ST mode.

- For workloads that are limited by the processor execution resources (such as technical workloads that exhibit high instruction level parallelism and consume large number of rename resources like floating-point registers (FPRs)), SMT will not help much.

Important: The POWER5 provides facilities for the operating system to dynamically switch from SMT to ST for such applications and workloads that might benefit from one mode but not the other.

5.3 Software considerations for SMT

In the hardware, a SMT hardware context (thread) can be in one of two states: live or dead. But software could maintain states of dead hardware contexts *if the hardware context can be turned on at some point of time. Hence, the software can have these three states. Live, dead and dormant.*

A thread is said to be live when it is *alive* as seen by the hardware and the software. The hardware still maintains the architected register states.

A thread is said to be dead when the thread is dead both in the hardware and software. The processor internal registers are set up so that the dormant thread can wake up on DECR or external interrupts. To revive a dead thread, the sibling thread is expected to execute the `mtctr1` special instruction (through the Hypervisor call).

A thread is said to be dormant when the thread is dead in hardware but alive in software. The architected register state is not maintained in the hardware, but the software maintains knowledge of the virtual processor, such as per-processor data. The 'software' here could be the operating system or the Hypervisor. The processor is setup so that the dormant thread can wake up on DECR or external interrupts.

5.3.1 Snooze and snooze delay

The process of putting a live thread into a dormant state is known as snoozing. If there are not enough tasks to run on the threads, threads could be running the operating system idle loop. It is better for the operating system to snooze the idle thread and switch over to single-threaded mode, so that the sibling thread gets all the processor resources and gets to run faster. To snooze a thread, the operating system will invoke the `H_CED` Hypervisor call (refer to Table 3-1 on page 45). The thread then goes to the dormant state. A snoozed thread is brought alive when a Decrementer, external interrupt or a `H_PROD` is received for the thread. If, when a hardware context is snoozed, the operating system finds more tasks on its run

queue and application throughput will improve if SMT is turned on, the processor must transition from single-threaded mode to SMT mode through any of the means mentioned earlier. This involves the snoozed thread beginning life at the system reset interrupt vector for the thread, and having the Hypervisor restore the operating system state, and then returning from the original `H_CED` Hypervisor call made by the thread to snooze. This means several thousand cycles of thread startup latency. Therefore, it does not make sense to snooze a thread as soon as the idle condition is detected. There could be another thread in the *ready-to-run* state in the **runqueue** by the time the snooze occurs, resulting in wasted cycles due to the thread start up latency. It is good for performance if the operating system waits for a small amount of time for work to come in before snoozing a thread. This short idle spinning time is known as SMT snooze delay. An operating system can optionally make this delay tunable.

Both AIX and Linux incorporate changes to snooze an idle thread. They also provide snooze delay tunables.

5.3.2 Process accounting

With single-threaded operation, a local timer tick (10 ms in AIX, 1ms in Linux with `HZ=1000`) was charged to whichever process was preempted by the timer interrupt. If the process was in kernel, the entire tick was charged to the process' system time. Else the process' user time was charged with 1 tick. But with SMT, the thread receiving the local timer interrupt most likely has not run for the entire tick duration as it shares the physical processor resources with its sibling. POWER5 provides a Processor Utilization Resource Register (PURR) to provide proper process/system accounting. System accounting on AIX using PURR has been discussed in "Processor Utilization Resource Register" on page 132.

PURR is a new per thread register introduced in POWER5. It is an incrementing 64 bit counter just like the TB. It gets incremented by one, every 8 processor cycles. The thread which dispatches a group in a cycle will increment its PURR by 1/8 in that cycle. If neither thread dispatches a group in a cycle, each thread increments its PURR by 1/16. The sum of the two PURRS of a processor over a period of time will be very close to the number of TB ticks over the same period of time, but never more than the TB ticks.

Process accounting in SMT mode should be done using the PURR registers. AIX uses PURR for process accounting. Instead of blindly charging the entire 10ms tick to the interrupted process, processes are charged based on PURR delta for the hardware thread since the last interval, which is an approximation of the computing resource that the thread actually received. This accounts for a more accurate accounting of processor time in the SMT environment, since the sum of the two PURRS is close to the TB ticks over the same period of time.

5.3.3 Processor utilization

There are different approaches to reporting processor utilization in an SMT environment.

One approach is to treat each hardware (SMT) thread (logical processor) as a separate processing engine. Under this approach, the processor utilization metric represents the proportion of time that work was dispatched on the logical processor. The processor utilization reported using this approach does not necessarily reflect the logical processing engine's utilization of the processor's physical execution resources since it does not account for factors such as the relative hardware priorities of the two SMT threads.¹

The second approach, used by AIX, is to report processor utilization from the perspective of the actual physical processor resource utilization. Under this approach, the processor utilization metric reflects the actual utilization of physical processor resources by the SMT threads. *The PURR registers are used to determine each hardware thread's processor utilization.*

The following example illustrates the difference between these two approaches. Consider a physical processor with two SMT threads. The OS sees the two SMT threads as two separate processing engines, and dispatches two separate tasks (processes), one on each logical engine. Under the first approach, each logical engine reports a utilization of 100%, representing the portion of time that the logical engine was busy. Under the second (AIX) approach, each logical engine reports a utilization of 50%, representing the proportion of physical processor resources that it used (assuming equal distribution of physical processor resources to both the hardware threads).

5.3.4 SMT aware scheduling

Although a multi processor kernel can run on a POWER5 SMT based system without any modifications, the kernel will just treat the logical processors as separate processors, if SMT awareness is not built into the kernel. For example, in a system with two physical processors (four SMT threads) and two runnable tasks, the scheduler could schedule tasks on two sibling threads of the same processor and keep the other core totally idle. Since the operating system is not SMT aware, there is no way the scheduler can distinguish between threads on the same processor or different processors. Obviously, this doesn't lead to efficient utilization of system processing capacity. Given this background the most obvious optimization for SMT is to make sure work is distributed to all the

¹ Each hardware thread of a processor represents a logical processing capacity, which is a portion of the physical processing capacity of the processor core. Each thread will have its own utilization of the logical processing capacity presented to it

primary² threads before work is dispatched to secondary threads. Secondary threads can be snoozed or put at very low priorities if they are idle.

Both AIX 5.3 and Linux 2.6 kernel have this optimization in place.

Another optimization is to consider the sibling threads of a core as one affinity (AIX) or scheduling (Linux) domain, so that the domain reflects sharing of resources such as the translation look-aside buffer (TLB), between the two sibling threads. It might be beneficial for software threads of the same process to run in the same domain so that the shared processor caches (L1, TLB) are effectively utilized by the software. It also makes sense to maintain the affinity of software tasks to domains where they ran earlier, so that they get a warmer cache.

Also, the bindprocessor command has been enhanced on AIX 5.3 to accept command line options to display all the primary threads or all the secondary threads. This is to help applications that use binding to bind on one physical processor.

These optimizations are present in both AIX 5.3 and Linux 2.6 kernel.

Above optimizations are meant to illustrate that SMT awareness will help the operating system perform better. There might be more such optimizations in the operating system which are not mentioned here.

5.3.5 Interrupts

The operating system has no impact on interrupt processing due to SMT. Each SMT thread has its own private decremter as well as its own interrupt server. Each hardware thread can asynchronously and simultaneously process its own interrupts just as if they were individual processors.

5.3.6 Effective use of adjustable thread priorities

POWER5 features adjustable thread priorities for better processor resource utilization. The feature has been explained in detail in “Adjustable thread priorities” on page 28. To summarize, POWER5 provides for eight levels of thread priorities (0-7). Please refer to Table 2-3 on page 28 for all the supported priorities. Ratio of decode slots allocated to a hardware thread is dependent on the thread priorities of the sibling threads. The table “Effect of thread priorities on execution resource sharing” on page 29 depicts the effect of thread priorities on execution resource sharing. The operating system can set priorities from one to

² For ease of explanation, it can be considered that there is one primary thread per core and one secondary thread per core in a two way SMT system -- although both the threads enjoy equal access to the execution resources with other factors like thread priorities being equal.

six, which correspond to 'very low' to 'high' priorities, respectively. Application programs (user space) can set thread priorities from two to four, which correspond to 'low' to 'normal' priorities, respectively.

By default, threads execute at 'normal' priority, both in kernel mode and user mode.

5.3.7 Thread priorities on AIX

AIX does not lower the priority of the idle thread if SMT is on. It searches for work in its own run queue and other run queues under normal priorities. This ensures that any information about available work is current and can be acted upon with least latency. If no work is available and if the snooze delay is not over, it will spin in a loop for a tunable number of times in very low priority loop (for power savings only, not really meant as a performance enhancement) checking for work only on its own run queue. After that it returns to top of the idle loop and repeats the search for work until snooze delay expires. For AIX instances in dedicated LPARs, AIX will lower priority to 'low' for the idle thread if SMT is on and wait for SMT snooze delay is set to 0 and so AIX always invokes **H_CED** hcall as soon as it enters the idle loop and only if it does not find any work on other run queues for idle stealing.

If a task in the kernel is waiting for a spinlock, AIX changes the thread priority of the hardware context executing that task to two or 'low' priority; so that the spinning thread, which is not doing any useful work yields processor resources to its sibling. For instances of AIX running in Micro-Partitioning environment, AIX waits for a spin delay after lowering priority to priority two and then invokes **H_CONFER** hcall in the Micro-Partitioning environment after spinning for a while only if the job (or software thread) is running disabled. If running enabled, AIX puts the job to sleep after spinning a number of times and goes into the dispatcher to dispatch another job. The Hypervisor will control priority management and redispach the physical processor if the sibling thread also cedes or confers.

Aix has tunable options to enable priority boosting for hot locks. Also, there is code in the FLIHs to reset priority5 back to normal priority, so the priority boost for hot locks does not boost interrupt handlers and exception handlers. There is a tunable to enable the priority boost to be preserved across the interrupts and to be kept until the job gets dispatched, but that is not the default.

Normally, AIX 5.3L maintains sibling threads at the same priority but will boost or lower thread priorities in a few key places to optimize performance. AIX 5L lowers thread priorities, when the thread is doing non-productive work spinning in the idle loop or on a kernel lock. When a thread is holding a critical kernel lock, AIX 5L boosts the thread priorities. These priority adjustments do not persist into user

mode. AIX 5L does not consider a software thread is dispatching priority, when choosing its hardware thread priority.

There where also several scheduling enhancements to exploit SMT. For example, work will be distributed across all primary threads before work is dispatched to secondary threads. The reason for this enhancement is that the performance of a thread is best when its sibling thread's is idle. AIX 5L also considers thread affinity in idle stealing and periodic run queue load balancing

5.3.8 Thread priorities on Linux

Linux will lower the thread priority for idle threads too. For dedicated LPARs, Linux will lower priority to 'low' for the idle thread and wait for SMT snooze delay period if SMT is on before snoozing the idle thread by means of a **H_CEDE** Hypervisor call. For Linux instances in a Micro-Partitioning environment, Linux always invokes **H_CEDE** for idle threads.

Linux lowers priority of the thread executing the task waiting for a spinlock too. The priority is lowered to '2' -- 'low' and restored back to '4' -- 'normal' during unlock.

5.4 Performance considerations

The basic question from a users point of view in an SMT environment evolves around which applications are going to benefit by the use of SMT and under what conditions. To answer this question several groups at IBM have started looking at performance under an SMT environment compared to ST. However, before diving into applications performance, it is beneficial to look at more fundamental issues. This includes differences on POWER4 and POWER5 due to the implementation of SMT on POWER5.

This has previously being presented elsewhere (see H. Mathis, et al. in Indications for Simultaneous Multi-Threading from workload characterization). Here we reproduce the table that compares POWER4 versus POWER5 resources.

Table 5-3 illustrates the differences due to SMT on POWER5 versus POWER4. In other words, these are some of the changes required to accomodate SMT on POWER5. The first column in the table corresponds the instruction units and caches:

- ▶ Instruction Fetch Unit (IFU)
- ▶ Instruction Decode Unit (IDU)
- ▶ Instruction Issue Unit (ISU)

- ▶ Fixed Point Unit (FXU)
- ▶ Floating Point Unit (FPU)
- ▶ Load Store Unit (LSU)
- ▶ Level 2 cache
- ▶ Level 3 cache

Table 5-3 Differences between POWER4 required to accomodate SMT on POWER5

| Unit | POWER4 | POWER5 |
|------|--|---|
| IFU | Direct mapped 64 KB Level 1 I-Cache | 2-way 64 KB Level 1 I-Cache |
| | 4-entry direct mapped prefetch buffer | Split, 2-entry per thread prefetch buffer |
| | 16-entry BIQ | Split, 8-entry per thread BIQ |
| | Branch prediction control | Replicated branch prediction control |
| | Link stack | Replicated link stack |
| IDU | 8-entry instruction fetch buffer (IFB) | 6-entry IFB per thread |
| ISU | 20-entry FIFO GCT | 20-entry linked list GCT |
| | 80 General purpose register (GPR), 72 Floating point Register (FPR) mapper | 120 GPR, 120 FPR mapper |
| | 32-entry Condition register (CR) mapper | 40-entry CR mapper |
| | 24-entry Fixed point exception register (XER) mapper | 32-entry XER mapper |
| | 20-entry Floating point issue queue (FPQ) | 24-entry packed FPQ |
| FXU | 80-entry GPR | 120-entry GPR |
| FPU | 72-entry FPR | 120-entry FPR |
| LSU | 32 K, 2-way D-Cache | 32 K, 4-way D-Cache |

| Unit | POWER4 | POWER5 |
|------|---|--|
| | 128-entry 2-way ERAT | 128-entry fully associative ERAT |
| | 64-entry segment lookaside buffer (SLB) | Replicated 64-entry SLB per thread |
| | 32-entry real LRQ | 16-entry real and 16-entry virtual LRQ per thread |
| | 32-entry real SRQ | 16-entry real and 16-entry virtual SRQ per thread |
| | 8-entry LMQ | 8-entry LMQ with thread control |
| | Interrupts | Interrupts replicated per thread |
| | | Replicated special purpose registers (SPRs) with thread ID |
| L2 | 1.45 MB on chip | 1.9 MB on chip |
| L3 | 16 MB Cache | 36 MB, directory, controller on-chip |

In the case of the Branch Information Queue (BIQ), one can see from the table 16-entry BIQ has been split into 8-entry BIO per each thread. The Load Reorder Queue (LRQ) and Store Reorder Queue (SRQ) are splitted as well, however, these 32-entry real has been further splitted into a real and virtual components. Register renaming resources were increased mostly to account for the architected state of the second thread. The Load Miss Queue (LMQ) with thread control allows for sharing it. The Global Completion Table (GCT) was shared but ordering must be maintained within a thread so the design changed. Some parts of the design required no changes for SMT, such as the Branch History Table. In the case of the caches, the associativity of the instruction cache, data cache, and data Effective-to-Real Address Translation (ERAT) table was increased.

The above paragraph may be summarized by saying that implementation of SMT on POWER5 required in some cases duplication of resources, provision of larger queue sizes, and better resource allocation algorithms.

In general, based on previous work the following rules can be summarized when it comes to application performance on SMT environments:

- ▶ Applications found in commercial environments showed higher SMT gain than scientific applications
- ▶ Selected set of experiments on different workloads have shown varying degrees of SMT gain ranging from -11% to 43%. Most of the workloads show a positive gain running on SMT environments
- ▶ An indication of SMT gain or loss may be provided by looking at LSU and LRQ. LRQ Full and Time (cycles) spent in LRQ give an indication of the amount of time that loads are waiting to be dispatched. At higher levels of these metrics, the two threads may be stifling each other with contention for cache and load/store resources. The same is true for very low rates of the LSU empty metric, which reflects higher rates of use of load/store resources. (see H. Mathis , et al. in Indications for Simultaneous Multi-Threading from workload characterization for further details). Applications with the following characteristics from the POWER4 counters most likely would show negative gain for SMT:
 - a. LRQ full > 30%
 - b. Time spent in LRQ > 150 cycles
 - c. LSU empty < 2% of the time
- ▶ Applications that show a negative SMT gain may be attributed due to rejects, L2 cache thrashing and increased local latency under SMT.

5.4.1 Cache effects

Since, with SMT, thread level parallelism is used to compensate for low instruction level parallelism, two possibly different tasks share the same processor core, on chip and off chip caches. This means there could be more associativity misses in the caches. To compensate for this, POWER5 has increased associativity of the L1 instruction cache and data cache to 2 way set associative and 4 way set associative from a zero way set associative and 2 way set associative on the POWER4. The L2 on POWER5 is now 1.88 MB 10 way set associative as against 1.5 MB 8 way set associative on POWER4. L3 on the POWER5 is now a victim cache of L2, unlike an inline L3 in POWER4. L3 runs at 1/2 the processor speed on POWER5 unlike 1/3 processor speed on POWER4. L3 being a victim cache of L2 behaves like a large albeit a bit slower L2 extension. The L3 on the POWER5 is 36 MB 12 way associative with 256 byte lines managed as two 128 byte sectors as against a 32 MB 8 way associative 512 byte lines managed as four 128 byte sectors on the POWER4. These processor enhancements help offset the cache effects due to SMT, resulting in overall improved application performance.

5.4.2 Exploitation of SMT

Performance measurements for various standard industrial benchmarks were made with AIX 5.3L on four-way pSeries ML4 POWER5 systems to validate gains from SMT. The measurements were made with SMT turned on and SMT turned off, and percentage throughput improvement calculated when SMT was turned on as against SMT turned off.

Figure 5-3 on page 107 illustrates SMT gains for various workloads for 4-way ML4 POWER5 systems. As can be seen from the chart, throughput improvement varies from 10% to 50% depending upon the workload.

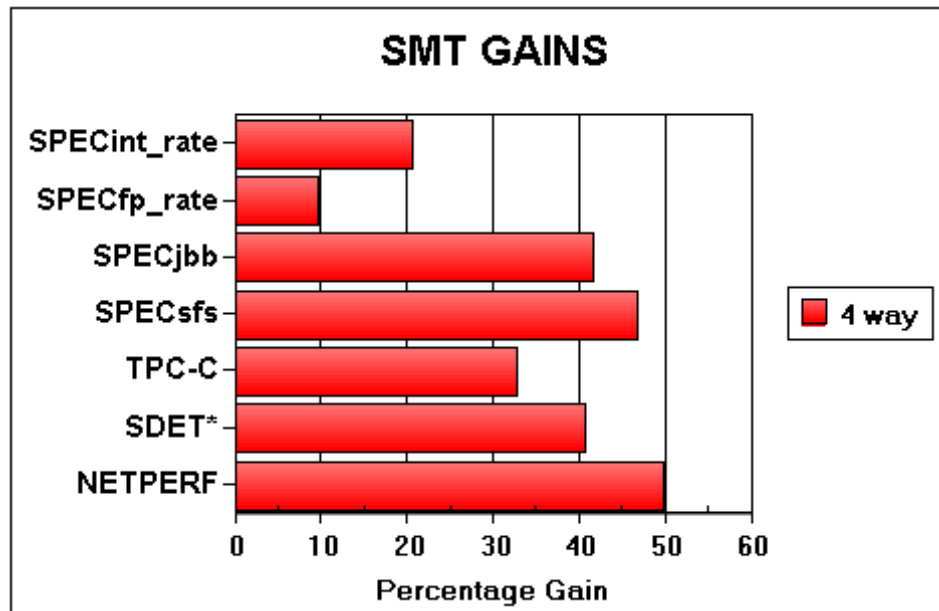


Figure 5-3 SMT gains for various workloads

5.5 Exploitation of SMT using scientific and engineering applications

In this section we present a series of examples that involve applications in the area of High-Performance Computing (HPC). These applications correspond to the Life Sciences and CAE (Computer Aided Engineering) industry:

- ▶ Gaussian03
- ▶ AMBER7
- ▶ BLAST

► FLUENT

Although all of these applications are in the same area of HPC, the algorithms utilized to carry out their simulations are not necessarily the same, thus, providing a good test for the performance of SMT under different conditions or workloads.

5.5.1 Life sciences applications

We selected three Life Sciences applications that rely on different computational methods to carry out molecular simulations. These applications are in the areas of quantum chemistry, molecular mechanics and molecular dynamics, and bioinformatics.

Gaussian (Gaussian03, Rev.C.01, Gaussian Inc., Wallingford, CT) is a connected series of programs that can be used for performing a variety of electronic structure calculations; molecular mechanics, semi-empirical, *ab initio*, and density functional theory. Gaussian consists of a collection of programs commonly referred as links; each link communicates through disk files and are grouped into overlays. Links are independent executables located labeled as lxxx.exe; where xxx is a unique number for each link.

The theoretical methods chosen in this study have been extensively discussed in the literature (W. J. Hehre, L. Radom, P. V. R. Schleyer, and J. A. Pople, *Ab Initio Molecular Orbital Theory*, John Wiley & Sons, New York, NY, 1985) and it is beyond the scope of this work to describe these methods. The approximation used in this work corresponds to Hartree-Fock. The case used in this book corresponds to one of the cases from previous studies. The molecule α -pinene at the HF level of theory using the 6-311G(df,p) basis set was used as our benchmark (A.E. Frisch and M. J. Frisch, *Gaussian03 User's Reference*, 2nd Edition, Gaussian Inc., Pittsburgh, PA).

AMBER (Assisted Model Building with Energy Refinement,) is a flexible suite of programs for performing molecular mechanics and molecular dynamics calculations based on force fields (D. A. Case, D. A. Pearlman, J. W. Caldwell, T. E. Cheatham III, J. Wang, W. S. Ross, C. Simmerling, T. Darden, K. M. Merz, R. V. Stanton, A. Cheng, J. J. Vincent, M. Crowley, V. Tsui, H. Gohlke, R. Radmer, Y. Duan, J. Pitera, I. Massova, P. A. Kollman, *AMBER7 User's Manual*, University of California San Francisco, CA.). Sander is the primary program used for molecular dynamics simulations and is the only program considered in our current study. The version used in this book corresponds to AMBER7 for IBM systems. The test that was selected to run AMBER is the jac benchmark. This is a joint AMBER-CHARMM benchmark. It considers a protein dhfr (dihydrofolate reductase) in an explicit water bath with cubic periodic boundary conditions. Details of system size and simulation conditions are: 23,558 atoms, cubic

periodic boundary box, 62.23 Å dimension, 9 Å nonbond cutoff with 2Å buffer, i.e., list with 11 Å cutoff, 1 fs time step, 1000 steps, NVE ensemble (constant energy, constant volume), bonds to hydrogen constrained (SHAKE). The particle mesh Ewald (PME) method was used for calculating the Lennard-Jones (LJ) and electrostatic interactions with 64x64x64 grid, equilibration temperature was 300K.

BLAST (Basic Local Alignment Search Tool) is a set of similarity search programs designed to explore all of the available sequence databases regardless of whether the query is protein or nucleic acid (S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, J. Mol. Biol. 215, 403 (1990)). The BLAST programs have been designed for speed, with a minimal sacrifice of sensitivity to distant sequence relationships. The scores assigned in a BLAST search have a well-defined statistical interpretation, making real matches easier to distinguish from random background hits. BLAST uses a heuristic algorithm which seeks local as opposed to global alignments and is therefore able to detect relationships among sequences which share only isolated regions of similarity.

BLAST reads a database via `mmap` and does a pattern search. The BLAST family of tools to search for similarities in pair sequences is developed at the National Center for Biotechnology Information (NCBI). BLAST is part of the development of software tools for analyzing genome data. The BLAST family set of tools are capable of searching databases regardless of whether the query is a sequence of amino acids or nucleotides. BLAST uses a heuristic algorithm to carry out local alignments. This type of alignment or search can be carried out with different programs available in BLAST.

BLAST may be considered as a three-step algorithm:

1. The program compiles a list of high-scoring strings
2. The program searches for hits, where for each successful hit it generates a seed
3. It extends the seeds

The version of BLAST used here is BLAST 2.2.6. NCBI BLAST has a wrapper called `blast all`, this wrapper then calls each of the BLAST modules, of course, depending on the nature of the query and the database. Throughout this work we only invoked `blastn`.

We used the `glil5706771|gb|AC007518.16|AC007518` Mus musculus chromosome 6 clone 345_D_4 map6 as our query. The database used is the human genome DNA sequence from Sanger center's ensemble server. The version used in this work contains 44521 sequences and 3200338544 letters.

5.5.2 CAE applications

FLUENT version 6.1.22 (FLUENT, Inc.) is a leading Computational Fluid Dynamics (CFD) application program for modeling fluid flow and heat transfer in complex geometries. FLUENT provides complete mesh flexibility, solving flow problems with unstructured meshes that can be generated about complex geometries with relative ease. CFD applications are "highly parallelizable". In order to solve the CFD problem in parallel, the mesh generated is partitioned and distributed among a set of processes started on a given set of processors. Then during the solution phase these processes perform iterative computation and cooperate with each other in arriving at the final solution. These processes (or tasks) can be distributed across several processors collocated on the same system. The IBM p570 used in these experiments is such a system with four POWER5 1.65 GHz processors where programs communicate through shared memory.

5.5.3 SMT benchmarks for sizing and capacity planing

We ran two sets of test cases for these three applications, one set for ST mode and one set for SMT. As previously described, in ST mode, only one thread is running and it can use many of the resources that would normally be allocated for the second thread.

The pSeries system used here corresponds to the IBM, 9117-570 model with 4 physical processors. The clock frequency is 1.65 GHz; where each processor has 1920 Kbytes of Level 2 cache. Memory on the system was configured to 16 GB. The level of AIX corresponds to AIX 5.3. The Fortran xlf 9.1 and the xlc 7.0 compilers were installed.

Although this section may be considered more as a benchmarks section, we hope that it can provide basic information for sizing and capacity planning for this type of applications. The objective of capacity planning is to provide an estimate of future systems resources requirements based on the present knowledge of the system utilization.

An extensive discussion of sizing and capacity planning, independent from scientific applications can be found in the following reference: *IBM @server pSeries Sizing and Capacity Planning*, SG24-7071

5.5.4 SMT exploitation via multi-processor benchmarks

Life sciences

The first benchmark test corresponds to Gaussian03. In both sets, we ran our test case 1-way (sequential), and 2-, 4-, and 8-way (parallel). In other words,

within the Gaussian notation we ran with nproc using 1, 2, 4, and 8 processors. It is important to note that the IBM eServer p5 570 had only 4 physical processors. *These benchmarks are important because they show (for the three applications presented here and with the inputs used here), first of all, that for these applications there is almost no difference running ST versus SMT when utilizing parallel jobs with the total number of physical processors or less. Secondly, we show that parallel jobs running two times the number of physical processors, when running on SMT mode, they still show additional scalability. On the other hand, that does not happen when running on ST mode.*

Figure 5-4 on page 112 illustrates the performance of Gaussian03 using multiple processors under ST and SMT modes. In this figure we can identify basically three trends. The first trend corresponds to the performance when running with 1 and 2 processors (1- and 2-way). In this case we see that running Gaussian03 under any of these two modes the performance is basically identical (less than 1% difference). The second trend may be observed when running with 4 processors or 4-way. Again, notice that this is the maximum number of physical processors in this particular system. In this case we see that when running in ST mode there is a slight advantage in performance, the percentage difference is approximately 4%. The third and last trend may be seen when requesting a run with 8 processors. Clearly, this case is requesting more than the physical number of processors available on this machine. In this case, when running on an SMT mode, as pointed out in previous chapters, the physical processors are considered logical processors (same can be said for 1-, 2-, and 4-way when running on an SMT mode). In this case we clearly see the advantage of SMT over the ST mode. The 8-way run on the SMT mode shows more than a 40% improvement in performance when compared to ST.

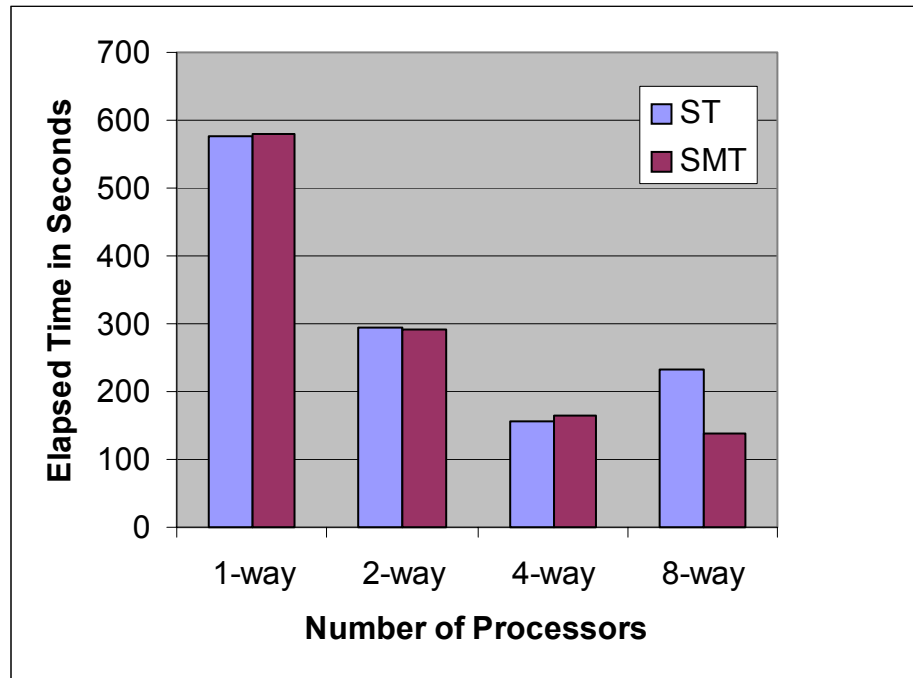


Figure 5-4 Gaussian03 parallel runs using ST and SMT modes.

The second application that we tested corresponds to AMBER7. Figure 5-5 shows similar results as in the case of Gaussian03. Again we observe exactly the same three trends. For the first trend, in the case of AMBER7 we see that there is no difference between ST and SMT. For the second trend, the performance improvement in ST mode is again, only about 4%. Finally, just as in the case of Gaussian03 we see a large performance improvement in SMT mode when running with 8 logical processors or 8-way. The SMT gain is of the order of 25%.

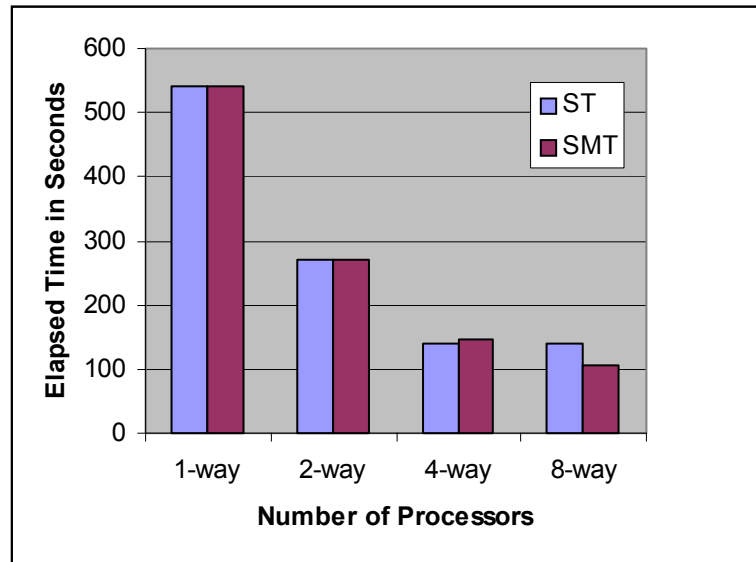


Figure 5-5 AMBER7 paprallel runs using ST and SMT modes.

Figure 5-6 illustrates the results for the last application that we tested, that is, BLAST. The same trends as before are observed here as well. However, in this case, BLAST tends to favor SMT more that the other two applications. Trend one is the same, except that for the 2-way run BLAST favors the SMT mode by about 2%. This difference is slightly higher than in the previous cases. The second trend is almost non-existent for BLAST, the difference is only 1% in favor of the ST mode.

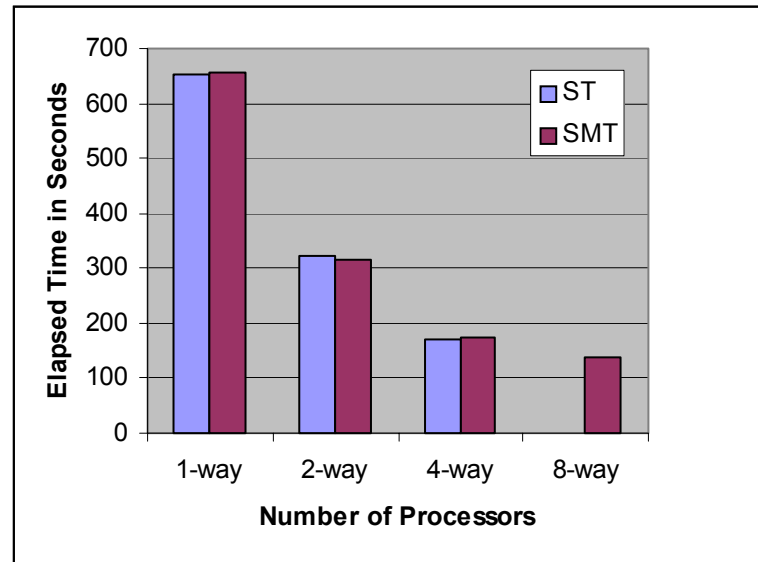


Figure 5-6 BLAST paprallel runs using ST and SMT modes.

CAE

In this section we try to determine whether FLUENT application could benefit from the SMT mode.. The measure of performance used in this experiment is "FLUENT rating" which is the number of FLUENT jobs that can be completed in a 24-hour time period. Higher values of FLUENT rating indicate better performance.

The application is submitted requesting 1-way (sequential), and 2-, and 4-way(parallel) when the system is configured on ST mode. To ensure that each thread is running on a different physical or logical processor, we use the **bindprocessor** command. When the system was configured on SMT mode, the single parallel job is submitted using 1, 2, 4, and 8 processes. In the experiments where SMT was used, one CPU was assigned to two processes of the parallel job. When the parallel job contained one process, complete resources of a CPU were assigned to the process under both ST and SMT.

The results of running single parallel job on the ST and SMT configurations are shown in Table 5-4. When the parallel job contains one process, the results for

both ST and SMT are almost identical indicating that the SMT feature imposes no overhead. The performance of ST and SMT is compared for a given number of physical CPUs. The number processes in the parallel job is equal to the number of physical CPUs in ST and it is double the number of physical CPUs in SMT. Based on the results in Table 5-4, the SMT gives 33% boost in performance for one physical CPU. When the system is fully loaded the improvement is slightly less at 23%. This improvement resulted in super-linear speed-up when the single process run is used to compute the speed-up.

Table 5-4 Performance of parallel FLUENT testcase: FL5M3. .

| Physical CPUs | Single Thread Mode | | | Simultaneous Multi-Thread Mode | | | |
|---------------|--------------------|---------------------------|---------|--------------------------------|---------------------------|---------|-------------------|
| | Processes | FLUENT Rating jobs/day(A) | Speedup | Processes | FLUENT Rating jobs/day(B) | Speedup | SMT vs ST (B)/(A) |
| 1 | | | | 1 | 166.6 | 1.0 | |
| 1 | 1 | 166.8 | 1.0 | 2 | 221.4 | 1.3 | 1.33 |
| 2 | 2 | 334.3 | 2.0 | 4 | 415.4 | 2.5 | 1.24 |
| 4 | 4 | 625.0 | 3.8 | 8 | 768.2 | 4.6 | 1.23 |

5.5.5 SMT exploitation via throughput benchmarks

Life sciences

In this section we look at the performance of the two modes, ST and SMT, via a series of throughput benchmarks. In the previous section we start with Gaussian03, then AMBER7 and finally BLAST. In this case, Figure 5-7 illustrates the performance of Gaussian03 with a series of throughput benchmarks. We ran the throughput benchmarks by carrying out a single calculation on a standalone system, this was the only process running. We refer to this scenario as a single job. Once the job was done, we simultaneously submitted two jobs, three jobs, and so on all the way up to eight simultaneous jobs.

Similarly as in the case of parallel jobs, we can identify three trends. The first trend corresponds to the throughput benchmarks consisted of 1, 2, and 3 simultaneous jobs. The second trend is with 4 simultaneous jobs and the final trend corresponds to 5, 6, 7, and 8 simultaneous jobs. In the case of the first trend we see that basically there is no difference in performance between the ST and SMT modes. The percentage differences are less than 1%. The second trend is when the number of processors is the same as the number of jobs submitted. In this case we already start seeing the benefit of the SMT mode, clearly the SMT mode outperforms the ST mode. As the number of simultaneous jobs is increased the effect becomes more dramatic.

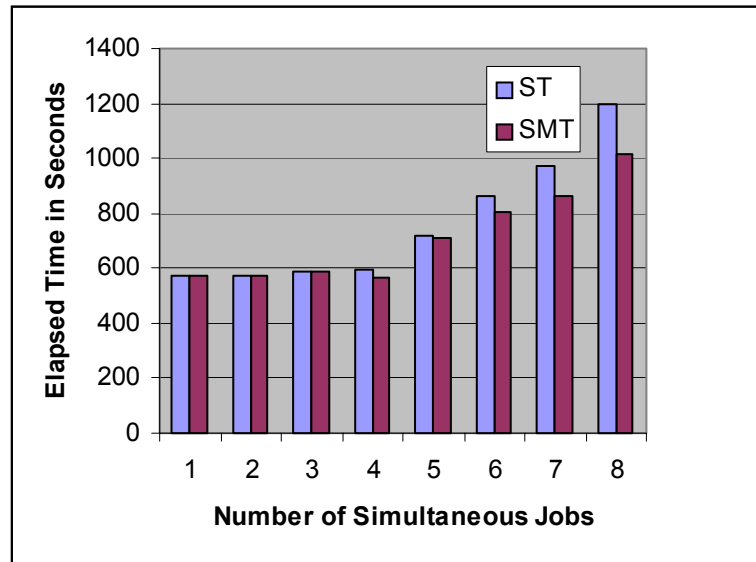


Figure 5-7 Gaussian03 Series of throughput benchmarks using ST and SMT modes.

This difference between ST and SMT can be seen in Figure 5-8. This figure clearly shows that from 1 to 3 simultaneous jobs, there is not much difference between ST and SMT. However, from 4 up to 8 simultaneous jobs the advantage of SMT is clear. In the case of 5 jobs running simultaneously, the difference that we are seeing in this case compared to 4 and 6 simultaneous jobs might be due to the fact that the operating system maintains daemons running on the background (very low CPU consumption) and therefore for a certain period of time they both were competing for resources. Explicitly binding to processor may alliviate this behavior.

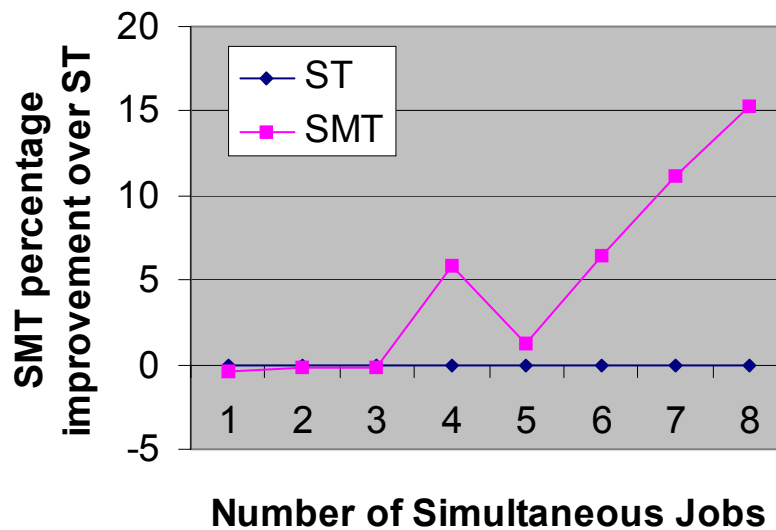


Figure 5-8 Performance advantage of the SMT mode over the ST mode.. ST was used as the baseline for all the measurements and it was set to zero.

The next case corresponds to AMBER7, similarly as in the case of Gaussian03 we have carried throughput benchmarks with 1, 2, ..., 8 running simultaneous copies of the JAC input. The results presented in Figure 5-9 are similar to the results discussed for Gaussian03. Perhaps the largest qualitative difference between this case and Gaussian03 is for 4 simultaneous jobs. AMBER7 does not seem to be taking as much advantage of SMT as Gaussian did for this particular case.

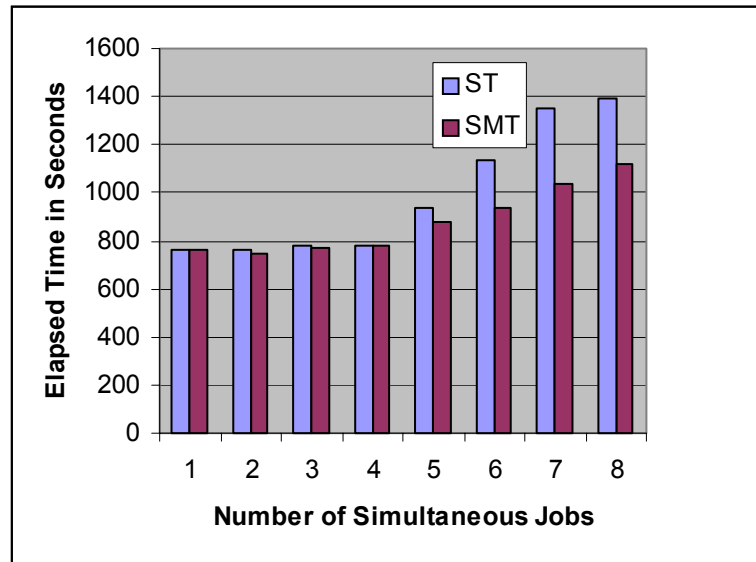


Figure 5-9 AMBER7Series of throughput benchmarks using ST and SMT modes.

Figure 5-10 illustrates how AMBER7 takes advantage of SMT as a function of the number of simultaneous jobs running on the machine. Again, from 1 to 4 simultaneous jobs, we see AMBER7 taking slight advantage of SMT. However, as the number of simultaneous jobs increases, so does the advantage of the SMT mode over the ST mode. We see that in this case, when running 7 simultaneous jobs, the improvement when compared to ST is as high as 25%. The behaviour of the case with 8 simultaneous jobs may be explain in a similar way as in the case of Gaussian, that is, operating system noise.

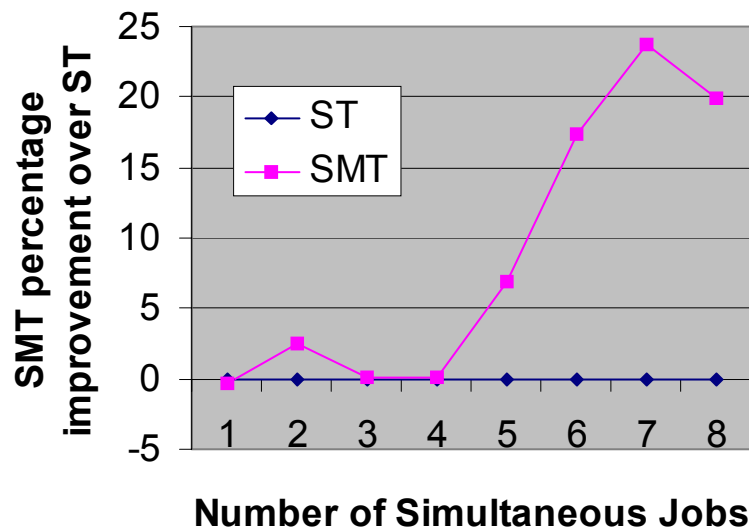


Figure 5-10 Performance advantage of the SMT mode over the ST mode. ST was used as the baseline for all the measurements and it was set to zero.

Figure 5-11 summarizes all the results for BLAST. BLAST is the last application that we tested. However, it represents an important case because computationally BLAST is different from the other two applications. Gaussian03 and AMBER7 are floating point intensive applications while BLAST relies on pattern matching, as described in Section 5.5.1. Again, we see that BLAST shows a similar behavior as the two previous applications. We see that for cases with 1 to 4 jobs running simultaneously, little advantage is taken of the SMT mode. However, as soon as the number of jobs exceeds the number of physical processors, the advantage of SMT is clear.

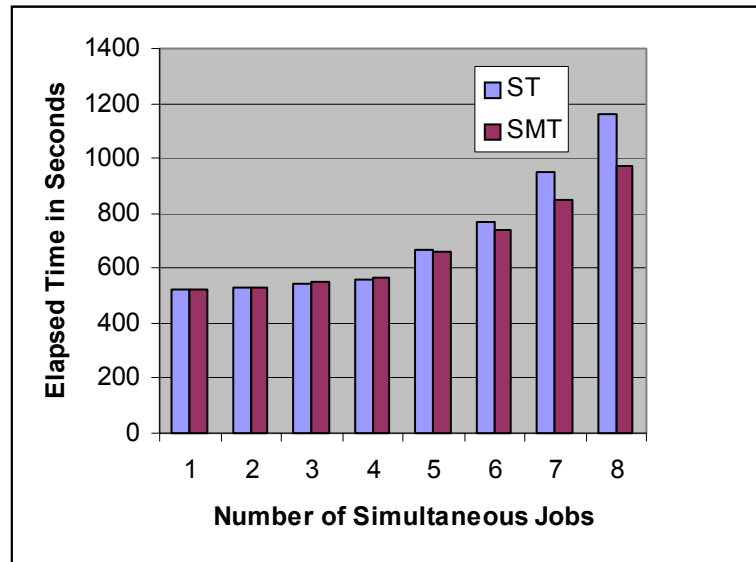


Figure 5-11 BLAST series of throughput benchmarks using ST and SMT modes.

A more dramatic difference showing the benefit of SMT can be seen in Figure 5-12. We see that in the case where we have doubled the number of jobs compared to the number of physical processors, SMT shows a performance improvement over ST by as much as 20% difference.

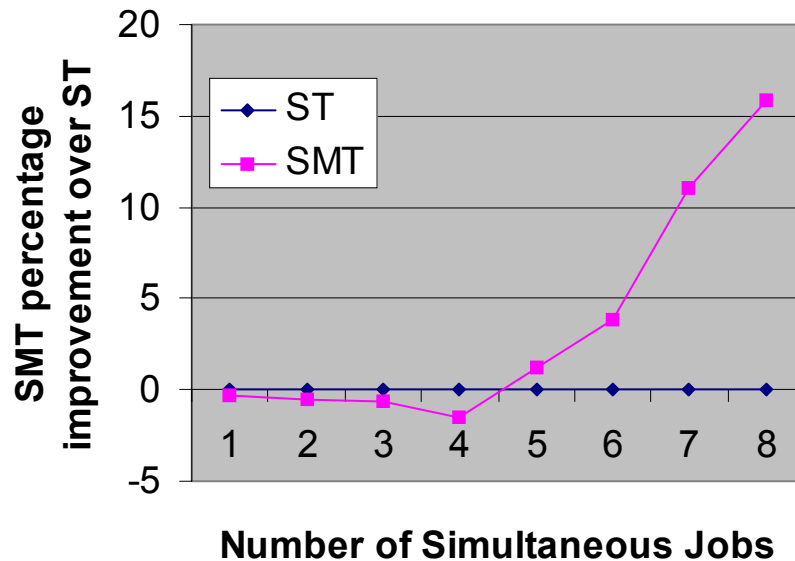


Figure 5-12 Performance advantage of the SMT mode over the ST mode. ST was used as the baseline for all the measurements and it was set to zero.

CAE

In order to evaluate the performance of ST and SMT features on a throughput benchmark, a set of several serial jobs was submitted simultaneously and the FLUENT rating for each job was measured. The total throughput was computed by multiplying the number of processes and the average throughput for the set of jobs. On SMT mode three sets of jobs were used. These sets contained 1, 2, and 4 jobs respectively. Each job in each of these sets was assigned to a CPU. For SMT configuration, four sets of jobs were submitted. These four sets contained 1, 2, 4, and 8 jobs. One CPU was used for two job in each these sets.

The results of running several serial jobs on the ST and SMT configurations are shown in Table 5-5. The performance of ST and SMT is compared for a given number of physical CPUs. The number of jobs in the parallel run is equal to the number of physical CPUs in ST, and it is double the number of physical CPUs in SMT. Based on the results in Table 5-5, the SMT gives 35% boost in performance for a single physical CPU.

Table 5-5 Throughput performance of serial FLUENT for testcase: FL5M3.

| Single Thread Mode | | | | Simultaneous Multi-Thread Mode | | | | |
|--------------------|------|----------------------------------|--|--------------------------------|------|----------------------------------|--|---------------------------|
| FLUENT Jobs | CPUs | FLUENT Rating for single job (B) | Total FLUENT Rating for all jobs (A)*(B) | FLUENT Jobs | CPUs | FLUENT Rating for Single job (E) | Total FLUENT Rating for all jobs (D)*(E) | SMT vs ST (D)*(E)/(A)*(B) |
| | | | | 1 | 1 | 166.6 | 166.6 | |
| 1 | 1 | 166.8 | 166.8 | 2 | 1 | 112.2 | 224.4 | 1.35 |
| 2 | 2 | 167.3 | 334.6 | 4 | 2 | 109.7 | 436.0 | 1.31 |
| 4 | 4 | 163.6 | 654.4 | 8 | 4 | 108.5 | 871.2 | 1.33 |

5.5.6 SMT exploitation on production system

In the previous section we tried to illustrate a series of scenarios where scientific applications can take full advantage of the SMT mode. We included the throughput benchmarks in order to replicate the workloads that a supercomputing center might experience on a day to day basis. However, these throughput benchmarks were carried by running multiple copies of a single application. Of course, if the input is identical, all the particular jobs will be competing for the same resources. However, in order to provide a more balanced representation of a real workload, we combined the three codes in one throughput benchmark. The results from this benchmark are summarized on Figure 5-13.

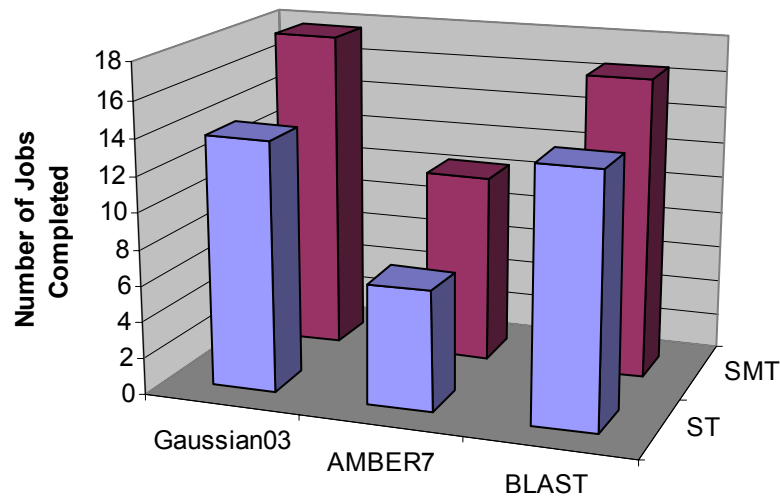


Figure 5-13 A balance throughput benchmark.

What we are trying to measure with this benchmark is which mode will provide the best throughput results. However, prior to discussing the results, it is important to define how we ran this benchmark. Here the amount of time that the benchmarks were going to run was pre-defined. We selected 90 minutes. This time was chosen based on how long individuals run take. Given that constraint the threshold of 90 minutes was arbitrary. Then we wrote a script that would submit, Gaussian03, AMBER7 and BLAST jobs simultaneously. Once the script reached 90 minutes, all the jobs were stopped and the total number of completed jobs in this period of time was used as the measurement of performance.

For this type of benchmarks, we see that the SMT benefit is clear. We see performance improvements from 20% to almost 60% difference when compared to ST. The largest improvement corresponds to AMBER7 with almost 60% in comparison with ST.

6

Chapter 6.

Micro-Partitioning

In this chapter we will discuss the detailed implementation for Micro-Partitioning, which is one of the key features provided in POWER5 and AIX 5.3.

The following topics will be included:

- ▶ Partitioning on POWER5
- ▶ Micro-Partition Implementation
- ▶ Performance Considerations
- ▶ Configuration Guidelines

6.1 Partitioning on the IBM @server p5

With technology inspired by IBM's zSeries heritage, logical partition (LPAR) appeared on IBM @server pSeries from the POWER4 based systems with AIX 5.1 environment in the year of 2001. Logical partitioning of a system allows more than one operating system to reside on the same platform simultaneously without interfering with each other. With POWER4 technology, the smallest granularity of an operating system image was *one* processor. All partitions are considered “dedicated”, in that the processors assigned to a partition can only be in whole multiples and only used by that partition. This means a *32-way* p690 can host up to 32 independent partitions for running a combination of AIX and Linux.

Continuing the evolution of the partitioning technology on pSeries servers, the IBM @server p5 extends its capabilities by further improving the flexibility in using partitions. There are two types of partitions in the IBM @server p5. Partitions can have processors dedicated to them (Dedicated Processor Partition), or they can have their processors virtualized from a pool of shared physical processors (Micro-Partitioning). Both types of partitions can coexist in the same system at a given time.

Dedicated Processor Partitions

A *dedicated processor partition*, like the partitions used on POWER4 processor based servers, have entire processor assigned to them. These processors are owned by the partition where they are running and are not shared with other partitions. The amount of processing capacity on the partition is limited by the total processing capacity of the processors configured in that partition, and it cannot go over this capacity (unless you add more processors inside the partition using a dynamic LPAR operation).

By default, a powered-off logical partition using dedicated processors will have its processors available for use by other partitions in the system.

An important difference between dedicated partitions in POWER5 systems and the partitions in POWER4 systems is the ability to use Virtual Ethernet and Virtual SCSI, although they can still have the physical I/O resources if desired. Virtual Ethernet is covered in section 7.3, “Virtual Ethernet” on page 181 and Virtual SCSI is covered in section 7.5, “Virtual SCSI” on page 225.

Micro-Partitions

The goal of *Micro-Partitioning* is to allow the resource definition of a partition to allocate processor resources with more granularity allowed than available with dedicated processor partitions and to share unused processing power between partitions. This allows a system to perform more efficiently than would be required with dedicated processor partitions.

Micro-Partitioning differs from the dedicated processor partition in the sense that physical processors are abstracted onto *virtual processors* which are then assigned to partitions. These virtual processors have capacities ranging from 10 percent of a physical processor, up to the entire processor. A system can therefore have multiple partitions sharing the same physical processor, and dividing the processing capacity among themselves, as shown in Figure 6-1.

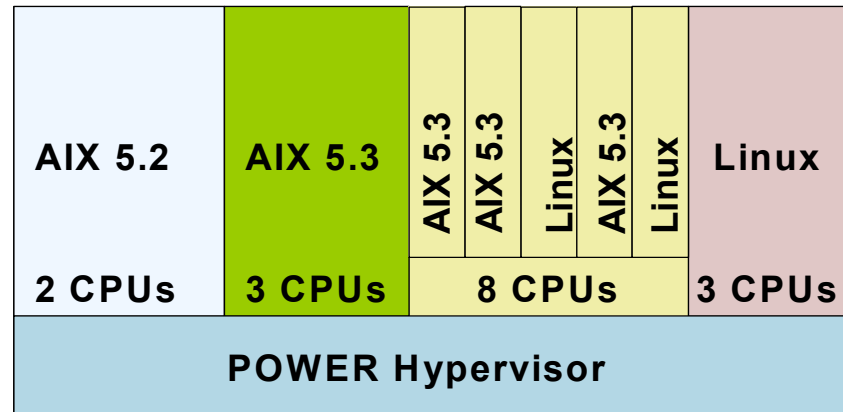


Figure 6-1 System with dedicated processor partition and Micro-Partitions

The virtual processor abstraction is implemented in the hardware and the POWER Hypervisor. From an operating system perspective, a virtual processor is indistinguishable from a physical processor. The key benefit of implementing partitioning in the hardware/firmware is to allow any operating system to run on POWER5 technology with little or no changes. Optionally, for most resource flexibility, the operating system can be enhanced to exploit Micro-Partitioning more in-depth. For instance an operating system may voluntarily relinquishes processor cycles to the Hypervisor when they are not needed. AIX 5L version 5.3 is the first version of AIX to support Micro-Partitioning, SUSE LINUX Enterprise Server 9 for POWER systems, and Red Hat Enterprise Linux AS for POWER Version 3 also includes such optimizations. A more detailed explanation will be introduced in the following sections.

Another virtualization technology provided on POWER5 is virtual I/O (VIO). This allows a physical adapter on one partition to be shared by one or more partitions, enabling customers to consolidate and potentially minimize their number of physical adapters. Virtual I/O may be used to reduce costs by eliminating the requirement that each LPAR have a dedicated network adapter, disk adapter and disk drive. Both dedicated LPAR and Micro-Partition on POWER5 can use VIO function. The detailed introduction on VIO will be described in the next chapter.

6.2 Micro-Partitioning Implementation

The virtualization of physical processors on POWER5-based server requires a new partitioning model, since it is fundamentally different from the partitioning model used on POWER4-based servers. With Micro-Partitioning, each physical processor is abstracted into one or more virtual processors, which are assigned to partitions, so physical processors are occupied by different partitions's virtual processors in a time-sliced fashion. There are several new terminologies and concepts introduced in Micro-Partitioning.

Processing Unit

Processing unit is define on HMC when creating a Micro-Partition. It corresponds to the processing capacity which can be configured in fractions of 1/100 of a physical processor. The minimum amount of processing capacity which can be assigned to a Micro-Partition is 1/10. For example, we define 0.1 processing unit on HMC, it corresponds to 10% of a physical processor capacity, and a processing unit of 0.25 means we can use 25% of physical processor capacity in our Micro-Partition.

When a Micro-Partition is activated, the system will choose the partition's entitled processing capacity (represented as *ent* in some AIX commands, such as **mpstat**, etc.) from the user specified range. The entitled processing capacity is a commitment of capacity that is reserved for the partition. The desired processing unit will be chosen if there is enough physical processor capacity available, or a value which must be greater than or equal to the minimum processing unit will be assigned. If these two conditions can not be satisfied, the partition will not be started.

Layers of Processor Abstraction

The following are the terminologies on processors used in Micro-Partitioning:

| | |
|--------------------|--|
| Physical Processor | The actual physical hardware resource. Currently, the maximum number of physical processors in the POWER5 systems is 64. This definition is the number of unique processor cores, not the number of processor chips (each of which contains two processing cores). |
| Virtual Processor | Entered from HMC, it defines the way that a partition's entitlement may be spread over physical processor. The virtual processor is the unit of Hypervisor dispatch and the granularity of processor dynamic reconfiguration. Currently, the maximum number of virtual processors is 64 per partition. |
| Logical Processor | A hardware thread, an operating system view of managed processor unit. In the AIX 5.3 operating system, each |

hardware thread appears as a unique processor (e.g. bindprocessor -q). The number of logical processors will be doubled the virtual processors with Simultaneous Multi-Threading (SMT) turning on, both hardware threads on one virtual processor must be in the same partition at the same time. Currently, the maximum number of logical processors is 128 per partition.

Processor Pools

Micro-Partitioning is enabled in the hardware and POWER5 Hypervisor which is a component of global firmware. The Hypervisor schedules Micro-Partition from a set of physical processors that is called the

pool. There are up to three pools of physical processor on a system, one is for dedicated processor partitions, one is for Micro-Partitioning (shared processor pool), and the other is for unallocated processors.

Each partition is presented with the resource abstraction for its partition and other required information through the Open Firmware Device Tree. The device tree is created by firmware and copied into the partition before the operating system is started. Operating systems receive resource abstractions, and also participate in the partitioning model by making Hypervisor calls at key points in their execution as defined by the model.

Micro-Partitioning allows several OS images to coexist on a physical processor in a time-sliced manner, dedicated memory is still needed, but the partitions I/O requirements can be supported through Virtual Ethernet and Virtual SCSI Server. Currently the maximum number of Micro-Partition for one system is the number of physical processors in whole system times 10, and up to 254 partitions per system is supported.

6.2.1 Implementation of Virtual Processor Dispatch

Micro-Partition runs on virtual processors. The capacity of these virtual processors comes from a shared pool of physical processors. There is no fixed relationship between a virtual processor and the physical processor that it operates upon. The virtual processors are dispatched in time-sliced manner on the physical processors under the control of the Hypervisor, much like the operating system time slices software threads.

The Hypervisor utilizes HDECRC register to drive its partition dispatcher. HDECRC is a clock interrupt source utilized by Hypervisor to preempt a dispatched partition and regain control of the physical processor. For the details of HDECRC, please refer Chapter 3, "POWER Hypervisor" on page 39 of this book.

Dispatch Wheel

The Hypervisor uses the architectural metaphor of a “dispatch wheel” with a fixed rotation period of 10 milliseconds to guarantee that each virtual processor receives its share of the entitlement in a timely fashion. This means that the entitled processing unit of each partition is distributed to one or more virtual processors which will then be dispatched onto physical processors in a time-slice manner during every 10 ms dispatch wheel. The time that each virtual processor gets dispatched depends on the number of virtual processors and the *entitled processing capacity* that has been assigned to that partition from HMC by system administrator. When a partition is fully busy, the partition entitlement is evenly distributed amongst its online virtual processors.

The Hypervisor manages a dispatch wheel for each physical processor in the shared pool. Figure 6-2 illustrates the assignment of virtual processors to a physical processor.

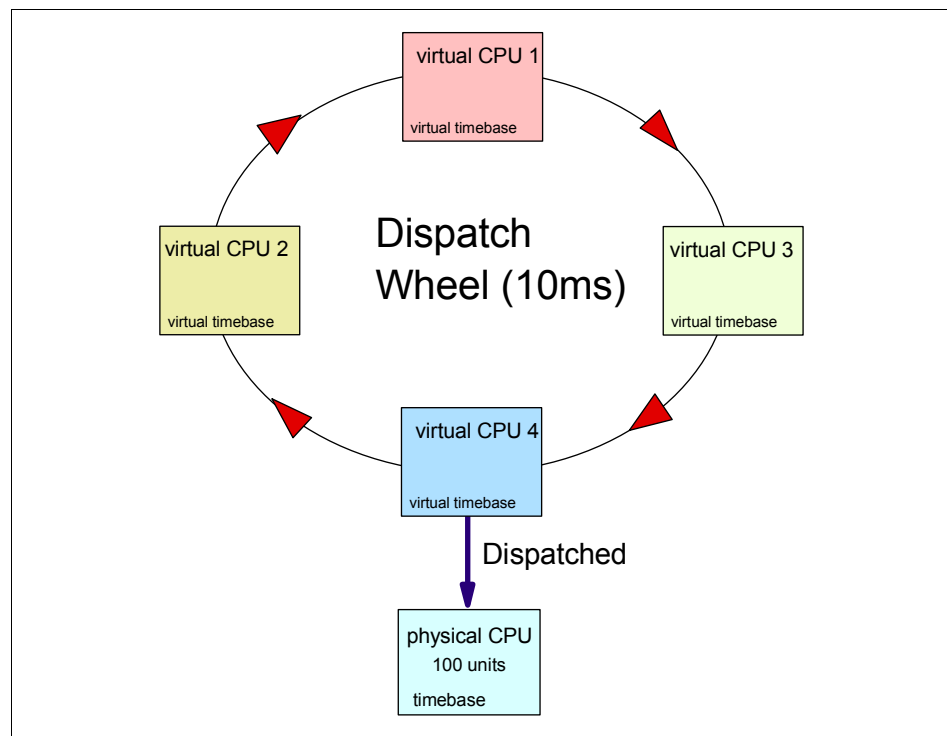


Figure 6-2 Dispatch wheel

Initially, if the available physical processor entitlement in the whole system meet the requirement defined for a Micro-Partition, the partition will be started, and Hypervisor will begin to dispatch the required virtual processors to each physical

processor evenly. For every subsequent time slice, Hypervisor does not commit that all the virtual processors will be dispatched in the same order, neither assures that virtual processors in a given partition are dispatched together, but it ensures every partition get its entitlement if it need (maybe the partition will cede or confer its cycles back).

The dispatch wheel works the same way when SMT is turned on for a processor or group of processors. The two logical processors (related to single virtual processor by SMT) are dispatched together whenever the Hypervisor schedules the virtual processor to run. The amount of time that each virtual processor run is split between the two logical processors. Figure 6-3 shows a diagram for a case when SMT is enabled.

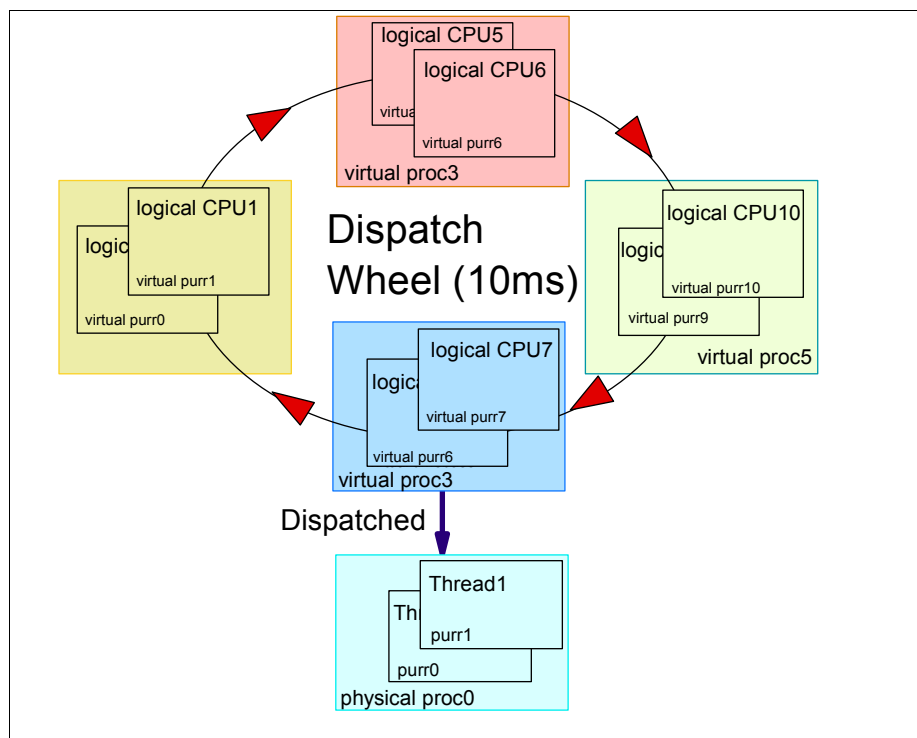


Figure 6-3 Dispatch wheel for SMT-enabled processors

Affinity Policy

The Hypervisor attempts to dispatch work in a way that maximizes processor, cache, and memory affinity. When the Hypervisor is dispatching a virtual processor (VP), it will first attempt to use the same physical processor this VP was previously dispatched on. Otherwise, it will be dispatched to the first available processor in the following order, same chip, same multi-chip module

(MCM) or same node. If a physical processor becomes idle, the Hypervisor will look for work for it. Priority will be given to VPs that have an affinity for that processor. If none can be found, then the Hypervisor will select a VP that has affinity to no real processor and, finally, will select a VP that is uncapped. The fields of “S0rd” to “S5rd” in the output of AIX command **mpstat** shows thread redispatch affinity domains in the partition.

Processor Utilization Resource Register

Since the amount of time a virtual processor run depends on the scheduling by the Hypervisor, the CPU time consumed, as perceived by a processor does not correspond to the real elapsed time.

The POWER5 processor architecture deals with this issue by introducing a new processor register that is intended for measuring utilization. This new register, called the Processor Utilization Resource Register (PURR), is described in Chapter 2, “POWER5 Architecture” on page 9 and is used to approximate the time that a virtual processor is actually running on a physical processor. The register increments automatically so that the operating system can always get the current up-to-date value. The Hypervisor saves and restores the register across virtual processor context switches to simulate a monotonically increasing atomic clock at the virtual processor level.

Each hardware thread has a PURR. It is accumulated at a fixed rate whenever the thread running on a virtual processor is dispatched on a physical processor. For a cycle in which no instructions are dispatched, the PURR of the thread that last dispatched an instruction is accumulated. By specifying a PURR per thread instead of a single PURR per processor, the POWER Hypervisor can measure performance for each hardware thread separately.

Operating System Support

In general, operating systems and applications running in Micro-Partitions need not be aware that they are sharing processors. However, overall system performance can be significantly improved by minor operating system changes. The main issue here is that the Hypervisor cannot distinguish between the operating system doing useful work, and, for example, spinning on a lock. The result is that the operating system may waste much of its entitlement doing nothing of value.

AIX 5L version 5.3 provides support for optimizing overall system performance of Micro-Partition. These optimizations are built around the idea that an operating system can provide hints to the Hypervisor about scheduling. For example, an operating system can signal to the Hypervisor when it is no longer able to schedule work, and it can give up the remainder of its time. This results in better utilization of the real processors in the shared processor pool.

The dispatch mechanism may utilize hcalls to communicate between the operating system and the Hypervisor. The major three hcalls used by operating systems are H_CED, H_CONFER and H_PROD. For the definition of hcall, please refer section 3.0.2, "Hypervisor Call Functions" on page 43.

When a virtual processor is active on a physical processor and the operating system detects an inability to utilize processor cycles, it may cede or confer its cycles back to the Hypervisor, enabling it to schedule another virtual processor on the physical processor for the remainder of the dispatch cycle. Reasons for a cede or confer may include the virtual processor running out of work and becoming idle, entering a spin loop to wait for a resource to free. There is no concept of credit for cycles that are ceded or conferred. Entitled cycles not used during a dispatch interval are lost.

A virtual processor that has ceded cycles back to the Hypervisor can be reactivated using a H_PROD Hypervisor call. If the operating system running on another virtual processor within the logical partition detects that work is available for one of its idle processors, it can use H_PROD to signal the Hypervisor to make the prodded virtual processor runnable again. Once dispatched, this virtual processor would resume execution at the return from the cede Hypervisor call.

Virtual Processor State

There are 4 logical states that a virtual processor could be in:

| | |
|----------------------------|---|
| Running | Currently dispatched onto a physical processor. |
| Runnable | Currently not running, but ready to run. The queue of runnable virtual processors represents a first-in, first out (FIFO) queue for selecting the next virtual processor to dispatch to a physical processor. |
| Not-Runnable | The state of a virtual processor that has released its cycles either by calling H_CED or H_CONFER Hypervisor calls. In the cede case, either an interrupt or an h_prod() call from another virtual processor makes this virtual processor runnable again. In the confer case, a h_prod() call, or a dispatch cycle granted to the conferred targets will make the virtual processor runnable again. |
| Entitlement Expired | The state of all virtual processors who have received their full entitlement for the current dispatch window. |

Tracing Virtual Processor Dispatch

The dispatching of virtual processors by the Hypervisor does not involve the operating system running in the partition. The operating system cannot directly monitor the rate or characteristics of context switching from the Hypervisor. However, there is a communication area which is shared between the Hypervisor

and each virtual processor in a partition, with which AIX operating system can, via the AIX trace, monitor some aspects of virtual processor context switching.

When AIX trace is used, the 419 major trace hook represents the information available about context switching. The duration of being undispached cannot be accurately determined from the last trace hook before the 419 hook until the 419 hook, since this time only represents the time between observing trace hooks. Rather, the time the virtual processor was undispached is encoded in the trace hook as well as the “start wait” field. The time that the virtual processor is redispached is encoded as the “end wait” field. We can see the statistics from the following sequence of an AIX trace.

Some measure of virtual processor to physical processor affinity is possible as well. The trace hook shows an index to the physical processor (field *vProcIndex*). The index is not necessarily an indicator of the physical processor number of the system, but the index are fixed over time. So, if a virtual processor number is dispatched to the same *vProcIndex* as the previous dispatch, affinity is maintained.

In this example, observed the CPU 3 preempt hook. This hook denotes the point at which the virtual processor is redispached on the physical processor. Since CPU 2 and 3 are the SMT threads associated with a virtual processor, they are dispatched and undispached together.

Example 6-1 Virtual processor dispatch from AIX tracing

| ID | PROCESS NAME | CPU | PID | TID | I | SYSTEM CALL | ELAPSED_SEC | DELTA_MSEC | APPL | SYS CALL | KERNEL | INTERRUPT | | |
|-----|--------------|-----|--------|--------|---|-------------|-------------|------------|------|-----------------------|-------------------------|---------------------------------|---------------|------------|
| 234 | -229498- | 3 | 229498 | 819359 | | | 0.011917924 | 0.030173 | | clock: | iar=000000000017B74C | lr=000000000017B770 [2196 usec] | | |
| 100 | -229498- | 2 | 229498 | 819359 | | | 0.011918199 | 0.000275 | | DECREMENTER INTERRUPT | iar=17B74C | cpuid=02 | | |
| 200 | -229498- | 2 | 229498 | 819359 | | | 0.011918585 | 0.000386 | | resume | -229498- iar=17B74C | cpuid=02 | | |
| 112 | -229498- | 2 | 229498 | 819359 | | | 0.011919343 | 0.000758 | | krlock: | cpuid=02 | addr=F100060004006B80 | action=spin | |
| 419 | -229498- | 3 | 229498 | 876731 | | | 0.020218416 | 5.926309 | | cpu preemption data | Preempted | vProcIndex=0005 | | |
| | | | | | | | | | | | rtrdelta=0000 | enqdelta=17321 | exdelta=202DC | |
| | | | | | | | | | | | start wait=2D33E3B52A87 | end wait=2D33E3CE5F7B | | |
| | | | | | | | | | | | SRR0=000000000017B770 | | | |
| | | | | | | | | | | | SRR1=8000000000009032 | | | |
| 234 | -229498- | 3 | 229498 | 876731 | | | 0.020219329 | 0.000913 | | clock: | iar=000000000017B770 | lr=000000000017B770 [8301 usec] | | |
| 100 | -229498- | 3 | 229498 | 876731 | | | 0.020219913 | 0.000584 | | DECREMENTER INTERRUPT | iar=17B770 | cpuid=03 | | |
| 200 | -229498- | 3 | 229498 | 876731 | | | 0.020221937 | 0.002024 | | resume | -229498- iar=17B770 | cpuid=03 | | |
| 112 | -229498- | 3 | 229498 | 876731 | | | 0.020223333 | 0.001396 | | krlock: | cpuid=03 | addr=F100060004006B80 | action=spin | |
| 419 | -229498- | 2 | 229498 | 819359 | | | 0.020325289 | 0.101956 | | cpu preemption data | Preempted | vProcIndex=0004 | | |
| | | | | | | | | | | | rtrdelta=0000 | enqdelta=1732A3 | exdelta=202DB | |
| | | | | | | | | | | | start wait=2D33E3B529FC | end wait=2D33E3CE5F7A | | |
| | | | | | | | | | | | SRR0=000000000017B74C | SRR1=8000000000009032 | | |
| 100 | -229498- | 2 | 229498 | 819359 | | | 0.020325820 | 0.000531 | | I/O INTERRUPT | iar=17B74C | cpuid=02 | | |
| 492 | -229498- | 2 | 229498 | 819359 | | | 0.020327162 | 0.001342 | | h_call: | start H_XIRR | iar=3A76BF0 | p1=188FD50 | p2=1795048 |
| | | | | | | | | | | | p3=2D33E3CEBBC1 | | | |
| 492 | -229498- | 2 | 229498 | 819359 | 1 | | 0.020332123 | 0.004961 | | h_call: | end H_XIRR | iar=3A76BF0 | rc=0000 | |
| 47F | -229498- | 2 | 229498 | 819359 | 1 | | 0.020332519 | 0.000396 | | phantom interrupt | cpuid=02 | | | |
| 200 | -229498- | 2 | 229498 | 819359 | 1 | | 0.020333634 | 0.001115 | | resume | -229498- iar=17B74C | cpuid=02 | | |
| 112 | -229498- | 2 | 229498 | 819359 | | | 0.020334513 | 0.000879 | | krlock: | cpuid=02 | addr=F100060004006B80 | action=spin | |
| 100 | -229498- | 3 | 229498 | 876731 | | | 0.020342957 | 0.008444 | | I/O INTERRUPT | iar=17B770 | cpuid=03 | | |
| 492 | -229498- | 3 | 229498 | 876731 | 1 | | 0.020343290 | 0.000333 | | h_call: | start H_XIRR | iar=3A76BF0 | p1=1880D50 | p2=1794248 |
| | | | | | | | | | | | p3=2D33E3CEC99D | | | |

Example on Virtual Processor Dispatch

The following is an example (showed in Table 6-1 on page 135) which will help us understand virtual processor (VP) dispatching further. In this example, there are three Micro-Partitions defines, sharing the processor cycles from two physical processors, spanning two 10 ms Hypervisor dispatch intervals.

Table 6-1 *Micro-Partition definition:*

| LPAR | Capacity Entitlement | Virtual Processors | Capped/ Uncapped |
|------|----------------------|--------------------|------------------|
| 1 | 0.8 | 2 | capped |
| 2 | 0.2 | 1 | capped |
| 3 | 0.6 | 3 | capped |

Logical partition 1 is defined with an entitlement capacity of 0.8 processing units, with two virtual processors. This allows the partition getting 80% of physical processor capacity from the shared processor pool for each 10 ms dispatch window. For each dispatch window, the workload is shown to use 40% of each physical processor capacity during each dispatch interval. It is possible for a VP to be dispatched more than one time during a dispatch interval. Note that in the first dispatch interval, the workload executing on VP 1 is not a continuous utilization of physical processor resource. This can happen if the operating system confers cycles, and is reactivated by a prod Hypervisor call.

Logical partition 2 is configured with one virtual processor and a capacity of 0.2 processing units, entitling it to 20% usage of a physical processor during each dispatch interval. In this example, the virtual processor dispatched during the two dispatch wheel are assigned to the same physical processor according to affinity dispatch policy.

Logical partition 3 contains three virtual processors, with an entitled capacity of 0.6 processing units. Each VP contains 20% of a physical processor in each dispatch interval, but in the case of VP 0 and VP2, the physical processor they run on is changed in the two dispatch intervals. The Hypervisor does attempt to maintain physical processor affinity when dispatching virtual processors. As described previously, it will always first try to dispatch the VP on the same physical processor as it last ran on, and depending on resource utilization, will broaden its search out to the other processor on the POWER5 chip, then to another chip on the same MCM, then to a chip on another MCM.

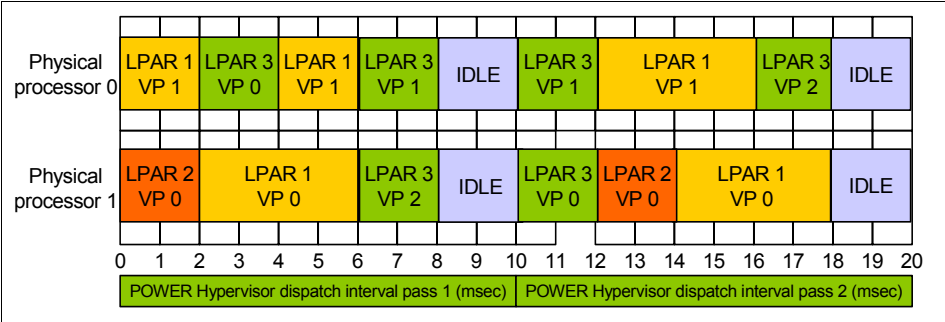


Figure 6-4 Example on Virtual Processor Dispatch

6.2.2 Types of Micro-Partitioning

In the configuration of Micro-partitioning, two modes of operating are available, *capped* and *uncapped*. The principle difference is in defining the ability of a partition to use extra capacity available in the system. As mentioned previously, if a processor donates unused cycles back to the shared pool, or if the system has idle capacity (because there is not enough workload running), the extra cycles may be used by other partitions, depending on their type and configuration.

Capped partitions

A *capped partition* is defined with a hard maximum limit of processing capacity. That means that it cannot go over its defined maximum capacity in any situation, unless you change the configuration for that partition (either by modifying the partition profile or by executing a DLPAR operation). Even if the system is otherwise idle, the capped partition cannot exceed its entitled capacity.

Figure 6-5 on page 137 shows an example where a Micro-Partition is capped at an entitlement of 9.5 (up to the equivalent of 9.5 physical processors). In some moments the processor usage goes up to 100%, and while the machine presents extra capacity not being used, the capped partition cannot use it by design.

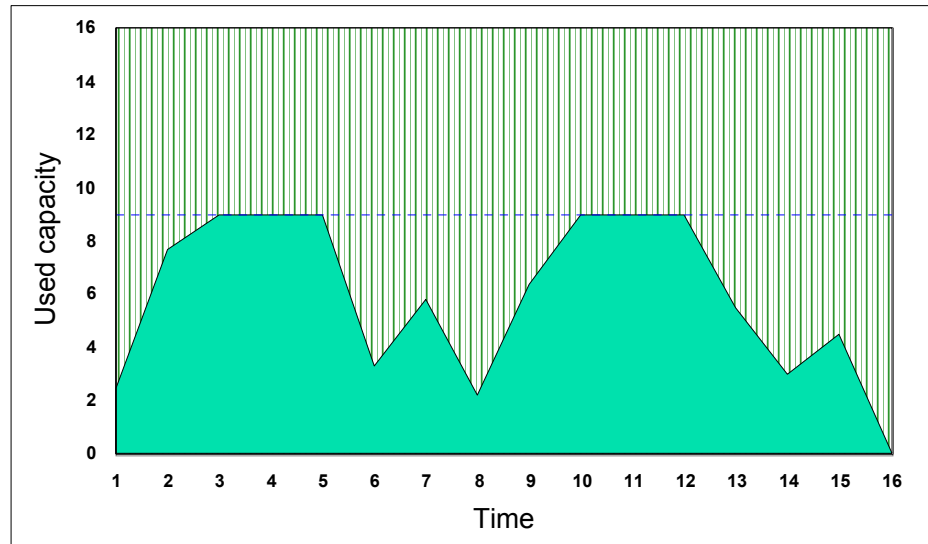


Figure 6-5 Capped partition

Uncapped partitions

An *uncapped partition* has the same definition of a capped partition, except that the maximum limit of processing capacity limit is a soft limit. That means that an uncapped partition may possibly receive more processor cycles than its entitled capacity.

In the case a partition is using 100% of the entitled capacity, and there are idle processor cycles available in the shared processor pool, the Hypervisor has the ability to dispatch virtual processors from the uncapped partitions to use the extra capacity.

In the example we used for the capped partition, if we change the partition from capped to uncapped, a possible chart for the capacity utilization is the one shown in Figure 6-6 on page 138. It still has the equivalent of 9.5 physical processors as its entitlement, but it can use more resources if required and available.

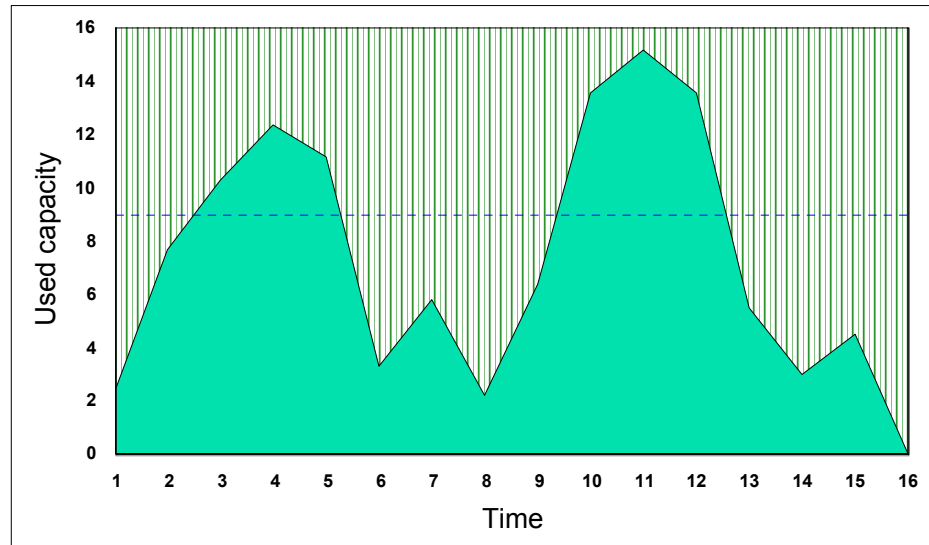


Figure 6-6 *Uncapped partition*

The number of virtual processors in an uncapped partition define the largest capacity it can use from the shared pool. If the amount of virtual processors configured inside the uncapped partition is equal to or more than the number of physical processors in the shared pool, then the uncapped partition can use the entire pool for its processing. Otherwise, the uncapped partition is then limited to a number of physical processors equal to the number of virtual processors. Figure 6-7 on page 139 shows a situation when an uncapped partition has less virtual processors than physical processors. In this example there were 11 virtual processors configured, and an entitlement of 9.5 physical processors. It is using more than its entitled capacity, but is limited by the number of virtual processors configured.

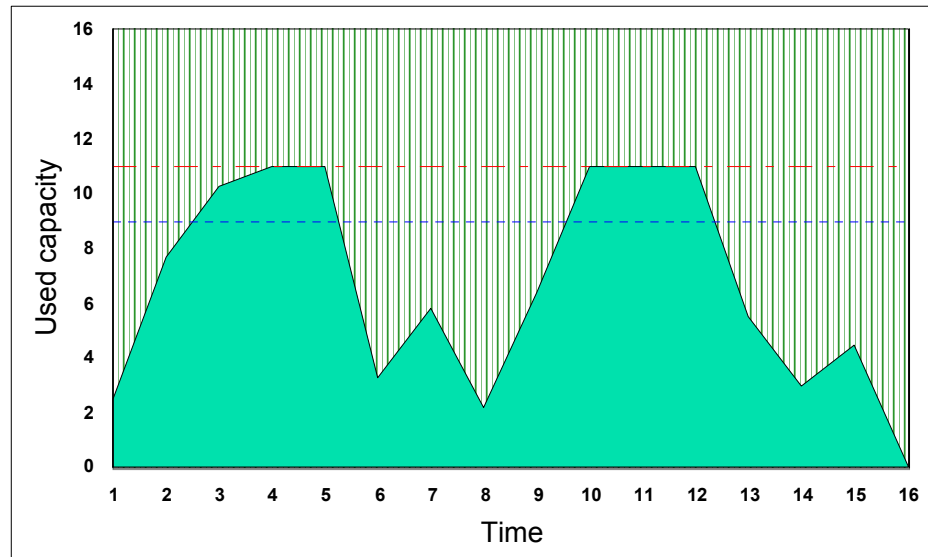


Figure 6-7 Uncapped partition example

Weight for uncapped partitions

You can determine how the Hypervisor should distribute the extra cycles between different uncapped partitions. When configuring an uncapped partition on the HMC, you are presented with an option to set the *variable capacity weight*. It is a number between 0 and 255 that represents the relative share of extra capacity that the partition is eligible to receive. For any uncapped partition, its eligible share is calculated by dividing its own variable capacity weight by the sum of the variable capacity weights for all uncapped partitions

6.2.3 Phantom Interrupt

In order to speed the processing of I/O interrupts, the delivery of interrupts to physical processors can happen without direct execution of the Hypervisor. Rather, the I/O interrupts are delivered by the I/O hardware directly to a physical processor executing a partition's virtual CPU. In the case that an I/O interrupt is delivered to a virtual processor for a partition that does not own the I/O hardware, the interrupt is ignored by the currently executing partition and queued for execution by the correct partition by the Hypervisor. An interrupt which is mistakenly delivered to an incorrect partition is termed a phantom interrupt.

