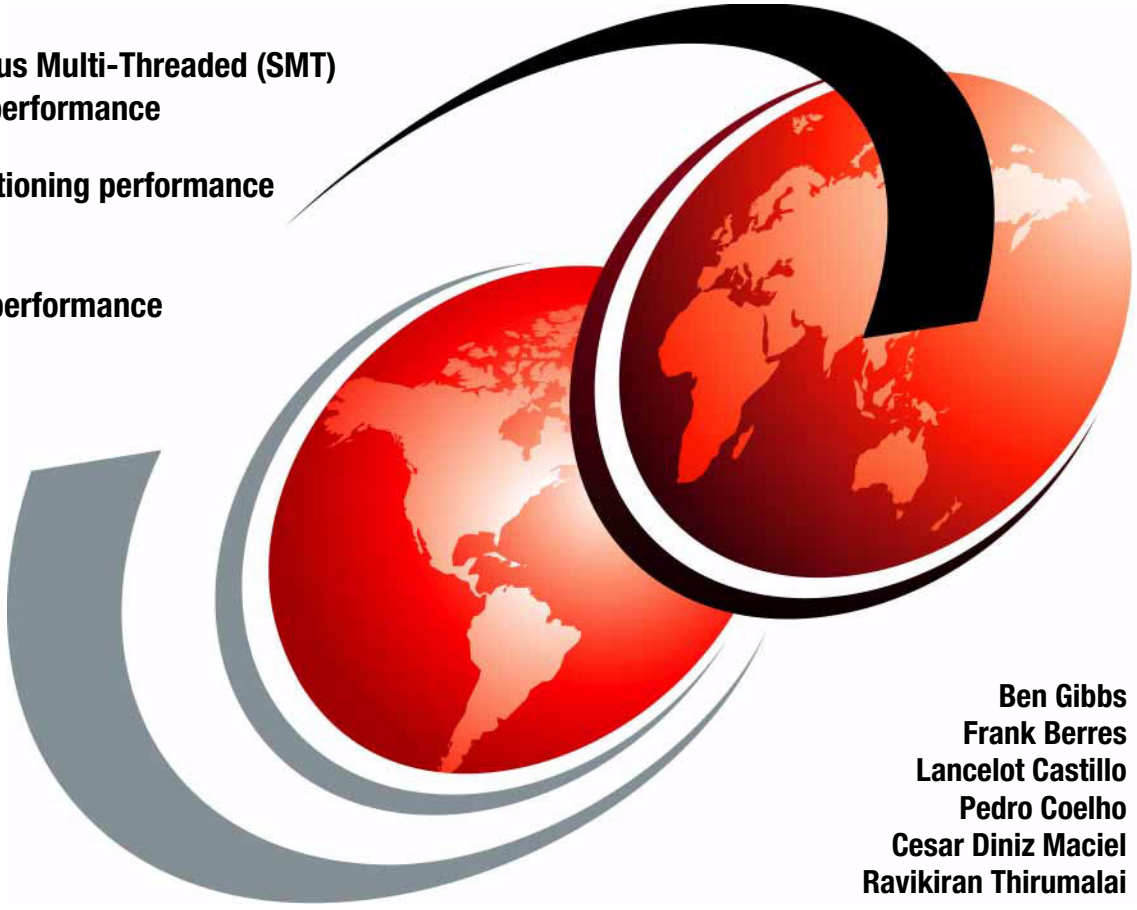
Draft Document for Review July 31, 2004 4:38 am

SG24-5768-00

IBM @server p5 Virtualization Performance Considerations

- Simultaneous Multi-Threaded (SMT) processor performance
- Micro-partitioning performance
- Virtual I/O performance



Ben Gibbs
Frank Berres
Lancelot Castillo
Pedro Coelho
Cesar Diniz Maciel
Ravikiran Thirumalai

Redbooks

ibm.com/redbooks

International Technical Support Organization

Performance Considerations in a Shared Processor LPAR Environment

July 2004

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

First Edition (July 2004)

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

Note: This book is based on a pre-GA version of a product and may not apply when the product becomes generally available. We recommend that you consult the product documentation or follow-on versions of this redbook for more current information.

This document created or updated on July 31, 2004.

Contents

Figures

Tables

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®
AIX 5L™
@server®
@server®
eServer™
HiperSockets™
IBM®
ibm.com®
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 team that wrote this redbook

This redbook was produced by a team 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 sytem in November of 1989. His areas of expertise include performance analysis and tuning, operating system internals, and device driver

development for the AIX operating sytem. He was the project leader for this IBM Redbook.

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.

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.

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.

The team from left to right: Lance Castillo, Cesar Maciel, Pedro Coelho, Frank Berres, Ravikiran Thirumalai, and Ben Gibbs

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

Jorge D Rodriguez
IBM Austin

Luc Smolders
IBM Austin

Luke Browning
IBM Austin

Octavian F. Herescu
IBM Austin

Kiet H. Laml
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

1

Introduction

What is *Virtualization*? Virtualization is the pooling of information technology (IT) resources in a way that shields the physical nature and boundaries of those resources from users.

Why would this be important to me? There are several reasons and some of those are:

- Reduce costs by increasing asset utilization.
- Re-deploy talent to manage your business, not the infrastructure.
- Rapidly provision new servers.
- Drive new levels of IT staff productivity.
- Simplify server management and operations.
- Communicate more securely with Virtual Ethernet.

The IBM eServer p5 family of servers include powerful new capabilities with Virtualization Engine system technology such as partioning processors to 1/10th of a CPU, share processor resources in a pool to drive up utilization, share physical disk storage and communications adapters between partitions and take advantage of corss partition workload management.

In the subsequent chapters you will find a indepth discussion about each component that makes up the IBM Virtualization Engine and some performance

issues that one must take into consideration. Some of the terminology and topics include:

POWER5 The POWER5 processor is IBMs latest 64-bit implementation of the PowerPC AS architecture (Version 2.02). It 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.

POWER Hypervisor Supports partitioning and dynamic resource movement across multiple operating system environments.

Micro-Partitioning™ Enables you to allocate the utilization of a physical processor to the logical partition.

Virtual LAN Provide network virtualization capabilities that allow you to prioritize traffic on shared networks.

Virtual I/O Provide the ability to dedicate I./O adapters and devices to a virtual server, allowing the on-demand allocation and management of I/O devices.

Capacity-On-Demand Allows system resources to increase overall resource utilization by virtualizing multiple physical CPUs through the use of multi-threading.

Simultaneous Multi-Threading (SMT) Allows applications to increase overall resource utilization by virtualizing multiple CPUs through the use of mult-threading.

1.1 Performance overview

Performance tuning is not straight-forward and should be looked upon as a complex task. It requires a great deal of discipline and exactness. Unless appropriate tests are used to identify specific bottlenecks within the system it is difficult to interpret any results. Occasionally performance tuning can be very frustrating and tedious at times especially when after a great deal of analysis the results are still inconclusive. Nevertheless, performance tuning can be very rewarding and provide long term benefits.

An IBM eServer p5 system is subjected to various loads from the partitions. 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.

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.

1.2 Understanding server performance in general

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 they each 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.

These resources comprise the fundamental subsystems found within a server:

- Central Processing Unit(s) (CPUs)
- Memory
- Disk I/O
- LAN I/O
- Controlling software

Obviously the controlling software in this case is AIX Version 5.3. Efficiency of the controlling software will maximize the hardware performance. As server performance is distributed throughout each server component it is essential to identify the most important factor(s), for example 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:

1. Server hardware configuration
2. Application software workload
3. Operating system configuration parameters
4. LAN Server software parameters

File servers need fast LAN adapters and fast disk subsystems. In contrast, database server environments typically produce high CPU and disk utilization requiring fast processors or multiple processors and fast disk subsystems. Both file and database servers require large amounts of memory for operating system caching.

1.3 Traditional approach

Traditionally there was a simplified approach to performance tuning. If the bottleneck is the processor then 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 cause excessive disk I/O by the operating system. Memory bottlenecks can also cause high disk utilization, if there is insufficient memory available or disk caching algorithms are ineffective for a particular application workload. If the bottleneck is the disk subsystem then either additional disks and/or disk adapters can be installed, or a specialized high performance disk subsystem could be used. If the bottleneck is the LAN adapter then a faster LAN interface could be installed. Another optimization technique that can be employed is to utilize multiple LAN 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 chose the best set of options it is vital to ensure that there is a balance between them. The trade-offs are:

Cost vs Performance

There are situations where the only way performance can be improved is by using more or faster hardware.

Conflicting Performance Requirements

When more than one application is used simultaneously, there may be conflicting performance requirements.

Speed vs 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

1.4 Micro-partitioning as a game changer

Micro-partioning changes the way we play the game. We still follow the same rules as to identifying existing or potential bottlenecks, but the remedy can be different. 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 the following chapters, we will look at each of the components that make up Virtualization in detail. These components are:

- POWER5 architecture
- AIX 5L Version 5.3 support
- Simultaneous Multi-Threading
- Virtual I/O
- Virtual Ethernet
- Virtual SCSI

2

Chapter 2.

Hardware and software components

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:

- POWER5 processor
- POWER Hypervisor™
- Operating systems

2.1 POWER5

The POWER5 processor is IBMs latest 64-bit implementation of the PowerPC AS architecture (Version 2.02). It 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 boast of a deeply pipelined design with 16 stages for fixed-point register-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 instructions and results. There has been a specific focus on storage latency management; the POWER5 can issue out of order and speculative load operations with support for up to eight outstanding L1 data cache line misses, hardware or software initiated instruction prefetching from L2, L3 and memory, hardware initiated data stream prefetching, and software instruction prefetching based on branch prediction hints. The POWER5 is built over the POWER4 base and maintains binary and structural compatibility with POWER4. The identical pipeline structure lets optimizations designed for POWER4 designs work equally well on POWER5 based systems.

2.2 POWER5 chip overview

The IBM POWER5 chip is a dual core multithreaded processor. It 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 -- resulting in reduced delays in wire dominated chip timing paths. The chip is implemented using 130 nm lithography; the chip uses eight metal levels and the die measures 389 mm^2. The chip is made up of 276 million transistors.

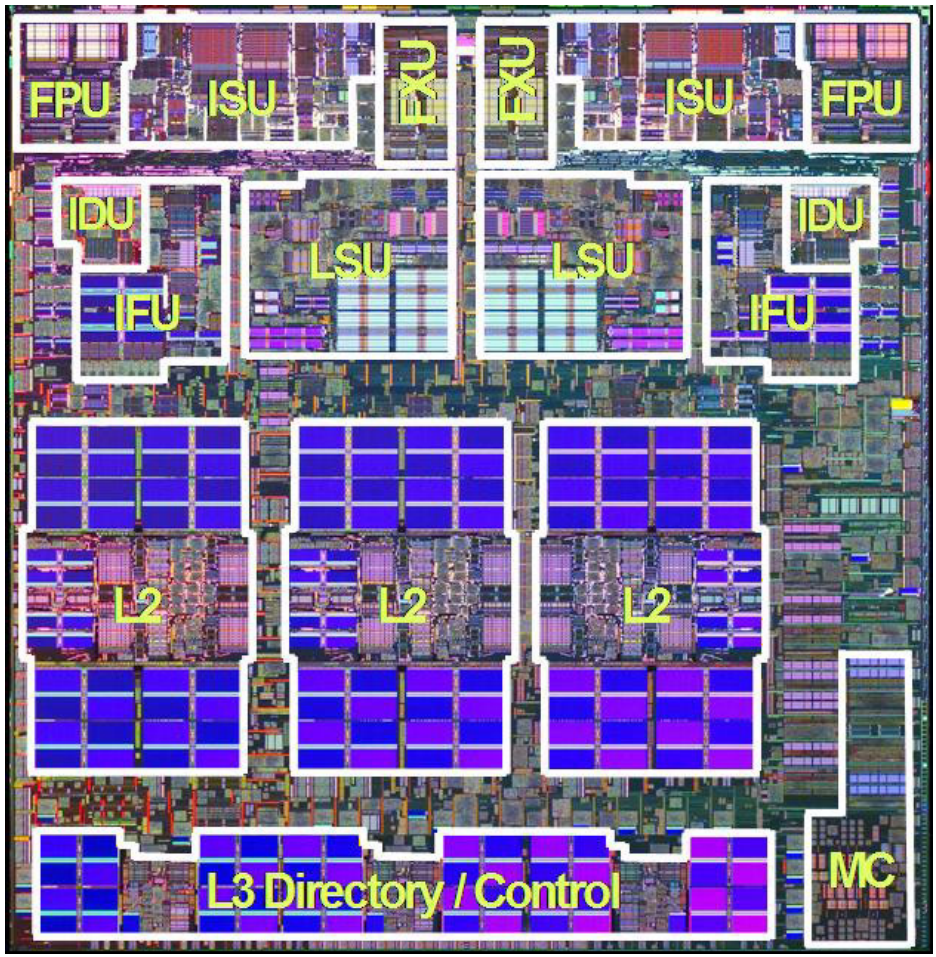


Figure 2-1 POWER5 processor chip

List of abbreviations used in Figure on page 9

- FXU -- Fixed point Execution Unit
- ISU -- Instruction Sequencing Unit
- IDU -- Instruction Decoding Unit
- LSU -- Load Store Unit
- IFU -- Instruction Fetch Unit
- FPU -- Floating Point Unit
- MC -- Memory Controller

Picture taken from IEEE paper on power5.

2.2.1 Chip organization

A single POWER5 die (chip) contains two identical processor cores. The POWER5 design implements two-way *Simultaneous Multithreading (SMT)* on *each* of the two processor cores. Figure 2-2 shows the high level structure of POWER5 based systems. Because of the dual core design, and two SMT hardware threads per core, a single POWER5 chip appears as a four cpu SMP system for the operating system.

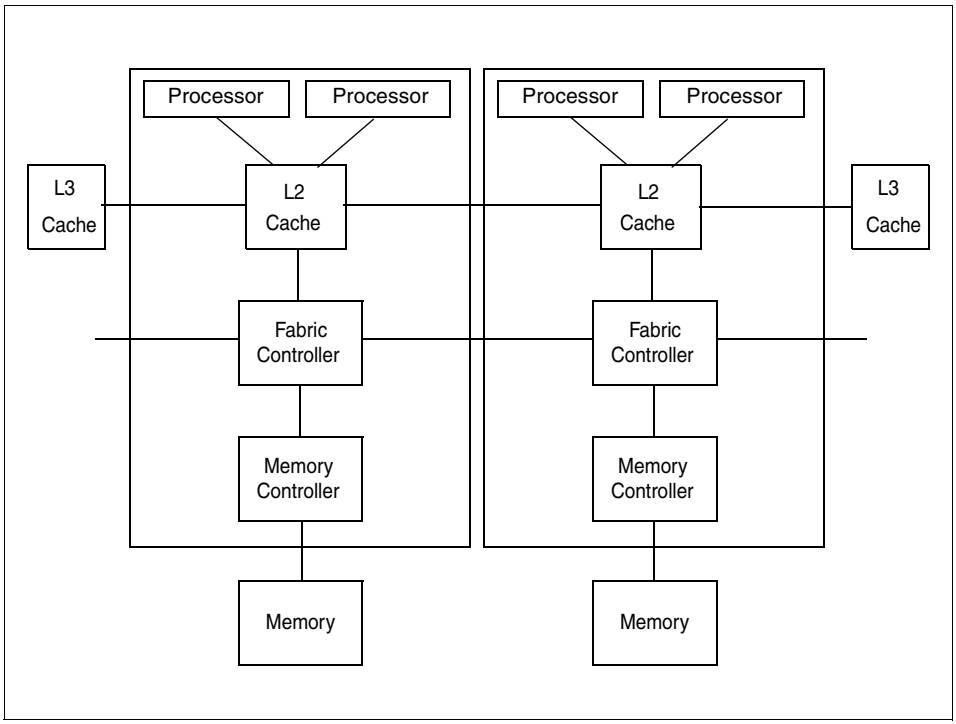


Figure 2-2 High level structure of POWER5

SMT is a multithreading[1] concept, which can greatly improve utilization of the processor's hardware resources, resulting in better software performance. While superscalar processors can issue multiple instructions in a single cycle from a single hardware thread, SMT processors can issue multiple instructions from multiple hardware threads in a single cycle. POWER5 provides for two hardware

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

threads per processor core. Hence, multiple instructions from both the hardware threads can be issued in a single processor cycle on the POWER5. POWER5 SMT will be discussed briefly later in this chapter.

L1 cache

Each processor core has separate 64 KB L1 instruction cache and a 32 KB L1 data cache. The L1 cache is shared by the two hardware threads of the processor 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 2-1 on page 11 lists the cache characteristics of the POWER5 processor architecture.

Table 2-1 Cache characteristics of the POWER5 processor

Cache characteristic	**L1 Instruction cache**	**L1 Data cache**	**L2 cache**	**L3 cache**
Data type	Instructions only	Data only	Instructions and Data	Instructions and Data
Size	64KB	32 KB	1.88MB	36MB
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 (2 * 128 B -- sectored)
Index	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 data cache an instruction cache	*Not* inclusive of L2 cache (victim cache of L2)
Hardware Coherency	Yes	Yes	Yes (separate snoop ports)	Yes (separate snoop ports)

Cache characteristic	L1 Instruction cache	L1 Data cache	L2 cache	L3 cache
Store policy	N/A	Store through. No allocate on store miss	Store back. Allocate on store miss	Store back

The L1 instruction cache is 2-way set associative with LRU replacement policy. L1 Instruction cache is also kept coherent with the L2 cache. The L1 instruction cache is indexed using the effective address bits. The L1 data cache is 4-way set associative with LRU replacement policy. Effective address is used to index into the L1 data cache also. The L1 data cache is a *store-through* design.

L2 cache

The POWER5 L2 cache is accessed by both the cores of the chip. It maintains full hardware coherence within the system and can supply intervention data to cores on other POWER5 chips. L2 is an in-line cache. Unlike L1s which are store-through, L2 is a store-in cache. It is fully inclusive of the two L1 data caches and L1 instruction caches (one L1 data and instruction cache per core). The 1.88 MB L2 is physically implemented in 3 slices, each of 640 KB size. Hashing is used to select one of the three slices for a given physical address. Each L2 slice is again comprised of 512 associative sets called congruence classes. Each congruence class contains 10 128 B cachelines. Real address bits 48-56 are used in the 9 bit congruence class address. Physical tag comparison (real address bits 14-47) are used to determine if the desired cacheline is resident in a set. Each of these 3 slices have separate L2 cache controllers. Either processor core of the chip can independently access each L2 controller.

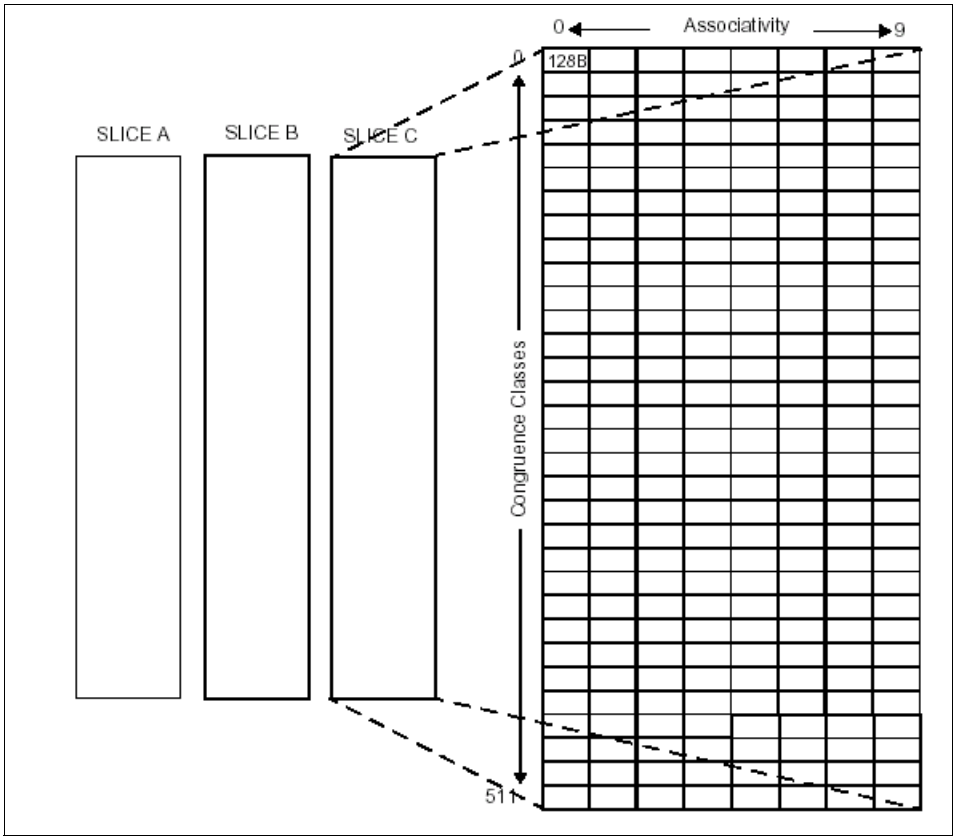


Figure 2-3 L2 cache organization

L3 cache

L3 is a unified 36MB cache accessed by both cores on the POWER5 processor chip. It maintains full hardware coherence with the system and can supply intervention data to cores on other POWER5 processor chips. Actually, L3 is a victim cache of the L2 -- that is, all valid cachelines evicted out of the L2 due to associativity (victimized) will be cast out to L3. The L3 is not inclusive of L2; the same line will never reside in both L2 and L3 at the same time. The L3 cache is implemented off-chip as a separate 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.

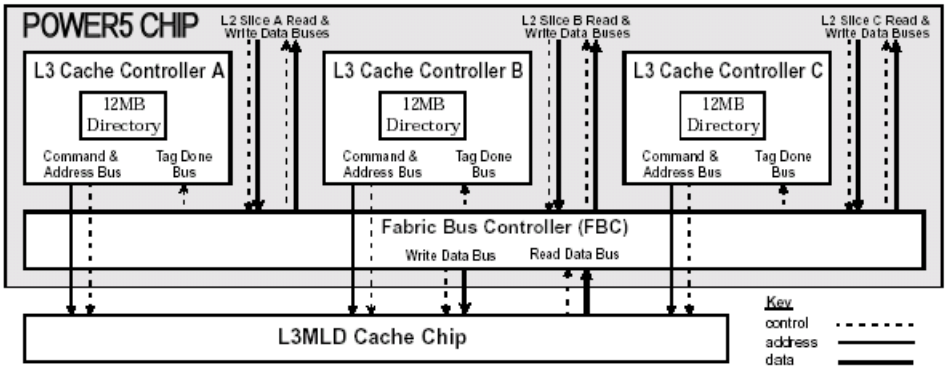


Figure 2-4 L3 Controller and MLD -- high level block diagram

The 36 MB L3 is split into three identical 12 MB slices on the L3 MLD cache chip. The same hashing algorithm used to select the L2 slices is used to select the L3 slices, for a given physical address. An L3 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 2-5 on page 15 depicts the L3 cache organization graphically.

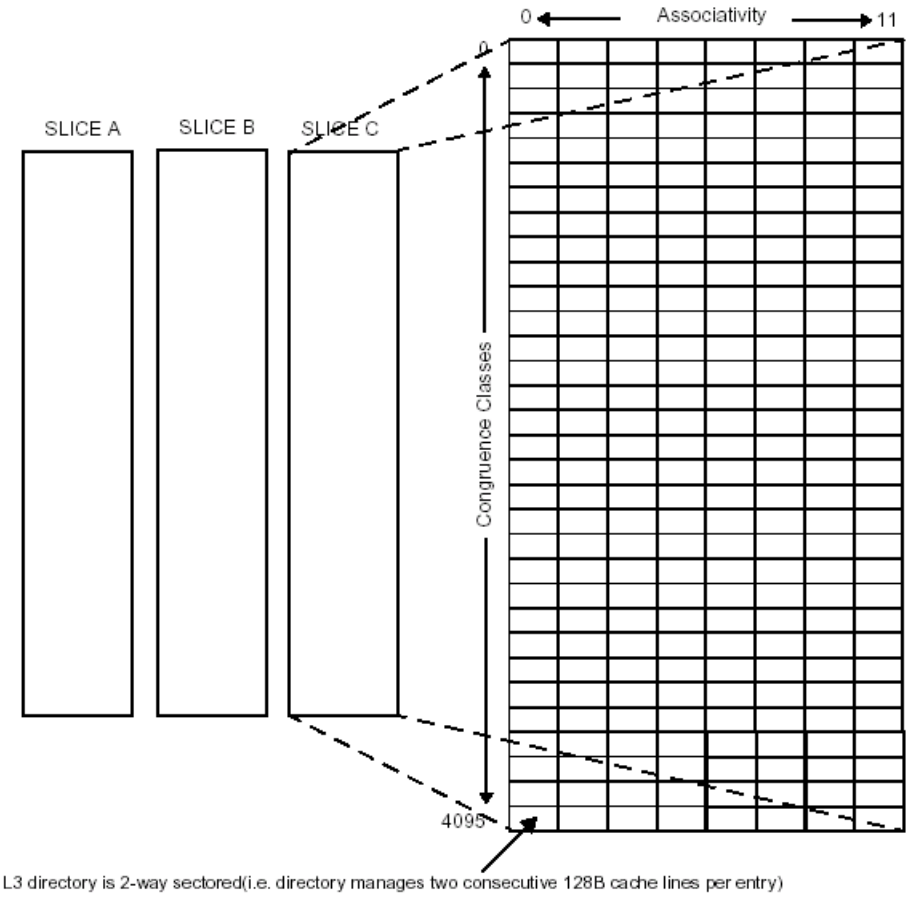


Figure 2-5 L3 cache organization

The L3 cache in POWER5 is on the processor side and not on the memory side of the fabric as in POWER4. This is well depicted in Figure 2-6 on page 16. This design lets the POWER5 satisfy L2 cache misses more frequently, with hits on the off chip 36MB MLD L3, thus avoiding traffic on the interchip fabric. References to data not on the on chip 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.

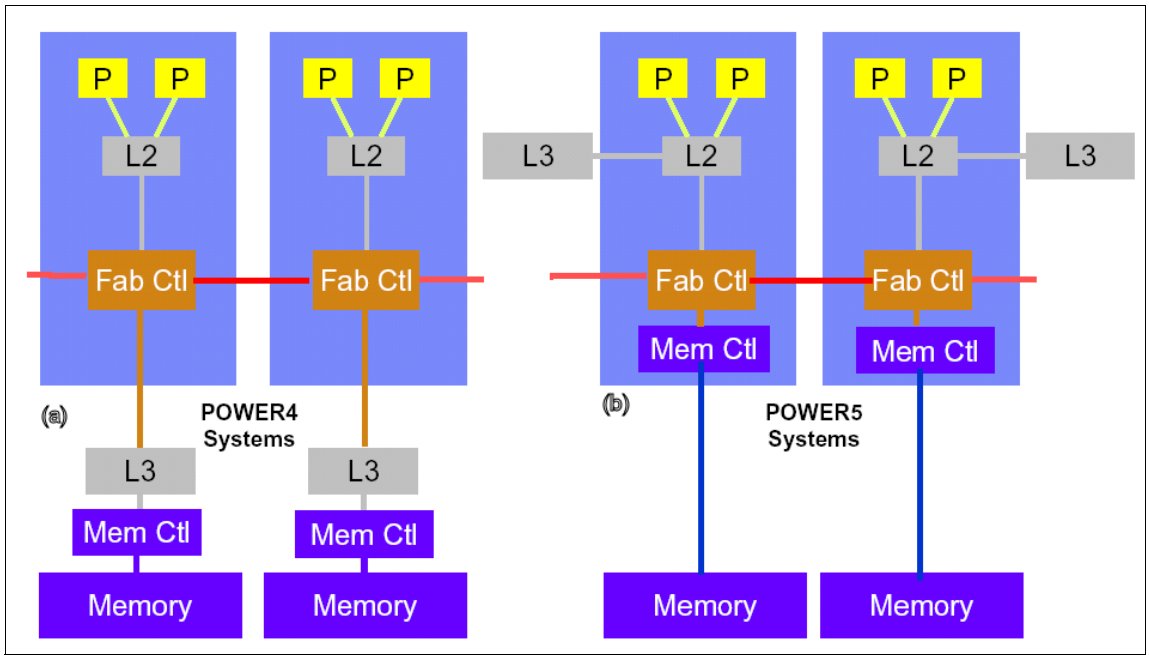


Figure 2-6 POWER4 (a) and POWER5 (b) system structures

Address translation resources

The POWER5 chip supports a 65 bit virtual address and a 50 bit physical (real) address. The PowerPC architecture specifies a translation lookaside buffer (TLB) and a segment lookaside buffer (SLB) to translate the *effective address* (EA also referred to as logical address) -- used by software to real address (physical address) used by the hardware. The processor core contains a unified 1024 entry 4 way set associative LRU TLB (*Translation Lookaside Buffer*). The TLB is a cache of recently looked up page table entries. The ERAT caches contain translations derived from page tables and SLB (*Segment Lookaside Buffer*). Each processor core has two 64 entry fully associative SLBs, one per each thread. Although the ERATs are shared by both the hardware threads of the core, each ERAT entry has a thread ID identifier since ERAT caches both SLB and TLB, and SLBs are process specific entries. Each processor core includes a 128 entry, fully associative, LRU *Data Effective to Real Address Translation* (D-ERAT) cache for fast translation of data effective address to physical (real) addresses. Each D-ERAT entry contains translation information for a 4 KB block of effective storage (even if this section of storage is translated via a large page translation). The POWER5 processor core includes a separate 128 entry fully associative

Instruction Effective to Real Address Translation cache (I-ERAT) for fast translation of instruction effective address to real (physical) address.

2.2.2 Processor Core

The POWER5 processor core is designed to support both SMT and ST (single threaded) modes. Software (operating systems using hypervisor calls) can switch the processor from SMT mode to ST mode. Figure on page 17 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 on POWER5 and POWER4. This design lets the software optimizations designed for POWER4 work equally well on POWER5.