The architecture is redefined to "virtualize" interrupts. The flow of I/O interrupt processing now becomes:

I/O interrupt is delivered to a physical processor:

- ▶ If the processor is idle (running in Hypervisor), then Hypervisor handles the interrupt and identifies the correct partition to make ready to run. The interrupt is queued to be delivered when a virtual processor for the correct partition runs.
- ▶ If the physical processor is running a virtual processor for a partition, the virtual processor receives the I/O interrupt. The operating system running on the virtual processor calls the Hypervisor (via the H_XIRR call) to determine the interrupt source. If the interrupt source is not for this partition, the interrupt is queued in the Hypervisor for delivery to the correct partition. If the interrupt source is for this partition, the correct device driver is invoked.

Figure 6-8 on page 141 shows the interrupt processing on POWER5-based system:

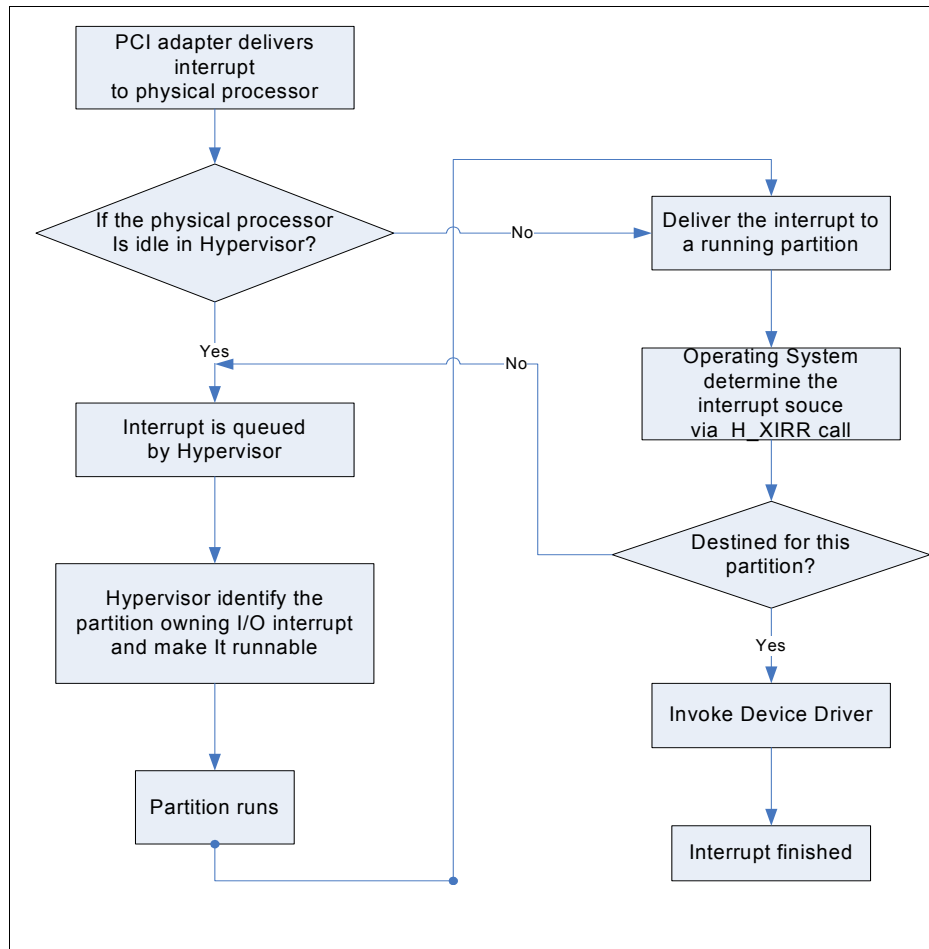


Figure 6-8 Interrupt Processing on POWER5

For dedicated processor partitions, phantom interrupts are extremely rare. This is because the I/O subsystem can be relatively certain of which physical processor the interrupt can be steered to for servicing. In micro-partitioning, phantom interrupts happen with statistical likelihood. The latency for interrupt servicing can become longer, due to partition dispatch. But normally interrupt latency will have an upper bound of the duration of the dispatch wheel.

Under normal and even heavy I/O load, the service time of phantom interrupts is very low. That is because the codepath to process one is small. In order to understand the rate of phantom interrupts, we allow extraction of their rates by the AIX command `mpstat`. The field “ph” shows the number of phantom interrupts for each logical processor. For example:

Example 6-2 mpstat command

(localhost:) # mpstat -i 1 100

System configuration: lcpu=8 ent=1.0

| cpu | mpcs | mpcr | dev | soft | dec | ph |
|-----|------|------|-------|------|-----|-------|
| 0 | 0 | 1 | 1939 | 47 | 183 | 2735 |
| 1 | 0 | 1 | 1946 | 1 | 83 | 2225 |
| 2 | 0 | 1 | 1815 | 1 | 100 | 2912 |
| 3 | 0 | 1 | 1870 | 1 | 89 | 1510 |
| 4 | 0 | 1 | 2000 | 11 | 102 | 3096 |
| 5 | 0 | 1 | 1951 | 1 | 100 | 1715 |
| 6 | 0 | 1 | 2093 | 1 | 112 | 1942 |
| 7 | 7 | 0 | 2101 | 0 | 100 | 2527 |
| ALL | 7 | 7 | 15715 | 63 | 869 | 18662 |

| | | | | | | |
|-----|---|---|-------|----|-----|-------|
| 0 | 0 | 1 | 1767 | 49 | 148 | 4131 |
| 1 | 0 | 1 | 1809 | 1 | 100 | 1843 |
| 2 | 0 | 1 | 1951 | 1 | 101 | 4062 |
| 3 | 0 | 1 | 1974 | 1 | 85 | 2602 |
| 4 | 0 | 1 | 1918 | 11 | 101 | 2264 |
| 5 | 0 | 1 | 1868 | 1 | 101 | 3492 |
| 6 | 0 | 1 | 1980 | 1 | 110 | 5461 |
| 7 | 7 | 0 | 1971 | 0 | 100 | 1841 |
| ALL | 7 | 7 | 15238 | 65 | 846 | 25696 |

In the AIX trace, the phantom interrupts are clearly marked. They start just like a normal I/O interrupt, but are marked as soon as the operating system determines they are phantom interrupts. We can check this from the following sequence of the AIX trace:

Example 6-3 AIX trace for phantom interrupt

| | | | | | | |
|----------|---|------|---|----------|----------|--|
| 100 wait | 0 | 8197 | | 0.340638 | 0.337172 | I/O INTERRUPT iar=2C514 cpuid=00 |
| 492 wait | 0 | 8197 | 1 | 0.340638 | 0.337173 | h_call: start H_XIRR iar=3B6B100 |
| | | | | | | p1=1857D50 p2=234E70 p3=9C6B508A638 |
| 492 wait | 0 | 8197 | 1 | 0.340642 | 0.337177 | h_call: end H_XIRR iar=3B6B100 rc=0000 |
| 47F wait | 0 | 8197 | 1 | 0.340642 | 0.337177 | phantom interrupt cpuid=00 |
| 200 wait | 0 | 8197 | 1 | 0.340643 | 0.337178 | resume wait iar=2C514 cpuid=00 |

6.2.4 Configuring Micro-Partition on HMC

The configuration for a dedicated processor logical partition on POWER5 based server is similar with POWER4 system. There is also an exception that there is

an option of “allow shared processor pool utilization authority” on HMC when changing dedicated processor partition profile on POWER5 servers. That allows the physical processors assigned to the dedicated processor partition can be used by Micro-Partitions when the dedicated processor partition is not activated. Here we will discuss the definition for POWER5 Micro-Partition.

The Hardware Management Console (HMC) provides the user interface for defining Micro-Partition. System administrator need to specify the following partition attributes which are used to define the dimensions and performance characteristics of Micro-Partitions:

- ▶ Minimum, desired, and maximum processor unit
- ▶ Minimum, desired, and maximum number of virtual processors
- ▶ Capped and uncapped
- ▶ Variable capacity weight

For detailed explanation on above terminologies, please refer to the redbook “*Advanced POWER Virtualization on IBM @server p5 Servers Introduction and Basic Configuration*” SG247940.

When a partition is started, the system chooses the partition’s entitled processor capacity from the specified capacity range. The value that is chosen represents a commitment of capacity that is reserved for the partition. This capacity cannot be used to start another Micro-Partition, otherwise capacity could be overcommitted. Preference is given to the desired value, but these values cannot always be used, because there may not be enough unassigned capacity in the system. In that event, a different value is chosen, which must be greater than or equal to the minimum capacity attribute. Otherwise, the partition cannot be started. The assigned amounts of processing units, the number of virtual processors, and the type of partition can be extracted from several AIX commands. For example, `lparstat`, `vmstat`, `mpstat`, etc. The detailed explanation for these commands will be introduced in Section 8.1, “Performance commands” on page 274”

The same basic process applies for selecting the number of online virtual processors with the extra restriction that each virtual processor must be greater than at least 10 processing units of entitlement. In this way, the entitled processor capacity may affect the number of virtual processors that are automatically brought online by the system during boot.

For example, if we define a partition with minimum processing unit as 0.2, and the minimum number of virtual processors is 3 on HMC, the definition will fail with the error “Minimum processing units times 10 must be greater than or equal to the minimum virtual processors”.

There is performance issues associated with the maintenance of online virtual processors, customers should carefully consider their capacity requirements before choosing values for these attributes. In general, the value of the minimum, desired, and maximum virtual processor attributes should parallel those of the minimum, desired, and maximum capacity attributes in some fashion. A special allowance should be for uncapped partitions, since they are allowed to consume more than their entitlement. If the partition is uncapped, then the system administrator may want to define the maximum virtual processor attributes $x\%$ above the corresponding entitlement attributes. The exact percentage is installation specific, but 25%-50% seems like a reasonable number.

The related topics on performance considerations for Micro-Partitioning will be described in Section 6.3, "Performance considerations" on page 145.

6.2.5 Typical usage of Micro-Partitions

With fractional processor allocations, more partitions can be created on a given platform which enables customers to maximize the number of workloads that can be supported on a server simultaneously. Micro-Partitioning enables both optimized use of processing capacity while preserving the isolation between applications provided by separate operating system images.

There are several scenarios where the use of Micro-Partitioning can bring advantages such as optimal resource utilization, rapid deployment of new servers and application isolation:

- | | |
|-----------------------------|---|
| Server consolidation | Consolidating several small systems onto a large and robust server brings advantages in management and performance, usually together with reduced total cost of ownership (TCO). A Micro-Partitioned system enables the consolidation from small and large systems without the burden of dedicating very powerful processors to a small partition. You can divide the processing power between several partitions with the adequate processing capacity for each one. |
| Server provisioning | With Micro-Partitioning and virtual I/O, a new partition can be deployed rapidly, to accommodate unplanned demands, or to be used as a test environment. |
| Virtual server farms | In environments where applications scale with the addition of new servers, the ability to create several partitions sharing processing resources is very useful and contributes to better use of processing resources by the applications deployed on the server farm. |

There are some considerations when implementing Micro-Partition, and a careful planning should be made in order to satisfy the application resource requirements, so that the system can be efficiently utilized with satisfactory performance from the application point of view.

6.3 Performance considerations

The following sections detail some considerations when using Micro-Partitioning, and provides some guidelines when configuring Micro-Partitions.

6.3.1 Micro-Partitioning considerations

Micro-Partitioning adds a layer of abstraction by virtualizing the physical processors onto virtual processors. This virtualization of the processors promotes greater flexibility in using the system and deploying fractional processing power, however there can be some performance issues if not managed properly.

The performance issues may be defined or measured in a number of different ways. For this section, performance issues are defined and measured in the following ways:

- The decrease in maximum throughput for a fixed entitlement due to activity in other partitions. Performance is measured in partitions with high CPU utilization levels.
- The change in processing time used by a partition to complete a fixed task due to activities in other partitions. Performance may be measured at any utilization level.
- The change in processing time measured in a software thread or process to complete a fixed task due to activities in other partitions. Performance may be measured at any utilization level.

With Micro-Partitioning, performance issues tend to be isolated to each partition. Put another way, performance issues can appear as changing amounts of CPU utilization to complete work. This changing time may be due to decreasing hardware efficiency or decreasing software efficiency. In most cases, significant performance issues only occur when multiple partitions are running on the system. There is of course, negligible performance impact when running only a single partition by itself.

For example, in an NFS test, throughput was measured on four dedicated processor partitions, each with one physical processor. The result was then compared to the throughput of 4 Micro-Partitions with 1.0 entitlement per

partition each essentially run on its own physical processor. The throughput in each partition was the same in both cases. Processor usage was about 2% higher in the case of Micro-Partitions. Similar performance was observed in another test conducted running a Java-based application server with Websphere and DB2 in an uncapped partition. In both cases the partitions have been stressed up to 100% of their capacity. Customer production systems generally do not run at 100% capacity, and therefore can expect even less impact on performance. These measurements are indicative of the minimum performance impact associated with the most simplistic Micro-Partitioning environment.

6.3.2 Simultaneous Multithreading and Micro-Partitioning

Simultaneous Multithreading (SMT) is a function implemented by the microprocessor that is exploited in the operating system. Hardware threads are treated by the operating system and applications as distinct processors. SMT behaves the same way either on dedicated processor partitions or Micro-Partitions, with a few differences explained here.

Dynamic switching between SMT and Single Threading

The purpose of SMT is to increase the number of instructions which can execute in a unit of time through the microprocessor. This is accomplished by the two threads executing instructions at the same time. Each thread uses some microprocessor resources to execute instructions. Under almost all circumstances, there are sufficient resources to have more throughput with two threads executing than with a single thread executing. But the execution of two threads at the same time results in the sharing of some microprocessor resources. This implies that the time to execute a fixed number of instructions by a measured thread increases with two threads active. But the sum of the instructions executed by two threads at the same time is greater than those that could be executed by a single thread. If a partition is executing a low to medium CPU utilization, there may not be enough software threads or software processes to keep all of the hardware threads busy. In this case, it is beneficial to be able to apply all of the microprocessor resources to a single thread.

AIX distinguishes the two threads on a microprocessor as a primary thread and a secondary thread. When running in a partition with SMT enabled, AIX dispatches work preferentially to primary hardware threads before the secondary threads. This helps to optimize performance for the case of single threads running on a microprocessor. Since the secondary threads do not get work to execute, they go into a snooze state and the primary thread runs at almost single thread performance.

In dedicated processor partitions, the Hypervisor can dynamically transition the processor from SMT to single-threaded, when requested by the operating

system. In the case a single hardware thread running on a processor becomes idle, the processor is changed to single-threaded mode, and the running thread benefits of full single thread performance. When the other thread is runnable again, the processor returns to SMT mode and run both threads.

This is not supported on Micro-Partitions however, and these must be explicitly changed by the `smtctl` command. On Micro-Partitions, if a hardware thread is idle, it waits on an idle spin, in a low priority. In that case, the running thread gets a large part of the processing capacity to itself.

The effect of SMT on processor usage

As explained in Chapter 5, “Simultaneous Multi-Threading (SMT)” on page 89, SMT enables two hardware threads to run simultaneously. For a processor to cede its idle cycles to the Hypervisor in the case of Micro-Partitioning, both hardware threads must be idle. If one thread is idle while the other is running, some idle capacity remains in the partition and cannot be given back to the Hypervisor. This effect is noted by comparing the CPU utilization of the partition versus the fraction of its entitlement used.

The behavior is more perceptible when partition CPU usage is between 40% and 70% of processing capacity. You can observe this effect by looking at the difference between partition entitlement utilization (that is seen as processing capacity consumed by the partition) and partition processor utilization (that is seen as processing capacity consumed by the threads inside the partition). AIX command `vmstat` shows these features with the output of “ec” and “pc”. Figure 6-9 on page 148 illustrates this effect observed when running a Java-based application server with Websphere and DB2.

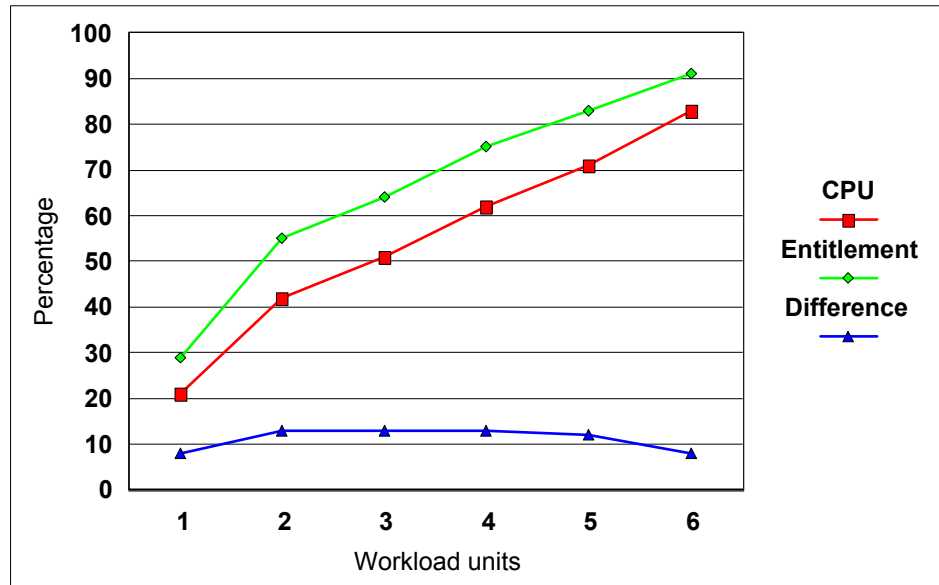


Figure 6-9 Effect of an SMT idle thread on Micro-Partitions

As partition utilization increases, this effect decreases (because the hardware threads get more work to be done, therefore the idle time for each thread decreases). Also, as expected, this effect is not present in partitions running in single-threaded mode.

6.3.3 Cache architecture and number of virtual processors

On POWER5 systems, the two processor cores on the same chip share the L2 and L3 caches. In a system running with dedicated partitions, if the cores are each one on a different partition, the cache is still shared, but the two cores will compete for cache capacity. Naturally each core can only access the cache lines correspondent to its memory addresses. But, the competition for cache capacity has direct impact on the performance each microprocessor can achieve. With Micro-Partitions, the same situation occurs, with the additional factor that during a given interval a physical processor may have executed code from several different partitions. When a virtual processor is dispatched onto a physical processor, all the memory addresses are relative to the partition where the virtual processor belongs. In this sense, cache usage becomes dependant on the memory access behavior of different applications running on different partitions. The competition for shared caches is a significant factor in Micro-Partitioning performance, as the cache hit ratios for a measured partition may change over time as other partitions run at varying levels of activity.

The Hypervisor is responsible for maintaining affinity between virtual and physical resources in a Micro-Partition environment. When dispatching a virtual processor onto a physical processor, the Hypervisor tries to redispach a virtual processor to the same physical processor that it ran on previously, in order to maximize cache affinity and reduce the need of reloading data from main memory. Nevertheless, since in a Micro-Partition environment there is the potential of having several partitions sharing a processor, there are several different memory contexts. Moreover, because of dispatching requirements, a physical processor may not be available when a virtual processor makes the transition from not-runnable to runnable. When a virtual processor is ready to run, the Hypervisor looks to see if the physical processor that ran this virtual processor for the last time is idle. If it is busy, then it starts looking in increasing levels of affinity scope for an idle processor (other cores on same chip, other processors within same MCM, and any other processor in the system) until one is found. If no processor is available, the virtual processor is enqueued onto the runnable queue. Figure 6-10 depicts the flow of actions described.

Even in the case when the virtual processor is dispatched on the same physical processor from its last run, data in cache may have been replaced by previous virtual processors dispatched in the same physical processor. The amount of cache context left over depends on the amount of data read from other applications running on the same processor, and also the ratio of virtual processors to physical processors. If an application running in a virtual processor does heavy memory access, data that was stored in cache from other virtual processors running before is replaced, and cache is refreshed when other virtual processors are later dispatched. Therefore, an application whose performance depends on cache efficiency will be affected when running in a Micro-Partition along with other partitions that do intensive memory access.

High Performance Computing (HPC) applications are more likely to have intense memory access behavior.

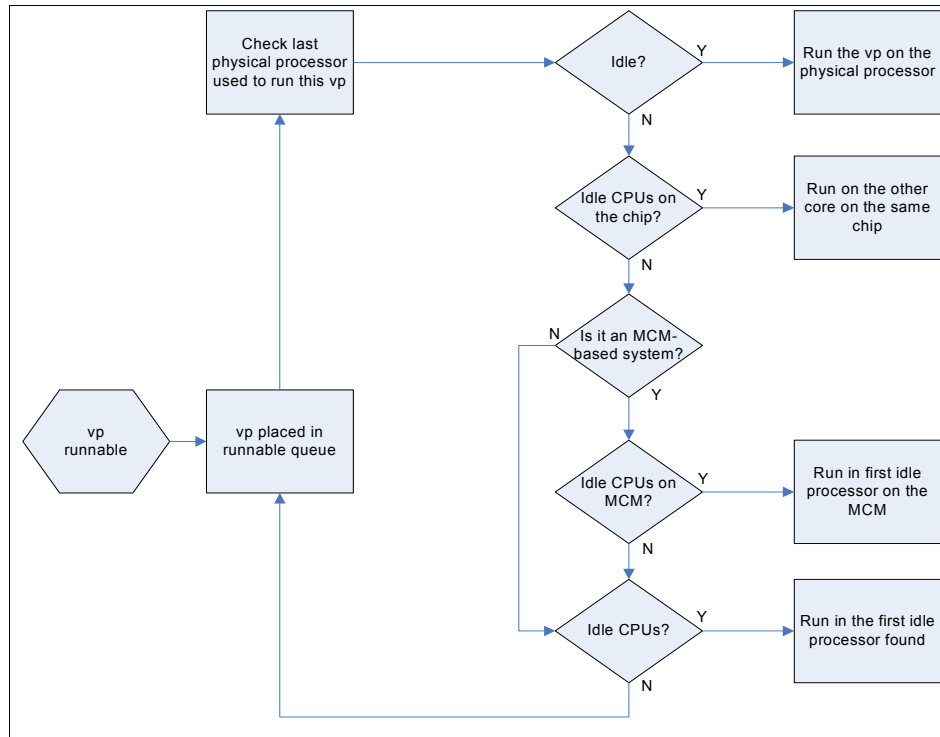


Figure 6-10 Affinity between virtual and physical processors

Number of virtual processors

When the number of virtual processors is much larger than the number of physical processors on the system, the time slice given to each virtual processor on the physical processors tends to get smaller. One way to view the size of the virtual processor time slice is to divide the partition entitlement by the number of virtual processors. Having effectively small virtual processor time slices contributes to diminishing efficiency in cache, since for each virtual processor that is dispatched, the memory context changes, data and code it needs will more likely need to be reloaded from memory.

When virtual processor capacity is small, the performance impact of reading data from memory is significantly high, due to the fact that the time to fetch data from memory is constant, and the time slice is small for small capacity entitlements. Therefore, the impact is more significant in virtual processors with small capacity.

Figure 6-11 on page 151 shows an example with two partitions, each with one virtual processor and same entitlement capacity, running on a system with one physical processor. Partitions A and B use the physical processor 5 milliseconds

each time slice. Even if data is loaded from memory into the cache, it remains in the same memory context for the duration of the time slice. Moreover, there is just one other partition with other memory context running on this processor. Partitions C, D, E, F and G are also running each with one virtual processor and same entitlement capacity. In this case each partition gets only a 2 millisecond time slice. Not only is the time to fetch data is more significant in this case (relatively to the time slice), but there are now five partitions with different memory context. It is much more likely that in this case the cache context becomes completely lost in the intervals between time slices.

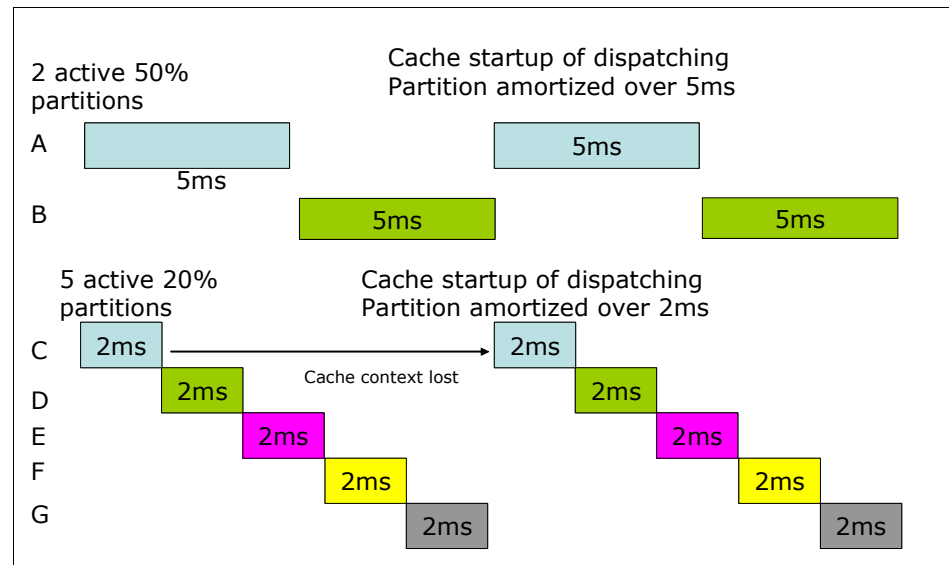


Figure 6-11 Impact on cache due to number of virtual processors

Effects of cache-friendly and cache-unfriendly applications

The performance impact of increasing cache miss rates in the partition performance due to interference from other partitions depends on the size of the partition, number of virtual processors, the nature of the other partitions, and type of application. A worst-case scenario is a partition that uses the cache moderately running on the same physical processor and another partition that uses memory and cache intensely on the same processor. This represents an example where a benchmark application A (composed by small but numerous tasks, involving process creation and termination) fits well in the cache by itself while application B which uses memory heavily for reading and writing large blocks of data. In all cases, each partition has two virtual processors, each with 0.1 processor capacity. The partition running benchmark A is uncapped, while the partitions running benchmark B are capped.

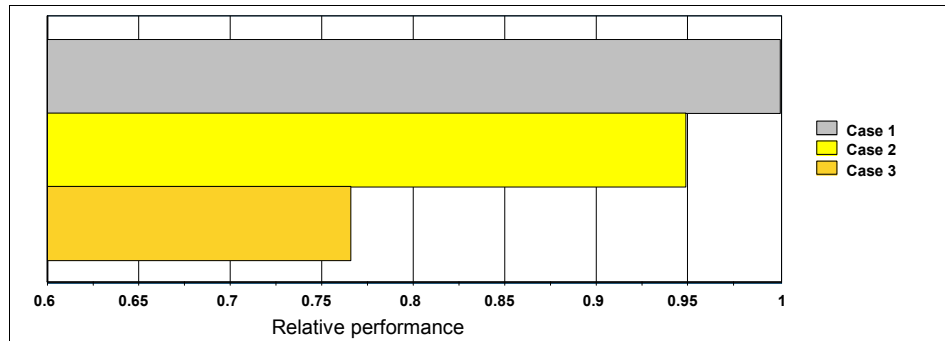


Figure 6-12 Measurements of cache effects in different partitions

Case 1 shows the throughput for the reference benchmark when A runs without other partitions running at the same time, and serves as the reference point. Case 2 shows the throughput for the reference benchmark when A runs in one partition, and one other partition runs benchmark B. Even with the effects of benchmark B reducing cache efficiency, the application running benchmark A does run well, with slightly more than 5% penalty for sharing the same physical processor with benchmark B. Case 3 shows the throughput for the reference benchmark when A runs on a partition, and seven other partitions run benchmark B. This case is much more destructive to the cache efficiency, since there are seven different partitions running memory intensive workloads, and also a large number of virtual processors. This case shows more performance impact on benchmark A, due to reduction in cache efficiency. As commented before, this is an extreme case where the workloads were selected so that the effect on cache usage would have the most impact in performance. Most applications used on UNIX systems, including commercial and technical workloads should observe a smaller impact in performance.

In addition to the changes in maximum throughput achieved by a partition for a fixed entitlement, changes in the CPU time to perform a task will occur. For example, consider a case where a partition requires 100 CPU seconds to complete a database sort when it runs on a processor alone when the other partitions on the system are relatively idle. Then the exact same sort is run at another time when a number of other partitions are active. Now, the task takes more CPU time, perhaps 130 seconds. Because Micro-Partitioning increases the possibility to run variability of CPU time, it is important to understand this phenomenon in environments where CPU billing is used.

6.3.4 SMP locking and number of virtual processors

When using Micro-Partitions, virtual processors are dispatched onto physical processors by the Hypervisor. The Hypervisor guarantees that every virtual

processor that needs to run is dispatched on average of 10 millisecond intervals. It does not guarantee, however, that the individual virtual processors in a single partition are dispatched simultaneously, or always run at the same time. Most operating systems and sophisticated applications use spin locks to serialize read/write access to shared memory. The effectiveness of spin locks is based on the notion that the locks are not held for long periods of time. But, when running with Micro-Partitioning, it is possible for a virtual processor holding a lock to be undispached for milliseconds. This increases the likelihood of lock contention when a partition is spread over a number of virtual processors.

AIX kernel locks that run with interrupts disabled benefit from special handling in Micro-Partitions. They are handled differently from locks which run with interrupts enabled, since having interrupts disabled prohibits simply undispaching the blocked software thread and running another. Consider the case that a virtual processor owning the lock is not running (for example it used up the entitled time slice), and there is other virtual processors which need the same lock run on the system concurrently. Without optimization, a blocked virtual processors will go into a spin lock wait until the lock is released by the owner. To effectively solve this situation without spending unnecessary cycles, the virtual processor waiting for lock will call the H_CONFER Hypervisor service to give its cycles to the virtual processor owning the lock, then it is undispached by the Hypervisor and the lock owner get dispatched to continue processing and release the lock. It is worth noting that SMT makes this mechanism relatively less effective. If there is heavy locking, running a partition in single-threaded mode may reduce the impact.

Lock contention can be tracked from AIX trace also. The following is an example which shows a case that virtual processor 0 and 1 attempt to acquire the same lock, and identifying that the lock is held by a software thread running on another virtual processor that is currently undispached, so they conferred the processor cycles they owned to the processor which the software thread is running on. After the lock is released by the software thread, they acquire the lock.

1. Both virtual processor 0 and 1 attempt to acquire the lock:

Example 6-4 AIX trace on lock contention - step 1

| ID | PROCESS NAME | CPU | PID | TID | I | SYSTEM CALL | ELAPSED_SEC | DELTA_MSEC | APPL | SYS CALL | KERNEL | INTERRUPT |
|-----|--------------|-----|--------|---------|---|-------------|-------------|------------|------|----------|----------------------------|-----------|
| 112 | -229498- | 1 | 229498 | 1294461 | | | 0.033375354 | 0.000376 | | lock: | dmiss lock | |
| | | | | | | | | | | | addr=F1000600234F0100 | lock |
| | | | | | | | | | | | status=B7060000000000 | |
| | | | | | | | | | | | requested_mode=LOCK_SWRITE | |
| | | | | | | | | | | | eturn addr=3CCF1EC | |
| | | | | | | | | | | | name=00000000.00000000 | |
| 112 | -229498- | 0 | 229498 | 700529 | | | 0.033375445 | 0.000091 | | lock: | dmiss lock | |
| | | | | | | | | | | | addr=F1000600234F0100 | lock |
| | | | | | | | | | | | status=B7060000000000 | |
| | | | | | | | | | | | requested_mode=LOCK_SWRITE | |
| | | | | | | | | | | | return addr=3CCF1EC | |
| | | | | | | | | | | | name=00000000.00000000 | |

2. After identifying lock contention, the lock transforms to a krlock:

Example 6-5 AIX trace on lock contention - step 2

| | | | | | | | |
|-----|----------|---|--------|---------|-------------|----------|--|
| 112 | -229498- | 1 | 229498 | 1294461 | 0.033376208 | 0.000763 | krlock: cpuid=01 addr=F100060004006B80 action=spin |
| 112 | -229498- | 0 | 229498 | 700529 | 0.033376227 | 0.000019 | krlock: cpuid=00 addr=F100060004006B80 action=spin |

3. The software thread that needs the lock next, because it is a hand-off lock, is on virtual CPU 2. Both virtual CPU 0 and 1 need this virtual CPU to run to process the lock, so they confer execution to that virtual CPU:

Example 6-6 AIX trace on lock contention - step 3

| | | | | | | | |
|-----|----------|---|--------|---------|-------------|----------|--|
| 112 | -229498- | 1 | 229498 | 1294461 | 0.033382927 | 0.006700 | krlock: cpuid=01 addr=F100060004006B80 action=confer (target cpuid=0002) |
| 112 | -229498- | 0 | 229498 | 700529 | 0.033382946 | 0.000019 | krlock: cpuid=00 addr=F100060004006B80 action=confer (target cpuid=0002) |

4. The Confer Hypervisor calls (h_call) are traced:

Example 6-7 AIX trace on lock contention - step 4

| | | | | | | | |
|-----|----------|---|--------|---------|-------------|----------|--|
| 492 | -229498- | 1 | 229498 | 1294461 | 0.033383293 | 0.000347 | h_call: start H_CONFER iar=17A8F0 p1=0002 p2=52904FF p3=2D33E3F7FA40 |
| 492 | -229498- | 0 | 229498 | 700529 | 0.033383307 | 0.000014 | h_call: start H_CONFER iar=17A8F0 p1=0002 p2=52904FF p3=2D33E3F7FA44 |

5. Since both virtual CPU's 0 and 1 have gracefully conferred, virtual CPU 2 and its partner CPU 3 can begin running immediately:

Example 6-8 AIX trace on lock contention - step 5

| | | | | | | | |
|-----|----------|---|--------|--------|-------------|----------|---|
| 419 | -229498- | 2 | 229498 | 819359 | 0.033598712 | 0.215405 | cpu preemption data Preempted vProcIndex=0004 rtrdelta=0000 enqdelta=4471D exdelta=A668 start wait=2D33E3F3B790 end wait=2D33E3F8A518 SRR0=000000000017B770 SRR1=8000000000009032 |
|-----|----------|---|--------|--------|-------------|----------|---|

6. Virtual CPU 2 was waiting on the lock and immediately acquires it:

Example 6-9 AIX trace on lock contention - step 6

| | | | | | | | |
|-----|----------|---|--------|--------|-------------|----------|--|
| 112 | -229498- | 2 | 229498 | 819359 | 0.033599436 | 0.000724 | krlock: cpuid=02 addr=F100060004006B80 action=acquire |
| 112 | -229498- | 2 | 229498 | 819359 | 0.033600580 | 0.001144 | lock: dlock lock addr=F1000600234F0100 lock status=B70800000C809F requested_mode=LOCK_SWRITE return addr=3CCF1EC name=00000000.00000000 |
| 112 | -229498- | 2 | 229498 | 819359 | 0.033601444 | 0.000864 | krlock: cpuid=02 addr=F100060004006B80 action=handoff (target cpuid=0003) |
| 419 | -229498- | 3 | 229498 | 876731 | 0.033602912 | 0.001468 | cpu preemption data Preempted vProcIndex=0005 rtrdelta=0000 enqdelta=446AF exdelta=A670 start wait=2D33E3F3B7FE end wait=2D33E3F8A51D SRR0=000000000017B74C SRR1=8000000000009032 |

7. Virtual CPU's 2 and 3 continue their processing

Example 6-10 AIX trace on lock contention - step 7

| | | | | | | | |
|-----|----------|---|--------|--------|-------------|----------|--|
| 254 | -229498- | 2 | 229498 | 819359 | 0.033603515 | 0.000603 | MBUF m_copydata mbuf=F100061008250C00 offset=0 len=26 cpaddr=F100061001480000 |
| 112 | -229498- | 2 | 229498 | 876731 | 0.033603524 | 0.000009 | krlock: cpuid=02 addr=F100060004006B80 action=acquire |
| 254 | -229498- | 3 | 229498 | 819359 | 0.033604287 | 0.000763 | MBUF return from m_copydata |

| | | | | | | | |
|-----|----------|---|--------|--------|-------------|----------|---|
| 254 | -229498- | 2 | 229498 | 819359 | 0.033604697 | 0.000410 | MBUF m_copydata mbuf=F100061008250C00 offset=26 len=8 |
| | | | | | | | cpaddr=F10006100148001A |
| 254 | -229498- | 2 | 229498 | 819359 | 0.033605015 | 0.000318 | MBUF return from m_copydata |

8. On the next pass of the Hypervisor dispatch wheel, virtual CPU's 0 and 1 are dispatched to run again. Note that approximately 7 milliseconds of time have passed:

Example 6-11 AIX trace on lock contention - step 8

| | | | | | | | |
|-----|----------|---|--------|--------|-------------|----------|---|
| 419 | -229498- | 1 | 229498 | 700529 | 0.040380275 | .069589 | cpu preemption data Unblocked vProcIndex=0007 |
| | | | | | | | rtrdelta=AA7D enqdelta=12875A exdelta=2E448 |
| | | | | | | | start wait=2D33E3F7FC15 end wait=2D33E40E1234 |
| | | | | | | | SRR0=00000000001EB274 SRR1=8000000000009032 |
| 492 | -229498- | 1 | 229498 | 700529 | 0.040381096 | 0.000821 | h_call: end H_CONFER iar=17A8F0 rc=0000 |

9. Virtual CPU 1 resumes executing in klock:

Example 6-12 AIX trace on lock contention - step 9

| | | | | | | | |
|-----|----------|---|--------|---------|-------------|----------|---|
| 112 | -229498- | 1 | 229498 | 700529 | 0.040381738 | 0.000642 | krlock: cpuid=01 addr=F100060004006B80 action=acquire |
| 419 | -229498- | 0 | 229498 | 1294461 | 0.040382945 | 0.001207 | cpu preemption data Unblocked vProcIndex=0006 |
| | | | | | | | rtrdelta=AA2E enqdelta=12882B exdelta=2E447 |
| | | | | | | | start wait=2D33E3F7FB93 end wait=2D33E40E1233 |
| | | | | | | | SRR0=00000000001EB274 SRR1=8000000000009032 |

10. Virtual CPU 1 gets the lock:

Example 6-13 AIX trace on lock contention - step 10

| | | | | | | | |
|-----|----------|---|--------|--------|-------------|----------|--|
| 112 | -229498- | 1 | 229498 | 700529 | 0.040383089 | 0.000144 | lock: dlock lock addr=F1000600234F0100 lock |
| | | | | | | | status=B70800000AB071 requested_mode=LOCK_SWRITE |
| | | | | | | | return addr=3CCF1EC name=00000000.00000000 |

11. Virtual CPU 0 takes a clock interrupt:

Example 6-14 AIX trace on lock contention - step 11

| | | | | | | | |
|-----|----------|---|--------|---------|-------------|----------|---|
| 100 | -229498- | 0 | 229498 | 1294461 | 0.040383947 | 0.000342 | DECREMENTER INTERRUPT iar=1EB274 cpuid=00 |
|-----|----------|---|--------|---------|-------------|----------|---|

Since lock contention is statistical, reducing the number of virtual processors in a partition will usually decrease lock contention just as increasing the number of virtual processors in a partition will usually increase lock contention. Environments that have responsiveness issues without full utilization of entitled capacity should be evaluated for possible lock contention issues. AIX kernel lock contention can be analyzed with the use of the **curt** tool.

There are two types of virtual processor context switches, voluntary and involuntary. Context switches initiated by H_CEDE, H_CONFER, and H_PROD Hypervisor calls are represented as voluntary context switch, while time slice related context switches are involuntary. The number of voluntary and involuntary

context switches can be extracted from the output field “vlcs” and “ilcs” by AIX command **mpstat**.

The number of virtual processor context switches is important since it is one that measures the performance impact of the Hypervisor. In some of the cases it is best to minimize the number of virtual processors in each partition, if there are lots of partitions. On the other hand, if more virtual processors are needed to satisfy peak load conditions and the capacity requirements vary greatly over time, it may be best to vary virtual processors offline when they are not needed. In such situation, the Partition Load Manager (PLM) may be used to automate this process as a function of load. The detailed explanation on PLM will be introduced in Chapter 9, “Partition Load Manager (PLM)” on page 329.

The context switch statistics and the number of hcalls can also be extracted from AIX high level commands such as **lpatstat** and **mpstat**. The detailed explanation for the commands will be discussed in Section 8.1, “Performance commands” on page 274. The following is an example for **lparstat** which shows the name of hcalls and its elapsed execution time:

Example 6-15 lparstat command

```
# lparstat -H 1 1
```

System configuration: type=Shared mode=Uncapped smt=0n lcpu=4 mem=256 ent=0.20

Detailed information on Hypervisor Calls

| Hypervisor Call | Number of Calls | %Total Time Spent | %Hypervisor Time Spent | Avg Call Time(ns) | Max Call Time(ns) |
|-----------------|-----------------|-------------------|------------------------|-------------------|-------------------|
| remove | 0 | 0.0 | 0.0 | 1 | 709 |
| read | 0 | 0.0 | 0.0 | 1 | 376 |
| nclear_mod | 0 | 0.0 | 0.0 | 1 | 0 |
| page_init | 4 | 0.0 | 0.1 | 655 | 1951 |
| clear_ref | 0 | 0.0 | 0.0 | 1 | 0 |
| protect | 0 | 0.0 | 0.0 | 1 | 0 |
| put_tce | 0 | 0.0 | 0.0 | 1 | 1671 |
| xirr | 6 | 0.0 | 0.1 | 638 | 1077 |
| eoi | 6 | 0.0 | 0.1 | 447 | 690 |
| ipi | 0 | 0.0 | 0.0 | 1 | 0 |
| cprr | 6 | 0.0 | 0.1 | 265 | 400 |
| asr | 0 | 0.0 | 0.0 | 1 | 0 |
| others | 0 | 0.0 | 0.0 | 1 | 0 |
| enter | 4 | 0.0 | 0.0 | 272 | 763 |
| cede | 357 | 1.3 | 98.4 | 7106 | 641022 |
| migrate_dma | 0 | 0.0 | 0.0 | 1 | 0 |
| put_rtce | 0 | 0.0 | 0.0 | 1 | 0 |
| confer | 0 | 0.0 | 0.0 | 1 | 0 |

| | | | | | |
|---------------------|----|-----|-----|------|------|
| prod | 55 | 0.0 | 0.8 | 391 | 1168 |
| get_ppp | 1 | 0.0 | 0.1 | 1738 | 2482 |
| set_ppp | 0 | 0.0 | 0.0 | 1 | 0 |
| purrr | 0 | 0.0 | 0.0 | 1 | 0 |
| pic | 1 | 0.0 | 0.0 | 260 | 656 |
| bulk_remove | 0 | 0.0 | 0.0 | 1 | 0 |
| send_crq | 0 | 0.0 | 0.0 | 1 | 2395 |
| copy_rdma | 0 | 0.0 | 0.0 | 1 | 0 |
| get_tce | 0 | 0.0 | 0.0 | 1 | 0 |
| send_logical_lan | 1 | 0.0 | 0.1 | 2685 | 4602 |
| add_logical_lan_buf | 6 | 0.0 | 0.2 | 686 | 859 |

From operating system point of view, there are software context switches to make a different thread execute. AIX command **vmstat** can be used to check context switch from operating system layer.

Since application locks do not invoke support from the AIX operating system, they are not optimized in the same way as AIX disabled locks. Also, it is not possible to directly track application lock contention through AIX tool.

The example shown in Figure 6-13 on page 158 represents the relative performance of various configurations when executing an NFS benchmark. It shows both the SMP scaling effect and the performance considerations when running several virtual processors. When the configuration changes from a 4-way dedicated processor SMP partition to four 1-way dedicated processor partitions, aggregate throughput is increased by a small margin. This is due to both decreased data movement between CPUs and locking. The next step, four Micro-Partitions each with two virtual processors sees a reduction in performance over four dedicated processor partitions. The next step compares two Micro-Partitions, each with four virtual processors. Having four virtual processors increases the lock contention in the test. Finally, we have four partitions each with four virtual processors. This case has the lowest performance, due to increased cache contentions and locking necessary to provide serialization and protection against race conditions. As we increase the number of virtual processors, the relative performance is impacted more because of the SMP scaling inside the partition and also because of cache contention due to dispatching of multiple virtual processors in the system.

For these reasons, it is recommended to have as few virtual processors configured as possible for each partition. It is better to have few virtual processors with higher capacity than a large amount of virtual processors each with a small amount of processing power. If it is necessary for expanding the partition to handle more workload, you can add more virtual processors by executing a dynamic LPAR operation.

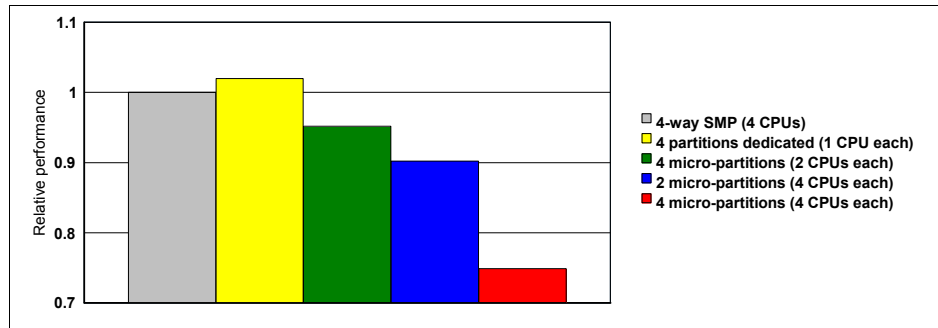


Figure 6-13 The effect of multiple virtual processors in overall performance

6.3.5 Memory affinity considerations

In the POWER5-based servers, memory is attached to processor modules and it has the same access characteristics for any processor within the module. This does not differ from the POWER4-based servers. Memory and processors directly connected are said to fall within a single affinity domain. A processor can access memory attached to its local memory domain faster (that is, lower latency) than it can access memory attached to other memory domains. AIX 5L has optional support for organizing its memory management strategies around these affinity domains.

With memory affinity support enabled, AIX attempts to satisfy page faults from the memory closest to the processor that generated the page fault. This is of benefit to the application because it is now accessing memory that is local to the MCM rather than memory scattered among different affinity domains. This is true for dedicated processor partitions. When using Micro-Partitions, however, virtual processors may be dispatched on different physical processors during the time a partition is running. Therefore, there is no way to implement affinity domains, and therefore memory affinity has no meaning in a Micro-Partition. Memory is allocated to partitions in a round-robin fashion, and this tends to reduce processor time consumption variability due to variation in memory allocation. High-bandwidth applications are the type of applications that benefit from memory affinity and should not typically be run in Micro-Partitions.

6.3.6 Idle partition performance impact

In a system running Micro-Partitions, the Hypervisor manages virtual processor dispatching between different partitions so that each partition gets the deserved processing entitlement. In the case of partitions running in the system, but in idle state (no work being done), the unused processing cycles may be dispatched to other partitions by the Hypervisor, leading to a more efficient usage of the CPU

resources. There are some activities that consume processor resources even when the partition is idle. Clock interrupts, hardware interrupts, daemons polling for events are some examples of such activities that use processing resources. Because of these activities, an idle partition still presents some load to the physical processor. Moreover, the Hypervisor also needs some processing resources to manage these idle partitions and the virtual processors running on them. Normally, a system is not expected to have a large number of idle virtual processors (if there are many, you should analyze whether they are really needed for the work that has to be done). AIX version 5.3 implements some timer-management functions to minimize resource consumption by the idle partitions. The base CPU performance impact of an idle partition should be less than 1% of a CPU. Figure 6-14 shows the impact of adding idle partitions to a system running a workload in one uncapped partition.

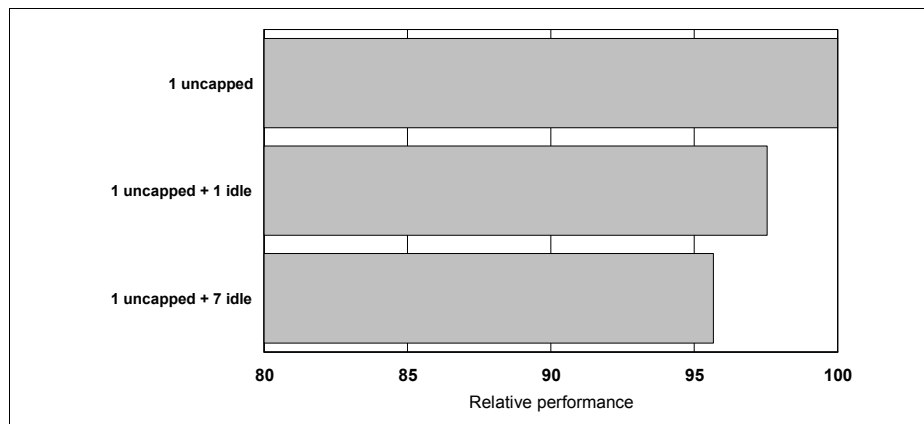


Figure 6-14 Uncapped partition performance example

Since idle partitions are not doing any productive work, in order to reduce further the performance impact associated to having idle partitions in the system, AIX version 5.3 introduces the idea of *slow ticks*. Slow ticks is an operation mode for idle processors where the timer tick rate gets reduced by an order of magnitude. In other words, the busy processors run normally, taking 100 timer ticks per second, while the idle processors go into slow tick mode, and take 10 ticks a second. Slow ticks are enabled in partitions running independently as a function of load average on each processor of a system. Note that daemons that run periodically for polling activities, or applications that present similar behavior can prevent the operation mode to change to slow ticks (since there are threads running periodically, and therefore the partition is not technically idle).

6.3.7 Application considerations for Micro-Partitions

Applications do not need to be aware of Micro-Partitioning, since it is completely transparent from the application perspective. There are however some considerations that should guide a decision on which applications are suitable for Micro-Partitions, and which are not.

Applications with response time requirements

The Micro-Partition environment is a dynamic, especially when capped and uncapped partitions are running on the same system.

As stated in Section 6.2, “Micro-Partitioning Implementation” on page 128, the Hypervisor attempts to dispatch all virtual processors in an interval of 10 milliseconds. It does not guarantee, however, that the elapsed time between one dispatch and the next one is fixed. Virtual processors can therefore be dispatched anytime between immediately (smallest latency) and 18 ms (largest latency) after the last dispatch, based on the virtual processor configured capacity, and the number of virtual processors in the shared pool. Figure 6-15 illustrates the case for the smallest capacity (10% of a physical processor), where the time slice is 1 milliseconds

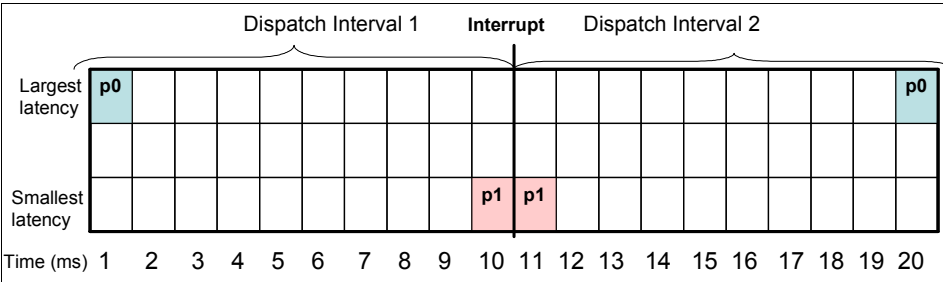


Figure 6-15 Dispatch latencies for virtual processors

Applications that have strong response time criteria for transactions may also not be good candidates for small Micro-Partitions. You can configure the processing capacity inside a partition using different ways, with different numbers of virtual processors, depending on the specific needs for the partition. If an application depends on the individual processing capacity of a processor to run efficiently, it likely observe higher response times when running on a partition with smaller (but more) virtual processors. Therefore, care must be taken when configuring a Micro-Partition to run response time critical workloads, in order to meet quality of service requirements. For planning purposes, if you decide to deploy applications that must have predictable response times, or applications that have transactions whose individual performance is a performance factor, you should consider

configuring the partition with extra capacity, in order to compensate for these effects.

Applications running in Micro-Partitions, like those running in dedicated processor partitions, see their response times as a function of the CPU utilization. Thus, if an application is run in a Micro-Partition and the CPU utilization within the Micro-Partition becomes very high, response time will degrade. The problem is magnified for small virtual processors, since each virtual processor is logically a slow CPU. In laboratory tests, it is frequently difficult to drive small virtual processor Micro-Partitions to high utilizations on heavy CPU transactions with acceptable quality of service. Applications without strong quality of service requirements are good candidates for small Micro-Partitions.

Applications that present polling behavior

Applications that rely on polling to execute their processing may not be good candidates for Micro-Partitions. Since they need to periodically poll to detect if the resource is available or condition is satisfied, they spend cycles that otherwise would be available for other partitions (since they are not actually doing work). If the application needs to periodically wake up a thread to do the polling, that means that a virtual processor must be dispatched to run that thread, and spend physical processor cycles, even if it is not producing work. This behavior is the same regardless of the application being run on a partitioned server or not. What makes a difference is that in the Micro-Partition environment you can use the spare cycles for other partitions. In other words, processor cycles are being used without doing real work, and could be better used by other partitions.

Applications with low average utilization and high peaks

Applications where average usage of processor resources is low, but have peaks of usage during a short period of time are good candidates for a Micro-Partition environment. More than one application can share the processor resources and run with the required performance, exploiting the benefits of sharing otherwise unused resources. Applications that perform online transaction processing (OLTP) generally fit into this category, since they are based on user input, and the load generated by users accessing a system tends to follow a pattern based on the user day of work. Usually there are distinct but independent peaks of utilization, and an average use significantly lower than the peaks. Examples of such applications are ERP systems, mail servers, web-based applications and directory servers.

Figure 6-16 on page 162 shows the user distribution for an ERP system, during the day, on a real customer scenario. You can clearly identify the peak times, and the periods of few user counts.

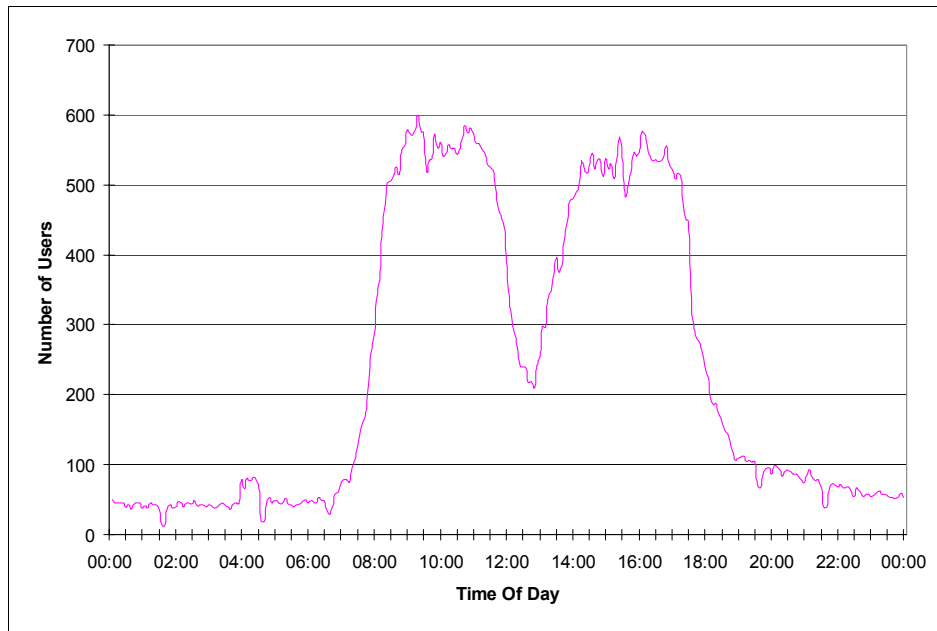


Figure 6-16 User distribution during the day in an ERP application server

For OLTP applications, the processor usage usually follows a similar distribution, as shown on Figure 6-17 for the same system.

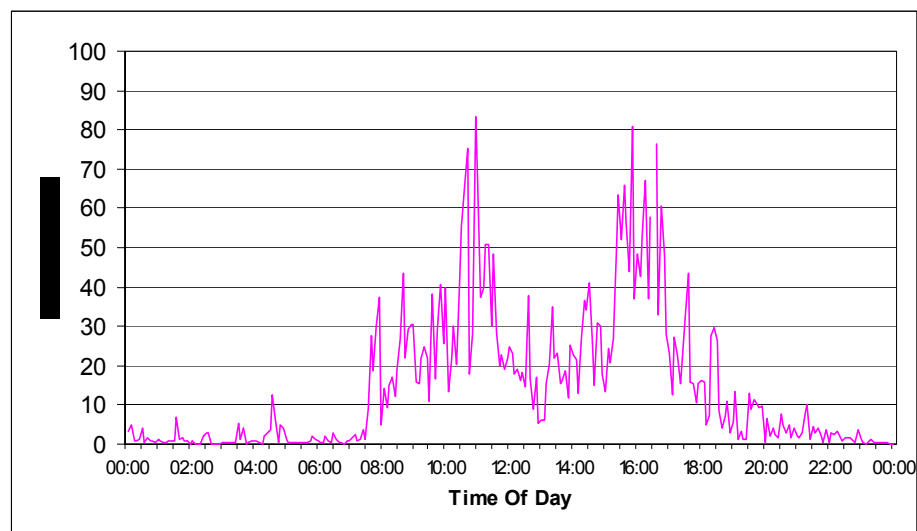


Figure 6-17 Processor utilization by the ERP application server

The same behavior is seen on mail servers usually. An analysis of a Lotus Domino server rendered a similar shape for number of users and processor usage.

If you have several workloads that have peak activity on different times, you can have each one running on a separate partition, and all partitions sharing the same physical processors. By adjusting each partition entitlement, and the partition mode (capped or uncapped), you can run the system at a higher average utilization, while fulfilling the processing requirements for each application.

Figure 6-18 illustrates a typical scenario where different applications are running on Micro-Partitions, with different peak times, and a mixed of capped and uncapped partitions. The system is running with four physical processors, virtualized into 20 virtual processors distributed between five partitions. Three partitions run OLTP types of applications, and two partitions run batch processing.

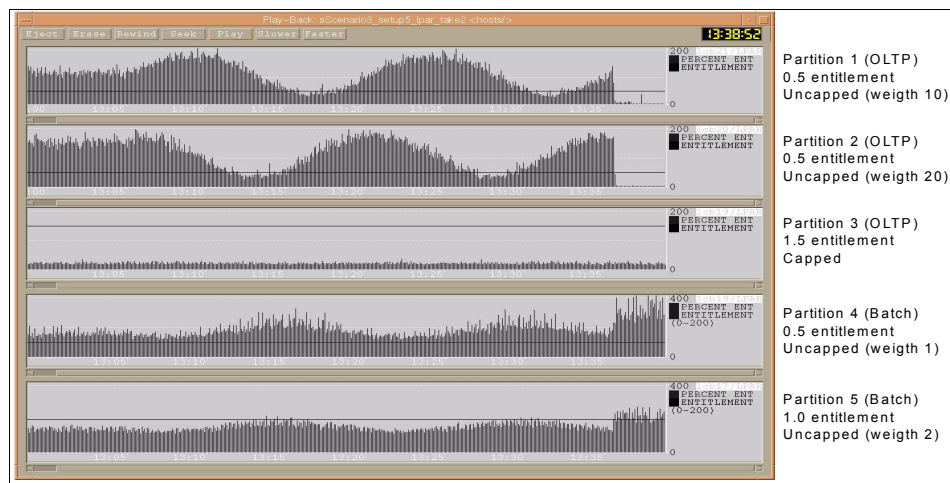


Figure 6-18 Processor utilization between five partitions with different workloads

From this chart we can see that partitions 1 and 2 have peak utilization at different times. Therefore, there is no need to duplicate the amount of resources to satisfy both partitions at peak processing. Partition 3 is capped and at a low utilization, so it remains constant during the time, and cedes the extra cycles not needed to other partition. Partitions 4 and 5 also benefit from the shared resources, receiving extra cycles whenever there are idle processors. And because of the nature of the applications (online and batch), the partition weight is a key factor to allocate the extra cycles to the uncapped partitions.

Application with a high processing capacity usage

If an application uses most of its entitled processing capacity during its execution, for the cases that the entitled processing capacity owns nearly the whole physical processor capacity, it may not be useful to put it into a shared processor pool. Since the requirements for the application are high, and constant during execution, a dedicated processor partition is a better choice for this application. In a dedicated processor partition it will receive the processing capacity it needs, and it is less suitable of interference from other workloads running in the system.

If the partition's entitled processing capacity does not own the most of the physical processor's capacity, it will get more benefit to run these applications in an uncapped Micro-Partition in case they can use the extra cycles that may eventually be available. In this case, the application can execute more work on a system that would otherwise be idle. That would be the case when running online applications in a system during daytime, and batch applications at night.

Typical applications falling in this scenario are decision support systems (DSS) and High Performance Computing (HPC) applications.

6.3.8 Guidelines for planning Micro-Partitions

When planning partitions to run applications using Micro-Partitioning, it is important to identify the application requirements and behavior, in order to properly size the partitions and maximize the system performance.

Planning for future applications is normally a case where estimates are the only information available. In these cases, the usage of Micro-Partitions can help greatly, since partitions can be adjusted for required capacities in a very flexible way. On the other hand, an estimate can always be larger than the actual requirements, or smaller. Because of this, you must always consider some buffer in the system to accommodate extra requirements.

When the application environments are already in production or test, the task of planning a shared processor partition becomes more close to reality. You can measure the resource consumption by the application on the running system, and use this as a base for a shared processor partition performance requirement. Based on the detailed information you are able to get, you can plan the shared processor partitions to make the most effective use of the physical resources.

When planning for Micro-Partitioning, there are three main strategies for defining configurations:

| | |
|----------------------------|---|
| Guaranteed Capacity | Basic definition, based on the sum of capacities from all servers being migrated, or based on sizing estimates using any published performance unit. In |
|----------------------------|---|

general the partitions are running in capped mode when using this strategy.

Harvested Capacity

Definition of partitions that have quality of service requirements, and allowing other partitions to run on the system with the resources eventually idle. You may have some partitions running uncapped when you use this approach so that they can use available resources in the system.

Planned Over-commit

Careful analysis of application resource usage and peak processing requirements, in order to deploy applications and substantially increase system utilization. You should run most of the partitions in uncapped mode.

Each of these strategies apply to different situations, depending on the amount of information you have for planning.

Guaranteed capacity

This is the basic rule of capacity planning for shared processor partitions. When you are planning a system for new applications, there is no performance data available on the resource consumption by the applications (since they are not installed). Therefore, you should rely on application sizing and performance requirements estimates in order to size the partitions, and add extra capacity in case the application needs more than initially planned.

This is also the case where you have the applications running, but cannot identify processing capacity consumption behavior (either because of insufficient information, or because of random behavior).

For these situations, the best approach is to size a system based on the required capacities, up to the peak capacity, and add an additional capacity for contingency. This method offers the smallest risk and is fairly simple to estimate. Moreover, since the system was planned based on the peak requirements for each application, you do not need a great effort in performance management, since there is installed capacity for all application needs.

The drawback of this strategy is that it does not optimize resource usage based on application behavior, and therefore a large fraction of the processing resources may be unused during hours of less activity, and also if applications present complementary processing needs (one application has a peak and other has a valley).

An application for this strategy is in case of new application deployment when a sizing is provided. One such application is a three-tiered ERP system. Based on

the functional requirements from the customer, a sizing tool generates an estimate for system requirements, based on peak requirements for each component of the solution.

A typical ERP solution is based on several servers running different functions. In general you have a database server, one or more application servers, one development system and one test system. An hypothetical example of a new system installation would be similar to the requirements listed in Table 6-2, where the different functions are listed with the peak performance requirements, at 100% processor usage.

Note: rPerf is an estimate of commercial processing performance relative between pSeries systems. It is derived from an IBM analytical model which uses characteristics from IBM internal workloads and industry transaction processing and web processing benchmarks. The rPerf model is not intended to represent any specific public benchmark result. It is being used here as an indication of the required performance in IBM systems for this specific scenario.

Table 6-2 An example of an ERP system requirements

| Function | Capacity in rPerf |
|----------------------|-------------------|
| DB Server | 4.1 |
| Application Server 1 | 3.5 |
| Application Server 2 | 3.5 |
| Development | 1.5 |
| Test | 1.0 |

Since we do not know how is the behavior of each individual system, in order to meet these requirements into a single system using micro-partitionin we sum the peak performance requirements for each function. We also factor in additional capacity necessary for micro-partitioning operations and activities. We consider adding an additional 20% of the required capacity to minimize the performance impact of micro-partitioning support logic.

If we were to use separate systems for each function, we would use five systems, with an adequate capacity to provide system usage within the performance requirements. In this example, we would have:

Table 6-3 Implementation with separate servers

| Function | Capacity requirement in rPerf | Server | Capacity provided in rPerf |
|--------------|-------------------------------|----------------------------------|----------------------------|
| DB Server | 4.4 | 2-way 1.5 GHz entry level server | 4.41 |
| App Server 1 | 3.7 | 2-way 1.2 GHz | 4.0 |
| App Server 2 | 3.7 | 2-way 1.2 GHz entry level server | 4.0 |
| Development | 2.0 | 2-way 1.2 GHz entry level server | 2.5 |
| Test | 1.5 | 2-way 1.2 GHz entry level server | 2.5 |

The amount of rPerf required for the application is 15.3. The amount of rPerf configured into the systems is 17.41, due to physical constraints (the number of processors must be an integer number). And while extra capacity is being configured, it cannot be allocated wherever it is needed, since these systems are separate.

If we use a more sophisticated approach by configuring a dedicated server, we will have more flexibility in moving extra resources among partitions, but still need to provide extra capacity than can be underutilized. Table 6-4 shows the same example using dedicated processor partitioning with 1.45 GHz pSeries middle range server:

Table 6-4 Dedicated processor partitioning with 1.45GHz middle range server

| Function | Capacity requirement in rPerf | Number of processors in the partition | Capacity provided in rPerf (entire machine) |
|--------------|-------------------------------|---------------------------------------|---|
| DB Server | 4.4 | 2 | 18.67 |
| App Server 1 | 3.7 | 2 | |
| App Server 2 | 3.7 | 2 | |
| Development | 2.0 | 1 | |
| Test | 1.5 | 1 | |

Again in this case, if much more extra resources are needed in the DB Server, for example, and the development partition is using only 30% of its capacity, there is no way to utilize the extra resources where they are required.

Now, if we consider to use a server with Micro-Partitioning, we can accommodate the different functions with more effective utilization. A single IBM @server p5 Model 550 can deliver up to 19.66 rPerf with four POWER5 processors running at 1.65 GHz. For this workload, we would have the configuration as shown on Table 6-5 using the same p5 server p550:

Table 6-5 Implementation with Micro-Partitioning

| Function | Capacity requirement in rPerf | Recommended capacity for Micro-Partitioning | Percentage of physical processor requirements |
|--------------|-------------------------------|---|---|
| DB Server | 4.4 | 5.28 | 1.07 |
| App Server 1 | 3.7 | 4.44 | 0.90 |
| App Server 2 | 3.7 | 4.44 | 0.90 |
| Development | 2.0 | 2.4 | 0.49 |
| Test | 1.5 | 1.8 | 0.37 |
| Total | 15.3 | 18.36 | 3.73 |

The extra resources on the machine can then be allocated to *any* of the partitions, whenever they require capacity. Moreover, once a partition is not using its total capacity, the remaining of its entitlement is automatically available in the shared processing pool. Also, once the applications are running, resource allocation can be fine tuned and allocated according to the partition needs.

Harvested capacity

When you have a mix of partitions that have a response time requirement (such as OLTP applications) and partitions that do not have response time requirements (such as batch applications, or test partitions), and you have some knowledge of the applications behavior, Micro-Partitioning gives you the ability of running the workloads without providing capacity for the peak processing of each partition. You can provide capacity for the partitions that have the response time requirements, up to peak capacity. Since they do not normally run in peak processing, the extra resources can be used by the partitions that do not have response time requirements. For these partitions, instead of specifying a peak capacity, you define a minimum capacity for them to run, and let them run uncapped, using the resources available from the other partitions.

Using the previous example, the DB Server and Application Server partitions still have their processing requirements guaranteed, and the Development and Test partitions could be configured as uncapped partitions, and use any available resources on the system.

Another good application of this strategy would be a case of a server farm running an application that receives load from load balancers. Normally the load will be balanced among the servers executing the application. In case one server gets more workload than others, it can use more resources from the processor pool, and return to normal behavior when the extra workload finishes.

Planned over-commit

This is the strategy where you make the most efficient use of the processing capacity in the system. On the other hand, it is the strategy that requires the most accurate planning and detailed knowledge about the applications behavior. It involves the utilization of different resource consumption from the applications, in order to share an amount of resources and deliver quality of service. Instead of planning for peak utilization for each application, you plan for the average and peak usage for one or a few partitions at time. In other words, on average processing, all partitions have their requirements fulfilled. If a few partitions consumes resources up to the peak, the system still fulfill all partitions requirements. If most or all of the application peaks at the same time, then the system is over-committed and some performance degradation may occur.

By adequate planning, a system can be configured with applications that do not overlap their peaks in processing, and therefore never over-commit the system. Total system usage will be high, and quality of service will be maintained, with maximum efficiency in resource usage.

Figure 6-19 on page 170 shows the processor usage for 3 different applications during the same period. From the charts you can see that the peaks in processing for each application are not at the same time. In that case, if you consolidate these application onto shared processor partitions, you can fulfill the processing requirements with less than the sum of peak requirements.

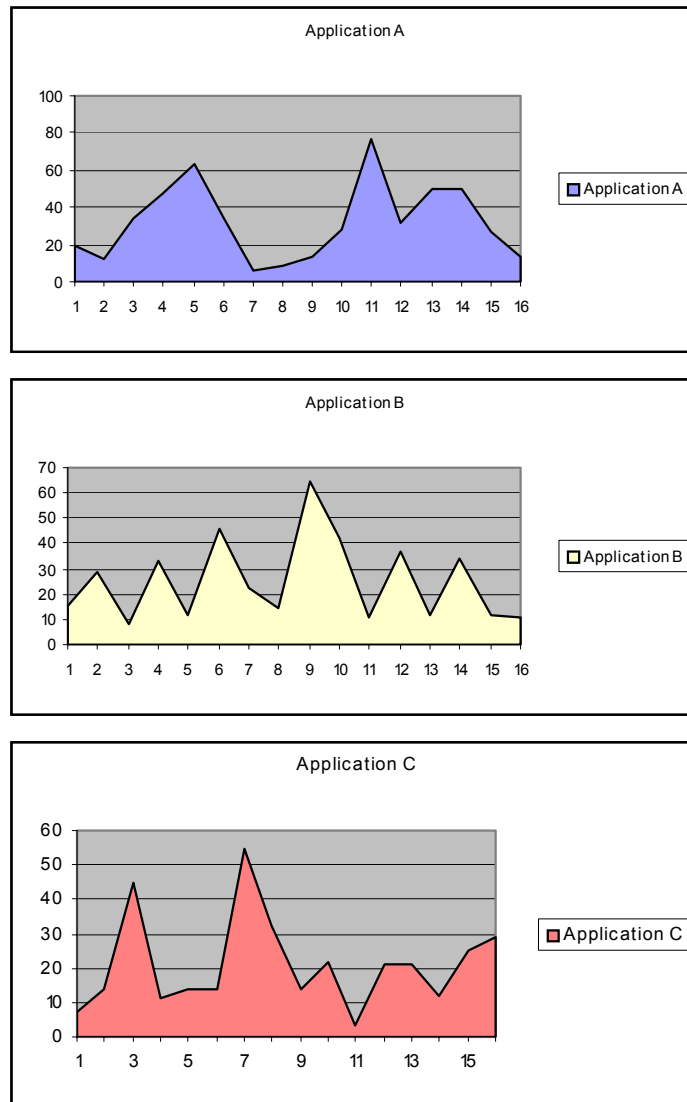


Figure 6-19 Processing resource consumption for three different applications

If we consolidate these applications on a server with Micro-Partitioning, we can benefit from their behavior to size a system with less capacity than the sum of all peaks. Table 6-6 shows the peak consumption for each partition, and the sum of the peaks.

Table 6-6 Peak consumption per partition

| Application | Peak processing (%) |
|-------------|---------------------|
| A | 77 |
| B | 65 |
| C | 55 |
| TOTAL | 197 |

If we hypothetically consider 0.1 rPerf for each percent user by the applications, in order to run the three applications with the peak performance requirements, you would need a server with 19.7 rPerf.

If, instead of this, we use Micro-Partitioning, then what you do is sum the usage for the three applications, at a given time. Figure 6-20 shows the result of this sum, and we can see that the maximum peak processing for the sum is 96%. Using the same consideration of 0.1 rPerf for each 1% consumption, we come to a requirement of 9.6 rPerf for all three applications.

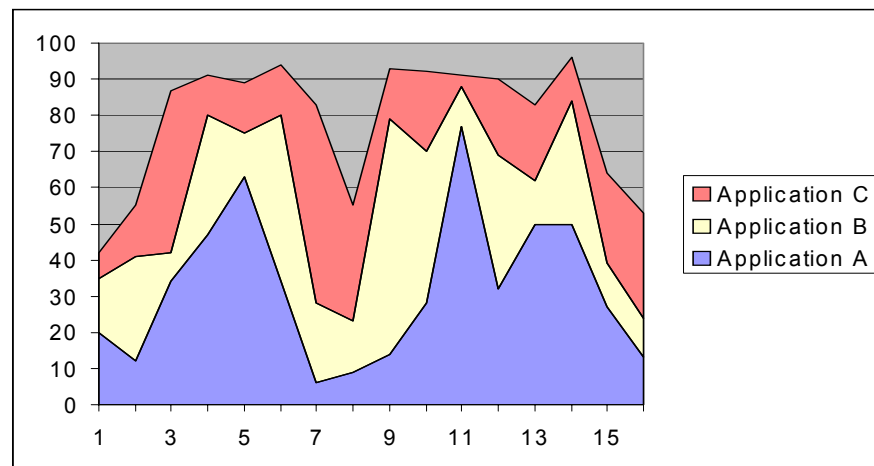


Figure 6-20 Application resource consumption example

By adding 20% extra capacity for Micro-Partitioning management activities, we come to a requirement of 11.52 rPerf. This is about half the capacity we would need if sizing for peak capacity of each application (and also including the 20% extra capacity).

As commented before, this is the most efficient strategy for consolidating running systems using Micro-Partitioning. It is important to notice that all partitions must be uncapped, so they can get the resources needed for peak processing. It is

also important to note that if for some reason the peaks in processing change, the partition entitlements must be recalculated and a new planning should be done. Otherwise, partitions may not be able to get the resources they need, and application performance will be as good as it could be.



Virtual I/O

This chapter provides a short introduction to virtual I/O and a deeper view of how the POWER5 Hypervisor handles the transactions between the partitions.

This chapter then addresses performance aspects for each of the components of the Virtual IO system.

For a general description of Virtual IO components and of their configuration please refer to chapters 3, 4 and 5 of *Advanced POWER Virtualization on IBM @server p5 Servers Introduction and Basic Configuration*, SG24-7940

The Virtual I/O product documentation can be found at the following URLs:

Using the Virtual I/O Server

http://publib.boulder.ibm.com/infocenter/eserver/v1r2s/en_US/info/iphb1/iphb1.pdf

Virtual I/O Server and Partition Load Manager Commands Reference

http://publib.boulder.ibm.com/infocenter/eserver/v1r2s/en_US/info/iphb1/commands/commands.pdf

7.1 Introduction

With the usage of virtual partitions in POWER5-based servers, the number of partitions that can be concurrently instantiated on a p5 server can be greater than the number of physical I/O slots in the *Central Electronic Complex* (CEC) and its RIO drawers. For example, at the time of this publication POWER5 processor-based servers support 254 Logical Partitions (LPARs), while a mid-range server p5-520 with RIO drawers could have up to 160 I/O slots.

Typically a small operating system instance needs at least one slot for a Network Interface Connector and one slot for a disk adapter (SCSI, Fiber Channel, ...), while more robust configurations often consist of two redundant NIC adapters and two disk adapters.

To be able to connect enough IO devices to each partition configured on a p5 server, IBM introduced virtual Input/Output technology for POWER5-based servers. Virtual I/O devices are an optional feature of a partition and can be used on POWER5 based systems in conjunction with AIX 5L Version 5.3 or Linux. Virtual I/O devices are intended as a complement to physical I/O adapters (also known as dedicated or local I/O devices). A partition can have any combination of local and virtual I/O adapters.

The generic term of “Virtual I/O” relates to 5 different concepts:

- ▶ three adapters: Virtual SCSI, Virtual Ethernet and Virtual Serial
- ▶ a special AIX partition, called Virtual IO server
- ▶ a mechanism to link virtual network devices to real devices, called Shared Ethernet Adapter (SEA).

We will address each of these concepts in sections 7.2 to 7.5.

But first, let us see how the hypervisor provides each partition with Virtual IO devices. Sections up to 7.1.4 presents underlying concepts and objects used to implement Virtual IO support.

Note: If you are not interested in the internals of Virtual I/O implementation, you can skip directly to section 7.1.4, “The Virtual I/O Server” on page 180.

7.1.1 Hypervisor support to Virtual I/O

Figure 7-1 shows the hypervisor providing the interconnection between the partitions and the Virtual IO server, as well as interconnection between partitions.

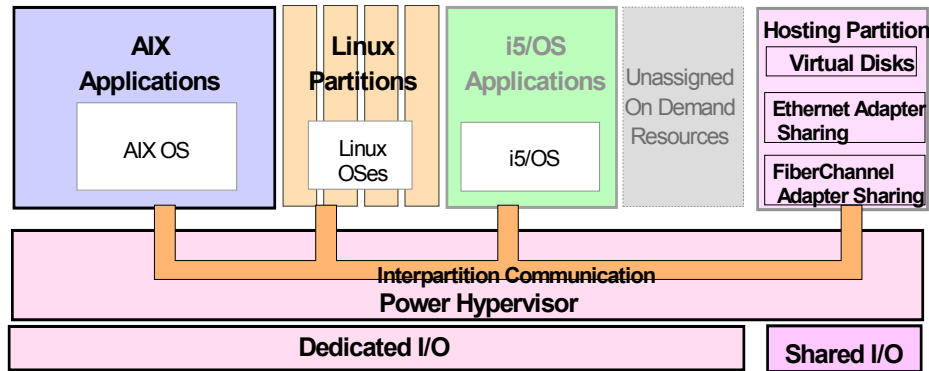


Figure 7-1 Virtual I/O provided by Hypervisor

To use the functionalities of Virtual I/O, a partition will use a virtual adapter. The Hypervisor provides the partition with a view of an adapter that has the appearance of a I/O adapter, but which may not correspond to a physical I/O adapter. The Hypervisor simulates this virtual adapter with one of three techniques:

- ▶ In the first case, the Hypervisor may totally simulate the adapter. This is used for example in the Virtual Ethernet support (see Chapter 7.2, “Virtual Serial adapter” on page 181). This technique is applicable to communications *between partitions that are created by a single Hypervisor instance*.
- ▶ In the second case, a server partition provides the services of one of its physical I/O adapters to a client partition(s). This for example is used for Virtual SCSI disks. The hypervisor provides the communication facility between the virtual adapters of the client partition and the server partitions

The server partition provides support to interpret I/O requests from the partner partition, perform those requests on one or more of its devices, targeting the client partition's DMA buffer areas using the Remote DMA (RDMA) facilities, (see “Logical Remote Direct Memory Access (LRDMA)” on page 180) and passing I/O responses back to the partner partition. I.

- ▶ The third case is just mentioned for the sake of completeness and will not be further handled in this chapter. It will be used in the future for other types of devices such as the InfiniBand Host Channel Adapter (HCA).

Figure 7-2 depicts graphically the first two cases. The Hypervisor (the box labeled PHYP for Power Hypervisor) either completely provide the simulated connectivity between the two partitions, or also provides the data transfer mechanism between a partition and a disk physically attached to another partition.

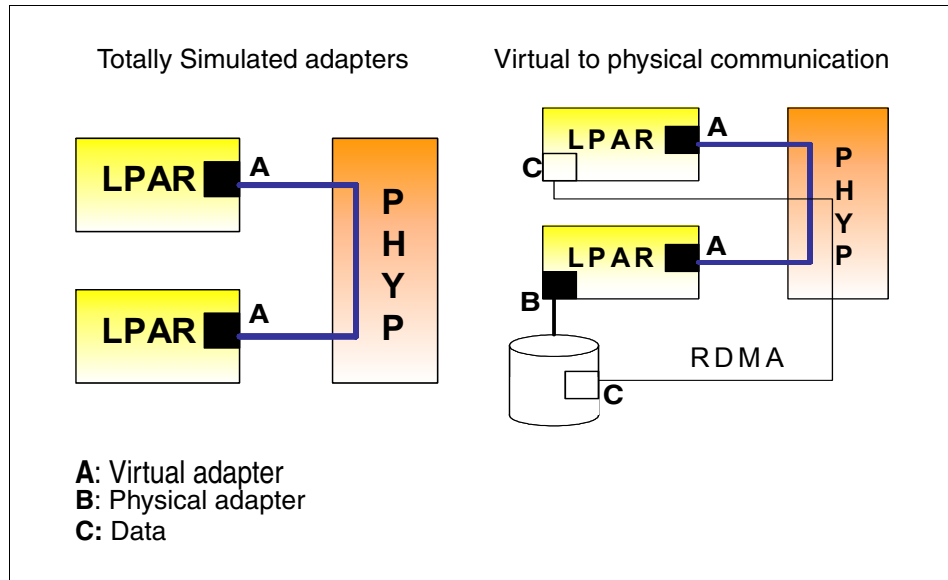


Figure 7-2 Hypervisor simulated adapters

7.1.2 Virtual I/O infrastructure

The VIO infrastructure is a complex subject, and it is not the purpose of this book to address it exhaustively. We only briefly present here some of its component which are relevant to the understanding of the performance issues.

The Open Firmware device tree

The Virtual I/O Adapters with the associated interpartition communications paths (if any) are defined from the HMC during the creation of the partitions profiles.

When a partition is booted, it receives from the Hypervisor the definition of all its available hardware resources as device nodes in what is called the partition Open Firmware device tree. This device tree also contains the definition of the virtual resources provided by the hypervisor to the partition.

Depending upon the specific virtual device their device tree node may be found as a child of / (the root node) or in the virtual I/O sub-tree.

In addition to the virtual I/O devices, the Open Firmware device tree also contains definition of virtual host bridge and virtual interrupt source controller, so that the partition can communicate with the virtual devices in the same way it communicates with physical devices.

For information, each virtual device node in the Open Firmware device tree contains the properties defined in Table 7-1.

Table 7-1 Required attributes of the /vdevice node

| Property Name | Req? | Definition |
|--------------------|------|--|
| name | Yes | Standard property name per IEEE 1275 specifying the virtual device name, the value shall be "vdevice" |
| device_type | Yes | Standard property name per IEEE 1275 specifying the virtual device type, the value shall be "vdevice" |
| model | N/A | Property not present |
| compatible | Yes | Standard property name per IEEE 1275 specifying the virtual device programming models, the value shall include "IBM,vdevice" |
| used-by-rtas | NA | Property not present |
| ibm,loc-code | N/A | The location code is meaningless unless one is doing dynamic reconfiguration as in the children of this node. |
| reg | N/A | Property not present |
| #size-cells | Yes | Standard property name per IEEE 1275, the value shall be 0. No child of this node takes space in the address map as seen by the owning partition |
| #address-cells | Yes | Standard property name per IEEE 1275, the value shall be 1 |
| #interrupt-cells | Yes | Standard property name per IEEE 1275, the value shall be 2. The first cell contains the interrupt# as will appear in the XIRR and is used as input to interrupt RTAS calls the second cell contains the value 0 indicating a positive edge sense |
| interrupt-map-mask | N/A | Property not present |
| interrupt-ranges | Yes | Standard property name that defines the interrupt number(s) and range(s) handled by this unit. |
| ranges | ? | These will probably be needed for IB virtual adapters. |

| Property Name | Req? | Definition |
|-----------------------|--------|---|
| interrupt map | NA | Property not present |
| interrupt-controller | Yes | The /vdevice node appears to contain an interrupt controller. |
| ibm,drc-indexes | for DR | For <i>Dynamic Reconfiguration (DR)</i> . Refers to the DR slots -- the number provided is the maximum number of slots that can be configured which is limited by, among other things, the RTCE tables allocated by the Hypervisor. |
| ibm,drc-power-domains | for DR | Value of -1 to indicate that no power manipulation is possible or needed. |
| ibm,drc-types | for DR | Value of "SLOT". Any virtual IOA can fit into any virtual slot. |
| ibm,drc-names | for DR | The virtual location code |

7.1.3 Types of Connections

The virtual I/O infrastructure provides several primitives that are then used to build connections between partitions for various purposes. These primitives include:

- ▶ A *Command/Response Queue (CRQ)* facility which provides a pipe between partitions. A partition can enqueue an entry on its partner's CRQ for processing by that partner. The partition can set up the CRQ to receive an interrupt when the queue goes from empty to non-empty, and hence this facility provides a method for an inter-partition interrupt.
- ▶ An *extended Translation Control Entry (TCE)* table called the *Remote DMA TCE* table which allows a partition to provide "windows" into the memory of its partition to its partner partition, while maintaining addressing and access control to its memory.
- ▶ *Remote DMA* services that allow a server partition to transfer data to a partner partition's memory via the RTCE table window panes. This allows a device driver in a server partition to efficiently transfer data to and from a partner, which is key in sharing of a virtual I/O adapter in the server partition with its client partition.

The Command/Response Queue (CRQ)

The CRQ facility *provides ordered delivery of messages* between authorized partitions. The facility is reliable in the sense that the messages are delivered in sequence, that the sender of a message is notified if the transport facility is

unable to deliver the message to the receiver's queue, and that a notification message is delivered (providing that there is free space on the receive queue), or if the partner partition either fails or deregisters its half of the transport connection.

The CRQ facility does not police the contents of the payload portions (after the 1 byte header) of messages that are exchanged between the communicating pairs, however, the architecture does provide means (via the Format Byte) for self describing messages such that the definitions of the content and protocol between using pairs may evolve over time without change to the CRQ architecture, or its implementation.

The CRQ is used to hold received messages from the partner partition. The CRQ owner may optionally choose to be notified via an interrupt when a message is added to their queue.

For the synchronous infrastructure, the CRQ facility defined above ,is implemented via the Reliable Command/Response Transport option. The synchronous nature of this infrastructure allows for the capability to immediately (synchronously) notify the sender of the message whether the message was delivered successfully or not.

Remote Translation Control Entry - RTCE

The TCE and RTCE tables are used to translate I/O DMA operations and provide protection against improper operations.

The RTCE table for *Remote DMA* (RDMA) is analogous to the TCE Table for dedicated I/O. The RTCE table does, however, have a little more information in it (as placed there by the Hypervisor). So it can, among other things, allow the Hypervisor to create links to dedicated I/O Adapters TCEs, that were created from the RTCE table TCEs. A TCE in an RTCE table is never accessed directly by the partitions software; only though Hypervisor `hca11()`.

Table 7-2 Comparing TCE and RTCE

| TCE (Translation Control Entry) | RTCE (Remote TCE) |
|--|--|
| In POWER4 based pSeries Servers | additionally In POWER5 based pSeries Servers |
| Translation Table for logical to dedicated I/O Bus Addresses | Is needed for Remote DMA |
| managed by the Hypervisor | managed by the Hypervisor |

| TCE (Translation Control Entry) | RTCE (Remote TCE) |
|-----------------------------------|---|
| addressed by the Operating System | <i>never</i> addressed directly by the Operating System. Addressed only through Hypervisor <code>hcall()</code> |

Logical Remote Direct Memory Access (LRDMA)

The Virtual I/O infrastructure can take advantage of different types of Direct Memory Access (DMA). The virtual SCSI feature only uses Logical Remote Direct Memory Access (LRDMA).

LRDMA allows for an I/O server to securely target memory pages within an I/O client for virtual I/O operations. The I/O server uses the `hcall()` of the Logical Remote DMA facility to manage the movement of commands and data associated with the client requests. The server driver may use this service if it has a connection established via a Command/Response Queue pair. Virtual SCSI defines two modes of LRDMA:

- ▶ Traditional Copy RDMA that involves the I/O server's I/O adapters targeting DMA buffers in the I/O server's memory and having the Hypervisor copy data between that DMA buffer and the I/O client's memory.
- ▶ Redirected RDMA allows for an I/O server to securely target its physical I/O adapter's DMA operations directly at the memory pages of the I/O client.

Hypervisor calls

The calls used by the Hypervisor, are described in "Hypervisor Call Functions" on page 43

7.1.4 The Virtual I/O Server

The IBM Virtual I/O Server is the link between the virtual and the real world. It owns the physical I/O resources. It can be seen as an appliance based on a specialized OS, and it is supported on POWER5-based Servers only. The Virtual I/O server runs in a special partition which can not be used for execution of application code.

It mainly provides two functions:

- ▶ Server Part for the Virtual SCSI devices (VSCSI target)
- ▶ Shared Ethernet adapter for Virtual Ethernet.

The Virtual I/O Server is currently shipped as one operating system image that can be restored on the partition either from the HMC, of from the CD-rom of the

p5 server. As described before, it is based on a Unix-like operating system, but it not accessible as a standard partition. Administrative access to the I/O Server partition is only possible as user padmin, not as id root. After login, user padmin gets a restricted shell, which is not escapable. This is called the I/O Server Command Line Interface (IOCLI).

The operating system of the I/O Server is hidden to simplify transitions to further versions. No specific OS skill is required for administration the I/O Server.

This server supports AIX 5L V5.3 and Linux client partitions.

The performance considerations on the Virtual I/O Server will be addressed in “Findings of Virtual I/O Server performance” on page 208.

A measurement of Virtual I/O in conjunction with the *Shared Ethernet Adapter* (SEA) functionality will be handled in “Shared Ethernet adapter functionality” on page 203

7.2 Virtual Serial adapter

There are three types of Virtual IO devices supported by the POWER5 Hypervisor. We discuss virtual Ethernet networks and virtual disks in the following sections.

The virtual Serial adapter can only be used for providing a virtual console to the partitions. This console is visible by the end user from the HMC display.

The virtual serial port cannot be used for any other purpose. For example, it cannot be used for HACMP heartbeat monitoring.

There are no specific performance considerations to address regarding the Virtual Serial adapter.

7.3 Virtual Ethernet

This chapter provides a brief description and addresses performance aspects of Virtual Ethernet and the POWER5 Hypervisor.

Attention: Virtual Ethernet is also called Virtual LAN or even VLAN, which can be confusing, because these terms are also used in network topology topics. But the Virtual Ethernet, which uses virtual devices has nothing to do with the VLAN known from Network-Topology, which divides a LAN in further Sub-LANs.

7.3.1 Introduction

Virtual Ethernet creates logical ethernet connections between one or more partitions. There are no physical Adapter needed for implementing a Virtual Ethernet. The Hypervisor copies frames from and to the virtual interfaces between the LPARs as shown in Figure 7-3. It acts as a virtual Ethernet switch between the logical Interfaces. So, Virtual Ethernet is a *memory based Interpartition LAN*. A Virtual Network may be “bridged” to an external network to permit partitions without physical network adapters to communicate outside of the server and vice versa. This functionality is given by the Virtual I/O Server, described in “Shared Ethernet adapter functionality” on page 203.

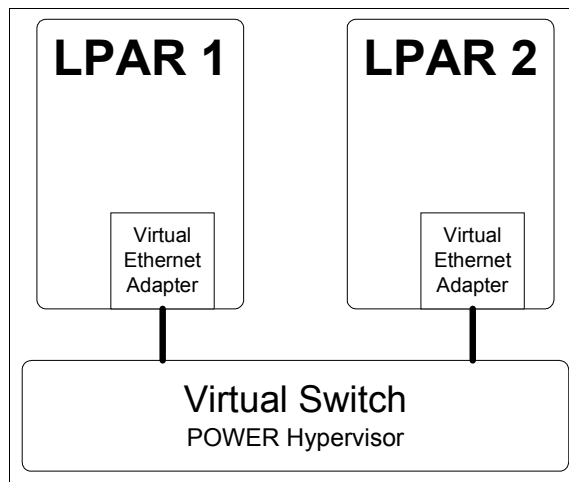


Figure 7-3 Virtual Ethernet

A virtual Ethernet adapter can be configured like a standard ethernet adapter. Virtual Ethernet adapters can be used in dedicated partitions and in micro-partitions.

In every partition, virtual and dedicated network devices can be used simultaneously for communication. Figure 7-4 shows adapters of a partitions that has one Virtual Ethernet adapter (ent0) and two real adapters (ent1 and ent2). Up to 256 Adapters (sum of virtual and real) are supported per LPAR.

| | | | | |
|---------------------|-----------|--------------------------------------|--|------------|
| # lsdev -Cc adapter | | | | |
| ent0 | Available | Virtual I/O Ethernet Adapter (1-lan) | | |
| ent1 | Available | 01-08 | 2-Port 10/100/1000 Base-TX PCI-X Adapter | (14108902) |
| ent2 | Available | 01-09 | 2-Port 10/100/1000 Base-TX PCI-X Adapter | (14108902) |
| vsa0 | Available | LPAR Virtual Serial Adapter | | |
| vscsi0 | Available | Virtual SCSI Client Adapter | | |

Figure 7-4 Virtual and local adapters on one partition

7.3.2 General Concepts

IEEE 802.1Q support (VLAN Tagging)

The Virtual Ethernet, shipped with AIX 5L V5.3 supports the IEEE 802.1Q standard. This standard describes the “Virtual Bridged Local Area Networks”. IEEE needs a Virtual LAN ID (VID). The LAN ID is optional in the above implementation. When this option is selected while adding a new Virtual LAN interface at the HMC, a VID can be chosen. Up to 4094 Virtual LANs are supported per CEC. Up to 21 VIDs can be configured per Virtual LAN port. See ““Comparing physical and Virtual Ethernet” on page 183” on how the Hypervisor handles this support.

Comparing physical and Virtual Ethernet

A virtual LAN adapter appears to the Operating System in the same way as a physical adapter. It also can be configured in the same manner. While the MAC (Media Access Control) Address of physical Ethernet is coded on the (hardware) adapter, the MAC-Address of the virtual adapter is generated by the HMC.

Figure 7-5 on page 184 shows the standard TCP/IP Suite of protocols. Every Layer adds a additional header to the frame. After Frames passed Transport and Network layer, they are received by the Network Interface layer. The Network Interface layer adds the Ethernet header, and sends the datagram to the hardware. Additionally, it handles segmentation and recalculates the header checksum of the outgoing IP-packets. In physical Ethernet, this is done by the physical adapter. In Virtual Ethernet, the Hypervisor is responsible for that.

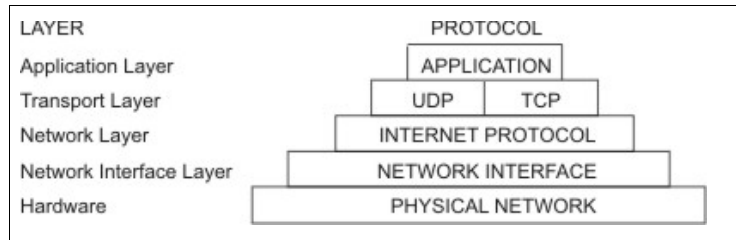


Figure 7-5 TCP/IP Suite of protocols

MTU Sizes

The Virtual Ethernet Adapter supports, as Gigabit (Gb) Ethernet, Standard MTU-Sizes of 1500 Byte and Jumbo frames with 9000 Byte. Additionally to Gb-Ethernet, the *MTU-Size of 65280 bytes* is also supported in Virtual Ethernet. So, the MTU of 65280 Bytes can be only used inside a Virtual Ethernet.

IPv6 Support

Virtual Ethernet supports multiple protocols, like IPv4 and IPv6.

Adapter configuration scenarios

There are two ways of configuring multiple virtual adapters in a partition. Figure 7-6 on page 185 shows 3 partitions with virtual Ethernet connections.

LPAR 2 has only one virtual adapter ent0.

LPAR3 has 2 virtual adapters, ent0 and ent1. Each of these adapters corresponds to a virtual adapter defined on the HMC for this partition. One network interface has been defined on each of these virtual adapter. ent0 and ent1 each have a Port Virtual ID, respectively 1 and 10.

LPAR 1 has 2 virtual adapters, ent0 and ent1. But only ent0 corresponds to an HMC defined adapter, while ent1 has been defined on top of ent0. ent0 has a Port Virtual ID of 1, while ent1 uses the VID 10.

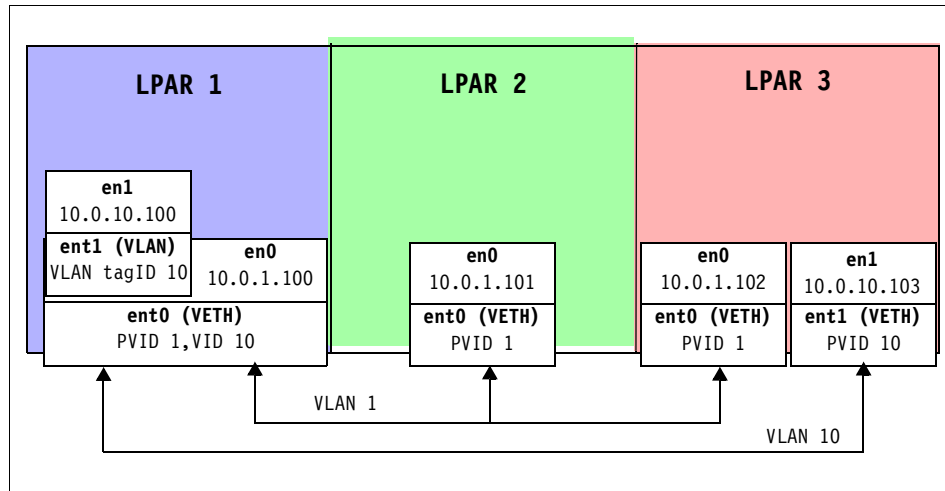


Figure 7-6 Example of two VLANs in a Virtual Ethernet Environment

The following lines shows how these adapters appears in the **lsdev** command on partition 3:

```
# lsdev -C -c adapter
ent0  Available  Virtual I/O Ethernet Adapter (1-lan)
ent1  Available  Virtual I/O Ethernet Adapter (1-lan)
```

and on partition 1:

```
# lsdev -C -c adapter
ent0  Available  Virtual I/O Ethernet Adapter (1-lan)
ent1  Available  VLAN
```

From a performance point of view, the configuration with the 2 HMC defined adapters will yield a better network throughput if the partition run on multiple processors, since each adapter will be processed by its own thread and will have its own interrupts.

In LPAR 1, the same thread (and therefore only one processor) will handle the traffic for both IP addresses. The effect on CPU utilization of adding an extra network adapter (ent1) on top of the first network adapter (ent0) to handle the VLAN tagged ID is minimal (less than 1.5% for each partition),

7.3.3 Hypervisor switch implementation

The Hypervisor switch is consistent with IEEE 802.1 Q. It works on OSI-Layer 2 and supports up to 4096 networks (4096 VIDs).

When a message arrives at a Logical LAN Switch port from a Logical LAN adapter, the Hypervisor caches the message's source MAC address (2nd 6 bytes) to use as a filter for future messages to the adapter. The Hypervisor then processes the message differently depending upon whether the port is configured for IEEE VLAN headers, or not. If the port is configured for VLAN headers, the VLAN header (bytes offsets 12 and 13 in the message) is checked against the port's allowable VLAN list. If the message specified VLAN is not in the port's configuration the message is dropped. Once the message passes the VLAN header check, it passes onto destination MAC address processing below.

If the port is NOT configured for VLAN headers, the Hypervisor (conceptually) inserts a two byte VLAN header (based upon the port's configured VLAN number) after byte offset 11 in the message. Next, the destination MAC address (first 6 bytes of the message) is processed by searching the table of cached MAC addresses (built from messages received at Logical LAN Switch ports see above).

If a match for the MAC address is not found and if there is no *trunk adapter* defined for the specified VLAN number, the message is dropped, otherwise if a match for the MAC address is not found and if there is a trunk adapter defined for the specified VLAN number, the message is passed on to the trunk adapter. If a MAC address match is found, then the associated switch port's configured, allowable VLAN number table is scanned for a match to VLAN number contained in the message's VLAN header. If a match is not found, the message is dropped.

Next the VLAN header configuration of the destination switch port is check, if the port is configured for VLAN headers the message is delivered to the destination Logical LAN adapters including any inserted VLAN header. If the port is configured for no VLAN headers the VLAN header is removed before being delivered to the destination Logical LAN adapter

Figure 7-7 on page 187 shows a graphical representation of the behavior of the Virtual Ethernet when processing packets.

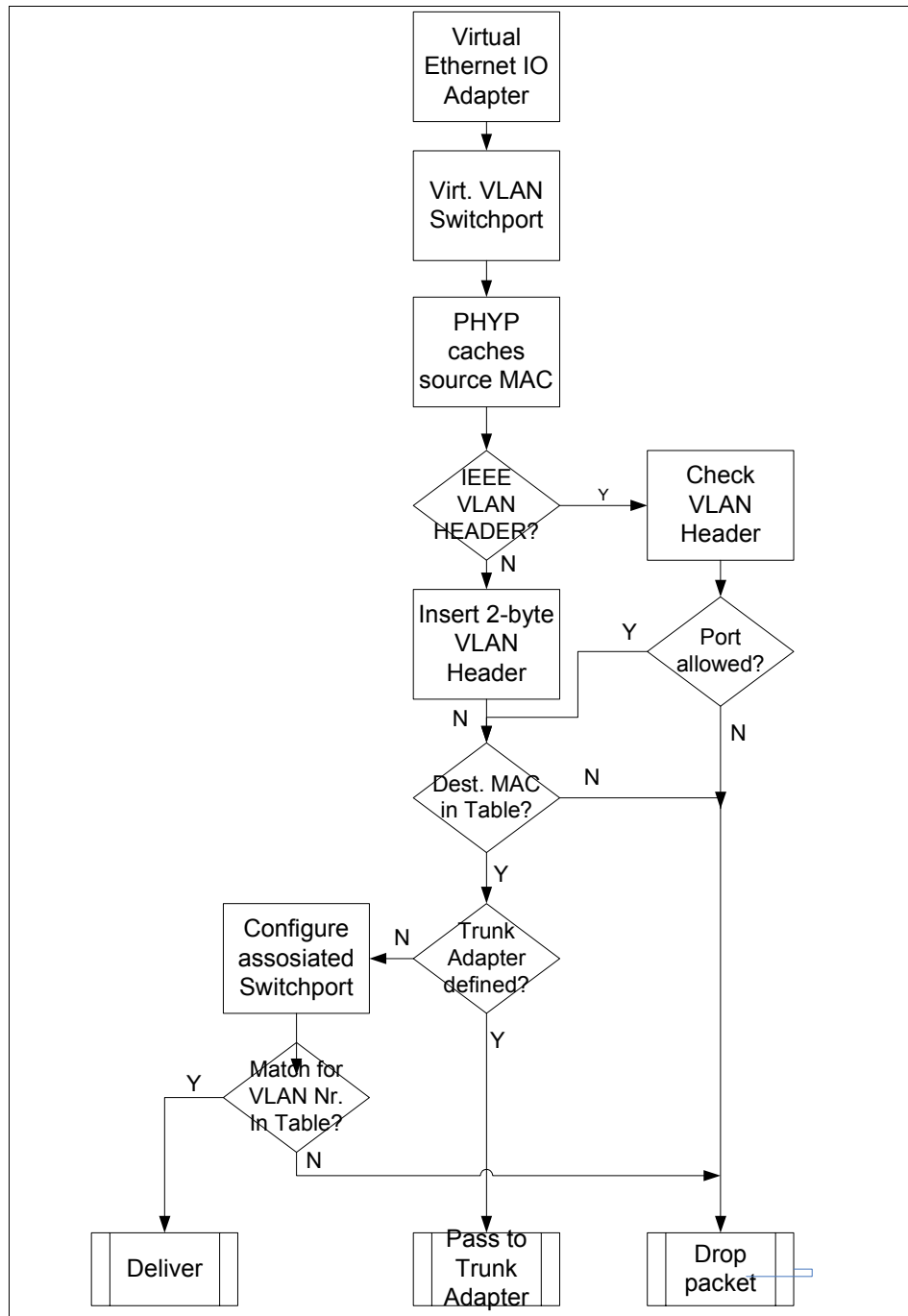


Figure 7-7 Flow chart of Virtual Ethernet

7.3.4 Performance Considerations and Measurements

This sections presents several experiments that were performed on a p5 server to measure the influence of some of the parameters a system administrator can set.

General comments to the measurements

The Operating System running on all partitions is AIX 5L v5.3 (GA code). The results of the measurements could vary if they will be repeated at a later time with other versions of operating system and firmware.

The platform that was used for these tests was a 4-way IBM eServer p5 570.

Unless otherwise mentioned, the Virtual Ethernet connections are set up between two partitions, each configured with one dedicated processor, and Simultaneous Multi Threading (SMT) is activated on the tested partitions.

The Virtual Ethernet and Gb ethernet adapters were tested with their default Interface Specific Network Options (as defined in the `no` command) and Object Data Manager (ODM) settings. Specifically, these were:

Virtual Ethernet

- For MTU 1500, tcp_sendspace=131072, tcp_recvspace=65536
- For MTU 9000, tcp_sendspace=262144, tcp_recvspace=131072, rfc1323=1
- For MTU 65394, tcp_sendspace=262144, tcp_recvspace=131072, rfc1323=1

Gigabit Ethernet (Feature Code 5700 or 5701)

- For MTU 1500, tcp_sendspace=131072, tcp_recvspace=65536
- For MTU 9000, tcp_sendspace=262144, tcp_recvspace=131072, rfc1323=1

The Gigabit adapter defaults were used, which include large_send (also known as TCP segmentation off load), TCP checksum off load, and interrupt coalescing. The ODM attributes were: large_send=1, chksum_offload=1 and intr_rate=10000.

Description of the performance tests and tools

To measure the Virtual LAN performance, the used benchmark is **netperf**.

netperf is a benchmark that can be used to measure various aspects of networking performance. Currently, its focus is on bulk data transfer (streaming)

and request/response performance using either TCP or UDP, with the Berkeley Sockets interface.

While this benchmark is now part of the public domain, IBM has developed a derivative tool which is more tightly integrated with the capabilities of the AIX operating system. All measurement described in this book are using the IBM modified version of **netperf**.

The experiment results presented later are using both operation modes of **netperf**: Streaming mode called TCP_STREAM, and transactional request/request mode called TCP_RR.

In each mode, four “sessions” are used: in other words, 4 programs are sending traffic over the connection, to simulate a real workload with multiple IP sessions flowing through the same adapter.

TCP Stream performance: TCP_STREAM

This benchmark will perform data streaming test between the local system and the remote system.

TCP_STREAM will be used in simplex and duplex mode. On simplex mode, one side will send and the other end will receive data, on duplex mode, both ends send and receive at the same time. So the amount of data, transported via the media will increase.

The TCP_STREAM benchmark can be performed with different data chunk size. The results presented here are for an application that sends data chunks between 16 Kbytes and 64 kbytes to the communication sockets (which then split them into IP packets depending on the MTU size).

TCP request/response performance: TCP_RR

Netperf request/response performance is quoted as “transactions per second” for a given request and response site. A transaction is defined as the exchange of a single request and a single response. From a transaction rate, one can infer one way round-trip average latency.

The TCP_RR benchmarks are done with one and 20 sessions. The 20 sessions test shows, in opposition to the 1 session test, how the response time and latency is growing with more load.

Overview of the following benchmark measurements

First, there will be a measurement that shows how throughput is growing by adding more *entitlements to a virtual processor*, then a benchmark test comparing parameters such as processor consumption, transaction rate and

latency from *dedicated* ethernet (gigabit adapter) to *Virtual Ethernet*. The last set of measurements will show the difference in performance of the Virtual Ethernet adapter between the single-threaded and SMT modes.

7.3.5 Virtual LAN throughput at different processor entitlements

This measurement shows, what throughput can be expected in a Virtual LAN. Because of the throughput varies on processor entitlements and MTU size, these parameters are variable in the following measurement.

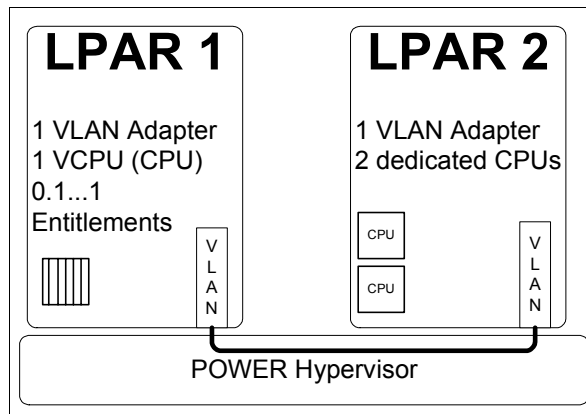


Figure 7-8 Processor entitlements and MTU sizes

Both LPARS have one Virtual LAN adapter and there are multiple sessions running between both adapters. The used benchmark for this is **netperf** TCP_STREAM as described before.

LPAR 1 (with varied processor entitlements) is sending a simplex stream, LPAR2 (2 way dedicated) receives it.

The goal of the test is to measure LPAR1, so resources for LPAR 2 are oversized (2-way partition) so there is no bottleneck on the receiving side that would affect the measurement, and so that the throughput of the Virtual LAN interface of LPAR1 can be effectively measured as a function of the CPU entitlement of LPAR1.

Results of Virtual Ethernet throughput

Figure 7-9 and Figure 7-10 on page 191 and Figure 7-11 on page 192 show the performance measurements taken.

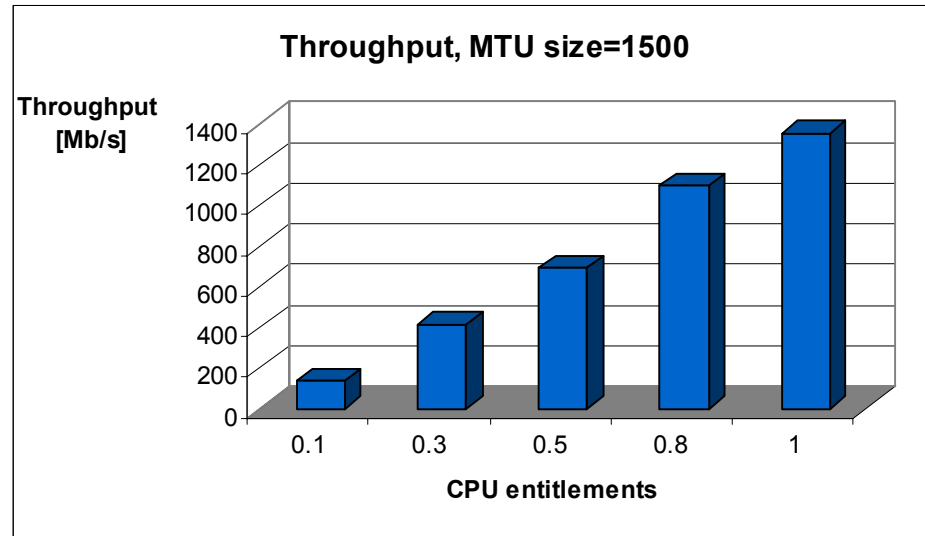


Figure 7-9 Throughput vs. CPU entitlements, MTU size=1500

Figure 7-9 to Figure 7-11 show the measured throughput at different processor entitlements and MTU sizes.

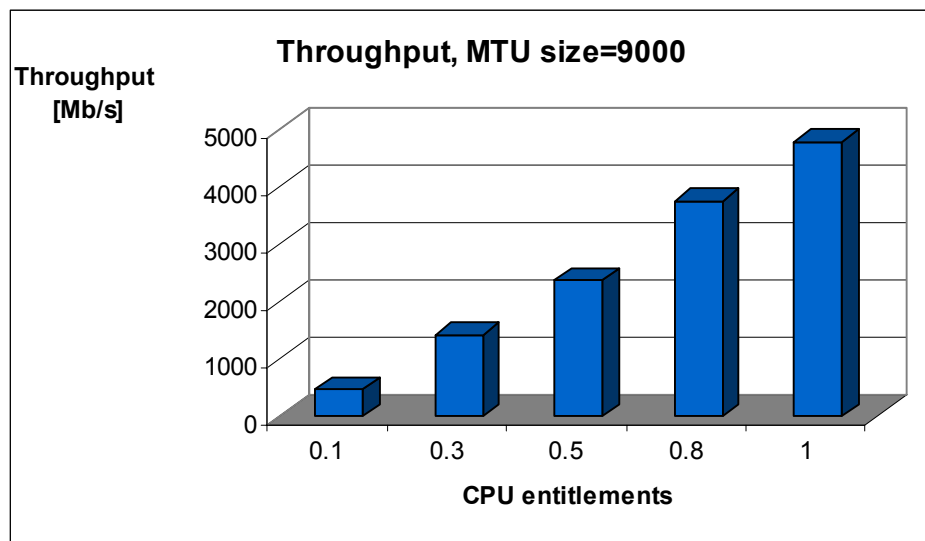


Figure 7-10 Throughput vs. CPU entitlements, MTU size=9000

There is one chart for every MTU size: 1500, 9000 and 65394.

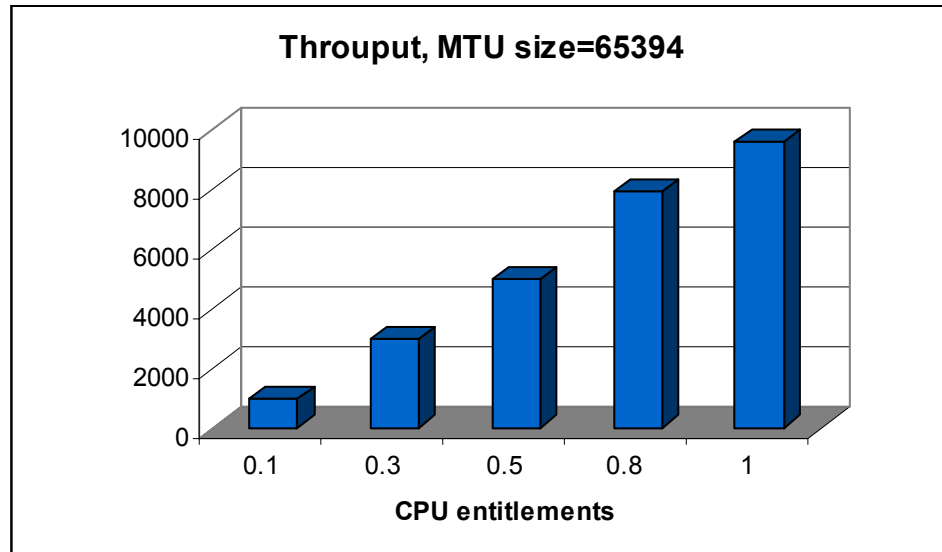


Figure 7-11 Throughput vs. CPU entitlements, MTU size=65394

Findings on Virtual Ethernet performance

The throughput of the Virtual Ethernet scales nearly linear with the allocated processor entitlements. Figures 7-9, 7-10 and 7-11 do not show this linearity, since the spacing of the column is not proportional to the values of the CPU entitlements (used on the X axis). For better comparison, all above measured data are *normalized to 0.1* processor entitlement: ($\text{Throughput} \times 0.1 / \text{Entitlements}$), and presented in Figure 7-12 on page 193, showing the linearity of the throughput.

Throughput with MTU=9000 is more than three times the rate with MTU=1500 and the throughput with MTU=65394 is more than seven times the rate with MTU=1500. This is due to improved efficiency of sending larger packets with one call up or down the TCP/IP protocol stack.

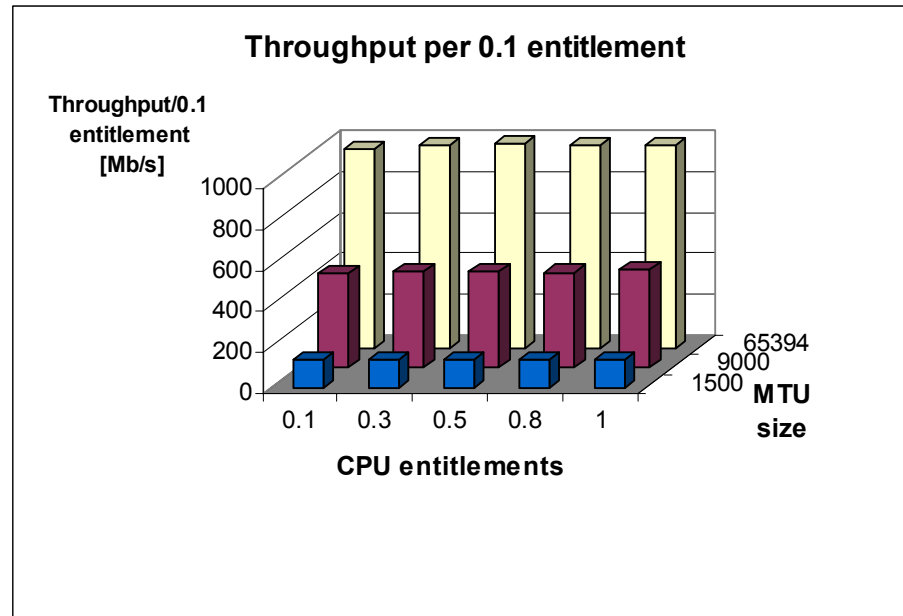


Figure 7-12 Throughput normalized to 0.1 Entitlement

7.3.6 Comparing throughput of Virtual LAN to Gb ethernet

With the next benchmark, we check performance of Virtual LAN versus Gigabit ethernet (GbEN). Both LPARS are assigned one dedicated POWER5 processor, and run in SMT mode.

Figure 7-13 and Figure 7-14 on page 194 show the two different types of connection between the LPARS: Virtual Ethernet via Hypervisor and Gigabit ethernet via a Gigabit Ethernet Switch.

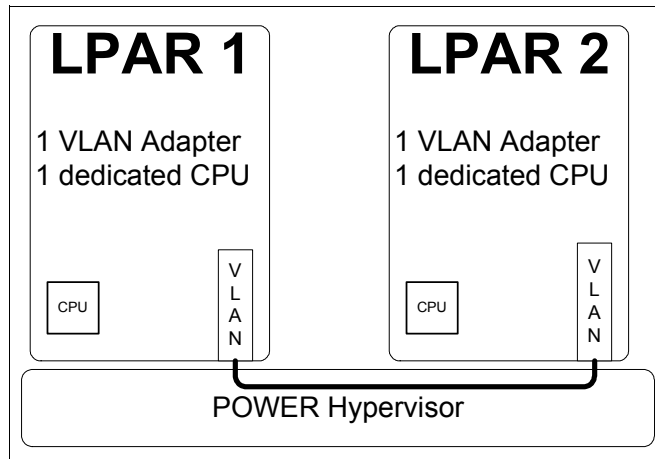


Figure 7-13 VLAN to VLAN performance

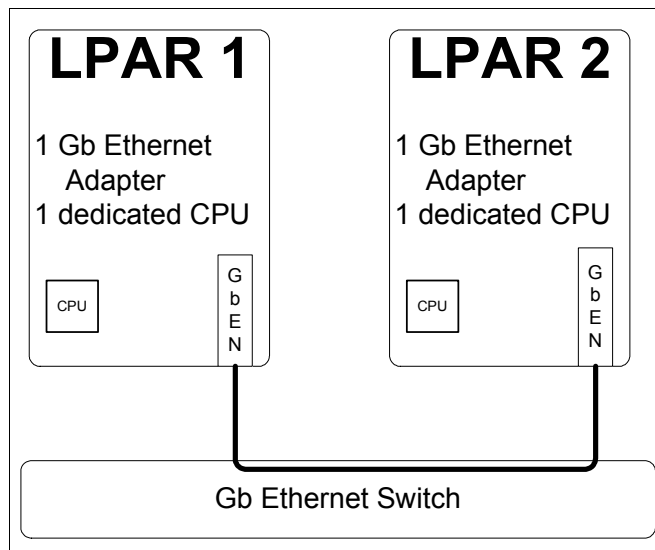


Figure 7-14 Gb Ethernet to Gb Ethernet

The benchmark TCP_STREAM was running in simplex and duplex mode at different MTU sizes on both setups, measuring throughput and processor consumption.

Throughput of Virtual LAN and gigabit ethernet

Figure 7-15 shows how throughput varies with different values of MTU size, in simplex and duplex modes. Since Gigabit Ethernet adapter doesn't support MTU size of 65394, there is only data for Virtual ethernet for this MTU size.

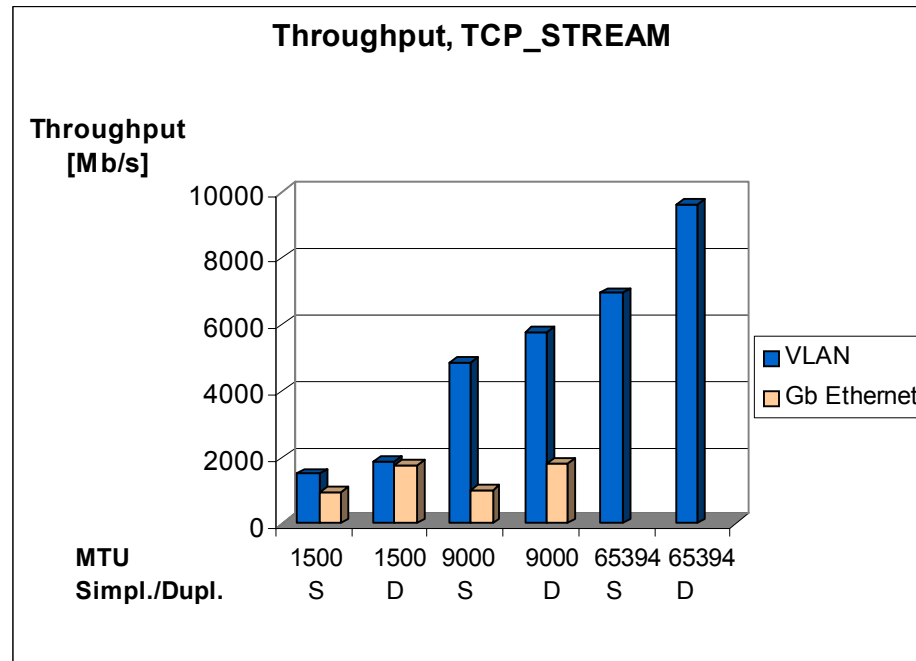


Figure 7-15 Throughput of Virtual LAN and Gb ethernet with TCP_STREAM

Findings of throughput for Virtual LAN and Gb ethernet

The Virtual Ethernet adapter has higher raw throughput at all MTU sizes.

On MTU 9000, the throughput difference is very large (four to five times) because the Gigabit adapter is running at wire speed (989 Mbit/s user payload) while the virtual Ethernet can run much faster as it is limited only by CPU and memory-to-memory transfer speeds.

Processor consumption of VLAN versus Gb Ethernet

This measurements use the same setups as shown in Figure 7-13 and Figure 7-14 with the same TCP_STREAM workload. Now, the processor consumption is recorded for different MTU sizes, in both simplex and duplex mode.

Results of comparing processor consumption

As shown in the measurement before, the throughput of the Virtual LAN is higher than the throughput of Gb Ethernet. So, to compare the processor consumption, this is *normalized to 1 Gb throughput* for both, Virtual LAN and Gigabit ethernet.

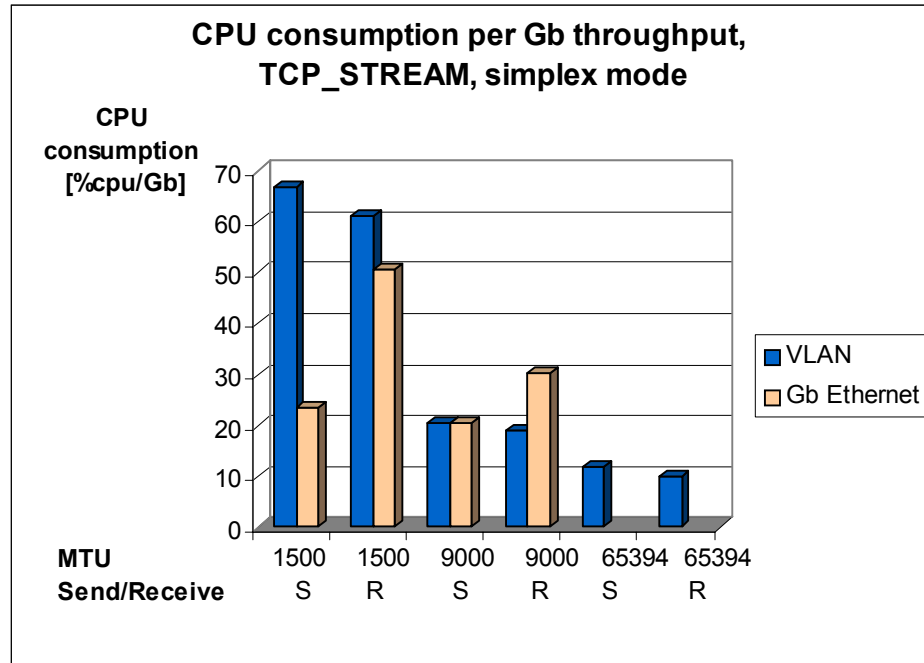


Figure 7-16 Processor consumption with TCP_STREAM, simplex mode

The results are split to two charts. One is simplex and one duplex mode.

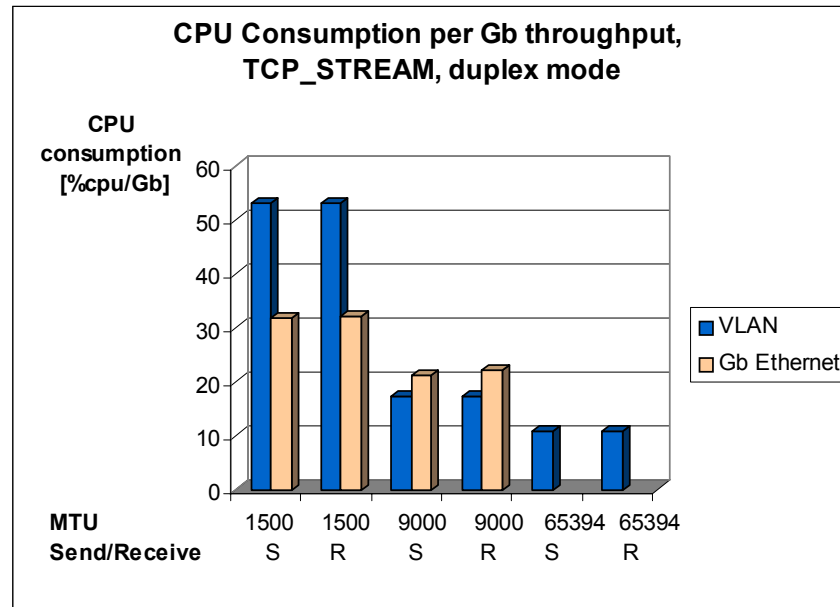


Figure 7-17 Processor consumption with TCP_STREAM, duplex mode

Findings of comparing processor consumption

The difference in processor consumption between the Virtual Ethernet and the Gb Ethernet adapter when using MTU 1500 is the effect of having the attributes `large_send` and `checksum_offload` enabled on the physical adapter. These two features reduce the CPU utilization for Gb ethernet, but they are not available on Virtual Ethernet.

Comparing transaction rate and latency

This measurements also use the setups shown in Figure 7-13 and Figure 7-14. The used workload is now TCP_RR to get a value for number of transactions and latency. TCP_RR is used with two different parameters for the number of sessions (1 and 20), which is a measure for different workloads.

Results of transaction rate and latency

The results are presented in two charts. Figure 7-18 show the transaction rate for MTU size of 1500 and 9000 and for 1 and 20 sessions. Figure 7-19 shows the latency for the same parameters.

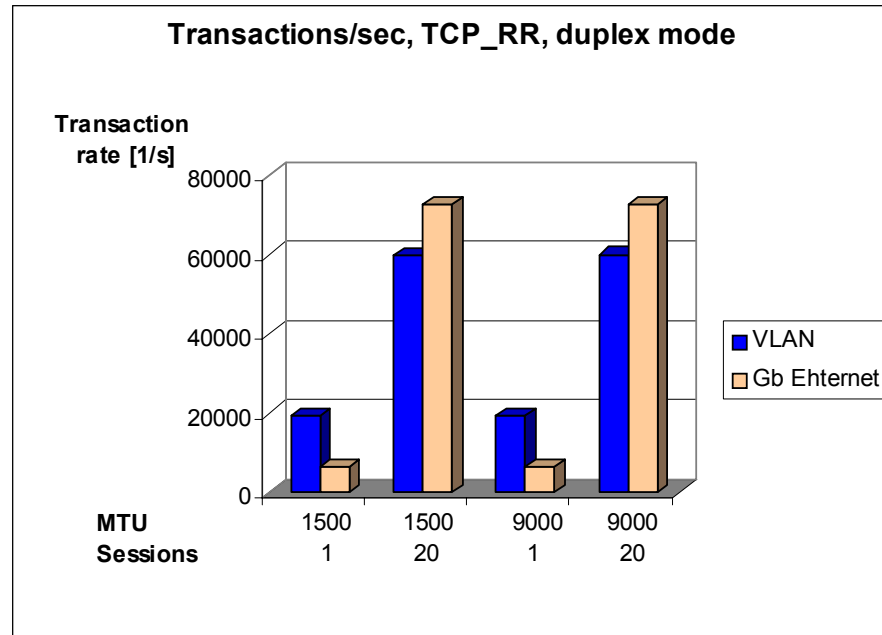


Figure 7-18 Transaction rate at different MTU sizes and 1/20 sessions

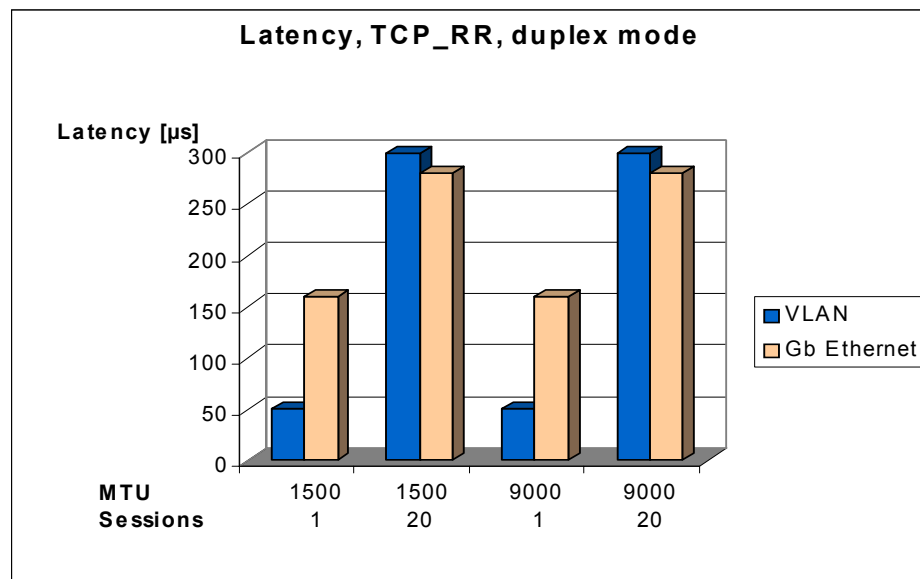


Figure 7-19 Latency at different MTU sizes and 1/20 sessions

Findings of transactions and latency

The Virtual Ethernet has lower latency for light workloads than the Gb ethernet adapter. The reason for this is because the ethernet adapter has interrupt coalescing enabled by default (ODM attribute `intr_rate=10000`). This adds latency to the adapter's single session test, but it helps reduce CPU requirements for higher transaction rate workloads like the 20 session test, which is why the throughput is similar at 20 sessions. The latency can be reduced by disabling interrupt coalescing (set the adapters `intr_rate=0`). The Virtual Ethernet does not support any method of interrupt coalescing

The Gb ethernet has lower latency in heavy workloads because interrupt coalescing is enabled by default on the adapter.

7.3.7 Virtual Ethernet in single-threaded and SMT mode

A new feature of the POWER5 processor is Simultaneous Multithreading (SMT) which presents the kernel with two logical processors per virtual or dedicated processor as described in Chapter 5, "Simultaneous Multi-Threading (SMT)" on page 89. This measurement shows the performance gain of SMT for Virtual Ethernet. The setups as shown in Figure 7-13 on page 194 can be used again. For this comparison, both workloads TCP_STREAM and TCP_RR are used.

Because the absolute SMT data was shown before (see Figure 7-15 on page 195), the following charts show the *gain of throughput in percent*, comparing SMT to single-threaded mode. Figure 7-20 presents the results for TCP_STREAM and Figure 7-21 the results for TCP_RR.

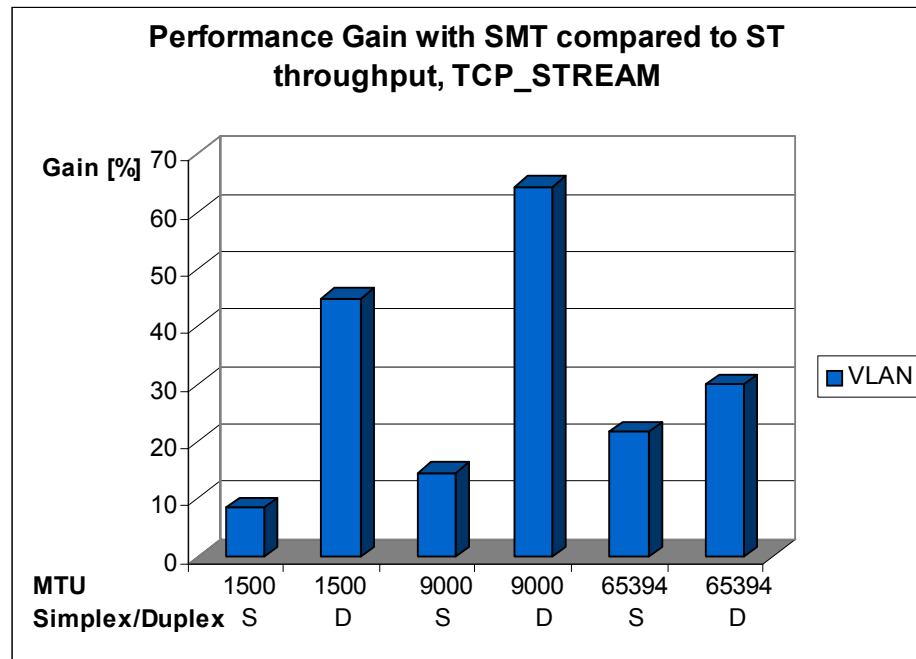


Figure 7-20 Performance gain with SMT, TCP_STREAM

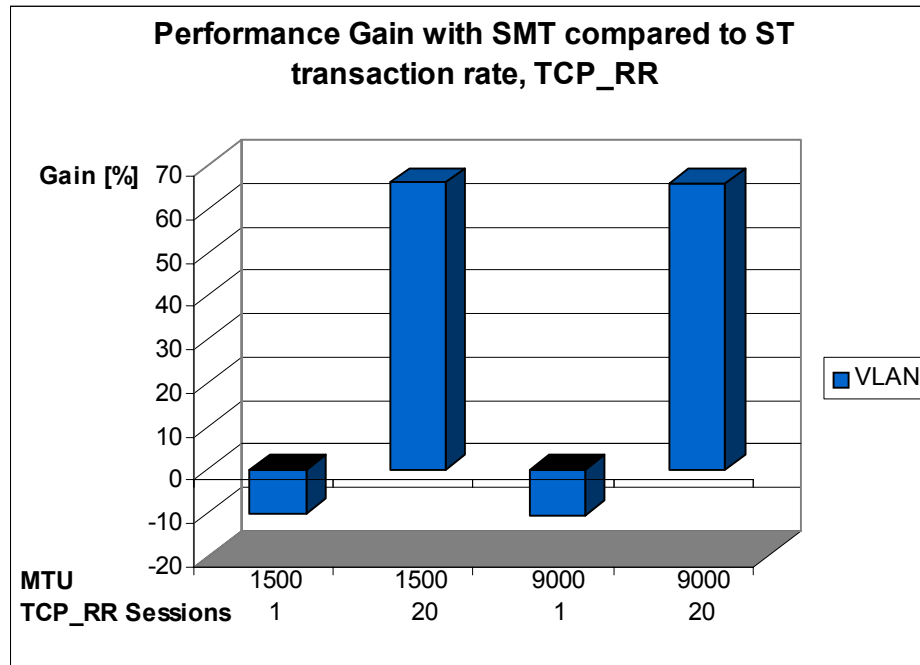


Figure 7-21 Performance gain with SMT, TCP_RR

Findings of single-threaded and SMT mode performance

The Virtual Ethernet adapter benefits from SMT because it is not limited by media speed and takes advantage of the extra available processor cycles.

The reason for negative scaling on Figure 7-21 when SMT is enabled is due to the fact that at very small workloads, which is the case when having only one TCP_RR session, running in single-threaded mode is more efficient. With SMT on, the system disables the second thread when the load on the system is light, but checks periodically to determine if it needs to reactivate it. This checking, disabling and enabling of the second thread tends to affect the latency of the TCP_RR transactions, thus reducing throughput.

7.3.8 Virtual Ethernet implementation guidelines

Because there is little experience with Virtual LANs until now, we will now try to present some rules of thumb for designing Virtual LANs.

Important: The following recommendations have no guarantee for good performance, they are given more as suggestions.

1. Know your environment and the network traffic
2. Choose the MTU size as high as it makes sense for the network traffic in the Virtual LAN.
3. Use a MTU size of 65394 if you expect large amount of data to be copied inside your Virtual LAN.
4. Keep tcp_pmtu_discover set to its default value (active discovery).
5. If the VLAN is to use a Shared Ethernet Adapter to forward packets to an external connection, set the MTU size of the virtual adapter in the client partition to the value used for the definition of the SEA physical adapter on the Virtual I/O server.
6. Do not turn off SMT unless your applications demand it.
7. The throughput in Virtual LANs scale linear with processor entitlements, so there is no need to dedicate processors to partitions because of Virtual LAN performance.

7.4 Shared Ethernet adapter functionality

7.4.1 Introduction

For implementing a virtual LAN, a Virtual I/O Server is not needed necessarily. Virtual ethernet adapters can communicate with each other via the Hypervisor without the functionality of the Virtual I/O Server.

The Virtual I/O Server is needed if *virtual adapters* need to communicate with a *physical adapter*. It can logically connect one or more Virtual Ethernet adapters to one or more physical ethernet adapters. This sharing of a physical adapter to multiple (or just one) virtual adapters is done by an internal implementation of a Layer2 Bridge.

Bridge implementation

Figure 7-22 shows the bridge functionality of the Virtual I/O Server.

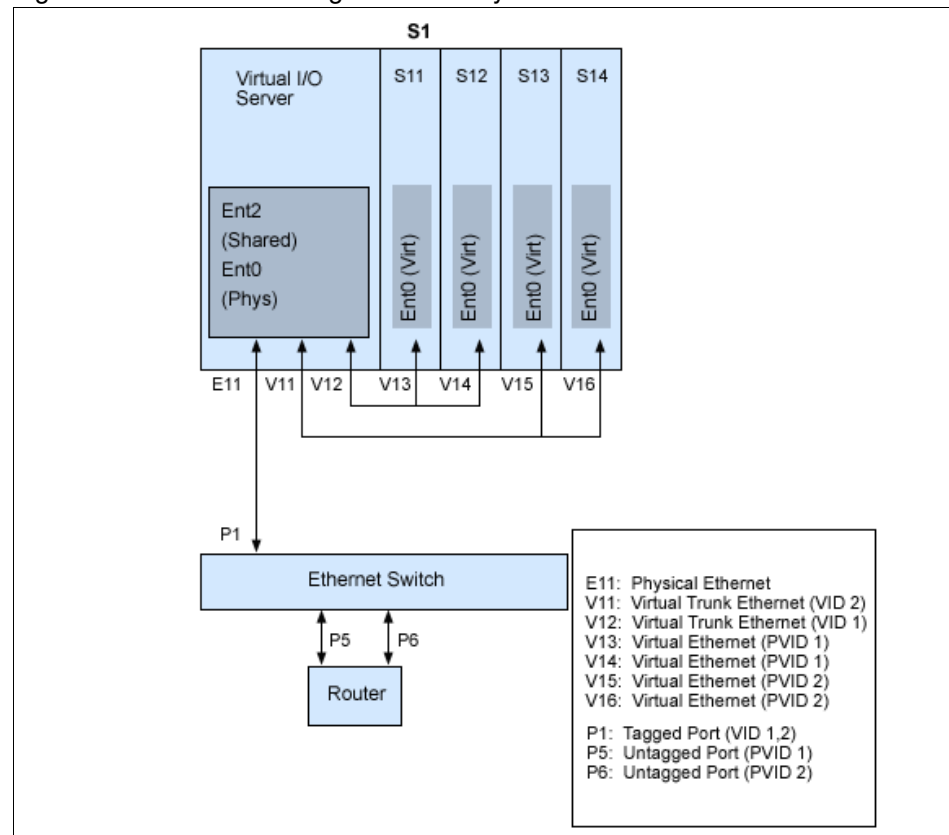


Figure 7-22 VIO Server bridging between an external network and internal VLANs

The bridge interconnects the logical and physical LAN segments at the network interface layer level and forwards frames between them. The bridge performs the function of a MAC relay (OSI-layer 2), and is independent of any higher layer protocol. Figure 7-23 chose a close up view of the Virtual I/O server (The left-most partition in Figure 7-22).

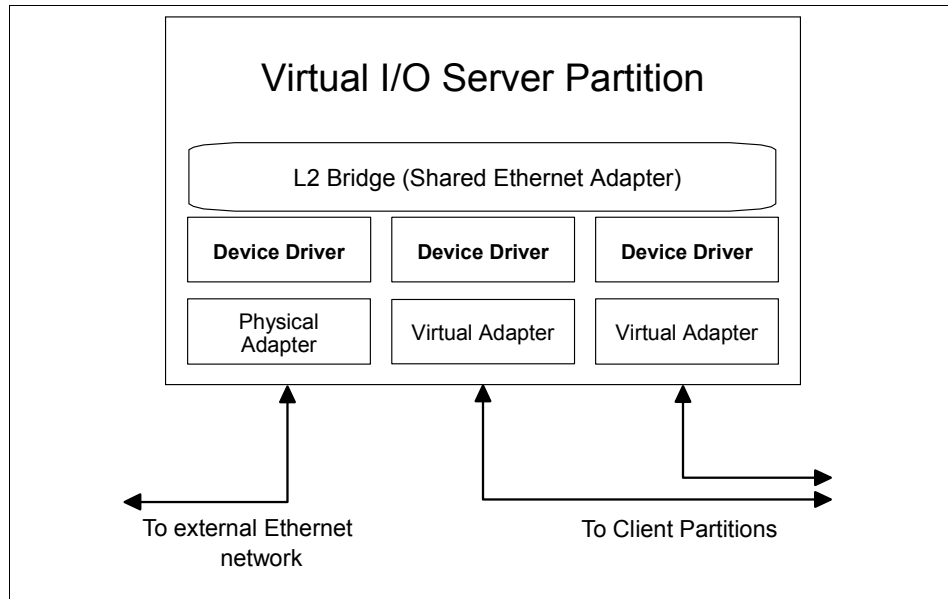


Figure 7-23 Sharing a (physical) ethernet adapter on OSI-Layers

The bridge is said to be transparent to the Internet Protocol (IP) layer. That is, when an IP host sends an IP datagram to another host on a network connected by a bridge, it sends the datagram directly to the host and the datagram “crosses” the bridge without the sending IP host being aware of it.

The I/O Server offers broadcast and multicast support. ARP (Address Resolution Protocol) and NDP (Neighbor Discovery Protocol) are also working across a shared ethernet adapter.

The I/O server does not reserve bandwidth on the physical adapter for any of the VLAN clients that sends data to the external network. Therefore, if one client partition of the I/O server sends data, it can take advantage of the full bandwidth of the adapter, assuming the other client partitions do not send or receive data over the network adapter at the same time .

7.4.2 Performance measurements with the Virtual I/O Server

This topic shows some measurements, done with the Virtual I/O Server, using the Shared Ethernet Adapter (SEA).

General comments to the measurements

This measurements are done under the same conditions as described in Section “General comments to the measurements” on page 188.

The operating system running on all partitions is AIX 5L Version 5.3 (GA code). The results of the measurements could vary if they will be repeated at a later time with other versions of operating system and firmware.

The hardware used for testing was a 4-way POWER5 based server.

Unless otherwise mentioned, all partitions are running in SMT mode.

7.4.3 Throughput and processor utilization

Figure 7-24 shows the setup of the experiment.

The communication path starts on a 1-way partition (LPAR 1) with a dedicated processor, connected via a Virtual LAN adapter to the Hypervisor and to the Virtual LAN adapter of the Virtual I/O Server, which “bridges” the Virtual Ethernet adapter to a Gb ethernet adapter that is connected via a gigabit ethernet network to a 2-way Power4+ based server.

The Virtual I/O Server runs on a partition with a dedicated 1.65 GHz processor (with higher clock speed, the throughput of the I/O Server would increase).

First, the workload TCP_STREAM as described in “Description of the performance tests and tools” on page 188 is used to examine the throughput.

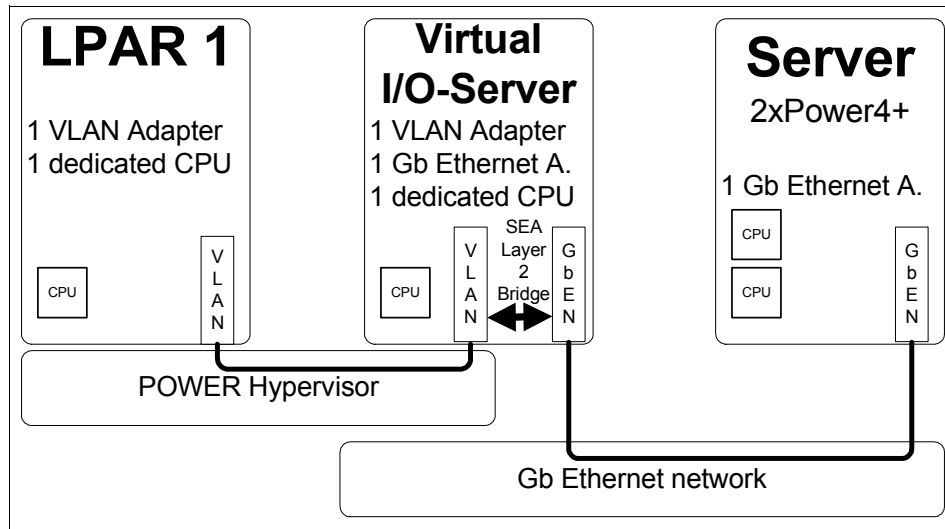


Figure 7-24 Setup for I/O Server performance measurements

Note: The measurements are not done with a gigabit ethernet switch. Instead, a physical point to point connection (crossover cable) was used so there is no falsification of the measurement due to the internal behaviour of a real switch.

Virtual I/O Server performance results

Figure 7-25 on page 207 and Figure 7-26 on page 208 show the results measured on the Virtual I/O Server. Figure 7-25 shows the throughput of the Virtual I/O Server at MTU sizes of 1500 and 9000 in both modes, simplex and duplex. Notice that this test has *maximized the line speed of the gigabit ethernet*. Therefore, the limitation is the physical network media speed (1Gb simplex or 2Gb duplex).

Figure 7-26 presents the utilization of the processor in the Virtual I/O Server. To provide a better comparison of processor utilization versus MTU size and simplex/duplex modes, the utilization is normalized to 1Gb data throughput.

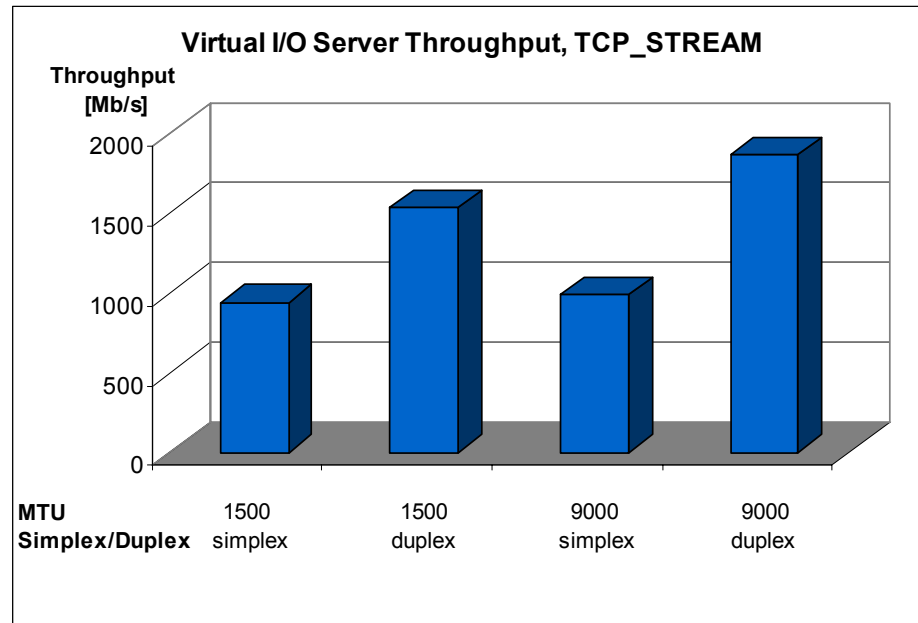


Figure 7-25 Throughput of the Virtual I/O Server

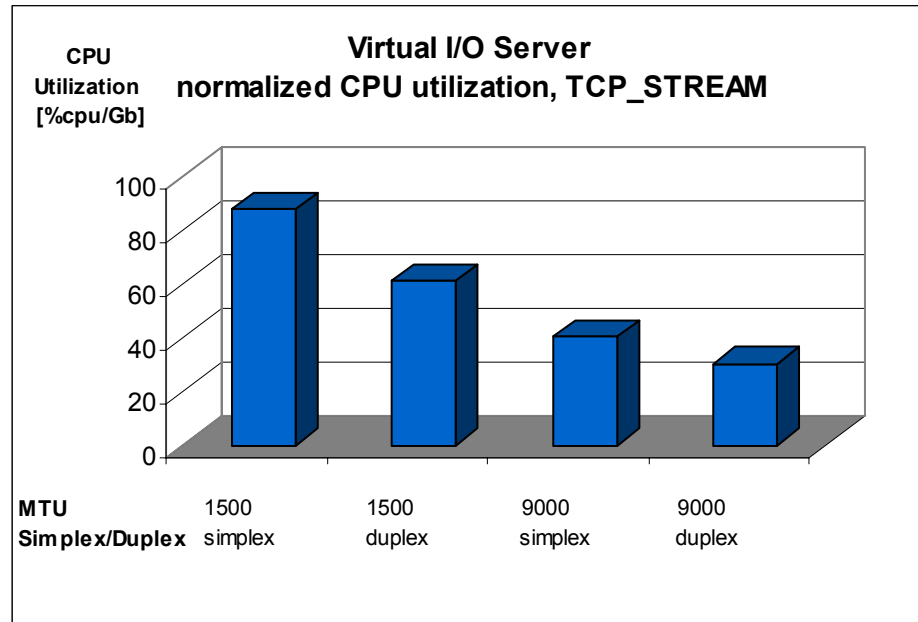


Figure 7-26 Processor utilization of the Virtual I/O Server

Findings of Virtual I/O Server performance

- The shared ethernet adapter allows the adapters to stream data at media speed as long as it has enough processor entitlements.

–

Author Comment: In Fig 7-26, how can we explain that the normalized CPU utilization is lower in full duplex than in simplex?
next paragraph: Higher than what?

- Processor utilization per gigabit of throughput is higher with the shared ethernet adapter as it has to receive from one end and send it out the other end and because of the bridging functionality in the Virtual I/O Server

7.4.4 Virtual I/O Server request/response time and latency

In this test, the workload TCP_RR is used to determine the difference in transaction rate and the latency when using Shared Ethernet Adapter (SEA) versus dedicated Gigabit Ethernet adapter of the POWER4+ server.

The measures for SEA use the setup showed in Figure 7-24 on page 206, with traffic exchanged between LPAR1 and Server.

The measure for dedicated Gigabit ethernet the setup presented in Figure 7-27, where the virtual I/O server is by-passed and the traffic flows directly from LPAR 1 to the Server through a physical network.

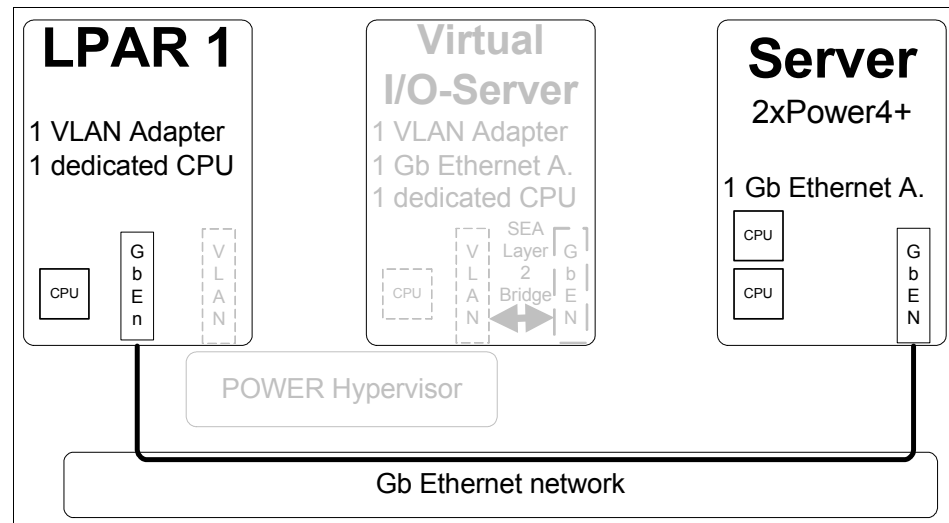


Figure 7-27 Dedicated connection between a partition and an external server.

Results of request/response time and latency

The following figures show the results of the TCP_RR benchmark. Figure 7-28 and Figure 7-29 show the number of transactions that were done by Shared Ethernet Adapter and the Gb ethernet.

Note that the values shown for one session are both limited by the default setting of the Gb Ethernet adapter's interrupt coalescing value. The Gb Ethernet adapter has interrupt coalescing enabled by default (`intr_rate=10000`) because this helps reduce CPU utilization at higher transaction rates. However, this adds latency when only a single transaction is running due to delaying the interrupt. Some workloads with small packets and light workload may benefit from disabling the interrupt coalescing on the Gigabit adapter.

Figure 7-28 and Figure 7-29 respectively shows the transaction rate when running the TCP_RR benchmark with 1 and 20 sessions.

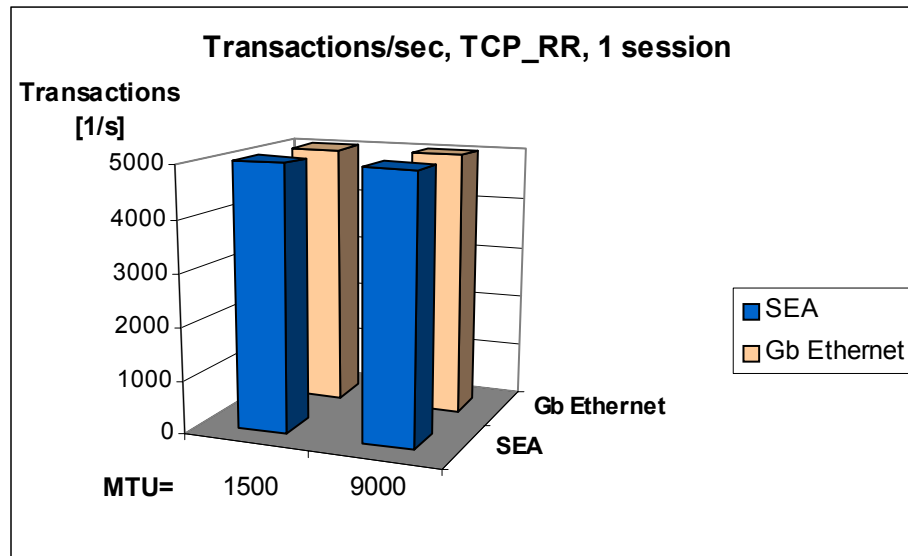


Figure 7-28 Transaction rates, TCP_RR, 1 session

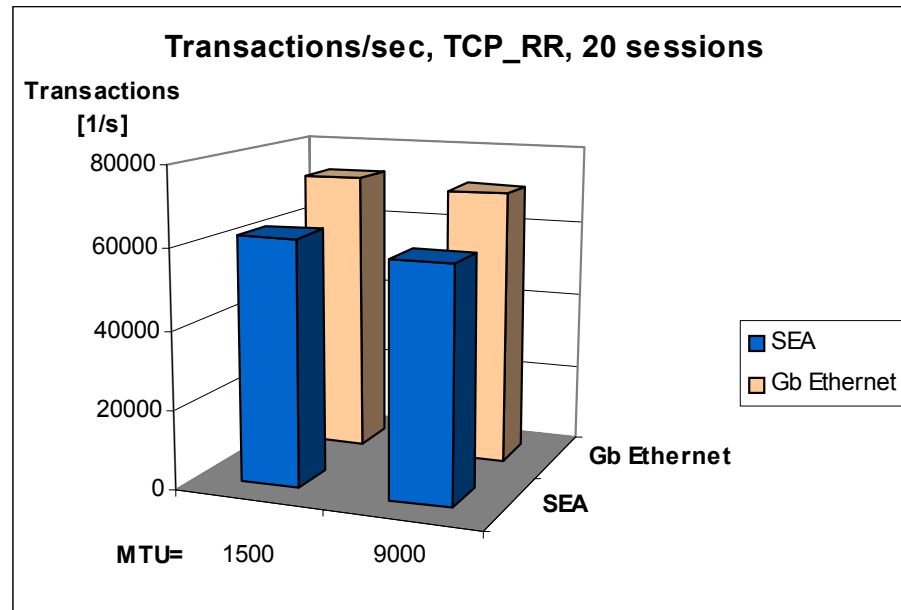


Figure 7-29 Transaction rates, TCP_RR, 20 sessions

Next is the examination of the latency in that environment. Latency is measured with the same parameters as the transaction rate. Figure 7-30 and Figure 7-31 show the differences between the Shared Ethernet adapter and Gb ethernet and the increasing latency, if the load grows to 20 sessions.

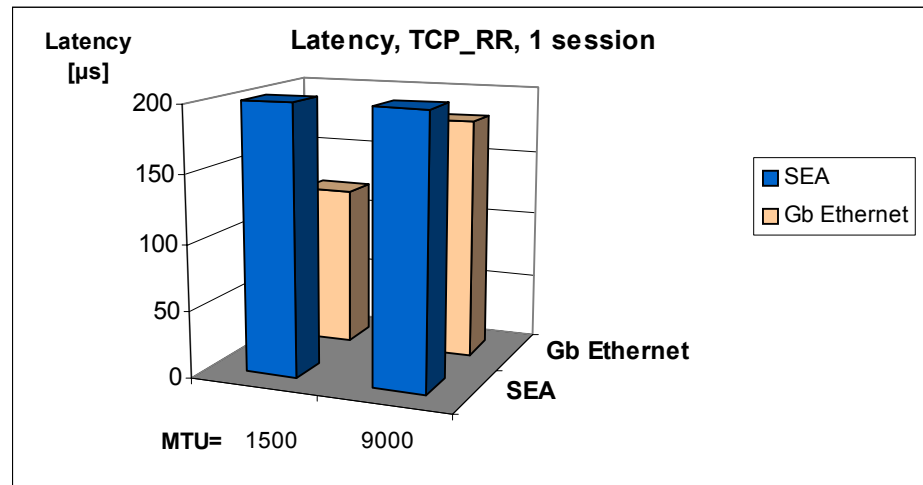


Figure 7-30 Latencies, TCP_RR, 1 session

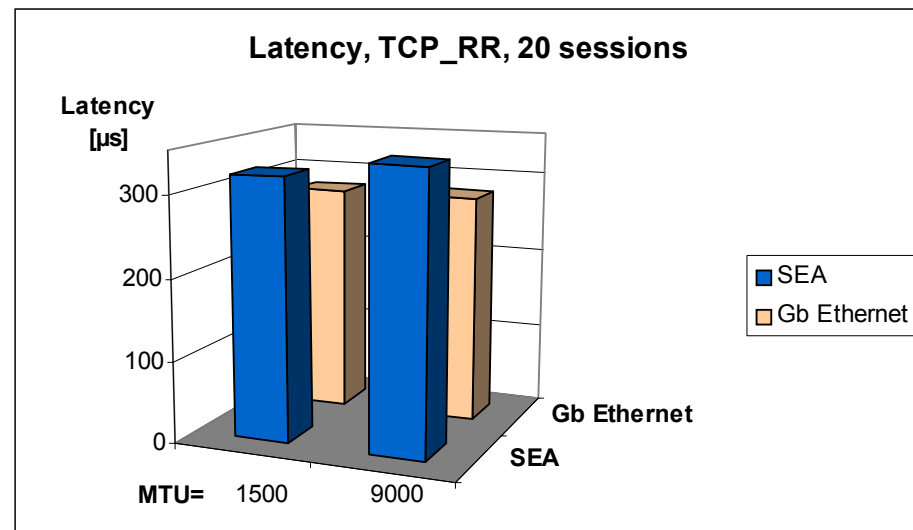


Figure 7-31 Latency, TCP_RR, 20 sessions

7.4.5 Implementation Rules of Thumb

Sizing a server can be somewhat complex and time consuming. Furthermore, it can be performed with more or less accuracy, depending on the amount of data you can collect about the resources requirements of your applications.

We will attempt to present some rules of thumb for designing a Virtual I/O Server. The intent of the rules of thumb is to give some very quick and simple sizing guidelines which may be simple enough for initial sizing when very little data is available about the application requirements.

Later, an I/O server could have its partition size increased or decreased to adjust for variations in the actual workload seen on the machine and based on the CPU utilization measured on the I/O server across a busy part of the workday. Because of the virtualization features of the hardware, the machine resources can be adjusted to meet the demands of the IO server. Section 7.4.6, “Basis for finer Shared Ethernet Adapter I/O server sizing” on page 215 provides a more accurate methods to adjust the VIO server resources.

Important: The following recommendations are no guarantee for good performance, they are given more as suggestions.

1. Know your environment and the network traffic
2. For the most demanding network traffic between Virtual LANs and local networks, use a dedicated network adapter.
3. For optimal performance, use dedicated processors for the Virtual I/O Server.
4. Choose 9000 for the MTU size or what makes sense for your network traffic.
5. If an application requires the least amount of network latency possible, avoid using a shared Ethernet adapter

Network rate, Rules of Thumb

Network throughput rules of thumb for easy estimating are listed in Table 7-3 on page 213. The speed numbers are a bit conservative but rounded down for easy estimating. These numbers are for POWER 5 systems with PCI-X slots.

Table 7-3 Network streaming rates

| Adapter speed | Throughput: MBytes/second | |
|------------------|---------------------------|-------------|
| | Simplex | Full Duplex |
| 10 Mbit Ethernet | 1 MB/s | 2 MB/s |

| Adapter speed | Throughput: MBytes/second | |
|--|---------------------------|-------------------------------------|
| | Simplex | Full Duplex |
| 100 Mbit Ethernet | 10 MB/s | 20 MB/s |
| 1000 Mbit Ethernet (Gigabit Ethernet) | 100 MB/s | 150 MB/s (1.5X the simplex rate) |

NOTE: Consult PCI Adapter Placement Reference for placement rules and/or limitations.

Author Comment: no info about squadron in PCI Adapter Placement Reference, put web page ref??? there is no web page that can be referenced in the Information center!

CPU required Rules of Thumb

Because Ethernet running MTU size of 1500 bytes consumes more CPU cycles than Ethernet running Jumbo frames (MTU 9000), the rules of thumb are different for each. In round numbers, the CPU utilization for large packet workloads on jumbo frames is about half the CPU required for MTU 1500.

MTU 1500

A basic rule of thumb is to provide 100% of one Power5 processor (1.65 Ghz) per Gigabit Ethernet to drive it to maximum bandwidth. This would translate into ten 100 Mbit Ethernet if using smaller networks.

For example, if two Gigabit Ethernet adapters will be used, then up to two processors should be allocated to the partition for example.

MTU 9000 (jumbo frames)

Rule of thumb is 50% of one Power5 processor (1.65 Ghz) per Gigabit Ethernet to drive to maximum bandwidth.

The processing power needed to transfer data over a network depends mainly on the number of packets to be handled. If you network traffic contains a lot of small transactions that do not take advantage of the jumbo frame payload, but use small packets, then you should plan on one full CPU to drive each Gigabit Ethernet adapter. (Jumbo frames do not help the small packet workload case).

7.4.6 Basis for finer Shared Ethernet Adapter I/O server sizing

A finer sizing of the I/O Server for the Shared Ethernet Adapter (SEA) component involves the following steps:

1. Defining the target bandwidth (e.g MB per second) or transaction rate requirements (e.g. operations per second)

These will be the target performance of the configuration and must be determined from your workload requirements

2. Defining the type of workload (Streaming or transaction oriented)
3. Identifying the MTU size that will be used (1500 or jumbo frames)
4. And finally, define the Virtual I/O server partition configuration (number of processors, number of I/O adapters, run the SEA in single-threaded or multi-threaded mode...) by using the tables provided in this section that shows:

- the throughput rates that various Ethernet adapters can provide, and
- the CPU cycles required per Byte of throughput or per transaction.

The first consideration is to determine the bandwidth target on the physical Ethernet side of the I/O server, as this will determine the rate that data can be transferred between the I/O server and the client partitions. Once the target rate is known, the proper type and number of network adapters can be selected. For example, various speed Ethernet adapters could be use, 10 Mbit, 100 Mbit or Gigabit. One or more adapters could be used on individual networks or they could be aggregated using port aggregation.

Another consideration is the type of workload, be it streaming of data for workloads like file transfer or data backup or small transaction workload like remote procedure calls. The streaming workload is mainly dominated by large full size network packets and associated small TCP Acknowledgement packets. Transaction workloads are typically involve smaller packets or may involve small requests, like a URL, and a larger response, like a web page. It will be common that an I/O server will need to support streaming and small packet I/O during various periods of time. In that case, the sizing should be approached from both models and the larger sizing used.

A secondary factor is the MTU size of the network adapters. The standard Ethernet MTU is 1500 bytes (1518 bytes on the wire). Gigabit Ethernet can also support MTU 9000 byte "jumbo frames". The larger jumbo frames can reduce the CPU cycles considerably for the streaming types of workloads. However for small workloads, the larger MTU size will not help reduce CPU cycles. In many cases, the MTU choice is driven by the existing network infrastructure, and cannot be freely chosen according to the application requirements.

Another issue that will affect the CPU cycles used is if the shared Ethernet device is configured to run with threads (threaded) or non-threaded. The threaded mode is mainly used when VSCSI will be run on the same IO server partition. The threaded mode help ensure that VSCSI and Shared Ethernet can share the CPU resource fairly. However, threading adds more instruction path length, thus using more CPU cycles. If the IO server partition will be dedicated to running shared Ethernet devices (and associated virtual Ethernet devices) only, then they should be configured with threading disabled in order to run in the most efficient mode. These two configurations are shown in the CPU sizing later in this section.

Important: The “thread” concept discussed here is the AIX software threading. It is not the Power 5 hardware feature that allows to run the virtual processors in Single-Threaded or Simultaneous Multi-Threading mode.

Section Section 7.4.7, on page 223, explains how to activate the threading mode.

Once the work load and type of adapters have been chosen, it must be determined how much CPU power is required to move data through the adapter(s) at the desired rate. The networking device drivers are CPU intensive. Small packets can come in at a faster rate and use more CPU cycles than larger packet workloads. Larger packet workloads are normally limited by network wire bandwidth and come in at a slower rate, thus requiring less CPU than small packet workloads for the amount of data transferred.

This section provides information on bandwidth for various Ethernet adapters, the CPU cycles required on the IO Server to move these packets through the IO server, and the formulas to use this figures to compute the server sizing.

In addition, there are three other configuration issues to address when planning for an IO server:

- ▶ whether or not the IO server is run as a dedicated partition or as a micro-partition,
- ▶ if in a micro-partition, what should be the processor entitlement,
- ▶ and the amount of memory allocated to the I/O server.

These will be discussed at the end of this section.

Adapter Sizing

Table 7-4 and Table 7-5 provide approximate throughput rates for various Ethernet adapters and MTU sizes and can be used to determine how many adapters and what speed adapters would be needed to configure an IO server, if you know the desired throughput rate of the application partitions.

The tables list maximum possible network payload speeds. These are user payload data rates that can be obtained by sockets based programs for applications that are streaming data (one program doing send() calls and the receiver doing recv() calls over a TCP connection. The rates are a function of the network bit rate, MTU size (frame size), physical level overhead like Inter-Frame gap and preamble bits, data link headers, and TCP/IP headers and assume a Gigahertz speed CPU. These are best case numbers for a single LAN, and may be lower if going through routers or additional network hops or remote links.

Note, In tables below, Raw bit rate is the physical media bit rate and does not reflect physical media overhead like Inter Frame gaps, preamble bits, Cell overhead (for ATM), data link headers and trailers. These all reduce the effective usable bit rate of the wire.

Single direction (simplex) TCP Streaming rates are rates that can be seen by a workload like FTP sending data from machine A to machine B in a memory to memory test. Note that full duplex media performs slightly better than half duplex media because the TCP Acks can flow back without contending for the same wire that the data packets are flowing on.

Table 7-4 Single direction TCP Streaming rates

| Network type | Raw bit rate (Mbit/s) | Payload rate Mbit/s | Payload rate MBytes |
|--------------------------------|----------------------------|---------------------|---------------------|
| 10 Mbit Ethernet, Half Duplex | 10 | 6 | .7 |
| 10 Mbit Ethernet, Full Duplex | 10 (20 Mbit full duplex) | 9.48 | 1.13 |
| 100 Mbit Ethernet, Half Duplex | 100 | 62 | 7.3 |
| 100 Mbit Ethernet, Full Duplex | 100 (200 Mbit full duplex) | 94.8 | 11.3 |

| Network type | Raw bit rate (Mbit/s) | Payload rate Mbit/s | Payload rate MBytes |
|---|------------------------------|---------------------|---------------------|
| 1000 Mbit Ethernet, Full Duplex, MTU 1500 | 1000 (2000 Mbit full duplex) | 948 | 113 |
| 1000 Mbit Ethernet, Full Duplex, MTU 9000 | 1000 (2000 Mbit full duplex) | 989 | 117.9 |

Two direction (duplex) TCP Streaming workloads have data streaming in both directions. For example a FTP running from machine A to B and another FTP running from machine B to A concurrently. Such workloads can take advantage of full duplex media that can send and receive concurrently. Some media, for example Ethernet in Half Duplex mode, can not send and receive concurrently, thus they will not perform any better (usually worse) when running duplex workloads. Duplex workloads will not scale up at a full two times the rate of a simplex workload because the TCP Ack packets coming back from the receiver now have to compete with data packets flowing in the same direction.