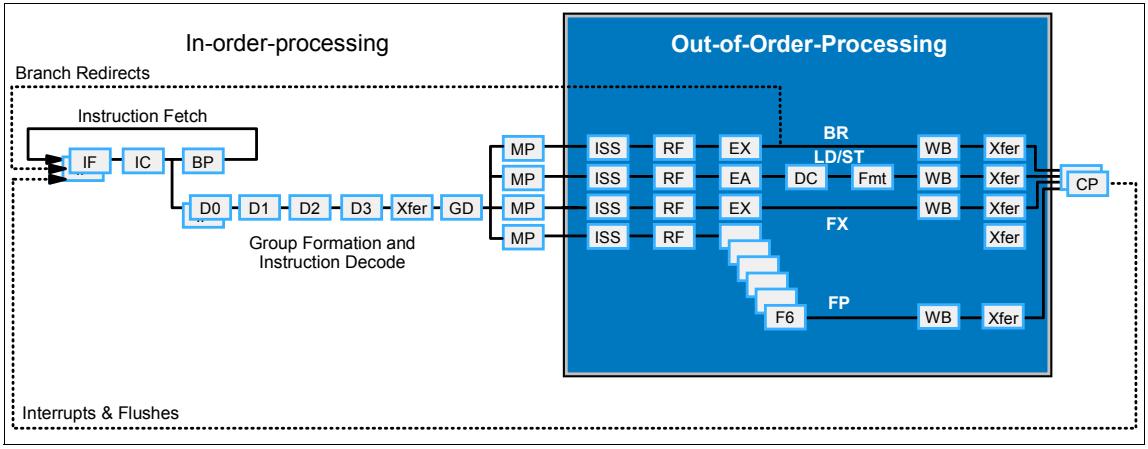


Figure 2-7 POWER5 instruction pipeline.

List of abbreviations used in Figure on page 17

- IF -- instruction fetch
- IC -- instruction cache
- BP -- branch predict
- D0 - D3 -- decode stage zero to three
- Xfer -- transfer
- GD -- group dispatch

- MP -- mapping
- ISS -- instruction issue
- RF -- register file read
- EX -- execute
- EA -- compute address
- DC -- data caches
- F6 -- six stage floating point execution pipe
- FMT -- format data
- WB -- write back
- CP -- group commit

POWER5 processor pipeline

The microprocessor pipeline structure can be subdivided into a master pipeline and several different execution pipelines. 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 mis-predicted branches. The execution pipelines allow out of order issuing of speculative and non speculative operations. The execution unit pipelines progress independently from the master pipeline, and one another.

Instruction Fetch stage

Figure 2-8 on page 19 depicts instruction flow on the POWER5. 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 on page 17). In ST 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 a oct-word aligned block of eight instructions per cycle. The two threads share the instruction cache and the instruction translation facility (L1 i-cache and I-ERAT). POWER5 also provides for a 4 entry 128B instruction prefetch queue above the i-cache, and hardware initiated prefetching. The first two entries of the instruction prefetch queue are ear-marked for thread 0 and the other remaining two entries for thread 1, irrespective of whether the core is running in SMT or ST mode. In a given cycle, instructions are fetched from the same thread.

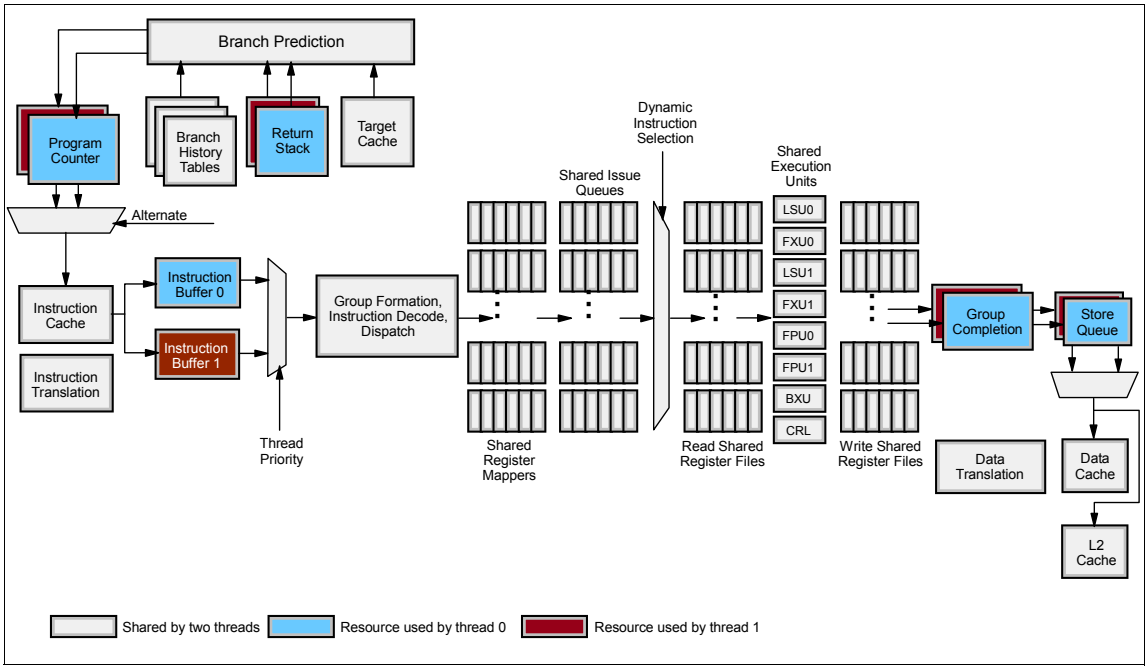


Figure 2-8 POWER5 instruction and data flow

Branch predict stage

The POWER5 scans the eight fetched instructions for branches each cycle (BP stage). If branch instructions are found, the branch direction is predicted using three branch history tables (BHT). The BHTs are shared by the two threads, and two of the BHTs use bimodal and path-correlated branch prediction mechanisms to predict branches. The third BHT is used as a selector -- to predict which of these prediction mechanisms is more likely to predict the right direction. 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 POWER5 has the capability to track up to eight outstanding branches per thread in SMT mode, and 16 outstanding branches in ST mode. The POWER5 also predicts the target of a taken branch in the current cycle's eight instruction group. Target of most branches are calculated from the instruction's address and offset value in the PowerPC architecture. For predicting targets of subroutine returns, the processor uses a per thread eight entry link stack (return stack). For predicting targets of branch to CR (conditional register) instructions, a 32 entry target cache shared by both the threads is used. If a branch is taken, the processor loads the program counter with the branch's target address. Else, address of the next sequential instruction to fetch from is loaded into the program counter.

Instruction decode and preprocessing

Instructions in the predicted path from BP stage are placed in per thread instruction fetch buffers (IFBs, also know as instruction fetch queues). This happens in the D0 stage (see Figure on page 17). POWER5 has two six entry IFBs in the core -- one for each thread. Each IFB entry can hold 4 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 a group 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 resulting out of 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 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 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 entry in the global completion table (GCT). The GCT is used to track the groups of five instructions formed in the D0-D3 stage. POWER5 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 are renamed by utilizing a large physical register file provided on the power5 core. Each register renamed by instructions in a group need to have a corresponding physical register. The rename mapper serves this purpose. This happens in the MP stage of the instruction pipeline. Table 2-2 on page 21 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 ST mode, allowing higher instruction level parallelism.

Table 2-2 POWER5 processor core Rename resources

Resource type	Logical size per thread	Physical size
GPRs	32 (36[a])	120
FPRs	32	120
XERs	4 fields[b]	32
LR/CTRs	2	16
CRs	8 (9[c]) 4-bit fields	40
FPCRs	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 pipeline

After the MP stage, instructions enter the issue queues shared by the two threads. These issue queues feed the execution pipelines. The POWER5 has:

- two fixed point execution pipelines[2]
  - both capable of basic arithmetic, logical and shifting operations
  - both capable of multiplies
  - one capable of divides and the other capable of 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
- one branch execution pipeline
- one condition register logical pipeline

[2] Figure on page 17 does not depict the number of execution units.

The following instruction queuing resources (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
- on 10 entry issue queue for condition register logical execution pipeline

All in all, like the POWER4, the POWER5 has eight execution units, each of which can issue out of order, with bias towards the oldest operations first. Each execution unit can issue an instruction each cycle. Each execution unit can remove an instruction every cycle.

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 -- ready instructions from either of the threads can be issued at any given time, simultaneously by 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 back to the output physical register (WB stage). In each load store unit, an adder computes the effective address to read from or write to (EA stage) and the data cache is subsequently accessed (DC stage). For load instructions, when data is available, 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 POWER5 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. The POWER5 has a 20 entry GCT shared by the two threads.

2.3 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 etc., have been arrived at by experimenting with workloads and varying 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 feature *dynamic resource balancing* and *adjustable thread priorities*.

2.3.1 Dynamic resource balancing (DRB)

The purpose of this feature is to ensure smooth flow of both the threads through the processor. If either of the hardware threads start hogging the processor resources, depriving the other thread, the dynamic resource balancing logic throttles down the thread hogging resources so that the other thread can flow smoothly without stalling. For example, if one of the hardware threads experiences multiple L2 data cache misses, dependant instructions can hog the issue queue slots preventing the other thread from dispatching instructions (refer to the processor pipeline discussion earlier in this section and Figure 2-8 on page 19). To prevent such stalls, the DRB logic monitors the miss queues, and if a particular thread reaches a threshold for L2 cache misses, throttles that thread down, so that the other thread can progress smoothly. Similarly, one thread could start using too many GCT entries, preventing the other thread from dispatching instructions. DRB logic then detects this condition and throttles down the thread hogging 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
2. Holding the thread from decoding instructions until resource congestion is cleared
3. Flushing all the (thread's) instructions waiting for dispatch and holding the thread's decoding unit until congestion clears

Mechanism 1 on page 23 is used in situations where a thread has used more than a predetermined number of GCT entries.

When number of L2 misses incurred by a thread reach a threshold, mechanism 2 on page 23 is used.

If a long executing instruction of a thread (such as memory ordering instructions -- e.g., **sync**) causes hogging of issue queues by instructions of that thread, mechanism 3 on page 23 is used.

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

2.3.2 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 etc. 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 can specify 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 per-thread status register (TSR is actually a per thread 64 bit register. A three bit field is used to indicate thread priority). A thread priority of 0 indicates that a thread has actually been shut off. Table 2-3 on page 24summarizes the thread priority numbers and their priorities. Upon power on or system reset, the TSR thread priority field is reset to priority 4 -- which is the 'Normal' priority. The thread priorities can be changed by either using the or x,x,x noop or by writing to the thread's TSR using the mtspr instruction. The TSR can be read using mfspr instruction. If the software running the or x,x,x noop or mtspr instructions to modify a thread's priority does not have the privilege level required, then that instruction is treated as a noop by the processor.

Table 2-3 Software specified thread priority levels on the POWER5

TSR Thread priority value	Priority Level	Privilege level for software to set this priority[a]	Equivalent noop instruction
0	Thread shut off	Hypervisor[b]	-
1	Very low	Supervisor	`or 31,31,31`
2	low	User	`or 1,1,1`
3	Medium low	User	`or 6,6,6`
4	Normal	User	`or 2,2,2`
5	Medium high	Supervisor	`or 5,5,5`
6	high	Supervisor	`or 3,3,3,`
7	Extra high	Hypervisor	`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 25 summarizes decode unit division among threads under different thread priority scenarios.

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

Thread0 priority	Thread1 priority	Decode slots status
0	0	Stopped
0	1	1 /32 of decode slots given to thread1 for power savings. Thread0 is stopped
0	>1	All of the decode slots go to thread1
1	1	Both the threads are given 1/32 of the total decode slots for power savings
1	>1	Thread1 gets all the execution resources ad thread0 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 2-9 on page 26 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.

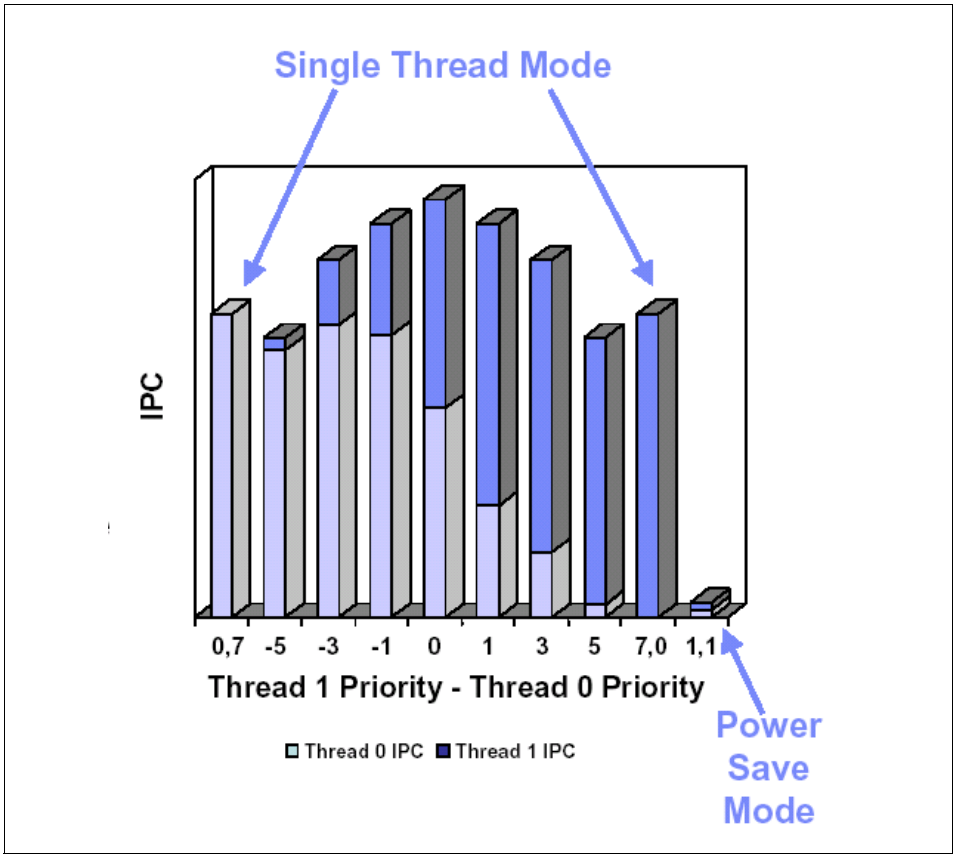


Figure 2-9 Thread priority pairs vs. instructions executed per second

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 ST 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 ST 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 ST for such applications and workloads.

2.4 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.

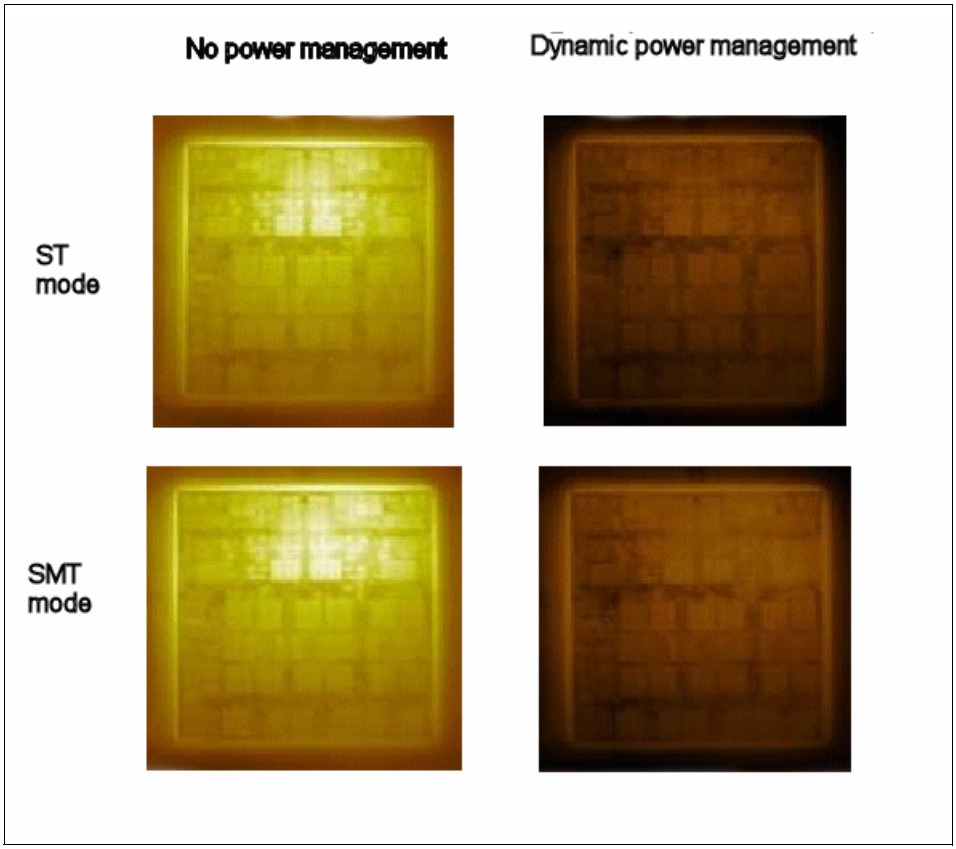


Figure 2-10 Photos taken with thermal sensitive camera while prototype POWER5

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 1 as shown in Table 2-4 on page 25, the POWER5 dispatches instruction utmost once in every 32 cycles saving on power. Figure 2-10 on page 27 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.

2.5 Large POWER5 SMPs

Somewhat like the POWER4, the POWER5 uses *dual chip modules* (DCM[3]s) and *multi chip modules* (MCMs) as the basic building blocks for low/midrange and high end servers respectively.

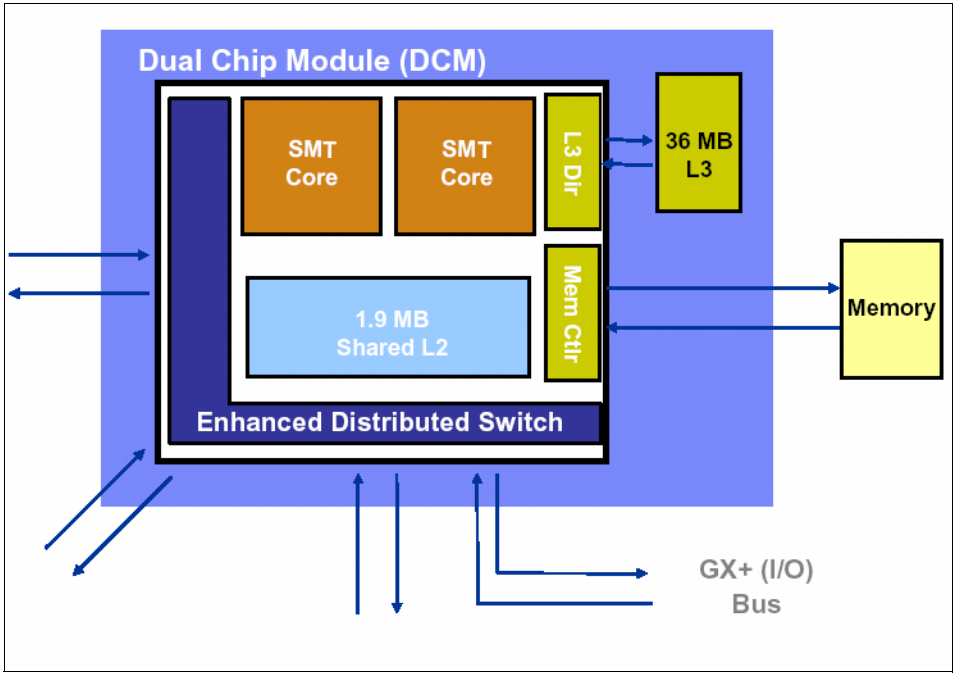


Figure 2-11 POWER5 DCM

[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.

Figure 2-11 on page 28 depicts a POWER5 DCM. This basic building block can be put together to from a 16 way system as shown in Figure 2-12 on page 29. 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 form a two way, four way , eight way, tweleve way and sixteen way smps with 1,2,4,6 and 8 DCMs respectively. The dark blue boxes represent memory.

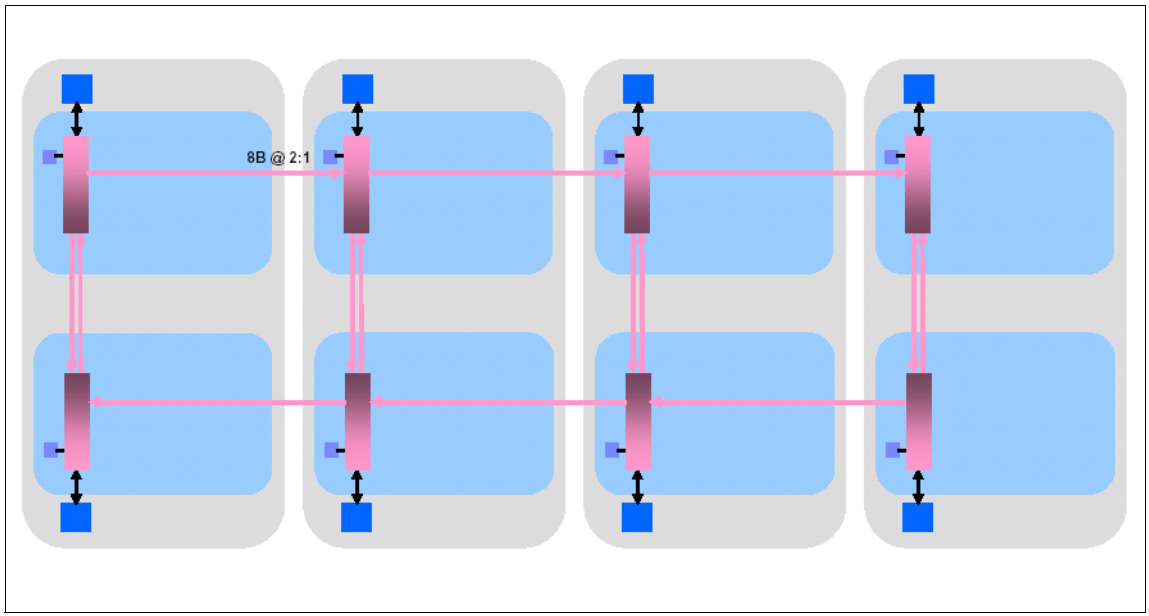


Figure 2-12 DCM interconnection for a 16 way SMP

Figure 2-13 on page 30 shows a picture of the POWER5 DCM. The chip to the left of the reader is the L3 MLD cache chip, and te larger blue chip onto the right is the POWER5 chip.

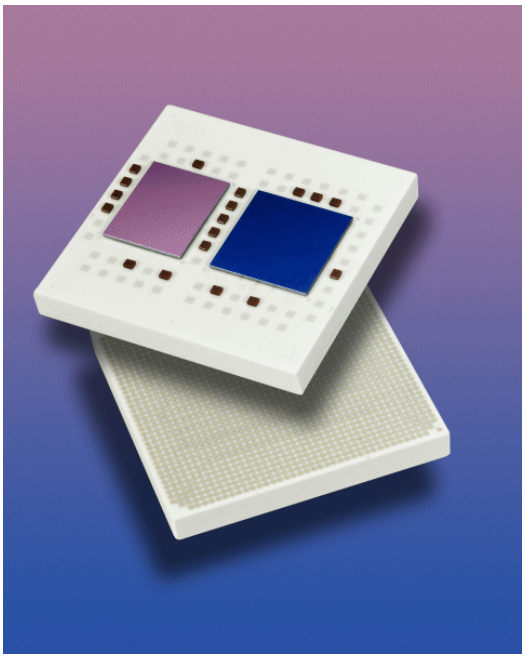


Figure 2-13 Picture of a POWER5 DCM

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.

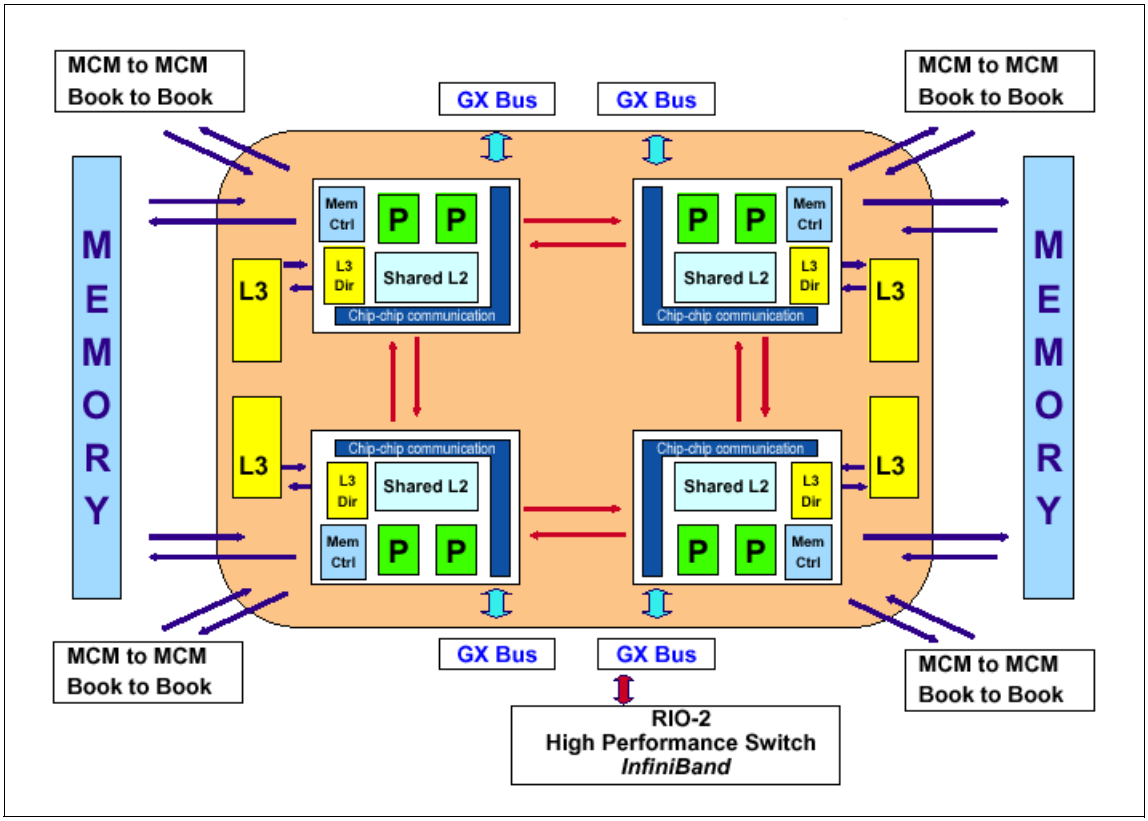


Figure 2-14 Logical view of the POWER5 MCM

Figure 2-14 on page 31 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 2-15 on page 32 shows a picture of a POWER5 MCM.

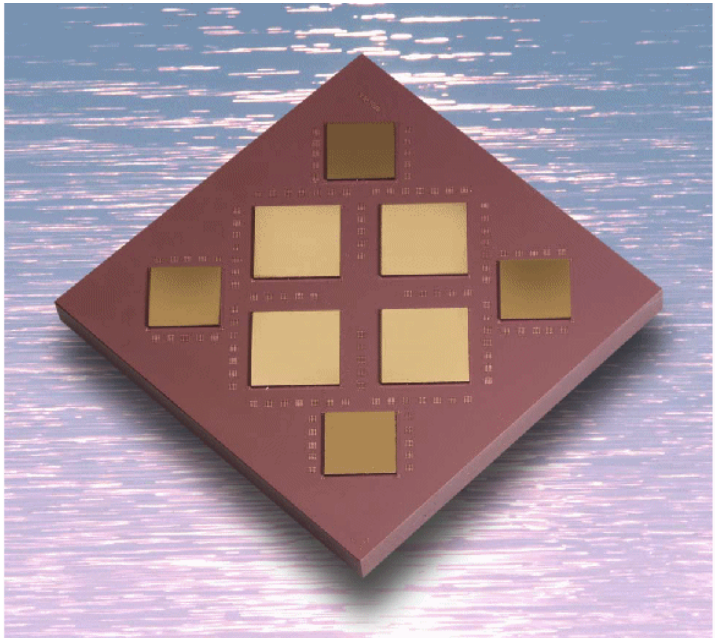


Figure 2-15 POWER5 MCM

Two POWER5 MCMs can be tightly coupled to form a book as shown in Figure 2-16 on page 33. These books are interconnected together again to form larger SMPs upto 64 ways as shown in Figure 2-17 on page 33. In the figure the light blue box represents a MCM. The MCMs and books can be interconnected to form a eight way, 16 way , 32 way, 48 way and 64 way smps with 1,2,4,6 and 8 MCMs respectively.