Table 7-5 Two direction (duplex) TCP Streaming rates

| Network type | Raw bit rate (Mbit/s) | Payload rate Mbit/s | Payload rate MBytes |
|--------------------------------|----------------------------|---------------------|---------------------|
| 10 Mbit Ethernet, Half Duplex | 10 | 5.8 | .7 |
| 10 Mbit Ethernet, Full Duplex | 10 (20 Mbit full duplex) | 18 | 2.2 |
| 100 Mbit Ethernet, Half Duplex | 100 | 58 | 7 |
| 100 Mbit Ethernet, Full Duplex | 100 (200 Mbit full duplex) | 177 | 21.1 |

| Network type | Raw bit rate (Mbit/s) | Payload rate Mbit/s | Payload rate MBytes |
|--|---------------------------------|------------------------|------------------------|
| 1000 Mbit Ethernet, Full Duplex, MTU 1500 | 1000 (2000 Mbit full duplex) | 1470 (1660 peak) | 175 (198 peak) |
| 1000 Mbit Ethernet, Full Duplex, MTU 9000 | 1000 (2000 Mbit full duplex) | 1680 (1938 peak) | 200 (231 peak) |

NOTE 1) Peak numbers represent best case throughput with multiple TCP sessions running in each direction. Other rates are for single TCP session.

NOTE 2) 1000 Mbit Ethernet (Gigabit Ethernet) duplex rates are for the PCI-X adapter in PCI-X slots.

NOTE 3) Data rates are for TCP/IP using IPV4 protocol. Adapters with MTU 9000 have RFC1323 enabled.

Processor sizing for SEA on dedicated partition I/O server

The sizing of the number of processors needed to support a SEA adapter can be computed by using the figures listed in Table 7-6, 7-7 , 7-8 and 7-9 .

The sizing provided here is divided into the two workload types, TCP Streaming and TCP Request/Response, for both MTU 1500 and MTU 9000 networks.

The sizing is provided in terms of number of machine cycles needed per byte of throughput or per transaction.

These tables were derived with this formula:

$$\left(\# \text{ CPUs} * \text{CPU_Utilization} * \text{CPU clock frequency} \right) / \text{Throughput rate in bytes per second} = \text{cycles per Byte}$$
or
$$\left(\# \text{ CPUs} * \text{CPU_Utilization} * \text{CPU clock frequency} \right) / \text{Throughput rate in Transactions per second} = \text{cycles per Transaction}.$$

The numbers were measured on a one CPU 1.65Ghz POWER5 processor, running with the default of SMT mode enabled.

For other CPU frequencies, the numbers in these tables can be scaled by the ratio of the CPU frequencies for approximate values to be used for sizing.

For example, for a 1.5 Ghz processor speed, use $1.65/1.5 * \text{cycles per byte or transaction value from the table}$. This example would result in a value of 1.1

times the value in the table, thus requiring 10% more cycles to adjust for the 10% slower clock rate of the 1.5Ghz processor.

To use these values, multiply your required throughput rate (in bytes or transactions) by the cycles per byte or transaction value in the tables below. This will give you the required machine cycles for the workload for a 1.65 Ghz speed. Then adjust this value by the ratio of the actual machine speed to this 1.65 Ghz speed.

Then to find the number of CPUs, divide the result by 1,650,000,000 cycles.

You would need that many CPUs to drive the workload.

In these tables, MByte is 1*1024*1024.

For example, if the I/O server must deliver 200 Mbytes of streaming throughput, it would take $200 \times 1024 \times 1024 \times 11.2 = 2,248,810,240$ cycles/ 1,650,000,000 cycles per CPU=1.42 CPUs. Thus, in round numbers, it would take 1.5 processors in the IO server to handle this workload.

This could be handled then with either a 2 CPU dedicated partition or a 1.5 CPU micro-partition.

Table 7-6 provides the figures to use for Streaming workload when the Threading option is enabled.

Table 7-6 Machine Cycles per Byte for TCP Streaming workload

| Streaming type | MTU 1500 rate and CPU util. | MTU 1500, cycles per byte | MTU 9000 rate and CPU util. | MTU 9000, cycles per byte |
|----------------|-----------------------------|---------------------------|-----------------------------|---------------------------|
| Simplex | 112.8 MByte at 80.6% CPU | 11.2 | 117.8 MByte at 37.7% CPU | 5.0 |
| Duplex | 162.2 MByte at 88.8% CPU | 8.6 | 217.0 MByte at 52.5% CPU | 3.8 |

Table 7-7 provides the figures to use when the Threading option is disabled.

Table 7-7 Machine Cycles per Byte for TCP Streaming workload

| Streaming type | MTU 1500 rate and CPU util. | MTU 1500, cycles per byte | MTU 9000 rate and CPU util. | MTU 9000, cycles per byte |
|-----------------------|-----------------------------------|---------------------------------|-----------------------------------|---------------------------------|
| Simplex | 112.8 MByte at 66.4% CPU | 9.3 | 117.8 MByte at 26.7% CPU | 3.6 |
| Duplex | 161.6 MByte at 76.4% CPU | 7.4 | 216.8 MByte at 39.6% CPU | 2.9 |

Table 7-8 provides the figures to use for Transaction workload when the Threading option is enabled. A "transaction" is a round trip request and reply of size listed in the first column of the table. Table 7-9 provides the figures for disable threading.

Table 7-8 Shared Ethernet with Threading option enabled

| Size of transactions | Transactions/second and IO server utilization | MTU 1500 or 9000, cycles per transaction |
|------------------------------|--|---|
| Small packets (64 bytes) | 59772 TPS at 83.4% CPU | 23022 |
| Large packets 1024 bytes) | 51956 TPS at 80.0% CPU | 25406 |

Table 7-9 Shared Ethernet with Threading option disabled

| Size of transactions | Transactions/second and IO server utilization | MTU 1500 or 9000, cycles per transaction |
|------------------------------|--|---|
| Small packets (64 bytes) | 60249 TPS at 65.6% CPU | 17965 |
| Large packets 1024 bytes) | 53104 TPS at 65.0% CPU | 20196 |

Threading versus. non-threading

From the above tables, it can be seen that the threading option of the shared ethernet adds overhead. It is about 16 to 20% more overhead for MTU 1500 streaming and 31 to 38% more overhead for MTU 9000. The threading option has more overhead at lower workloads due to the threads being started for each packet. At higher workload rates, like full duplex or the request/response workloads, the threads can run longer without waiting and being re-dispatched. The thread option is a per interface option that can be configured by IO server commands. The thread option should be disabled if the shared ethernet is running in a IO server partition by itself (without VSCSI in the same partition).

Sample computation

A very simple can be used to compute this number of CPU.

Author Comment: Add the correct reference to the spreadsheet on the ITSO web page, additional document.

To use this spreadsheet, you only need to enter in column E2 or E3 the value of the traffic you want to support, respectively in MBytes per second or number of transaction. You then read the resulting values in the columns highlighted in blue or green.

Figure 7-32 shows a hardcopy of this spreadsheet.

| | | | | | | | | | | | |
|-----------------------|--|---------------------------|---------------|---------------------------|--------------------|------------------|--------------------|------------------|--------------------|------|---|
| Streaming Traffic: | | Network Throughput (MB/s) | | 200 | | | | | | | |
| Transaction traffic: | | Transactions per seconds | | 60000 | | | | | | | |
| | | | | Processor Frequency (GHz) | | | | | | | |
| | | | | 1.5 | | 1.65 | | 1.9 | | | |
| | | | | Exact # Proc. | Rounded # Proc. | Exact # Proc. | Rounded # Proc. | Exact # Proc. | Rounded # Proc. | | |
| Streaming Traffic: | | Mtu size | Cycles/Bytes | | | | | | | | |
| SEA with threading | | Simplex | 1500 | 11.2 | 1.57 | 2 | 1.42 | 2 | 1.24 | 2 | |
| | | | 9000 | 5.0 | 0.70 | 1 | 0.64 | 1 | 0.55 | 1 | |
| SEA without threading | | Duplex | 1500 | 8.6 | 1.20 | 2 | 1.09 | 2 | 0.95 | 1 | |
| | | | 9000 | 3.8 | 0.53 | 1 | 0.48 | 1 | 0.42 | 1 | |
| | | Simplex | 1500 | 9.3 | 1.30 | 2 | 1.18 | 2 | 1.03 | 2 | |
| | | | 9000 | 3.6 | 0.50 | 1 | 0.46 | 1 | 0.40 | 1 | |
| | | | Duplex | 1500 | 7.4 | 1.03 | 2 | 0.94 | 1 | 0.82 | 1 |
| | | | | 9000 | 2.9 | 0.41 | 1 | 0.37 | 1 | 0.32 | 1 |
| Transaction traffic: | | Packet Size | Cycles/packet | | | | | | | | |
| SEA with threading | | 64 | 23022 | 0.92 | 1 | 0.84 | 1 | 0.73 | 1 | | |
| | | 1024 | 25406 | 1.02 | 2 | 0.92 | 1 | 0.80 | 1 | | |
| SEA without threading | | 64 | 17965 | 0.72 | 1 | 0.65 | 1 | 0.57 | 1 | | |
| | | 1024 | 20196 | 0.81 | 1 | 0.73 | 1 | 0.64 | 1 | | |

Figure 7-32 Number of processor needed to support network traffic

Processor Sizing for SEA on Micro-Partitioned IO server

Creating a micro-partition for an IO server can be done if the IO server is running slower speed networks (for example 10/100 Mbit) and a full CPU partition is not needed. This should probably only be done if the IO Server workload is less than half a CPU or if the workload is very bursty. Configuring the IO server partition as uncapped, can also allow it to use more CPU cycles as needed to handle bursty throughput. For example if the network is only used at night, when other processors may be idle, the IO partition may be able to use other machine cycles and could be created with minimal CPU to handle light load during the day but the uncapped processor could use more machine cycles at night.

If creating a micro-partition for the I/O server, it is suggested to increase the entitlement to accommodate for the extra resources needed by the hypervisor.

Memory Sizing for SEA server

The memory requirements for an Virtual I/O partition that provides the SEA functions only (No VSCSI) are minimal. In general a 512 MB partition should work for most configurations.

Enough memory must be allocated for the I/O server data structures.

For the Ethernet and virtual devices, there are dedicated receive buffers that each device will use. These buffers are used to store the incoming packets, and then they are sent out the outgoing device, so they are very transient.

For physical Ethernet network, the system typically uses 4 MBytes for MTU 1500 or 16 MBytes for MTU 9000 for dedicated receive buffers. For Virtual Ethernet, the system typically uses 6 MBytes for dedicated receive buffers, however this number can vary based on load.

Each instance of a physical or virtual Ethernet would need memory for this many buffers.

In addition the system has a mbuf buffer pool per CPU that is used if additional buffers are needed. These mbufs typically occupy 40 MBytes.

7.4.7 Turning threading of the shared Ethernet adapter on or off

Here are the commands to use to switch the thread mode between enabled and disabled.

1. You first need to log onto the Virtual I/O server.
2. Then you list the virtual adapters to find the SEA, with the **lsdev** command, as shown in Example 7-1

Example 7-1 Looking for shared ethernet adapters.

```
$ lsdev -virtual
name          status      description

ent2          Available  Virtual I/O Ethernet Adapter (1-lan)
vsa0          Available  LPAR Virtual Serial Adapter
ent3          Available  Shared Ethernet Adapter
$
```

3. In this example, the SEA is ent3. You now use the **lsdev** command with other flags to find out the current settings of the adapter. Example 7-2 shows that the thread mode is currently disabled.

Example 7-2 Displaying the SEA adapters attributes

```
$ lsdev -dev ent3 -attr
attribute      value description
user_settable

pvid           100    PVID to use for the SEA device                True
pvid_adapter   ent2    Default virtual adapter to use for non-VLAN-tagged packets    True
real_adapter   ent0    Physical adapter associated with the SEA                    True
thread         0       Thread mode enabled (1) or disabled (0)                True
virt_adapters ent2    List of virtual adapters associated with the SEA (comma separated) True
```

4. Assuming you want to enabled threading, you use the **chdev** command as shown in Example 7-3

Example 7-3 Changing the threading mode.

```
$ chdev -dev ent3 -attr thread=1
ent3 changed
$
```

5. You can now check that the threading mode is enabled, as shown in Example 7-4

Example 7-4 Checking the new threading mode.

```
$ lsdev -dev ent3 -attr
attribute      value description
user_settable

pvid           100    PVID to use for the SEA device                True
pvid_adapter   ent2    Default virtual adapter to use for non-VLAN-tagged packets    True
real_adapter   ent0    Physical adapter associated with the SEA                    True
thread         1       Thread mode enabled (1) or disabled (0)                True
virt_adapters ent2    List of virtual adapters associated with the SEA (comma separated) True
$
```

7.5 Virtual SCSI

Virtual I/O allows the POWER5 to support more partitions than it has slots for I/O devices by enabling the sharing of I/O adapters amongst partitions. Virtual SCSI (VSCSI) enables a partition to access block-level storage that is not a physical resource of that partition, but a physical storage area attached to another partition (the Virtual I/O server).

I/O Server Partition A server partition is a partition that has physically attached I/O devices and exports one or more of these devices to other partitions.

I/O Client Partition A client partition is a partition that has a virtual client adapter node defined in its Open Firmware device tree. The client partition relies on another partition (the server partition) to provide access to one or more block interface devices.

Virtual SCSI adapters The Hypervisor architecture defines two distinct virtual adapters, one being a Virtual SCSI initiator (defined in the I/O client) and the other being a Virtual SCSI target (defined in the I/O server) which implement the SCSI Initiator Port and SCSI Target Port in the SCSI Architecture Model.

The VSCSI design is such that the virtual storage can be backed on the Virtual I/O Server by either an entire physical disk, or only a logical volume on a portion of a physical disk. This virtual disks appears as SCSI disks to the client partition, which gives the system administrator maximum flexibility in configuring partitions. VSCSI support is provided by a service running in an I/O server that uses two primitive functions, Reliable Command / Response Transport and Logical Remote DMA to service I/O requests for an I/O client, such that, the I/O client appears to use the services of its own SCSI adapter. The terms *I/O server* and *I/O client* refer to platform partitions that are respectively servers and clients of requests, usually I/O operations, using the I/O server's physical I/O adapters. This allows a platform to have more I/O clients than it may have I/O adapters because the I/O clients share physical I/O adapters via the I/O server, as shown on Figure 7-33. In this example, the 3 client partitions access data that are

located on different disks, all connected through the same physical adapter to the server partition.

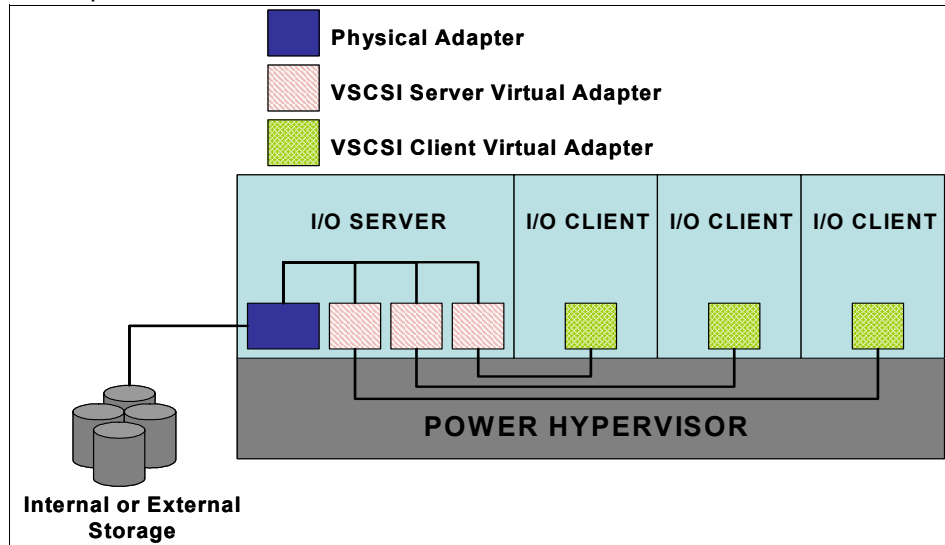


Figure 7-33 Virtual I/O Server and Client Partitions

Virtual I/O provides a high performance I/O mechanism by minimizing the number of times data is copied within the memory of the physical system. The virtual I/O model described herein allows for either zero copy, if data is being retrieved from a physical device and DMAed directly to the memory of the partition using virtual I/O using the redirected DMA described in “Logical Remote Direct Memory Access (LRDMA)” on page 232, or single copy of the data is first moved to the memory space of the I/O server before being DMAed to the I/O client. The IBM Virtual I/O server uses LRDMA for all virtual SCSI data transfers.

7.5.1 Virtual SCSI communication basic concepts

The implementation of virtual SCSI between an I/O Server partition and an IO client partition involves three components: the client SCSI driver, the server SCSI driver, and the interpartition communication between these drivers.

The client and server adapters operate as a pair, in a point-to-point configuration, with the Hypervisor providing the means of communication between the two. The SCSI requests/responses use the SCSI Remote DMA Protocol (SRP, detailed in Section “SCSI Remote DMA Protocol (SRP)” on page 233). The SCSI client emulates for the client operating system a physical SCSI adapter connecting to disks. The client driver accepts requests for storage services from the client operating system, converts those requests into SRP Information Units, then uses interpartition communication facilities to transmit those requests to the server

driver. The server driver completes the requests using a combination of software emulation and services provided by the Virtual I/O server operating system and its physical devices, then converts the results into SRP Information Units and returns those results back to the client driver.

In SCSI terminology, the client virtual SCSI adapter is called the initiator adapter and the server virtual SCSI adapter is called the target adapter.

The target and initiator virtual adapters are created on the HMC during creation (or modification) of the partitions profiles.

The target and initiator adapters are always connected in a point-to-point configuration,

One initiator adapter can connect with at most one target adapter. This target adapter is defined on the HMC in the I/O server partition profile.

A target adapter can provide storage services to multiple initiators but not at the same time. When initiator virtual adapters of clients partitions are created on the HMC, they can be assigned to a target adapter of a Virtual I/O server that is already assigned to other partition profiles.

In Figure 7-34, partitions "I/O Client 1" and "I/O Client 2" have respectively virtual SCSI client adapters C1 and C2, which both have been defined to communicate with the same virtual server S of the "I/O Server" partition. In this example, the communication is established between S and C1. As long as this communication exists, C2 cannot communicate with S, and the second partition cannot use any disks attached to the I.O server physical adapter.

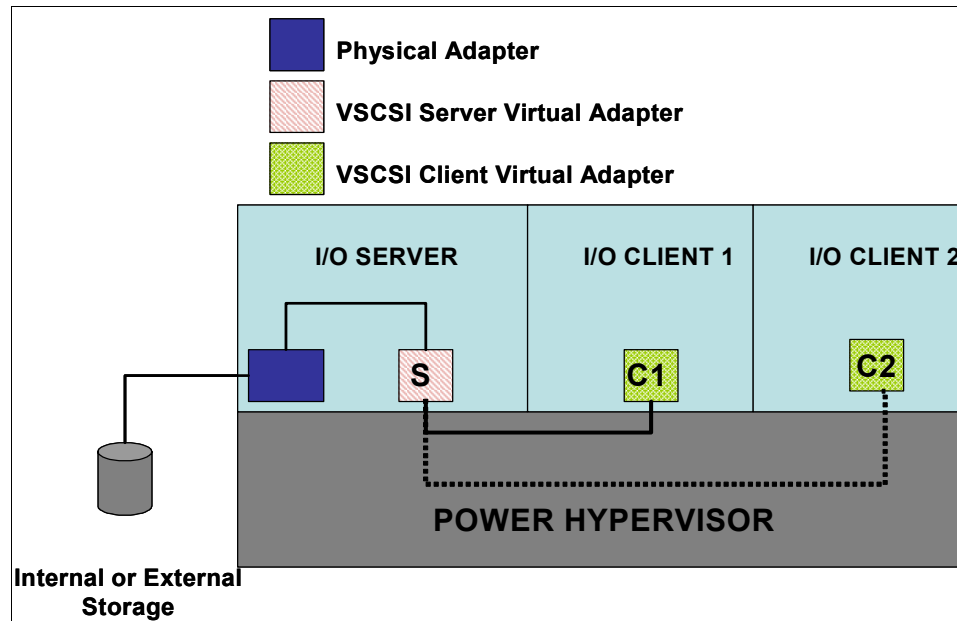


Figure 7-34 Server SCSI adapter possibly connecting to 2 client SCSI adapters

When a C1 disconnects from an initiator driver, if the “I/O Client 1” partition is shut down or if the C1 adapter is put in the “defined” state, rather than the “Available” state, then the Communication between C2 and S can be established.

The virtual SCSI architecture allows a partition to have instances of both client and server drivers. This could be the case when a server partition exports the disks it directly manages through its physical adapters, and boots from disks exported from another server partitions. *We strongly recommend not to use such configurations.* This could lead to deadlocks where two I/O server partitions depends on the each other being running before being able to boot. “Cascaded” devices (virtual devices that are backed by virtual devices) are not supported.

The relationships between the client driver, server driver, peripheral drivers, and the Hypervisor are shown in Figure 7-35 on page 229. In this example, the I/O server partitions exports logical volume as shown by the arrow between the VSCSI target adapter and the Logical Volume Manager. Were it exporting a physical disk, the VSCSI target adapter would communicate directly with the Disk Device Driver.

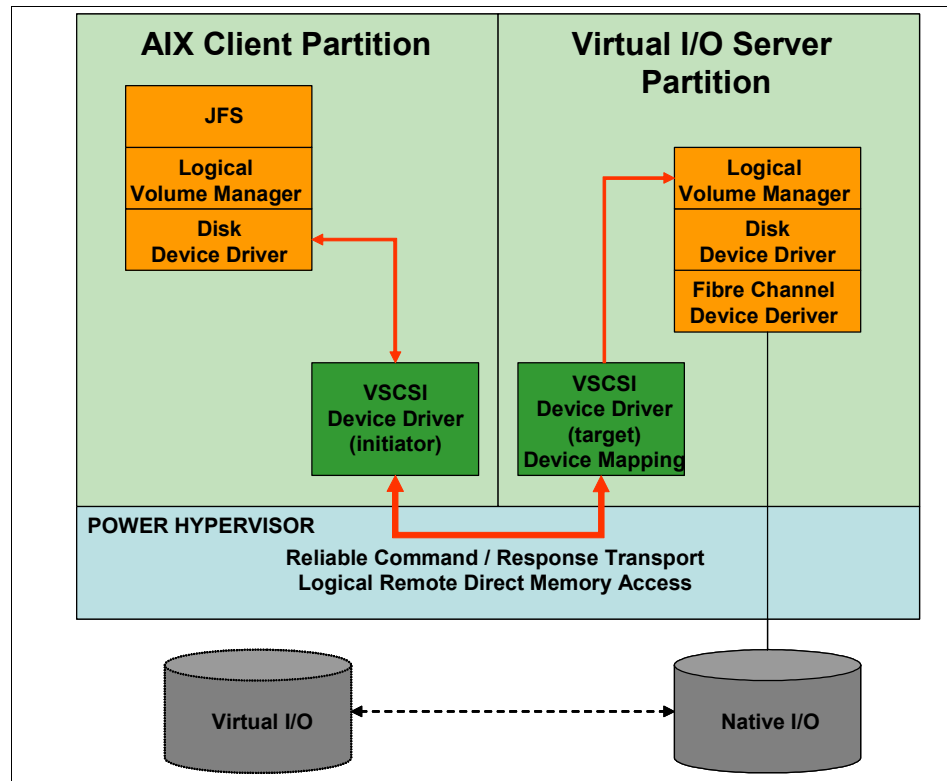


Figure 7-35 Reliable Command / Response Transport and LRDMA

Sections 7.5.2 to 7.5.5 describe in more details the components involved in the deployment of storage using VSCSI.

7.5.2 Server Partition

The virtual SCSI adapter driver on the server partition is a dynamically loadable kernel extension and its entry points are contained in the `devswitch` table. It is the SRP target. The primary function of the target driver is to convert SRP requests from the initiator driver into I/O requests that are forwarded to the device via the native stack. Data is then transferred directly to the client memory using LRDMA.

7.5.3 Client Partition

The virtual client adapter device driver (`vscsi_initdd`) is a dynamically loadable kernel extension and its entry points are contained in the `devswitch` table. It is the SRP initiator. The primary function of the initiator driver is to convert I/O requests

from the peripheral or media device drivers to SRP IUs, then to forward the SRP IU to the SRP target for LRDMA.

The virtual adapter on the client partition is in many ways similar to a physical SCSI adapter. While a typical SCSI adapter has a parallel bus or optical link attached to it, the virtual adapter's link is the Hypervisor's Reliable Command/Response Transport.

For Editors: is it acceptable to use the wording "DASD" below or should that be replaced with another word

7.5.4 Emulated DASD

The Emulated Direct Access Storage Device (DASD) is the virtual disk device that is exported by the I/O server to the client. It can be mapped by the server device driver to

- ▶ either a logical volume, defined on a *slice* of a physical volume
- ▶ or an entire physical disk.

It is seen by the I/O client as a physical direct access device. There can be many emulated DASD devices mapped onto a single physical DASD. The system administrator will create an emulated DASD device by choosing a logical volume and binding it to a VSCSI hosting adapter. The command adding virtual devices will create an ODM entry for the emulated DASD device.

It is expected that most of the SCSI commands targeting an emulated DASD device will be either reads or writes. Reads and writes are serviced by the LVM..

Figure 7-36 on page 231 shows the possible partitioning of a physical disk on the I/O server where there are two logical volumes that support two emulated DASD devices, hdx and hdy. In this example, hdx and hdy could be exported to two different partitions.

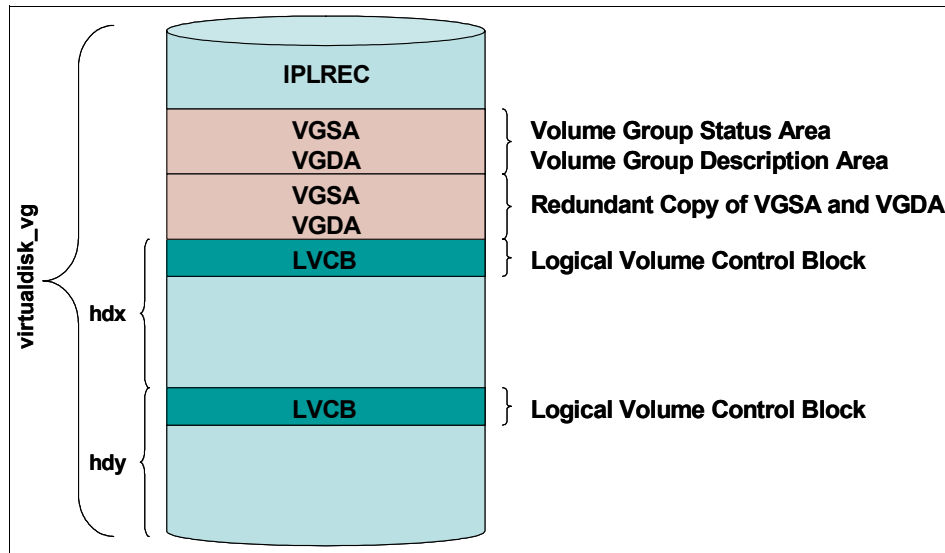


Figure 7-36 Volume group on Virtual I/O Server

7.5.5 Interpartition Communication

Interpartition communication involves the client device node in the Open Firmware device tree of one partition, the server device node in the Open Firmware device tree of another partition, the interpartition communication channel provided by the hypervisor and a communication protocol definition. The interpartition communication use two primitive functions:

Reliable Command/Response Transport

The Reliable Command / Response Transport facility provides ordered delivery of messages between authorized partitions. In order to communicate, a client/server partition pair must establish a Command/Response Queue (CRQ). For a detailed description of the CRQ, see “The Command/Response Queue (CRQ)” on page 178

A CRQ is established during configuration by a virtual SCSI driver, given the presence in the Open Firmware device tree of a virtual SCSI device. The initiator driver registers a response queue and the target driver registers a command queue. Both use the `h_reg_crq` kernel service to call the Hypervisor. The Hypervisor creates a connection between the two partitions through the queues.

Once the queues are established, the virtual SCSI drivers can use the `h_send_crq` kernel service to put queue elements on each other's queues. The initiator driver attempts to queue an element to the target driver's command queue to initiate a transaction. If it is successful, the initiator driver returns, waiting for the interrupt indicating that a response has been posted by the target driver to the initiator driver's response queue.

The client partition only uses the Reliable Command / Response Transport. It does not use the Logical Remote DMA. Since the server partition's RTCE tables are not authorized for access by the client partition, any attempt by the client partition to modify server partition memory would be prevented by the Hypervisor. RTCE table access is granted on a connection by connection basis (client/server virtual device pair).

The target driver is notified via an interrupt that it has received a message on its command queue. The target driver decodes the I/O request and routes it through the server partition's file sub-system for processing. When the request completes, the file sub-system calls the target driver and it packages a response into a queue element that is then queued to the initiator driver's response queue.

Logical Remote Direct Memory Access (LRDMA)

Logical Remote Direct Memory Access (LRDMA) allows for an I/O server to securely target memory pages within an I/O client for virtual I/O operations. The I/O server uses the `hca11()` of the Logical Remote DMA facility to manage the movement of commands and data associated with the client requests. The server driver may use this service if it has a connection established via a Command/Response Queue pair. Virtual SCSI defines two modes of LRDMA:

- Traditional Copy RDMA that involves the I/O server's I/O adapters targeting DMA buffers in the I/O server's memory and having the Hypervisor copy data between that DMA buffer and the I/O client's memory. In Figure 7-37 on page 233, the arrow marked with the letter "A" is an example of using traditional Copy RDMA.
- Redirected RDMA allows for an I/O server to securely target its physical I/O adapter's DMA operations directly at the memory pages of the I/O client. In Figure 7-37 on page 233, Redirected RDMA in the communication represented by the arrow marked with the letter "B".

The platform overhead of Copy RDMA is generally greater than Redirected RDMA, but this overhead may be offset if the I/O server's DMA buffer is being used as a data cache for multiple virtual I/O operations.

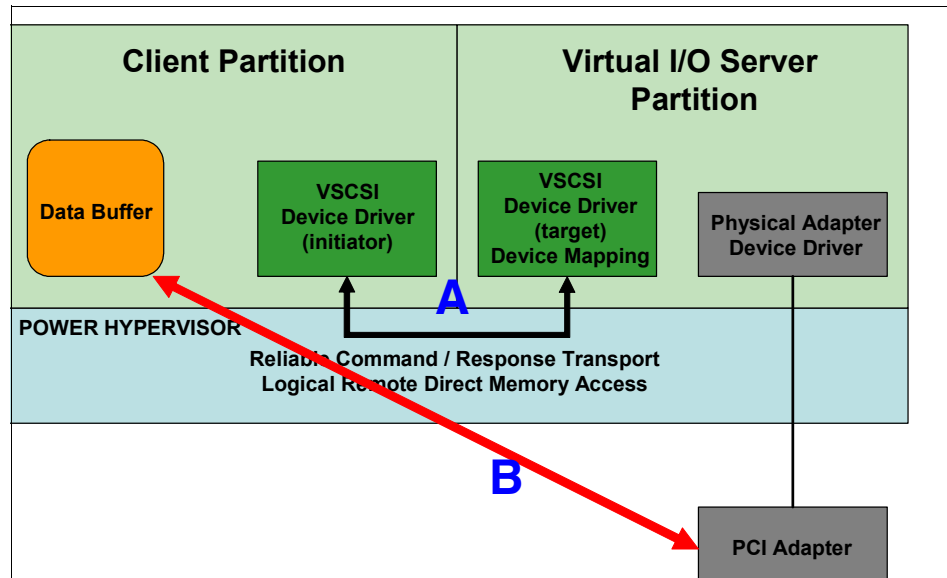


Figure 7-37 Logical Remote Direct Memory Access

LRDMA defines an extended type of TCE table called a Remote DMA TCE table (RTCE). An RTCE is used by the Hypervisor to translate a server partition's Logical Remote DMA hcalls()'s DMA addresses. RTCE tables have extra data to help manage the use of its mappings by server partitions. Note that only the target driver uses the Logical Remote DMA primitives, not the initiator driver. The server partition's RTCE tables are not authorized for access by the client driver.

The use of Redirected RDMA is completely invisible to the I/O client, and has no impact on the VSCSI architecture defined in this document. It is left entirely to the discretion of the I/O server whether it first moves data from a physical device into its own memory before moving (DMAing) the data to the I/O client, or whether the I/O server sets up the I/O request to the physical device in such a way that the physical device DMA's directly to the memory of the I/O client. The I/O server uses the RDMA mode that best suits its needs for a given virtual I/O operation.

The logical remote direct memory service allows the server driver to read and write to a well defined part of the I/O client's memory. This service is unidirectional, i.e. the client driver cannot use the service to write to, or read from, the I/O server's memory.

SCSI Remote DMA Protocol (SRP)

SCSI Remote DMA Protocol (SRP) defines a method of encapsulating SCSI Command Data Blocks (CDBs) and is the protocol used for interpartition communication for Virtual SCSI on IBM @server p5 logical partition. Because

virtual SCSI involves heterogeneous operating systems (AIX 5L and Linux) it is important to implement a common industry standard protocol for communicating I/O operations between partitions. SRP has defined the message format and protocol using an RDMA communication service. The SCSI RDMA Protocol defines the rules for exchanging SCSI information in an environment where SCSI initiators and targets have the ability to directly transfer information between their respective address spaces.

All SRP communication is accomplished via SRP Informational Units (IUs). An IU is an organized collection of data specified by the SRP to be transferred as login data, reject data or a message on an RDMA channel. Thus all SCSI commands, and their associated data and status, are encapsulated in an SRP IU. Note that the protocol used for interpartition communication has no bearing on the makeup of the destination device. The SRP protocol works just as well if the target device is a physical device or a logical device (logical volume).

SRP Memory Descriptor Mapping

The SRP architecture defines a “memory descriptor”, which is a 16-byte structure that identifies a memory segment upon which DMA operations can be performed.

The VSCSI architecture is defined such that DMA operations need never be initiated from the I/O client (from the initiator port.) Since the I/O server's RTCE tables are not authorized for access by the I/O client, any attempt by the I/O client to modify I/O server's memory would be prevented by the Hypervisor. RTCE table access is granted on a connection by connection basis (hosted/hosting virtual device pair), if a I/O client happens to be hosting some other logical device then the partition is entitled to use Logical Remote DMA for the virtual devices that it is hosting.

Memory descriptors sent in IUs defined in this architecture always reference memory in the initiator, and are always used in DMA operations initiated by the target.

SRP initiator ports and SRP target ports shall be determined by both their role during RDMA channel establishment and by the adapter types on which the messages are sent and received. VSCSI Message Formats

Virtual SCSI Flow

An example of a typical interaction between the target and initiator device drivers is a file read from a virtual DASD device. A virtual DASD is a virtual device on the client partition, which is backed by a logical volume exported from a DASD device that is physically attached to the server partition. The client stack considers the initiator driver a SCSI-3 device with access to the virtual DASD.

A typical I/O request involves the following steps:

- (Client) The application program initiates a read() system call to the filesystem (JFS).
- (Client) The filesystem requests a read from the LVM. LVM forms a buf struct with DMA buffer addressing information, as well as DASD block information.
- (Client) The buf struct is passed to the disk device driver which creates a scsi_buf and sends it to the vscsi_initdd driver.
- (Client) The initiator driver takes the information in the scsi_buf and creates an SRP IU. If the I/O request includes data to be transferred, the initiator driver maps the data buffers for DMA.
- (Client) The client builds a CRQ command element containing a pointer to the SRP IU and sends the CRQ command element through the Hypervisor to vscsi_targetdd.
- (Server) The target driver receives an interrupt indicating that an element has been queued to its command queue.
- (Server) The target driver uses the pointer to the SRP IU in the CRQ command element and LRDMA services to copy the SRP IU from the client partition to the server partition's memory.
- (Server) The target driver uses the information in the SRP IU to create a buf struct.
- (Server) The target driver passes the buf struct to the LVM running in the server partition. The request ultimately makes its way to the adapter device driver. This driver calls the usual kernel DMA services, which have been extended to map the buffers for DMA using LRDMA services.
- (Server) When the transaction is complete, the target driver constructs an appropriate SRP response and uses LRDMA services to copy the response to the client's memory. It then builds a CRQ command element containing the TAG or "correlator field" from the original SRP IU and sends the CRQ element through the Hypervisor to the initiator.
- (Client) The initiator driver receives an interrupt indicating that a CRQ element has been queued to its response queue.
- (Client) The initiator driver uses the information in the SRP response to give status back to its child head driver. The head driver passes the results back up to LVM.

7.5.6 Disks considerations

SCSI RESERVE and RELEASE

The VSCSI virtual adapter driver emulates the SCSI RESERVE and RELEASE commands instead of passing them on to the device. That emulation is limited in scope to a single I/O server. When one I/O client wins a reservation on a logical volume, the VSCSI virtual adapter target driver will have to refuse access by other I/O clients to the logical volume. And when the I/O client holding a reservation fails, the VSCSI virtual adapter target driver will have to break the reservation on that logical volume. This will enable configurations where one I/O server provides storage services for multiple I/O clients.

However, this will not provide an adequate emulation of RESERVE and RELEASE for configurations where the same physical storage can be accessed by multiple AIX instances executing in different physical servers. This emulation will not prevent access by the native stack on that I/O server.

Command Tag Queueing

SCSI Command tag queueing refers to queuing commands to a SCSI device. Command tag queueing requires the SCSI adapter, the SCSI device, the SCSI device driver, and the SCSI adapter driver to support this capability. The VSCSI architecture supports command tag queueing.

7.5.7 Redundant Configurations

In order to minimize the adverse impacts that would result from the loss of server partition or physical adapter, a system administrator can use two different ways to create redundant configurations. Each of these techniques will allow a client partition to continue to function while maintenance is being done on the server partition.

Logical volume mirroring

AIX Logical Volume Manager supports mirroring of virtual disks. This mirroring is configured on the I/O Client partition. For every write to a logical volume, the LVM generates a write request for every mirror copy. The system administrator can define two virtual disk devices, either hosted by two distinct I/O servers or two devices on the same I/O server, and mirror the client partition's data on the two devices. Mirroring makes no requirements on either the client, or hosting drivers. It is cost effective and the system configuration is readily understood. Figure 7-38 on page 237 presents a configuration of mirrored virtual disks, backed by physical disks.

Note: The Virtual I/O server does not support mirroring. Each disk (either native or logical volume) exported from the VIO server maps to only one physical disk. The disk mirroring must be defined in the Client I/O partition.

In the same way, the Virtual I/O server does not support data striping over several disks. If Striping is require, it must be defined in the Client partition.

If mirroring is needed, set the scheduling policy to parallel and allocation policy to strict. The parallel scheduling policy will enable reading from the disk that has the least outstanding requests, and strict allocation policy allocates each copy on separate physical volume(s).

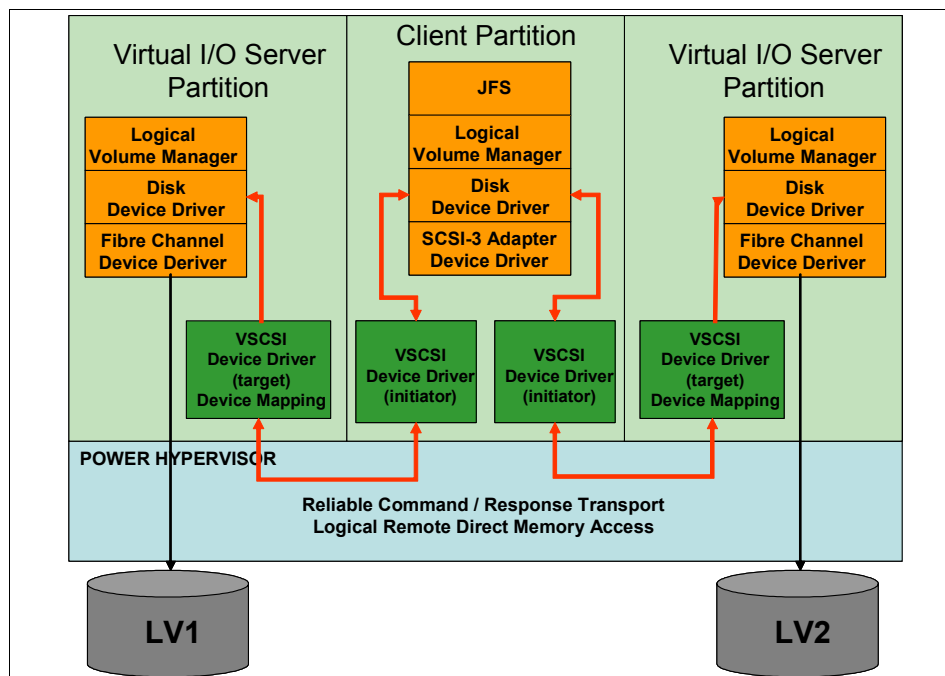


Figure 7-38 Using LVM mirroring for virtual SCSI

Physical Mirroring

Physical mirroring is provided by the physical adapter rather than the operating system. If the physical adapter used in the I/O server to connect to the disk provides RAID support, it can be used along with VSCSI to provide a more reliable storage solution.

Multi-path I/O

Multi-path I/O (MPIO) offers another possible solution to the redundancy requirement. MPIO is a feature of AIX 5L that permits a volume accessed by multiple physical paths to be seen as a single hdisk. It is therefore logically similar to IBM Subsystem Device Driver (SDD), which allows a volume on the TotalStorage® Enterprise Storage Subsystem™ (ESS) that is accessed through multiple paths to be seen as a single vpath disk. However, the SDD logical construct of a vpath disk is above the level of the hdisk, whereas MPIO combines the paths underneath the level of the hdisk. MPIO is intended to support additional disk subsystems besides ESS. These disk subsystems are themselves capable of supporting multiple physical (parallel or fibre SCSI) attachments.

MPIO has numerous possible configuration parameters. A detailed discussion of them is beyond the scope of this redbook. However, to gain the benefits of high availability and throughput that MPIO offers, it is recommended that it be configured with a “round robin” algorithm, “health check” enabled, and a reserve policy of “no reserve”. This allows the best combination of throughput and reliability, since all paths are used for data transfer, and failed paths are detected and reacted to in a timely fashion.

7.5.8 Performance Considerations

Before addressing specific performance topics, let us restate that the first goal of virtualization is not to improve system performance. The primary benefits of virtualization is to lower the TCO¹ of equipment, making a better usage of the overall system resources and lowering the manpower required to operate servers.

Thanks to virtualization, the pSeries can now be used in a way similar to the way mainframes have been used for decades, sharing the hardware between many programs, services, applications or users. Of course, for each of these individual users of the hardware, sharing resources may results in lower performance than having a dedicated hardware. But the overall cost is usually far lower than when dedicating hardware to each user. The decision of using virtualization is therefore a trade-off between cost and performance.

The performance considerations detailed in this section must be balanced against the savings made on the overall system cost. Let us take an example: the smallest physical disk that is available in a Power 5 pSeries is 36 Gigabytes large. A typical operating system requires 4 Gigabytes of disk. If one disk is dedicated to the operating system (rootvg), nearly 90% of this physical disk space is unused. Furthermore. the system disk I/O rate is very often very low.

¹ TCO: Total Cost of Ownership

With the help of VSCSI, it is possible to split the same disk in nine logical disks of 4 Gigabytes each. If each of these disks is used to install the rootvg of one partition, you can support 9 partition operating systems with 9 times less disks (and nine time less SCSI adapters) than on a traditional non virtualized environment. These savings have to be compared with the little extra cost of processing power needed to handled the virtual disks.

Enabling VSCSI result in using extra processing power compared to directly attached disks, due to the processing of extra Hypervisor call. and the paths involved for exchanging I/O requests between the initiator and target adapters, This may not yield the same performance from VSCSI devices as from the dedicated devices. If a partition has high performance and disk I/O requirement, that justify the cost of dedicated hardware, then it is not recommended to use VSCSI. However, partitions with non critical performance and low disk I/O requirements can often be configured to use VSCSI, at a much lower hardware and operating cost. Using a logical volume for virtual storage means that the number of partitions is no longer limited by hardware, but the trade-off is that some of the partitions may experience a slightly less than optimal storage performance.

The use of Virtual SCSI will roughly double the amount of processor time to perform each I/O as compared when using directly attached storage. This processor load is split between the Virtual I/O Server and the Virtual SCSI Client. This relative figure may look poor, but what does it really translate into, in absolute figures. The extra processing time to process one 4Kbytes I/O request is less than 50k CPU cycles. On a 1.65 GHz processor, this represents only 0.03 milliseconds. This is less then 1% of the average seek time of any high performance 15k rpm SCSI disk. Section "VSCSI Performance Characteristics" on page 240 gives detailed figures.

When multiple partitions are sharing a physical disk, sizing must be performed using the sum of I/O loads of all partitions using the physical disk.

Virtual storage can still be manipulated using Logical Volume Manager (LVM) just like an ordinary physical disk. Some performance considerations from dedicated storage are still applicable when using virtual storage, such as spreading hot logical volumes across multiple volumes on multiple virtual SCSI so that parallel access is possible, the intra-disk policy (from the server's point of view, a virtual drive can be served using an entire drive, or a logical volume of a drive. If the entire drive is served to the client, then the rules and procedures apply on the client side, as if the drive were local. If the server partition provides the client with a partition of a drive and a logical volume, then the server decides the area of the drive to serve to the client when the logical volume is created and sets the inter-policy to maximum. This will spread each logical volume across as many

virtual storage as possible, allowing reads and writes to be shared among several physical volumes.

The following are general performance considerations when using Virtual SCSI:

- If not constrained by processor performance, Virtual disk I/O throughput is comparable to local attached I/O.
- Since VSCSI is a client/server model, the combined CPU cycles required on the I/O client and the I/O server will always be higher than local I/O. A reasonable expectation is a total of twice as many cycles to do VSCSI as a locally attached disk I/O (more or less evenly distributed on the client and server).
- If multiple partitions are competing for resources from a VSCSI server, care must be taken to ensure enough server resources (processor, memory, and disk) are allocated to do the job.
- There is no data caching in memory on the server partition. Thus, all I/O's which it services are essentially synchronous disk I/O's. Because there is no caching in memory on the server partition, it's memory requirements should be modest.

VSCSI virtual adapter initiator and target drivers collect and report statistics on throughput and errors to enable performance tuning and problem determination. These statistics are reported via the **iostat** command or other methods. For an example of the **iostat** command output, see XXX

7.5.9 VSCSI Performance Characteristics

In general, applications are functionally isolated from the exact nature of their storage subsystems by the operating system. An application doesn't need to be aware if its storage is contained on one type of disk or another when performing I/O. But, different I/O subsystems have subtly different performance qualities and VSCSI is no exception. What differences might an application observe using VSCSI versus directly attached storage? Broadly, we can categorize the possibilities into I/O latency and I/O bandwidth.

We define I/O latency as the time which passes between the initiation of a disk I/O and completion as observed by the thread. Latency is a very important attribute of disk I/O. Consider a program which performs 1000 random disk I/O's one at a time. If the time to complete an average I/O is 6 milliseconds, the program will run in no less than 6 seconds. However, if the average I/O response time is reduced to 3 milliseconds, the program's run time could be reduced by 3 seconds. Applications which are multi-threaded or use asynchronous I/O may be less sensitive to latency, but under almost any conceivable circumstance, less latency is better for performance.

We define I/O bandwidth as the maximum data which can be read or written to storage in a unit of time. Bandwidth can be measured from a single thread, or from a set of threads executing concurrently. Though many commercial codes are more sensitive to latency than bandwidth, bandwidth is crucial for many typical operations such as backup and restore of persistent data.

Because disks are mechanical devices, they tend to be rather slow when compared to high performance microprocessors such as POWER5. As such, we will show that VSCSI performance is comparable to directly attached storage under most workload environments.

VSCSI Latency

Because VSCSI is implemented as a client/server model, there is naturally some extra latency that does not exist with direct attached storage. We define this extra latency as the additional amount of time necessary to complete an I/O operation, when compared to the same operation on a locally attached device. Figure 7-39 on page 241 shows this additional time varies from 0.03-0.06 ms per I/O operation depending primarily on the block-size of the request using a dedicated partition I/O server. It is comparable for both the physical disk and logical volume backed virtual drives. The latency experienced when using an I/O server in a micro-partition may be higher and certainly more variable than using a dedicated partition I/O server.

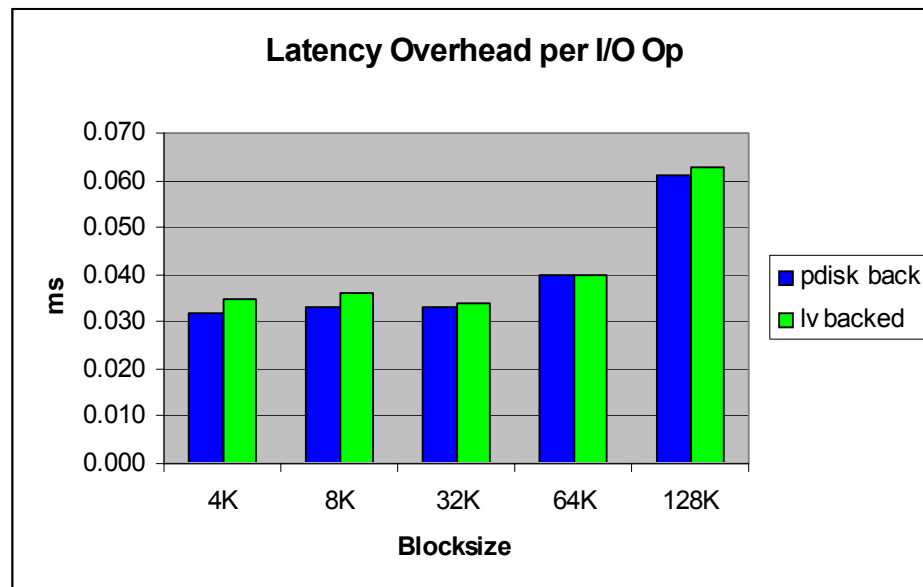


Figure 7-39 Additional Latency per I/O operation

For comparison purposes, Figure 7-40 on page 242 shows the average response times for locally attached I/O using one FASTT700 RAID0 LUN with 5 physical drives, cache enabled without write-cache mirroring. These measurements conduct sequential I/O, allowing the reads to be satisfied from the disk read cache and the writes to be cached in the FASTT700 controller. Because of caching, the physical I/O's in the test have much lower latency than in typical commercial environments, where random reads are not so often satisfied from cache. None the less, the additional VSCSI latency for these low latency caches is small compared to the actual disk latency. For I/O's with reads which are not cached by the controller, the VSCSI latency overhead is so small as to be inconsequential.

Also observed the average disk response time increases with the block size. The latency increases in performing a VSCSI operation are relatively greater on smaller block sizes because of their shorter response time.

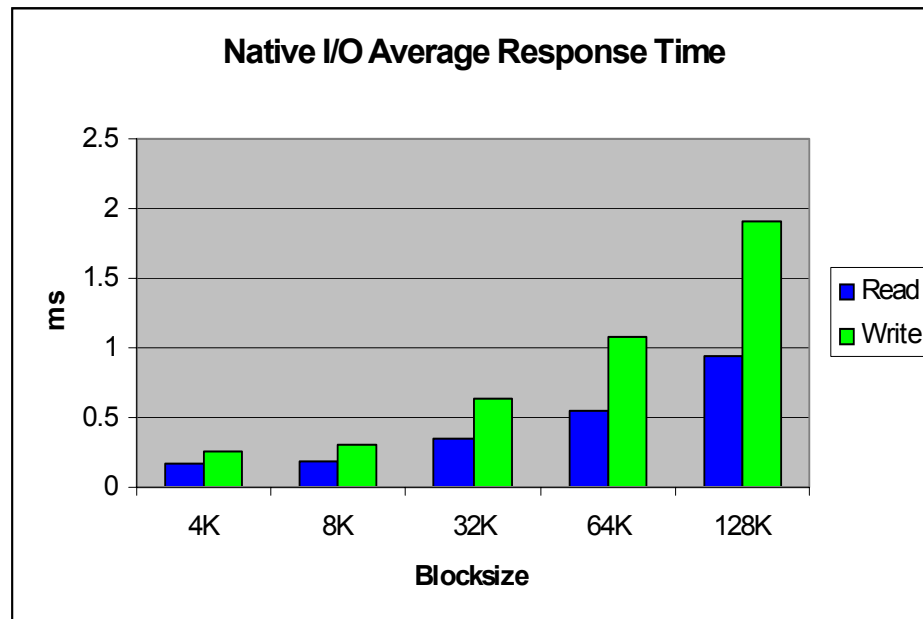


Figure 7-40 Native I/O average response time

VSCSI Bandwidth

Figure 7-41 on page 243 compares VSCSI to native I/O performance on bandwidth tests. In these tests, a single thread operates sequentially on a constant file which is 256MB in size, again with a dedicated partition I/O server. More I/O operations are issued when reading or writing to the file using a small block size versus a larger block size. This figure shows a comparison of

measured bandwidth using VSCSI and local attachment for reads with varying block sizes of operations. The difference between virtual I/O and native I/O in these tests is attributable to the increased latency using virtual I/O. Because of the larger number of operations, the bandwidth measured with small block sizes is much lower than with large block sizes.

Figure 7-42 on page 244 shows VSCSI performance using a dedicated I/O server partition scales comparably to that of a similar native I/O attached configuration to very high bandwidths. The experiment uses one FastT disk, and arrays of seven FastT disks. Each array is attached to one Fiber Channel adapter. All I/Os use a blocksize of 128 Kbytes. The difference in bandwidth between reads and writes is due to the cache in the FastT controller. The experiment shows that the difference in bandwidth using VSCSI or native disks is not significant.

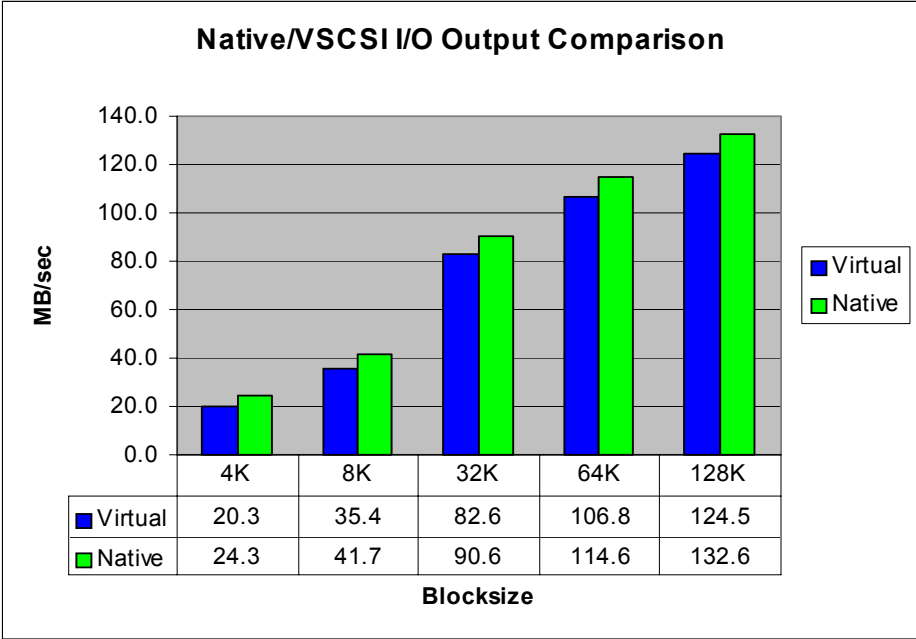


Figure 7-41 Native to VSCSI I/O comparison

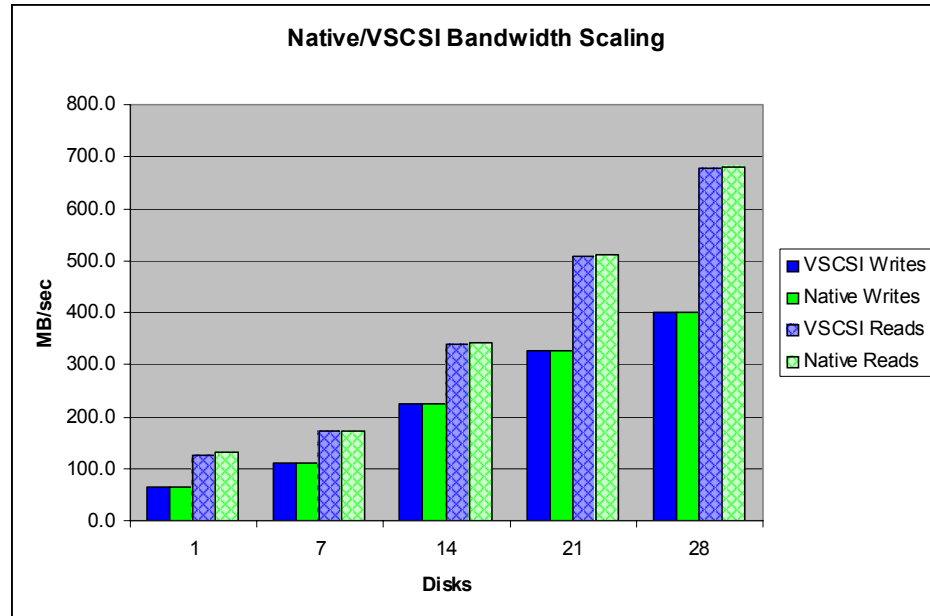


Figure 7-42 Native/VSCSI bandwidth scaling

7.5.10 VSCSI Server Sizing

There are a number of considerations to address designing and implementing a VSCSI application environment. The three primary considerations are

- ▶ the memory allocated to the I/O server,
- ▶ the processor entitlement of the I/O server,
- ▶ and whether or not the I/O server is run as a micro-partition or as a dedicated processor partition.

One thing that doesn't need to be factored into sizing is the processor impacts of using virtual I/O on the client. The processor cycles executed on the client to do a VSCSI I/O are comparable to that of a locally attached I/O. Thus, there is no increase or decrease in sizing on the client partition for a known task. These sizing techniques do not anticipate using combining the function of virtual shared Ethernet with the VSCSI server. If the two are combined, additional resources must be anticipated to support virtual shared Ethernet activity.

VSCSI Server Sizing using dedicated processor partitions

The amount of processor entitlement required for a VSCSI server is necessarily based on the maximum I/O rates required of it. Since VSCSI servers will not

normally run at maximum I/O rates all of the time, the use of surplus processor time is potentially wasted by using dedicated processor partitions. In this section, we will propose two sizing methodologies. In the first, you will need to have a fair understanding of the I/O rates and I/O sizes required of the VSCSI server. In the second, we will size the VSCSI server based more on the I/O configuration.

Sizing against expected I/O traffic

Our sizing methodology is based on the observation that the processor time required to perform an I/O on the VSCSI server is fairly constant for a given I/O size. It is a simplification to make this statement, in particular different device drivers (e.g. SCSI and FastT) have subtly varying efficiencies. But under most circumstances the I/O devices supported by the VSCSI server are sufficiently similar to allow good rules of thumb. Table 7-10 shows the rules of thumb we recommend for both physical disk and LVM operations on a 1.65Ghz POWER5 processor. These numbers are measured at the physical processor, SMT operation is assumed. For other I/O server CPU frequencies, you can adjust the cycles in Table 7-10 by multiplying the cycles per operation times the ratio of the frequencies. For example, to adjust for a 1.5 GHz CPU, $1.65\text{GHz}/1.5\text{GHz} = 1.1$, so multiply the CPU cycles in the table by 1.1 to get the required cycles per operation.

Table 7-10 I/O CPU cycles required for various block sizes

| | 4K | 8K | 32K | 64K | 128K |
|------------|-------|-------|-------|-------|--------|
| Phys. Disk | 45000 | 47000 | 58000 | 81000 | 120000 |
| LVM | 49000 | 51000 | 59000 | 74000 | 105000 |

Figure 7-43 on page 246 shows a comparison of native I/O and VSCSI Cycles per Byte (CPB) using both logical volume backed storage and physical disk backed storage. The VSCSI measures are of the I/O server only, the client overhead is not included in the comparison. The processor efficiency of I/O improves with larger I/O size. Effectively, there is a fixed overhead to start and complete an I/O, with some additional cycle time based on the size of the I/O.

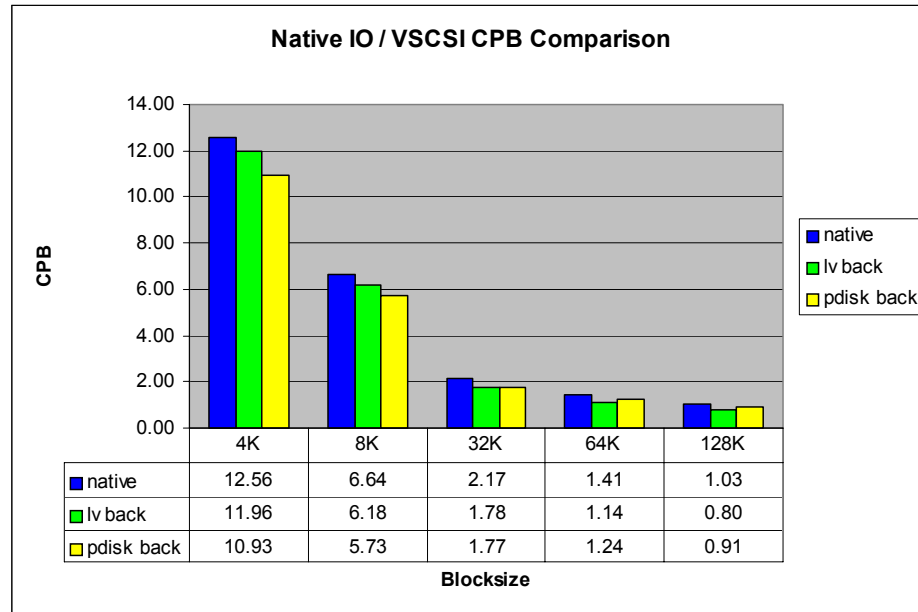


Figure 7-43 Comparison of native I/O to VSCSI

Configuration example

Consider an I/O server that supports three client partitions on physical disk backed storage. The first client partition requires a maximum of 7,000 8KB operations per second. The second client partition requires a maximum of 10,000 8KB operations per second. The third client partition requires a maximum of 5,000 128KB operations per second. The number of 1.65Ghz processors for this requirement is approximately:

$$(7,000 \times 47,000 + 10,000 \times 47,000 + 5,000 \times 120,000) / 1,650,000,000 = 0.85 \text{ processors}$$

which we need to round up to a single processor since we are not using Micro-Partitioning.

Sizing against installed storage

An alternative approach, if you don't know the I/O rates of the client partitions, is to size the VSCSI server to the maximum I/O rate of the storage subsystem attached. The sizing could be biased to small I/O's or large I/O's. Sizing to maximum capacity for large I/O's will balance the processor capacity of the VSCSI server to the potential I/O bandwidth of the attached I/O. The negative facet of this sizing methodology is that we will, in nearly every case, assign more processor entitlement to the VSCSI server than it will typically consume.

Consider a case where an I/O server manages 32 physical SCSI disks. We can arrive at an upper bound of processors required based on assumptions about the I/O rates that the disks can achieve. If it is known that the workload is dominated by 8KB operations which are random, we could assume that each disk is capable of approximately 200 disk I/O's per second (15Krpm drives). At peak, the I/O server would need to service approximately 32 disks * 200 I/O's per second times 120,000 cycles per operation, resulting in a requirement for approximately 19% of one processor performance. Viewed another way, an I/O server running on a single processor should be capable of supporting more than 150 disks serving 8KB random I/O's for other partition's CPUs.

Alternatively, if the server is sized for maximum bandwidth, the calculation will result in a much higher processor requirement. The difference is that maximum bandwidth assumes sequential I/O. Since disks are much more efficient doing large sequential I/O's than small random I/O's, we can drive a much higher number of I/O's per second. Assume that the disks are capable of 50MB/sec. when doing 128KB I/O's. That implies each disk could average 390 disk I/O's per second. Thus, the entitlement necessary to support 32 disks, each doing 390 I/O's per second with an operation cost of 120,000 cycles $(32 * 390 * 120,000 / 1,650,000,000)$ approximately 0.91 processors. More simply, an I/O server running on a single processor should be capable of driving approximately 32 fast disks to maximum throughput.

This sizing method can be very wasteful of processor entitlement when using dedicated processor partitions but will guarantee peak performance. It is most effective if the average I/O size can be estimated, so that peak bandwidth does not need to be assumed.

VSCSI Server sizing using Micro-Partitioning

Defining VSCSI servers in Micro-partitions allows much better granularity of processor resource sizing and potential recovery of unused processor time by uncapped partitions. Tempering those benefits, use of Micro-partitions for VSCSI servers will slightly increase I/O response time and make for somewhat more complex processor entitlement sizings.

The sizing methodology should be based on the same operation costs for dedicated partition I/O servers. However, addition entitlement should be added for running in Micro-partitions (See the Section "Micro-Partitioning considerations" on page 145 for comparison of dedicated partitions versus micro-partitions). We would recommend that the I/O server be configured as uncapped, that way if the I/O server is under sized, there is opportunity to get more processor time to service I/O.

Since I/O latency with virtual SCSI will vary with the machine utilization and I/O server topology, consider the following:

1. For the most demanding I/O traffic(high bandwidth or very low latenc), try to use native I/O.
2. If native I/O is not an option and the system contains enough processors, consider putting the I/O server in a dedicated processor partition.
3. If using a Micro-Partitioning I/O server, use as few virtual processors as possible.

VSCSI Server Memory Sizing

The architecture of VSCSI simplifies memory sizing in that there is no caching of file data in the memory of the VSCSI server. Since there is no caching of data, the memory requirements for the VSCSI server are fairly modest. With large I/O configurations and very high data rates, a 1GB memory allocation for the VSCSI server is more than sufficient. For low I/O rate cases with a small number of attached disks, 512MB is a sufficient memory allocation.

7.6 Virtual SCSI

<Replace diagrams with server and client instead of hosted and hosting.>

Virtual I/O allows the POWER5 to support more partitions than it has slots for I/O devices by enabling the sharing of I/O adapters amongst partitions. Virtual SCSI (VSCSI) will enable a partition to access block-level storage that is not a physical resource of that partition. The VSCSI design is that the virtual storage be backed by a logical volume on a portion of a disk rather than an entire physical disk. These logical volumes appear to be the SCSI disks on the client partition, which gives the system administrator maximum flexibility in configuring partitions. VSCSI support is provided by a service running in an I/O server that uses two primitive functions, Reliable Command / Response Transport and Logical Remote DMA to service I/O requests for an I/O client, such that, the I/O client appears to enjoy the services of its own SCSI adapter. The terms *I/O server* and *I/O client* refer to platform partitions that are respectively servers and clients of requests, usually I/O operations, using the I/O server's I/O adapters. This allows a platform to have more I/O clients than it may have I/O adapters because the I/O clients share I/O adapters via the I/O server.

Virtual I/O will provide a high performance I/O mechanism by minimizing the number of times data is copied within the memory of the physical system. The virtual I/O model described herein allows for either zero copy, if data is being retrieved from a physical device and DMAed directly to the memory of the partition using virtual I/O using the redirected DMA described in "Logical Remote Direct Memory Access (LRDMA)" on page 252, or single copy of the data is first moved to the memory space of the I/O server before being DMAed to the I/O client.

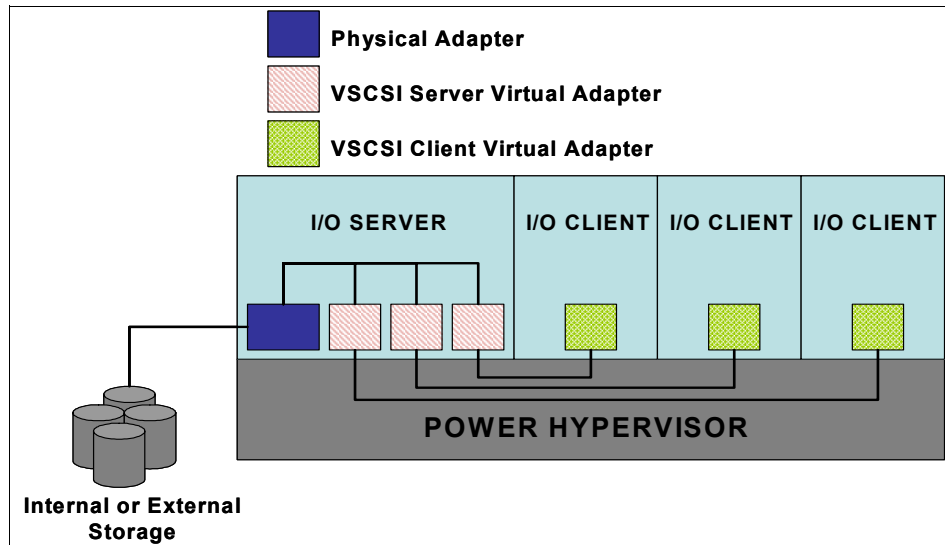


Figure 7-44 AIX 5L Server and Client Partitions

7.6.1 Virtual SCSI Structure and Concepts

The effort to implement virtual SCSI on AIX 5L can be organized into three primary components: client driver, server driver, and interpartition communication.

The client and server drivers operate as a pair, in a point-to-point configuration, with the Hypervisor providing the means of communication between the two. The client emulates a physical SCSI adapter to the disk, tape, and CD-ROM peripheral (or “head”) drivers. The client driver accepts requests for storage services from these peripheral drivers, converts those requests into SRP Information Units, then uses interpartition communication facilities to transmit those requests to the server driver. The server driver completes the requests using some combination of software emulation and physical devices, then converts the results into SRP Information Units and returns those results back to the client driver.

The target and initiator drivers are always connected in a point-to-point configuration, with one initiator driver communicating with at most one target driver. A target driver can provide storage services to multiple initiator drivers serially. That is, when a target driver disconnects from an initiator driver, that target driver is then available for use by another initiator driver on another partition.

A partition may have instances of both client and server drivers defined in it. However, “cascaded” devices (virtual devices that are backed by virtual devices) are not supported.

Virtual SCSI does not support “target mode”. SRP does not define a method for a target to send a request for service to the initiator.

The relationships between the client driver, server driver, peripheral drivers, and the Hypervisor are shown in Figure 7-45 on page 251.

Interpartition Communication Overview

Interpartition communication involves a client device node in the Open Firmware device tree of one partition, a server device node in the Open Firmware device tree of another partition, an interpartition channel, and a protocol definition. Interpartition communication requires the use of two primitive functions:

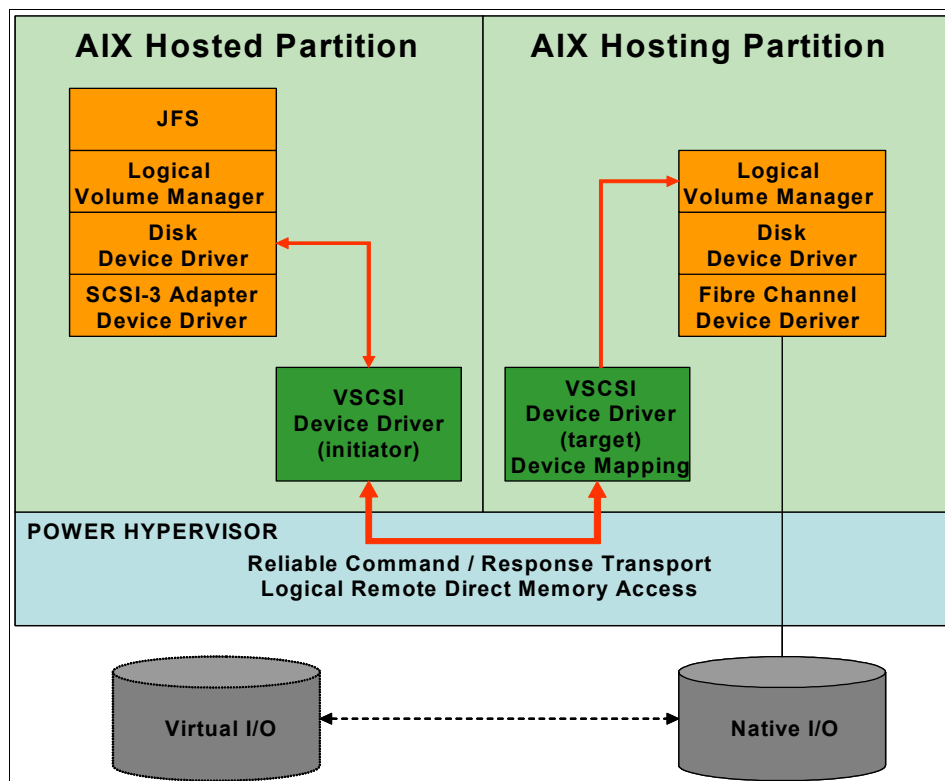


Figure 7-45 Reliable Command / Response Transport and LRDMA

Reliable Command/Response Transport

The transport over which SRP runs is the Reliable Command/Response Transport facility provided by the Hypervisor. The Reliable Command / Response Transport facility provides ordered delivery of messages between authorized partitions. In order to communicate, a client/server partition pair must establish a Command/Response Queue (CRQ). For a detailed description of the CRQ, see “The Command/Response Queue (CRQ)” on page 178

A CRQ is established during configuration by a virtual SCSI driver, given the presence in the Open Firmware device tree of a virtual SCSI device. The initiator driver registers a response queue and the target driver registers a command queue. Both use the `h_reg_crq` kernel service to call the Hypervisor. The Hypervisor creates a connection between the two partitions through the queues.

Once the queues are established, the virtual SCSI drivers can use the `h_send_crq` kernel service to put queue elements on each other's queues. The initiator driver attempts to queue an element to the target driver's command queue to initiate a transaction. If it is successful, the initiator driver returns, waiting for the interrupt indicating that a response has been posted by the target driver to the initiator driver's response queue.

The client partition only uses the Reliable Command / Response Transport. It does not use the Logical Remote DMA. Since the server partition's RTCE tables are not authorized for access by the client partition, any attempt by the client partition to modify server partition memory would be prevented by the Hypervisor. RTCE table access is granted on a connection by connection basis (client/server virtual device pair); if a client partition happens to be serving some other logical device then the partition is entitled to use Logical Remote DMA for the virtual devices that are serving.

The target driver is notified via an interrupt that it has received a message on its command queue. The target driver decodes the I/O request and routes it through the server partition's file sub-system for processing. When the request completes, the file sub-system calls the target driver and it packages a response into a queue element that is then queued to the initiator driver's response queue.

Logical Remote Direct Memory Access (LRDMA)

Logical Remote Direct Memory Access (LRDMA) allows for an I/O server to securely target memory pages within an I/O client for virtual I/O operations. The I/O server uses the `hcall()` of the Logical Remote DMA facility to manage the movement of commands and data associated with the client requests. The server driver may use this service if it has a connection established via a Command/Response Queue pair. Virtual SCSI defines two modes of LRDMA:

- Traditional Copy RDMA that involves the I/O server's I/O adapters targeting DMA buffers in the I/O server's memory and having the Hypervisor copy data between that DMA buffer and the I/O client's memory.
- Redirected RDMA allows for an I/O server to securely target its physical I/O adapter's DMA operations directly at the memory pages of the I/O client.

The platform overhead of Copy RDMA is generally greater than Redirected RDMA, but this overhead may be offset if the I/O server's DMA buffer is being used as a data cache for multiple virtual I/O operations.

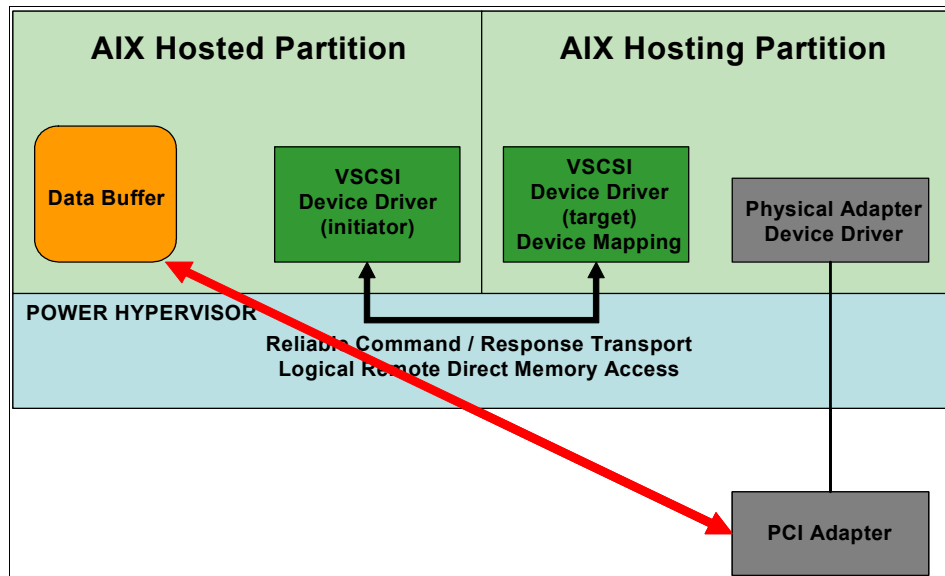


Figure 7-46 Logical Remote Direct Memory Access

LRDMA defines an extended type of TCE table called a Remote DMA TCE table (RTCE). An RTCE is used by the Hypervisor to translate a server partition's Logical Remote DMA hcalls()'s DMA addresses. RTCE tables have extra data to help manage the use of its mappings by server partitions. Note that only the target driver uses the Logical Remote DMA primitives, not the initiator driver. The server partition's RTCE tables are not authorized for access by the client driver.

The use of Redirected RDMA is completely invisible to the I/O client, and has no impact on the VSCSI architecture defined in this document. It is left entirely to the discretion of the I/O server whether it first moves data from a physical device into its own memory before moving (DMAing) the data to the I/O client, or whether the I/O server sets up the I/O request to the physical device in such a way that

the physical device DMAs directly to the memory of the I/O client. The I/O server uses the RDMA mode that best suits its needs for a given virtual I/O operation.

The logical remote direct memory service allows the hosting driver to read and write to a well defined part of the I/O client's memory. This service is unidirectional, i.e. the hosted driver cannot use the service to write to, or read from, the I/O server's memory.

SCSI Remote DMA Protocol (SRP)

SCSI Remote DMA Protocol (SRP) defines a method of encapsulating SCSI Command Data Blocks (CDBs) and is the protocol used for interpartition communication for Virtual SCSI on IBM @server p5 logical partition. Because virtual SCSI involves heterogeneous operating systems (AIX 5L and Linux) it is important to implement a common industry standard protocol for communicating I/O operations between partitions. SRP has defined the message format and protocol using an RDMA communication service. The SCSI RDMA Protocol defines the rules for exchanging SCSI information in an environment where SCSI initiators and targets have the ability to directly transfer information between their respective address spaces.

All SRP communication is accomplished via SRP Informational Units (IUs). An IU is an organized collection of data specified by the SRP to be transferred as login data, reject data or a message on an RDMA channel. Thus all SCSI commands, and their associated data and status, are encapsulated in an SRP IU. Note that the protocol used for interpartition communication has no bearing on the makeup of the destination device. The SRP protocol works just as well if the target device is a physical device or a logical device (logical volume).

SRP Memory Descriptor Mapping

The SRP architecture defines a “memory descriptor”, which is a 16-byte structure that identifies a memory segment upon which DMA operations can be performed.

The VSCSI architecture is defined such that DMA operations need never be initiated from the I/O client (from the initiator port.) Since the I/O server's RTCE tables are not authorized for access by the I/O client, any attempt by the I/O client to modify I/O server's memory would be prevented by the Hypervisor. RTCE table access is granted on a connection by connection basis (hosted/hosting virtual device pair), if a I/O client happens to be hosting some other logical device then the partition is entitled to use Logical Remote DMA for the virtual devices that it is hosting.

Memory descriptors sent in IUs defined in this architecture always reference memory in the initiator, and are always used in DMA operations initiated by the target.

SRP initiator ports and SRP target ports shall be determined by both their role during RDMA channel establishment and by the adapter types on which the messages are sent and received. VSCSI Message Formats

7.6.2 Virtual SCSI Model Overview

Server Partition

A server partition is a partition that has physically attached I/O devices and exports one or more of these devices to other partitions. The virtual SCSI adapter driver on the server partition (`vscsi_targetdd`) is a dynamically loadable kernel extension and its entry points are contained in the `devswitch` table. It is the SRP target. The primary function of the target driver is to convert SRP requests from the initiator driver into I/O requests that are forwarded to the device via the native stack, and then use LRDMA services to copy a response to the initiator's memory.

The `vscsi_targetdd` driver receives command queue elements from the client partition delivered by the Hypervisor. These elements contain DMA handles that are used to read the SRP Information Units built by the client partition, and extract the SCSI Command Descriptor Block (CDB). The SCSI CDB within the SRP IU, and the nature of the target, determine whether a command is emulated, passed to the native stack, or both. Once a command is completed, `vscsi_targetdd` builds an SRP Response with the returned status and uses LRDMA services to copy the response to the client's memory.

The `vscsi_targetdd` driver provides a `struct buf` interface to the LVM. And it provides an SRP target interface to its partner initiator driver across the Hypervisor command/response queue (CRQ) connection.

Client Partition

A client partition is a partition that has a virtual client adapter node defined in its Open Firmware device tree. The client partition relies on another partition (the server partition) to provide access to one or more block interface devices. The virtual client adapter device driver (`vscsi_initdd`) is a dynamically loadable kernel extension and its entry points are contained in the `devswitch` table. It is the SRP initiator. The primary function of the initiator driver is to convert I/O requests from the head or media device drivers to SRP IUs, then make the SRP IU available to the SRP target for LRDMA.

The virtual adapter on the client partition is in many ways similar to a physical SCSI adapter. While a typical SCSI adapter has a parallel bus or optical link attached to it, the virtual adapter's link is the Hypervisor's Reliable Command/Response Transport.

The `vscsi_initdd` driver provides a `scsi_buf` interface and SRP initiator interface to its partner target driver across the Hypervisor Command / Response Queue (CRQ) connection.

Virtual SCSI Flow

An example of a typical interaction between the target and initiator device drivers is a file read from a virtual DASD device. A virtual DASD is a virtual device on the client partition, which is backed by a logical volume exported from a DASD device that is physically attached to the server partition. The client stack considers the initiator driver a SCSI-3 device with access to the virtual DASD.

A typical I/O request involves the following steps:

- (Client) The application program initiates a `read()` system call to the filesystem (JFS).
- (Client) The filesystem requests a read from the LVM. LVM forms a `buf` struct with DMA buffer addressing information, as well as DASD block information.
- (Client) The `buf` struct is passed to the disk device driver which creates a `scsi_buf` and sends it to the `vscsi_initdd` driver.
- (Client) The initiator driver takes the information in the `scsi_buf` and creates an SRP IU. If the I/O request includes data to be transferred, the initiator driver maps the data buffers for DMA.
- (Client) The client builds a CRQ command element containing a pointer to the SRP IU and sends the CRQ command element through the Hypervisor to `vscsi_targetdd`.
- (Server) The target driver receives an interrupt indicating that an element has been queued to its command queue.
- (Server) The target driver uses the pointer to the SRP IU in the CRQ command element and LRDMA services to copy the SRP IU from the client partition to the server partition's memory.
- (Server) The target driver uses the information in the SRP IU to create a `buf` struct.
- (Server) The target driver passes the `buf` struct to the LVM running in the server partition. The request ultimately makes its way to the adapter device driver. This driver calls the usual kernel DMA services, which have been extended to map the buffers for DMA using LRDMA services.
- (Server) When the transaction is complete, the target driver constructs an appropriate SRP response and uses LRDMA services to copy the response to the client's memory. It then builds a CRQ command element

containing the TAG or “correlator field” from the original SRP IU and sends the CRQ element through the Hypervisor to the initiator.

- (Client) The initiator driver receives an interrupt indicating that a CRQ element has been queued to its response queue.
- (Client) The initiator driver uses the information in the SRP response to give status back to its child head driver. The head driver passes the results back up to LVM.

Virtual SCSI adapters

The Hypervisor architecture defines two distinct virtual adapters, one being a Virtual SCSI initiator, and the other being a Virtual SCSI target which implement the SCSI Initiator Port and SCSI Target Port in the SCSI Architecture Model. Both the CRQ and SRP architectures are asymmetrical. This architecture requires that protocol messages defined as being sent from an initiator to a target only be sent from the initiator adapter to the target adapter.

Emulated DASD

Emulated Direct Access Storage Device (DASD) is a virtual disk device that is mapped by the server device driver to a logical volume and presented to the I/O client as a physical direct access device. There can be many emulated DASD devices mapped onto a single physical DASD. The system administrator will create an emulated DASD device by choosing a logical volume and binding it to a VSCSI hosting adapter. The command to add virtual devices will create an ODM entry for the emulated DASD device.

It is expected that most of the SCSI commands targeting an emulated DASD device will be either reads or writes. Reads and writes are serviced by the LVM. The routine `target_interrupt` calls the `edasd_scheduler` function, using the scheduler function pointer, and passes a `vadapter_lun` pointer and a command element. The `buf` structure fields relating to DMA addressing, including the cross memory descriptor and DMA buffer addresses, are set by `target_interrupt`. The `type` field of the command element determines whether the `scsi_req` buffer describes a SCSI CDB or a SCSI task. SCSI commands and SCSI task management are discussed in separate subsections.