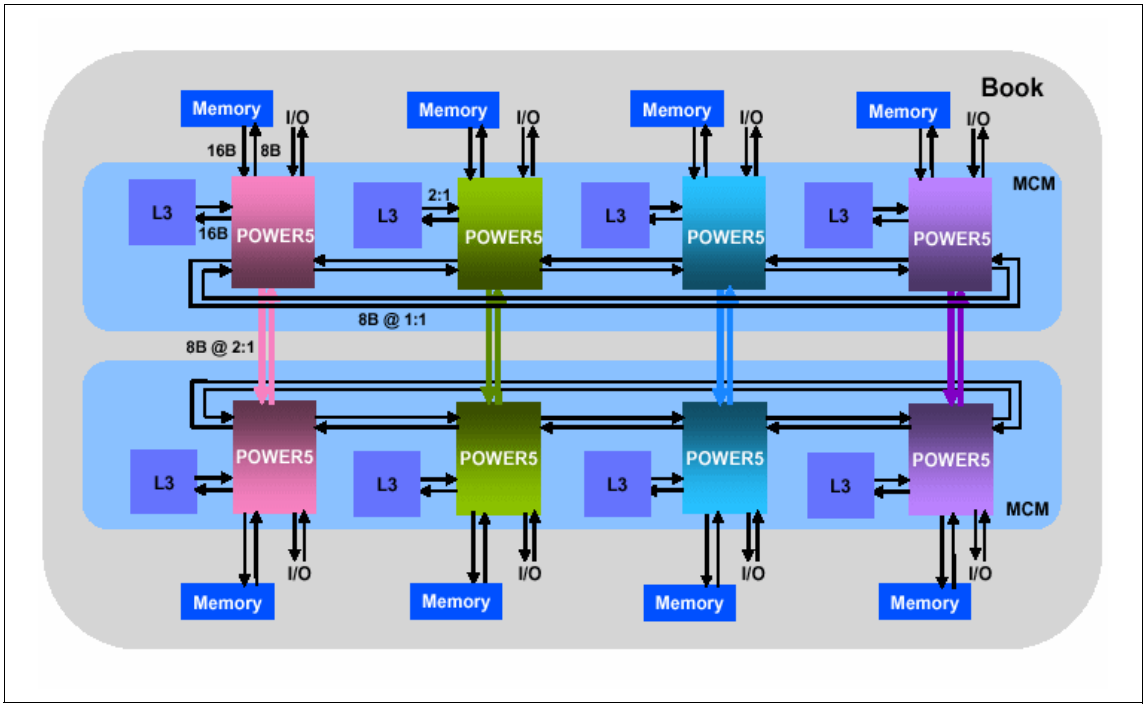


Figure 2-16 POWER5 book

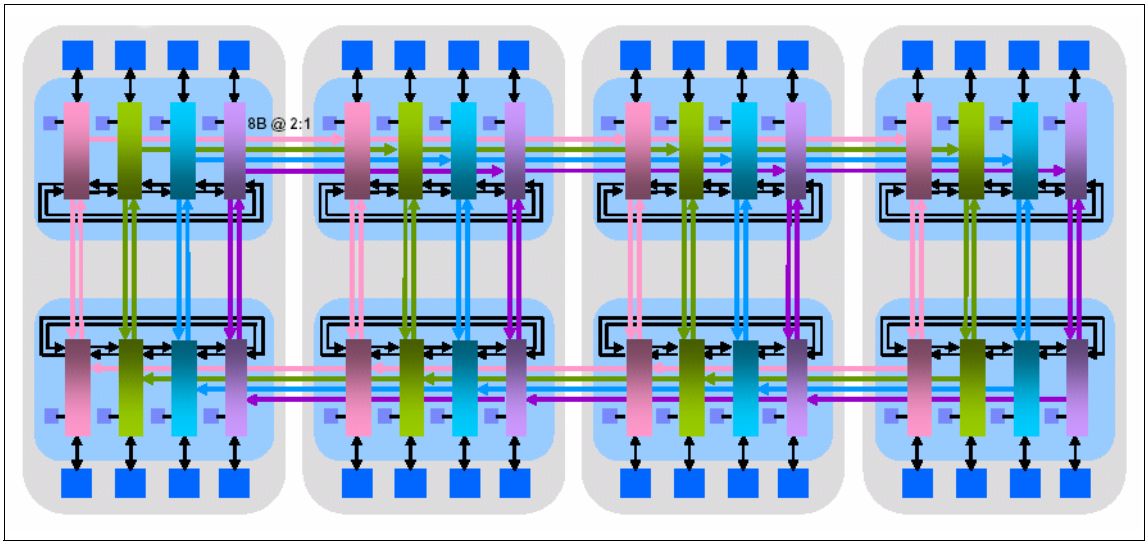


Figure 2-17 MCMs interconnected to make a 64 way SMP

2.6 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 POWER 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 @server p5 and IBM @server i5 servers

The POWER Hypervisor supports many advanced functions compare to the previous hypervisor, these includes 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.

POWER Hypervisor is the underlying control mechanism that resides below the operating system itself but above the hardware level. The hypervisor owns all system resources and creates partitions by allocating these resources and sharing them.

The layers above the POWER Hypervisor are different for each supported operating system. See Figure 2-18 on page 36.

For the AIX 5L and Linux operating systems, the layer above the POWER 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).

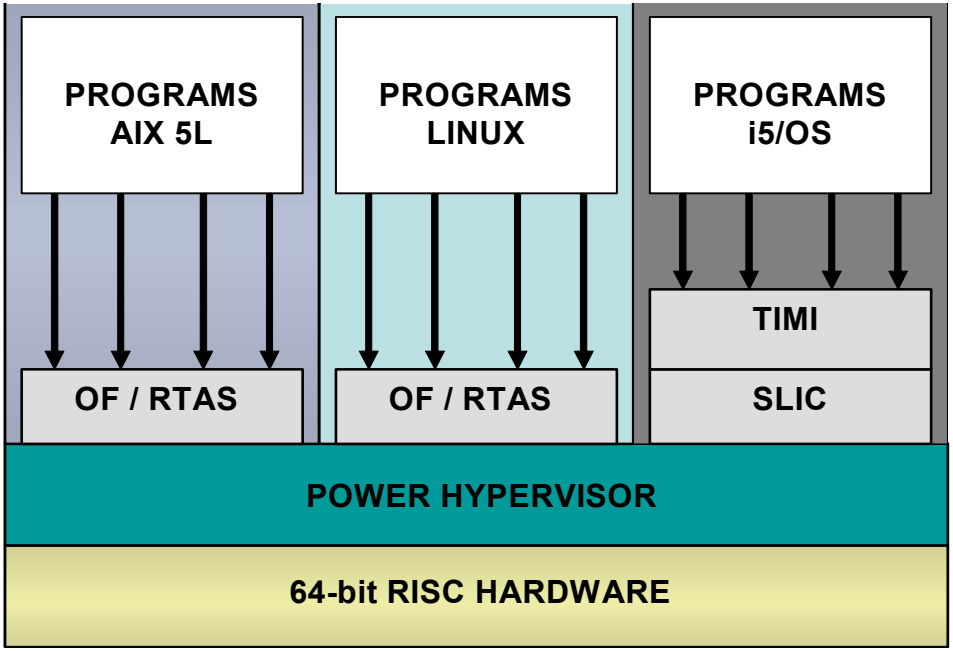


Figure 2-18 POWER Hypervisor

System Firmware is composed of Low Level Firmware which is a 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 consist of code that supplies platform dependent accesses and can be called from the operating system. These calls are passed to the POWER 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 software, 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.

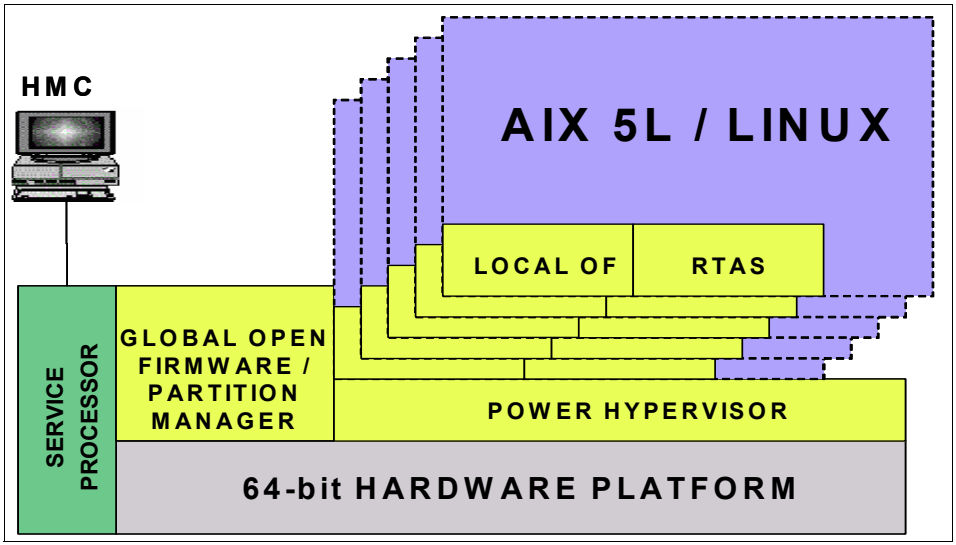


Figure 2-19 POWER Hypervisor on AIX 5L and Linux

For i5/OS, Technology Independent Machine Interface (TIMI) and the layers above the POWER Hypervisor are still in place. System Licensed Internal Code (SLIC), however, is changed and enabled for interfacing with the POWER Hypervisor. The POWER 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 POWER Hypervisor.

All POWER5-based servers require the use of the POWER Hypervisor. A Customer 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.

2.6.1 POWER Hypervisor Support

The POWER5 processor supports a special form of instructions. These instructions are exclusively used by enhanced controlling firmware named POWER Hypervisor. If an operating system instance in a partition requires access to hardware, it first invokes hypervisor using hypervisor calls [hcall()]. Hypervisor allows privileged access to an operating system instance for dedicated hardware facilities and includes protection for those facilities in the processor and memory locations.

Architecturally, the POWER 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 didn't 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

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

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 have 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 [hcall()] interrupt is a special variety of the system call instruction. The parameter to the hcall() 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 hcall(). This minimizes the address translation problem pointer parameters would cause. 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 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.

Hypervisor Call 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, MMIO space, NVRAM, etc. 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 hcall()s 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 hcall()s

Debugger Support hcall()s provide the capability for the real mode debugger to be able to get its async port and beyond the real mode limit register without turning on virtual address translation.

Virtual Terminal Support

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 hcall()s

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 hcall()s. This hcall() is enabled or disabled (default disabled) via the Hardware Management Console.

Memory Migration Support hcall()

Memory Migration Support hcall() was provided to assist the operating system in 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 hcall()s

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 s 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.

The *lparstat* command in AIX 5L Version 5.3 with -H flag will display the partition data with detailed breakdown of hypervisor time by hcall type. See example below.

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

           Detailed information on Hypervisor Calls

Hypervisor         Number of    %Total Time   %Hypervisor   Avg Call     Max Call
  Call               Calls        Spent      Time Spent    Time(ns)     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_rdma                 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_logicl_lan_buf        0            0.0           0.0           1            0
```

Figure 2-20 lparstat -H command output

H_REGISTER_VPA is a data area registered with POWER 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 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.

H_PUT_TCE this hcall() enters mapping of a single 4096 byte page into the specified TCE.

H_READ this hcall() returns the contents of a specific PTE in registers R4 and R5.

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 R4.

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 LA IOA.

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.

H_PUT_RTCE this hcall() maps “count” number of contiguous TCEs in an RTCE to the same number of contiguous IOA TCEs.

H_PAGE_INIT this hcall() initializes pages in real mode either to zero or to the copied contents of another page.

H_GET_TCE this standard hcall() s used to manage the interpartition logical LAN IOA's I/O translations.

H_COPY_RDMA this hcall() 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 hcall().

H_SEND_CRQ this hcall() sends one 16 byte 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 hcall() sends a logical LAN message.

H_ADD_LOGICAL_LAN_BUF this hcall() adds receive buffers to the logical LAN receive buffer pool.

H_PIC this hcall() returns the summation of the physical processor pool's idle cycles.

H_XIRR this hcall() is extended to report the virtual interrupt source number associated with virtual interrupts associated with an interpartition logical LAN IOA.

H_POLL_PENDING this hcall() provide the operating system to perform background administrative functions task, and providing one time implementation with indication of pending work so that it may more intelligently manage the use of hardware resources.

H_PURR is a new resource provided for micro-partitioning and simultaneous multi-threading (SMT), which provides an actual count of tics that the shared resource is used (per virtual processor, or per SMT thread). In the case of micro-partitioning, the virtual processor's PURR tics when the virtual processor is dispatched onto a physical processor. Therefore, comparisons of elapsed PURR with elapsed Timebase provide 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 hcall() time stamping should be converted to use PURR instead of timebase.

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 prod).
- The old virtual processor exhausted its time slice (HDERC interrupt).
- The old virtual processor ceded / 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 decrementer. The virtual processor's Processor Utilization Register (PUR) 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 / 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.

Virtualized Input / Output

Virtual I/O support is one of the advanced features of the new POWER Hypervisor. Virtual I/O (VIO) provides a given partition the appearance of I/O adapters (IOAs) that do not necessarily have direct correspondence with a physical IOA. Virtual I/O is covered in detail in "Virtualized I/O and the POWER Hypervisor" on page 128

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 mush 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.

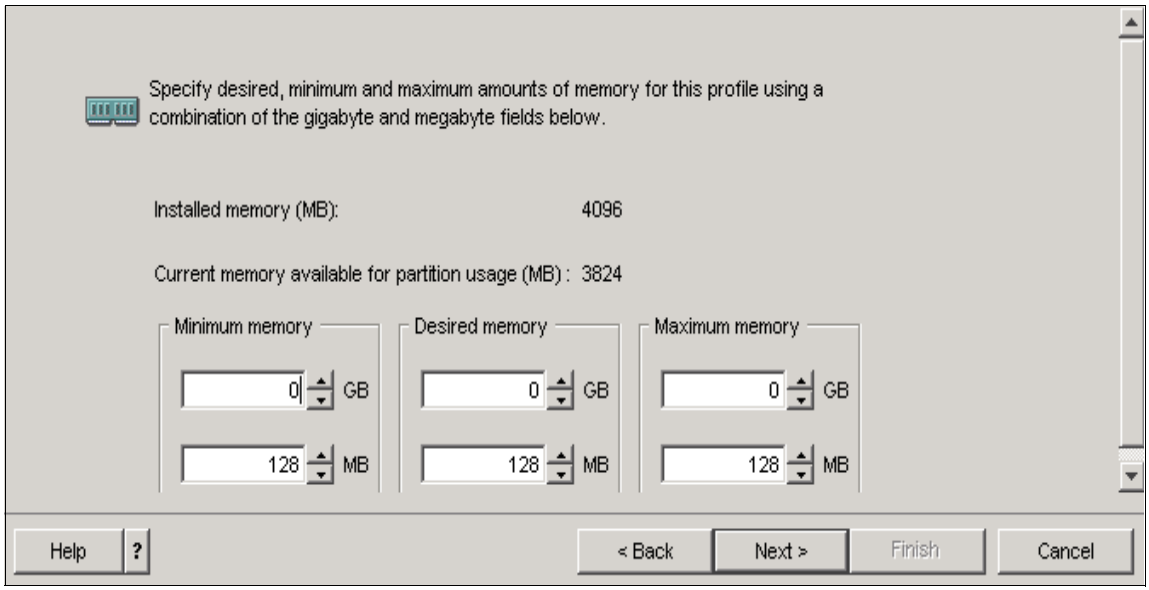


Figure 2-21 Current memory available for partition usage using HMC

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 2-21 on page 45. 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.

2.6.2 POWER Hypervisor Design

POWER 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 CPUs within the pool have the same affinity characteristics, at least at some level. Then, the partition is guaranteed to only execute on that pool of processors, barring events such as a CPU being GARD'ed off due to predictive failures, and possibly replaced with a spare CPU from another affinity domain.

The hypervisor will continue to provide affinity domain information in the device tree for CPUs, which are actually virtual CPUs 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 CPU it might stop at the lowest common layer of all CPUs in the shared pool. POWER 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, etc..

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 esoteric 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 would be required for proper functioning in shared processor environment.

Preemption of a Virtual Processor

The POWER 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 decrementer (HDECR). This is a clock interrupt source utilized by POWER 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. POWER 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 a complete support for the hypervisor decrementer. The SRR0 / 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 (by the hardware) for N (programmable hardware value) cycles if MSR[R1]=0. N will be large enough to allow a partition to safely execute sequences where SRR0/1 are live without taking HDECR interrupt (and suffering the corresponding loss of state). this places the requirement that an operating system use MSR[R1] property 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 a live SRR0/1.

The hypervisor will issue 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 mechanisms on the system in the order it was intended. The hypervisor will also do the equivalent of a dummy stwcx in yield and preemption sequences to cancel a reservation that may be held by the yielding or preempted virtual processor.

Cache Invalidations

The Segment Lookaside Buffer SLB (saved when the virtual processor yielded or was preempted) will be 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 will be 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 in the shared pool, all processors in the pool flush their Icache prior to switching in a virtual processor from the partition being IPLed.

Hypervisor Dispatching Algorithm

Each shared pool will have 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 1 msec per processor; the hypervisor will calculates 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 decrementor 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 decrementor 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 1 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.

2.6.3 Performance Considerations

The hypervisor does use a small percentage of the system CPU and memory resources. These overhead in memory resources is associated with the virtual memory management that will be used for the hypervisor dispatcher, virtual processor data structures (including save areas for virtual processor) 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.

The other sources of the performance overhead in the hypervisor are the following:

- Increase path length
- Dispatching overhead of virtual processors (i.e. saving and restoring state).
- TLB flush when a virtual processor is dispatched.
- Increased misses in a shared processors caches.

The performance overhead associated when using POWER Hypervisor is equal or less than the fraction of time spent in the previous hypervisor. The overhead is normally 1-5% compared to running without the hypervisor on POWER4 systems. The overhead is related primarily to rates of I/O and paging, more I/O or more paging means more time in the hypervisor. From a performance point of view, there is nothing that you need to fine-tune with the hypervisor. However, it is important to understand it from a conceptual point of view. Operating system and applications run inside the logical partition in the same way they run on a

stand-alone server or cluster node, so it's all transparent to the end users when dealing with the hypervisor. POWER5-based servers cannot run without the hypervisor, so all Benchmarks Published Results from POWER5 processor already contain the hypervisor overhead.

The output of lparstat with -h flag will display the percentage spent in hypervisor (%hypv) and the number of hcalls. Notice from the example below that 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 CPU 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 2-22 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 with -i flag command 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 2-23 lparstat -i command output

2.7 Partitioning on the IBM @server p5

Micro-partitioning is a technology inspired on the zSeries heritage that allows a partition in a system to have virtualized resources that it can use as the were real resources. Physical processors and I/O devices have been virtualized enabling these resources to be shared by multiple partitions.

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, or they can have their processors virtualized from a pool of shared physical processors. Both types of partitions can coexist in the same system at a given time.

A *dedicated processor partition* is much like the partitions used on POWER4 processor based servers, where whole processors are assigned to partitions. These processors are owned by the partition where they are running and are not shared with other partitions. Also, 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 to be used 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 Storage, although they can still have the physical resources if desired. Both Virtual Ethernet and Virtual Storage are covered in <xref to VLAN and VSCSI chapters>.

A *shared processor partition* differs itself in the sense that physical processors are abstracted into *virtual processors* which are then assigned to partitions. These virtual processor have capacities ranging from 10 percent of a physical processor, up to a whole processor. A system can therefore have multiple partitions sharing the same processors, and dividing the processing capacity among themselves, as shown in Figure 2-24.

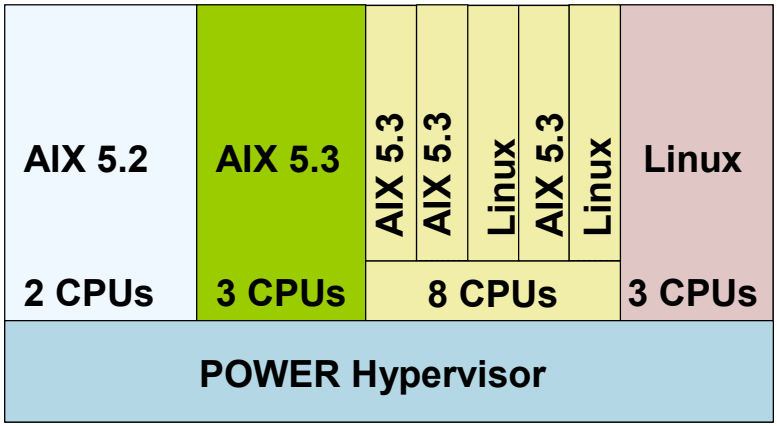


Figure 2-24 A system with dedicated and shared processor partitions

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 from the operating system perspective. 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 optimal performance, the operating system can be enhanced to exploit micro-partitioning more in-depth, for instance by voluntarily relinquishing processor cycles to the POWER Hypervisor, when they are not needed. AIX 5L version 5.3, the first version of AIX to support micro-partitioning, includes such optimizations.

There are several advantages associated with this technology including finer grained resource allocations, more partitions and higher resource utilization. Partitions running with a certain amount of processing capacity may give back to the POWER Hypervisor.

2.8 Micro-partitioning implementation

The shared processor partitions are implemented on POWER5-based servers using the POWER Hypervisor as the abstraction layer between the physical processors and the virtual processors as seen by the partitions. These virtual processor objects and hypervisor calls have been added to the existing partitioning implementation 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 a 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.

A virtual processor can have as little as 10% of the capacity of a physical processor, and can have its capacity increased up to 100% of a physical processor. This granularity is implemented by varying the time a virtual processor is actually running in a physical processor. Each virtual processor gets dispatched on a physical processor by the POWER Hypervisor for a time that depends on the number of virtual processors in the system, and the *entitled processor capacity* that has been assigned to that partition. The POWER Hypervisor dispatches all virtual processors in a 10 ms interval, called a *dispatch wheel*. Each physical processor has its own dispatch wheel assigned by the hypervisor. Figure 2-25 illustrates the assignment of virtual processors to a physical processor.

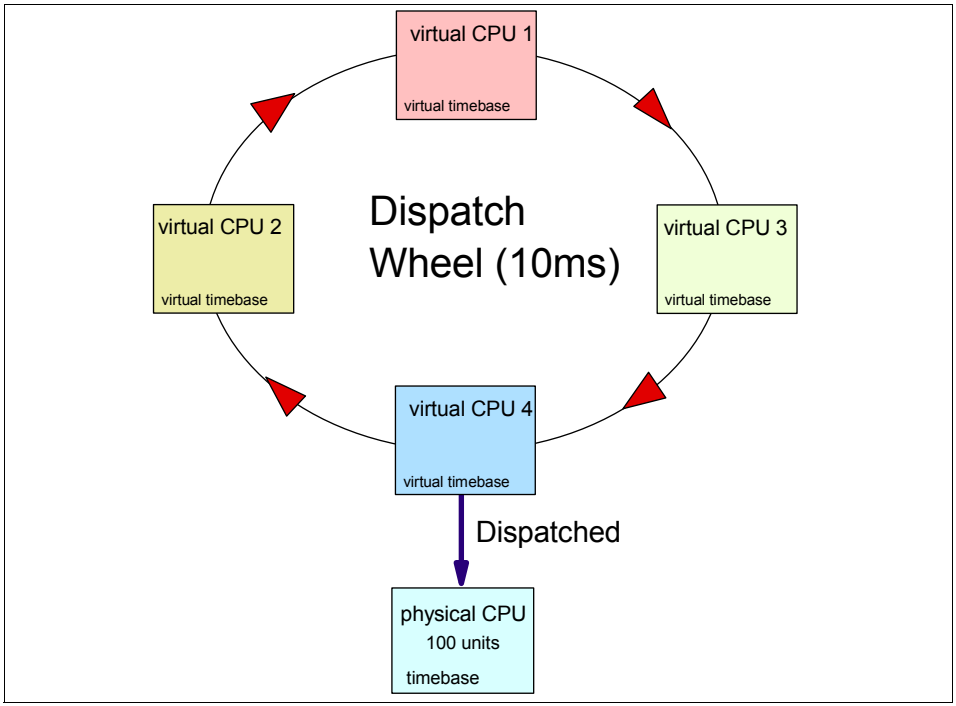


Figure 2-25 Dispatch wheel for allocating physical processor time to virtual processors

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 a single virtual processor by SMT) are dispatched together whenever the POWER Hypervisor schedules the virtual processor to run. The amount of time that each virtual processor run is split between the two logical processors. Figure 2-26 shows a diagram for a case when SMT is enabled.

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

The POWER5 processor architecture attempts to deal with these complex issues by introducing a new processor register that is intended for measuring utilization. This new register, called Processor Utilization Resource Register (PURR), is used to approximate the time that a virtual processor is actually running on a physical processor. The register advances 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. The hardware increments the PURRs based on how each thread is using the resources of the processor including the dispatch cycles that are allocated to each thread. For a cycle in which no instructions are dispatched, the PURR of the thread that last dispatched an instruction is incremented. By specifying a PURR per thread instead of a single PURR per processor, the POWER Hypervisor can measure performance for each hardware thread separately.

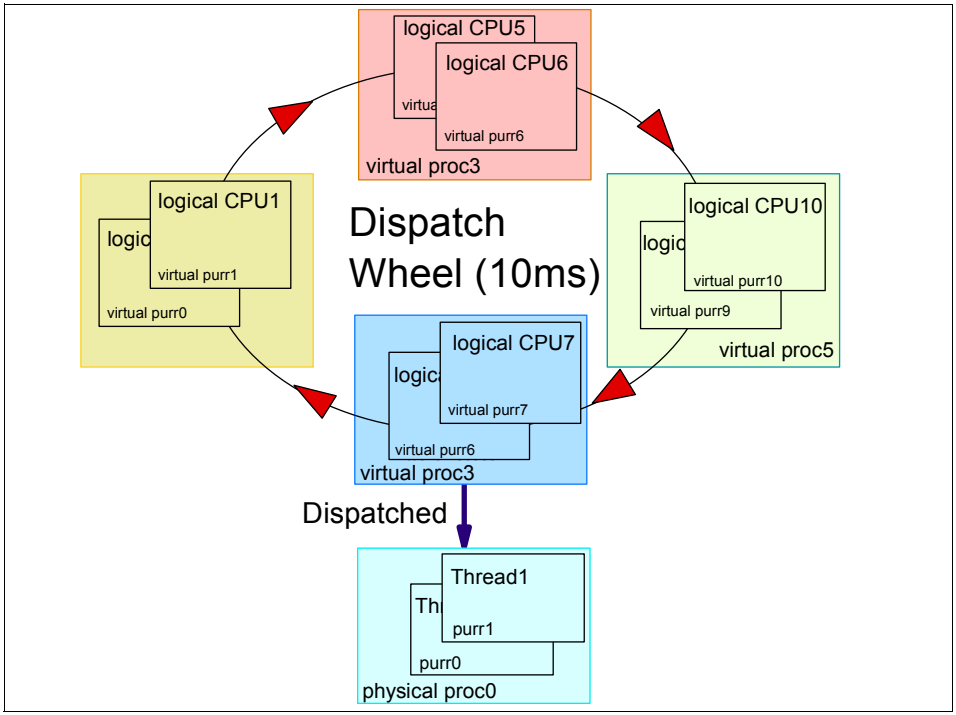


Figure 2-26 Dispatch wheel for SMT-enabled processors

Virtual processors are dispatched if there is a thread waiting to be executed by that processor. Also, virtual processors on a same partition do not necessarily get dispatched at the same time. The POWER Hypervisor dispatch mechanism assures that every virtual processor are dispatched at each 10 ms interval. It does not specify when in that interval the processors are dispatched, neither assures that virtual processors in a given partition are dispatched together.

Also, virtual processors that are idle can give the cycles that are not being needed back to the POWER Hypervisor, so that it can dispatch other virtual processors and therefore increase the system utilization. In order to do this, the operating system must be able to call the hypervisor system calls that control this behavior, as described in <xref to hypervisor chapter and h_calls>

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_cede() or h_confer(). 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, an interrupt, h_prod() call, or dispatch cycle granted to the confer 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.

2.8.1 Types of shared processor partitions

Shared processor partitions can be of two different types, depending on the capacity they have of using idle processing resources available on the system. Like commented before, 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 some partitions, depending on their type and configuration.

Capped partition

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 idle, the capped partition may reach a CPU utilization of 100%.

Figure 2-27 shows an example where a shared processor partition is capped at an entitlement of 9.5 (up to the equivalent of 9.5 physical processors). In some moments the CPU usage goes up to 100%, and while the machine presents extra capacity not being used, by design the capped partition cannot use it.

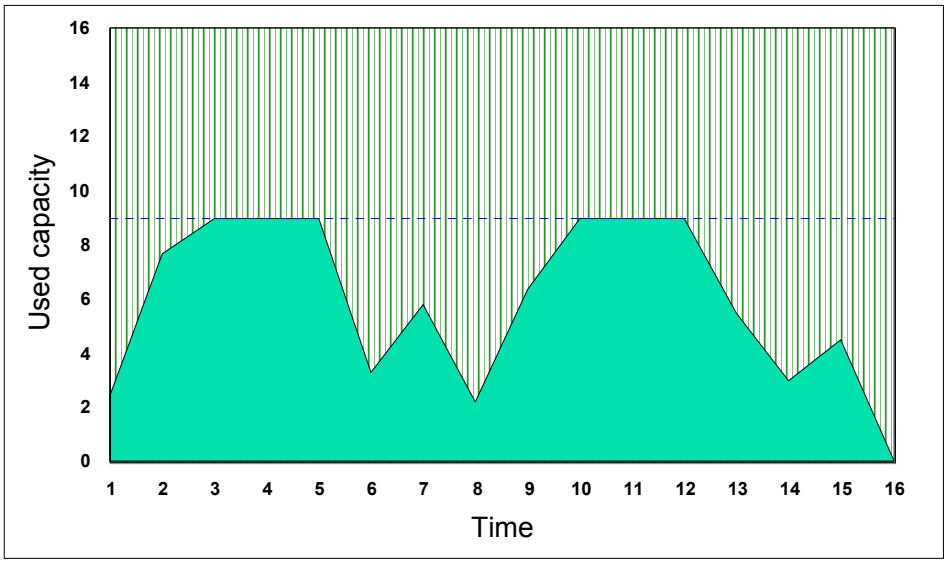


Figure 2-27 Capped partition

Uncapped partition

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 eventually receive more CPU cycles than its entitled capacity.

In the case it is using 100% of the entitled capacity, and there are idle processors in the shared processor pool, the POWER 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 2-28. It still has the equivalent of 9.5 physical processors as its entitlement, but it an use more resources if required.

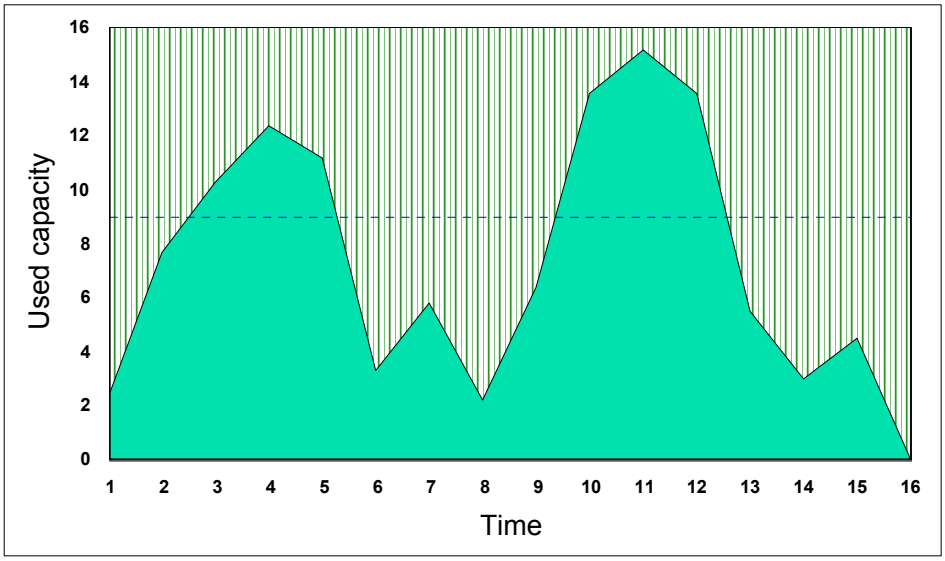


Figure 2-28 Uncapped partition

The number of virtual processors on 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 or more than the number of physical processors in the shared pool, than 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 2-29 shows a situation when an uncapped partition has 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.

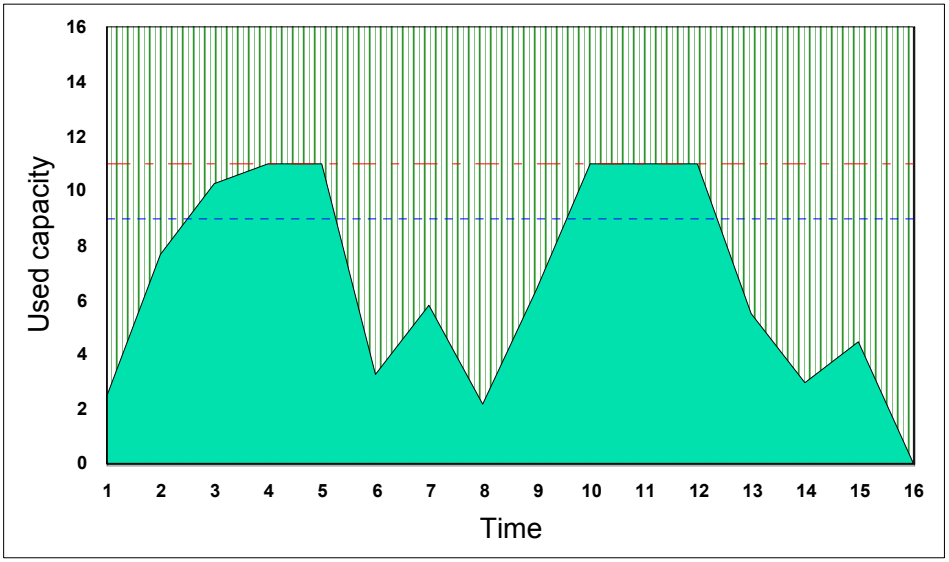


Figure 2-29 Uncapped partition with less virtual processors than physical processors

Weight for uncapped partitions

You can determine how the POWER 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.

2.8.2 Typical usage of shared processor partitions

With fractional processor allocations, more partitions can be created on a given platform enabling customers to maximize the number of workloads that can be supported simultaneously. Shared processor partitions enable both optimized use of processing capacity while preserving the isolation between applications provided by different operating system images.

There are several scenarios where the usage 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 costs reduction. A

shared processor system enables the consolidation from small to large systems without the burden of dedicating very powerful processors to a small partition. You can divide the processor between several partitions with the adequate processing capacity for each one.

Server provisioning With the capacity of sharing a processor and using virtual devices, 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 for a better use of processing resources by the applications deployed on the server farm.

There are some considerations when implementing shared partitions, 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. Section 4.1, “Micro-partitioning considerations for performance” on page 102 details some considerations when using micro-partitioning, and provides some guidelines when configuring shared processor partitions.

2.9 AIX 5L Version 5.3

The appropriate version of AIX for POWER5 is AIX 5L v5.3. This means that this version has modifications to acknowledge the new functionalities of the POWER5 processor. AIX 5L v5.2 is also supported but versions prior to AIX 5L v5.2 are not.

2.9.1 Introduction

The implementation of the virtual processor abstraction is in the hardware and in the POWER 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.

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 CPU in the IDLE process so that another virtual CPU within our partition might use it, or even so that another partition could use it.

We can call two functions to control those optimizations:

H_CEDE	Used to give CPU cycles to the pool.
H_PROD	Used to restore CPU cycles to the CPU that has ceded them.

Binary compatibility

As with every release of AIX, the maintenance of the binary compatibility is a requirement. In a shared processor LPAR, things like bindprocessor continue to work, albeit binding to the virtual CPU and not a physical CPU. This aspect could possibly cause problems for an application or kernel extension, which is dependent on executing on a specific physical CPU. 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 CPU in the system. Another example is the bindintcpu command, which allows an administrator to bind bus interrupt levels to specific CPUs. In Micro-Partitioning, AIX 5L v5.3 supports it, and will bound interrupts to virtual CPUs, 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 CPUs when running in shared processor mode. We do not expect to be a significant risk since that type of physical resource

management does not make sense in a shared processor 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 SPLPAR aware.

2.9.2 Simultaneous Multi-Threading (SMT)

On AIX 5L v5.3 with SMT enabled, each hardware thread is supported as a separate logical CPU.

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 4-way by default. Logically, the only supported kernel in a SMT environment is the MP.

In traditional CPU 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 10ms tic. These were documented and reported in sysinfo (system-wide) and cpuinfo (per-cpu) structures and in order to preserve binary compatibility, the mechanism should stay unchanged.

Regarding to performance tools like **`vmstat`**, **`iostat`** or **`sar`**, they would convert tic counts from sysinfo into utilization percentages for machine/partition. Other tools like **`sar`** -P ALL and the **`topas`** hot cpu section there would be a conversion of tic counts from cpuinfo into utilization percentages for a processor/thread.

This, of course, affects greatly on the metrics. Traditional utilization metrics are misleading because they think we have 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 HMT and the same problem exists with hyperthreading.

Processor Utilization Resource Register

In order to solve these problems there's a new register implemented by Power5, the Processor Utilization Resource Register also know as PURR. Each thread has its own PURR. 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. 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 CPU 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.

For physical resource utilization metric for a logical processor we use

(delta PURR/delta TB)

Represents the fraction of the physical processor consumed by a logical processor

(delta PURR/delta TB)*100 over an interval

Represents the percentage of dispatch cycles given to a logical processor

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