Figure 7-47 on page 258 shows the possible partitioning of a physical disk on the I/O server where there are two logical volumes that support two emulated DASD devices, `hdx` and `hdy`. The DASD emulation code does addresses the logical volume from block zero which overwrites the LVCB.

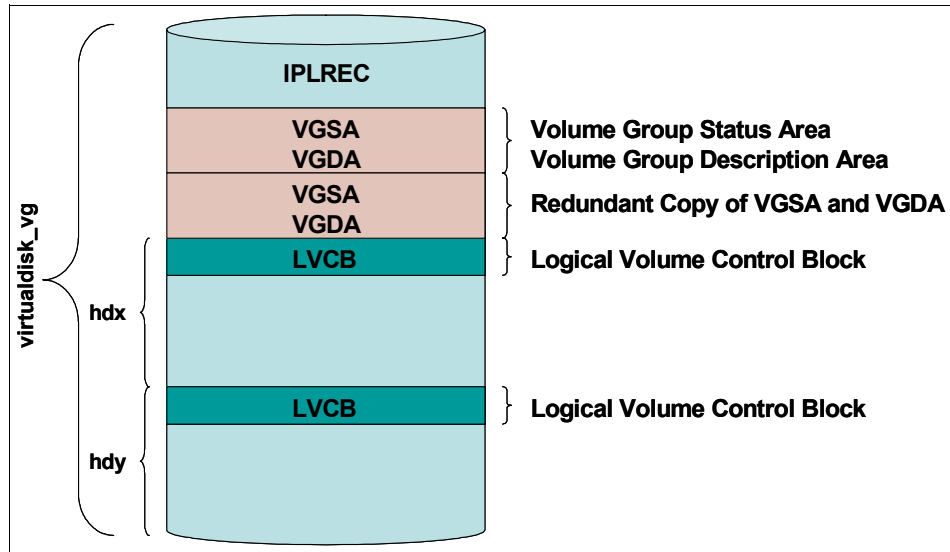


Figure 7-47 Volume group on Virtual I/O Server

SCSI RESERVE and RELEASE

Since the physical storage for VSCSI is provided by a logical volume instead of a physical device, the VSCSI virtual adapter driver will have to emulate the SCSI RESERVE and RELEASE commands instead of passing them on to the device. That emulation will be limited in scope to a single I/O server. When one I/O client wins a reservation on a logical volume, the VSCSI virtual adapter target driver will have to refuse access by other I/O clients to the logical volume. And when the I/O client holding a reservation fails, the VSCSI virtual adapter target driver will have to break the reservation on that logical volume. This will enable configurations where one I/O server provides storage services for multiple I/O clients. However, this will not provide an adequate emulation of RESERVE and RELEASE for multi-path configurations, where the same physical storage can be accessed by multiple adapters from multiple partitions. This emulation will not prevent access by the native stack on that I/O server.

Command Tag Queueing

SCSI Command tag queueing refers to queuing commands to a SCSI device. Command tag queueing requires the SCSI adapter, the SCSI device, the SCSI device driver, and the SCSI adapter driver to support this capability. The VSCSI architecture supports command tag queueing.

Redundant Configurations

In order to minimize the adverse impacts that would result from the loss of server partition or physical adapter, a system administrator can use two different ways to create redundant configurations. Each of these techniques will allow a client partition to continue to function while maintenance is being done on the server partition.

LVM Mirroring

The recommended solution for VSCSI I/O redundancy is using LVM mirroring. The Logical Volume Manager supports mirroring, for every write to a logical volume, the LVM generates a write request for every mirror copy. The system administrator can define two virtual DASD devices, either hosted by two distinct I/O servers or two devices on the same I/O server, and mirror the client partition's data on the two devices. Mirroring makes no requirements on either the client, or hosting drivers. It is cost effective and the system configuration is readily understood.

From a performance point of view, LVM mirroring does not give you the best performance. With the overhead on the Hypervisor calls to perform I/O requests plus the extra copies that you need to maintain (writing to two or three copies takes longer than writing to one), these will have an effect on disk performance.

If mirroring is needed, set the scheduling policy to parallel and allocation policy to strict. The parallel scheduling policy will enable reading from the disk that has the least outstanding requests, and strict allocation policy allocates each copy on separate physical volume(s). And locate intensive mirrored logical volumes on the outer edge because, in that situation, the mirror write consistency cache must be updated on a regular basis.

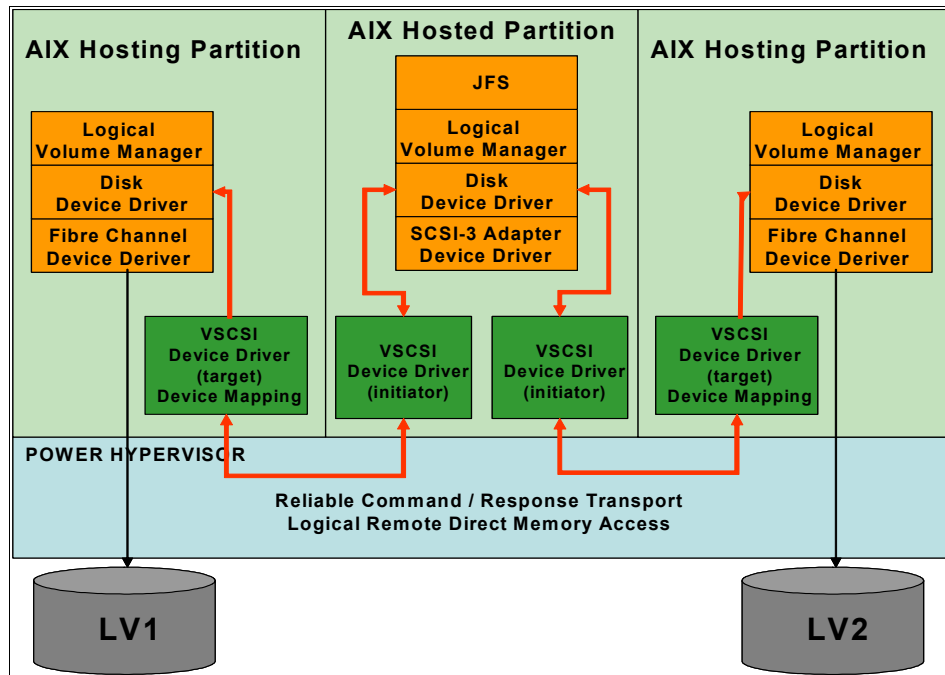


Figure 7-48 Using LVM mirroring for virtual SCSI

Multi-path I/O

Multi-path I/O (MPIO) offers another possible solution to the redundancy requirement. MPIO is a feature of AIX 5L that permits a volume accessed by multiple physical paths to be seen as a single hdisk. It is therefore logically similar to IBM Subsystem Device Driver (SDD), which allows a volume on the TotalStorage® Enterprise Storage Subsystem™ (ESS) that is accessed through multiple paths to be seen as a single vpath disk. However, the SDD logical construct of a vpath disk is above the level of the hdisk, whereas MPIO combines the paths underneath the level of the hdisk. MPIO is intended to support additional disk subsystems besides ESS. These disk subsystems are themselves capable of supporting multiple physical (parallel or fibre SCSI) attachments.

MPIO has numerous possible configuration parameters. A detailed discussion of them is beyond the scope of this redbook. However, to gain the benefits of high availability and throughput that MPIO offers, it is recommended that it be configured with a “round robin” algorithm, “health check” enabled, and a reserve policy of “no reserve”. This allows the best combination of throughput and reliability, since all paths are used for data transfer, and failed paths are detected and reacted to in a timely fashion.

7.6.3 Performance Considerations

Enabling VSCSI may not result in a performance benefit. This is because there is an overhead associated with Hypervisor calls and the several path involves for the I/O requests from the initiator to target partition, VSCSI will use additional processor cycles when processing I/O requests. This will not give the same performance from VSCSI devices as from the dedicated devices. Partition with high performance and disk I/O requirements is not recommended to implement VSCSI. Partitions with very low performance and disk I/O requirements can be configured at minimum expense to use only a portion of a logical volume. Using a logical volume for virtual storage means that the number of partitions is no longer limited by hardware, but the trade-off is that some of the partitions will have less than optimal storage performance. The suitable applications for VSCSI might be the boot disks for the operating system or web servers which will typically cache a lot of data.

The use of Virtual SCSI will *roughly double* the amount of processor time to perform I/O as compared when using directly attached storage. This processor load is split between the Virtual I/O Server and the Virtual SCSI Client. Performance is expected to degrade when multiple partitions are sharing a physical disk, and actual impact on overall system performance will vary by environment. The best-case configuration is when one physical disk is dedicated to a partition.

Virtual storage can still be manipulated using Logical Volume Manager (LVM) just like an ordinary physical disk. Some performance considerations from dedicated storage are still application specific when using virtual storage, such as spreading hot logical volumes across multiple volumes on multiple virtual SCSI so that parallel access is possible, the intra-disk policy (from the server's point of view, a virtual drive can be served using an entire drive, or a logical volume of a drive. If the entire drive is served to the client, then the rules and procedures apply on the client side, as if the drive were local. If the server partition provides the client with a partition of a drive and a logical volume, then the server decides the area of the drive to serve to the client when the logical volume is created and sets the inter-policy to maximum. This will spread each logical volume across as many virtual storage as possible, allowing reads and writes to be shared among several physical volumes.

The following are general performance considerations when using Virtual SCSI:

- Since VSCSI is a client/server model, the combined CPU cycles required on the I/O client and the I/O server will always be higher than local I/O. A reasonable expectation is a total of twice as many cycles to do VSCSI as a locally attached disk I/O (more or less evenly distributed on the client and server).

- If multiple partitions are competing for resources from a VSCSI server, care must be taken to ensure enough server resources (processor, memory, and disk) are allocated to do the job.
- If not constrained by processor performance, dedicated partition throughput is comparable to doing local I/O.
- There is no data caching in memory on the server partition. Thus, all I/O's which it services are essentially synchronous disk I/O's. Because there is no caching in memory on the server partition, it's memory requirements should be modest.

The path of each virtual I/O request involves several sources of overhead that are not present in a non-virtual I/O request. For a virtual disk backed by the LVM, there is also the performance impact of going through the LVM and disk device drivers twice. From a performance point of view, each I/O request to a virtual device backed by the LVM involves the following:

- (Initiator) Conversion of `scsi_buf` to SRP IU and queue element
- (Initiator) `H_SEND_CRQ` hcall to put queue element on target driver's command queue
- (Target) Interrupt when an I/O request is put on the command queue
- (Target) `H_COPY_RDMA` hcall to get the SRP IU from the client partition
- (Target) Conversion of queue element and SRP IU to `buf` struct
- (Target) Overhead from sending request to the LVM rather than a physical disk
- (Target) Overhead from `D_MAP_LIST` and `D_MAP_PAGE` mapping remote bus memory (kernel services using RTCEs rather than regular TCEs)
- (Target) Conversion of `buf` struct response to SRP IU and queue element
- (Target) `H_SEND_CRQ` hcall to put queue element on initiator driver's response queue
- (Target) `H_COPY_RDMA` hcall to copy the SRP response to the initiator driver
- (Initiator) Interrupt when a response is received on the response queue
- (Initiator) Conversion of queue element and SRP IU to `scsi_buf`

VSCSI virtual adapter initiator and target drivers will collect and report statistics on throughput and errors to enable performance tuning and problem determination. These statistics will be reported via the **`iostat`** command or other methods. For an example of the **`iostat`** command output, see “Existing performance commands enhancement” on page 67

7.6.4 VSCSI Performance Characteristics

In general, applications are functionally isolated from the exact nature of their storage subsystems by the operating system. An application doesn't need to be aware if its storage is contained on one type of disk or another when performing I/O. But, different I/O subsystems have subtly different performance qualities and VSCSI no exception. What differences might an application observe using VSCSI versus directly attached storage? Broadly, we can categorize the possibilities into I/O latency and I/O bandwidth.

We define I/O latency as the time which passes between the initiation of a disk I/O and completion as observed by the thread. Latency is a very important attribute of disk I/O. Consider a program which performs 1000 random disk I/O's one at a time. If the time to complete an average I/O is 6 milliseconds, the program will run in no less than 6 seconds. However, if the average I/O response time is reduced to 3 milliseconds, the program's run time could be reduced by 3 seconds. Applications which are multi-threaded or use asynchronous I/O may be less sensitive to latency, but under almost any conceivable circumstance, less latency is better for performance.

We define I/O bandwidth as the maximum data which can be read or written to storage in a unit of time. Bandwidth can be measured from a single thread, or from a set of threads executing concurrently. Though many commercial codes are more sensitive to latency than bandwidth, bandwidth is crucial for many typical operations such as backup and restore of persistent data.

Because disks are mechanical devices, they tend to be rather slow when compared to high performance microprocessors such as POWER5. As such, we will show that VSCSI performance is comparable to directly attached storage under most workload environments.

VSCSI Latency Overhead

Because VSCSI is implemented as a client/server model, there is naturally some latency overhead that does not exist with direct attached storage. We define the latency overhead as the additional amount of time necessary to complete an I/O operation, when compared to the same operation on a locally attached device. Figure 7-49 on page 264 shows the overhead ranges from 0.03-0.06 ms per I/O operation depending primarily on the block-size of the request using a dedicated partition I/O server. The average latency overhead is comparable for both the physical disk and logical volume backed virtual drives. The latency experienced when using an I/O server in a micro-partition may be higher and certainly more variable than using a dedicated partition I/O server.

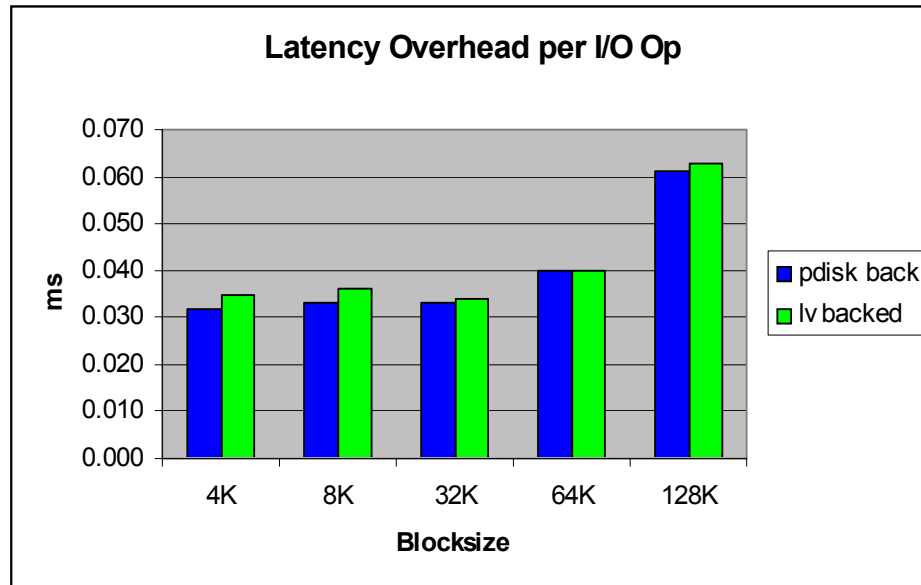


Figure 7-49 Latency overhead per I/O operation

For reference purposes, Figure Figure 7-50 on page 265 shows the average response times for locally attached I/O using one FASTt700 RAID0 LUN with 5 physical drives, cache enabled without write-cache mirroring. These measurements conduct sequential I/O, allowing the reads to be satisfied from the disk read cache and the writes to be cached in the FASTt700 controller. Because of caching, the physical I/O's in the test have much lower latency than in typical commercial environments, where random reads are not so often satisfied from cache. None the less, the additional VSCSI latency for these low latency caches is small compared to the actual disk latency. For I/O's with reads which are not cached by the controller, the VSCSI latency overhead is so small as to be inconsequential.

Also observed the average disk response time increases with the block size. The latency increases in performing a VSCSI operation are relatively greater on smaller block sizes because of their shorter response time.

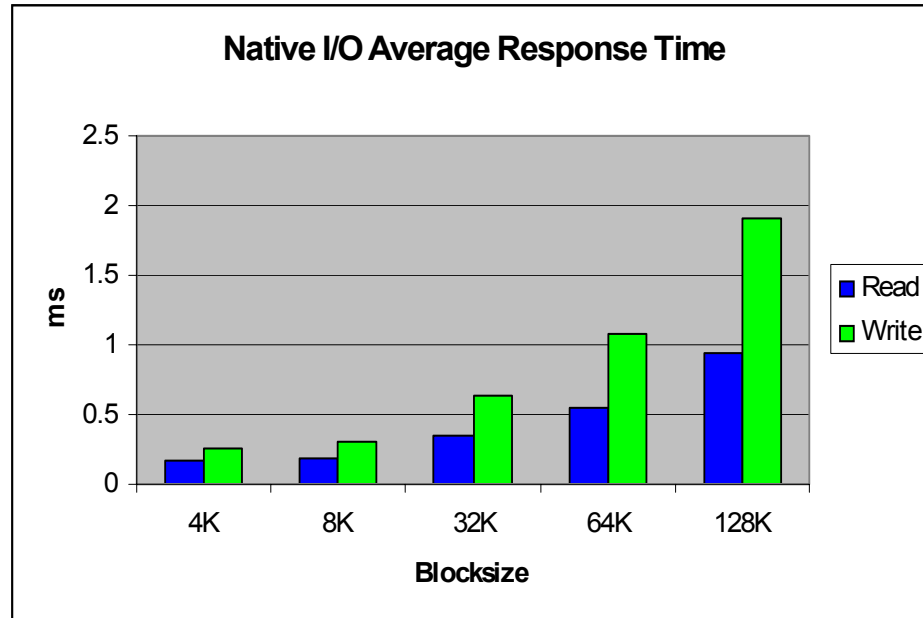


Figure 7-50 Native I/O average response time

VSCSI Bandwidth

Figure 7-51 on page 266 compares VSCSI to native I/O performance on bandwidth tests. In these tests, a single thread operates sequentially on a constant file which is 256MB in size, again with a dedicated partition I/O server. More I/O operations are issued when reading or writing to the file using a small block size versus a larger block size. This figure shows a comparison of measured bandwidth using VSCSI and local attachment for reads with varying block sizes of operations. The difference between virtual I/O and native I/O in these tests is attributable to the increased latency using virtual I/O. Because of the larger number of operations, the bandwidth measured with small block sizes is much lower than with large block sizes.

Figure 7-52 on page 266 shows VSCSI performance using a dedicated I/O server partition scales comparably to that of a similar native I/O attached configuration to very high bandwidths.

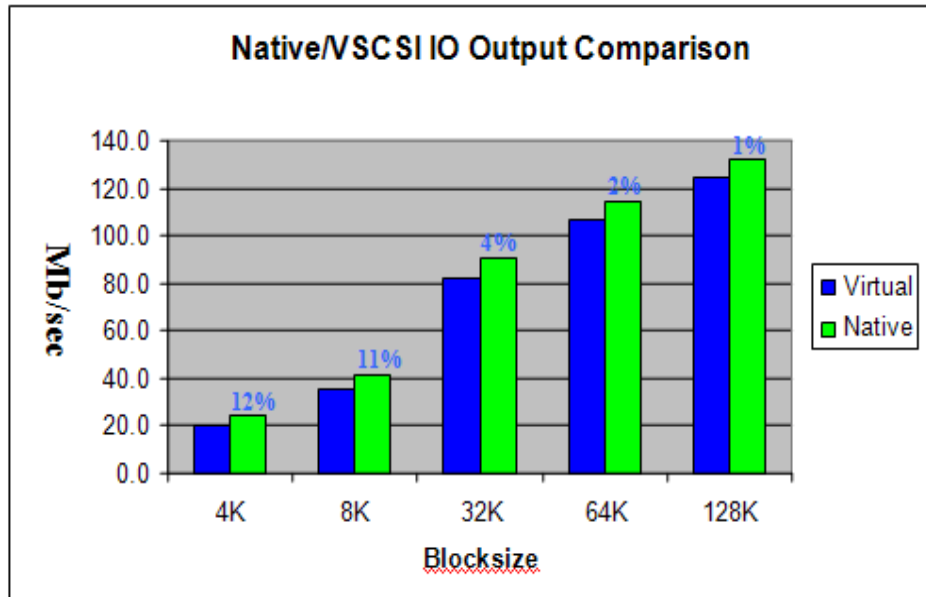


Figure 7-51 Native to VSCSI I/O comparison

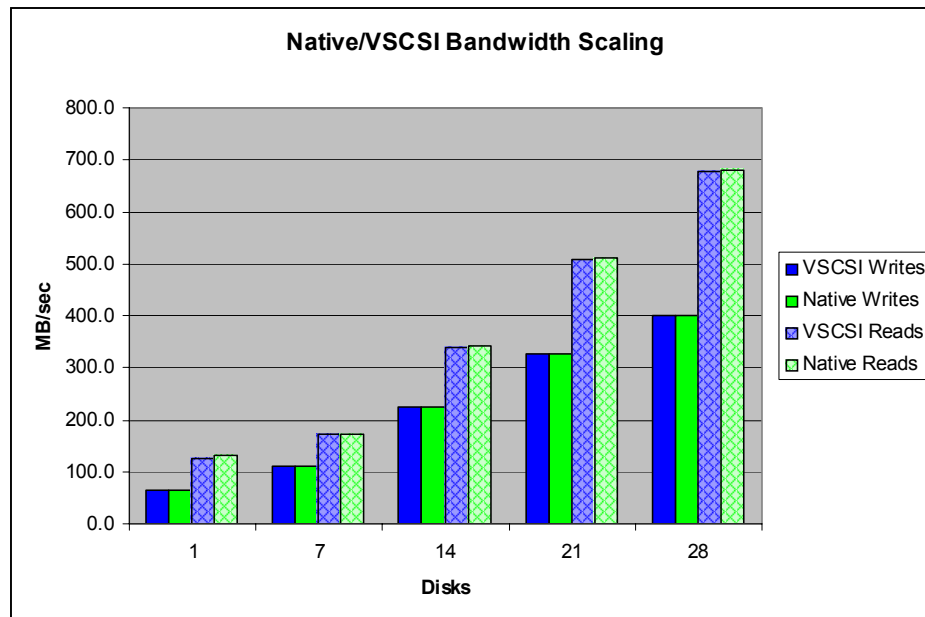


Figure 7-52 Native/VSCSI bandwidth scaling

VSCSI Server Sizing

There are a number of considerations required when designing and implementing a VSCSI application environment. The three primary considerations are based around the sizing of the I/O server. They include the memory allocated to the I/O server, the processor entitlement of the I/O server, and whether or not the I/O server is run as a micro-partition or as a dedicated processor partition. One thing that doesn't need to be factored into sizing is the processor impacts of using virtual I/O on the client. The processor cycles executed on the client to do a VSCSI I/O are comparable to that of a locally attached I/O. Thus, there is no increase or decrease in sizing on the client partition for a known task. These sizing techniques do not anticipate using combining the function of virtual shared Ethernet with the VSCSI server. If the two are combined, additional resources must be anticipated to support virtual shared Ethernet activity.

VSCSI Server Sizing using dedicated processor partitions

The amount of processor entitlement required for a VSCSI server is necessarily based on the maximum I/O rates required of it. Since VSCSI servers will not normally run at maximum I/O rates all of the time, the use of surplus processor time is potentially wasted by using dedicated processor partitions. In this section, we will propose two sizing methodologies. In the first, you will need to have a fair understanding of the I/O rates and I/O sizes required of the VSCSI server. In the second, we will size the VSCSI server based more on the I/O configuration.

Our sizing methodology is based on the observation that the processor time required to perform an I/O on the VSCSI server is fairly constant for a given I/O size. It is a simplification to make this statement, in particular different device drivers (e.g. SCSI and FastT) have subtly varying efficiencies. But under most circumstances the I/O devices supported by the VSCSI server are sufficiently similar to allow good rules of thumb. Table 7-11 shows the rules of thumb we recommend for both physical disk and LVM operations on a 1.65Ghz POWER5 processor. These numbers are measured at the physical processor, SMT operation is assumed. For other I/O server CPU frequencies, you can adjust the cycles in Table 7-11 by multiplying the cycles per operation times the ratio of the frequencies. For example, to adjust for a 1.5 GHz CPU, $1.65\text{GHz}/1.5\text{GHz} = 1.1$, so multiply the CPU cycles in the table by 1.1 to get the required cycles per operation.

Table 7-11 I/O CPU cycles required for various block sizes

| | 4K | 8K | 32K | 64K | 128K |
|------------|-------|-------|-------|-------|--------|
| Phys. Disk | 45000 | 47000 | 58000 | 81000 | 120000 |
| LVM | 49000 | 51000 | 59000 | 74000 | 105000 |

Figure 7-53 on page 268 shows a comparison of native I/O and VSCSI Cycles per Byte (CPB) using both logical volume backed storage and physical disk backed storage. The VSCSI measures are of the I/O server only, the client overhead is not included in the comparison. The processor efficiency of I/O improves with larger I/O size. Effectively, there is a fixed overhead to start and complete an I/O, with some additional cycle time based on the size of the I/O.

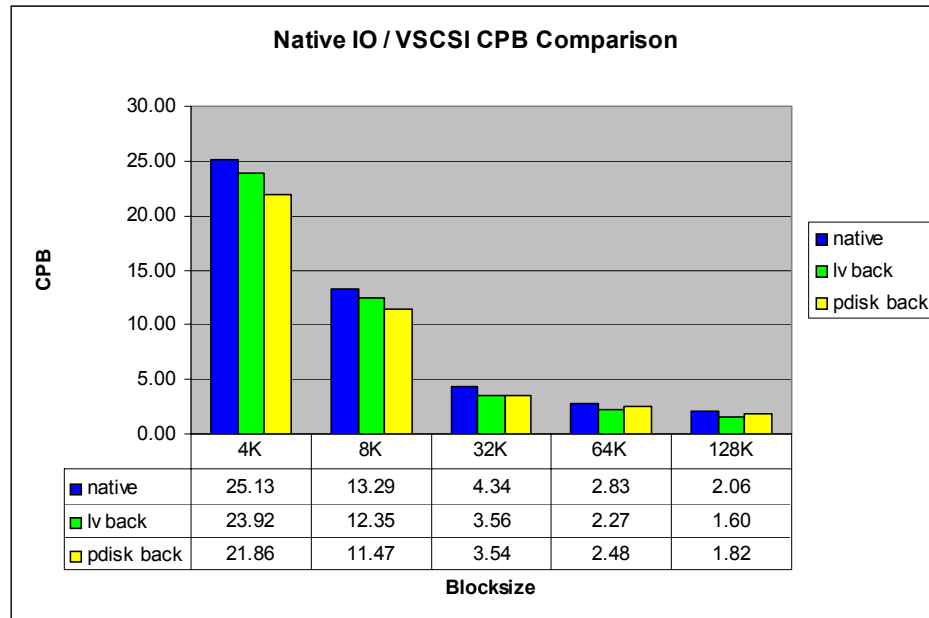


Figure 7-53 Comparison of native I/O to VSCSI

Configuration example

Consider an I/O server that supports three client partitions on physical disk backed storage. The first client partition requires a maximum of 7,000 8KB operations per second. The second client partition requires a maximum of 10,000 8KB operations per second. The third client partition requires a maximum of 5,000 128KB operations per second. The number of 1.65Ghz processors for this requirement is approximately:

$$(7,000 \times 47,000 + 10,000 \times 47,000 + 5,000 \times 120,000) / 1,650,000,000 = 0.85 \text{ processors}$$

which we need to round up to a single processor since we are not using Micro-Partitioning.

An alternative approach, if you don't know the I/O rates of the client partitions, is to size the VSCSI server to the maximum I/O rate of the storage subsystem attached. The sizing could be biased to small I/O's or large I/O's. Sizing to

maximum capacity for large I/O's will balance the processor capacity of the VSCSI server to the potential I/O bandwidth of the attached I/O. The negative facet of this sizing methodology is that we will, in nearly every case, assign more processor entitlement to the VSCSI server than it will typically consume.

Consider a case where an I/O server manages 32 physical SCSI disks. We can arrive at an upper bound of processors required based on assumptions about the I/O rates that the disks can achieve. If it is known that the workload is dominated by 8KB operations which are random, we could assume that each disk is capable of approximately 200 disk I/O's per second (15Krpm drives). At peak, the I/O server would need to service approximately 32 disks * 200 I/O's per second times 120,000 cycles per operation, resulting in a requirement for approximately 19% of one processor performance. Viewed another way, an I/O server running on a single processor should be capable of supporting more than 150 disks doing 8KB random I/O's.

Alternatively, if the server is sized for maximum bandwidth, the calculation will result in a much higher processor requirement. The difference is that maximum bandwidth assumes sequential I/O. Since disks are much more efficient doing large sequential I/O's than small random I/O's, we can drive a much higher number of I/O's per second. Assume that the disks are capable of 50MB/sec. when doing 128KB I/O's. That implies each disk could average 390 disk I/O's per second. Thus, the entitlement necessary to support 32 disks, each doing 390 I/O's per second with an operation cost of 120,000 cycles ($32 * 390 * 120,000 / 1,650,000,000$) approximately 0.91 processors. More simply, an I/O server running on a single processor should be capable of driving approximately 32 fast disks to maximum throughput.

This sizing method can be very wasteful of processor entitlement when using dedicated processor partitions but will guarantee peak performance. It is most effective if the average I/O size can be estimated, so that peak bandwidth does not need to be assumed.

VSCSI Server sizing using Micro-Partitioning

Defining VSCSI servers in Micro-partitions allows much better granularity of processor resource sizing and potential recovery of unused processor time by uncapped partitions. Tempering those benefits, use of Micro-partitions for VSCSI servers will frequently increase I/O response time and make for somewhat more complex processor entitlement sizings.

The sizing methodology should be based on the same operation costs for dedicated partition I/O servers. However, addition entitlement should be added for running in Micro-partitions. We would recommend that the I/O server be configured as uncapped, that way if the I/O server is under sized, there is opportunity to get more processor time to service I/O.

Since I/O latency with virtual SCSI will vary with the machine utilization and I/O server topology, consider the following:

1. If a partition has very high I/O requirements or requires very low latency, try to configure it with native I/O.
2. If native I/O is not a good option and the system contains enough processors, consider putting the I/O server in a dedicated processor partition.
3. If it is necessary to use a Micro-Partitioning I/O server, use as few virtual processors as possible because this will minimize the overhead of the Hypervisor.

VSCSI Server Memory Sizing

The architecture of VSCSI simplifies memory sizing in that there is no caching of file data in the memory of the VSCSI server. Since there is no caching of data, the memory requirements for the VSCSI server are fairly modest. With large I/O configurations and very high data rates, a 1GB memory allocation for the VSCSI server is more than sufficient. For low I/O rate cases with a small number of attached disks, 512MB is a sufficient memory allocation.



Part 4

Performance monitoring and management

Note to Author: Optionally, describe the book part here. If you are not using Part files, you need to restart the page numbering in the first chapter file of your book:

- ▶ In FrameMaker 6.0: **Open .book > select first chapter> Format > Document > Numbering > Page “tab” > select First Page # radio button and set Page # to 1 > Set**
 - Now delete the three Part files (p01, p02 and p03) from your book:
Open .book > select a part file > Edit > Delete File from Book

In this part we introduce/provide/describe/discuss...

8



POWER5 system performance

This chapter provides information and knowledge on system performance, how to diagnose a processor, a memory or an I/O problem. We mainly focus on new components introduced by the p5 architecture and virtualization.

The following topics are discussed:

- ▶ AIX commands useful for performance analysis.
- ▶ Approach to performance and tuning on a system.

8.1 Performance commands

Table 8-1 on page 274 summarizes all the AIX commands described in this section. Those commands are used to show typical performance issues in this chapter.

Table 8-1 Commands summary

| Command | Function | Main measurement | Page number |
|----------|---|------------------|-------------|
| lparstat | Logical partition information and statistics | CPU, Hypervisor | 274 |
| mpstat | Physical and logical processors statistics | CPU | 281 |
| vmstat | CPU and virtual memory monitoring | CPU, memory | 285 |
| iostat | System input/output device monitoring | Disk I/O | 287 |
| sar | Physical, logical processors and I/O monitoring | CPU | 289 |
| topas | Displays dynamically system statistics. | CPU, memory, I/O | 292 |
| xmperf | Displays a great amount of system statistics | CPU, memory, I/O | 295 |

8.1.1 lparstat command

The **lparstat** command reports logical partition information and statistics as well as Hypervisor statistics. It displays on its first line a summary of the partition configuration. Table 8-2 on page 274 gives a summary for **lparstat** command (a + sign means the command covers this topic).

Table 8-2 lparstat command summary

| Command name | lparstat |
|------------------------|----------|
| Interface type | CLI |
| Updated or new command | new |
| AIX package | bos.acct |

| Command name | lparstat |
|--|------------------------|
| Measurement CPU memory disk I/O network Hypervisor | + + |
| Environment Dedicated Partition Micro-Partition SMT I/O server | + + + |

Usage

```
lparstat { -i | [-H | -h] [Interval [Count]] }
```

Most important flags

- i

Displays information about the configuration of the logical partition.
- h

Adds summary Hypervisor information to the default output.
- H

Displays detailed Hypervisor information, including statistics for each of the Hypervisor calls.

Output Examples

In the default mode and on a dedicated partition, **lparstat** command shows the processor utilization in the usual manner (%user, %sys, %idle, %wait) as shown in Example 8-1 on page 275.

Example 8-1 lparstat default mode on a dedicated partition

```
# lparstat 2

System configuration: type=Dedicated mode=Capped smt=0n lcpu=2 mem=1024

%user  %sys  %wait  %idle
-----
 99.2   0.7   0.0   0.0
 99.7   0.3   0.0   0.0
 99.8   0.2   0.0   0.0
```

In the default mode and on a Micro-Partition, **lparstat** command adds the following information to the output as shown in Example 8-2 on page 276:

| | |
|--------------|--|
| psize | Number of online physical processors in the shared pool. |
| physc | Shows the number of physical processor consumed. |
| %entc | Shows the percentage of the entitled capacity consumed. |
| lbusy | Shows the percentage of logical processor(s) utilization that occurred while executing at the user and system level. |
| app | Shows the available processing capacity in the shared pool. |
| vcsw | Number of virtual context switches which are the virtual processor hardware preemptions. |
| phint | Shows the number of phantom (targeted to another shared partition in this pool) interruptions received. |

Example 8-2 lparstat default mode on a Micro-partition

| | | | | | | | | | |
|--|-------|-------|-------|-------|-------|-------|------|--------|-------|
| # lparstat | | | | | | | | | |
| System configuration: type=Shared mode=Uncapped smt=0n lcpu=6 mem=512 psize=1 ent=0.30 | | | | | | | | | |
| %user | %sys | %wait | %idle | physc | %entc | lbusy | app | vcsw | phint |
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | --- | ----- | ----- |
| 8.2 | 3.0 | 0.2 | 88.6 | 0.00 | 0.2 | 3.3 | 0.89 | 168708 | 148 |

An *interval* and a *count* can be added to the command, it displays statistics every *interval* seconds for *count* iterations. In Example 8-3 on page 276 the interval is 2 seconds and the count is 5.

Example 8-3 lparstat monitoring

| | | | | | | | | | |
|--|-------|-------|-------|-------|-------|-------|------|-------|-------|
| # lparstat 2 5 | | | | | | | | | |
| System configuration: type=Shared mode=Uncapped smt=0n lcpu=6 mem=512 psize=1 ent=0.30 | | | | | | | | | |
| %user | %sys | %wait | %idle | physc | %entc | lbusy | app | vcsw | phint |
| ----- | ----- | ----- | ----- | ----- | ----- | ----- | --- | ----- | ----- |
| 70.2 | 14.1 | 0.0 | 15.6 | 0.53 | 175.9 | 23.4 | 0.00 | 2453 | 31 |
| 62.6 | 19.8 | 0.0 | 17.6 | 0.49 | 161.7 | 18.9 | 0.08 | 2611 | 24 |
| 67.6 | 16.8 | 0.0 | 15.6 | 0.35 | 117.9 | 14.8 | 0.23 | 2409 | 15 |
| 52.6 | 20.5 | 0.0 | 26.8 | 0.27 | 88.6 | 11.4 | 0.27 | 2486 | 8 |
| 61.5 | 21.2 | 0.0 | 17.4 | 0.32 | 106.3 | 11.9 | 0.22 | 2829 | 13 |

If the partition does not have the “shared processor pool utilization authority”, the *app* column will not be displayed as shown in Example 8-4 on page 277. This

option allows the logical partition to collect information statistics from the managed system about shared processing pool utilization. Shared processors are processors that are shared between two or more logical partitions. The processors are held in the shared processor pool and are shared among the logical partitions.

In order to select this option, connect to the Hardware Management Console (HMC), edit the partition properties, select **Hardware** tab, **Processor and Memory** tab and then select the **Allow shared processor utilization authority** check box like in Example 8-1 on page 278.

Example 8-4 Logical partition without “pool utilization authority”

| | | | | | | | | | |
|--|------|-------|-------|-------|-------|-------|------|-------|---|
| # lparstat 2 5 | | | | | | | | | |
| System configuration: type=Shared mode=Uncapped smt=0n lcpu=6 mem=512 psize=1 ent=0.30 | | | | | | | | | |
| %user | %sys | %wait | %idle | physc | %entc | lbusy | vcsw | phint | |
| ----- | ---- | ----- | ----- | ----- | ----- | ----- | ---- | ----- | |
| 67.8 | 16.7 | 0.0 | 15.5 | 0.66 | 219.6 | 16.2 | 3343 | | 3 |
| 60.1 | 21.6 | 0.1 | 18.2 | 0.47 | 157.1 | 7.3 | 2402 | | 2 |
| 65.6 | 18.7 | 0.0 | 15.6 | 0.86 | 287.9 | 17.3 | 3714 | | 2 |
| 67.2 | 16.8 | 0.0 | 16.0 | 0.71 | 235.6 | 16.4 | 3411 | | 2 |
| 67.4 | 16.5 | 0.0 | 16.1 | 0.66 | 220.7 | 17.3 | 3299 | | 2 |

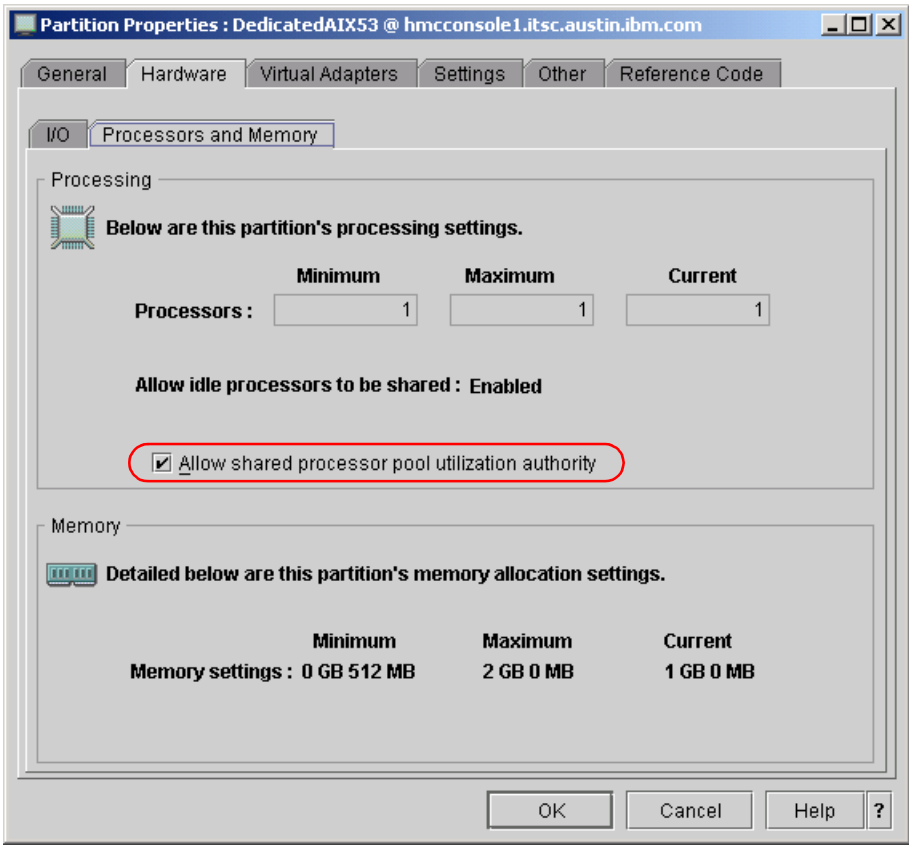


Figure 8-1 “shared processor utilization authority” activation

The -h flag adds the percentage of time spent in Hypervisor (*%hypv*) and the number of Hypervisor calls executed (*hcalls*) to the default output as shown in Example 8-5 on page 278.

Example 8-5 *lparsta -h* output

```
# lparstat -h 2

System configuration: type=Shared mode=Capped smt=Off lcpu=2 mem=512 psize=2
ent=0.30

%user  %sys  %wait  %idle  physc  %entc  lbusy   app   vcsw  phint  %hypv  hcalls
-----
 98.4   1.1    0.0    0.5   0.30  99.7   58.7   0.69  309   3     0.6   153
 98.0   1.1    0.0    0.9   0.30  99.3   55.5   0.69  303   1     0.5   146
 97.9   1.1    0.0    1.0   0.30  99.2   56.9   0.69  304   0     0.6   147
 93.7   1.1    0.0    5.2   0.28  95.0   58.4   0.70  292   0     0.6   143
```


| | | | | | | | | | | | |
|------|-----|-----|------|------|------|------|------|-----|---|-----|-----|
| 72.2 | 0.9 | 0.0 | 26.9 | 0.22 | 73.3 | 59.5 | 0.77 | 208 | 0 | 0.5 | 91 |
| 95.3 | 1.0 | 0.0 | 3.7 | 0.29 | 96.4 | 53.2 | 0.70 | 184 | 0 | 0.2 | 36 |
| 86.9 | 1.0 | 0.0 | 12.2 | 0.26 | 88.0 | 55.9 | 0.72 | 258 | 0 | 0.4 | 100 |
| 95.9 | 1.1 | 0.0 | 3.0 | 0.29 | 97.2 | 52.8 | 0.70 | 298 | 0 | 0.6 | 146 |
| 98.1 | 1.0 | 0.0 | 0.9 | 0.30 | 99.4 | 56.4 | 0.69 | 312 | 1 | 0.5 | 157 |
| 98.4 | 1.1 | 0.0 | 0.5 | 0.30 | 99.7 | 59.8 | 0.69 | 314 | 0 | 0.6 | 162 |
| 97.4 | 1.1 | 0.0 | 1.5 | 0.30 | 98.7 | 55.6 | 0.69 | 303 | 0 | 0.6 | 157 |
| 98.2 | 1.1 | 0.0 | 0.7 | 0.30 | 99.5 | 56.3 | 0.69 | 311 | 0 | 0.5 | 158 |

To get information about the partition, like minimum, maximum of CPU and memory, partition type and mode use **lparstat -i** command as shown in Example 8-6 on page 279.

The partition type can be one of the following:

| | |
|----------------------|--|
| Dedicated | Processors are dedicated to the partition, Simultaneous Multi-Threading (SMT) mode is off. |
| Dedicated-SMT | Processors are dedicated to the partition, SMT mode is on. |
| Shared | Partition is configured for Micro-Partitioning, SMT mode is off. |
| Shared-SMT | Partition is configured for Micro-Partitioning, SMT mode is on. |

For more information on SMT refer to Chapter 5, “Simultaneous Multi-Threading (SMT)” on page 89.

The partition mode can be:

| | |
|-----------------|--|
| Capped | Partition is not allowed to consume idle cycles from the shared pool. Dedicated LPAR is implicitly capped. |
| Uncapped | Partition may use idle cycles from the shared pool if needed. |

For more information on Micro-Partitions mode refer to

Author Comment: Add XXXRef to Micro-Partitioning chapter, section 6.2.2 Type of Micro-Partitions

Example 8-6 *lparstat -i* output

| | |
|----------------|-----------------------|
| # lparstat -i | |
| Node Name | : LPARmicro |
| Partition Name | : MicroPartitionAIX53 |

| | |
|---------------------------------|--------------|
| Partition Number | : 4 |
| Type | : Shared-SMT |
| Mode | : Uncapped |
| Entitled Capacity | : 0.30 |
| Partition Group-ID | : 32772 |
| Shared Pool ID | : 0 |
| Online Virtual CPUs | : 3 |
| Maximum Virtual CPUs | : 5 |
| Minimum Virtual CPUs | : 2 |
| Online Memory | : 512 MB |
| Maximum Memory | : 1024 MB |
| Minimum Memory | : 128 MB |
| Variable Capacity Weight | : 128 |
| Minimum Capacity | : 0.20 |
| Maximum Capacity | : 0.50 |
| Capacity Increment | : 0.01 |
| Maximum Dispatch Latency | : 17995218 |
| Maximum Physical CPUs in system | : 2 |
| Active Physical CPUs in system | : 2 |
| Active CPUs in Pool | : 1 |
| Unallocated Capacity | : 0.00 |
| Physical CPU Percentage | : 10.00% |
| Unallocated Weight | : 0 |

Detailed information about the Hypervisor calls are displayed by **lparstat -H** command as shown in Example 8-7 on page 280, specially the *cede* and *confer* values used by the operating system to give back processor resource to the hardware when it no longer has a demand for it or when it is waiting on an event to complete. In this example, time spend for cede is 99.2% of the total time spend in the Hypervisor.

Example 8-7 *lparstat -H* output

```
# lparstat -H 2 1

System configuration: type=Shared mode=Uncapped smt=0n lcpu=6 mem=512 psize=1
ent=0.30
```

Detailed information on Hypervisor Calls

| Hypervisor Call | Number of Calls | %Total Time Spent | %Hypervisor Time Spent | Avg Call Time(ns) | Max Call Time(ns) |
|-----------------|-----------------|-------------------|------------------------|-------------------|-------------------|
| remove | 6 | 0.0 | 0.0 | 268 | 647 |
| read | 2 | 0.0 | 0.0 | 50 | 449 |
| nclear_mod | 0 | 0.0 | 0.0 | 1 | 0 |
| page_init | 3 | 0.0 | 0.0 | 405 | 1989 |
| clear_ref | 0 | 0.0 | 0.0 | 1 | 0 |

| | | | | | |
|---------------------|-----|-----|------|-------|-------|
| protect | 0 | 0.0 | 0.0 | 1 | 0 |
| put_tce | 141 | 0.0 | 0.2 | 689 | 2100 |
| xirr | 81 | 0.0 | 0.2 | 790 | 2791 |
| eoi | 80 | 0.0 | 0.1 | 449 | 927 |
| ipi | 0 | 0.0 | 0.0 | 1 | 0 |
| cppr | 80 | 0.0 | 0.0 | 221 | 434 |
| asr | 0 | 0.0 | 0.0 | 1 | 0 |
| others | 0 | 0.0 | 0.0 | 1 | 0 |
| enter | 11 | 0.0 | 0.0 | 250 | 874 |
| cede | 211 | 6.9 | 99.2 | 51344 | 70485 |
| migrate_dma | 0 | 0.0 | 0.0 | 1 | 0 |
| put_rtce | 0 | 0.0 | 0.0 | 1 | 0 |
| confer | 0 | 0.0 | 0.0 | 1 | 0 |
| prod | 151 | 0.0 | 0.1 | 338 | 1197 |
| get_ppp | 1 | 0.0 | 0.0 | 850 | 2583 |
| set_ppp | 0 | 0.0 | 0.0 | 1 | 0 |
| purr | 0 | 0.0 | 0.0 | 1 | 0 |
| pic | 1 | 0.0 | 0.0 | 125 | 758 |
| bulk_remove | 0 | 0.0 | 0.0 | 1 | 0 |
| send_crq | 70 | 0.0 | 0.1 | 777 | 2863 |
| copy_rdma | 0 | 0.0 | 0.0 | 1 | 0 |
| get_tce | 0 | 0.0 | 0.0 | 1 | 0 |
| send_logical_lan | 3 | 0.0 | 0.0 | 2178 | 4308 |
| add_logical_lan_buf | 9 | 0.0 | 0.0 | 638 | 1279 |
| ----- | | | | | |

8.1.2 mpstat command

The **mpstat** command collects and displays performance statistics for all logical CPUs in the system. It can show up to 29 new metrics (when using -a option). Table 8-3 on page 281 gives a summary on **mpstat** command.

Table 8-3 mpstat command summary

| Command name | mpstat |
|---|----------|
| Interface type | CLI |
| Updated or new command | new |
| AIX package | bos.acct |
| Measurement Processor Memory Disk I/O Network Hypervisor | + |

| Command name | mpstat |
|---------------------|--------|
| Environment | |
| Dedicated Partition | + |
| Micro-Partition | ++ |
| SMT | ++ |
| I/O server | |

Usage

mpstat [{ -a | -d | -i | -s }] [-w] [interval [count]]

Most important flags

- d Displays detailed affinity and migration statistics for AIX threads.
- i Displays detailed interrupt statistics.
- s Displays SMT utilization report if SMT is enabled.

The default mode shows:

- Utilization metrics (%user, %sys, %idle, %wait).
- Major and minor page faults (with and without disk I/O).
- Number of syscalls and interrupts.
- Dispatcher metrics namely the number of migrations, voluntary and involuntary context switches, logical processor affinity (percentage of redispatches inside MCM), and the run queue size.
- Fraction of processor consumed (SMT or Micro-Partitioning mode only).
- Percentage of entitlement consumed (Micro-Partitioning mode only).
- Number of logical context switches (Micro-Partitioning mode only) meaning the hardware preemptions.

Output Examples

The default mode of **mpstat** command shown in Example 8-8 on page 283 displays the following information to show activity for each logical processor:

- mpc** Total number of inter-processor calls.
- cs** Total number of logical processor context switches.
- ics** Total number of involuntary context switches.
- mig** Total number of thread migrations to another logical processor.

lpa

Logical processor affinity. The percentage of logical processor redispatches within the scheduling affinity domain 3 (same Multi Chip Module).

At the end of the output, the *U* line displays the unused capacity, the *ALL* line is the sum of all virtual processors.

Example 8-8 mpstat default output

mpstat 2 2

System configuration: lcpu=6 ent=0.3

| cpu | min | maj | mpc | int | cs | ics | rq | mig | lpa | sysc | us | sy | wa | id | pc | %ec | lcs |
|-----|-----|-----|-----|------|-----|-----|----|-----|-----|------|----|----|----|----|------|------|-----|
| 0 | 135 | 0 | 0 | 688 | 358 | 179 | 0 | 0 | 100 | 3343 | 38 | 57 | 0 | 4 | 0.04 | 12.0 | 295 |
| 1 | 0 | 0 | 0 | 46 | 0 | 0 | 0 | 0 | - | 0 | 0 | 11 | 0 | 89 | 0.01 | 3.1 | 290 |
| 2 | 0 | 0 | 0 | 188 | 100 | 50 | 0 | 0 | 100 | 0 | 1 | 51 | 0 | 47 | 0.00 | 0.9 | 179 |
| 3 | 0 | 0 | 0 | 43 | 0 | 0 | 0 | 0 | - | 0 | 0 | 59 | 0 | 41 | 0.00 | 0.9 | 179 |
| 4 | 0 | 0 | 0 | 56 | 308 | 157 | 0 | 1 | 100 | 9 | 0 | 69 | 0 | 31 | 0.00 | 1.3 | 215 |
| 5 | 0 | 0 | 0 | 37 | 0 | 0 | 0 | 1 | 100 | 0 | 0 | 30 | 0 | 70 | 0.00 | 0.7 | 195 |
| U | - | - | - | - | - | - | - | - | - | - | - | - | 0 | 81 | 0.24 | 81.3 | - |
| ALL | 135 | 0 | 0 | 1058 | 766 | 386 | 0 | 2 | 100 | 3352 | 5 | 9 | 0 | 86 | 0.06 | 18.9 | 676 |

-

| | | | | | | | | | | | | | | | | | |
|-----|----|---|---|-----|-----|-----|---|---|-----|-----|----|----|---|----|------|------|-----|
| 0 | 29 | 0 | 0 | 261 | 34 | 17 | 0 | 0 | 100 | 725 | 43 | 52 | 0 | 4 | 0.01 | 3.6 | 69 |
| 1 | 0 | 0 | 0 | 14 | 0 | 0 | 0 | 0 | - | 0 | 0 | 12 | 0 | 88 | 0.00 | 0.9 | 69 |
| 2 | 15 | 0 | 0 | 166 | 108 | 53 | 0 | 0 | 100 | 5 | 3 | 59 | 0 | 37 | 0.00 | 0.8 | 129 |
| 3 | 0 | 0 | 0 | 15 | 0 | 0 | 0 | 0 | - | 0 | 0 | 29 | 0 | 71 | 0.00 | 0.5 | 129 |
| 4 | 0 | 0 | 0 | 23 | 60 | 35 | 0 | 0 | 100 | 0 | 0 | 66 | 0 | 34 | 0.00 | 0.4 | 61 |
| 5 | 0 | 0 | 0 | 13 | 0 | 0 | 0 | 0 | - | 0 | 0 | 20 | 0 | 80 | 0.00 | 0.2 | 59 |
| U | - | - | - | - | - | - | - | - | - | - | - | - | 0 | 94 | 0.28 | 93.7 | - |
| ALL | 44 | 0 | 0 | 492 | 202 | 105 | 0 | 0 | 100 | 730 | 2 | 3 | 0 | 96 | 0.02 | 6.3 | 258 |

In the **mpstat -d** output shown in Example 8-9 on page 283, the *rq* column shows the run queue size for each logical processor. The columns from *S0rd* to *S5rd* show the percentage of thread redispatches within a scheduling affinity domain.

Author Comment: add XXXref to redispach affinity explanation

Example 8-9 mpstat -d output

mpstat -d 2 1

System configuration: 1cpu=6 ent=0.3

| cpu | cs | ics | bound | rq | push | S3pull | S3grd | S0rd | S1rd | S2rd | S3rd | S4rd | S5rd | ilcs | vlcs |
|-----|-----|-----|-------|----|------|--------|-------|------|-------|------|------|------|------|------|------|
| 0 | 202 | 150 | 0 | 0 | 0 | 0 | 0 | 99.1 | 0.0 | 0.0 | 0.9 | 0.0 | 0.0 | 134 | 285 |
| 1 | 5 | 1 | 0 | 0 | 0 | 0 | 0 | 0.0 | 100.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0 | 411 |
| 2 | 222 | 158 | 0 | 0 | 0 | 0 | 0 | 97.1 | 2.2 | 0.0 | 0.7 | 0.0 | 0.0 | 182 | 330 |
| 3 | 28 | 15 | 0 | 0 | 0 | 0 | 0 | 93.3 | 6.7 | 0.0 | 0.0 | 0.0 | 0.0 | 0 | 520 |
| 4 | 320 | 223 | 0 | 0 | 0 | 0 | 0 | 98.9 | 0.7 | 0.0 | 0.4 | 0.0 | 0.0 | 83 | 370 |
| 5 | 11 | 9 | 0 | 0 | 0 | 0 | 0 | 86.7 | 13.3 | 0.0 | 0.0 | 0.0 | 0.0 | 2 | 452 |
| ALL | 788 | 556 | 0 | 0 | 0 | 0 | 0 | 97.1 | 2.4 | 0.0 | 0.5 | 0.0 | 0.0 | 200 | 1184 |

A logical partition receives different kind of interrupts, Example 8-10 on page 284 shows for each logical processor the following interrupt metrics:

- mpcs, mpcr**Interrupts used to communicate between processors.
- dev**Number of hardware interrupts, in other words external interrupts.
- soft**Number of software interrupts, when a hardware interrupt takes too much time to complete, a software interrupt is created to finish the processing.
- dec**Number of decrementer interrupts. The decrementer is the register used to generate time-based interrupts. AIX loads a value in it, the processor decrements the register and when it reaches zero, an interrupt is sent.
- ph**Number of phantom interrupts, the number of device interrupts received by the partition but targeted to another partition in the pool. Operating system simply returns those to Hypervisor.

Example 8-10 mpstat -i output

mpstat -i 2 1

System configuration: 1cpu=6 ent=0.3

| cpu | mpcs | mpcr | dev | soft | dec | ph |
|-----|------|------|-----|------|-----|----|
| 0 | 0 | 0 | 20 | 5 | 105 | 0 |
| 1 | 0 | 0 | 23 | 0 | 10 | 0 |
| 2 | 0 | 0 | 25 | 48 | 195 | 1 |
| 3 | 0 | 0 | 24 | 0 | 11 | 0 |
| 4 | 0 | 0 | 23 | 0 | 101 | 0 |
| 5 | 0 | 0 | 21 | 0 | 11 | 0 |
| ALL | 0 | 0 | 136 | 53 | 433 | 1 |

If SMT mode is turned on, the **mpstat -s** command displays physical as well as logical processors usage as shown in Example 8-11 on page 285. Physical

processor *Proc0* is busy at 17.80%, which is dispatched on logical processor *cpu0* (14.75%) and on logical processor *cpu1* (3.05%). In this case, *cpu0* and *cpu1* are hardware threads for *proc0*.

Example 8-11 *mpstat -s* output

mpstat -s 2 1

System configuration: lcpu=6 ent=0.3

| | | | | | |
|--------|-------|--------|-------|--------|-------|
| Proc0 | | Proc2 | | Proc4 | |
| 17.80% | | 16.24% | | 13.67% | |
| cpu0 | cpu1 | cpu2 | cpu3 | cpu4 | cpu5 |
| 14.75% | 3.05% | 13.51% | 2.73% | 11.18% | 2.49% |

8.1.3 vmstat command

The **vmstat** command reports statistics about kernel threads, virtual memory, disks, traps and processor activity. Table 8-4 on page 285 gives a summary for **vmstat** command.

Table 8-4 *vmstat* command summary

| Command name | vmstat |
|--|---------------|
| Interface type | CLI |
| Updated or new command | update |
| AIX package | bos.acct |
| Measurement Processor Memory Disk I/O Network Hypervisor | ++ ++ + |
| Environment Dedicated Partition Micro-Partition SMT I/O server | + + + |

Usage

vmstat [-fsviItlw] [Drives] [Interval [Count]]

Most important flags

- t Prints the time-stamp next to each line of output.
- v Writes to standard output various statistics maintained by the Virtual Memory Manager.

Output examples

In Example 8-12 on page 286 we are running `vmstat` command on an uncapped Micro-Partition with 0.3 processing unit. At the beginning the partition is idle, the processor consumed *pc* is 0 and the percentage of entitlement consumed *ec* is 1.4%.

Then activity begins on the partition and the percentage of used CPU increases to 93%, due to the fact that the partition is uncapped and the processor pool is not fully consumed, the percentage of entitlement consumed goes up to 330%, and the processor consumed is nearly 1. This means that the partition is running on almost a full processor although it has been given only 0.3 processing unit.

Example 8-12 Activity on uncapped partition shown by `vmstat` command

```
# lparstat

System configuration: type=Shared mode=Uncapped smt=0n lcpu=6 mem=512 psize=2
ent=0.30

%user  %sys  %wait  %idle  physc  %entc  lbusy  app  vcsw  pshint
-----
  0.0   0.0   0.2   99.7   0.00   0.1    0.4   1.17 1063520 333

# vmstat 5

System configuration: lcpu=6 mem=512MB ent=0.30

kthr    memory                page                faults                cpu
-----
r  b   avm   fre  re  pi  po  fr  sr  cy  in  sy  cs  us  sy  id  wa   pc   ec
0  0 46863 1408  0  0  0  0  0  0  0  32 139  0  0 99  0  0.00  1.4
0  0 46865 1406  0  0  0  0  0  0  3  26 177  0  1 99  0  0.00  1.4
1  0 46903 1368  0  0  0  0  0  0  1 1566 143 93  1  6  0  0.35 118.3
1  0 46903 1368  0  0  0  0  0  0  1 4323 137 93  0  6  0  0.99 330.8
1  0 46903 1368  0  0  0  0  0  0  0 4333 137 93  0  6  0  0.99 330.8
1  0 46903 1368  0  0  0  0  0  0  0 4330 139 93  0  6  0  0.99 330.7
1  0 46903 1368  0  0  0  0  0  0  0 4259 130 93  0  6  0  0.99 330.5
1  0 46903 1368  0  0  0  0  0  0  1 4010 130 93  1  6  0  0.91 301.9
1  0 46903 1368  0  0  0  0  0  0  0 2934 136 93  1  6  0  0.68 225.5
1  0 46903 1368  0  0  0  0  0  0  0 3578 140 93  1  6  0  0.82 272.6
1  0 46903 1368  0  0  0  0  0  0  1 2694 134 93  1  7  0  0.63 208.4
```


| | | | | | | | | | | | | | | | | | | |
|---|---|-------|------|---|---|---|---|---|---|---|------|-----|----|---|----|---|------|-------|
| 1 | 0 | 46865 | 1406 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2311 | 136 | 93 | 1 | 6 | 0 | 0.53 | 177.6 |
| 0 | 0 | 46865 | 1406 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 8 | 136 | 0 | 0 | 99 | 0 | 0.00 | 1.1 |
| 0 | 0 | 46865 | 1406 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 10 | 142 | 0 | 0 | 99 | 0 | 0.00 | 1.2 |

8.1.4 iostat command

The **iostat** command is used for monitoring system input/output device loading. Table 8-5 on page 287 gives a summary for **iostat** command.

Table 8-5 iostat command summary

| Command name | iostat |
|--|-------------------------|
| Interface type | CLI |
| Updated or new command | update |
| AIX package | bos.acct |
| Measurement Processor Memory Disk I/O Network Hypervisor | + ++ |
| Environment Dedicated Partition Micro-Partition SMT I/O server | + + + + |

Usage

```
iostat [-astTdmAPqQ1] [Drives] [Interval [Count]]
```

Most important flags

- d Displays drive report only.
- t Displays tty/cpu report only.
- T Prints the time-stamp next to each line of output.

Output examples

The output of **iostat** command in Example 8-13 on page 288 shows the two new columns:

%physc The percentage of physical processor consumed.

%entc The percentage of entitled capacity consumed.

Example 8-13 iostat command

```
# iostat 2 1

System configuration: lcpu=6 drives=2 ent=0.30

tty:  tin  tout  avg-cpu:  % user  % sys  % idle  % iowait  % physc  % entc
      0.0  25.0           58.4   22.6   19.1    0.0    0.5   182.4

Disks:      % tm_act    Kbps    tps    Kb_read  Kb_wrtn
hdisk0      40.0      440.4    110.6        0      880
cd0          0.0       0.0     0.0         0        0
```

A new way to look at asynchronous I/O is provided by **iostat** command. You can either check the statistics of “legacy” asynchronous I/O or POSIX asynchronous I/O. You can use several flags:

- A** Shows processor utilization and asynchronous I/O statistics.
- q** Shows adapter individual queues and their request counts.
- Q** Shows mounted file systems and their associated adapter queue and request counts.
- P** Is similar to -A option, but for the POSIX adapter extension data.

When using -A or -P, new columns replace the tty information as shown in Example 8-14 on page 288:

- avgc** Average global non fastpath adapter request count per second for the specified interval.
- avfc** Average fastpath request count per second for the specified interval.
- maxg** Maximum global non fastpath adapter request count since the last time it fetched this value.
- maxf** Maximum fastpath request count since the last time it fetched this value.
- maxr** Maximum adapter I/O requests allowed on queue.

Example 8-14 iostat “legacy” adapter I/O

```
# iostat -A 3 3
```

```
System configuration: lcpu=3 drives=3 ent=2.00

aio: avgc avfc maxg maxf maxr avg-cpu: %user %sys %idle %iow physc %entc
      0    0    0    0 4096          42.9 23.4 13.4 20.3   0.0 80.4

Disks:      % tm_act      Kbps      tps      Kb_read      Kb_wrtn
hdisk0      0.0          0.0          0.0          0          0
hdisk2      24.3        2293.4        573.4        6972         0
hdisk1      98.0        32000.0       125.0        48640       48640

aio: avgc avfc maxg maxf maxr avg-cpu: %user %sys %idle %iow physc %entc
      0    0    0    0 4096          43.5 23.3 12.6 20.6   0.0 81.3

Disks:      % tm_act      Kbps      tps      Kb_read      Kb_wrtn
hdisk0      0.0          0.0          0.0          0          0
hdisk2      21.3        2328.0        582.0        6984         0
hdisk1      99.3        28832.3       127.3        42041       44456

aio: avgc avfc maxg maxf maxr avg-cpu: %user %sys %idle %iow physc %entc
      0    0    0    0 4096          43.0 23.5 13.2 20.3   0.0 80.9

Disks:      % tm_act      Kbps      tps      Kb_read      Kb_wrtn
hdisk0      0.0          0.0          0.0          0          0
hdisk2      24.0        2265.3        566.3        6796         0
hdisk1      98.3        28576.3       126.3        41785       43944
```

Important: Some system resources are consumed in maintaining disk I/O history for the **iostat** command. Use the **sysconfig** subroutine, or the System Management Interface Tool (SMIT) to stop history accounting if it is not needed.

8.1.5 sar command

The **sar** command writes to standard output the contents of selected cumulative activity counters in the operating system.

Table 8-6 on page 289 gives a summary for **sar** command.

Table 8-6 sar command summary

| | |
|------------------------|----------|
| Command name | sar |
| Interface type | CLI |
| Updated or new command | update |
| AIX package | bos.acct |

| Command name | sar |
|--|-------------|
| Measurement Processor Memory Disk I/O Network Hypervisor | + + |
| Environment Dedicated Partition Micro-Partition SMT I/O server | + + + |

Usage

sar [-Aabcdkmqruvwy] [Interval [Number]

Most important flags

- t Prints the time-stamp next to each line of output.
- v Writes to standard output various statistics maintained by the Virtual Memory Manager.

Output examples

The default output of **sar** command in Example 8-15 on page 290 shows the two new columns:

- physc** The number of physical processor consumed.
- %entc** The percentage of entitled capacity consumed.

Example 8-15 Default sar output

sar 2 5

AIX LPARmicro 3 5 00CDDEDC4C00 10/20/04

System configuration: lcpu=6 ent=0.30

| 11:28:00 | %usr | %sys | %wio | %idle | physc | %entc |
|----------|------|------|------|-------|-------|-------|
| 11:28:02 | 70 | 16 | 0 | 15 | 0.67 | 224.1 |
| 11:28:04 | 62 | 20 | 0 | 18 | 0.47 | 156.0 |
| 11:28:06 | 68 | 17 | 0 | 15 | 0.78 | 258.6 |
| 11:28:08 | 66 | 18 | 0 | 16 | 0.78 | 258.9 |
| 11:28:10 | 68 | 16 | 0 | 16 | 0.65 | 216.9 |

| | | | | | | |
|---------|----|----|---|----|------|-------|
| Average | 67 | 17 | 0 | 16 | 0.67 | 222.9 |
|---------|----|----|---|----|------|-------|

When looking at the -P ALL for all logical processors view option of the **sar** command with SMT on or in Micro-Partitioning in Example 8-16 on page 291, it shows the physical processor consumed *physc* (delta PURR/delta TB). This column shows the relative SMT split between processors, i.e., shows the measurement of fraction of time a logical processor was getting physical processor cycles. When running in shared mode, **sar** displays the percentage of entitlement consumed *%entc* which is $((PPFC/ENT)*100)$. This gives relative entitlement consumption for each logical processor and allows system average utilization calculation from logical processor utilization.

Example 8-16 Logical processor usage

```
# sar -P ALL 2 2
```

```
AIX LPARmicro 3 5 00CDEDC4C00 10/20/04
```

```
System configuration: lcpu=6 ent=0.30
```

| | cpu | %usr | %sys | %wio | %idle | physc | %entc |
|----------|-----|------|------|------|-------|-------|-------|
| 11:30:25 | 0 | 60 | 38 | 0 | 2 | 0.09 | 30.6 |
| | 1 | 0 | 2 | 0 | 98 | 0.02 | 7.2 |
| | 2 | 75 | 24 | 0 | 1 | 0.17 | 55.3 |
| | 3 | 0 | 1 | 0 | 99 | 0.03 | 11.4 |
| | 4 | 78 | 21 | 0 | 1 | 0.15 | 49.9 |
| | 5 | 0 | 2 | 0 | 98 | 0.03 | 9.5 |
| | - | 60 | 22 | 0 | 18 | 0.49 | 163.9 |
| 11:30:29 | 0 | 78 | 22 | 0 | 0 | 0.23 | 77.8 |
| | 1 | 0 | 1 | 0 | 99 | 0.04 | 14.9 |
| | 2 | 74 | 25 | 0 | 1 | 0.19 | 63.6 |
| | 3 | 0 | 1 | 0 | 99 | 0.03 | 9.8 |
| | 4 | 84 | 15 | 0 | 1 | 0.19 | 63.9 |
| | 5 | 0 | 1 | 0 | 99 | 0.03 | 8.9 |
| | - | 68 | 18 | 0 | 14 | 0.72 | 238.8 |
| Average | 0 | 73 | 26 | 0 | 1 | 0.16 | 54.3 |
| | 1 | 0 | 1 | 0 | 99 | 0.03 | 11.0 |
| | 2 | 74 | 25 | 0 | 1 | 0.18 | 59.5 |
| | 3 | 0 | 1 | 0 | 99 | 0.03 | 10.6 |
| | 4 | 82 | 17 | 0 | 1 | 0.17 | 56.9 |
| | 5 | 0 | 1 | 0 | 99 | 0.03 | 9.2 |
| | - | 65 | 19 | 0 | 16 | 0.60 | 201.4 |

Whenever the percentage of entitled capacity consumed is under 100%, a line beginning with *U* is added to represent the unused capacity as shown in Example 8-17 on page 292.

Example 8-17 Unused capacity displayed by sar command

| | | | | | | | |
|--|----------|------|------|----------|-----------|-------------|-------------|
| # sar -P ALL 2 1 | | | | | | | |
| AIX LPARmicro 3 5 00CDDDC4C00 10/20/04 | | | | | | | |
| System configuration: lcpu=6 ent=0.30 | | | | | | | |
| 11:31:22 | cpu | %usr | %sys | %wio | %idle | physc | %entc |
| 11:31:24 | 0 | 21 | 66 | 0 | 13 | 0.00 | 0.5 |
| | 1 | 0 | 11 | 0 | 89 | 0.00 | 0.1 |
| | 2 | 0 | 37 | 0 | 63 | 0.00 | 0.4 |
| | 3 | 0 | 5 | 0 | 95 | 0.00 | 0.2 |
| | 4 | 9 | 48 | 0 | 43 | 0.00 | 0.3 |
| | 5 | 0 | 4 | 0 | 96 | 0.00 | 0.2 |
| | U | - | - | 0 | 98 | 0.29 | 98.3 |
| | - | 0 | 1 | 0 | 99 | 0.01 | 1.7 |

8.1.6 topas command

The **topas** command reports selected statistics about the activity on the local system. The command displays its output in a format suitable for viewing on an 80x25 character-based display.

Table 8-3 on page 281 gives a summary for **topas** command.

Table 8-7 topas command summary

| Command name | topas |
|--|------------------|
| Interface type | CLI |
| Updated or new command | update |
| AIX package | bos.perf.tools |
| Measurement Processor Memory Disk I/O Network Hypervisor | + + + + |
| Environment Dedicated Partition Micro-Partition SMT I/O server | + + + + |

Usage

topas [-dhimnpwcPLUDW]

Most important flags

- i** Sets the monitoring interval in seconds. The default is 2 seconds.
- L** Displays the logical partition display. This display reports similar data to what is provided to **mpstat** and **lparstat**.

Output examples

The **topas** output as shown in Example 8-18 on page 293 has been modified. In addition to changes on the main screen, a new one dedicated to virtual processors has been added. The new metrics have been applied so processor utilization is calculated using new PURR-based register and formula when running in SMT or Micro-Partitioning mode. When running in Micro-Partitioning mode **topas** adds automatically the two new information:

Physc The fractional number of processors consumed.

%Entc The percentage of entitled capacity consumed.

Example 8-18 topas output

| | | | | | | | | | | | |
|--------------------------------------|---------|---------|--------|---------|--------------|---------------|----------|-----------|-----------------|--------|--------------|
| Topas Monitor for host: LPARmicro | | | | | | EVENTS/QUEUES | | FILE/TTY | | | |
| Wed Oct 20 14:20:07 2004 Interval: 2 | | | | | | Cswitch | 4080 | Readch | 46.9M | | |
| | | | | | | Syscall | 15352 | Writech | 38.2M | | |
| Kernel | 28.4 | ##### | | | | Reads | 3386 | Rawin | 0 | | |
| User | 56.1 | ##### | | | | Writes | 2383 | Ttyout | 302 | | |
| Wait | 0.4 | # | | | | Forks | 68 | Igets | 0 | | |
| Idle | 15.1 | ##### | | | | Execs | 56 | Namei | 1145 | | |
| Physc = 0.77 | | | | | | %Entc= 257.2 | Runqueue | 1.0 | Dirblk | 0 | |
| | | | | | | Waitqueue | 0.0 | | | | |
| Network | KBPS | I-Pack | O-Pack | KB-In | KB-Out | | | | | | |
| en0 | 39830.3 | 54316.0 | 3313.0 | 78470.1 | 194.7 | PAGING | MEMORY | | | | |
| lo0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | Faults | 15912 | Real,MB | 511 | | |
| | | | | | | Steals | 0 | % Comp | 51.1 | | |
| Disk | Busy% | KBPS | TPS | KB-Read | KB-Writ | PgspIn | 0 | % Noncomp | 10.4 | | |
| hdisk0 | 46.5 | 6583.3 | 144.3 | 12630.0 | 372.0 | PgspOut | 0 | % Client | 12.0 | | |
| | | | | | | PageIn | 9 | | | | |
| Name | PID | CPU% | PgSp | Owner | PAGING SPACE | | | | | | |
| ftpd | 200756 | 8.3 | 0.9 | root | PageOut | 0 | Sios | 9 | Size,MB | 512 | |
| ksh | 323618 | 0.0 | 0.6 | root | | | | | % Used | 0.9 | |
| topas | 290946 | 0.0 | 1.1 | root | | | | | NFS (calls/sec) | % Free | 99.0 |
| gil | 69666 | 0.0 | 0.1 | root | | | | | ServerV2 | 0 | |
| getty | 270468 | 0.0 | 0.4 | root | | | | | ClientV2 | 0 | Press: |
| rpc.lockd | 237690 | 0.0 | 0.2 | root | | | | | ServerV3 | 0 | "h" for help |
| syncd | 94344 | 0.0 | 0.5 | root | | | | | ClientV3 | 0 | "q" to quit |

| | | | | |
|-----------|--------|-----|-----|------|
| netm | 65568 | 0.0 | 0.0 | root |
| IBM.CSMAg | 274568 | 0.0 | 2.1 | root |
| rmcd | 262274 | 0.0 | 1.4 | root |
| wlmsched | 73764 | 0.0 | 0.1 | root |
| sendmail | 82016 | 0.0 | 0.9 | root |

The new LPAR screen as shown in is accessible from -L command line flag or typing L while topas is running. It splits the screen in an upper section, showing a subset of **lparstat** metrics, and a lower section that shows a sorted list of logical processors with **mpstat** columns. The *%hypv* and *hcalls* give the percentage of time spent in Hypervisor and the number of calls made. The *pc* value is the fraction of physical processor consumed by a logical processor. When in Micro-Partitioning there are additional metrics:

| | |
|----------------------|---|
| Psize | Number of online physical processors in the pool. |
| physc | Number of physical processor(s) consumed. |
| %entc | Percentage of entitlement consumed. |
| %lbusy | Logical processor utilization. |
| app | Available pool processors, the number of physical processor available in the shared pool. |
| lcsw and vcsw | Logical and virtual context switches per second over the monitoring interval. |
| phint | Number of phantom interrupts. |
| %hypv | Shows the percentage of time spent in Hypervisor. |

Example 8-19 Logical processors view using topas

| | | | | | | | | | | | | | | |
|---|-------|----------------|-------|-------|--------|--------|------|--------|-------|------------------------|--------|-----|------|------|
| Interval: 2 Logical Partition: MicroPartitionAIAvWed Oct 20 14:23:51 2004 | | | | | | | | | | | | | | |
| Psize: 1 | | Shared SMT ON | | | | | | | | Online Memory: 512.0 | | | | |
| Ent: 0.30 | | Mode: UnCapped | | | | | | | | Online Logical CPUs: 6 | | | | |
| Partition CPU Utilization | | | | | | | | | | Online Virtual CPUs: 3 | | | | |
| %usr | %sys | %wait | %idle | physc | %entc | %lbusy | app | vcsw | phint | %hypv | hcalls | | | |
| 59 | 27 | 0 | 13 | 1.0 | 324.30 | 47.54 | 0.01 | 11807 | 9 | 0.0 | 0 | | | |
| ===== | | | | | | | | | | | | | | |
| LCPU | minpf | majpf | intr | csw | icsw | runq | lpa | scalls | usr | sys | wt | idl | pc | lcsw |
| Cpu0 | 0 | 0 | 824 | 224 | 220 | 1 | 100 | 25698 | 73 | 25 | 0 | 2 | 0.23 | 2043 |
| Cpu1 | 0 | 0 | 785 | 2265 | 1137 | 1 | 100 | 3270 | 54 | 21 | 0 | 25 | 0.17 | 2089 |
| Cpu2 | 0 | 0 | 985 | 509 | 337 | 0 | 100 | 4392 | 56 | 30 | 0 | 14 | 0.16 | 2099 |
| Cpu3 | 0 | 0 | 728 | 943 | 568 | 1 | 100 | 3325 | 56 | 33 | 0 | 11 | 0.18 | 2064 |
| Cpu4 | 0 | 0 | 777 | 230 | 223 | 0 | 100 | 10586 | 69 | 29 | 0 | 2 | 0.19 | 1725 |
| Cpu5 | 0 | 0 | 635 | 41 | 26 | 1 | 100 | 43 | 0 | 29 | 0 | 70 | 0.05 | 1787 |

8.1.7 xmperf command

Table 8-3 on page 281 gives a summary for **xmperf** command.

Table 8-8 *xmperf command summary*

| Command name | xmperf |
|--|---|
| Interface type | GUI |
| Updated or new command | update |
| AIX package | perfmgr.network which is part of the Performance Tool Box |
| Measurement Processor Memory Disk I/O Network Hypervisor | + + + + |
| Environment Dedicated Partition Micro-Partition SMT I/O server | + + + |

Usage

xmperf [-vauxz] [-h hostname]

Most important flags

- h Localhostname (host to be "local" host).
- o Configuration file, default is /\$HOME/xmperf.cf.

Output examples

Figure 8-2 on page 296 shows the standard output of **xmperf** command called Local System Monitor. This console is composed by eights monitors to display information on CPU usage, disk access, network traffic, paging space occupancy, memory and process activity.

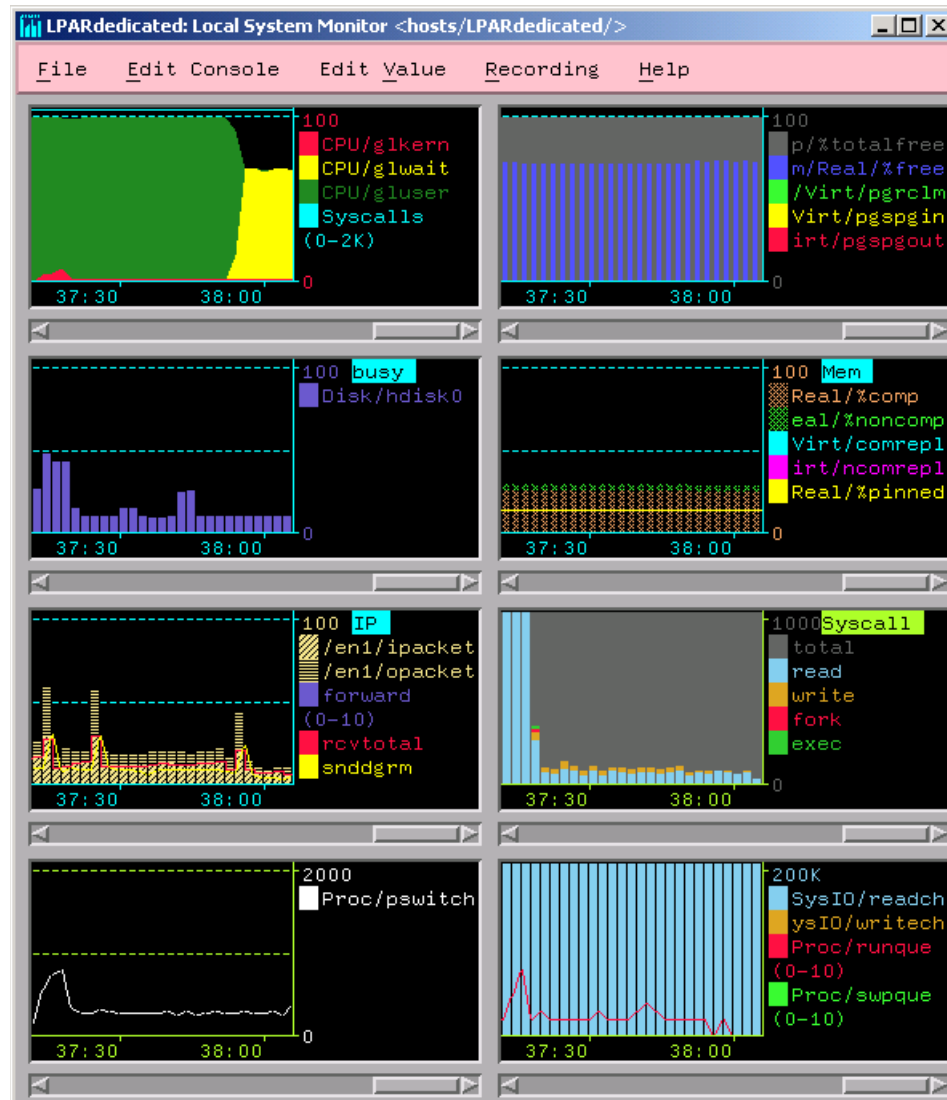


Figure 8-2 xmperv - Local System Monitor

System activity and processor consumption

In the following example, **xmperv** is used to show system activity and processor usage. One logical uncapped micro-partition is created with 3 logical processors, the system has a pool size of two processors.

Several CPU consuming tasks are started in the partition, the system activity and the CPU usage is displayed in Figure 8-3 on page 298. The first three diagrams

show activity on each processor *cpu0*, *cpu1* and *cpu2*. The last part of the figure shows the whole system activity *user* and the number of physical processor consumed *phpsc*.

Three tasks are launched one after the other and then a partition is started:

1. The first task starts at 00:30 on CPU1, at this moment the system activity increases up to 100% and because only one mono thread task is running, only one physical processor is consumed.
2. The second task starts at 01:00 on CPU0, the system activity remains a 100% (it is obvious that the system cannot consume more) and almost a second physical processor is consumed.
3. The third task starts at 01:30 on CPU2, the system activity remains a 100% and physical processor consumed stays around two, because only two processors were available in the shared pool.
4. At 02:00, a logical partition with one dedicated processor is started in the same system, this removes processing units equivalent to one processor from the micro-partition. At this point, three tasks are running in the partition, each ones on a distinct logical processor and only one physical processor is consumed. This means that each tasks is consuming one third of physical processor.

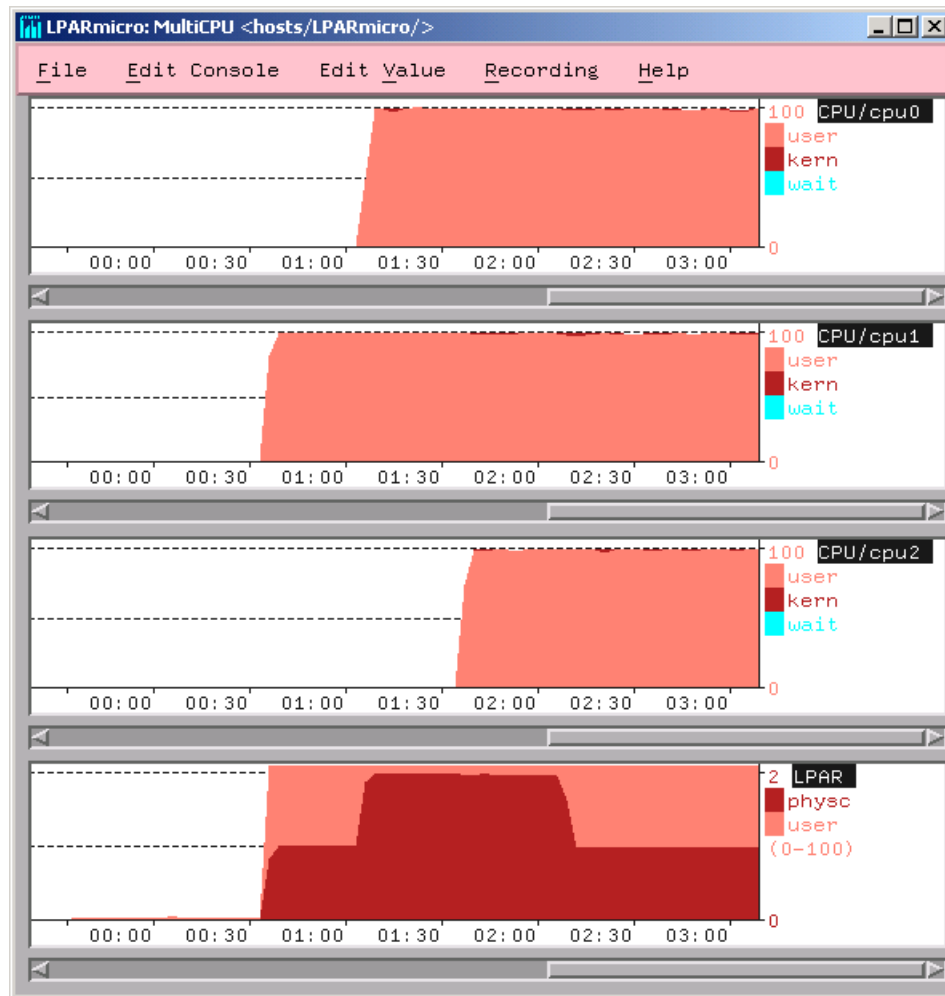


Figure 8-3 Physical and logical processors usage

Redispatch activity

When a process is running on a server, by default it is not bound to a processor, that means it can run on any available processor. Furthermore the Hypervisor dispatches virtual processors on physical processor, so a process may not stay all its life on the same processor.