```
%sys = (delta PURR in system mode/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. For that we just take `(PPC/ENT)*100`.

Another useful metric is the available pool of processors. Taking PIC as the Pool Idle Count, which represents clock ticks where PHYP 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 10ms-tic-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 purr0/purr1. To know the fraction of time partition1 ran on a physical processor, i.e. the relative amount of processing units consumed use (purr0+purr1)/timebase0.

Priorities

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

2.9.3 Performance tools

Traditional utilization metrics are misleading which means that the old tools are not useful anymore. They need to be changed.

We also need to be measure the new features.

New text based tools

Some new text tools are useful for performance and tuning.

smtctl

This first tool serves the purpose of controlling SMT. To turn it off or on, whether at boot time or immediately you can use:

```
smtctl [ -m { off  | on } [ { -boot | -now } ] ]
```

When you enter the command without any flags it returns information on the status of SMT on your system.

Example 2-1 smtctl information

```
# 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
```

lparstat

lparstat is a tool that can show the configuration settings and the performance settings at the partition level. Its usage is:

```
lparstat { -i | [-H | -h] [Interval [Count]] }
```

It has three modes of operation:

information (-i) For the configuration setting of the lpar

POWER Hypervisor (-H)

For detailed information on POWER Hypervisor calls

monitor which is the default

Example 2-2 lparstat information

```
# lparstat -i
Node Name                                  : p5_1test2
Partition Name                             : Test2_AIX_0425A
```

```
Partition Number                          : 3
Type                                      : Shared-SMT
Mode                                      : Uncapped
Entitled Capacity                         : 30
Partition Group-ID                        : 32771
Shared Pool ID                            : 0
Online Virtual CPUs                       : 1
Maximum Virtual CPUs                      : 6
Minimum Virtual CPUs                      : 1
Online Memory                             : 512 MB
Maximum Memory                            : 1024 MB
Minimum Memory                            : 256 MB
Variable Capacity Weight                  : 128
Minimum Capacity                          : 10
Maximum Capacity                          : 60
Capacity Increment                        : 1
Maximum Dispatch Latency                  : 13999999
Maximum Physical CPUs in system           : 6
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
```

The next example shows the use of lparstat to show the POWER Hypervisor calls.

Example 2-3 lparstat POWER Hypervisor

```
# lparstat -H

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

           Detailed information on Hypervisor Calls

Hypervisor        Number of    %Total Time   %Hypervisor   Avg Call    Max Call
  Call              Calls         Spent      Time Spent    Time(ns)    Time(ns)

remove                   2           0.0           5.3         497         512
read                     0           0.0           0.0           1           0
nclear_mod               0           0.0           0.0           1           0
page_init                9           0.0          50.1        1040        2681
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           0
xirr                     0           0.0           0.0           1           0
eoi                      0           0.0           0.0           1           0
```

```
ipi                       0            0.0           0.0          1            0
cppr                      0            0.0           0.0          1            0
asr                       0            0.0           0.0          1            0
others                    0            0.0           0.0          1            0
enter                    11            0.0          28.8        489          541
cede                      0            0.0           0.0          1            0
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                      0            0.0           0.0          1            0
get_ppp                   1            0.0          13.2       2463         2463
set_ppp                   0            0.0           0.0          1            0
purr                      0            0.0           0.0          1            0
pic                       1            0.0           2.6        492          492
bulk_remove               0            0.0           0.0          1            0
send_crq                  0            0.0           0.0          1            0
copy_rdma                 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_logicl_lan_buf        0            0.0           0.0          1            0
------------------------------------------------------------------------------
```

In the monitor mode, **lparsat** shows the CPU utilization in the usual manner (%user, %sys, %idle, %wait). Optionally, it also shows the percentage spent in POWER Hypervisor (%hypv) and number of hcalls (hcalls). There are additional shared mode only metrics like:

physc	Physical Processor Consumed
%entc	Percentage of Entitlement Consumed
%lbusy	Logical CPU Utilization
app	Available Pool Processors
vcsw	Number of virtual context switches which are the virtual processor hardware preemptions
phint	Number of phantom interrupts that are the interrupts received for other partitions

Example 2-4 lparstat monitoring

```
# lparstat 2 5

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

%user  %sys  %wait  %idle physc %entc  lbusy   app  vcsw phint
-----  ----  -----  ----- ----- ----- ------   ---  ---- -----
```

```
  3.0  11.5    1.8   83.6  0.05  16.7    5.0     -   586     8
 15.6  76.9    0.6    7.0  0.99 329.1   89.2     -   707   296
 15.6  77.2    0.0    7.2  1.00 333.2   90.8     -   715   285
 15.6  77.0    0.0    7.4  0.74 245.2   76.5     -   861   220
  0.0   0.3    0.0   99.7  0.00   0.8    0.0     -   574     0
```

mpstat

One other tool shows detailed logical processor information. This is the **mpstat** tool. It can show up to 29 new metrics (when using -a option). 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 shared mode only)
- Percentage of entitlement consumed (shared mode only)
- Number of logical context switches (shared mode only) meaning the hardware preemptions

Example 2-5 mpstat monitoring

```
# mpstat 2 2

System configuration: lcpu=2 ent=0.3

cpu  min  maj  mpc  int   cs  ics   rq  mig lpa sysc us sy wa id   pc  %ec  lcs
  0    0    0    0   58   82   38    1    0 100   57 26 29  0 45 0.00  0.6  288
  1    0    0    0  340    0    0    0    0   -    0  0 22  0 78 0.00  0.4  288
  U    -    -    -    -    -    -    -    -   -    -  -  -  0 99 0.30 99.0    -
ALL    0    0    0  398   82   38    1    0 100   57  0  0  0 100 0.00  1.0
288
--------------------------------------------------------------------------------
-
  0    0    0    0   57   63   29    1    1 100   30  5 37  0 58 0.00  0.4  275
  1    0    0    0  335    2    1    0    1 100    0  0 27  0 73 0.00  0.3  278
  U    -    -    -    -    -    -    -    -   -    -  -  -  0 99 0.30 99.2    -
ALL    0    0    0  392   65   30    1    2 100   30  0  0  0 100 0.00  0.8
276
```

The most important options we can use with **mpstat** are:

-d	Shows detailed software and hardware dispatchers' metrics.
-i	Shows detailed interrupt metrics.
-s	Shows SMT utilization.

Example 2-6 mpstat SMT utilization

```
# mpstat -s 2 2

System configuration: lcpu=2 ent=0.3

     Proc0
      0.35%
 cpu0    cpu1
  0.21%   0.14%
------------------------------------------------------------------------------
     Proc0
      0.29%
 cpu0    cpu1
```

Modified text based tools

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

vmstat, **iostat** and **sar** automatically use new PURR-based data and formula for %user, %sys, %wait and %idle if SMT is on or if you are in a shared processor environment.

Example 2-7 vmstat monitoring

```
# vmstat 2 3

System configuration: lcpu=2 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 59358 52711   0   0   0   0    0   0   1   30 132  0  0 99  0  0.00   1.5
 0  0 59358 52711   0   0   0   0    0   0   0    8 137  0  0 99  0  0.00   0.9
 0  0 59358 52711   0   0   0   0    0   0   0   13 133  0  1 99  0  0.00   1.6
```

In addition, in this kind of environment it automatically adds two new columns: Physical Processor Consumed (pc or physc) by the partition and Percentage of Entitlement Consumed (pec or %entc) by the partition, which can go as high as 1000% for uncapped partitions.

Regarding **iostat** only, there is a new way to look at asynchronous I/O. You can either check the statistics of “legacy” asynchronous I/O or Posix asynchronous I/O. You can use several flags:

-A Shows CPU utilization and asynchronous I/O statistics.

-q Shows AIO individual queues and their request counts.

-Q Shows mounted filesystems and their associated AIO queue and request counts.

-P Is similar to -A option, but for the POSIX AIO extension data.

Example 2-8 iostat monitoring

```
# iostat 2 2

System configuration: lcpu=2 drives=2 ent=0.20

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
    physc   % entc
          0.0         26.1              27.2      0.7       72.1       0.0
    0.00      31.6

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk1           0.0       0.0       0.0          0         0
cd0              0.0       0.0       0.0          0         0

tty:      tin         tout   avg-cpu:  % user    % sys     % idle    % iowait
    physc   % entc
          0.0        191.3              26.1      0.7       73.2       0.0
    0.00      30.2

Disks:        % tm_act     Kbps      tps    Kb_read   Kb_wrtn
hdisk1           0.0       0.0       0.0          0         0
cd0              0.0       0.0       0.0          0         0
```

When using -A or -P, new columns replace the tty columns:

avgc Average global non fastpath AIO request count per second for the specified interval.

avfc	Average fastpath request count per second for the specified interval.
maxg	Maximum global non fastpath AIO 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 AIO requests allowed on queue.