In order to illustrate this, the following case use an uncapped Micro-Partition with three virtual processors and one processor available in he shared pool.

A CPU intensive job is started around 06:32 as shown in Figure 8-4 on page 299. It is dispatched on *cpu0*, the partition activity increases to 100% (*user*) and it consumes one physical processor (*physc*). The job is then redispached on *cpu1*, *cpu2*, *cpu0*, *cpu1* and *cpu0*. During all this run, the system is always 100% busy and only one physical processor is used although a processor is available in the pool, and the job is not running on the same processor all the time.

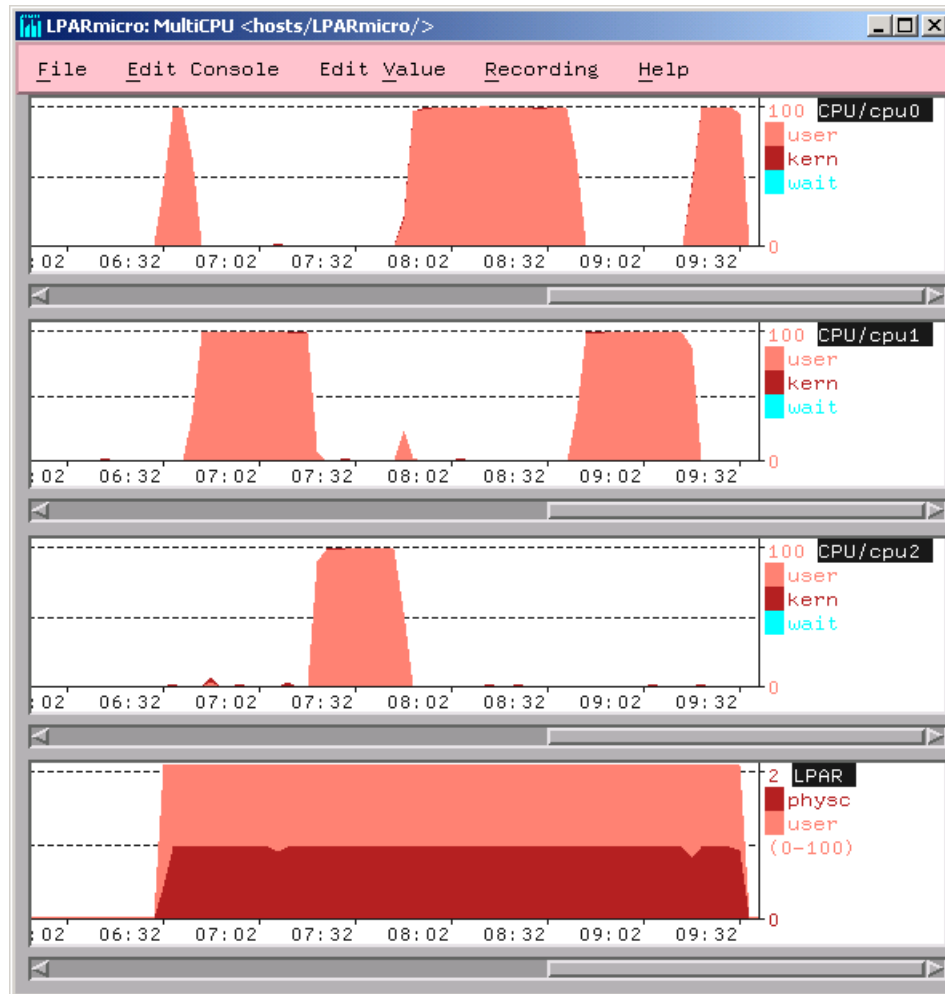


Figure 8-4 xmp perf - process redispach

Author Comment: add XXXref to redispach affinity explanation

8.2 Performance tuning approach

A system may experience performance problems for many reasons, hardware problems, software problems or human expectations.

In this section we mainly focus on hardware problems linked to the new POWER5 architecture.

The performance analysis and tuning demands great skills, knowledge, experience and methodology. To determine which of the monitored values are high in a particular environment, it is a good idea to gather the data on the system during an optimal performance state. This baseline information is very useful to compare with data when having a performance problem. The **xmperf** command can be used to collect data, screen shots of **topas** command also provide brief overview of all the major performance information.

8.2.1 Global performance analysis

In order to solve performance problems, the investigation to find the root cause will be done for the following categories:

- ▶ CPU bound system on page 306.
- ▶ Memory bound system on page 311.
- ▶ DISK I/O bound system on page 313.
- ▶ Network I/O bound system on page 322.

The Figure 8-5 on page 301 gives the chronological order to follow when trying to identify a performance issues, first check the CPU, then the memory, the disk and finally the network.

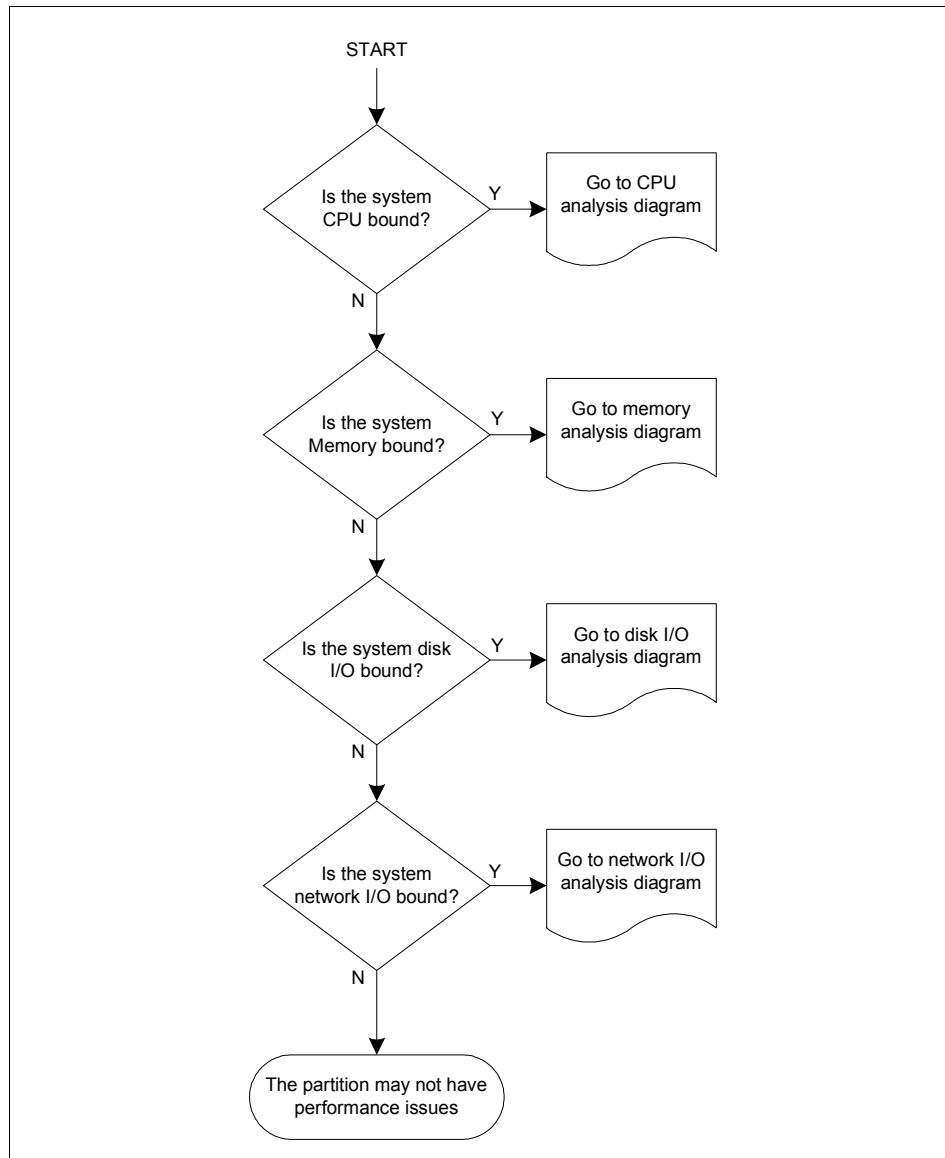


Figure 8-5 Global performance diagram

CPU bound system

A system that is CPU bound means obviously all the processors are 100% busy, but also that some jobs are waiting for CPU in the run queue. A system with 100% busy CPU, a large run queue compare to the number of CPU and a

greater amount of context switches than usual has a lot of chance to be CPU bound.

In Example 8-20 on page 302 the system has two dedicated processors which are 99.8% busy, the run queue is four (twice the number of processors) and 5984 context switches (this system for its everyday work usually has around 500). This system is CPU bound.

Example 8-20 CPU bound system

| | | | | | | | | | | |
|---------------------------------------|--------|--------|--------|---------|----------|---------------|--------------|-----------|-----------------|-------------|
| Topas Monitor for host: LPARdedicated | | | | | | EVENTS/QUEUES | | FILE/TTY | | |
| Wed Oct 27 15:24:31 2004 Interval: 2 | | | | | | Cswitch | 5984 | Readch | 808 | |
| | | | | | | Syscall | 12132 | Writech | 8078.3K | |
| Kernel | 0.2 | # | | | | Reads | 1 | Rawin | 0 | |
| User | 99.8 | ##### | | | | Writes | 2035 | Ttyout | 309 | |
| Wait | 0.0 | | | | | Forks | 0 | Igets | 0 | |
| Idle | 0.0 | | | | | Execs | 0 | Namei | 5 | |
| | | | | | | Runqueue | 4.0 | Dirblk | 0 | |
| Network | KBPS | I-Pack | O-Pack | KB-In | KB-Out | Waitqueue | 0.0 | | | |
| en1 | 0.9 | 13.0 | 11.0 | 0.6 | 1.2 | | | | | |
| lo0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | | |
| | | | | | | PAGING | MEMORY | | | |
| | | | | | | Faults | 0 | Real,MB | 1023 | |
| | | | | | | Steals | 0 | % Comp | 26.0 | |
| Disk | Busy% | KBPS | TPS | KB-Read | KB-Writ | PgspIn | 0 | % Noncomp | 4.0 | |
| hdisk0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | PgspOut | 0 | % Client | 4.8 | |
| | | | | | | PageIn | 0 | | | |
| Name | PID | CPU% | PgSp | Owner | | | | | | |
| vpross | 278618 | 25.0 | 15.4 | root | PageOut | 0 | PAGING SPACE | | | |
| yes | 291018 | 25.0 | 0.1 | root | Sios | 0 | Size,MB | 512 | | |
| ksh | 286946 | 25.0 | 0.5 | root | | | | | % Used | 0.8 |
| ksh | 241918 | 25.0 | 0.5 | root | | | | | NFS (calls/sec) | % Free 99.1 |
| topas | 266278 | 0.0 | 1.1 | root | ServerV2 | 0 | | | | |
| snmpmibd6 | 155772 | 0.0 | 0.7 | root | ClientV2 | 0 | Press: | | | |
| xmgc | 45078 | 0.0 | 0.0 | root | ServerV3 | 0 | "h" for help | | | |
| netm | 49176 | 0.0 | 0.0 | root | ClientV3 | 0 | "q" to quit | | | |
| IBM.CSMAg | 262280 | 0.0 | 2.2 | root | | | | | | |
| getty | 258174 | 0.0 | 0.4 | root | | | | | | |
| gil | 53274 | 0.0 | 0.1 | root | | | | | | |
| aixmibd | 139414 | 0.0 | 0.6 | root | | | | | | |
| syncd | 65710 | 0.0 | 0.5 | root | | | | | | |
| rpc.lockd | 209122 | 0.0 | 0.2 | root | | | | | | |
| nfsd | 196776 | 0.0 | 0.2 | root | | | | | | |
| lvmbb | 86076 | 0.0 | 0.0 | root | | | | | | |
| dog | 90184 | 0.0 | 0.1 | root | | | | | | |

Memory bound system

System memory includes real memory and paging space. The AIX operating system uses the Virtual Memory Manager (VMM) to control real memory and

paging space on the system. The VMM maintains a list of free memory pages, a page replacement algorithm is used to determine which pages

A system is memory bound if it has high memory occupancy and high paging space activity. The activity of the paging space is given by the number of pages read from disk to memory (page ins) and number of pages written to disk (page out).

The amount of used memory and paging space activity can be obtained by **topas** command. In Example 8-21 on page 303 the memory is 100% consumed (*Comp*, *Noncomp*), paging space is 61% consumed (*% Used*), a lot of pages are written to disk (*PgspOut*) and the system needs real memory, that is why VMM steals pages (*Steals*). Because the system is using all the memory and asking for more, this partition is memory bound.

Example 8-21 Memory bound system.

| Topas Monitor for host: LPARdedicated | | | | | | EVENTS/QUEUES | | FILE/TTY | |
|---------------------------------------|--------|---------|--------|---------|---------|-----------------|--------|--------------|-------|
| Wed Oct 27 18:19:37 2004 Interval: 2 | | | | | | Cswitch | 998 | Readch | 12122 |
| | | | | | | Syscall | 406 | Writech | 290 |
| Kernel | 4.1 | ## | | | | Reads | 17 | Rawin | 0 |
| User | 64.1 | ##### | | | | Writes | 0 | Ttyout | 290 |
| Wait | 4.0 | ## | | | | Forks | 0 | Igets | 0 |
| Idle | 27.8 | ##### | | | | Execs | 0 | Namei | 12 |
| | | | | | | Runqueue | 1.0 | Dirblk | 0 |
| Network | KBPS | I-Pack | O-Pack | KB-In | KB-Out | Waitqueue | 1.0 | | |
| en1 | 0.4 | 3.0 | 1.0 | 0.1 | 0.6 | | | | |
| lo0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | |
| | | | | | | PAGING | MEMORY | | |
| | | | | | | Faults | 5039 | Real,MB | 1023 |
| Disk | Busy% | KBPS | TPS | KB-Read | KB-Writ | Steals | 4963 | % Comp | 100.3 |
| hdisk0 | 19.8 | 19863.0 | 419.9 | 444.0 | 39580.0 | PgspIn | 32 | % Noncomp | 0.5 |
| | | | | | | PgspOut | 4946 | % Client | 0.6 |
| Name | PID | CPU% | PgSp | Owner | | | | | |
| perl | 307230 | 25.0 | 1102.5 | root | | | | | |
| lrud | 20490 | 0.0 | 0.1 | root | | | | | |
| telnetd | 245904 | 0.0 | 0.2 | root | | | | | |
| topas | 266340 | 0.0 | 1.1 | root | | | | | |
| rgsr | 69756 | 0.0 | 0.0 | root | | | | | |
| rmcd | 229496 | 0.0 | 1.4 | root | | | | | |
| init | 1 | 0.0 | 0.6 | root | | | | | |
| netm | 49176 | 0.0 | 0.0 | root | | | | | |
| IBM.CSMAg | 262280 | 0.0 | 2.2 | root | | | | | |
| getty | 258174 | 0.0 | 0.4 | root | | | | | |
| gil | 53274 | 0.0 | 0.1 | root | | | | | |
| aixmibd | 139414 | 0.0 | 0.6 | root | | | | | |
| | | | | | | PageIn | 56 | | |
| | | | | | | PageOut | 4946 | PAGING SPACE | |
| | | | | | | Sios | 5016 | Size,MB | 512 |
| | | | | | | % Used | | | |
| | | | | | | | | | |
| | | | | | | NFS (calls/sec) | % Free | 38.4 | |
| | | | | | | ServerV2 | 0 | | |
| | | | | | | ClientV2 | 0 | Press: | |
| | | | | | | ServerV3 | 0 | "h" for help | |
| | | | | | | ClientV3 | 0 | "q" to quit | |

Disk I/O bound system

A system that is disk I/O bound means that at least one disk is busy and cannot fulfill other requests, process are blocked and are waiting for the I/O operation to complete. The limitation can be either physical or logical. The physical limitation involves hardware like bandwidth of disks, adapters and system bus. The logical limitations involves the organization of the logical volumes on disks and Logical Volume Manager (LVM) tunings and settings, such as striping or mirroring.

Example 8-22 on page 304 shows a system with a high wait I/O at 86.6% (*Wait*), a percentage of time that hdisk0 was active at 98.7% (*Busy%*) and more than 5 processes which are waiting for paging space operation to complete (*Waitqueue*). This system is waiting for write operation on hdisk0, it is disk I/O bound.

Example 8-22 Disk I/O bound system

| | | | | | | | | | |
|---------------------------------------|-------|--------|--------|---------|---------|-----------------|--------|--------------|------|
| Topas Monitor for host: LPARdedicated | | | | | | EVENTS/QUEUES | | FILE/TTY | |
| Thu Oct 28 11:03:43 2004 Interval: 2 | | | | | | Cswitch | 678 | Readch | 0 |
| | | | | | | Syscall | 97 | Writech | 317 |
| Kernel | 1.1 | # | | | | Reads | 0 | Rawin | 0 |
| User | 12.4 | #### | | | | Writes | 0 | Ttyout | 317 |
| Wait | 86.6 | ##### | | | | Forks | 0 | Igets | 0 |
| Idle | 0.0 | | | | | Execs | 0 | Namei | 0 |
| | | | | | | Runqueue | 2.5 | Dirblk | 0 |
| Network | KBPS | I-Pack | O-Pack | KB-In | KB-Out | Waitqueue | 5.6 | | |
| en1 | 0.5 | 3.0 | 1.0 | 0.3 | 0.7 | | | | |
| lo0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | | | | |
| | | | | | | PAGING | | | |
| | | | | | | Faults | 826 | Real,MB | 1023 |
| Disk | Busy% | KBPS | TPS | KB-Read | KB-Writ | Steals | 0 | % Comp | 99.7 |
| hdisk0 | 98.7 | 2645.1 | 185.3 | 4.0 | 5220.0 | PgspIn | 0 | % Noncomp | 1.2 |
| | | | | | | PgspOut | 652 | % Client | 0.5 |
| | | | | | | PageIn | 0 | | |
| | | | | | | PageOut | 652 | PAGING SPACE | |
| | | | | | | Sios | 588 | Size,MB | 512 |
| | | | | | | | | % Used | 47.0 |
| | | | | | | NFS (calls/sec) | % Free | | 52.9 |
| | | | | | | ServerV2 | 0 | | |
| | | | | | | ClientV2 | 0 | Press: | |
| | | | | | | ServerV3 | 0 | "h" for help | |
| | | | | | | ClientV3 | 0 | "q" to quit | |
| Name | | PID | CPU% | PgSp | Owner | | | | |
| perl | | 291012 | 0.0 | 983.2 | root | | | | |
| topas | | 266340 | 0.0 | 1.1 | root | | | | |
| getty | | 258174 | 0.0 | 0.4 | root | | | | |
| lrud | | 20490 | 0.0 | 0.1 | root | | | | |
| netm | | 49176 | 0.0 | 0.0 | root | | | | |
| vpross | | 303230 | 0.0 | 15.4 | root | | | | |
| IBM.CSMAg | | 262280 | 0.0 | 2.2 | root | | | | |
| gil | | 53274 | 0.0 | 0.1 | root | | | | |
| syncd | | 65710 | 0.0 | 0.5 | root | | | | |
| aixmibd | | 139414 | 0.0 | 0.6 | root | | | | |
| rpc.lockd | | 209122 | 0.0 | 0.2 | root | | | | |
| nfsd | | 196776 | 0.0 | 0.2 | root | | | | |
| rgsr | | 69756 | 0.0 | 0.0 | root | | | | |
| errdemon | | 73858 | 0.0 | 0.5 | root | | | | |
| j2pg | | 77868 | 0.0 | 0.2 | root | | | | |
| lvmbb | | 86076 | 0.0 | 0.0 | root | | | | |
| dog | | 90184 | 0.0 | 0.1 | root | | | | |

```
hostmibd      94238    0.0    0.4 root
```

Network I/O bound system

A system that is network I/O bound means that the bandwidth of at least one network adapter is totally (or almost totally) used. Processes which needed to send or receive data must wait for other processes I/O to complete.

Example 8-23 on page 305, shows a system using all the bandwidth of its network adapter nd having some wait I/O. The maximum bandwidth of the network adapter depends on its type.

Example 8-23 Network I/O bound system

| Topas Monitor for host: LPARdedicated | | | | | | EVENTS/QUEUES | | FILE/TTY | |
|---------------------------------------|---------|---------|---------|---------|-----------------|---------------|------|-----------|-------|
| Fri Oct 29 14:36:34 2004 Interval: 2 | | | | | | Cswitch | 1636 | Readch | 11.6M |
| | | | | | | Syscall | 422 | Writech | 11.6M |
| Kernel | 11.0 | #### | | | | Reads | 185 | Rawin | 0 |
| User | 0.1 | # | | | | Writes | 186 | Ttyout | 267 |
| Wait | 13.8 | ##### | | | | Forks | 0 | Igets | 0 |
| Idle | 75.1 | ##### | | | | Execs | 0 | Namei | 0 |
| | | | | | | Runqueue | 1.0 | Dirblk | 0 |
| Network | KBPS | I-Pack | O-Pack | KB-In | KB-Out | Waitqueue | 0.0 | | |
| en1 | 12285.3 | 10513.0 | 16175.0 | 472.3 | 23914.1 | | | | |
| lo0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | PAGING | | | |
| | | | | | | Faults | 0 | Real,MB | 1023 |
| Disk | Busy% | KBPS | TPS | KB-Read | KB-Writ | Steals | 0 | % Comp | 23.5 |
| hdisk0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | PgspIn | 0 | % Noncomp | 33.8 |
| | | | | | | PgspOut | 0 | % Client | 34.4 |
| | | | | | | PageIn | 0 | | |
| Name | PID | CPU% | PgSp | Owner | PageOut | | | | |
| ftpd | 266380 | 0.0 | 0.8 | root | 0 PAGING SPACE | | | | |
| topas | 278692 | 0.0 | 1.1 | root | Sios | | | | |
| getty | 245882 | 0.0 | 0.4 | root | 0 Size,MB | | | | |
| aixmibd | 204912 | 0.0 | 0.6 | root | % Used | | | | |
| rpc.lockd | 159988 | 0.0 | 0.2 | root | NFS (calls/sec) | | | | |
| nfsd | 188590 | 0.0 | 0.2 | root | ServerV2 | | | | |
| xmhc | 45078 | 0.0 | 0.0 | root | ClientV2 | | | | |
| netm | 49176 | 0.0 | 0.0 | root | ServerV3 | | | | |
| IBM.CSMAg | 241790 | 0.0 | 2.0 | root | ClientV3 | | | | |
| gil | 53274 | 0.0 | 0.1 | root | 0 Press: | | | | |
| ftpd | 250008 | 0.0 | 0.8 | root | 0 "h" for help | | | | |
| syncd | 65710 | 0.0 | 0.5 | root | 0 "q" to quit | | | | |
| rgsr | 69758 | 0.0 | 0.0 | root | | | | | |
| errdemon | 73858 | 0.0 | 0.5 | root | | | | | |
| j2pg | 77866 | 0.0 | 0.2 | root | | | | | |

8.2.2 CPU analysis

Now that we know how to recognize a CPU bound system, Figure 8-6 on page 306 help to determine the root cause for this activity.

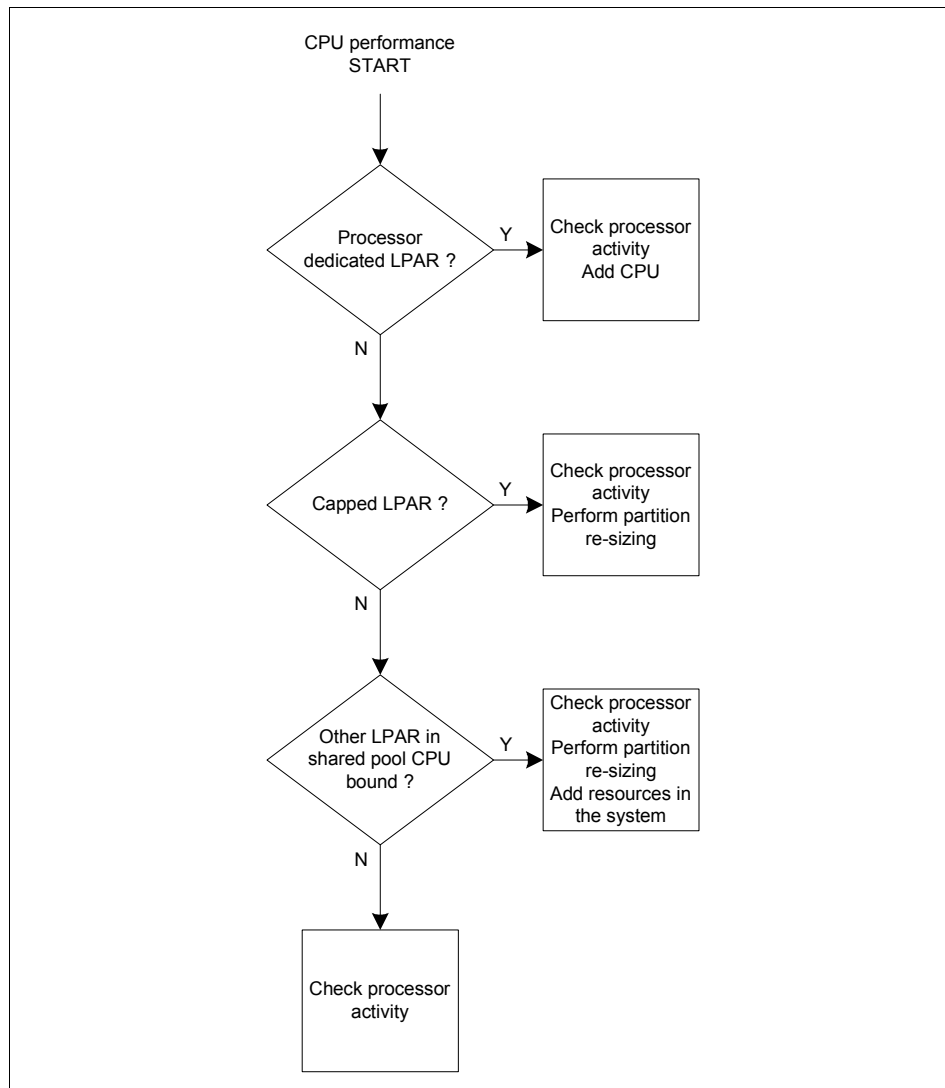


Figure 8-6 CPU analysis diagram.

Identify CPU consumers

In order to understand why the partition is CPU bound we have to find which processes are the most CPU consumers using **ps** command as shown in

Example 8-24 on page 307. The %CPU column gives the percentage of time the process has used the CPU since the process started. In this example three processes are using the CPU: ./vpross, yes and ./loop. You then must verify if those processes are running correctly and if they are using the same amount of CPU than usual.

Example 8-24 ps - most CPU consumers

```
# ps aux|more
USER          PID %CPU %MEM    SZ   RSS    TTY STAT      STIME   TIME  COMMAND
root        295058 49.7   5.0 15952 15980 pts/1 A       08:34:25 66:04 ./vpross
root        139278 20.0   0.0   152   156 pts/0 A       09:40:46  0:02 yes
root        262232 14.3   0.0   676   712 pts/0 A       10:12:03  0:04 ./loop
root        241790  0.2   1.0  2856  2548    - A       11:26:34  4:08 /usr/sbin/rsct/b
root        258174  0.0   0.0   464   488    - A       11:26:29  0:16 /usr/sbin/getty
root         53274  0.0   0.0   116   116    - A       11:24:25  0:12 gil
root        155888  0.0   0.0   960   960    - A       11:26:15  0:04 /usr/sbin/aixmib
root         65710  0.0   0.0   500   508    - A       11:25:41  0:04 /usr/sbin/syncd
root        172270  0.0   0.0   200   200    - A       11:26:22  0:02 rpc.lockd
root        209028  0.0   0.0   200   200    - A       11:26:21  0:02 nfsd
root        196726  0.0   0.0   444   464    - A       11:26:03  0:00 /usr/sbin/inetd
root        200844  0.0   0.0  1692  1296    - A       11:25:57  0:00 sendmail: accept
...
```

Dedicated LPAR

If applications really needs processing power, then you may add dynamically a processor (if any available in the system) to the partition. In order to know the available processors in the system, connect to the HMC, edit the system properties and select **Processor** tab. For example in Figure 8-7 on page 308 only 0.3 processors are available, a list of partitions with their amount of processors used is given in the bottom of the figure.

If no processor are available, check the CPU usage of all other partitions in the system to see if you can free an unused processor. Then if all the processor resources are used, upgrade the system with new processors.

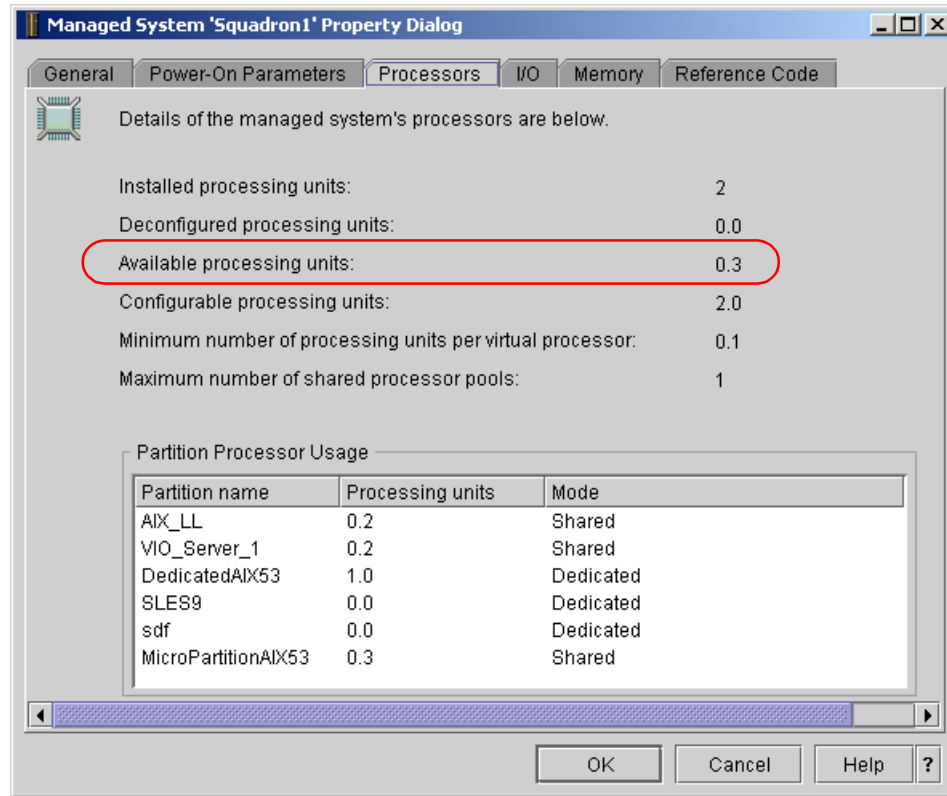


Figure 8-7 System properties - Processors

Capped LPAR

If applications really needs processing power and if there is any available processing unit in the system, you can increase the entitlement of the partition. Example 8-25 on page 308 shows a capped Micro-Partition with one logical processor, an entitlement of 0.5 (*ent*) and 1.5 available processor pool size (*app*). Only one process (*testp*) is requesting processing power and it consumes half a processor (*pc*). In this case we can increase the entitlement up to 1.0 and the processes will run faster.

Example 8-25 Entitlement limited partition

```
# lparstat 2 2
```

```
System configuration: type=Shared mode=Capped smt=Off lcpu=1 mem=512 psize=2
ent=0.50
```

```
%user %sys %wait %idle physc %entc lbusy app vcs w pinct
```

```
-----
99.3  0.7  0.0  0.0  0.50 100.0 100.0 1.49 100  0
98.6  0.9  0.0  0.5  0.50 99.5 100.0 1.49 100  0

# vmstat 2 3

System configuration: lcpu=1 mem=512MB ent=0.5

kthr    memory                page                faults                cpu
-----
r  b   avm   fre  re  pi  po  fr   sr  cy  in   sy  cs  us  sy  id  wa    pc    ec
2  0 49165 72012   0   0   0   0   0   0   2 3079 142 99   1   0   0  0.50 100.2
2  0 49167 72010   0   0   0   0   0   0   3 3031 142 99   1   0   0  0.50  99.8
2  0 49167 72010   0   0   0   0   0   0   3 3030 142 99   1   0   0  0.50 100.0

# ps aux|more
USER          PID %CPU %MEM  SZ  RSS  TTY STAT   STIME  TIME  COMMAND
root          254108 43.8  0.0  152  156 pts/0 A    14:36:12  1:18  testp
root          172124  0.0  0.0  940  940      - A    14:29:38  0:00  /usr/sbin/aixmib
root          168030  0.0  0.0  532  504      - A    14:29:32  0:00  /usr/sbin/hostmi
root          163920  0.0  1.0 1156 1192      - A    14:29:29  0:00  /usr/sbin/snmpd
root          184422  0.0  0.0  676  700      - A    14:34:17  0:00  telnetd -a
root          180312  0.0  0.0  664  688      - A    14:29:41  0:00  /usr/sbin/muxatm
root          176218  0.0  0.0 1048  916      - A    14:29:35  0:00  /usr/sbin/snmpmi
...
```

If many processes request power processing you can increase the number of virtual processors (and of course the entitlement). Example 8-26 on page 309 shows a capped Micro-Partition with one logical processor, an entitlement of 1.0 (*ent*) and 0.99 available processor pool size (*app*). Two processes are requesting processing power: *testp* and *loop*. They consume one physical processor (*physc*). Because two processes are running on one physical processor, we can increase the number of virtual processor in order to run each process on a distinct processor.

Example 8-26 Logical processor limited partition

```
# lparstat 2 2

System configuration: type=Shared mode=Capped smt=Off lcpu=1 mem=512 psize=2
ent=1.00

%user  %sys  %wait  %idle  physc  %entc  lbusy  app  vcsw  phint
-----
99.4   0.6   0.0    0.0    1.00 100.1  100.0  0.99   0    1
99.5   0.5   0.0    0.0    1.00 100.0  100.0  0.99   0    0
```

```
# ps aux|more
USER          PID %CPU %MEM    SZ   RSS   TTY STAT      STIME   TIME  COMMAND
root         229502 48.9   0.0   152   156 pts/0 A       16:21:00 0:23  testp
root         241812 37.4   0.0   152   156 pts/0 A       16:00:07 8:06  loop
root         237690  0.2   1.0  1852  1828    - A       15:55:10 0:01  /usr/sbin/rsct/b
root         172126  0.0   0.0  1048   916    - A       15:54:50 0:00  /usr/sbin/snmpmi
root         188572  0.0   0.0   200   200    - A       15:55:01 0:00  rpc.lockd
root         176216  0.0   0.0   940   940    - A       15:54:54 0:00  /usr/sbin/aixmib
root         180312  0.0   0.0   664   688    - A       15:54:57 0:00  /usr/sbin/muxatm
root         184422  0.0   0.0   676   700    - A       15:59:46 0:00  telnetd -a
root         168030  0.0   0.0   532   504    - A       15:54:47 0:00  /usr/sbin/hostmi
root         147588  0.0   0.0   720   756 pts/0 A       15:59:46 0:00  -ksh
...
```

Micro-Partition

Verify if other partitions can give some processing unit or processors back to the shared pool. For example check if idle processor are allowed to be shared

In order to select this option, connect to the HMC, edit the profile properties, select **Processor** tab, then select the **Allow idle processors to be shared** check box like in Figure 8-8 on page 311. If this option is not selected, when the partition will be stopped, the unused processors will not be available for other partitions, but this also means that your partition is guaranteed to have its processor if needed.

Check profile properties for capped and dedicated partitions to be sure that no CPU resources are allocated but not used.

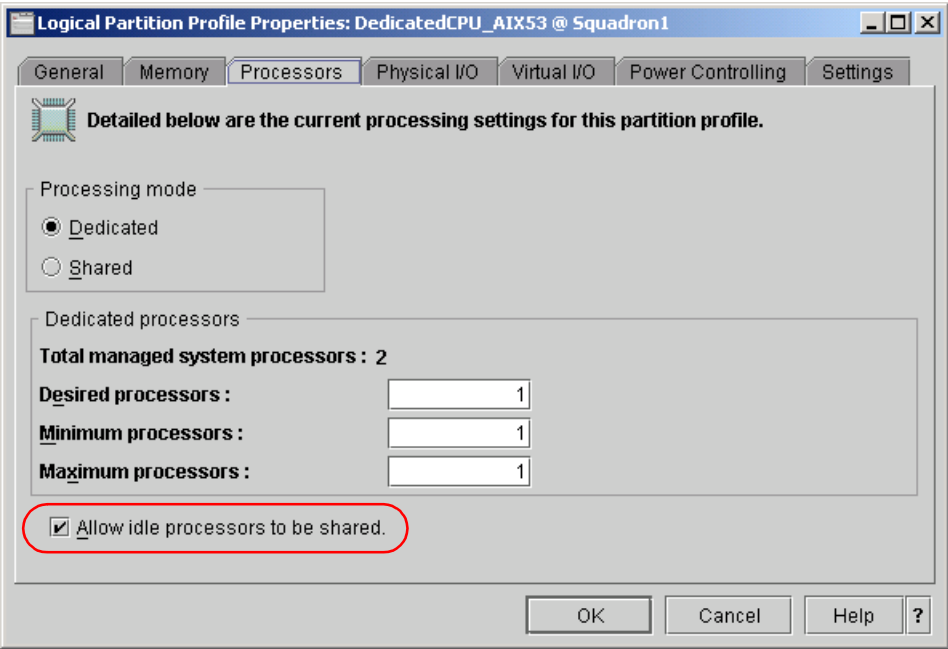


Figure 8-8 Allow idle processors to be shared

In all the previous cases, if many processes need processors, you may turn on SMT mode to get benefit of simultaneous multithreading, for more information on SMT, refer to Chapter 5, “Simultaneous Multi-Threading (SMT)” on page 89.

8.2.3 Memory analysis

The goal is to determine which processes are making the system memory bound. Use **ps** or **svmon** commands to look for process who are consuming a lot of memory. In Example 8-27 on page 311, *perl* is the largest memory consumer application with a total number of pages in real memory of 218933 (around 875 MB) and total number of pages reserved or used on paging space of 97963 (nearly 400 MB). The second application *vpross* use only 48 MB of memory and less than 7 MB of paging space, therefore the *perl* application is the root cause of this memory problem in this example.

Example 8-27 svmon - process report

svmon -P|more

| Pid | Command | Inuse | Pin | Pgsp | Virtual | 64-bit | Mthrd | LPage |
|--------|---------|--------|------|-------|---------|--------|-------|-------|
| 332008 | perl | 218933 | 4293 | 97963 | 318471 | N | N | N |

| Vsid | Esid | Type | Description | LPage | Inuse | Pin | Pgsp | Virtual |
|-------|------|------|---------------------|-------|-------|------|-------|---------|
| 7380 | 6 | work | working storage | - | 65536 | 0 | 0 | 65536 |
| 1383 | 3 | work | working storage | - | 47302 | 0 | 18215 | 65515 |
| 15389 | 7 | work | working storage | - | 44717 | 0 | 0 | 44717 |
| 17388 | 4 | work | working storage | - | 38021 | 0 | 27528 | 65536 |
| 21393 | 5 | work | working storage | - | 15031 | 0 | 50528 | 65536 |
| 0 | 0 | work | kernel segment | - | 6843 | 4290 | 1621 | 8454 |
| 3f8bd | d | work | loader segment | - | 1352 | 0 | 71 | 3062 |
| 29397 | f | work | shared library data | - | 83 | 0 | 0 | 83 |
| d385 | 2 | work | process private | - | 32 | 3 | 0 | 32 |
| 3362 | 1 | clnt | code,/dev/hd2:12435 | - | 16 | 0 | - | - |
| 2f374 | a | work | working storage | - | 0 | 0 | 0 | 0 |
| 3f37c | 9 | work | working storage | - | 0 | 0 | 0 | 0 |
| 3d37d | 8 | work | working storage | - | 0 | 0 | 0 | 0 |

| Pid | Command | Inuse | Pin | Pgsp | Virtual | 64-bit | Mthrd | LPage |
|--------|---------------|-------|------|------|---------|--------|-------|-------|
| 303122 | vpross | 12193 | 4293 | 1696 | 15465 | N | N | N |

| Vsid | Esid | Type | Description | LPage | Inuse | Pin | Pgsp |
|---------|------|------|----------------------|-------|-------|------|-----------|
| Virtual | | | | | | | |
| 0 | 0 | work | kernel segment | - | 6843 | 4290 | 1621 8454 |
| 17368 | 2 | work | process private | - | 3913 | 3 | 4 3917 |
| 3f8bd | d | work | loader segment | - | 1352 | 0 | 71 3062 |
| 1936f | 1 | pers | code,/dev/lv00:83977 | - | 53 | 0 | - - |
| 1136b | f | work | shared library data | - | 32 | 0 | 0 32 |
| 1336a | - | pers | /dev/lv00:83969 | - | 0 | 0 | - - |

| Pid | Command | Inuse | Pin | Pgsp | Virtual | 64-bit | Mthrd | LPage |
|--------|---------------|-------|------|------|---------|--------|-------|-------|
| 299170 | IBM.CSMAgentR | 8572 | 4306 | 1965 | 12130 | N | Y | N |

| Vsid | Esid | Type | Description | LPage | Inuse | Pin | Pgsp |
|---------|------|------|----------------|-------|-------|------|-----------|
| Virtual | | | | | | | |
| 0 | 0 | work | kernel segment | - | 6843 | 4290 | 1621 8454 |
| 3f8bd | d | work | loader segment | - | 1352 | 0 | 71 3062 |
| ... | | | | | | | |

- At this point there is one question, is this process running correctly and really needs this amount of memory?
- *Yes*, increase memory. This can be done obviously with adding physically memory, but also with dynamic operation if some memory is available in the system. In order to know the available memory in the system, connect to the HMC, edit the system properties and select **Memory** tab. In Figure 8-9 on page 313, 688 MB of memory are available in the system.
 - *No*, check, debug or tune the application.

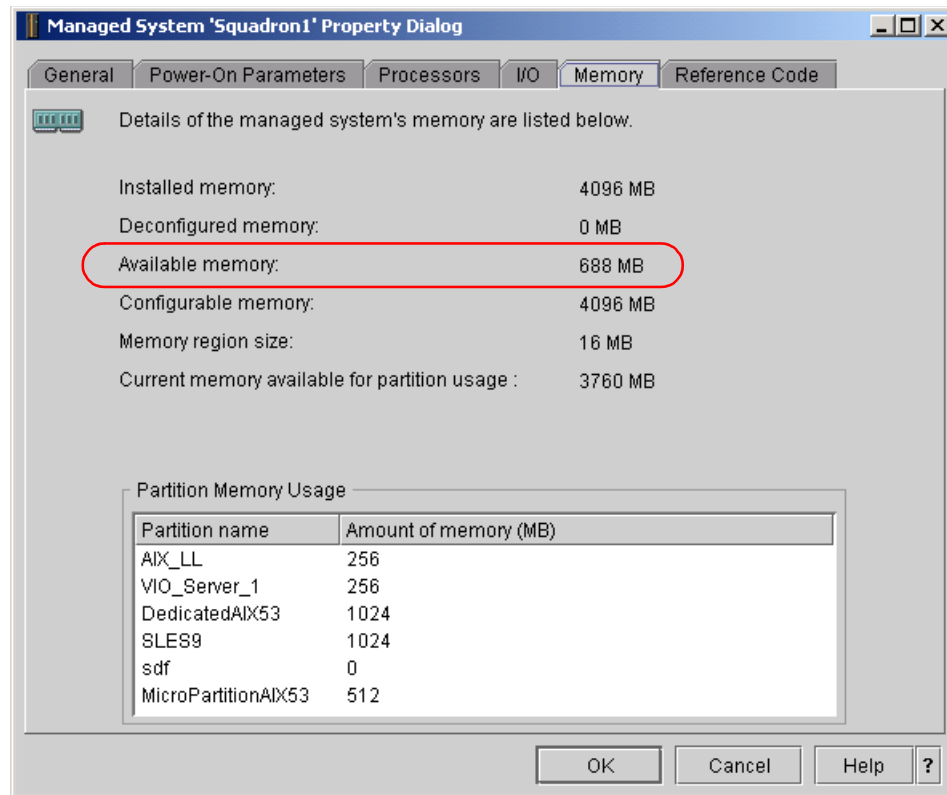


Figure 8-9 System properties - Memory

8.2.4 Disk I/O analysis

When a system has been identified having disk I/O performance problems, the next point is to find where does the problem come from. Figure 8-10 on page 314 shows the steps to follow on a disk I/O bound system, the items to verify are:

1. For a dedicated device.
 - a. Check the dedicated adapter.
 - b. Check the dedicated disk.
2. For a virtual device.
 - a. Check CPU on virtual I/O server.
 - b. Check adapter on virtual I/O server.
 - c. Check disk on virtual I/O server.

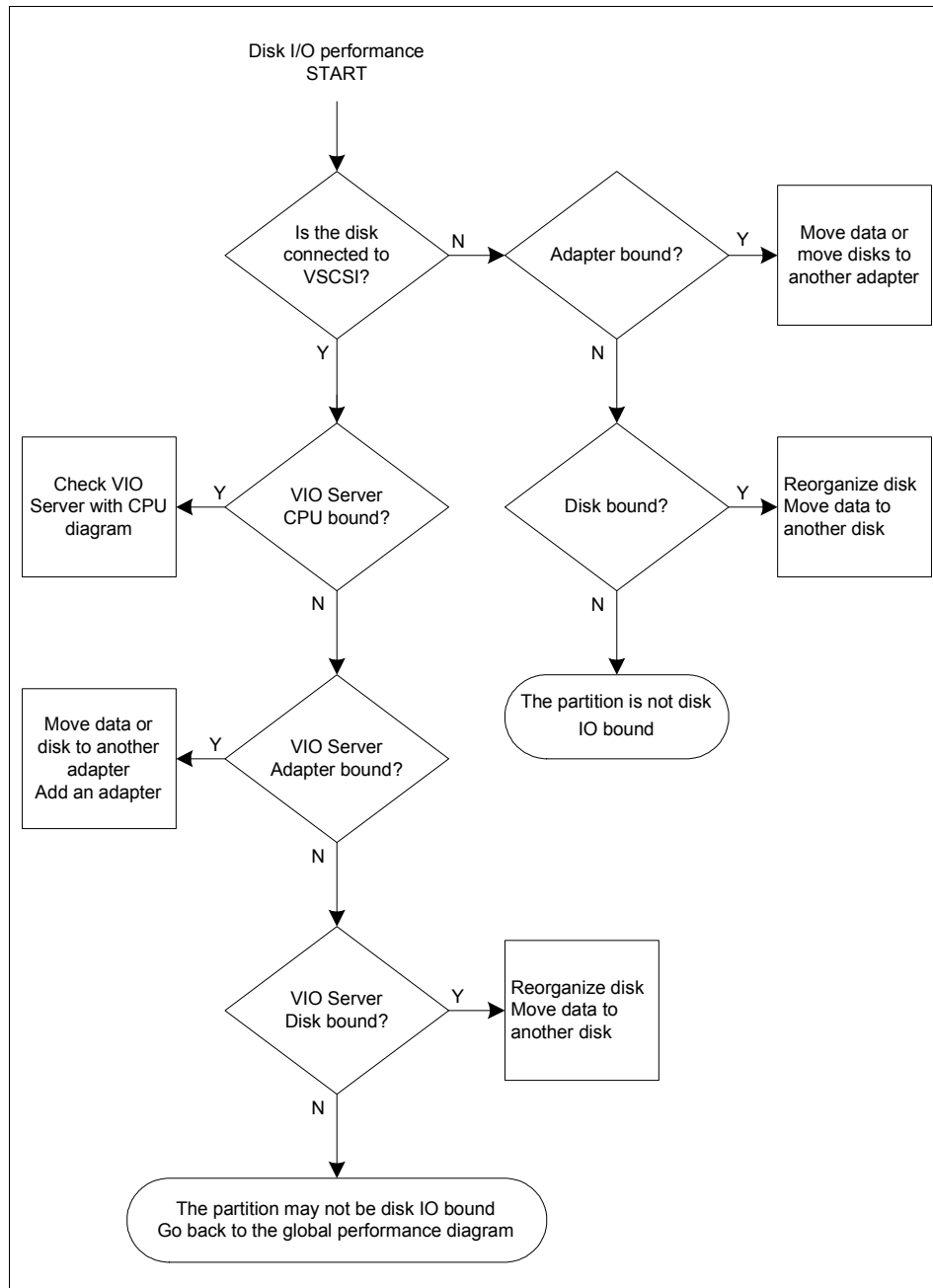


Figure 8-10 Disk I/O analysis diagram.

Dedicated or virtual device

First check if the disk is a physical one belonging to the partition or a Virtual SCSI disk. This can be done with **lscfg** command as shown in Example 8-28 on page 315, hdisk0, hdisk1 and hdisk2 are physical drives and hdisk3 is a Virtual SCSI drive. The command **lsdev** command gives the adapter name to which the disk is connected.

Example 8-28 Physical and virtual disks

```
# lsdev -Ccdisk
hdisk0 Available 02-08-00-3,0 16 Bit LVD SCSI Disk Drive
hdisk1 Available 02-08-00-4,0 16 Bit LVD SCSI Disk Drive
hdisk2 Available 02-08-00-5,0 16 Bit LVD SCSI Disk Drive
hdisk3 Available Virtual SCSI Disk Drive

# lsdev -Cl hdisk0 -F parent
vscsi0
```

Dedicated device

If a dedicated device (a physical adapter allocated to the partition) is bound, first check the adapter, then the disk.

Dedicated adapter

The activity of a disk adapter is given by **iostat -a** command. In Example 8-29 on page 315 the adapter sisscsia0 has a throughput of 28537 kilobytes per second. Because the maximum bandwidth of an adapter depends on its type and technology, compare the statistics given by **iostat** to the theoretical value to know the load percentage of the adapter. If the adapter is overloaded, try to move some data to another disk on a distinct adapter, move a physical disk to another adapter or add a disk adapter.

Example 8-29 iostat - adapter statistics

```
# iostat -a 5

System configuration: lcpu=1 drives=3

tty:      tin      tout  avg-cpu:  % user   % sys    % idle   % iowait
          0.0      36.4          16.5    35.2     26.2     22.1

Adapter:           Kbps    tps    Kb_read  Kb_wrtn
sisscsia0          28537.3   132.1    72619    70424

Paths/Disk:      % tm_act  Kbps    tps    Kb_read  Kb_wrtn
hdisk2_Path0      9.6      28537.3  132.1    72619    70424
hdisk0_Path0      0.0       0.0     0.0       0         0
```

| | | | | | |
|--------------|-----|-----|-----|---|---|
| hdisk1_Path0 | 0.0 | 0.0 | 0.0 | 0 | 0 |
|--------------|-----|-----|-----|---|---|

Dedicated disk

The disk may be bound just because all the data are not well organized. Verify the placement of logical volumes on the disk with **lspv** command. If logical volumes are fragmented across the disk like in Example 8-30 on page 316, reorganize them with **reorgvg** or **migratepv** command.

Example 8-30 Fragmented logical volumes

```
# lspv -p hdisk0
hdisk0:
```

| PP RANGE | STATE | REGION | LV NAME | TYPE | MOUNT POINT |
|----------|-------|--------------|---------|---------------|-------------|
| 1-3 | used | outer edge | hd5 | boot | N/A |
| 4-13 | used | outer edge | hd6 | paging | N/A |
| 14-49 | free | outer edge | | | |
| 50-124 | used | outer edge | fslv01 | jfs2 | /test2 |
| 125-125 | used | outer edge | hd10opt | jfs2 | /opt |
| 126-135 | free | outer edge | | | |
| 136-154 | used | outer edge | hd10opt | jfs2 | /opt |
| 155-165 | free | outer middle | | | |
| 166-166 | used | outer middle | hd6 | paging | N/A |
| 167-176 | free | outer middle | | | |
| 177-189 | used | outer middle | hd6 | paging | N/A |
| 190-199 | used | outer middle | hd2 | jfs2 | /usr |
| 200-210 | used | outer middle | hd6 | paging | N/A |
| 211-220 | used | outer middle | hd2 | jfs2 | /usr |
| 221-282 | used | outer middle | hd6 | paging | N/A |
| 283-283 | used | outer middle | loglv00 | jfslog | N/A |
| 284-287 | free | outer middle | | | |
| 288-291 | used | outer middle | hd1 | jfs2 | /home |
| 292-301 | used | outer middle | hd3 | jfs2 | /tmp |
| 302-307 | used | outer middle | hd9var | jfs2 | /var |
| 308-308 | used | center | hd8 | jfs2log | N/A |
| 309-316 | used | center | hd4 | jfs2 | / |
| 317-320 | used | center | hd2 | jfs2 | /usr |
| 321-330 | used | center | hd10opt | jfs2 | /opt |
| 331-380 | used | center | hd2 | jfs2 | /usr |
| 381-390 | free | center | | | |
| 391-460 | used | center | hd2 | jfs2 | /usr |
| 461-590 | used | inner middle | hd2 | jfs2 | /usr |
| 591-591 | free | inner middle | | | |
| 592-601 | used | inner middle | hd6 | paging | N/A |
| 602-613 | free | inner middle | | | |
| 614-688 | used | inner edge | fslv02 | jfs2 | /test1 |
| 689-689 | free | inner edge | | | |
| 690-700 | used | inner edge | hd6 | paging | N/A |
| 701-712 | free | inner edge | | | |

| | | | | | |
|---------|------|------------|-----|--------|-----|
| 713-722 | used | inner edge | hd6 | paging | N/A |
| 723-767 | free | inner edge | | | |

If logical volumes are well organized in the disks, the problem may come from the file distribution in the file system. The **fileplace** command displays the file organization as shown in Example 8-31 on page 317. In this case space efficiency is near 100% that means the file has really few fragments and they are contiguous. In order to increases a file system's contiguous free space by reorganizing allocations to be contiguous rather than scattered across the disk, use the **defragfs** command.

Example 8-31 fileplace output

```
# fileplace -lv testFile

File: testFile Size: 304998050 bytes Vol: /dev/fslv00
Blk Size: 4096 Frag Size: 4096 Nfrags: 74463
Inode: 4 Mode: -rw-r--r-- Owner: root Group: system

Logical Extent
-----
00000064-00000511      448 frags      1835008 Bytes,   0.6%
00003328-00028511    25184 frags    103153664 Bytes,  33.8%
00054176-00077759    23584 frags    96600064 Bytes,  31.7%
00000048-00000063      16 frags      65536 Bytes,   0.0%
00003312-00003327      16 frags      65536 Bytes,   0.0%
00032760-00032767       8 frags      32768 Bytes,   0.0%
00077760-00077767       8 frags      32768 Bytes,   0.0%
00000044-00000047       4 frags      16384 Bytes,   0.0%
00003296-00003308      13 frags      53248 Bytes,   0.0%
00028512-00032735     4224 frags     17301504 Bytes,   5.7%
00032768-00053725    20958 frags     85843968 Bytes,  28.1%

74463 frags over space of 77724 frags:   space efficiency = 95.8%
11 extents out of 74463 possible:   sequentiality = 100.0%
```

Author Comment: if time, talk more about fileplace, defragfs, filemon ioo

For more details on disk performance refer to AIX 5L Performance Tools Handbook, SG24-6039.

Virtual device

If the bound device is a Virtual SCSI disk, refer to 8.2.2, “CPU analysis” on page 306 to check CPU activity on the VIO Server.

The following steps describe how to find in a VIO Server a physical volume hosting a virtual disk allocated to a partition:

1. Get the slot number of the Virtual SCSI adapter for the partition as shown in Example 8-32 on page 318.

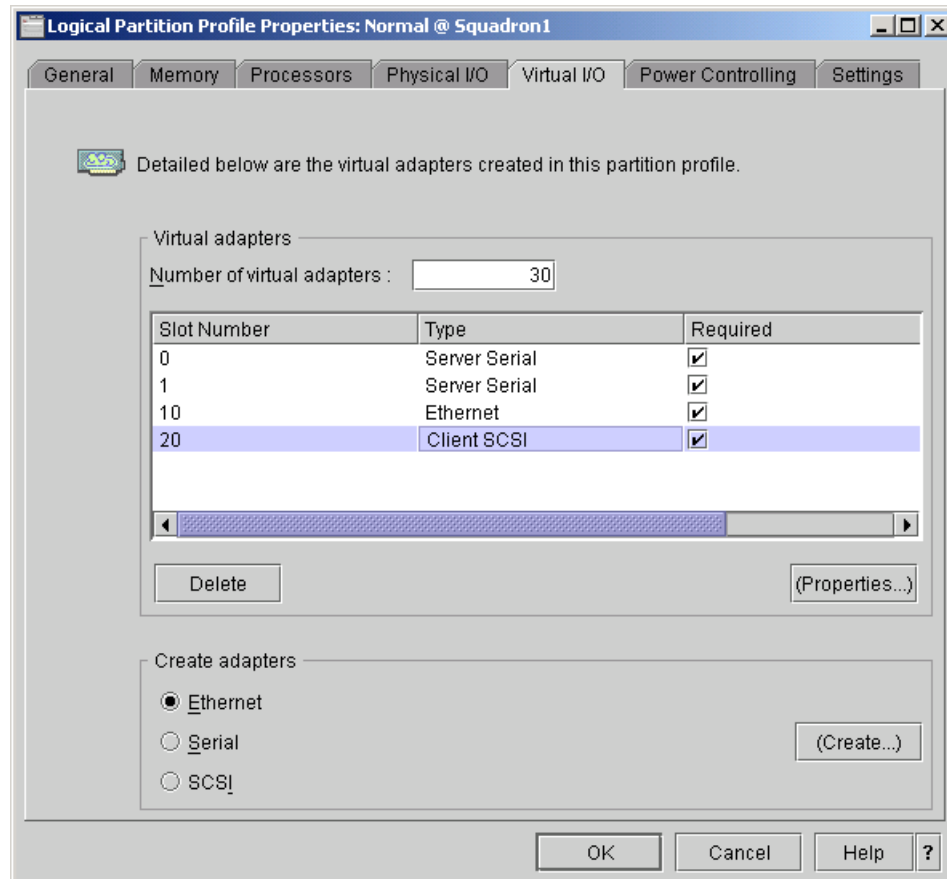
Example 8-32 Virtual SCSI adapter slot number

```
# lsdev -Cl hdisk0 -F parent
vscsi0

# lscfg -vl vscsi0
vscsi0          U9111.520.10DDEDC-V4-C20-T1  Virtual SCSI Client Adapter

Device Specific.(YL).....U9111.520.10DDEDC-V4-C20-T1
```

2. Check the partition profile on the HMC and collect the slot number of the VIO Server which is associated with the slot number found previously in the partition. To do this, connect to the HMC, edit the profile properties, click on **Virtual I/O** tab, select the Client SCSI line and click on **(Properties...)** as shown in Figure 8-11 on page 319. A new window displays the virtual SCSI adapter properties with slot numbers as shown in Figure 8-12 on page 320.

*Figure 8-11 Virtual I/O adapters*

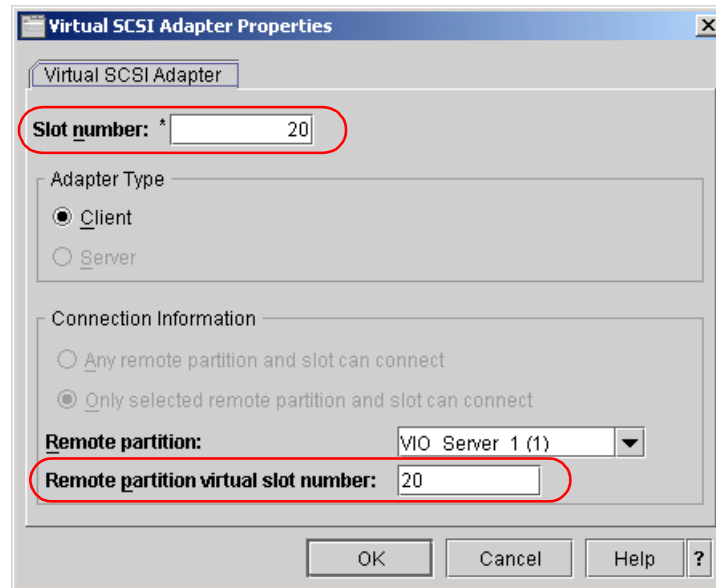


Figure 8-12 Virtual SCSI Adapter Properties.

3. Find the name of the disk which contains the partition's data as shown in Example 8-33 on page 320 by following the next steps:
 - a. Find the Virtual SCSI Server adapter with **lsdev** command.
 - b. Find the logical volume name with **lsmap** command.
 - c. Find the volume group name with **lslv** command.
 - d. Find the disk name with **lsvg** command.

Example 8-33 VIO Server commands to find a disk.

```
$ lsdev -vpd | grep vhost.*C20
vhost2          U9111.520.10DDEDC-V1-C20          Virtual SCSI Server Adapter

$ lsmap -vadapter vhost2
SVSA          Physloc          Client Partition
ID
-----
vhost2        U9111.520.10DDEDC-V1-C20          0x00000004

VTD          vMicroPartAIX53
LUN          0x8100000000000000
Backing device MicroPartAIX53
Physloc
```

```
$ ls1v MicroPartAIX53
LOGICAL VOLUME:      MicroPartAIX53          VOLUME GROUP:  rootvg_clients
LV IDENTIFIER:       00cddedc00004c000000000102a1f53e.4  PERMISSION:
read/write
VG STATE:            active/complete          LV STATE:       opened/syncd
TYPE:                jfs                      WRITE VERIFY:   off
MAX LPs:             32512                    PP SIZE:       32 megabyte(s)
COPIES:              1                       SCHED POLICY:  parallel
LPs:                 96                      PPs:           96
STALE PPs:           0                      BB POLICY:     non-relocatable
INTER-POLICY:        minimum                 RELOCATABLE:   yes
INTRA-POLICY:        middle                  UPPER BOUND:   1024
MOUNT POINT:         N/A                     LABEL:         None
MIRROR WRITE CONSISTENCY: on/ACTIVE
EACH LP COPY ON A SEPARATE PV ?: yes
Serialize IO ?:      NO
DEVICESUBTYPE : DS_LVZ

$ ls1vg -pv rootvg_clients
rootvg_clients:
PV_NAME      PV STATE      TOTAL PPs   FREE PPs   FREE DISTRIBUTION
hdisk1       active        1082        26         00..00..00..00..26
```

Virtual adapter

If the VIO Server is not CPU bound, check the adapter activity. if many disks experience performance problems on the same adapter, it may be overloaded. In that case move some of the data to another disk on a different adapter (if any available) or add physical adapter.

Virtual disk

If only one disk has performance problem, verify the placement of logical volumes on the disk on both side, on the VIO Server as shown in Example 8-34 on page 321 and on the partition using this Virtual SCSI disk. If logical volumes are fragmented across the disk reorganize them with **reorgvg** or **migratepv** command. For more details on virtual disks refer to section 7.5, “Virtual SCSI” on page 225.

Example 8-34 lspv - logical volume placement

```
$ lspv -pv hdisk1
hdisk1:
PP RANGE  STATE  REGION      LV NAME          TYPE  MOUNT POINT
  1-25    used   outer edge   rootvg_l1        jfs   N/A
 26-121   used   outer edge   MicroPartAIX53   jfs   N/A
122-217   used   outer edge   rootvg_l1        jfs   N/A
218-433   used   outer middle rootvg_aix53      jfs   N/A
434-537   used   center      rootvg_aix53      jfs   N/A
```

| | | | | | |
|-----------|------|--------------|--------------|-----|-----|
| 538-649 | used | center | rootvg_sles9 | jfs | N/A |
| 650-857 | used | inner middle | rootvg_sles9 | jfs | N/A |
| 858-865 | used | inner middle | fs1_ll | jfs | N/A |
| 866-985 | used | inner edge | fs1_ll | jfs | N/A |
| 986-1056 | used | inner edge | rootvg_ll | jfs | N/A |
| 1057-1082 | free | inner edge | | | |

For more details on disk performance refer to AIX 5L Performance Tools Handbook, SG24-6039.

8.2.5 Network I/O analysis

When a system has been identified having network I/O performance problems, the next point is to find where does the problem come from. Figure 8-13 on page 323 shows the steps to follow on a disk I/O bound system, the items to verify are:

- 1. For a dedicated adapter:
 - a. Check the dedicated adapter statistics.
- 2. For a virtual adapter:
 - a. Virtual Ethernet adapter
 - i. Check CPU utilization
 - ii. Check physical adapter
 - b. Shared Ethernet adapter
 - i. Check adapter statistics

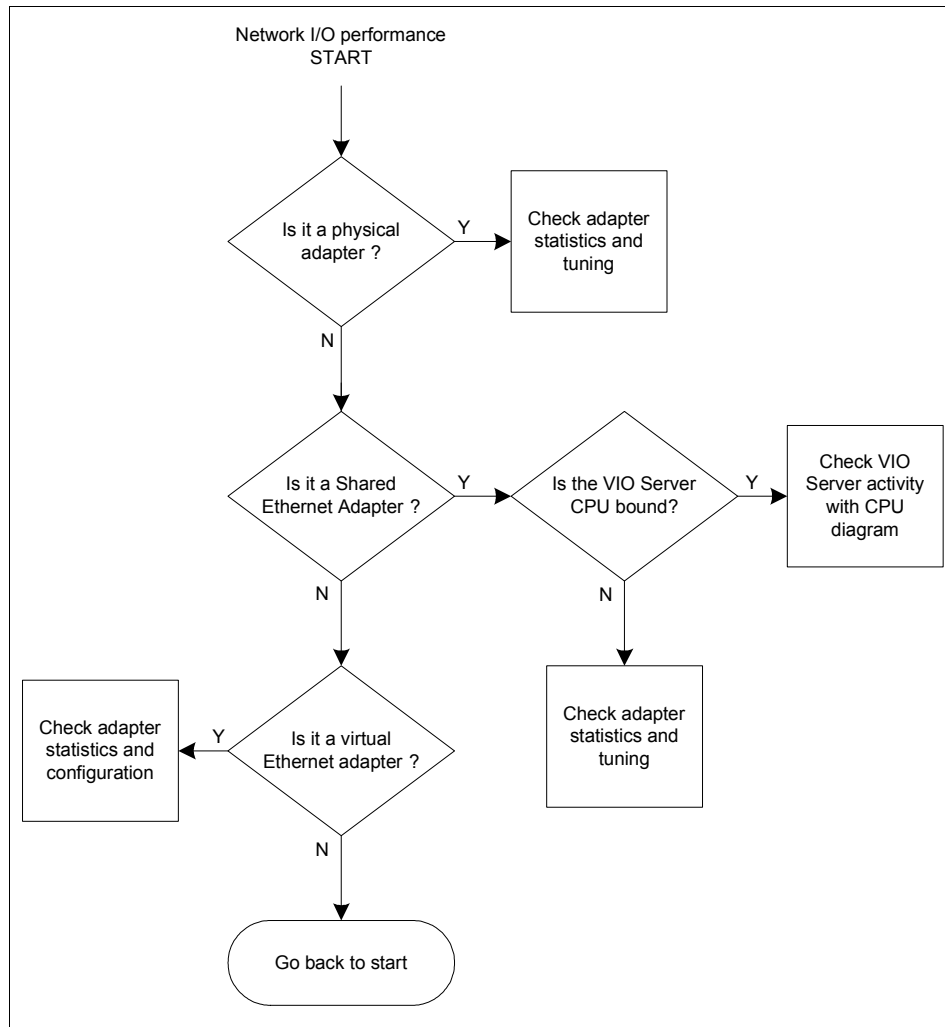


Figure 8-13 Network I/O analysis diagram.

Dedicated adapter

If the system is network I/O bound because of a dedicated adapter, check it with **netstat**, **entstat** commands, modify the configuration **no** and **chdev** commands. If the partition is using NFS, check the statistics with **nfsstat** command.

For more details on network performance refer to AIX 5L Performance Tools Handbook, SG24-6039.

Virtual adapter

A virtual adapter is provided by the VIO server, it can be a Virtual Ethernet adapter or a Shared Ethernet adapter. For more details on VIO server refer to Chapter 7, “Virtual I/O” on page 173.

Virtual Ethernet adapter

If a virtual Ethernet adapter is bound, check the adapter statistics with **entstat** command. Check adapter memory usage with **netstat** command to validate there is enough buffer allocated to this adapter and check if the system is not memory bound.

Verify the adapter configuration with **lsattr** command like in Example 8-35 on page 324. If the adapter is only for communication between partitions using the same VLAN (no traffic going outside the system) then the mtu can be increased up to 64000.

Example 8-35 Network adapter parameters

| | | | |
|------------------|-------------|--|------|
| # lsattr -El en1 | | | |
| alias4 | | IPv4 Alias including Subnet Mask | True |
| alias6 | | IPv6 Alias including Prefix Length | True |
| arp | on | Address Resolution Protocol (ARP) | True |
| authority | | Authorized Users | True |
| broadcast | | Broadcast Address | True |
| mtu | 1500 | Maximum IP Packet Size for This Device | True |
| netaddr | 9.3.5.150 | Internet Address | True |
| netaddr6 | | IPv6 Internet Address | True |
| netmask | 255.255.0.0 | Subnet Mask | True |
| prefixlen | | Prefix Length for IPv6 Internet Address | True |
| remmtu | 576 | Maximum IP Packet Size for REMOTE Networks | True |
| rfc1323 | | Enable/Disable TCP RFC 1323 Window Scaling | True |
| security | none | Security Level | True |
| state | up | Current Interface Status | True |
| tcp_mssdflt | | Set TCP Maximum Segment Size | True |
| tcp_nodelay | | Enable/Disable TCP_NODELAY Option | True |
| tcp_recvspace | | Set Socket Buffer Space for Receiving | True |
| tcp_sendspace | | Set Socket Buffer Space for Sending | True |

For more detail on VLAN refer to section 7.3, “Virtual Ethernet” on page 181.

Shared Ethernet Adapter (SEA)

If the system is network I/O bound because of a SEA, check it on the VIO server with **netstat**, **entstat** commands, modify the configuration **chdev** command. Example 8-36 on page 325 shows an Ethernet adapter statistics, there is a lot of errors and collisions reported by this adapter.

Example 8-36 entstat - VIO server

```
$ entstat en3
```

ETHERNET STATISTICS (en3) :

Device Type: Shared Ethernet Adapter

Hardware Address: 00:09:6b:6b:05:b1

Elapsed Time: 0 days 0 hours 0 minutes 0 seconds

Transmit Statistics:

Packets: 5656592

Bytes: 7666680307

Interrupts: 0

Transmit Errors: 359712

Packets Dropped: 0

Receive Statistics:

Packets: 4189578

Bytes: 365071314

Interrupts: 3841545

Receive Errors: 0

Packets Dropped: 0

Bad Packets: 0

Max Packets on S/W Transmit Queue: 98

S/W Transmit Queue Overflow: 0

Current S/W+H/W Transmit Queue Length: 2

Broadcast Packets: 1615551

Multicast Packets: 622570

No Carrier Sense: 0

DMA Underrun: 0

Lost CTS Errors: 0

Max Collision Errors: 0

Late Collision Errors: 359712

Deferred: 207086

SQE Test: 0

Timeout Errors: 0

Single Collision Count: 191677

Multiple Collision Count: 48

Current HW Transmit Queue Length: 2

Broadcast Packets: 1610367

Multicast Packets: 622568

CRC Errors: 0

DMA Overrun: 0

Alignment Errors: 0

No Resource Errors: 0

Receive Collision Errors: 0

Packet Too Short Errors: 0

Packet Too Long Errors: 0

Packets Discarded by Adapter: 0

Receiver Start Count: 0

General Statistics:

No mbuf Errors: 0

Adapter Reset Count: 0

Driver Flags: Up Broadcast Running

Simplex 64BitSupport

Network parameters like *thewall*, *tcp_sendspace*, *tcp_recvspace* can be tuned with **optimizenet** command, information about available parameters like default, current and range is displayed in Example 8-37 on page 326.

Example 8-37 Network parameters list

| | | | | | | | |
|----------------------|-----|-----|------|-----|-------|-------------|------|
| \$ optimizenet -list | | | | | | | |
| NAME | CUR | DEF | BOOT | MIN | MAX | UNIT | TYPE |
| DEPENDENCIES | | | | | | | |
| arptab_bsiz | 7 | 7 | 7 | 1 | 32K-1 | bucket_size | R |
| NAME | CUR | DEF | BOOT | MIN | MAX | UNIT | TYPE |
| DEPENDENCIES | | | | | | | |
| arptab_nb | 73 | 73 | 73 | 1 | 32K-1 | buckets | R |
| NAME | CUR | DEF | BOOT | MIN | MAX | UNIT | TYPE |
| DEPENDENCIES | | | | | | | |
| clean_partial_conns | 0 | 0 | 0 | 0 | 1 | boolean | D |
| NAME | CUR | DEF | BOOT | MIN | MAX | UNIT | TYPE |
| DEPENDENCIES | | | | | | | |
| net_malloc_police | 0 | 0 | 0 | 0 | 8E-1 | numeric | D |
| NAME | CUR | DEF | BOOT | MIN | MAX | UNIT | TYPE |
| DEPENDENCIES | | | | | | | |
| rfc1323 | 0 | 0 | 0 | 0 | 1 | boolean | C |
| NAME | CUR | DEF | BOOT | MIN | MAX | UNIT | TYPE |
| DEPENDENCIES | | | | | | | |
| route_expire | 1 | 1 | 1 | 0 | 1 | boolean | D |
| NAME | CUR | DEF | BOOT | MIN | MAX | UNIT | TYPE |
| DEPENDENCIES | | | | | | | |
| tcp_pmtu_discover | 1 | 1 | 1 | 0 | 1 | boolean | D |
| NAME | CUR | DEF | BOOT | MIN | MAX | UNIT | TYPE |
| DEPENDENCIES | | | | | | | |

| | | | | | | | |
|---------------|-------|-------|-------|-----|------|-------|------|
| tcp_recvspace | 16K | 16K | 16K | 4K | 8E-1 | byte | C |
| sb_max | | | | | | | |
| ----- | | | | | | | |
| NAME | CUR | DEF | BOOT | MIN | MAX | UNIT | TYPE |
| DEPENDENCIES | | | | | | | |
| ----- | | | | | | | |
| tcp_sendspace | 4K | 16K | 16K | 4K | 8E-1 | byte | C |
| sb_max | | | | | | | |
| ----- | | | | | | | |
| NAME | CUR | DEF | BOOT | MIN | MAX | UNIT | TYPE |
| DEPENDENCIES | | | | | | | |
| ----- | | | | | | | |
| thewall | 128K | 128K | 128K | 0 | 1M | kbyte | S |
| ----- | | | | | | | |
| NAME | CUR | DEF | BOOT | MIN | MAX | UNIT | TYPE |
| DEPENDENCIES | | | | | | | |
| ----- | | | | | | | |
| udp_recvspace | 42080 | 42080 | 42080 | 4K | 8E-1 | byte | C |
| sb_max | | | | | | | |
| ----- | | | | | | | |
| NAME | CUR | DEF | BOOT | MIN | MAX | UNIT | TYPE |
| DEPENDENCIES | | | | | | | |
| ----- | | | | | | | |
| udp_sendspace | 9K | 9K | 9K | 4K | 8E-1 | byte | C |
| sb_max | | | | | | | |
| ----- | | | | | | | |

For more detail on SEA refer to section 7.4, “Shared Ethernet adapter functionality” on page 203.



Partition Load Manager (PLM)

A general presentation of Partition Load Manager is provided in chapters 3 and 6 of *Advanced POWER Virtualization on IBM eServer p5 Servers Introduction and Basic Configuration*, SG24-7940. Chapter 3 contains a description of the general behavior of PLM, while Chapter 6 contains a detailed explanation of PLM installation and configuration.

In addition, you can find the PLM product documentation in:

The *eServer Partitioning for AIX* guide:

http://publib.boulder.ibm.com/infocenter/eserver/v1r2s/en_US/info/iphbk/iphbk.pdf

The *eServer Virtual I/O Server and Partition Load Manager Commands Reference guide*:

http://publib.boulder.ibm.com/infocenter/eserver/v1r2s/en_US/info/iphb1/commands/commands.pdf

We only present in this redbook chapter information about PLM that is not already described in the *Advanced POWER Virtualization on IBM eServer p5 Servers Introduction and Basic Configuration*, SG24-7940.

9.1 When and how should I use PLM

Partition Load Manager (PLM) for AIX 5L is a load manager that provides automated processor and memory resource management across DLPAR capable logical partitions running AIX 5L v5.2 or AIX 5L v5.3. PLM allocates resources to partitions on-demand, within the constraints of a user-defined policy. It assigns resources from partitions with low usage to partitions with a higher demand, improving the overall resource utilization of the system. PLM works with both dedicated partitions and micro-partitions environment.

PLM is an optional features of p5 servers. In many cases, you can perfectly run your p5 server without PLM. First, this section shows how PLM relates to other workload management tools available on the p5 servers. This section then describes the cases where you would want to take advantage of PLM. Finally, it provides best practices for deploying PLM.

Note: PLM can be used to manage AIX partitions. PLM does not currently manage Linux partitions. PLM can manage Virtual I/O server partitions

Although it is possible to manage the resources of a Virtual I/O server with PLM, this requires a manual setup of PLM, so that RSH can be used on this partition. We therefore do not recommend you perform this kind of resource management.

9.1.1 PLM and other load balancing tools.

The Power 5 hypervisor provides some built-in features to automatically allocate physical CPU resources. AIX also provides load balancing possibilities using Workload Manager (WLM). We will position PLM versus these tools.

PLM versus hypervisor resource allocation.

For dedicated partitions and capped micro-partitions, the Hypervisor only deals with resource allocation at boot time and during DLPAR operations generated from the HMC. At boot time, the hypervisor will check for available resources and present the booting partition with either its desired amount of resources if available, or a smaller amount if not. After the partition is booted, the hypervisor will not modify the resources allocated to the partition unless a system administrator issues commands from the HMC.

Furthermore, if dedicated partitions (that are defined on the servers) are not booted, their unused processors will be moved by the hypervisor to the free processor pool for use by micro-partitions, and given back to the dedicated partitions when they boot.

For uncapped micro-partitions, AIX provides additional resource load balancing. For example a partition's unused processor time is given to other partitions when the kernel calls the H-CEDE or H-CONFER hypervisor calls. This freed processor time is then shared between the uncapped micro-partitions. This resource management is limited to processor and does not balance memory.

Once partitions are booted, the hypervisor makes sure that each partition gets its share of resources as defined on the HMC:

- ▶ memory and number of physical processors for dedicated partitions
- ▶ memory, processing units and number of virtual processors for micro-partitions.
- ▶ possibly additional processing power for uncapped micro-partitions.

But the hypervisor does not by itself dynamically modify these resource shares.

PLM adds a level of automation in the allocation of resources. PLM dynamically changes the amount of resources given to each partition.

PLM versus WLM

PLM and WLM can be used concurrently. There is no overlap in their scope:

- ▶ The scope of PLM is the set of hardware resources (processors and memory) that are configured in a physical server. PLM dynamically allocates these physical resources to partitions.
- ▶ The scope of WLM is the set of resources (processor, memory and disk I/O) within one partition. WLM will dynamically allocate these resources among the processes running within one partition.

Since WLM allocates resources according to a configuration files that contains relative values (percentage or shares) and not absolute values, WLM can work independently of the allocation of resources by PLM to a partition.

Let us take a (very simplified) example. A partition contains 2 processes P1 and P2 that are managed by WLM, so that P1 gets 3 shares of CPU and P2 gets 2 shares. Initially, the partition is allocated 2 physical processors, so P1 is using 60% of 2 processors = 1.2 processor and P2 is using 40% of 2 processors = 0.8 processor. If later on, processors become unused in the server and the partition is given 3 extra processors by PLM, P1 will now get 60% of 5 processors = 3 processors, and P2 will get 2 processors. The WLM priorities defined by shares are kept unchanged and each process gets more physical resources thanks to the PLM action, without any need to modify the WLM settings. In other words, WLM manages percentages of available resources given by PLM.

9.1.2 When to use PLM

PLM provides an automated way to move processing power and memory between DLPAR capable logical partitions. The p5 servers support two types of partitions (dedicated partitions and micro-partition), and PLM provides slightly different benefits in each case. The following sections presents situations where PLM can be used, first for managing dedicated partitions, then to manage micro-partitions.

PLM for dedicated partitions

In dedicated partitions PLM is a replacement for the manual DLPAR reallocation of resources that a system administrator performs using the HMC. Rather than having to monitor partitions for lack or excess of computing resources, the system administrator can define, in a configuration file, thresholds for the use of these resources by the partitions. PLM will monitor the actual resources utilization against these thresholds, and automatically move some resources from the partition with a low demand to the partition with a high demand of these resources.

If your system contains partitions with a fairly constant workload and equal relative priority, PLM may offer little benefit. If however your manually reconfigure your partitions using the HMC DPLAR feature, or if your partitions have a changing resource demand over time, PLM can help you. Let us take a few examples.

Partitions with unpredictable workload peaks.

You may have decided to consolidate onto a large p5 server, several applications that were previously running on independent servers. Each independent server was sized with enough processors and memory to satisfy the application peak loads, but these resources were unused a significant part of the time. Because the peaks of all applications are statistically spread over time, you were able to install a new server with less overall computing resources. Each application now runs in its own partition, for which you define a minimum, a desired and a maximum amount of processing power and memory. When all partitions are booted, they are each given their desired amount of resources (assuming the sum of desired values is less than the overall server capacity). The amount of resources allocated to each partition will not change when one partition reaches a peak of activity. By activating PLM with one configuration file using the same minimum/desired/maximum threshold than those defined in the HMC, the partition with a peak of activity will automatically take advantage of the unused resources.

Partitions with time based priorities.

Let us assume that a server contains two partitions which must be given different priority over time.

- ▶ For example, one partition is providing interactive service to human end-users, and we only want to give priority to this partition when the end-users are awake. The other partition is processing batch type jobs and the we want to give it priority at night time, even though some human end-users may access the system.
- ▶ Another example of such a scenario is with one application that processes weekly reports, that will get most of the server resources on Saturdays and Sundays, while another application produces the data during the week.

In these cases, we can define two PLM configurations that will be activated at different times. Let's take a concrete example.

The server is an 8 way server with 32 Gbytes of memory.

On the HMC, we defined partitions P1 and P2 with the same amount of resources, so they can each use up to 7 processors and 28 Gbytes of memory as shown in Table 9-1.

Table 9-1 Partitions resources definition on the HMC

| | Minimum | Desired | Maximum |
|------------|---------|---------|---------|
| Processors | 1 | 2 | 7 |
| Memory | 4 | 8 | 28 |

Then we defined two PLM configurations, for each of the time periods (Night/day or weekdays/weekends). Table 9-2 shows the settings of the resources in PLM for the time period where partition P2 will have priority over P1.

Table 9-2 Resources allocations for time period 1

| | | Minimum | Desired | Maximum |
|----|------------|---------|---------|---------|
| P1 | Processors | 1 | 2 | 3 |
| | Memory | 4 | 8 | 12 |
| P2 | Processors | 5 | 6 | 7 |
| | Memory | 20 | 24 | 28 |

The table for the other time period would give the opposite values to each partitions. With these values, we guarantee that the partition with the highest priority gets at least 5 processors (and up to 7). The partition with the low priority

could go to as little as one processor if the other partition has a very high resource demand. Furthermore, compared to not using PLM, this configuration adds some flexibility to the configuration within each time-period, since the memory and processing resource of each partition can still fluctuate. For example, if the high priority partitions does not use all its processing resource, the low priority partition will be able to automatically use up to 3 processors.

PLM does not provide a time based re-configuration feature. The change of configuration at the boundary between the 2 time periods would be implemented using for example the **cron** command to load a new configuration file in the PLM manager.

When you plan to activate several PLM policies over time, using low minimum and high maximum values in the partition profiles on the HMC allow for more flexibility with the PLM policies since these (HMC profile) values bound the values that can be used in the PLM policy.

Partitions subsets: multiple customers

This scenario addresses the case where a large server is used to host applications belonging to several customers. Each customers has paid for a fixed amount of resources. However, each customers needs to run several partitions.

You can then take advantage of the PLM concept of partition group.

A group is allocated a number of processors and a chunk of memory. PLM will then reallocate these resources between the partitions belonging to this group. At least one group must be defined, and the simplest way to use PLM is to gather all managed partitions within the same

When managing multiple customers, a better way is to define for each customer a group which is allocated exactly the resources for which the customers pays for. For example, the server contains 64 processors, and one customer pays to use 16 processors. If the customer needs 5 partitions, the customer can then define the PLM configuration file with one group owning these 16 processors. All this customer partitions belong to this group. The customer can then configure the policies according to his preferences for allocation of free resources between his own partitions. This would have no impact on the allocation of the remaining 48 processors belonging to other customers.

PLM with micro-partitions

The scenarios described previously for dedicated partitions are also valid for micro-partitions. But there are other cases specific to micro-partitioning where you may want to use PLM.

Splitting the shared pool

All processors that will be used by micro-partitions belong to only one shared pool. By using the PLM groups, you can divide the shared pool in several subset, so that processor entitlement and number of virtual processors resource balancing by PLM is performed only within each groups. This is similar to the “*Partitions subsets: multiple customers*” described above for dedicated partitions.

Let us be more accurate on the behaviour of the system in the case multiple groups are created each with a max entitled capacity. PLM will distribute that capacity amongst the partitions within that group. However, if any of the running partitions in the system is uncapped, then unused cpu cycles from any PLM group can be given to the uncapped partition, whether it be within or outside the PLM group. So if a customer creates a PLM group and defines a max of 5 for the entitlement, if any of those partitions are uncapped, then that group can technically run with more than an entitlement of 5, since hypervisor can assign unused cycles from other partitions outside of our group.

PLM does not prevent the hypervisor from trying to optimize the overall system throughput.

Author Comment: Section ***Partitions with concurrent peak loads*** is a last minute addition. Check with Dean if he agrees with this test.

Partitions with concurrent peak loads

It often happens that several services have their workload peaks at the same time, with workload profile similar to those presented in Figure 6-16 and Figure 6-17 on page 162. If these services are running in partitions of the same p5 server, they will compete for resources. The business case for sizing the server may have used the “planned over-commit” strategy as described in Section 6.3.8, “Guidelines for planning Micro-Partitions” on page 164, for example because some of these services are not business-critical, and there is no justification for affording a system that can handle all applications workload peaks during small periods, while being under-utilized for long periods.

PLM can help to allocate the systems resources to the partitions that have the highest business priority while only providing the other partitions with left-over resources. There are two parameters that the system administrator can use for this purpose: `cpu_guaranteed` and `cpu_shares` (These parameters are described in Section 9.2.3, “Configuration file and tunables” on page 341).

- ▶ The first parameter represents an absolute value of processing power that a partition is guaranteed to be allocated only if it needs it. If the partition runs

below this guaranteed amount, the remaining capacity is available for the other partitions. By setting this parameters to a high value for all partitions considered as critical, you ensure them to have this processing power available whatever the workload of the other partitions.

- ▶ The `cpu-shares` parameter represents a relative value of processing power. It is only used to allocate the CPU resources in excess of the sum of the guaranteed CPU resources, between the partitions that needs extra power. Using a higher value of `cpu_share` for high priority partition than for lower priority partitions, allows to prioritize the distribution of the extra processing power that cannot be allocated otherwise.

Let us take an example of how to use a combination of these parameters. We assume that a systems runs 10 micro-partitions with 10 CPU in the free pool (total entitlement equal to 1000). 5 partitions have high priority and are allocated a guaranteed CPU power of 180. The 5 partitions with low priority are assigned only 20 guaranteed CPU power unit. If all partitions are experiencing a workload peak, except one of the high priority partitions which is idle, each partition is given its guaranteed power, and there is 180 unit left to distribute among them. It is then the values of each partition `cpu_share` that defines how to distribute this remaining power between the 9 competing partitions.

Managing the number of virtual processors

The processing power of a micro-partition is defined by two parameters:

- ▶ its entitlement (or percentage of one physical processor processing capacity)
- ▶ its number of virtual processors.

For an uncapped partition, the maximum processing power is reached when the hypervisor has allocated to this partition an entitlement equivalent to its allocated number of virtual processors running at 100%. If there are still some idle processors in the server, the micro-partition cannot use them and this unused processing power is wasted, since the hypervisor will not automatically add virtual processors to the partition. To take advantage of the available processing power, you need to:

- ▶ have defined the uncapped partition with a large maximum number of virtual processor.
- ▶ define PLM policies that will add virtual processors to the partition when the entitlement go above a threshold (default is 80% of potential entitlement).

Assuming that the uncapped partition has been booted with its desired number of processor, when it reaches a peak of activity and if there is free processing capacity in the free processor pool, the hypervisor will give the partition extra CPU cycles in the limit of its current number of virtual processors. In addition to the hypervisor action, PLM monitors several thresholds, and if one is crossed,

PLM will receive a message (through RMC), and will start increasing the partition entitled capacity and number of virtual processors, up to the maximum number defined in the HMC for the partition. AIX will automatically take into account the extra processors.

In a similar way, you can define low utilization threshold that will lower the number of virtual processors used by a partition when it has a low processing activity.

There is no rule that is valid for all application profiles, but in general, for the same overall CPU entitlement, the performance of a micro-partition is better with a small number of virtual processors running with a high CPU utilization, than with a large number of virtual processor running at a low CPU utilization. There are many factors that can influence this behavior. For example, an application that is not programmed to use parallelism or AIX software multithreading does not benefit from availability of multiple processors. If this application was to run on a physical SMP system, it would not take advantage of the physical processors. In the same way, when running on a micro-partition, it would not take advantage of the multiple virtual processor. On a micro-partition, you can improve the throughput by reducing the number of virtual processors: the hypervisor will spend less time dispatching these virtual processor and AIX will spend less time trying to allocate processes to the processors.

If your create uncapped partition on your system, and your workload profile is such that processing resources could be moved from partitions to partitions, we recommend that you start deploying PLM with the default values of the processor/entitlement ratios. Then, you can start changing the ratios to find the best fit for your applications.

Managing the memory

All the scenarios we just described were related to reallocation of processing power, PLM is also able to reallocate physical memory between partitions, and the scenarios we presented can also apply to memory resource balancing.

9.1.3 How to deploy PLM

Once you have chosen to use PLM, you need to decide where to install the PLM manager. Here are a few considerations to take into account:

- ▶ Currently, the PLM server can only run on AIX. Therefore, you cannot use the HMC as the PLM manager. You need to find an AIX server: it could be either a dedicated server, or a partition Power 4 or Power 5 based system.
- ▶ PLM does not require a dedicated AIX instance. It can run on a system which is also running other applications.
- ▶ One PLM instance manages only partitions within one physical server (one CEC¹). However, you can run multiple PLM instance on the same AIX system.

If you plan to manage partitions in many physical servers, you may want to centralize all PLM management functions within the same AIX instance, to provide a single point of control for all PLM operation within your computer environment.

- ▶ PLM uses very little processing resources. PLM uses Resource Management and Control (RMC) to communicate between the PLM manager and the managed partitions. When PLM is activated, it will set up monitoring of threshold values on each managed partition. When a threshold is reached, the managed partition will send an event through RMC to the management server who will take the appropriate action. The PLM manager does not poll the managed partitions. You can find example of resource requirements for a PLM server in section 9.4.1, "PLM resource requirements" on page 352.
- ▶ The PLM manager can run in one of the partitions that it manages.

With all these considerations in mind, we can propose a few recommended configurations:

- ▶ For a server farm or a large computing center, it is likely that there already exists some AIX servers that are dedicated to infrastructure support. These could be the control workstation of a PSSP² cluster or Management Station of a CSM³ cluster. They could also be software repository like a NIM⁴ server, or they could be a monitoring server like a Tivoli TEC or TMR. Since the PLM management function does require very little processor cycles, memory and disk space, it could be implemented on one of these infrastructure server as long as it has IP connectivity to all managed partitions. This solution has the advantage of providing a single point of control for all PLM related operations.
- ▶ For managing partitions in a single physical server, a very cheap PLM server can be instantiated by dedicating a small micro-partition. The needed disk space can be provided by a virtual disk exported from a VIO server (logical volume). The IP connection with the managed partitions can be implemented through virtual Ethernet in memory VLAN, so that no hardware ethernet adapters is required. One Physical Ethernet adapter would be needed for communication with the HMC, unless a put Shared Ethernet adapter can provide this connectivity.
- ▶ An even cheaper configuration is to install the PLM management function on one of the partitions running applications.

The choice will then depends on the operations guidelines of each site.

¹ CEC: Central Electronics Complex

² PSSP: Parallel System Support Program

³ CSM: Cluster Systems Management

⁴ NIM: The Network Installation Manager feature of AIX

9.2 More about PLM installation and setup

Chapter 6 of *Advanced POWER Virtualization on IBM @server p5 Servers Introduction and Basic Configuration*, SG24-7940, contains a detailed explanation of basic PLM installation and configuration.

In this redbook, we present more advanced installation and configuration options, but let's first briefly describe how PLM works before we investigate details of its configurations.

9.2.1 Overview of PLM behavior

PLM involves 3 (types of) entities:

- ▶ The PLM server that execute the PLM code and decides of the resources reallocation actions.
- ▶ The managed partitions, which can request more or less resources.
- ▶ The HMC that drives the physical server hosting the managed partitions. The HMC actually performs the resource reallocation actions decided by the PLM manager.

The PLM resource manager is the server part of this client-server model and it runs on AIX 5L v5.2 and AIX 5L v5.3. When it starts, it registers, using RMC services, several events on every client partition that will be managed by PLM. In order for PLM to get system information and dynamically reconfigure resources, it requires an SSH network connection from the PLM manager to the HMC, as well as IP connectivity between the PLM manager, the HMC and the managed partitions. The RMC services are responsible to gather all the status information. The RMC daemon exports system status attributes and processes the re-configuration requests from HMC. With this data and in conjunction with the user-defined resource management policy, PLM decides what to do. Every time a partition exceeds a threshold, PLM receives a RMC event. When a node requests additional resources, PLM determines whether the node can accept additional resources. If the node can accept additional resources, PLM conducts a search for available resources. It then checks the policy file in order to see if a partition is more or less deserving of the resources. Only then, PLM allocates the resources requested.

PLM uses a Micro-Partitioning entitlement model with a guaranteed or desired amount of resource, a shares amount and an optional minimum and maximum amounts (The guaranteed amount of resources is the amount guaranteed to a partition when demanded). It can get the resource from free pool if available and group not over its maximum, take under utilized resource from other partitions or take utilized resource from partitions over their guaranteed resource. The allocated resource will vary between minimum and maximum values defined in

the PLM configuration file. For a partition to be allocated resources above the guaranteed amount, PLM needs to know its share amount (relative priority versus other partitions priority). This amount is a unit less factor between 1 and 255. The formula to calculate the ratio of resource allocated to each partition is (shares of the partition) / (sum of shares from competing partitions).

PLM manages partitions within groups. Each partition must be a member of a group. At least one group must be defined in the PLM policy. One PLM server can manage independent groups of partitions but it cannot share resources across groups. It cannot take unused resources in one group in order to satisfy a demand for resources by another group. The partitions belonging to a group must be of the same type: they are either micro-partitions or dedicated partitions. One group may contain both capped and uncapped partitions. PLM manages the entitled processor capacity, memory, and number of virtual processors for both capped and uncapped partitions.

System administrators must set up PLM partition definitions in a way compatible with the HMC policy definition. The PLM is not able to decrease a partition's minimum below the HMCs minimum nor is the PLM able to increase a partition's maximum over the HMCs maximum. PLM will use HMC partition definition minimum, desired, and maximum partition resource values as PLM minimum, guaranteed, and maximum values if not specified in the PLM policy. If the PLM minimum and maximum values are not within the range defined on the HMC, PLM will use an effective range defined by the intersection of the ranges defined on the HMC and in the PLM configuration file.

9.2.2 Management versus monitoring modes

PLM can execute in 2 modes: Management or Monitoring modes, which are analogous to the WLM active and passive mode.

- ▶ In monitoring mode, PLM receives through RMC the request from partitions for resource reallocation when thresholds are reached. However, PLM does not take any action. PLM will append an entry in its log for each received RMC message.
- ▶ In management mode, PLM will take actions for each RMC message, according to the policies defined in the configuration file.

If you do not have a test environment on which to define the best PLM settings for your environment, if your production environment is critical, or if you do not have a thorough understanding of the partitions workload profile, we recommend you start using PLM in monitoring mode only, with the default configuration values. You can then run it for a significant time duration (one day for example), and then analyze the log, looking for the frequency at which PLM would take actions.

Note: Since PLM does not take actions when in monitoring mode, a request for additional resource will not be satisfied, and the requesting partition will repeat the request until it no longer needs extra resources. When analyzing the log, you should only take into account the resource need changes.

You can also use the **xplstat** command to recognize workload patterns. Once you understand the workload profile of the managed partitions, you can decide of the values to use in the PLM configuration files.

Partition re-configuration is not an instantaneous action, especially for dedicated processor or memory migration between partitions. You may not want to generate such an action to respond to a very short activity peak, when you know that the resources would no longer be needed in the following seconds.

9.2.3 Configuration file and tunables

The system administrator who decides of deploying PLM has only two ways of defining PLM behavior:

1. The values that are set in the PLM configuration file.
2. The arguments that are given to the **xlp1m** command (used to start PLM).

The **xlp1m** command is addressed in 9.3, “Managing and monitoring with PLM” on page 346.

The PLM configuration file is also called policy file in the PLM documentation and on the WebSM panels. You need to create at least one policy file for each CEC on which you want to manage partitions. Policy files are ASCII files with a formal syntax. The policy file can either be created (and modified) through WebSM panels, or by using a text editor (vi, emacs, ...) .

Note: Be careful when you edit the file manually to respect the syntax.

If you start PLM using a policy file with an incorrect syntax, PLM startup will fail. If PLM is already running, and you try to load a new policy with an incorrect syntax, PLM will continue executing with the previously loaded policy.

The policy file contains different variables, called attributes, which are grouped in several sections, each with a different scope: global to one CEC, global for a group of partitions, or specific to one partition. A subset of the attributes are called tunables. They define PLM behavior. Tunables can be set at a system wide level, and optionally overridden for some groups or partitions. Example 9-1

shows a policy for a physical server on which two partitions are managed by PLM.

Example 9-1 PLM policy file

#Example PLM policy file.

```
globals:
    hmc_host_name = p5hmc1
    hmc_user_name = hscroot
    hmc_cec_name = p5Server1
```

```
example:
    type = group
    cpu_type = shared
    cpu_maximum = 2
    mem_maximum = 0
```

```
p5_1test1:
    type = partition
    group = example
    cpu_guaranteed = 0.3
    cpu_maximum = 0.6
    cpu_minimum = 0.1
    cpu_shares = 2
    cpu_load_high = 0.3
    cpu_load_low = 0.2
    cpu_free_unused = yes
```

```
p5_1test3:
    type = partition
    group = example
    cpu_guaranteed = 0.3
    cpu_maximum = 0.5
    cpu_minimum = 0.1
    cpu_shares = 2
```

PLM can read the definition of partition on the HMC using SSH. When starting managing partitions, PLM will read the definition of all partitions to extract default values for these attributes. Then PLM will read the Policy file to override the HMC defined values.

Note: PLM never overwrite a partition profile in the HMC.

Let us discuss some of the attributes.

cpu_minimum, cpu_guaranteed, cpu_maximum**memory_minimum, memory_guaranteed, memory_maximum**

The values are optional. If not present, PLM will access the definition of the partition in the HMC and extract the values of minimum, desired and maximum values for the CPU or memory to set these values.

You can set these attributes to values different from the HMC ones for several reasons.

For example, the HMC minimum value is defined as the minimum amount of resources needed to start a partition. However, you may know that if the partition is only given that amount of resource, it will work in with degraded performance. In this case, you may want to define a PLM `cpu_minimum` with a higher value than the HMC minimum, so that PLM will never make the partition work with bad performance. In other word, the HMC value is the bare minimum of resources needed to run the partition, while the PLM minimum is the lowest reasonable value to get acceptable performance.

Let us take another example. The HMC desired memory is the size of memory a partition will be allocated at boot time, when the system has enough memory resources for all partitions to be started. Once running, the partition may run perfectly well with less memory. For example, an ftp server needs memory to load the files requested by FTP clients, but does not longer need this memory when the files are sent. Since there is no memory clean-up in AIX, the memory used for these files will remain occupied. By setting the `memory_guaranteed` value to the same value as `memory_minimum`, you allow PLM to request the partition to release the memory it no longer needs, to give it to other partitions with real needs for memory.

group

This tunable must appear in each partition stanza. One partition can only belong to one group.

Tip: There is a concept of group in the HMC that can be used when defining the partition. If you plan to use PLM groups, we recommend that you do not use the HMC defined groups

cpu-shares

When PLM is not managing an uncapped partition, the hypervisor allocates unused processor time to uncapped partition according to the current active

weight of the partition, defined by the value of the partition weight as defined on the HMC.

When the uncapped partition is managed by PLM, its current active weight is overridden by the `cpu_shares` value defined on the PLM policy file.

`cpu_shares` define the relative priority of the partitions. Unused resources are allocated to partitions that have their guaranteed (desired) amount or more, in the ratio of their share value to the number of active shares.

Since the default values of the HMC defined weight is 128, and the default value of the `cpu_shares` value is 1, it is important to make sure that all partition within a PLM group use a current active weight set from the same source: HMC definition of PLM policy file.

hmc_command_wait

As mentioned before, the reallocation of resource is not immediate. It takes some time for the HMC to ask a partition to release a cpu, move it from one partition to another, and then tell the target partition to activate the CPU. The `hmc_command_wait` attribute is the delay PLM will wait when asking the HMC to perform an operation, before considering the HMC failed to process the request. If you have already used the DLPAR feature, you can setup this attribute to the value of the Dynamic Re-configuration time out used for DLPAR operations on your system.

cpu_load_low, cpu_load_high

These tunables are the threshold values beyond which PLM decides a partitions has unneeded CPU or not enough CPU. The difference between these two values must be greater than 0.1 (entitlement measured as a fraction of one processor capacity). When reaching such a threshold, dedicated partitions will give or receive one dedicated processors, while micro-partition will give or receive an amount of entitled capacity defined by the `ec_delta` tunable. When adding capacity, `ec_delta` is a percentage of current capacity, while for removing capacity, `ec_delta` is a percentage of the resulting capacity.

The metric for these two tunables is the average number of runnable threads per processor. This is the value you would obtain when dividing the “r” column of the `vmstat` command, or the “runq_sz” column of the `sar -q` command, or the “load average” field of the `uptime` command, by the number of logical processors of the partitions. You can also find this value in the “rq” field of the new AIX 5.3 `mpstat -d` command.

cpu_free_unused, mem_free_unused

These tunables define if unneeded resources are taken from a partition when they are detected as unneeded (when set to yes), or only when another partitions needs them (when set to no).

Immediately returning unused resources to the free pool improves the time to complete a resources allocation to another partition. However, it decreases the performance of the partition that returns the resources if it needs again extra resources later on.

These values should therefore be set to “yes” for partitions that have a seldom use of extra resources and to “no” for partition that have frequent peak of resources utilization.

ec_per_vp_min, ec_per_vp_max

These are the two tunables to use to have PLM automatically change the number of virtual processors of a micro-partition. PLM does not directly handle the number of virtual processors of micro-partitions. Virtual processor addition and removal is triggered by capacity changes. When entitlement must be added or removed, if the reallocation of entitled capacity results in crossing the `ec_per_vp_min` or `ec_per_vp_max` threshold, then virtual processors are also added or removed from the partition.

mem_util_low, mem_util_high, mem_pgstl_high

These tunables are the threshold values beyond which PLM decides a partitions has unneeded or not enough memory.

To be considered as a memory requester, the partition must reach **both** `mem_util_high` and `mem_pgstl_high`. There is no lower limit for the page steal rate.

The unit used to measure `mem_util_low` and `mem_util_high` are is a percentage of memory. The memory utilization of the partition is defined as:

$$\text{pct} = ((\text{memory pages} - \text{free pages}) / \text{memory pages}) * 100$$

where `memory pages` and `free pages` are the values returned by the `vmstat -v` command.

The unit used to measure `mem_pgstl_high` is a number of page steal per second. The value compared to the `mem_pgstl_high` threshold is the value returned in the “fr” field of the `vmstat` command.

9.3 Managing and monitoring with PLM

There is a single point of control to configure, manage and operate PLM: the PLM manager. The PLM policy and log files are stored on the PLM server. The PLM commands can only be used on the PLM server. There are no PLM commands that can be used from the HMC or the PLM managed partitions.

PLM can be operated in two ways:

1. The classical Unix way, using the AIX command line interface to edit the policy files, browse the logs, start and stop PLM, ...
2. The GUI way using WebSM to access the PLM server.

Note: There is no SMIT panel to configure and operate PLM.

Operating PLM is very simple, since PLM has only two commands:

| | |
|----------------|---|
| x1plm | Used to start, stop, modify or query PLM. |
| x1pstat | Used to display statistics about the managed partitions |

WebSM provides the equivalent functions in a graphical environment.

9.3.1 Managing multiple partitions CEC from the same PLM manager.

We have mentioned that one PLM policy file controls partitions that must all run on the same CEC, and that one PLM manager can manage several partitions executing onto different physical servers. Let us be more accurate on how this works.

We will call:

| | |
|---------------------|--|
| PLM server | the AIX instance in which the PLM code has been installed. |
| PLM server instance | the server part of the PLM client/server relationship: the clients are the managed partitions. |

Note: Each PLM server instance executes under AIX as an **x1plmd** daemon.

On the PLM server, you can start several PLM server instances, by using the **x1plm** command several times, with different Policy files.

One PLM server instance executes only one policy file at a time, and therefore manages only partitions within the same CEC.

The way to start a PLM server instance is to use the `xlplm` command with the `-S` (Start) argument, and the name of a policy file:

```
xlplm -S -p policy_file
```

This command start a server instance which will be named default. If you want to start multiple PLM server instances from the same PLM server, you need to give a name to each PLM server instance:

```
xlplm -S -p policy_file server_instance_name
```

Each policy file starts with a global stanza containing the hostname of the HMC that manages a CEC and the name of the CEC as seen from the HMC. Furthermore, for each set of managed partitions, you may want to use different set of PLM policies (for example to change the PLM behavior with the time of day). A good practice is therefore to create policy filenames and PLM server names according to mnemonic naming conventions.

An simple example would be:

```
<Server_Instance_Name> =:: <HMC_shortcode><CEC_name>  
<Policy_Name> =:: <Server_Instance><Suffix>
```

so that if you type 2 commands in a row like:

```
xlplm -S -p hmc1cec2policy3 hmc1cec2  
xlplm -S -p hmc2cec1policy4 hmc2cec1
```

you know that you have applied the third policy defined for the second CEC managed by your first HMC, and the fourth policy defined for the first CEC managed by the second HMC.

In the same way, you should use a meaningful name for the log files. There is one log file per PLM server instance.

You can start two PLM server instances to manage partitions that are hosted on the same CEC. This can be useful for examples when two different system administration teams manage different sets of partitions on one system. Each team can have the full control of its own PLM policy file.

Using the `xlplm` command with the `-Q` (Query) will display the policy used for each PLM server controlled from one PLM Manager.

```
xlplm -Q                                shows all active instance names.
```

```
xlplm -Q server_instance_name          shows all details for one instance.
```

9.3.2 Extra tips about the xlp1m command

The **xlp1m** commands has multiple arguments and flags. Please refer to the official documentation for an exhaustive explanation of this command.

Let us just give a few tips.

-C the -C (Check) flag is used to verify the syntax of a policy file. It is useful when the file has been edited manually, rather than through WebSM. The -C flag does not check the policy file tunables consistency versus the values defined in the HMC.

-M The -M (Modify) flag enables the system administrator to dynamically load a new policy, to ask for writing the log in a different logfile, or to switch from monitoring to managing mode. This flag can be used when the **xlp1m** command is called by a **crontab** for example, when time-based policies must be used.

The Modify flag can be used to change resources groups. For example, if a system is intensively used during daytime and all partitions are competing for resources, it can be a good practice to define multiple groups and to gather partitions by workload type, so that each partitions is ensure to have a fair share of the resources. This will improve the overall system throughput. However if the system is under-used at night-time, optimizing the throughput is no longer a critical issue. It can then be better to gather all partitions in one large group, so that in the case one partition has a burst of activity, it can use all available resources. In this case, the policy help improving the response time of one partition rather than the overall throughput.

Author Comment: What are reserved resources? the official doc is not sufficient for these options. The -f options is not documented in the official doc.. Still old -a

The -R option need not be described here It could be left for description in the oncoming PLM redbook.

-R what are reserved resource?, Is that to override values in the config file without reloading a new config file? when would you use this?

- f

By default, the -Q (query) option displays for the min and max values the intersection of the range defined in the HMC and in the PLM policy file. When used with the -f flag, the command returns the values defined in the PLM policy file.
- v

The -v (Verbose) option when used with the -Q option provides extra information. In particular, this is the option to use to check if the policy is used in management or monitoring mode, and to see the thresholds set for each partition and group.

9.3.3 Examples of PLM commands output

This section shows some examples of PLM command output. Example 9-2 displays the output of the `x1plm` query command when used in command line mode.

Example 9-2 `x1plm query`

```
# x1plm -v -Q benchtest
PLM Instance: benchtest

      CEC Name      Server-9117-570-SN105428C
      Mode          manage
      Policy        /etc/plm/policies/testBenchmark
      Log           /var/test.out
      HMC Host      isvlab064.austin.ibm.com
      HMC User      hscroot

GROUP: Benchmark1

      CPU:      CUR      MAX      AVAIL      RESVD      MNGD
      MEM:      14336      0      0      0      No

      CPU TYPE: shared

      basil.austin.ibm.com

      RESOURCES:

      CPU:      CUR      MIN      GUAR      MAX      SHR
      MEM:      7168      1024      7168      15360      1

      TUNABLES:

      CPU:      INTVL      FRUNSD      LOADLO      LOADHI      DELTA      PGSTL
      MEM:      6      0      50%      90%      16      0
```

| | | | | | | |
|---------------------|-------|--------|--------|--------|-------|-------|
| | | MIN | MAX | | | |
| PER VP CAP: | | 0.74 | 0.90 | | | |
| sage.austin.ibm.com | | | | | | |
| RESOURCES: | | | | | | |
| | CUR | MIN | GUAR | MAX | SHR | |
| CPU: | 1.04 | 0.10 | 1.00 | 4.00 | 128 | |
| MEM: | 7168 | 1024 | 7168 | 15360 | 1 | |
| TUNABLES: | | | | | | |
| | INTVL | FRUNSD | LOADLO | LOADHI | DELTA | PGSTL |
| CPU: | 3 | 1 | 0.75 | 1.00 | 20% | - |
| MEM: | 6 | 0 | 50% | 90% | 16 | 0 |
| | | MIN | MAX | | | |
| PER VP CAP: | | 0.74 | 0.90 | | | |
| # | | | | | | |

You can also use Web System Management tool (WebSM) to manage PLM.

Figure 9-1, 9-2 and 9-3 shows how to get the same information using WebSM. The first figure shows where to find PLM in the webSM navigation area, and the other figures show respectively the partitions processor statistics and memory statistics.

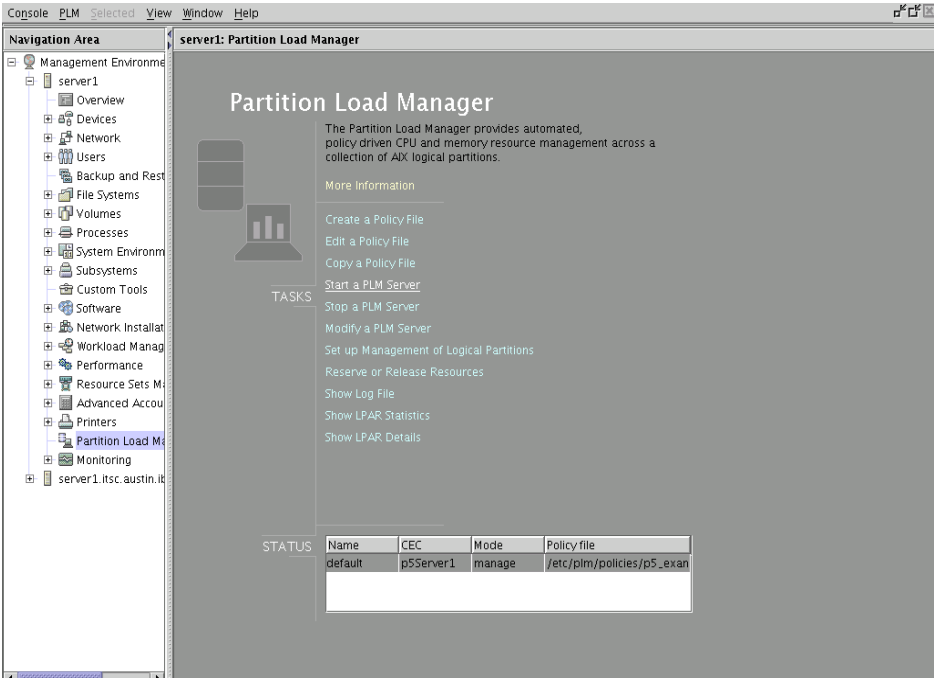


Figure 9-1 PLM management using WebSM

| CPU Statistics | | Memory Statistics | | | | | | |
|-----------------|---------|-------------------|---------|-------|---------|-------|--------------|--|
| Name | Minimum | Guaranteed | Maximum | Share | Current | Use % | Load average | |
| p5Server1 | | | 2.00 | | | | | |
| --- example | | | 2.00 | | | | | |
| ----- p5_1test1 | 0.00 | 0.00 | 0.00 | 2 | 0.00 | 0.00 | 0.00 | |
| ----- p5_1test3 | 0.10 | 0.30 | 0.50 | 2 | 0.50 | 60.80 | 0.09 | |

Close

Refresh

Help

Figure 9-2 PLM processor statistics using WebSM

| CPU Statistics | | Memory Statistics | | | | | |
|-----------------|---------|-------------------|---------|-------|---------|-------|-----------|
| Name | Minimum | Guaranteed | Maximum | Share | Current | Use % | Pagesteal |
| p5Server1 | | | 0 | | | | |
| --- example | | | 0 | | | | |
| ----- p5_1test1 | 0 | 0 | 0 | 1 | 0 | 0.00 | 0 |
| ----- p5_1test3 | 512 | 1024 | 2048 | 1 | 1024 | 32.46 | 0 |

Close

Refresh

Help

Figure 9-3 PLM memory statistics using WebSM

9.4 PLM performance impact

The performance aspect of PLM is twofold: the resource requirement needed to run the PLM server, and the impact of PLM actions on managed partitions performance.

9.4.1 PLM resource requirements

On the server, the PLM function is provided by the xlplmd daemon. We present here some measurement of the resources requirements for this daemon.

First, we set up an environment where PLM manages 2 partitions and configure it so that PLM receives 3 RMC events every minute. Example 9-3 shows how you can use the `tail` command to monitor the PLM log and see these events.

Example 9-3 Checking the PLM log

```
# tail -f test.out
<04/06/70 17:38:40> <PLM_TRC> Event notification of CPUZone low for Benchmark1 .
<04/06/70 17:39:05> <PLM_TRC> Event notification of CPUZone low for Benchmark2 .
<04/06/70 17:39:20> <PLM_TRC> Event notification of CPUZone low for Benchmark1 .
<04/06/70 17:39:34> <PLM_TRC> Event notification of CPUZone low for Benchmark2 .
<04/06/70 17:40:00> <PLM_TRC> Event notification of CPUZone low for Benchmark1 .
<04/06/70 17:40:14> <PLM_TRC> Event notification of CPUZone low for Benchmark2 .
<04/06/70 17:40:40> <PLM_TRC> Event notification of CPUZone low for Benchmark1 .
<04/06/70 17:40:54> <PLM_TRC> Event notification of CPUZone low for Benchmark2 .
```

In a stable environment, the frequency of these events would be much lower.

Under these conditions, PLM uses approximately 1.4 Megs of memory, as shown in Example 9-4: see column *RSS* displaying the size (in 1Kbytes units) of the real memory used by **x1plmd**.

Example 9-4 Memory utilization for the PLM daemon

```
# ps xvg 344232
      PID  TTY STAT  TIME PGIN  SIZE   RSS   LIM  TSIZ  TRS %CPU %MEM COMMAND
344232    -  A    0:00   66  1288  1396   xx   78   120  0.0  0.0 x1plmd
b
#
```

We also use the **ps** command to measure the amount of processing resources used by the PLM daemon. Example 9-5 shows that after managing partitions for one and a half hour, PLM has only used 11 seconds of CPU time. This example also shows that the percentage of memory and CPU used by the daemon is not measurable: percentage equal to zero.

Example 9-5 Processing resources needed by PLM

```
# ps -o uid,pid,pmem,etime,time,pcpu,comm -p 405676
UID    PID  %MEM    ELAPSED      TIME  %CPU COMMAND
  0 405676   0.0    01:29:36    00:00:11   0.0 x1plmd
#
```

Furthermore, the PLM daemon does not generate any significant disk or network activity.

Conclusion

The resources required to run the PLM daemon are not a criteria for deciding where to instantiate the PLM server. It can run on any existing AIX system with no visible impact on this system performance.

9.4.2 PLM impact on managed partitions

Besides the PLM commands described in Section 9.3, “Managing and monitoring with PLM”, one way of visualizing the effect of PLM is to use the AIX performance toolbox (PTX)

Figure 9-4 shows a PTX screen, that was taken while CPU intensive programs were executed. A 4-way 5-570 is split in two uncapped micro-partitions, called Basil and Sage. Sage is the partition where a benchmark program is run (the benchmark partition). This benchmark consists of many processes running in parallel, so that it can take advantage of as many logical processors as are given to the partition. Basil is the monitoring partition. It runs the PLM server and the

PTX management program (**xmperf**). It is also loaded with dummy programs to show the impact on the benchmark partition of varying workload in the other partitions.

The screen is divided in three parts (also called “instruments” in the PTX terminology):

- ▶ The top part shows the workload of the monitoring partition. It contains two lines.
 - The blue line displays the run queue.
 - The red line displays the number of virtual processors.

Note: To help the reader of the paper copy of this redbook which is printed in black and white, we have annotated the figure. The arrow with the circled A points to the blue line, and the arrow with the circled B points to the red line. The blue line presents many peaks, while the red line is looking like stairs.

- ▶ The middle part shows the workload of the benchmark partition. It also displays the run queue size and the number of virtual processors, using the same graphical conventions than for the top part.
- ▶ The bottom part shows the cpu utilization of the benchmark partitions. It shows the percentage of CPU time spent in user, kernel, wait or idle mode.

In each part, the X-axis represents time, with a scale using a 24h base time representation.

The figure demonstrate that PLM will adjust the number of Virtual processors allocated to a partition, according to its resources needed and to the amount of available resources in the server. PLM also adjust the cpu entitlement, but for the sake of clarity of the diagram, we have chosen to only plot two values (run queue size and number of virtual processors):

- ▶ Initially, both partitions only have one processor, since this is the minimum as defined in the PLM policy, and there's no activity.



Figure 9-4 PLM impact on number of virtual processors.

- ▶ Shortly before 14h58, the benchmark is started on Sage, while Basil is nearly idle. PLM notices the need for extra resources of Sage, and gradually increase its number of allocated virtual processors up to the maximum (4), so it can use all resources in the server.
- ▶ Around 15h00, the workload on Basil increases (due to starting some CPU intensive programs).

- ▶ Around 15h01, PLM notices the resources need of Basil is persistent, and start increasing its entitlement and number of virtual processors: set to 2.
- ▶ At the same time, since all resources in the system are used, PLM decrease the entitlement of Sage, and since the ratio of entitlement per processor falls below the `ec_per_vp_min` threshold, PLM removes one virtual processor from Sage.
- ▶ At 15h02, the system is stable. Both partitions are running CPU intensive applications. Since they are defined with the same thresholds, tunables and priority in the policy file, PLM allocates 2 virtual processors to each of the partitions.

10



Applications Tuning

This chapter provides an introduction for Fortran and C/C++ programmers or users that are interested in tuning their applications for POWER5. Although this chapter assumes that the reader has a working knowledge of these concepts, there are sections that are very well suited for those of you that are beginning to measure applications performance and would like to start improving the performance of a particular application. Also, it is important to point out that for POWER3 and POWER4 an entire book was dedicated to providing a tuning guide. Here much of that work has been condensed to a single chapter. Therefore, we make reference to those books as much as possible to reserve this chapter issues that are related to POWER5. In this chapter we cover the following major topics:

- ▶ Identification of the type of bottleneck and location within the code
- ▶ Tuning using compiler flags
- ▶ Profiling the code to uncover performance bottlenecks
- ▶ General tuning for single processor performance
- ▶ Making use of highly optimized libraries
- ▶ General tuning for parallel performance

10.1 Performance bottlenecks identification

This section gives an overview of the basic steps that are required to localize performance bottlenecks. In this chapter we assume that performance limitations are not hardware related but the performance constraints are a function of how an application was coded. The following defines what we consider in this book performance bottlenecks and code optimization:

Performance bottlenecks:

Sections of a code that tend to consume most of the user time and/or elapsed time (also referred as real time or wall-clock time) and after careful analysis require code optimization.

Also, in this book we define code optimization, user time, elapsed time, and system time as follows:

Code optimization: A series of steps required to modify a section or sections of the code, manually or via the compiler, to improve performance.

Elapsed time: The time that it took the program to run from the beginning to end. This is the sum of all of the factors that can delay the program, plus program's own attributed costs.

User time: This is the time used by itself and any library routine that it calls while it is attached to a processor.

System time: The time used by system calls invoked by the program, directly or indirectly.

In the above definition, optimization falls into two categories:

Code optimization using advanced compiler flags:

In this type of optimization we simply rely on how much performance we can gain by selecting the best combination of compiler flags for a particular application

Hand-tuning: In this type of optimization it is required to manually modify the code to improve performance. In some extreme case it might be required to use different methodology

The following diagram illustrates the basic steps that are required to localize performance bottlenecks.

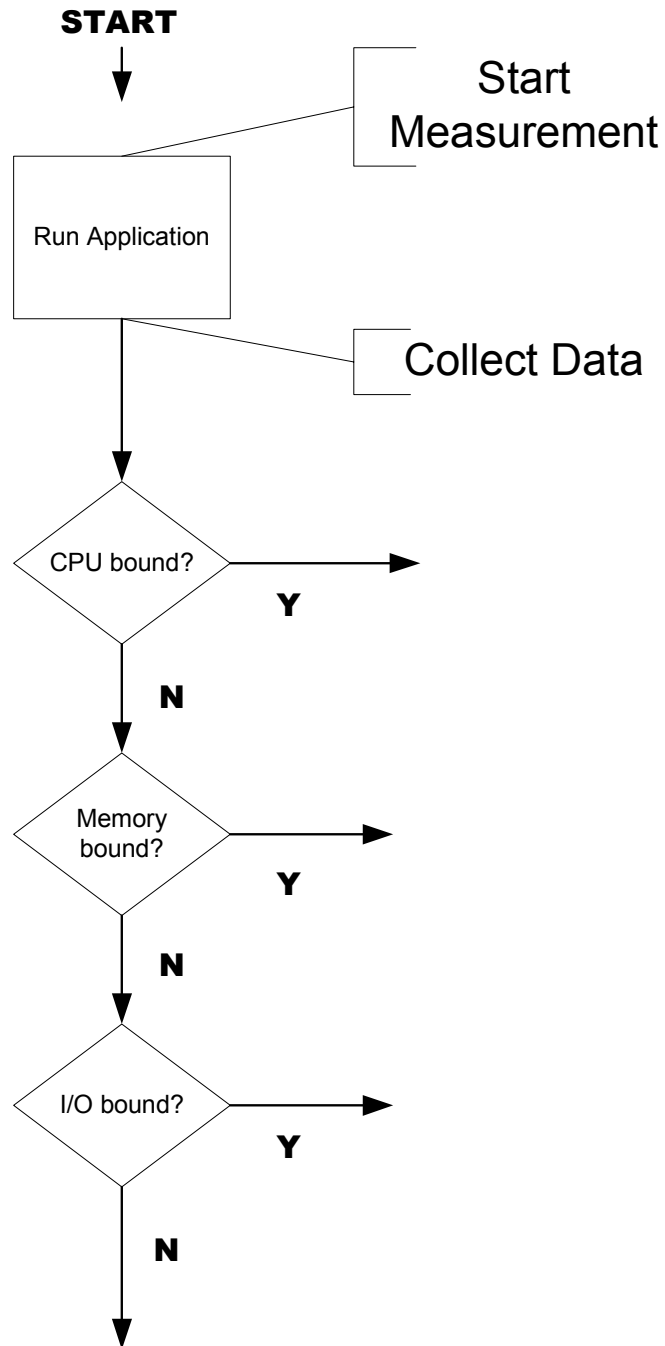


Figure 10-1 This flowchart illustrates the first step in applications tuning.

The first step in the above diagram requires running the application and collecting data from some sort of “measurement”. The simplest way to start analyzing an application is by looking at timers. It does not require recompiling nor modifying the code. The most common timing routines can be invoked at run time. Of course, carrying out timing of certain routines inside the code requires recompiling and building the program. These two techniques will be covered in the next section.

10.1.1 Time commands, time utilities and time routines

In this section we describe commands, utilities, and routines that may be used to identify bottlenecks in the application. The set described here is far from being the only set of tools available for performing a coarse analysis but we have found them to be fairly common and easy to use. The commands described here are:

- ▶ **time**
- ▶ **timex**
- ▶ **vmstat**
- ▶ **irtc**
- ▶ **rtc**

The first and simplest set of commands are **time** and **timex**. They both print elapsed time, user time, and system time of a command during execution in seconds. In their simplest form they can be invoked as follows:

```
$/usr/bin/timex a.out
```

or

```
$/usr/bin/time a.out
```

A version of **time** that produces additional information corresponds to the C shell built-in command. The following example illustrates the additional information obtained with this built-in version:

```
%time a.out
```

```
%5.1u 0.1s 0:05 98% 137+91548k 0+0io 27pf+0w
```

Table 10-1 illustrates the meaning of the different fields from **time** C shell built-in version.

Table 10-1 time description fields.

| Fields | Description |
|-------------|---|
| 5.1 u | Number of seconds of user time |
| 0.1 s | Number of seconds of user time consumed by system calls invoked by the program |
| 0:05 | Elapsed time |
| 98% | Total user time plus system time, as a percentage of elapsed time |
| 137+91548 k | Average amount of shared memory used, plus average amount of unshared data space used, in kilobytes |
| 0+0 io | Number of blocks input and output operations |
| 27pf+0w | Page faults plus number of swaps |

Prior to provide empirical rules to classify the performance of a particular application, let us proceed to define what CPU-bound, Memory-bound, and I/O-bound applications are:

| | |
|--------------|--|
| CPU-bound | If sections of the code dominate most of the run time by performing processor calculations, then it is said that the application is CPU-bound. |
| Memory-bound | If sections of the code dominate most of the run time by memory issues or memory limitations, then the application is considered memory-bound. |
| I/O-bound | If sections of the code dominate most of the run by performing I/O, then it is said that the application is I/O-bound. |

The output provided with this simple commands can give an initial indication on the type of bottleneck in our particular application. *The following empirical rules should be viewed as a guidance for this classification:*

- ▶ An excessive large user time can be an indication of a CPU-bound application that might not be running optimally and requires tuning.
- ▶ The ratio between elapsed time and the user time ($r_{WIO} = \text{elapsed time} / \text{user time}$) may provide an indication of an I/O-bound application. A ratio larger than 1 represents an unbalance between elapsed time and user time. For certain cases this may be interpreted as a large I/O wait time. A ratio large than 2.5 is for us an empirical threshold that the I/O performance needs to be

considered. In addition, the C shell built-in time function provides information about the number of blocks input and output which can be an indication of the amount of I/O that an application is performing for a particular run.

- ▶ In general, user time tends to be larger, by at least an order of magnitude, than the system time. A large system time could be attributed to a memory-bound program. If page faults plus number of swaps and the average amount of shared memory used is large, this may help consolidate any suspicion of being a memory-bound code.

In this chapter we briefly mention **vmstat** as an alternative way to get similar information as in the case of **time** or **timex**. However, an extensive discussion can be found in Chapter 8. On a system, **vmstat** can provide statistics about kernel threads, virtual memory, disks, and CPU capacity. These systemwide statistics are calculated as averages for values reported as percentages or sums. Since this is a systemwide resource, to analyze just one application a dedicated system is required.

Attention: vmstat provides systemwide statistics.

Example 10-1 illustrates an example where a small program that performs a matrix multiplication is monitored using **vmstat**.

Example 10-1 Example invoking vmstat

\$ matmul.F2000

\$ vmstat 1 10

| kthr | | memory | | | | page | | | | faults | | | | cpu | | | |
|------|---|---------|---------|----|----|------|----|----|----|--------|-----|------|-----|-----|----|----|---|
| r | b | avm | fre | re | pi | po | fr | sr | cy | in | sy | cs | us | sy | id | wa | |
| 1 | 1 | 2533495 | 1516953 | | 0 | 0 | 0 | 0 | 0 | 0 | 445 | 1249 | 336 | 0 | 0 | 99 | 0 |
| 0 | 0 | 2533499 | 1516949 | | 0 | 0 | 0 | 0 | 0 | 0 | 447 | 3397 | 336 | 0 | 0 | 99 | 0 |
| 0 | 0 | 2533499 | 1516949 | | 0 | 0 | 0 | 0 | 0 | 0 | 445 | 3234 | 329 | 0 | 0 | 99 | 0 |
| 0 | 0 | 2533499 | 1516949 | | 0 | 0 | 0 | 0 | 0 | 0 | 444 | 3246 | 335 | 0 | 0 | 99 | 0 |
| 1 | 0 | 2556989 | 1493459 | | 0 | 0 | 0 | 0 | 0 | 0 | 449 | 3301 | 351 | 16 | 3 | 81 | 0 |
| 1 | 0 | 2556989 | 1493459 | | 0 | 0 | 0 | 0 | 0 | 0 | 444 | 3198 | 338 | 25 | 0 | 75 | 0 |
| 1 | 0 | 2556989 | 1493459 | | 0 | 0 | 0 | 0 | 0 | 0 | 447 | 3267 | 356 | 25 | 0 | 75 | 0 |
| 7 | 0 | 2556989 | 1493459 | | 0 | 0 | 0 | 0 | 0 | 0 | 446 | 3227 | 340 | 25 | 0 | 75 | 0 |
| 2 | 0 | 2556993 | 1493455 | | 0 | 0 | 0 | 0 | 0 | 0 | 448 | 3297 | 347 | 25 | 0 | 75 | 0 |
| 2 | 0 | 2556993 | 1493455 | | 0 | 0 | 0 | 0 | 0 | 0 | 446 | 3210 | 338 | 25 | 0 | 75 | 0 |

Finally, if fine grain timing is required, we provide templates for the use **irtc()** and **rtc()**. These two functions require code modification and recompilation.

irtc

The irtc function returns the number of nanoseconds since the initial value of the machine's real-time clock.

Example 10-2 irtc() template.

```
template for irtc()
```

```
integer(8) T1, T2, IRTC
```

```
T1 = IRTC()
```

```
Your section of the code
```

```
T2 = IRTC()
```

```
write(*,*) 'Untuned loop took', (T2-T1)/1000000, 'msec'
```

```
end
```

rtc

The rtc function returns the number of seconds since the initial value of the machine's real-time clock.

Example 10-3 rtc() template.

```
template for rtc()
```

```
real*8 T1, T2, RTC
```

```
T1 = RTC()
```

```
Your section of the code
```

```
T2 = RTC()
```

```
write(*,*) 'Untuned loop took', (T2-T1), 'sec'
```

```
end
```

10.2 Tuning applications using only the compiler

In this section we provide an overview of the compiler, selected features of the XL Fortran and XL C and C++ compiler that relates to the optimization of applications running on POWER5 processors, and we examine some of the compiler flags that are relevant for scientific and engineering applications. The emphasis in this section is to illustrate the compiler capabilities to carry out code optimization. This is what we consider the first step in the process of code optimization. However, for many programmers that are not interested in optimizing their applications further, this step usually turns out to be the last step. However, this is an important section for all programmers.

10.2.1 Compiler brief overview

Compiler technology represents a formidable challenge, in particular the development of an optimizer that can tune any code.

The Toronto Portable Optimizer (TPO) is designed to operate on many source languages and generate code for many target platforms. Figure 10-1 shows that TPO is a key component in the overall compiler architecture.

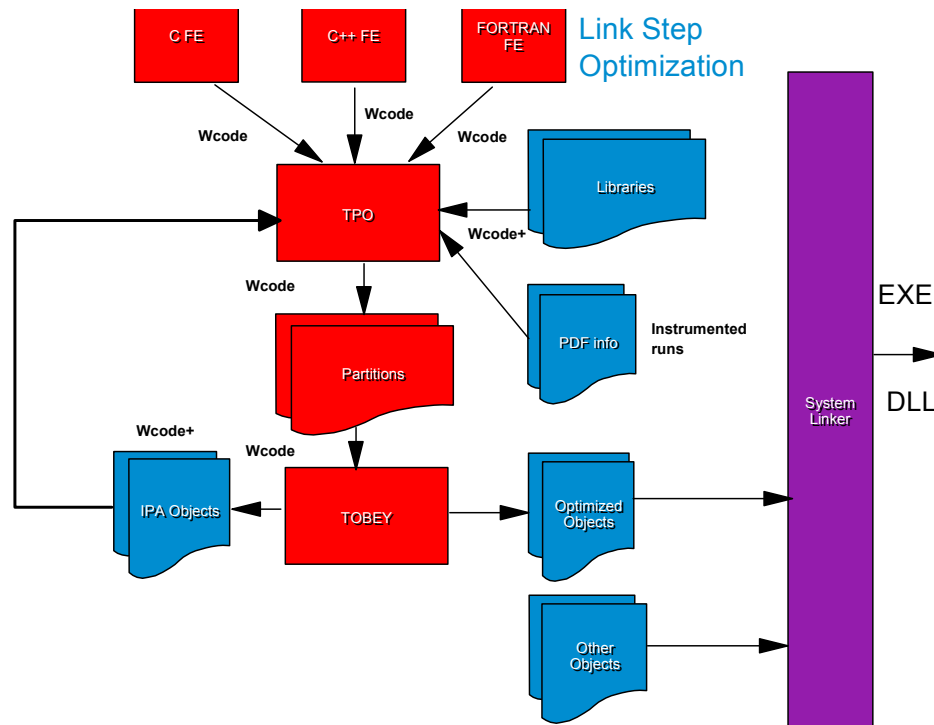


Figure 10-2 IBM compiler architecture.

Figure 10-2 goes deeper into the compiler architecture and shows the steps that TPO goes through to optimize codes.

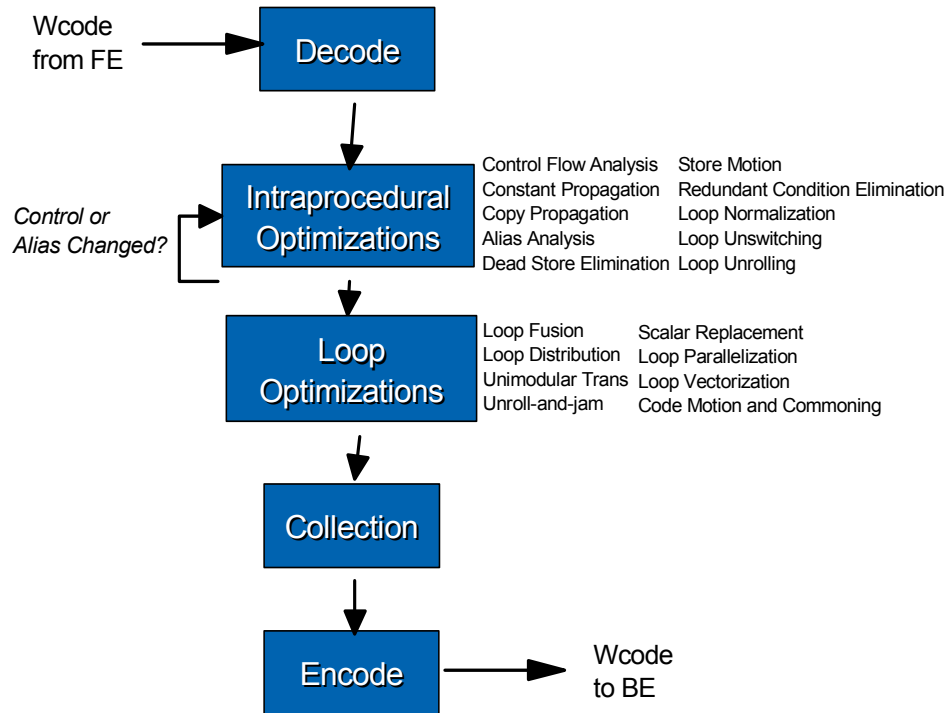


Figure 10-3 Inside TPO compile time.

The details of loop optimization are summarized in Figure 10-3.

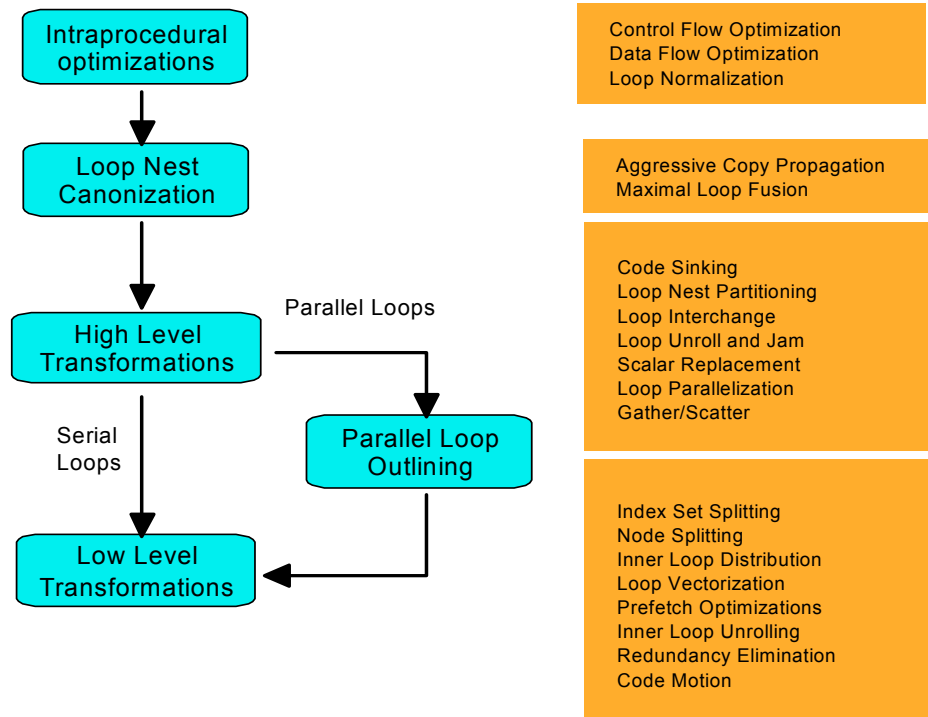


Figure 10-4 Loop optimization overview in TPO.

10.2.2 Most commonly used flags

In this section we look at the compiler flags that affect the performance of an application. This is not an exhaustive list, for a comprehensive list, visit the IBM AIX Compiler information center. *It is important to always check answers as you increase the level of compiler optimization or aggressivity.* This is due to the fact that the compiler makes certain assumptions about some of the statements in the code that can potentially be optimized by re-writing that section of the code.

A typical example is the property of associativity in a product, at low levels of compiler optimization, e.g. -O2, XL Fortran will evaluate $a*b*c$ always starting from a , even if $b*c$ has already being computed. Although more time will be consumed, it is safer since the answer might be dependent on the order of execution. As the level of optimization increases, some of these restrictions might be eliminated.

Optimization level

A few basic rules to remember when using the compiler for optimization:

- ▶ Optimization requires additional compilation time
- ▶ Optimization produces faster code; but always check answers, specially when using aggressive levels of compiler optimization
- ▶ By default, the compiler chooses -O0 or -qnoot
- ▶ Enable compiler optimization with -O[N]; where N is 0, 2, 3, 4, or 5
 - Example: `$x1f -O3`

Next, we discuss the different levels of compiler optimization, not to provide an exhaustive list of flags but to convey information about the effects on the so called performance flags on scientific and engineering applications.

Level 0: -O0

This option is recommended for debugging. It is the fastest way to compile the program. *It preserves program semantics.* This is also useful to see the effect of hand tuning small kernels or certain Do loops.

Level 2: -O2

This is the same as -O. At this level the compiler performs conservative optimization. The optimization techniques used at this level are:

- ▶ Global assignment of user variables to registers, also known as graph coloring register allocation.
- ▶ Strength reduction and effective use of addressing modes.
- ▶ Elimination of redundant instructions, also known as common subexpression elimination
- ▶ Elimination of instructions whose results are unused or that cannot be reached by a specified control flow, also known as dead code elimination.
- ▶ Value numbering (algebraic simplification).
- ▶ Movement of invariant code out of loops.
- ▶ Compile-time evaluation of constant expressions, also known as constant propagation.
- ▶ Control flow simplification.
- ▶ Instruction scheduling (reordering) for the target machine.
- ▶ Loop unrolling and software pipelining.

Level 3: -O3

At this level the compiler performs more extensive optimization. This includes:

- ▶ Deeper inner loop unrolling.

- ▶ Better loop scheduling.
- ▶ Increased optimization scope, typically to encompass a whole procedure.
- ▶ Specialized optimizations (those that might not help all programs).
- ▶ Optimizations that require large amounts of compile time or space.
- ▶ Implicit memory usage limits are eliminated (equivalent to compiling with `qmaxmem=-1`).
- ▶ Implies `-qnostrict`, which allows some reordering of floating-point computations and potential exceptions.

Important: At this level `-qnostrict` is invoked by default. This implies:

- ▶ Reordering of floating-point computations.
- ▶ Reordering or elimination of possible exceptions (e.g., division by zero, overflow)

Level 4: -O4

At this level the compiler introduces more aggressive optimization. This option includes:

- ▶ Includes `-O3`
- ▶ `-qhot`
- ▶ `-qipa`
- ▶ `-qarch=auto`
- ▶ `-qtune=auto`
- ▶ `-qcache=auto`

Level 5: -O5

At this level the compiler introduces more aggressive optimization. This option includes:

- ▶ Includes `-O4`
- ▶ `-qipa=level=2`

Important: If `-O5` is specified on the compile step, then it should be specified on the link step.

Machine specific flags

This set of flags are specific for a family architecture. The idea is to provide code that is optimized for a particular architecture.

Important: By default, the compiler generates code that runs on all supported systems, however, it might not be optimized for a particular system.

| Option | Description |
|---------|--|
| -q32 | For 32-bit execution mode. |
| -q64 | For 64-bit execution mode. |
| -qarch | Selects specific architecture for which instruction is generated |
| -qtune | Biases optimization toward execution on a given processor, without implying anything about the instruction set architecture to use as a target |
| -qcache | Defines a specific cache or memory |

High-order transformations

-qhot optimization is targeted to improve the performance of loops and array language. This may include:

- ▶ Loop interchange
- ▶ Loop fusion
- ▶ Loop unrolling
- ▶ Reducing the generation of temporary arrays

The goals of high-order transformation are:

- ▶ Reducing the costs of memory access through the effective use of caches and translation look-aside buffers.
- ▶ Overlapping computation and memory access through effective utilization of the data prefetching capabilities provided by the hardware
- ▶ Improving the utilization of processor resources through reordering and balancing the usage instructions with complementary resource requirements.

Interprocedural analysis

-qipa allows the compiler to perform optimization across different files. In other words, it provides analysis for the entire program. The interprocedural analysis has the following suboptions:

Table 10-2 -qipa suboptions.

| Suboption | Description |
|-----------|--|
| level=0 | Automatic recognition of standard libraries. |
| | Localization of statistically bound variables and procedures. |
| | Partitioning and layout of procedures according to their calling relationships, which is also referred to as their call affinity. |
| | Expansion of scope for some optimizations, specially register allocation. |
| level=1 | Procedure inlining. |
| | Partitioning and layout of static data according to reference affinity. |
| level=2 | Whole-program alias analysis. This level includes the disambiguation of pointer dereferences and indirect function calls, and the refinement of information about the side effects of a function call. |
| | Intensive interprocedural optimizations. This can take the form of value numbering, code propagation and simplification, code motion into conditions or out of loops, elimination of redundancy. |
| | Interprocedural constant propagation, dead code elimination, pointer analysis. |
| | Procedure specialization. |

XL Fortran new and changed functionality

Some features have been added or improved in the XL Fortran compiler. Here we provide a brief overview of this new functionality.

Table 10-3 New or changed options and suboptions

| Options and suboptions | Comments |
|------------------------|---|
| -qfltrap=nanq | The suboption detects all NaN values handled or generated by floating point instructions, including those not created by invalid operations. |
| -qport=nullarg | The suboption treats an empty argument, which is delimited by a left parenthesis and a comma, two commas, or a comma and a right parenthesis, as a null argument. |
| -qmodule=mangle81 | The option provides compatibility with Version 8.1 module naming conventions for non-intrinsic modules. |
| -qsaveopt | The option saves the command-line options used for compiling a source file in the corresponding object file. |
| -qversion | The option provides the version and release for the invoking compiler. |

In addition, with the XL Fortran 9.1 compiler new options and suboptions that affect performance have been added. Table 10-4 summarizes these newly added options and suboptions. Some of the options presented in this table will be discussed in more detail in other sections.

Table 10-4

| Option/Suboption | Description |
|--------------------------------|---|
| -qarch and -qtune | These two options now provide support for POWER5 and PowerPC 970 architectures (pwr5 and ppc970) |
| -qshowpdf and -qpdf1 | Provide additional call and block count profiling information to an executable |
| showpdf and mergepdf utilities | Provide enhanced information about PDF-directed compilation |
| -qdirecstorage | Informs the compiler that a given compilation unit may reference write-through-enabled or cache-inhibited storage |

| Option/Suboption | Description |
|----------------------------------|--|
| SWDIV and SWDIV_NOCHK intrinsics | Provide software floating-point division algorithms |
| FRE and FRSQRTES intrinsic | Floating-point reciprocal estimate and floating-point square root reciprocal |
| POPCNT and POPCNTB intrinsics | Provide set bit counts in registers for data bojects |
| POPPAR intrinsic | Determines the parity for a data object |

10.2.3 Compiler directives for performance

Once the compiler flags have been optimized, the programmer can still use highly optimized libraries and/or compiler directives to improve performance without major changes to the code. The use of highly optimized optimized libraries will be covered later. Int his section we only mention compiler directives. In particular, we look at directives for code tuning and hardware-specific directives that potentially can help improving performance.

To identify a sequence of characters, called triggers constants, XL Fortran uses the -qdirective option:

Note: directive flag

-qdirective [=directive_list] | -qnodirective [=directive_list]

The compiler recognizes the default trigger constant **IBM***. Table 10-4 provides a list of assertive, loop optimization and hardware-specific directives.

Table 10-5 Assertive, loop optimization and hardware-specific directives.

| Directive | Type | Description |
|-----------------|-------------------|---|
| ASSERT | Assertive | Provides characteristics of do loops for further optimization; requires -qsmp or -qhot |
| CNCALL | | Declares that no loop-carried dependencies exist within any procedure called from the Do loop; requires -qsmp or -qhot |
| INDEPENDENT | | Must precede a Do loop, FORALL statement; it specifies that the loop can be executed and iterations in any order without affecting semantics; requires -qsmp or -qhot |
| PERMUTATION | | Specifies that the elements of each array listed in the integer_array_name_list have no repeated values; requires -qsmp or -qhot |
| BLOCK_LOOP | Loop optimization | Allows blocking inside nested loops |
| LOOPID | | Allows the assignment of unique identifier to loop within a scoping unit |
| STREAM_UNROLL | | Allows for a combination of software prefetch and loop unrolling; requires -qhot, -qipa=level=2, or -qsmp, and -O4 |
| UNROLL | | Allows loop unrolling where applicable |
| UNROLL_AND_FUSE | | Allows loop unrolling and fuse where applicable |

| Directive | Type | Description |
|-----------------------------|-------------------|--|
| CACHE_ZERO | Hardware-specific | Invokes machine instruction dcbz; sets the data cache block corresponding to the variable specified to zero |
| ISYNC | | Enables to discard any prefetched instructions after all preceding instructions complete |
| LIGHT_SYNC | | Ensures that all stores prior to LIGHT_SYNC complete before any new instructions can be executed on the processor that executed the LIGHT_SYNC directive |
| PREFETCH_BY_STREAM | | Uses the prefetch engine to recognize sequential access to adjacent cache lines and then requests anticipated lines from deeper levels of memory hierarchy |
| PREFETCH_BY_STREAM_BACKWARD | | Fetches data from incremental memory addresses |
| PREFETCH_BY_STREAM_FORWARD | | Loads data from main into the cache from main memory |
| PREFETCH_FOR_LOAD | | Prefetches data into the cache for reading by way of a cache prefetch instruction |
| PREFETCH_FOR_STORE | | Prefetches data into the cache for writing by way of a cache prefetch instruction |

| Directive | Type | Description |
|--|------|--|
| PROTECTED_UNLIMITED_STREAM_SET_GO_FORWARD | | Establishes an unlimited-length protected stream that begins with the cache line at the specified prefetch variable and fetches from increasing memory addresses |
| PROTECTED_UNLIMITED_STREAM_SET_GO_BACKWARD | | Fetches from decreasing memory addresses |
| PROTECTED_STREAM_SET_GO_FORWARD | | Establishes a limited-length protected stream that begins with the cache line at the specified prefetch variable and fetches from increasing memory |
| PROTECTED_STREAM_SET_GO_BACKWARD | | Fetches from decreasing memory addresses |
| PROTECTED_STREAM_COUNT | | Sets the number of cache lines for the specified limited-length stream |
| PROTECTED_STREAM_GO | | Starts to prefetch all limited-length streams |
| PROTECTED_STREAM_STOP | | Stops prefetching the specified protected stream |
| PROTECTED_STREAM_STOP_ALL | | Stops prefetching all protected streams |

Directives Usage

In this section we provide a series of examples that illustrate how to apply some of these compiler directives. Although some of them are not difficult to implement in the code, others are more involved.

As we previously described, the ASSERT directive provides a way to specify that a particular DO loop does not have dependencies. The assertion can take the following forms:

- ▶ INTERCNT(n); where n specifies the number of iterations for a given DO loop. n must be positive, scalar, and integer initialization expression.
- ▶ NODEPS specifies that no loop dependencies exist within a particular DO loop.

Important: The ASSERT directive applies only to the DO loop following the directive. It does not apply to nested DO loops.

Example 10-4 ASSERT directive

```

c  ASSERT Directive
    program dir1
    implicit none
    integer i,n, fun
    parameter (n = 100000)
    real*8 a(n)
    integer(8)  t0, tfin, irtc
c ...  start timer
    t0 = irtc()
!IBM*  ASSERT (NODEPS)
    do i = 1, n
        a(i) = a(i) * fun(i)
    end do
c ...  time
    tfin = (irtc() - t0)/1000000
    write(6,*) 'Time: ',tfin, 'msec.'
    stop
end

C
    function fun(i)
    fun = i * i
    return
end

```

In the above example we have used the idea of “loop-carried dependencies or data dependency, since this is a concept that is commonly used throughout this chapter, let us proceed to properly define “loop-carried dependencies”:

Dependencies Two iterations within the DO loop interfere with one and other.

An example of interference would be when there is a transfer of control outside the DO loop or execution of an EXIT, STOP, or PAUSE.

Loop optimization directive is BLOCK_LOOP. This directive relies on a well known optimization technique called blocking. This directive separates large iteration into smaller groups of iterations. Hence, the name blocking. The basic idea is to increase the utilization of the submemory heirarchy.

Example 10-5 BLOCK_LOOP directive

```

c  BLOCK_LOOP Directive
    program dir4
    implicit none
    integer i,j,k,n
    integer L3_cache_size, L3_cache_block
    integer L2_cache_size, L2_cache_block
    parameter (n = 100)
    integer a(n,n), b(n,n), c(n,n)
    integer(8) t0, tfin, irtc
    do j = 1,n
        do i = 1,n
            a(i,j) = rand()
            b(i,j) = rand()
        enddo
    enddo
    do j = 1, n
        do i = 1, n
            c(i,j)=0.0
        enddo
    enddo
c ...  start timer
    t0 = irtc()
!IBM* BLOCK_LOOP(L3_cache_size, L3_cache_block)
    do i = 1, n

!IBM* LOOPID(L3_cache_block)
!IBM* BLOCK_LOOP(L2_cache_size, L2_cache_block)
        do j = 1, n

!IBM* LOOPID(L2_cache_block)
            do k = 1, n
                c(i,j) = c(i,j) + a(i,k) * b(k,j)
            enddo
        enddo
    enddo
c ...  time
    tfin = (irtc() - t0)/1000000
    write(6,*)'Time: ',tfin, 'msec.'
    call dummy (c,n)
    stop

```

```

        end
c

```

Finally in this section we illustrate one of the hardware-specific directives. In the next section we shall cover some of the directives that are specific for POWER5. The next example corresponds to the `PREFETCH_BY_STREAM_FORWARD`. This set of directives makes use of the prefetching capabilities of POWER4 and POWER5 architectures by recognizing contiguous access to adjacent cache lines and then attempting to get them from memory heirarchy.

Example 10-6 PREFETCH_BY_STREAM_FORWARD directive

```

c  PREFETCH_BY_STREAM_FORWARD Directive
    program dir8
    implicit none
    integer*4 i,k,l,n
    parameter (n = 2**5, l = 2**10)
    real*4 a, b, c, temp
    real*4 aa(n),ac(n),ab(l)
    integer(8)  t0, tfin, irtc
    k = 32
c ...  start timer
    t0 = irtc()
!IBM*  PREFETCH_BY_STREAM_FORWARD(aa(1))
    do i = 1, n
        a = -i
        b = i + 1
        c = i + 2
        temp = sqrt(b*b - 4*a*c)
        aa(i) = ac(i) + (-b + temp) / (2*a)
        ab(i*k) = (-b - temp) / (2*a)
    enddo
c ...  time
    tfin = (irtc() - t0)/1000000
    write(6,*)'Time: ',tfin, 'msec.'
    call dummy (c,n)
    stop
    end

c
    subroutine dummy(aa,ab,n,l)
c
    dimension aa(n),ac(l)
c
    return
    end

```

10.2.4 POWER5 compiler features

These are some of options and suboptions that perform specific optimization for the POWER5 microarchitecture:

- ▶ -qarch=pwr5
- ▶ -qtune=pwr5
- ▶ -qcache=auto

Also, the following intrinsic are provided:

- ▶ SWDIV
- ▶ SWDIV_NOCHK
- ▶ FRE
- ▶ FRSQRTES
- ▶ POPCNT
- ▶ POPCNTB
- ▶ POPPAR

Compiler directives

XL Fortran 9.1 introduces a few new directives for POWER5. These streams are protected from being replaced by any hardware-detected streams. These directives correspond to:

Note: Valid for PowerPC 970 and POWER5

PROTECTED_UNLIMITED_STREAM_SET_GO_BACKWARD

Note: Valid for PowerPC 970 and POWER5

PROTECTED_STREAM_SET_GO_FORWARD

Note: Valid for POWER5

PROTECTED_STREAM_SET_GO_BACKWARD

PROTECTED_STREAM_COUNT

PROTECTED_STREAM_GO

PROTECTED_STREAM_STOP

PROTECTED_STREAM_STOP_ALL

The following example illustrates the usage of some of these directives and their performance benefits. The performance improvement for n between 10 and 2000 oscillates from a value of 2% to as large as 11%. In this four stream case the prefetch directives improve performance of this short vector lengths.

Example 10-7 New POWER5 prefetch directives.

```

c  PROTECTED_STREAM_SET_FORWARD Directive
c  PROTECTED_STREAM_COUNT Directive
    program dir8
    implicit none
    integer i,j,k,n,m,nopt2,ndim2,lcount
    parameter (n=2000)
    parameter (m=1000)
    parameter (ndim2 = 1000)
    real*8 x(n,ndim2),a(n,ndim2),b(n,ndim2),c(n,ndim2)
    integer(8) irtc, t0, tfin
c ...  start timer
    t0 = irtc()
    do k = 1, m
        lcount = 1 + n/16
        do j = ndim2, 1, -1
!IBM*  PROTECTED_STREAM_SET_FORWARD(x(1,j),0)
!IBM*  PROTECTED_STREAM_COUNT(lcount,0)
!IBM*  PROTECTED_STREAM_SET_FORWARD(a(1,j),1)
!IBM*  PROTECTED_STREAM_COUNT(lcount,1)
!IBM*  PROTECTED_STREAM_SET_FORWARD(b(1,j),2)
!IBM*  PROTECTED_STREAM_COUNT(lcount,2)
!IBM*  PROTECTED_STREAM_SET_FORWARD(c(1,j),3)
!IBM*  PROTECTED_STREAM_COUNT(lcount,3)
!IBM*  EIEIO
!IBM*  PROTECTED_STREAM_GO
            do i = 1, n
                x(i,j) = x(i,j) + a(i,j) * b(i,j) + c(i,j)
            enddo
        enddo
    enddo

```

```

        enddo
        call dummy(x,n)
    enddo
c ...   time
        tfin = (irtc() - t0)/1000000
        write(6,*)'Time: ',tfin, 'msec.'
        stop
    end

c
        subroutine dummy(x,n)
c
        dimension x(n)
c
        return
    end

```

Recommended compiler flags

In the POWER3 tuning guide the compiler flags for POWER3 recommended are:

-O3 -qarch=pwr3 -qtune=pwr3 [-qcache=auto] or

-O3 -qstrict -qarch==pwr3 -qtune=pwr3 [-qcache=auto]

In the case of POWER4 the recommendation for a starting compiler options are:

-O3 -qarch=pwr4 -qtune=pwr4

In this POWER5 version the recommendation is:

-O3 -qarch=auto -qtune=auto -qcache=auto

-O level

The -O3 option provides conservative optimization and it is currently used on most major scientific and engineering applications. There are some cases where -O2 might be preferable over -O3, this is particularly true for very large applications where after extensive testing the -O3 does not show better performance. Also, it is a good starting point for somebody that is planning optimize the application using higher compiler levels of optimization such as -O4 and -O5. When using that -O3, the next step that should be taken to improve the level of compiler optimization is by introducing -qhot or also called higher-order transformations. The objective of this option is to optimize loops. For very large scientific and engineering applications, the use of -O4 or -O5 will most likely has to be restricted to a few routines.

- qarch: This option and suboptions present several possibilities to choose from. The selection of the suboption will depend on how the binaries will be used. In our recommendation we assume that the application will be built on the system where the executables will be used. With -qarch=auto the particular architecture where the binaries are built will be recognized. If only one architecture will be used then -qarch=[pwr5] or [pwr4], etc. is recommended. On the other hand, if only one set of binaries will be used across all platforms, then you might want to consider -qarch=com.
- qtune: Instruction selection, scheduling and other implementation-dependent performance enhancements for a specific implementation of hardware architecture we also recommend “auto”. Similarly as in the previous case, this option is application (or where executable will run) specific.

Note: When running an application on more than one architecture, but the desire is to tune it for a particular architecture, the combination of -qarch and -qtune can be used.

Important: -qtune can improve performance but only has an effect when used in combination with options that enable optimization

- qcache: We recommend this option as auto.

Finally, there are other options used by scientific and engineering applications that might be applicable for your particular application as well. These correspond to:

- ▶ -qstrict
- ▶ -qmaxmem
- ▶ -qfixed
- ▶ -q64

10.3 Profiling applications

In Section 10.1.1 where we introduced the flowchart to identify applications bottlenecks, we learned as a first approximation how to classify applications as

CPU-bound, memory-bound, and I/O bound. Here we introduce a series of tools that will help us localize crucial sections in the code. These crucial sections correspond to a section or sections of the code that tend to dominate CPU utilization, this is reflected in the user time.

10.3.1 Hardware performance monitor

A key feature of this utility is its ability to provide *hardware performance counters information*. It can provide very fine grain information on how the application that it being monitored takes (or not) advantage of the POWER architecture. This utility is part of the IBM High Performance Computing Toolkit. This package includes the following new software (and a new installation method):

- ▶ Watson Sparse Matrix Library (WSMP)
- ▶ Modular I/O Performance Tool (MIO)
- ▶ MPI Tracer
- ▶ SHMEM Profiler
- ▶ OpenMP Profiler (PompProf)
- ▶ Graphical Interface tool with source code traceback (PeekPerf)
- ▶ New installation method via RPM modules

Similarly to the flowchart tha two introduced in the first section of this chapter, this set of tools can be classified based on the type of performance that the programmer is interested in analyzing. Based on the functionality of the different utilities the following components can be analyzed with this package:

- ▶ Hardware performance (HPM Toolkint)
 - catch
 - hpmcount
 - libhpm
 - hpmstat
- ▶ Shared memory performance (DPOMP)
 - pomprof
- ▶ Message-passing performance
 - MP_profiler, MP_tracer
 - TurboSHMEM, TurboMP
- ▶ Memory performance
 - Sigma

- ▶ Performance visualization
 - PeekPerf
- ▶ I/O performance
 - MIO (modular I/O)
- ▶ Mathematics performance
 - Watson sparse matrix (WSMP)

Finally this toolkit can be obtain as follows:

- ▶ Acquire as part of new procurement
- ▶ Acquire as part of ACTC performance tuning workshop
- ▶ Purchase license directly from IBM Research

In this section we discuss the version of hpmcount that works on POWER4. The hpmcount usage for POWER4 is as follows:

```
$hpmcount [-o <file>] [-n] [-g <group>] <a.out>
```

To get help and additional information from hpmcount:

```
$hpmcount [-h] [-c] [-l]
```

Table 10-6 *hpmcount flags and description.*

| Flag | Description |
|------------|--|
| -h | Display all available flags and brief description |
| -c | List all events from all counters |
| -l | List all the groups available on POWER4 |
| -o <file> | Creates an output file with all collected statistics called >file>.<pid>. For parallel runs this flags generates one file for each process |
| -n | Forces hpmcount NOT to send output to stdout. Only active in combination with -o <file> |
| -g <group> | List selected groups |

The groups available on POWER4 are extensive, the valid groups go from 0 to 60. Although selecting the groups to analyze will depend on the type of information that is being searched, the groups that have been recommended for applications tuning are presented in Table 10-7. A complete listing of the groups for POWER4 may be found in: /usr/pmapi/lib/POWER4.gps.

Table 10-7 Selected set of groups for applications tuning.

| Group | Information provided |
|-------|--|
| 5 | pm_lsource, information on data source, counts of loads from L2, L3, and memory |
| 53 | pm_pe_bench1, information for fp analysis, counts of cycles on instructions, fixed-point operations, and FP operations (includes divides, SQRT, FMA, and FMOV or FEST) |
| 56 | pm_pe_bench4, information for L1 and TLB analysis, counts of cycles on instructions, TLB misses, loads, stores, and L1 misses |
| 58 | pm_pe_bench6, information for L3 analysis, counts of cycles on instructions, loads from L3, and loads from memory |
| 60 | pm_hpmcount2, information for computation intensity analysis, counts of cycles on instructions on FP operations (including divides, FMA, loads, and stores) |

The output obtained from hpmcount can be divided into several sections:

1. In this section prints the hpmcount version and the total elapsed time that took to perform a particular run.
2. Prints the resource usage statistics
3. Hardware counter information for the ones that hpmcount is following for a particular run
4. Performance of miscellaneous hardware features. On POWER4 this report depends on the group that was selected at run time.

Here we shall not describe each of the parameters printed by **hpmcount**, most of them are self explained and they are explained in the hpmcount manual. Instead, the following two examples illustrates how to use hpmcount for a simple performance tuning exercise. Example 10-8 is the case of a simple program

where the double nested Do loops have not been optimized (blocked) and therefore incur in L2, L3 ,and TLB misses. In other words, no data can be reused.

Example 10-8 Double nested Do loop without blocking

```

program reuse
c
    IMPLICIT NONE
    integer ARRAY_SIZE
    parameter(ARRAY_SIZE = 25000)
    integer I, J, II, JJ
    real A(ARRAY_SIZE, ARRAY_SIZE)
    real B(ARRAY_SIZE, ARRAY_SIZE)
    real S, SS
    integer(8) T1, T2, IRTC
C*****
C*                               Untuned Loop                               *
C*****
    T1 = IRTC()

    DO J=1, ARRAY_SIZE
        DO I=1, ARRAY_SIZE
            S = S + A(I,J)*B(J, I)
        ENDDO
    ENDDO

    T2 = IRTC()
    write(*,*)'Untuned loop took', (T2-T1)/1000000, 'msec'

C    Need to actually use the results of the calculations or else the
C    optimizing compiler may just skip doing them...so we'll just
C    print them. This will force the optimizer to actually do the work.
    print *, A(ARRAY_SIZE, ARRAY_SIZE), S

999  end

```

Example 10-9 shows a double nested Do loop that has been optimized by blocking the both Do loops. In this case there is data reuse since this example is keeping data in the different levels of cache. **hpmcount** helps to see this.

Example 10-9 Tuned version of the reuse program.

```

program reuse
c
    IMPLICIT NONE
    integer ARRAY_SIZE, NB
    parameter(ARRAY_SIZE = 25000, NB = 4)
    integer I, J, II, JJ

```

```

      real A(ARRAY_SIZE, ARRAY_SIZE), AA(ARRAY_SIZE, ARRAY_SIZE)
      real B(ARRAY_SIZE, ARRAY_SIZE), BB(ARRAY_SIZE, ARRAY_SIZE)
      real S, SS
      integer(8) T1, T2, IRTC
C*****
C*                               Tuned Loop                               *
C*****
      T1 = IRTC()

      DO JJ = 1, ARRAY_SIZE, NB
        DO II = 1, ARRAY_SIZE, NB
          DO J = J, JJ, MIN(ARRAY_SIZE, JJ+NB-1)
            DO I = II, MIN(ARRAY_SIZE, II+NB-1)
              SS = SS + AA(I,J)*BB(J,I)
            ENDDO
          ENDDO
        ENDDO
      ENDDO

      T2 = IRTC()
      write(*,*) 'Tuned loop took', (T2-T1)/1000000, 'msec'

C      Need to actually use the results of the calculations or else the
C      optimizing compiler may just skip doing them...so we'll just
C      print them. This will force the optimizer to actually do the work.
      print *, AA(ARRAY_SIZE, ARRAY_SIZE), SS

999  end

```

We have selected only part of the **hpmcount** output in both cases. Mainly the section that illustrates the performance with respect to L2, L3, and TLB using the two versions of the code. Each section is separated by a dollar sign and **hpmcount** and the group that was selected. We used the 5 and 56 groups.

\$ hpmcount -g 5 reuse_u

Execution time (wall clock time): 170.883827 seconds

| | | |
|---|---|-------------------|
| PM_DATA_FROM_L3 (Data loaded from L3) | : | 464363407 |
| PM_DATA_FROM_MEM (Data loaded from memory) | : | 151160443 |
| PM_DATA_FROM_L35 (Data loaded from L3.5) | : | 7981418 |
| PM_DATA_FROM_L2 (Data loaded from L2) | : | 29723097 |
| PM_DATA_FROM_L25_SHR (Data loaded from L2.5 shared) | : | 0 |
| PM_DATA_FROM_L275_SHR (Data loaded from L2.75 shared) | : | 4 |
| PM_DATA_FROM_L275_MOD (Data loaded from L2.75 modified) | : | 16 |
| PM_DATA_FROM_L25_MOD (Data loaded from L2.5 modified) | : | 0 |
| | | |
| Total Loads from L2 | : | 29.723 M |
| L2 load traffic | : | 3628.310 MBytes |
| L2 load bandwidth per processor | : | 21.233 MBytes/sec |

| | | |
|-------------------------------------|---|--------------------|
| L2 Load miss rate | : | 95.450 % |
| Total Loads from L3 | : | 472.345 M |
| L3 load traffic | : | 57659.280 MBytes |
| L3 load bandwidth per processor | : | 337.418 MBytes/sec |
| L3 Load miss rate | : | 24.244 % |
| Memory load traffic | : | 18452.203 MBytes |
| Memory load bandwidth per processor | : | 107.981 MBytes/sec |

\$ hpmcount -g 5 reuse_t

Execution time (wall clock time): 1.48904 seconds

| | | |
|---|---|------------------|
| PM_DATA_FROM_L3 (Data loaded from L3) | : | 16 |
| PM_DATA_FROM_MEM (Data loaded from memory) | : | 93 |
| PM_DATA_FROM_L35 (Data loaded from L3.5) | : | 10 |
| PM_DATA_FROM_L2 (Data loaded from L2) | : | 1167 |
| PM_DATA_FROM_L25_SHR (Data loaded from L2.5 shared) | : | 0 |
| PM_DATA_FROM_L275_SHR (Data loaded from L2.75 shared) | : | 10 |
| PM_DATA_FROM_L275_MOD (Data loaded from L2.75 modified) | : | 10 |
| PM_DATA_FROM_L25_MOD (Data loaded from L2.5 modified) | : | 0 |
| | | |
| Total Loads from L2 | : | 0.001 M |
| L2 load traffic | : | 0.145 MBytes |
| L2 load bandwidth per processor | : | 0.097 MBytes/sec |
| L2 Load miss rate | : | 9.112 % |
| Total Loads from L3 | : | 0.000 M |
| L3 load traffic | : | 0.003 MBytes |
| L3 load bandwidth per processor | : | 0.002 MBytes/sec |
| L3 Load miss rate | : | 78.151 % |
| Memory load traffic | : | 0.011 MBytes |
| Memory load bandwidth per processor | : | 0.008 MBytes/sec |

\$ hpmcount -g 56 reuse_u

Execution time (wall clock time): 170.550834 seconds

| | | |
|--|---|--------------------|
| PM_DTLB_MISS (Data TLB misses) | : | 628854887 |
| PM_ITLB_MISS (Instruction TLB misses) | : | 149977 |
| PM_LD_MISS_L1 (L1 D cache load misses) | : | 930603848 |
| PM_ST_MISS_L1 (L1 D cache store misses) | : | 130067750 |
| PM_CYC (Processor cycles) | : | 231231424980 |
| PM_INST_CMPL (Instructions completed) | : | 20001760251 |
| PM_ST_REF_L1 (L1 D cache store references) | : | 2391568051 |
| PM_LD_REF_L1 (L1 D cache load references) | : | 7406257146 |
| | | |
| Utilization rate | : | 93.272 % |
| %% TLB misses per cycle | : | 0.272 % |
| number of loads per TLB miss | : | 11.777 |
| Total l2 data cache accesses | : | 1060.672 M |
| %% accesses from L2 per cycle | : | 0.459 % |
| L2 traffic | : | 129476.513 MBytes |
| L2 bandwidth per processor | : | 759.167 MBytes/sec |
| Total load and store operations | : | 9797.825 M |

```

number of loads per load miss          :          7.959
number of stores per store miss        :          18.387
number of load/stores per D1 miss      :           9.237
L1 cache hit rate                      :          89.174 %
MIPS                                   :          117.277
Instructions per cycle                  :           0.087
$ hpmcount -g 56 reuse_t
Execution time (wall clock time): 1.488173 seconds
PM_DTLB_MISS (Data TLB misses)        :           452
PM_ITLB_MISS (Instruction TLB misses)  :            106
PM_LD_MISS_L1 (L1 D cache load misses) :           1998
PM_ST_MISS_L1 (L1 D cache store misses) :           4629
PM_CYC (Processor cycles)              :       2156932098
PM_INST_CMPL (Instructions completed)  :       1172053048
PM_ST_REF_L1 (L1 D cache store references) :       237543426
PM_LD_REF_L1 (L1 D cache load references) :       377625191

Utilization rate                       :          99.710 %
%% TLB misses per cycle                 :           0.000 %
number of loads per TLB miss           :       835453.962
Total l2 data cache accesses            :           0.007 M
%% accesses from L2 per cycle           :           0.000 %
L2 traffic                             :           0.809 MBytes
L2 bandwidth per processor              :           0.544 MBytes/sec
Total load and store operations         :           615.169 M
number of loads per load miss          :       189001.597
number of stores per store miss        :       51316.359
number of load/stores per D1 miss      :       92827.617
L1 cache hit rate                      :          99.999 %
MIPS                                   :       787.578
Instructions per cycle                  :           0.543

```

The execution time reported for the program that has the double nested loop that is not optimized (reuse_u) is approximately 180 Seconds. On the other hand, the time for reuse_t (optimized version) is only approximate 2 Seconds. For this trivial example, clearly this is an indication that there is a problem with that double nested loop. Further analysis of the hpmcount output reveals that the version that is not optimized is not taking advantage of the memory hierarchy. A simple inspection of some of the counters such as: PM_DATA_FROM_L3, PM_DATA_FROM_MEM, PM_DATA_FROM_L2, PM_LD_MISS_L1, and PM_ST_MISS_L1 indicate that the versionis that is not optimized is not re-using data and it is incurring in very large number of caches misses.

10.3.2 Profiling utilities

This series of utilities are just a set of tools that can help you find bottlenecks at a very fine grain level of granularity. Scientific and engineering applications consist

not only of the actual code that performs simulations but it involves system library calls and system specific routines. Normally when an application has not been optimized, they tend to spend most of the CPU time executing critical sections of the code. This critical sections of the code are normally localized to a subroutine or a few Do loops. These utilities are essential to find these bottlenecks and filter them from system or kernel related software. Although there might be other utilities, in this section will focus on tprof, gprof, and xprofiler.

tprof

In AIX **tprof** is part of the performance toolbox This utility reports CPU usage for both individual programs and the system as a whole. It is useful to identify sections of the code that are most heavily using CPU. The raw data from tprof is obtained via the Trace facility (see Performance analysis with the trace facility). In the following examples, **tprof** was used in conjunction with a scientific application, namely AMBER7. We already introduced this life sciences application in previous chapters, so we recommend the reader to see Chapter 5 for an overview of AMBER7. Here we instrumented the **sander** module.

\$ tprof -z -u -p PID -x read sander

Example 10-10 tprof output obtained with the sander module.

| Process | FREQ | Total | Kernel | User | Shared | Other |
|----------------------------------|------|-------|--------|-------|--------|-------|
| ===== | ==== | ===== | ===== | ===== | ===== | ===== |
| /scratch/amber7/exe/sander | 1 | 6978 | 3 | 6975 | 0 | 0 |
| wait | 2 | 957 | 957 | 0 | 0 | 0 |
| IBM.CSMAgentRMd | 1 | 87 | 2 | 0 | 85 | 0 |
| /home/db2inst1/sqllib/adm/db2set | 2 | 2 | 1 | 0 | 1 | 0 |
| /home/db2as/das/bin/db2fm | 2 | 2 | 2 | 0 | 0 | 0 |
| /home/db2inst1/sqllib/bin/db2fm | 1 | 1 | 1 | 0 | 0 | 0 |
| /usr/sbin/muxatmd | 1 | 1 | 1 | 0 | 0 | 0 |
| /usr/bin/sh | 1 | 1 | 1 | 0 | 0 | 0 |
| ===== | ==== | ===== | ===== | ===== | ===== | ===== |
| Total | 11 | 8029 | 968 | 6975 | 86 | 0 |

| Process | PID | TID | Total | Kernel | User | Shared | Other |
|----------------------|--------|--------|-------|--------|-------|--------|-------|
| ===== | === | === | ===== | ===== | ===== | ===== | ===== |
| ch/amber7/exe/sander | 348388 | 774299 | 6978 | 3 | 6975 | 0 | 0 |
| wait | 8196 | 8197 | 838 | 838 | 0 | 0 | 0 |
| wait | 12294 | 12295 | 119 | 119 | 0 | 0 | 0 |
| IBM.CSMAgentRMd | 163856 | 544779 | 87 | 2 | 0 | 85 | 0 |
| /db2as/das/bin/db2fm | 274632 | 782505 | 1 | 1 | 0 | 0 | 0 |
| /db2as/das/bin/db2fm | 274620 | 606215 | 1 | 1 | 0 | 0 | 0 |
| /usr/bin/sh | 188626 | 786589 | 1 | 1 | 0 | 0 | 0 |
| t1/sqllib/adm/db2set | 188620 | 786583 | 1 | 0 | 0 | 1 | 0 |
| /usr/sbin/muxatmd | 225408 | 344283 | 1 | 1 | 0 | 0 | 0 |
| t1/sqllib/adm/db2set | 188614 | 733255 | 1 | 1 | 0 | 0 | 0 |

| | | | | | | | |
|----------------------|--------|--------|-------|-------|-------|-------|-------|
| st1/sqllib/bin/db2fm | 266376 | 639199 | 1 | 1 | 0 | 0 | 0 |
| ===== | === | === | ===== | ===== | ===== | ===== | ===== |
| Total | | | 8029 | 968 | 6975 | 86 | 0 |

Total Samples = 8029 Total Elapsed Time = 71.74s

Total Ticks For All Processes (USER) = 6975

| | | | | |
|----------------------------|-------|-------|----------|--------|
| User Process | Ticks | % | Address | Bytes |
| ===== | ===== | ===== | ===== | ===== |
| /scratch/amber7/exe/sander | 6975 | 86.87 | 10000150 | 116478 |

Profile: /scratch/amber7/exe/sander

Total Ticks For All Processes (/scratch/amber7/exe/sander) = 6975

| | | | | | |
|------------|-------|-------|----------------------|---------|-------|
| Subroutine | Ticks | % | Source | Address | Bytes |
| ===== | ===== | ===== | ===== | ===== | ===== |
| ._log | 3816 | 47.53 | ib/libm/POWER/logF.c | 93490 | 380 |
| .egb | 2118 | 26.38 | _egb_.f | d0ad0 | 4ea0 |
| ._exp | 901 | 11.22 | ib/libm/POWER/expF.c | 16a70 | 2d0 |
| .daxpy | 89 | 1.11 | daxpy.f | 9c7c0 | 230 |
| .ephi | 18 | 0.22 | _ene_.f | 8bd30 | 21c0 |
| ._sin | 13 | 0.16 | ib/libm/POWER/sinF.c | ac90 | 200 |
| ._acos | 6 | 0.07 | b/libm/POWER/acosF.c | 18470 | 2b0 |
| .shake | 5 | 0.06 | _shake_.f | f42f0 | 9e0 |
| ._cos | 4 | 0.05 | ib/libm/POWER/cosF.c | aa70 | 220 |
| .angl | 4 | 0.05 | _ene_.f | 8a9b0 | e00 |
| .bond | 1 | 0.01 | _ene_.f | 89330 | e80 |

Profiling of the sander module means that the Trace facility is activated and instructed to collect from the trace hook that records the contents of the Instruction Address Register when a system-clock interrupts occurs. tprof reports the distribution of address occurrences as “ticks” across the programs involved in the workload.

Example 10-10 illustrates the output of tprof for the sander module. Notice that one the three more time consuming routines correspond to logF.c, egf.f, and expF.c. Furthermore, the two C routines are part of the libm, which can be easily replaced by the equivalent routine in the MASS library. In example 10-11 we should the output of tprof after replacing expF.c with the vectorized version in the MASS library.

Example 10-11 tprof output obtained with the sander module using MASS libraries

| | | | | | | |
|---------|-------|-------|--------|-------|--------|-------|
| Process | FREQ | Total | Kernel | User | Shared | Other |
| ===== | ===== | ===== | ===== | ===== | ===== | ===== |

| | | | | | | |
|----------------------------------|------|-------|-------|-------|-------|-------|
| /scratch/amber7/exe/sander_mass | 1 | 5956 | 1 | 5954 | 1 | 0 |
| wait | 2 | 764 | 764 | 0 | 0 | 0 |
| /home/db2inst1/sqllib/adm/db2set | 1 | 1 | 1 | 0 | 0 | 0 |
| ===== | ==== | ===== | ===== | ===== | ===== | ===== |
| Total | 4 | 6721 | 766 | 5954 | 1 | 0 |

| Process | PID | TID | Total | Kernel | User | Shared | Other |
|----------------------|--------|--------|-------|--------|-------|--------|-------|
| ===== | === | === | ===== | ===== | ===== | ===== | ===== |
| ber7/exe/sander_mass | 356544 | 753849 | 5956 | 1 | 5954 | 1 | 0 |
| wait | 8196 | 8197 | 669 | 669 | 0 | 0 | 0 |
| wait | 12294 | 12295 | 95 | 95 | 0 | 0 | 0 |
| t1/sqllib/adm/db2set | 188632 | 786595 | 1 | 1 | 0 | 0 | 0 |
| ===== | === | === | ===== | ===== | ===== | ===== | ===== |
| Total | | | 6721 | 766 | 5954 | 1 | 0 |

Total Samples = 6721 Total Elapsed Time = 64.13s

Total Ticks For All Processes (USER) = 5954

| User Process | Ticks | % | Address | Bytes |
|---------------------------------|-------|-------|----------|--------|
| ===== | ===== | ===== | ===== | ===== |
| /scratch/amber7/exe/sander_mass | 5954 | 88.59 | 10000150 | 114238 |

Profile: /scratch/amber7/exe/sander_mass

Total Ticks For All Processes (/scratch/amber7/exe/sander_mass) = 5954

| Subroutine | Ticks | % | Source | Address | Bytes |
|------------|-------|-------|----------------------|---------|-------|
| ===== | ===== | ===== | ===== | ===== | ===== |
| ._log | 4066 | 60.50 | ib/libm/POWER/logF.c | 90590 | 380 |
| .egb | 1555 | 23.14 | _egb_.f | cdbd0 | 4e60 |
| vrsqrt | 148 | 2.20 | vrsqrt_p4.32s | 7ef50 | 7a0 |
| .daxpy | 67 | 1.00 | daxpy.f | 998c0 | 230 |
| .vexp | 59 | 0.88 | vexp_p4.32s | d2a30 | 8e8 |
| .ephi | 16 | 0.24 | _ene_.f | 88e30 | 21c0 |
| ._cos | 13 | 0.19 | ib/libm/POWER/cosF.c | aa70 | 220 |
| ._acos | 11 | 0.16 | b/libm/POWER/acosF.c | 18470 | 2b0 |
| .angl | 9 | 0.13 | _ene_.f | 87ab0 | e00 |
| ._sin | 7 | 0.10 | ib/libm/POWER/sinF.c | ac90 | 200 |
| .runmd | 3 | 0.04 | _runmd_.f | e9a30 | 63c0 |

Here we see that one of the bottlenecks has totally dissapeared, that is, the one coming from the expF.c rouine. We shall show similar type of information with **gprof** and **xprofiler**.

gprof

Similarly to tprof this utility allows you to look at the code to identify the critical sections of the code. **gprof** is a utility developed by GNU. For more information about this utility see: The GNU Profiler by J. Fenlason and R. Stallman. To use gprof the following steps are required:

- ▶ The application needs to be compiled and linked with profiling enabled.
- ▶ The application has to run to generate a profile data file.
- ▶ **gprof** has to be run to analyze the data.

Here, rather than illustrating the use of gprof or tprof with a trivial example we apply it to the life sciences application AMBER7. The first step involves modifying the script that compiles and builds the sander module. This may be seen where we included the -pg option.

Tip: You want to make sure that -pg is also included when the loader is invoked.

This corresponds to the Machine.ibm_aix modified to invoke either **gprof** or the **xprofiler**.

```
##### LOADER/LINKER:
# Use Standard options
setenv LOAD "xlf90 -bmaxdata:0x80000000 -pg "
# Load with the IBM MASS & ESSL libraries
setenv LOADLIB " "
if ( $HAS_MASSLIB == "yes" ) setenv LOADLIB "-L$MASSLIBDIR -lmassvp4 "
if ( $VENDOR_BLAS == "yes" ) setenv LOADLIB "$LOADLIB -lblas "
if ( $VENDOR_LAPACK == "yes" ) setenv LOADLIB "$LOADLIB -lessl "

# little or no optimization:
setenv L0 "xlf90 -qfixed -c -pg"

# modest optimization (local scalar):
setenv L1 "xlf90 -qfixed -O2 -c -pg"

# high scalar optimization (but not vectorization):
setenv L2 "xlf90 -qfixed -O3 -pg -qmaxmem=-1 -qarch=auto -qtune=auto -c"

# high optimization (may be vectorization, not parallelization):
setenv L3 "xlf90 -qfixed -O3 -pg -qmaxmem=-1 -qarch=auto -qtune=auto -c"
```

Example 10-12

ngranularity: Each sample hit covers 4 bytes. Time: 90.27 seconds

| index | %time | self | descendents | called/total called+self called/total | parents name children | index |
|-------|-------|-------|-------------|---|-----------------------------|-------|
| [1] | 89.6 | 0.00 | 80.92 | 1/1 | .__start | [2] |
| | | 0.00 | 80.92 | 1 | .main | [1] |
| | | 0.00 | 80.92 | 1/1 | .runmd | [3] |
| | | 0.00 | 0.00 | 1/1 | .rdparm2 | [26] |
| ----- | | | | | | |
| 6.6s | | | | | <spontaneous> | |
| [2] | 89.6 | 0.00 | 80.92 | | .__start | [2] |
| | | 0.00 | 80.92 | 1/1 | .main | [1] |
| ----- | | | | | | |
| [3] | 89.6 | 0.00 | 80.92 | 1/1 | .main | [1] |
| | | 0.00 | 80.92 | 1 | .runmd | [3] |
| | | 0.01 | 80.89 | 100/100 | .force | [4] |
| | | 0.02 | 0.00 | 100/100 | .shake | [20] |
| | | 0.00 | 0.00 | 1/366954097 | ._log | [6] |
| | | 0.00 | 0.00 | 300/1429 | .timer_start | [37] |
| ----- | | | | | | |
| [4] | 89.6 | 0.01 | 80.89 | 100/100 | .runmd | [3] |
| | | 0.01 | 80.89 | 100 | .force | [4] |
| | | 26.21 | 54.04 | 100/100 | .egb | [5] |
| | | 0.20 | 0.29 | 200/200 | .ephi | [11] |
| | | 0.04 | 0.09 | 200/200 | .angl | [15] |
| | | 0.02 | 0.00 | 100/100 | .bond | [19] |
| | | 0.00 | 0.00 | 1300/1300 | .get_stack | [40] |
| ----- | | | | | | |
| [5] | 88.9 | 26.21 | 54.04 | 100/100 | .force | [4] |
| | | 26.21 | 54.04 | 100 | .egb | [5] |
| | | 43.93 | 0.00 | 366954096/366954097 | ._log | [6] |
| | | 10.11 | 0.00 | 99723844/99723844 | ._exp | [7] |
| | | 0.00 | 0.00 | 225/1429 | .timer_start | [37] |
| | | 0.00 | 0.00 | 225/1429 | .timer_stop | [38] |
| ----- | | | | | | |
| [6] | 48.7 | 0.00 | 0.00 | 1/366954097 | .runmd | [3] |
| | | 43.93 | 0.00 | 366954096/366954097 | .egb | [5] |
| | | 43.93 | 0.00 | 366954097 | ._log | [6] |
| ----- | | | | | | |
| [7] | 11.2 | 10.11 | 0.00 | 99723844/99723844 | .egb | [5] |
| | | 10.11 | 0.00 | 99723844 | ._exp | [7] |
| ----- | | | | | | |
| 6.6s | | | | | <spontaneous> | |
| [8] | 8.2 | 7.42 | 0.00 | | ._mcount | [8] |
| ----- | | | | | | |
| 6.6s | | | | | <spontaneous> | |

```

[9]      0.8    0.70      0.00      .daxpy [9]
-----
6.6s
[10]     0.7    0.62      0.00      <spontaneous>
      .qincrement [10]
-----
[11]     0.5      0.20      0.29    200/200      .force [4]
      0.20      0.29    200      .ephi [11]
      0.19      0.00  1618200/2075531      ._sin [14]
      0.06      0.00  809100/1266400      ._acos [17]
      0.05      0.00  1618200/1618231      ._cos [18]
-----
6.6s
[12]     0.3    0.25      0.00      <spontaneous>
      .__stack_pointer [12]
-----
6.6s
[13]     0.3    0.25      0.00      <spontaneous>
      .qincrement1 [13]
-----
      0.00      0.00    31/2075531      .dihpar [25]
      0.05      0.00  457300/2075531      .angl [15]
      0.19      0.00  1618200/2075531      .ephi [11]
[14]     0.3    0.24      0.00  2075531      ._sin [14]
-----
.....
-----

```

ngranularity: Each sample hit covers 4 bytes. Time: 90.27 seconds

| % | cumulative | self | self | total | | |
|------|------------|---------|-----------|---------|---------|-----------------------|
| time | seconds | seconds | calls | ms/call | ms/call | name |
| 48.7 | 43.93 | 43.93 | 366954097 | 0.00 | 0.00 | ._log [6] |
| 29.0 | 70.14 | 26.21 | 100 | 262.10 | 802.50 | .egb [5] |
| 11.2 | 80.25 | 10.11 | 99723844 | 0.00 | 0.00 | ._exp [7] |
| 8.2 | 87.67 | 7.42 | | | | .__mcount [8] |
| 0.8 | 88.37 | 0.70 | | | | .daxpy [9] |
| 0.7 | 88.99 | 0.62 | | | | .qincrement [10] |
| 0.3 | 89.24 | 0.25 | | | | .__stack_pointer [12] |
| 0.3 | 89.49 | 0.25 | | | | .qincrement1 [13] |
| 0.3 | 89.73 | 0.24 | 2075531 | 0.00 | 0.00 | ._sin [14] |
| 0.2 | 89.93 | 0.20 | 200 | 1.00 | 2.47 | .ephi [11] |
| 0.1 | 90.03 | 0.10 | | | | .EndIORWFmt [16] |
| 0.1 | 90.12 | 0.09 | 1266400 | 0.00 | 0.00 | ._acos [17] |
| 0.1 | 90.17 | 0.05 | 1618231 | 0.00 | 0.00 | ._cos [18] |
| 0.0 | 90.21 | 0.04 | 200 | 0.20 | 0.63 | .angl [15] |
| 0.0 | 90.23 | 0.02 | 100 | 0.20 | 0.20 | .bond [19] |
| 0.0 | 90.25 | 0.02 | 100 | 0.20 | 0.20 | .shake [20] |
| 0.0 | 90.26 | 0.01 | 15149 | 0.00 | 0.00 | .cvtloop [22] |

Xprofiler

This utility is a graphical tool that allows you to perform profiling of your code. The output is similar as the other two tools, except that **xprofiler** is more flexible and it is more powerful when it comes to analyzing large applications. The procedure to build and run the application with **xprofiler** enabled is similar to **gprof**. Once this is done, just invoke **xprofiler**.

\$ xprofiler

Once **xprofiler** has been started, you must be able to select your executable and the gmon.out file. This is all what you need to do to start displaying information related to your application. Although there are different viewing options in xprofiler, in Figure 10-5 we illustrate one of them, here, although difficult to read due to the business of the display, it illustrates the entire calling three of AMBER7.

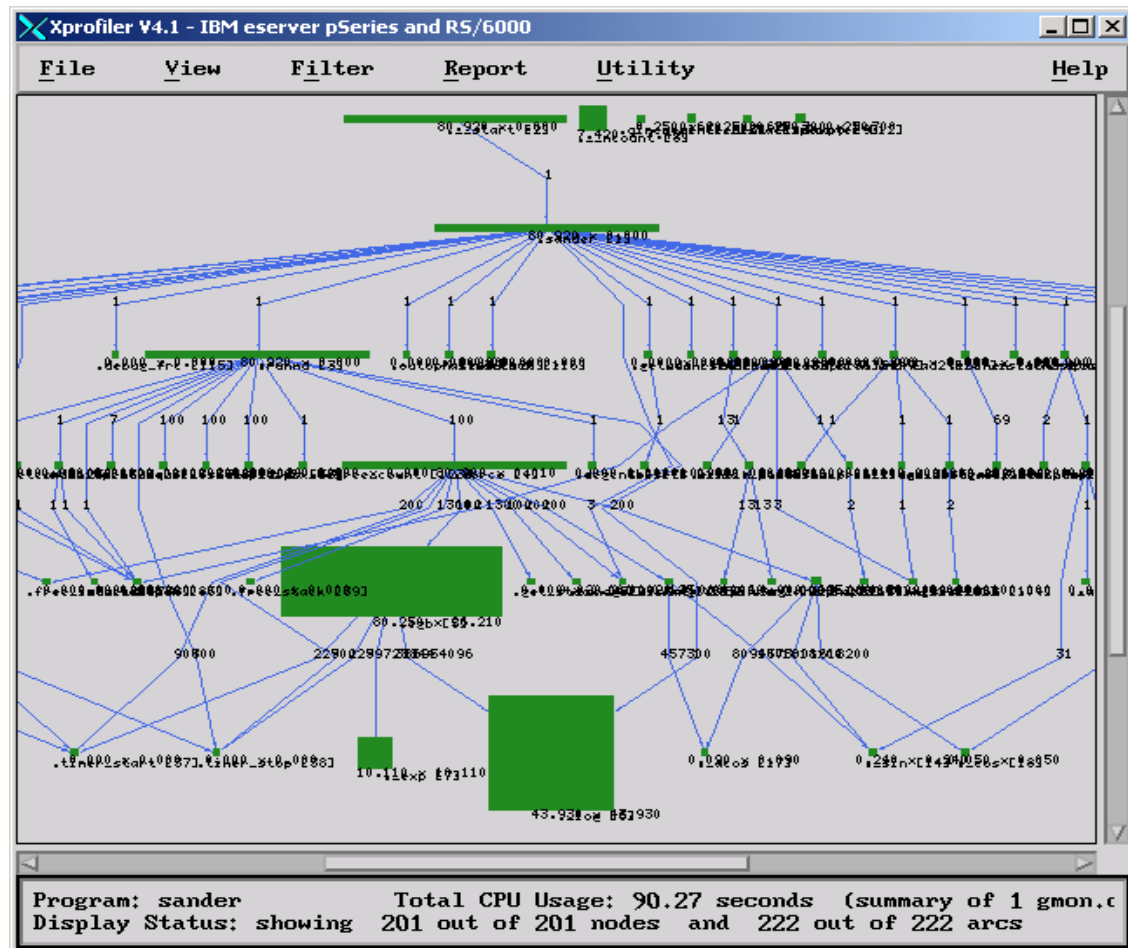


Figure 10-5 xprofiler calling tree for the sander module

In Figure 10-6 we focus our attention on the routines that are accumulating most of the CPU time. This is very easy to do with **xprofiler**. Again, at this point the easiest path would be to replace the libm routines with equivalent MASS library routines.

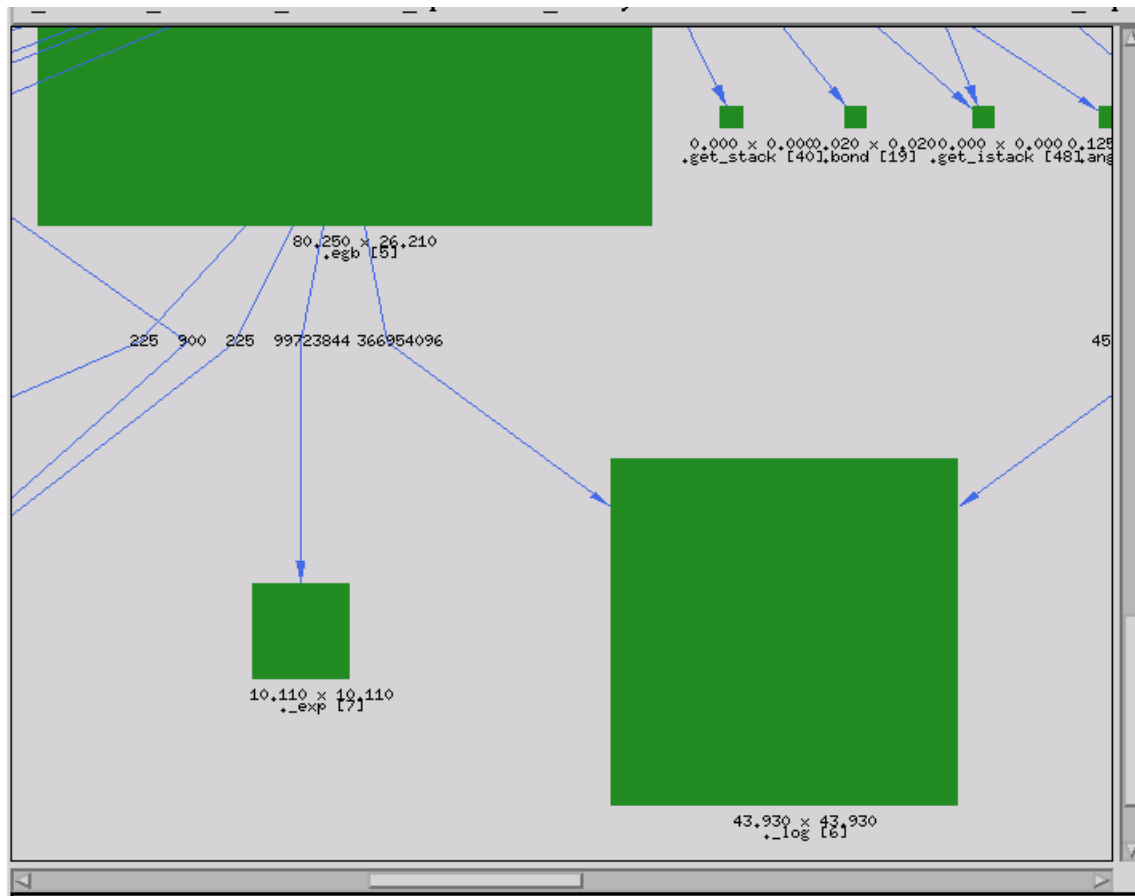
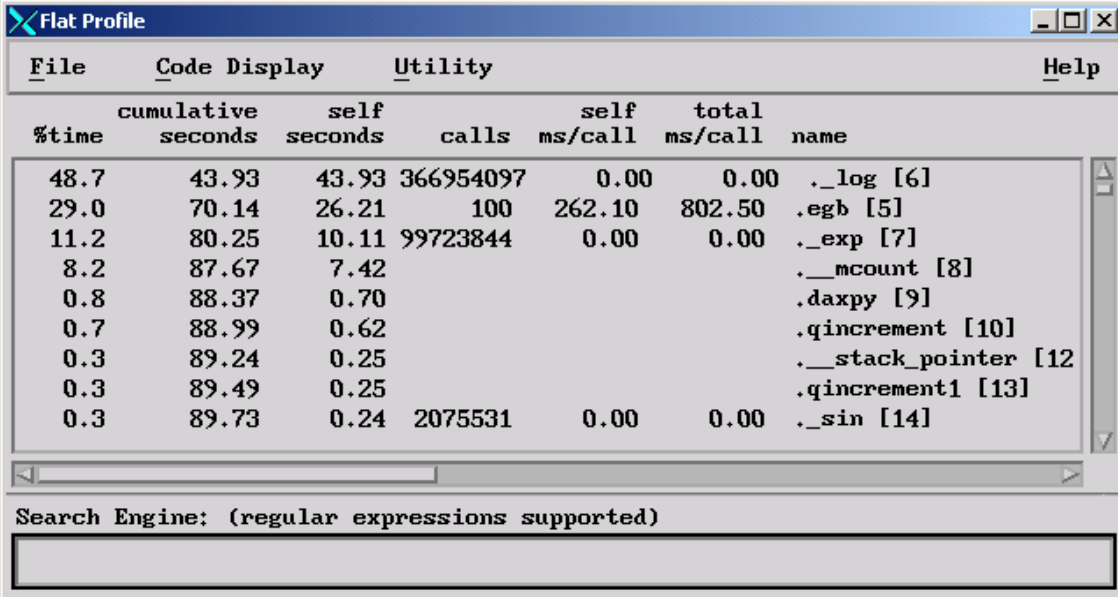


Figure 10-6 xprofiler view of the time consuming routines.

The other option that we illustrate regarding **xprofiler** is the flat profile that can be produced as part of the functionality of **xprofiler**. Figure 10-7 illustrates the flat profile for the AMBER7 run.



| Flat Profile | | | | | | |
|--------------|--------------------|--------------|-----------|--------------|---------------|----------------------|
| File | Code Display | Utility | Help | | | |
| %time | cumulative seconds | self seconds | calls | self ms/call | total ms/call | name |
| 48.7 | 43.93 | 43.93 | 366954097 | 0.00 | 0.00 | ._log [6] |
| 29.0 | 70.14 | 26.21 | 100 | 262.10 | 802.50 | .egb [5] |
| 11.2 | 80.25 | 10.11 | 99723844 | 0.00 | 0.00 | ._exp [7] |
| 8.2 | 87.67 | 7.42 | | | | ._mcount [8] |
| 0.8 | 88.37 | 0.70 | | | | .daxpy [9] |
| 0.7 | 88.99 | 0.62 | | | | .qincrement [10] |
| 0.3 | 89.24 | 0.25 | | | | ._stack_pointer [12] |
| 0.3 | 89.49 | 0.25 | | | | .qincrement1 [13] |
| 0.3 | 89.73 | 0.24 | 2075531 | 0.00 | 0.00 | ._sin [14] |

Search Engine: (regular expressions supported)

Figure 10-7 Flat profile produced using xprofiler.

Finally, in this section we have provided a series of tools that can be utilized to analyze the performance of scientific and engineering applications. Clearly xprofiler provides a very powerful utility with a friendly interface that can help expedite the localization of critical sections in the code. Now, if there is no access to the source code, then tprof provides a good alternative. As we illustrated in this section, with tprof is possible to attached and get information to an application that is running by attaching to the process ID (PID).

10.4 Memory management

In this section we try to emphasize the fact that understanding data locality is of key importance when it comes to code optimization. Figure 10-8 illustartes how memory heirarchy is organized on POWER5. A detailed description of each of the component may be found in a previous chapter.

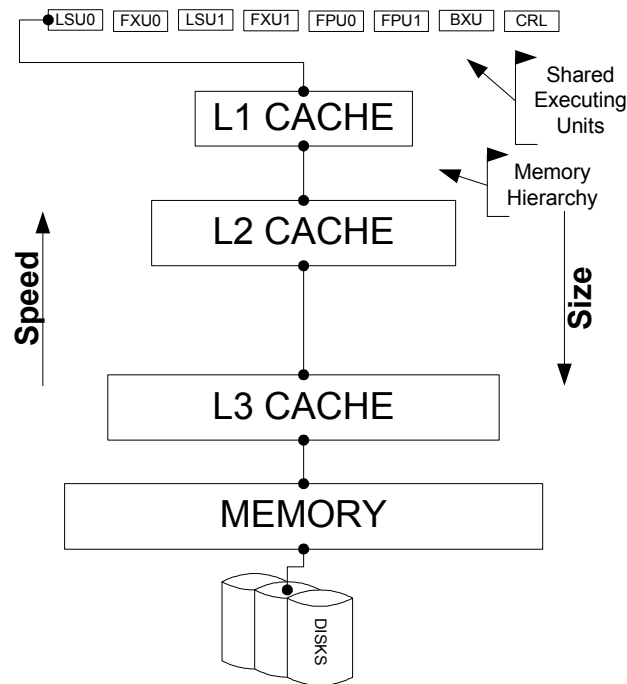


Figure 10-8 Memory heirarchy

10.4.1 Memory Hierarchy

Add text here (Body0).

10.4.2 L1, L2, and L3 caches

Add text here (Body0).

10.4.3 Translation Lookaside Buffer (TLB)

Add text here (Body0).

10.5 Optimization of critical sections in the code

This is a very extensive section and we do not attempt to cover all the possible techniques to optimize scientific and engineering applications. Instead we focus our attention on certain techniques that tend to benefit the POWER architecture. A good example of this is matrix multiplication. In the past, we managed to achieve approximately 60% of peak performance via matrix multiplication routines. Now with the advent of POWER5 that has more rename registers, it is feasible to achieve near peak performance with a well optimized routine.

In this section we try to illustrate some simple techniques using standard examples as well as using an example of matrix multiplication. We look at matrix multiplication from an over simplistic point of view, which does not have a good performance. However, we take this examples and provided a Fortran version version that compares well with highly optimized version of matrix multiplication routines.

However, prior to doing that, it is important to remind the reader of a series of general optimization rules that can improve the performance of critical sections of the code within scientific and engineering applications. To make better use of memory is important to keep in mind that whenever possible data should be access sequentially. In a Do loop, this is call accessing memory with unity stride or stride of 1. In large applications Do loops usually tend to grow in size or operation that are carried out within the loop. However, if possible, try to keep the Do loop small and maneagable, the smaller the Do loop the better. It is important to avoid expensive operations, such as divides, square-root, exponential, and others. For this type of transcendental functions, if they are required for the code consider using the MASS library which is explained in this chapter. In a loop, if statements and calls to subroutines tend to introduce data dependency and usually inhibit optimization. The following are less common problems and we just provide a listing:

- ▶ Avoid using EQUIVALENCE for critical variables
- ▶ Avoid implicit type conversions
- ▶ Try to reduce the number of arguments that are passed from the caller to the callee
- ▶ If multiple “if” statements are required, evaluate the most likely if-statement first. In other words, try to reduce the number if-statements that have to be evaluated.

In general, the key to performance is to be able to map the application as close as possible to the POWER Series architecture. The use of the POWER5 memory hierarchy can be cleverly manipulated in an algorithm to gain efficiency. This may include prefetching to facilitate accessing memory that is currently not in cache.

Prefetching provides a mechanism for hiding memory latency due to cache misses. We shall see that loop unrolling is very important to proper use of the memory hierarchy. Simulation of higher precision arithmetic helps performance. Specially when this process is highly repetitive.

10.5.1 Array Optimization

In order to take advantage of the memory hierarchy on the POWER Series systems, the first step requires understanding of the multiple arrays used in the code how their elements will be used. This is particularly true, in general, for numerically intensive applications that spend large amount of CPU performing repetitive tasks, such as loading and storing data from arrays.

Like in any popular Fortran or C textbook, we start by describing how elements in a matrix are stored differently between these two popular programming languages. For C codes, matrix rows are stored contiguously. For Fortran codes, matrix columns are stored contiguously.

4x4 Matrix with one-dimensional index in square-brackets:

$$\begin{bmatrix} a(1,1)[1] & a(1,2)[2] & a(1,3)[3] & a(1,4)[4] \\ a(2,1)[5] & a(2,2)[6] & a(2,3)[7] & a(2,4)[8] \\ a(3,1)[9] & a(3,2)[10] & a(3,3)[11] & a(3,4)[12] \\ a(4,1)[13] & a(4,2)[14] & a(4,3)[15] & a(4,4)[16] \end{bmatrix}$$

C: Row-major order, matrix rows are stored contiguously

$$[1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 14 \ 15 \ 16]$$

Fortran: Column-major order, matrix columns are stored contiguously

$$[1 \ 5 \ 9 \ 13 \ 2 \ 6 \ 10 \ 14 \ 3 \ 7 \ 11 \ 15 \ 4 \ 8 \ 12 \ 16]$$

Important: Keeping this ordering in mind will help the efficiency of your code

Next, we proceed to define *stride*, this is relevant when addressing elements of a matrix.

Stride Distance between successively accessed matrix elements in successive loop iterations

Example 10-13 Do loop with two different strides

```
do i = 1, 200
  do j = 1, 500
    a(i,j) = 1.d0    !stride 500
    b(j,i) = 1.d0    !stride 1
  enddo
enddo
```

Tip: Stride of 1 ensures sequential access to memory and provides best performance

Stride of 1 is beneficial because it ensures that once a line of cache has been loaded, the next set of elements needed will be available in the same cache line. This decreases the number of times that is required to go to memory hierarchy to load data. On POWER5 the size of a cache line is 128 bytes. The implication of this size is that for 32-bit words, strides larger than 32 will reduce performance (and 16 for 64-bit words) since only one element can be fetched per cache line. In addition, stride 1 will make use of the prefetch engine.

10.5.2 Loop Optimization

The technique of loop optimization is basic for taking advantage of memory hierarchy. A good example that is commonly used to illustrate multiple optimization techniques is matrix multiplication. Although optimizing the matrix multiplication triple-nested loop has been extensively discussed, here we only summarize how this is done and show results with respect to POWER5.

Example XXX shows the typical triple-nested loop of the main kernel in a matrix multiplication routine. The triple-nested loop comes from the following expression:

$$[C] = [C] + [A]^T [B]$$

Example 10-14 Matrix multiplication triple-nested loop

```

do i = 1, n
  do j = 1, n
    do k = 1, n
      c(i,j) = c(i,j) + a(j,k)*b(k,i)
    enddo
  enddo
enddo

```

This loop has been written without any particular attention to an optimal arrangement of the loop indices. In fact, optimizing all three loops for stride 1 may not be possible. This makes matrix multiplication a good candidate for loop unrolling.

Example 10-15 illustrates an implementation of a 2x2 loop unrolling.

Example 10-15 Matrix 2x2 loop unrolling

```

do i = 1, 1, 2
  do j = 1, m, 2
    s00 = zero
    s21 = zero
    s12 = zero
    s22 = zero
c
    do k = 1, n
      s11 = s11 + a(k,i)*b(k,j)
      s21 = s21 + a(k,i+1)*b(k,j)
      s12 = s12 + a(k,i)*b(k,j+1)
      s22 = s22 + a(k,i+1)*b(k,j+1)
    enddo
    c(i,j) = c(i,j) + s11
    c(i+1,j) = c(i+1,j) + s21
    c(i,j+1) = c(i,j+1) + s12
    c(i+1,j+1) = c(i+1,j+1) + s22
  enddo
enddo

```

In Table XYZ we show the level of unrolling that has been reported as optimal for different POWER series, including what we have found that appears to work well for POWER5.

Table 10-8 Optimal unrolling levels for some IBM POWER Series machines

| System | Unrolling Level |
|--------|-----------------|
| POWER1 | 2x2 |
| POWER2 | 4x4 |
| POWER3 | 4x4 |
| POWER4 | 5x4 |
| POWER5 | 4x4 |

The POWER1 had only a single floating-point unit, the 2x2 unrolling achieved very good performance. The POWER2 and POWER3 have dual floating-point unit, for these systems, 4x4 appeared to be favored. In the case of POWER4 the unrolling reported is 4x5.

In addition to unrolling, it is required to block matrices to be able to make use of memory hierarchy close to the registers as much as possible. Blocking the matrix multiplication procedure simply mean:

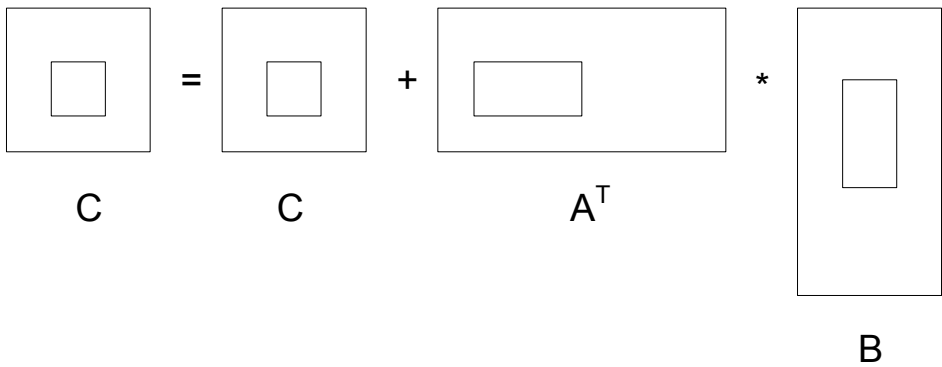


Figure 10-9 Blocking matrix diagram

In order to carry out our performance measurements, we used a hand tuned version of DGEMM. This version is not publically available. Table 10-9 illustrates the performance of the different level of unrolling. Column 1 is used as reference, it corresponds to a well coded DGEMM version without unrolling. The next set of

columns illustrate the impact of the various unrolling levels on POWER5. Notice that in all these cases the blocking size was kept constant (256). The only variable was unrolling. The main difference between 4x4 unrolling and p5-4x4 unrolling is in the use of two extra compiler directives in p5-4x4. One is to further unroll the inner most loop of the matrix multiplication and the second one is to enhance prefetching.

A simple inspection of Table 10-9 shows that 4x4 is the best suited technique for POWER5. p5-4x4 shows the additional benefit of using compiler directives. The recommended version for square matrices is p5-4x4.

Table 10-9 Performance of matrices with different unrolling levels.

| Dimension | No unrolling | 2X2 | 5X4 | 4X4 | 4X4 new |
|------------|--------------|-------------|-------------|-------------|-------------|
| 100 | 1580 | 3080 | 3938 | 4763 | 4902 |
| 200 | 1786 | 3583 | 4141 | 5340 | 5486 |
| 500 | 1797 | 3732 | 3906 | 5106 | 5691 |
| 1000 | 1848 | 3835 | 3522 | 5166 | 5944 |
| 2000 | 1812 | 3754 | 3731 | 5224 | 5837 |
| 5000 | 1355 | 3622 | 3809 | 5320 | 6028 |

Figure 10-9 illustrates all the different techniques that were utilized to unroll the matrix. In this case the reference is the peak performance for this POWER5 running at 1.65 GHz. Again, it is easy to see that p5-4x4 shows the best performance and it is remarkably close to peak performance. This is due to the fact that POWER5 introduces more rename registers.

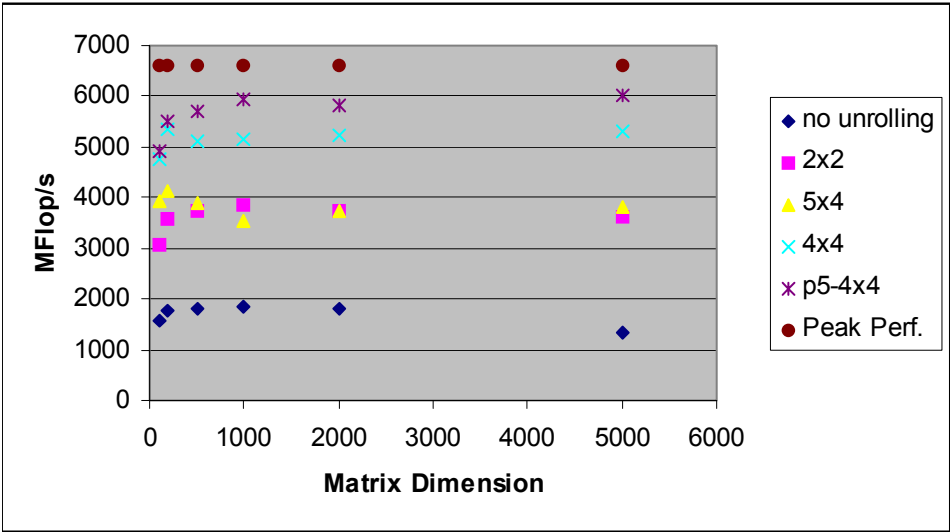


Figure 10-10 Graphical representation of the different techniques against peak performance.

Next we examine the effect of blocking and large pages on matrix multiplication. We looked at block sizes of 32, 64 96, 128, 160, 192, 224, and 256. Table 10-10 summarizes the performance of a matrix multiplication for a subset of the above sizes.

Table 10-10 Blocking and large pages effect on matrix multiplication.

| Dimension | 32 | 64 | 128 | 256 |
|-------------|------|------|------|------|
| 100 | | | | |
| Small pages | 4560 | 4596 | 3920 | 3795 |
| Large pages | 4588 | 4880 | 4917 | 4902 |
| 500 | | | | |
| Small pages | 4850 | 5080 | 5375 | 5363 |
| Large pages | 5030 | 5232 | 5679 | 5691 |

| Dimension | 32 | 64 | 128 | 256 |
|-------------|------|------|------|------|
| 1000 | | | | |
| Small pages | 4050 | 4751 | 5396 | 5609 |
| Large pages | 4391 | 5103 | 5843 | 5944 |
| 2000 | | | | |
| Small pages | 3629 | 4810 | 5249 | 5483 |
| Large pages | 4190 | 5365 | 5702 | 5837 |
| 5000 | | | | |
| Small pages | 3401 | 4498 | 5143 | 5350 |
| Large pages | 4068 | 5082 | 5906 | 6028 |
| 10000 | | | | |
| Small pages | 3448 | 4490 | 4923 | 5098 |
| Large pages | 4207 | 5372 | 5546 | 5900 |

Figure 10-10 illustrates the effect of different block sizes for three matrices with three different dimensions. The dimensions that we have selected for the three square matrices correspond to: 100x100, 1000x1000, and 10000x10000. From this plot we have identified that for all the matrices tested here, block sizes between 224 and 256 give the best results.

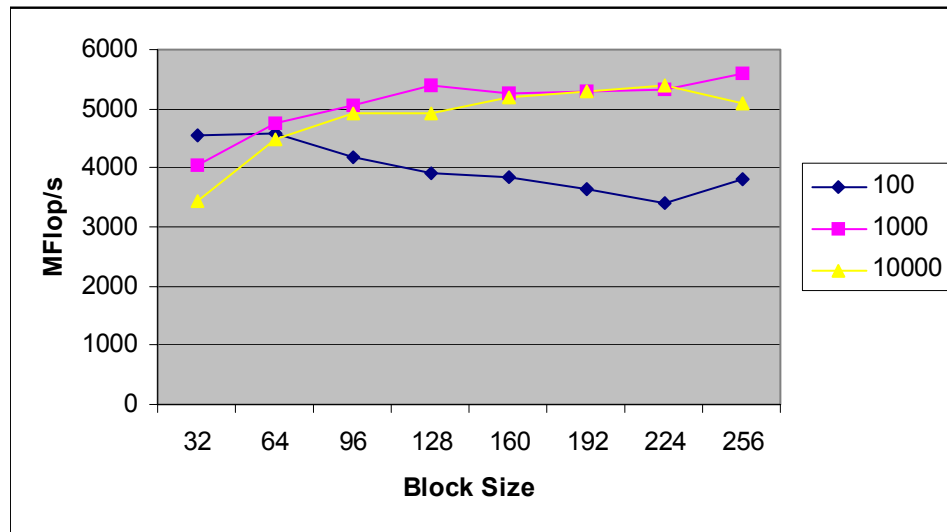


Figure 10-11 Matrix multiplication as a function of block size.

The last figure in this section illustrates the effect of large pages on matrix multiplication. Figure 10-11 shows that independently of the block size and dimensions of the matrix, the large pages effects is proportional to the dimensions of the matrices. Although not shown in the table nor in the figure, we found that the improvement measured in percentage difference (in going from small to large pages) for matrices: 100x100, 200x200, 500x500, 1000x1000, 2000x200, 5000x5000, and 10000x10000 are 1%, 2%, 4%, 8%, 15%, 20%, and 22%, respectively.

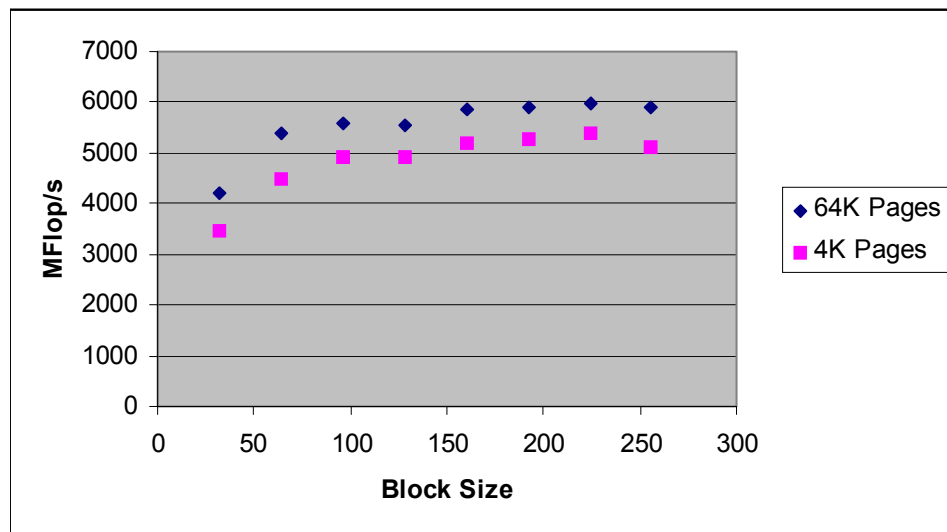


Figure 10-12 Large pages effect on performance for matrix multiplication.

10.5.3 Tuning for Arithmetic Operations

Add text here (Body0).

10.5.4 Multiple Loads and Prefetch

Add text here (Body0).

10.5.5 Divides

Add text here (Body0).

10.5.6 Floating Point-to-Integer Conversion

Add text here (Body0).

10.6 Optimized Libraries

IBM provides collections of routines that have been fully optimized to a particular architecture, in this case POWER5. As previously mentioned, the advantage of using highly tuned libraries is that in many cases the code requires very little changes to take full advantage of these library functions. In the next section we show how scientific applications can be customized to fully utilize these libraries. In this section we cover two of them:

- ▶ MASS library: <http://www-1.ibm.com/support/docview.wss?uid=swg24007650>
- ▶ ESSL library:
http://publib.boulder.ibm.com/clresctr/windows/public/esslbooks.html#essl_42

10.6.1 MASS Library

In the very first section of this chapter we presented a flowchart that illustrates how to analyze applications performance (a section is reproduced in Figure 10-12. As part of the steps in this flowchart, we mentioned that prior to attempting any hand tuning, the programmer should rely on compiler flags, compiler directives and *highly optimized libraries*. This chapter starts with the MASS libraries.

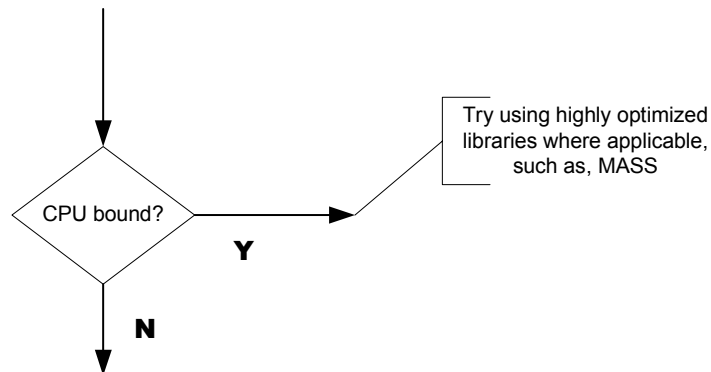


Figure 10-13 CPU-bound code might benefit from optimized libraries.

MASS is a set of libraries highly tuned mathematical intrinsic functions for C, C++, and Fortran applications that are optimized for for specific POWER architectures. The MASS library consists of the scalar and vector libraries. The MASS scalar library, *libmass.a*, contains an accelerated set of the most frequently used math intrinsic functions in the AIX system library *libxlf90.a*. This library can be used under AIX and Linux on POWER5 family of systems.

Important: In some cases MASS is not as accurate as the system library and it might handle certain cases differently. It is recommended to always check answers when using any kind of optimized libraries for the first time.

The second set of libraries correspond to the MASS vector library. The general vector library, *libmassv.a*, *libmassvp3.a*, and *libmassvp4.a* contain functions that be been tuned, in general, POWER3, and POWER4, respectively.

The vector libraries libmassv.a, libmassvp3.a, and libmassvp4.a can be used with either FORTRAN or C applications. When calling the library functions from C, only call by reference is supported, even for scalar arguments. As with the scalar functions, the vector functions must be called with the IEEE rounding mode set to round-to-nearest and with exceptions masked off. The accuracy of the vector functions is comparable to that of the corresponding scalar functions in libmass.a, though results may not be bit-wise identical.

The MASS vector Fortran source library enables application developers to write portable vector codes. The source library, libmassv.f, includes Fortran versions of all the vector functions in the MASS vector libraries. The following performance tables show that performance improvements are possible even when the MASS scalar functions are used in the vector loops of libmassv.f.

For information on performance for each of the MASS library functions please visit the the MASS library URL. Here we show how the vector MASS library can be implemented to a scientific application such as AMBER. This is illustrated in Example 10-16. In this example **sqrt** is replaced by **vrsqrt**.

Example 10-16 MASS vector library example

```

icount = 0
      do 25 j=i+1,natom
c
          xij = xi - x(3*j-2)
          yij = yi - x(3*j-1)
          zij = zi - x(3*j )
          r2 = xij*xij + yij*yij + zij*zij
          if( r2.gt.cut ) go to 25
c
          icount = icount + 1
          jj(icount) = j
          r2x(icount) = r2
c
      25  continue
c
c
c
#ifdef MASSLIB
      call vrsqrt( vectmp1, r2x, icount )
#else
      do j=1,icount
          vectmp1(j) = 1.d0/sqrt(r2x(j))
      end do
#endif

```

Table 10-11 illustrates the performance improvement by using the MASS libraries. The particular case presented in this table corresponds to the

Generalized Born myoglobin simulation. This protein has 2492 atoms, and is run with a 20A cutoff and a salt concentration of 0.2 M, with **nrespa=4** (long range forces computed every 4 steps.)

Table 10-11 AMBER7 performance with the **sqrt** vector MASS routine.

| Elapsed time in Seconds | |
|-------------------------|------------------|
| Without vector MASS | With vector MASS |
| 68 | 79 |

In this particular example, the MASS library are used in only three locations in the routine that is using most of the CPU time. One time for `exp()` and two for `sqrt()`. Thable 10-11 illustrates that by performing these simple substitutions there is an almost 15% improvement in single processor performance.

10.6.2 ESSL Library

In this section we look at performance improvements using highly optimized library routines. In this case the IBM Engineering and Scientific Subroutine Library (ESSL). more information can be found at the ESSL URL:
<http://publib.boulder.ibm.com/clresctr/windows/public/esslbooks.html>

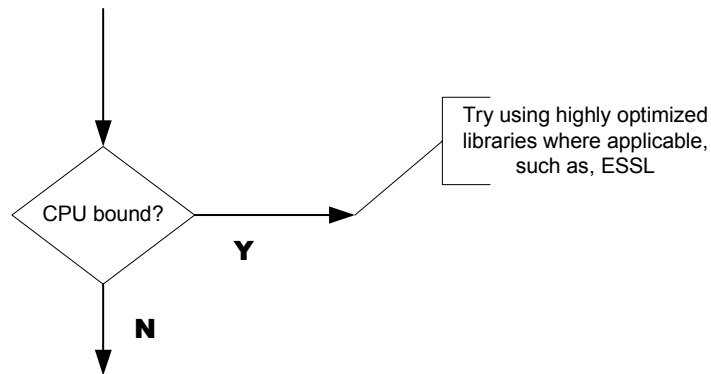


Figure 10-14 CPU-bound code might benefit from optimized libraries.

This is a state-of-the-art collection of mathematical routines. The ESSL family of subroutines for AIX and Linux contains:

- ▶ Basic Linear Algebra Subprograms (BLAS)
- ▶ Linear Algebraic Equations
- ▶ Eigensystem Analysis
- ▶ Fourier Transforms

To illustrate that ESSL provides the best performance we used ESSL version 4.2 available for AIX 5.3 on POWER5. This version requires XL Fortran Enterprise Edition Version 9.1 for AIX and the XL Fortran Enterprise Edition Run-Time Environment Version 9.1 for AIX. For this test we use DGEMM which has been extensively optimized for L1 and L2 caches. The results from ESSL are compared against the hand tuned Fortran version of DGEMM presented in previous sections. Table 10-11 summarizes the results for DGEMM ESSL and DGEMM Fortran.

Table 10-12 DGEMM routine optimized in the ESSL library.

| Dimension | Fortran | ESSL |
|-----------|---------|------|
| 100 | 4902 | 4403 |
| 200 | 5486 | 5494 |
| 500 | 5691 | 5893 |
| 1000 | 5944 | 6126 |
| 2000 | 5837 | 6140 |
| 5000 | 6028 | 6124 |
| 10000 | 5900 | 6105 |

These results are also summarized in Figure 10-10. All benchmarks were carried out on an IBM pSeries POWER5 with a clock speed of 1.65 GHz. The only case where the Fortran version has a slight advantage over ESSL is for small square matrices, in this case 100x100. In all the other cases the version of DGEMM in ESSL outperforms the Fortran version by as much as 5%.

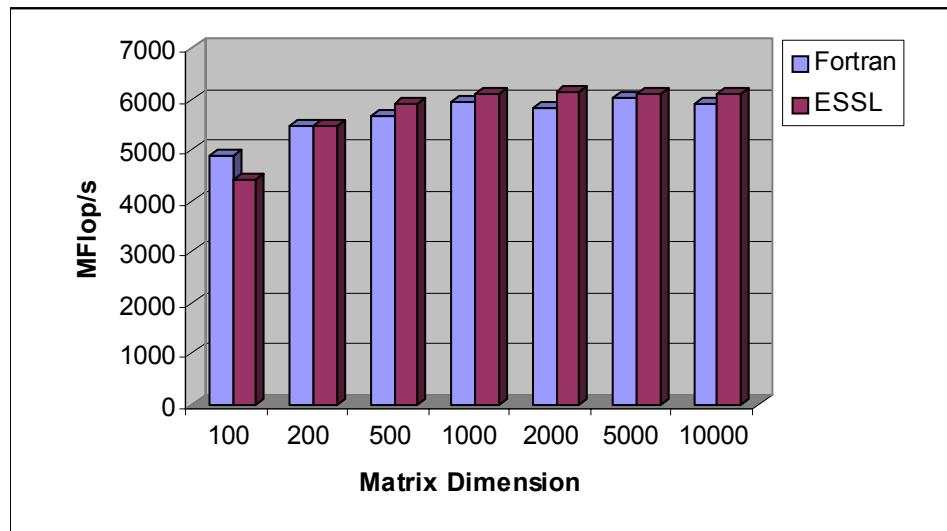


Figure 10-15 DGEMM optimized routine in ESSL versus Fortran version.

The ESSL DGEMM routine is consistently higher 90% of peak performance for matrices with rank higher than 500. Peak performance for this particular POWER5 is 6.6 GFlop/s.

10.7 Parallel Programming for Performance

Add text here (Body0).

10.7.1 General Concepts

Parallel computing involves dividing a task into smaller and more manageable blocks and distributing these blocks, in our case, among processors (physical or logical processors). In parallel computing, scientific and engineering applications are very important. They can take full advantage of a system with multiple processors, such as the IBM POWER5 server. In order to take full advantage of all the power of a system with many or few processors, it is necessary to consider the following issues:

- ▶ Parallel algorithms: To be able to make use of all the processors available on POWER5, we need algorithms that can be efficiently parallelized.
- ▶ Parallel languages: To implement a parallel algorithm, a parallel language is required. AIX and Linux on POWER5 support the most common parallel paradigms
- ▶ Parallel programming tools: This may involve evaluating performance of a particular application. Just like in the previous sections, this allows to answer questions such as how efficiently applications are taking advantage of the POWER5 architecture. In addition, parallel programming tools may involve interfaces that assist programmers debugging and shielding them from any low-level machine characterization.
- ▶ Parallel compiler programming: As compilers become more sophisticated and more information regarding the behavior of applications is put into the compiler, the more are the chances to provide programmers with automatic parallelization.

Metrics

In the case of parallel applications, in general, performance is a function of more parameters than in the case of serial applications. Performance not only depends on the characteristics of the input but basic considerations such as number of processors, memory, and processor communication are very important. The optimal combination of all these variables is what defines good scalability. In this book we define scalability as follows:

Scalability Is a measurement of how close an application performs proportionally to the number of processors and and it is expressed as parallel speedup.

To measure the parallel speedup, we rely on elapsed time which was defined previous sections. The parallel time is not just the cumulative time for the parallel regions but it is the elapsed time from beginning to end of the simulation. In effect, this definition includes sequential and parallel regions. In scientific and engineering applications this is the most practical way to define it. The parallel speedup is defined as follows:

Parallel speedup Is a measurement that reflects scalability or the ability of an application to reduce the time to solution proportionally to the number of processors.

$$S = \frac{T_{\text{sequential}}}{T_{\text{parallel}}}$$

and efficiency is simply:

$$e = \frac{S}{p}$$

where p is the number of processors.

10.7.2 Shared-Memory Parallel Performance

Add text here (Body0).

10.7.3 Distributed-Memory Parallel Performance

Add text here (Body0).

Sample level 4 heading (Head 4)

Add text here (Body0).

Sample level 5 heading (Head 5)

Add text here (Body0)



Sizing and Capacity Planning

Note to Author: Describe the chapter contents here using these or similar words.
Optionally add level 2 headings to a list using:
Special > Cross-Reference > Format: Head > Insert

This chapter provides/describes/discusses/contains ...

In this chapter we introduce/provide/describe/discuss ...

In this chapter:

In this chapter, the following topics are discussed/described:

This chapter provides/describes/discusses/contains the following:

- ▶ ...
- ▶ ...
- ▶ ...
- ▶ Sample level 2 “n.n” chapter heading
(created by **Special > Cross-Reference > Format: Head > Insert**)
- ▶ Sample next level 2 heading

11.1 Sample level 2 “n.n” chapter heading (Head 1) new page

Note to Author: The first level 2 “n.n” heading in a chapter should be the (Head 1) tag, skip to new page.

Add text here (Body0).

11.2 Sample level 2 “n.n” chapter heading (Head 2)

Add text here (Body0).

11.2.1 Sample level 3 “n.n.n” chapter heading (Head 3)

Add text here (Body0).

Sample level 4 heading (Head 4)

Add text here (Body0).

Sample level 5 heading (Head 5)

Add text here (Body0)



Part 5

Appendixes

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “New AIX/pSeries doc in eserver information center” on page 429. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *IBM @server pSeries Sizing and Capacity Planning*, SG24-7071
- ▶ *AIX 5L Differences Guide Version 5.3 Edition*, SG24-7463
- ▶ Logical Partitions on IBM PowerPC: A Guide to Working with LPAR on POWER5, SG24-8000
- ▶ AIX Logical Volume Manager, From A to Z: Introduction and Concepts, SG24-5432
- ▶ A Practical Guide for Resource Monitoring and Control (RMC), SG24-6606
- ▶ Managing AIX Server Farms, SG24-6606
- ▶ The Complete Partitioning Guide for IBM @server pSeries Servers, SG24-7039
- ▶ RS/6000 Scientific and Technical Computing: POWER3 Introduction and Tuning Guide, SG24-5155
- ▶ The POWER4 Processor Introduction and Tuning Guide, SG24-7041
- ▶ Understanding IBM @server pSeries Performance and Sizing, SG24-4810
- ▶ IBM @server pSeries Facts and Features, G320-9878
- ▶ *Advanced POWER Virtualization on IBM @server p5 Servers Introduction and Basic Configuration*, SG24-7940
- ▶ AIX 5L Performance Tools Handbook, SG24-6039

Other publications

These publications are also relevant as further information sources:

- ▶ *IBM Power5 Chip: A Dual-Core Multithreaded processor*, Ron Kalla, Balaram Sinharoy, Joel M. Tendler, IBM, *IEEE micro* March/April 2004 (Vol. 24, No. 2) pp 40-47
- ▶ *Performance workloads in a hardware multi threaded environment* by Bret Olszewski and Octavian F. Herescu
(<http://www.hpcacnf.org/hpca8/sites/caecw-02/s3p2.pdf>)
- ▶ *IBM @server POWER4 System Microarchitecture*, J. M. Tendler, J. S. Dodson, J. S. Fields, Jr., H. Le, and B. Sinharoy -- *IBM Journal of Research and Development* Vol. 46, No. 1, 2002, pp 5-26
- ▶ *Simultaneous Multithreading A Platform for Next-Generation Processors*, Susan J. Eggers, Joel S. Emer, Henry M. Levy, Jack L. Lo, Rebecca L. Stamm, Dean M. Tullsen, *IEEE micro* September/October 1997(Vol. 17, No. 5), pp12-19
- ▶ *Advanced Microprocessors* by Daniel Tabak
- ▶ *Electronic Service Agent for pSeries and RS/6000 User's Guide*, available at:
ftp://ftp.software.ibm.com/aix/service_agent_code/AIX/svcUG.pdf
- ▶ *Electronic Service Agent for pSeries Hardware Management Console User's Guide*, available at:
ftp://ftp.software.ibm.com/aix/service_agent_code/HMC/HMCSAUG.pdf
- ▶ *An Introduction to Virtualization*, Amit Singh, available at:
<http://www.kernelthread.com/publications/virtualization/index.html>
- ▶ *Evaluation of Multithreading Uniprocessors for Commercial Application Environments*, 23rd Annual International Symposium on Computer Architecturre, R. J. Eickemeyer, R. E. Johnson, S. R. Kunkel, M. S. Squillante, S. Liu, *IEEE*, May 1996, pp. 203-212.
- ▶ *Simultaneous Multithreading: Maximizing On-Chip Parallelism*, Proceedings 22nd International Symposium on Computer Architecture, D. M. Tullsen, S. J. Eggers, H. M. Levy, June 1995, pp. 392-403.
- ▶ *Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor*, 23rd Annual International Symposium on Computer Architecturre, D. M. Tullsen, S. J. Eggers, J. S. Emer, H. M. Levy, J. L. Lo, R. L. Stamm, *IEEE*, May 1996, pp. 191-202.
- ▶ *Initial Observations of the Simultaneous Multithreading Pentium 4 Processor*, Proceeding of the 12th International Conference on Parallel Architectures and Compilation Techniques, N. Tuck and D. M. Tullsen, *IEEE*, September 2003, pp. 26-35.

- ▶ AIX Support for Micro-Partitioning and SMT, L. Browning, available at:
http://www-1.ibm.com/servers/aix/whitepapers/aix_support.pdf
- ▶ Performance Workloads in a Hardware Multi Threaded Environment, B. Olszewski and O. F. Herescu, 5th Workshop on Computer Architecture Evaluation using Commercial Workloads, IEEE, February 2002.
<http://www.hpcaconf.org/hpca8/sites/caecw-02/s3p2.pdf>
- ▶ Indications for Simultaneous Multi-Threading from Workload Characterization, H. Mathis, A. Mericas, J. McCalpin, R. J. Eickemeyer, and S. Kunkel, to be submitted.
- ▶ C. P. Sosa, S. Behling and D. Barrs, Single Processor Tuning on the IBM POWER4 Processor, Education pSeries, IBM, Minneapolis.
http://www-1.ibm.com/servers/enablesite/education/eibp/p4_processor/index.html
- ▶ Gaussian 03, Revision C.01, M. J. Frisch, G. W. Trucks, H. B. Schlegel, G. E. Scuseria, M. A. Robb, J. R. Cheeseman, J. A. Montgomery, Jr., T. Vreven, K. N. Kudin, J. C. Burant, J. M. Millam, S. S. Iyengar, J. Tomasi, V. Barone, B. Mennucci, M. Cossi, G. Scalmani, N. Rega, G. A. Petersson, H. Nakatsuji, M. Hada, M. Ehara, K. Toyota, R. Fukuda, J. Hasegawa, M. Ishida, T. Nakajima, Y. Honda, O. Kitao, H. Nakai, M. Klene, X. Li, J. E. Knox, H. P. Hratchian, J. B. Cross, C. Adamo, J. Jaramillo, R. Gomperts, R. E. Stratmann, O. Yazyev, A. J. Austin, R. Cammi, C. Pomelli, J. W. Ochterski, P. Y. Ayala, K. Morokuma, G. A. Voth, P. Salvador, J. J. Dannenberg, V. G. Zakrzewski, S. Dapprich, A. D. Daniels, M. C. Strain, O. Farkas, D. K. Malick, A. D. Rabuck, K. Raghavachari, J. B. Foresman, J. V. Ortiz, Q. Cui, A. G. Baboul, S. Clifford, J. Cioslowski, B. B. Stefanov, G. Liu, A. Liashenko, P. Piskorz, I. Komaromi, R. L. Martin, D. J. Fox, T. Keith, M. A. Al-Laham, C. Y. Peng, A. Nanayakkara, M. Challacombe, P. M. W. Gill, B. Johnson, W. Chen, M. W. Wong, C. Gonzalez, and J. A. Pople, Gaussian, Inc., Wallingford CT, 2004. For additional information visit the Gaussian, Inc. official web site:
<http://www.gaussian.com>
- ▶ AE. Frisch and M. J. Frisch, Gaussian03 User's Reference, 2nd Edition, Gaussian Inc., Pittsburgh, PA
- ▶ W. J. Hehre, L. Radom, P. V. R. Schleyer, and J. A. Pople, Ab Initio Molecular Orbital Theory, John Wiley & Sons, New York, NY, 1985.
- ▶ D. A. Case, D. A. Pearlman, J. W. Caldwell, T. E. Cheatham III, J. Wang, W. S. Ross, C. Simmerling, T. Darden, K. M. Merz, R. V. Stanton, A. Cheng, J. J. Vincent, M. Crowley, V. Tsui, H. Gohlke, R. Radmer, Y. Duan, J. Pitera, I. Massova, P. A. Kollman, AMBER7 User's Manual, University of California San Francisco, CA. For more information on AMBER on IBM systems visit:
<http://www.msi.umn.edu/~cpsosa/ChemApps/MolMech/amber/amber.html>

- ▶ S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, J. Mol. Biol. 215, 403 (1990).
- ▶ J. Setubal and J. Meidanis, Introduction to Computational Biology, PWS Publishing Company, Boston, MA, 1997.
- ▶ Sanger Institute Human Genome Server:
http://www.ensembl.org/Homo_Sapiens/
- ▶ Fluent, Inc. For more information visit:
<http://www.fluent.com>
- ▶ The GNU Profiler by J. Fenlason and R. Stallman. For more information visit:
<http://www.gnu.org>

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ Simultaneous Multithreading Project
<http://www.cs.washington.edu/research/smt/index.html>
- ▶ AIX 5L operating system and related IBM products information
<http://www.ibm.com/servers/aix/>
- ▶ AIX toolkit for Linux applications
<http://www-1.ibm.com/servers/aix/products/aixos/linux/download.html>
- ▶ Application availability on the AIX 5L operating system (alphabetical listing and advanced search options for IBM software products and third-party software products)
<http://www-1.ibm.com/servers/aix/products/>
- ▶ CuOD process, brief explanation
<http://www-1.ibm.com/servers/eserver/pseries/cuod/tool.html>
- ▶ IBM AIX 5L Solution Developer Application Availability Web page
<http://www-1.ibm.com/servers/aix/isv/availability.html>
- ▶ IBM AIX: IBM Application Availability Guide Web page
<http://www-1.ibm.com/servers/aix/products/ibmsw/list>
- ▶ IBM Configurator for e-business (Business Partner site)
<http://xconfig.boulder.ibm.com/ssguide/announce/>
- ▶ IBM Configurator for e-business (IBM internal site)
<http://econfig.boulder.ibm.com/ssguide/announce/>

- ▶ IBM @server pSeries LPAR documentation and references Web site
<http://www-1.ibm.com/servers/eserver/pseries/lpar/resources.html>
- ▶ Linux for pSeries system guide
<http://www.ibm.com/servers/eserver/pseries/linux>
- ▶ Linux on pSeries information
<http://www.ibm.com/servers/eserver/pseries/linux/>
- ▶ Microcode Discovery Service information
<http://techsupport.services.ibm.com/server/aix.invsoutMDS>
- ▶ OpenSSH Web site
<http://www.openssh.com>

New AIX/pSeries doc in eserver information center

- ▶ *Using the Virtual I/O Server*
http://publib.boulder.ibm.com/infocenter/eserver/v1r2s/en_US/info/iphb1/iphb1.pdf
- ▶ *Virtual I/O Server and Partition Load Manager Commands Reference*
http://publib.boulder.ibm.com/infocenter/eserver/v1r2s/en_US/info/iphb1/commands/commands.pdf

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Symbols

/vdevice Open Firmware tree node 176

Numerics

3dmon command 72

A

Address Resolution Protocol (ARP) 204

Address translation 25–26

 effective to real address translation (ERAT) 54

Adjustable thread priorities 28

 effective use of 101

Affinity 131

AIX Version 5.2 60

AIX Version 5.3 3, 102, 127, 339–352

 3dmon command 72

 bindintcpu command 66

 chlvcopy command 81

 cpuinfo structure 63

 extendlv command 81

 extendvg command 81

 gprof command 71

 importvg command 81

 ioo command 81

 iostat command 63, 67, 287

 jtopas command 74

 Logical Volume Manager

 max_vg_pbuf_count 81

 pb_pbuf_count 81

 pervg_blocked_io_count 81

 total_vg_pbufs 81

 lparstat command 48, 56, 75, 274

 lslv command 81

 lspv command 81

 lvmo command 81

 mklvcopy command 81

 mkvg command 81

 mpstat command 76, 281

 Partition Load Manager (PLM) 83

 xlplm command 346

 xlpstat command 346

 perfwb command 76

 sar command 63, 67–68, 289

 sysconfig subroutine 289

 topas command 63, 68, 293

 trace command 69

 trcrpt command 69

 varyonvg command 81

 vmo command 79

 vmstat command 63, 67, 285

 xmperf command 73, 295

B

Berkeley Sockets 189

bindprocessor command 66

Branch History Tables (BHT) 14

C

Cache effects due to SMT 103

Capacity-On-Demand

 definition 2

Capacity-on-Demand 5

Capped partitions 136

chlvcopy command 81

Clock interrupts 159

Command/Response Queue (CRQ) 178

Condition Register (CR) 16

Congruence classes 20

Count Register (CTR) 16

D

Daemons polling for events 159

Decision support systems (DSS) 164

Decrementer 49

Dedicated Processor Partitions 126

Dedicated processor partitions 126

Dispatch Wheel 130

Dispatching and interrupt latencies 55

Dual Chip Modules (DCMs) 32

Dynamic power management 31

Dynamic resource balancing (DRB) 27, 93

E

Effective to Real Address Translation (ERAT) 54

Entitlement Expired 133
extendlv command 81
extendvg command 81

F

Fixed-Point Exception Register (XER) 16
FLloating Point Register (FPR) 16
Floating-Point Status and Control Register (FPSCR)
16

G

General Purpose Register (GPR) 16
gprof command 71
Guaranteed capacity 165

H

Hardware Page Table (HPT) 50
Harvested capacity 168
High Performance Computing (HPC) 149, 164
hyperthreading 64

I

I/O Server Command Line Interface (IOCLI) 181
i5/OS 41
IBM software products 428
IBM Subsystem Device Driver (SDD) 238, 260
IEEE 754 17
IEEE VLAN 186
importvg command 81
Instrucion cracking in POWER5 15
Internet Protocol (IP) layer 204
ioo command 81
iostat command 67, 287
iSeries Partition Licensed Internal Code (PLIC) 41

J

jtopas command 74

L

L1 caches 19–20
 cache line size 19
 data cache size 19
 instruction cache size 19
L2 cache 11, 20–21
 cache size 20
 congruence classes 20

 line size 20
L3 cache 11, 22–23
 cache line size 22
 cache size 22
Least-Recently-Used (LRU) replacement policy 19
Link Register (LR) 16
Linux 40, 99, 103
 on pSeries 429
Logical LAN Switch 186
Logical Processor 128
Logical processors 131
Logical Remote Direct Memory Access (LRDMA)
232, 252
lparstat command 48, 56, 75, 274
lslv command 81
lspv command 81
LTG 80
LVM 79
lvmo command 81

M

Machine State Register (MSR) 53
Memory affinity considerations 158
Memory considerations for the POWER Hypervisor
50
Memory Mapped I/O (MMIO) 53
Merged Logic DRAM (MLD) 19
Micro-Partitioning 5, 125–126
 capped partitions 136
 dedicated processor partitions 126
 implementation 128
 Processor Utilization Resource Register (PURR)
132
 shared processor partitions 127
Micro-partitioning 126
Micro-Partitions 126
mklvcopy command 81
mkvg command 81
mpstat command 76, 281
Multi-Chip Modules (MCMs) 32
Multi-path I/O 237, 260
Multithreading technology 10

N

Neighbor Discovery Protocol 204
Not-Runnable 133

O

Open Firmware 40, 176, 231, 251
 OSI-Layer 2 185
 OSI-layer 2 204

P

Page Frame Table 43
 Partition Load Manager (PLM) 83
 xlplm command 346
 xlpstat command 346
 Partitioning on the IBM Eserver p5 126
 Performance tools
 3dmon command 72
 gprof command 71
 ioo command 81
 iostat command 67, 287
 jtopas command 74
 lparstat command 75, 274
 lvmo command 81
 mpstat command 76, 281
 perfwb command 76
 sar command 67–68, 289
 topas command 68, 293
 trace command 69
 trcrpt command 69
 vmo command 79
 vmstat command 67, 285
 xmperf command 73, 295
 Performance considerations
 bottlenecks 4
 Hypervisor 55
 virtual Ethernet 188
 virtual SCSI 238, 261
 virtual I/O Server 205
 Performance Monitor Support 45
 Performance tuning 2
 memory affinity considerations 158
 rPerf numbers 166
 traditional approach 5
 perfwb command 76
 Phantom Interrupt 139
 Physical Processor 128
 Planned over-commit 169
 POSIX 288
 POWER Hypervisor ??–57, 149
 cache invalidations 54
 call functions 43
 debugger support 44

 design of 52
 dispatching algorithm 54
 dispatching and interrupt latencies 55
 dump support 44
 extensions for Micro-Partitioning 49
 Machine Check Interrupt 42
 memory considerations 50
 memory migration support 44
 Page Frame Table 43
 performance considerations 55
 performance monitor support 45
 saved and restored registers 52
 support 41
 System (Hypervisor) Call Interrupt 42
 System Reset Interrupt 42
 table of calls 45
 virtual I/O support 50
 virtual terminal support 44
 POWER4 10, 53–54, 92, 106
 POWER5
 address translation 25–26
 architecture ??–26
 Branch History Tables (BHT) 14
 branch prediction 14
 cache comparison 25
 cache invalidations 54
 chip design 12
 CR 16
 CTR 16
 Decrementer 49
 Dual Chip Modules (DCMs) 32
 dynamic power management 31
 exceptions 18
 FPRs 16
 FPSCR 16
 FPU 11
 FXU 11
 Global Completion Table (GCT) 15
 GPRs 16
 group dispatch 15
 IDU 11
 IFU 11
 instruction cracking 15
 instruction decoding and preprocessing 15
 instruction execution 17
 Instruction Fetch Address Register (IFAR) 13
 Instruction Fetch Buffers (IFBs) 15
 instruction fetching 13
 instruction pipelines 13

- instruction size 13
 - issue queues 15, 17
 - ISU 11
 - L1 caches 19–20
 - cache line size 19
 - data cache size 19
 - instruction cache size 19
 - L2 cache 11, 20–21
 - cache size 20
 - congruence classes 20
 - line size 20
 - L3 cache 11, 22–23
 - cache line size 22
 - cache size 22
 - link stack 15
 - load reorder queue 15
 - LR 16
 - LSU 11
 - Machine State Register (MSR) 53
 - maximum outstanding branches 15
 - MC 11
 - Merged Logic DRAM (MLD) 19
 - Multi-Chip Modules (MCMs) 32
 - multithreading technology 10
 - Processor Utilization Resource Register (PURR) 132
 - register renaming 16
 - rPerf numbers 166
 - store reorder queue 15
 - Thread Status Register (TSR) 28, 95
 - Time Base register 49
 - total execution units 17
 - XER 16
 - POWER5 Hypervisor 181
 - PowerPC 15, 53
 - PowerPC AS architecture (Version 2.02) 10
 - Preemption of a virtual processor 52
 - Processing Unit 128
 - Processing unit 128
 - Processor Pools 129
 - Processor Utilization Resource Register 132
 - Processor Utilization Resource Register (PURR) 99, 132
 - PURR 64
- R**
- Redbooks Web site 429
 - Contact us xxii
- Register renaming in POWER5 16
 - Remote DMA 178
 - Resource Management and Control (RMC) 84, 338
 - daemon 84, 338
 - rPerf numbers 166
 - Runnable 133
 - Running 133
 - Run-Time Abstraction Services (RTAS) 40
- S**
- sar command 67–68, 289
 - SCSI Remote DMA Protocol (SRP) 233, 254
 - Server consolidation 144
 - Server provisioning 144
 - Shared Ethernet adapter 203
 - Shared processor partitions 127
 - guidelines 164
 - Silicon-on-insulator (SOI) 10
 - Simultaneous Multithreading (SMT) 10, 89, 146
 - adjustable thread priorities 27
 - cache effects 103
 - definition 2
 - dynamic resource balancing 27
 - dynamic switching 146
 - effective use of adjustable thread priorities 101
 - implementation 92
 - interrupts 101
 - performance benefits 107
 - process accounting 99
 - processor utilization 100
 - scheduling 100
 - snooze and snooze delay 98
 - software considerations 98
 - slow ticks 159
 - SMP locking 152
 - SVG 79
 - sysconfig subroutine 289
 - System Licensed Internal Code (SLIC) 41
- T**
- Technology Independent Machine Interface (TIMI) 41
 - third-party software products 428
 - Time Base 49
 - topas command 68, 293
 - TotalStorage® Enterprise Storage Subsystem™ (ESS) 238, 260
 - trace command 69

Translation Control Entry (TCE) 44, 178
Translation Lookaside Buffer (TLB) 54
trcrpt command 69

U

Uncapped partitions 137

V

Variable capacity weight 139
varyonvg command 81
VGDA 79
VGSA 79
Virtual adapters 203
Virtual Ethernet 181
 bridge implementation 203
 comparing virtual to physical 183
 Hypervisor switch implementation 185
 IEEE 802.1Q support 183
 implementation guidelines 202
 IPv6 support 184
 MAC address 186
 MTU sizes 184
 performance considerations 188
 shared Ethernet adapter functionality 203
 trunk adapter 186
Virtual I/O Server 180
 implementation guidelines 213
 performance results 206
Virtual Input/Output 50–180
 Hypervisor relationship 174
 infrastructure 176
 Remote DMA 178
 types of connections 178
 virtual Ethernet 181
 virtual SCSI 225, 249
Virtual Processor 128
Virtual Processor State 133
Virtual processors 49, 152
 preemption of 52
 states of 133
Virtual SCSI 225, 249
 adapters 225, 257
 command tag queueing 236, 258
 configuration example 246, 268
 emulated DASD 236, 257
 flow of data 234, 256
 interpartition communication 229, 251
 LVM mirroring 237, 259

memory descriptor mapping 234, 254
model overview 255
multi-path I/O 237, 260
performance considerations 238, 261
redundant configurations 236, 259
Reliable Command/Response Transport 231, 252
SCSI Remote DMA Protocol (SRP) 233, 254
server sizing 244, 267
structure and concepts 226, 250
Virtual server farms 144
Virtualization ??–248, ??–270
 application considerations for shared processor partitions 160
 cache architecture 148
 dynamic switching 146
 effect of SMT on processor usage 147
 effects of cache-friendly and cache-unfriendly applications 151
 idle partition overhead 158
 memory affinity considerations 158
 number of virtual processors 148
 SMP locking 152
vmo command 79
vmstat command 67, 285

W

WebSM 350
Weight for uncapped partitions 139
Workload Management (WLM) 5

X

xlplm command 346
xlpstat command 346
xmperf command 73, 295



Advanced POWER Virtualization on IBM *e*server p5 Servers

Architecture and Performance Considerations



The POWER5 Architecture

This Redbook addresses the issues surrounding performance of the new IBM eServer p5 systems. Many features have been introduced in these systems and are discussed in this book.

Simultaneous Multi-threading Performance

The new systems use the POWER5 processors. These processors consist of a dual processor core with each core supporting two hardware threads of execution. This technology is referred to as Simultaneous Multi-Threading (SMT). This Redbook provides an indepth look into the POWER5 processor architecture.

Virtual I/O performance

The POWER Hypervisor has been enhanced in these new systems. In this Redbook a detailed description is provided of the Hypervisor and Hypervisor system calls.

Introduced also in these systems is the concept of virtual processors, virtual storage and virtual networking. This Redbook takes an indepth look into these areas.

With all these new technologies and features, it becomes necessary for the system administrator to realize the performance impact that may be present. This Redbook takes a look at these performance issues by providing results of performance tests that were conducted.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(--->Hide:-->Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine.frm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.

IBM CONFIDENTIAL - Draft Document December 10, 2004 9:42 pm

5768_Spine.frm 439



Architecture and Performance Considerations in POWER Virtualization

(1.5" spine)
1.5"<=> 1.998"
789 <=> 1051 pages



Architecture and Performance Considerations in POWER

(1.0" spine)
0.875"<=>1.498"
460 <=> 788 pages



Architecture and Performance Considerations in POWER

(0.5" spine)
0.475"<=>0.875"
250 <=> 459 pages



Architecture and Performance Considerations in POWER

(0.2" spine)
0.17"<=>0.473"
90 <=> 249 pages

(0.1" spine)
0.1"<=>0.169"
53 <=> 89 pages

To determine the spine width of a book, you divide the paper PPI into the number of pages in the book. An example is a 250 page book using Plainfield opaque 50# smooth which has a PPI of 526. Divided 250 by 526 which equals a spine width of .4752". In this case, you would use the .5" spine. Now select the Spine width for the book and hide the others: **Special>Conditional Text>Show/Hide>SpineSize(-->Hide:)>Set** . Move the changed Conditional text settings to all files in your book by opening the book file with the spine.frm still open and **File>Import>Formats** the Conditional Text Settings (ONLY!) to the book files.

IBM CONFIDENTIAL - Draft Document December 10, 2004 9:42 pm

5768_Spine.frm 440



Redbooks

Architecture and Performance Considerations in

(2.5" spine)
2.5"<->nnn.n"
1315<-> nnn pages



Redbooks

Architecture and Performance Considerations in

(2.0" spine)
2.0"<-> 2.498"
1052 <-> 1314 pages