Example 2-9 iostat "legacy" AIO

```
# iostat -A 2 3

System configuration: lcpu=2 drives=2 ent=0.30

aio: avgc avfc maxg maif maxr avg-cpu: %user %sys %idle %iow physc %entc
        0    0    0    0    0           0.0  0.4  99.6  0.0   0.0   1.0
hdisk1           0.0       0.0       0.0          0         0
cd0              0.0       0.0       0.0          0         0
        0    0    0    0    0           0.0  0.4  99.6  0.0   0.0   1.0
hdisk1           0.0       0.0       0.0          0         0
cd0              0.0       0.0       0.0          0         0
```

While the iostat command is running for Count of iterations and if there is a change in system configuration that affects the output of iostat command, it prints a warning message about the configuration change. It then continues the output after printing the updated system configuration information and the header.

Some system resource is 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.

When looking at the -P ALL (logical processors view) option of the **sar** command with SMT on or in shared mode, it shows a new column: Physical Processor Fraction 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 adds an additional new column automatically called 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. The option -d makes avwait, avserv and avque columns become "real". avque changes from an instantaneous count of "in flight" requests to a real average of the number of requests waiting to be sent to the adapter. avserv becomes the average time a request took before coming back from the adapter and avwait is the average time a request spent waiting to be sent to the adapter.

Example 2-10 sar -P ALL

```
# sar -P ALL 2 2

AIX sq1test1 3 5 00CDDEDC4C00    06/23/04

System configuration: lcpu=2 ent=0.30

11:55:18 cpu    %usr    %sys    %wio   %idle   physc   %entc
11:55:20  0        0       6       0      94    0.00     0.7
          1       16      60       0      25    0.00     0.9
          U        -       -       0      98    0.30    98.4
          -        0       1       0      99    0.00     1.6
11:55:22  0        0       7       0      93    0.00     0.4
          1        3      64       0      33    0.00     0.7
          U        -       -       0      99    0.30    98.9
          -        0       0       0     100    0.00     1.1

Average   0        0       6       0      94    0.00     0.6
          1       10      61       0      29    0.00     0.8
          U        -       -       0      99    0.30    98.6
          -        0       1       0      99    0.00     1.4
```

Regarding **topas** we have a modified screen, the main screen, and a new one dedicated to LPAR. For the main screen we get the new metrics applied and so CPU utilization is calculated using new PURR-based data and formula when running in SMT or shared mode. **Topas** adds them automatically when running in shared mode. The new cpu section metrics on physical processing resources consumed are:

Physc The amount consumed in fractional number of processors

%Entc The amount consumed in percentage of entitlement

Example 2-11 topas main screen

```
Topas Monitor for host:    sq1test1            EVENTS/QUEUES    FILE/TTY
Mon Jun 28 16:13:56 2004   Interval:  2        Cswitch   28623  Readch   165.2M
                                               Syscall  103.0K  Writech  165.1M
Kernel   71.0   |###################        |  Reads     50690  Rawin         0
User      4.7   |##                         |  Writes    50637  Ttyout      208
Wait     16.1   |#####                      |  Forks         7  Igets         0
Idle      8.2   |###                        |  Execs         7  Namei        969
Physc =  0.30                   %Entc= 100.0  Runqueue    2.0  Dirblk        0
                                              Waitqueue   1.5
Network  KBPS   I-Pack  O-Pack   KB-In  KB-Out
en0       0.4     6.0     1.0     0.3     0.5  PAGING           MEMORY
lo0       0.0     0.0     0.0     0.0     0.0  Faults      696  Real,MB     511
                                               Steals      136  % Comp     56.7
Disk    Busy%     KBPS     TPS KB-Read KB-Writ  PgspIn        0  % Noncomp  44.1
hdisk1   98.2    627.3    86.4     0.0  1264.0  PgspOut       5  % Client   46.5
hdisk0   85.3    341.4    72.5   504.0   184.0  PageIn       70
cd0       0.0      0.0     0.0     0.0     0.0  PageOut     112  PAGING SPACE
                                               Sios        183  Size,MB     512
Name            PID  CPU%  PgSp Owner                          % Used      8.6
dd           581878   9.9   0.1 root            NFS (calls/sec)  % Free     91.3
dd           491566   2.3   0.1 root            ServerV2       0
errdemon     147540   0.2   0.6 root            ClientV2       0   Press:
backbynam    417962   0.1   0.2 root            ServerV3       0   "h" for help
restbynam    458804   0.1   1.0 root            ClientV3       0   "q" to quit
```

The new LPAR screen is accessible from -L or the **L** command. 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 processor with **mpstat** columns. The %hypv and hcalls give you the percentage of time in POWER Hypervisor and number of calls made. The pc is the fraction of physical processor consumed by a logical processor. When in shared mode there are additional metrics:

Poolsize	Number of processors in LPAR pool this partition belongs
App	Available pool processors
Physc	Number of physical processor(s) consumed
%entc	Percentage of entitlement consumed
%lbusy	Logical CPU utilization
lcsw and vcsw	Logical and virtual context switches
phint	Number of phantom interrupts

Example 2-12 topas -L

```
Interval:    2    Logical Partition: Test1_AIX_0425A   Wed Jun 23 17:08:51 2004
Psize:   0                    Shared SMT  ON           Online Memory:   512.0
Ent: 0.30                      Mode: Capped            Online Logical CPUs:  2
Partition CPU Utilization                              Online Virtual CPUs:  1
%usr %sys %wait %idle physc %entc %lbusy   app vcsw phint %hypv   hcalls
   4   10     0    86   0.2  69.3  14.25  0.00 8152   304   0.0        0
================================================================================
LCPU  minpf majpf  intr   csw icsw runq lpa scalls usr sys _wt idl   pc   lcsw
Cpu0      0     0  1822     1    1    1 100      0   0  13   0  87 0.05   3457
Cpu1      0     0  2291  5697 2846  429 100  33471  23  48   0  29 0.16   4695
```

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 shared processor environment. All trace based tools will adjust cpu times using preemption hook. In addition, most hcalls are traceable. This means they will appear in **trcrpt** output.

curt and **splat** can optionally use the PURR values to calculate cpu 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 shared processor environment. It also shows the 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 CPU. There are new NFS reports in **curt**. It now adds new category in System and Processor Summary reports, introduces new type of kproc (N), and marks all NFS kproc. In these reports, **curt** also adds new NFS Calls Summary section with V2 and V3 sub-sections and adds similar sections in process and thread reports.

Example 2-13 curt preemptions, H_CEDE and H_CONFER

```
Total number of preemptions = 1022
Total number of H_CEDE      = 0      with preeemption = 0
Total number of H_CONFER    = 0      with preeemption = 0
```

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.

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 fix number of CPU's 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.

Graphical tools

As with text based tools, there were problems in the graphical tools regarding the shared mode environment and SMT. They were modified.

PTX 3dmon

The most complete graphical tool, PTX, now monitors the full CEC. AIX 5L supports this monitoring but Linux does not, although there are plans for it. This means that if you have a Linux partition then the full CEC monitoring is not yet available. When using this option it lists all hostnames from a physical machine together. PTX has another option to show automatically all partitions in a physical machine where the input is the hostname of one of the partitions. For Micro-Partitioning PTX uses PURR-based utilization metrics and entitlement utilization. It also does automatic scans for new partitions from the same CEC every minute and adds the newly discovered partitions to display.

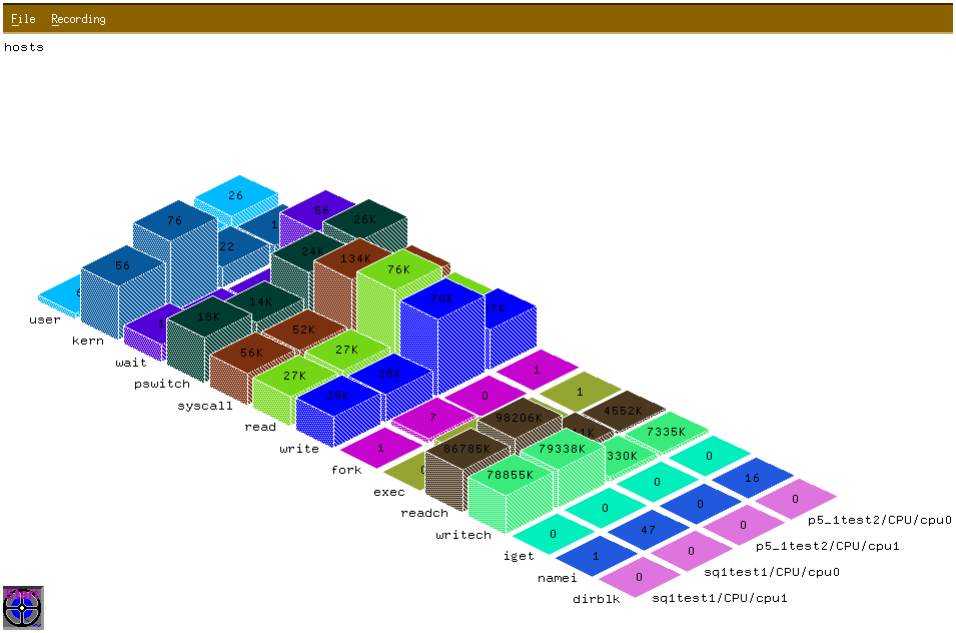


Figure 2-30 3dmon cpu monitoring two LPARs

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 scrollbar. **jtopas** uses **xmtrend** daemon.

trace GUI viewer

Trace GUI viewer is a Client-server GUI version of **trcrpt** where the client does the displaying and the server processes the trace file. Since this is a graphical tool, there are several advantages over **trcprt** namely a slide bar and a dynamically customizable output in which you can add, delete or move columns around. The client works remotely on any platform supporting Java 1.3.1, including on Windows. When navigating through trace file, you may bookmark a specific entry and later seek to any bookmarked entries. You can also use predefined bookmarks to facilitate quick navigation. When the trace GUI viewer reads in a trace, it automatically generates these predefined bookmarks. Another advantage over **trcrpt** is the use of filters. They use a fairly simple syntax for example: ((Hookid = 0x11f) OR ((Pid = 11345) AND (Hookid = 0x104))). You can combine multiple filters. Once you select them, you may apply the filter to one or more subtraces. Each subtrace is a split of 10000 hooks of the original trace. Paging down the trace file yields entries matching the filter until you select a new filter by seeking to another bookmark.

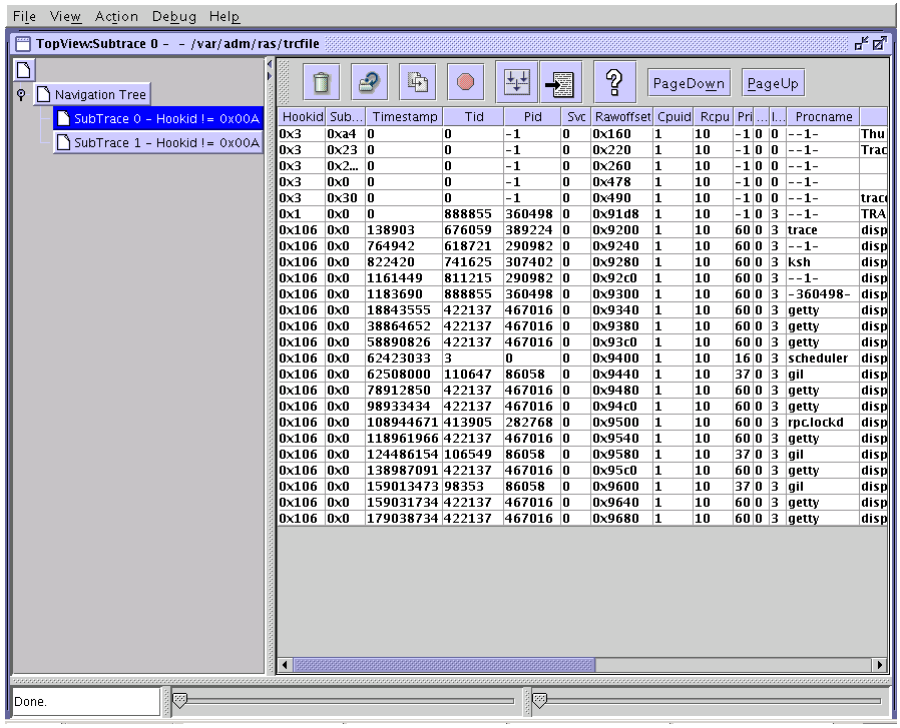


Figure 2-31 trace GUI viewer - tgv -client

PMAPI

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 POWER 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 mode as opposed to the old 1:1 mode. There is a new set of APIs for third party calls (debuggers) 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, **`pmlist`** for example. The **`pmlist`** is a utility to dump and search processors event and group tables. It currently supports text and spreadsheet output formats.

2.9.4 Logical Volume Manager

There have been several improvements to the AIX 5L Logical Volume Manager that concern performance. The first is the fact that we now have a new type of volume group called Scalable Volume Group (SVG). The second is that there are new ways to read and write metadata. AIX 5L v5.3 writes all metadata in parallel. There is one thread for each disk in the vg. In addition, some commands that would read data, utilize a small piece, then read again, utilize a small piece, etc. now read the metadata once and keep the metadata accessible throughout the life of the command.

The new SVG supports 1024 Disk VG. This expands the capacity of the volume groups but needs a substantially larger VGDA space. Every VGDA update operation (creating a logical volume, changing a logical volume, adding a physical volume, and so on) might take considerably longer to run. In addition, increasing max lvs or max pps/vg beyond the defaults towards the limits increases the amount of metadata that must be read / written during lvm operations. The larger SVG gets, the slower it gets. The limits for each type of volume group are the following:

Table 2-5 vgtype limits

vgtype	PVs	**LVs**	**PPs/VG**
oldvg	32	256	~32 k (32*1016)
bigvg	128	512	~128 k (128*1016)
svg	1024	4096	2048 k (2097152)

Prior to AIX 5L v5.3 there was no good way to extend a striped lv if one of the disks is full. We needed to backup the data, delete the striped lv, remake the lv 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 upperbound. Prior to AIX 5L 5.3, the stripe width and upperbound were required to be equal. In AIX 5L v5.3, the upperbound can be a multiple of stripe width, where you can think of each stripe as a "column". You can use **extendlv** to extend a striped lv into the next column. If you use **extendlv -u** then you can raise the upperbound and extend the lv all in one operation (like an **extendlv** and a **chlv -u** all in one).

Ltg stands for logical track group. The LVM device driver breaks io down into ltg size chunks before passing the io down to the device driver of the underlying disks. The ltg size is an attribute of the vg. AIX 5L v5.3 allows the stripe size of an lv to be larger than the ltg size of the vg, which didn't allow in previous versions. In addition, AIX 5L now supports larger ltg sizes and stripe sizes.

Table 2-6 ltg and strip valid sizes

	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

When it comes to pbuf, AIX 5L v5.2 and earlier versions had a systemwide pbuf pool. Now, each vg gets its own pbuf pool. To manage pbuf we use the lvmo command which displays and tunes several vg specific items:

pb_pbuf_count number of pbufs added when a pv is added to the vg (tunable w/**lvmo**, takes effect at lvmo time)

total_vg_pbufs number of pbufs currently available for the vg (tunable w/**lvmo**, takes effect at varyonvg time)

max_vg_pbuf_count maximum number of pbufs for this vg (tunable w/**lvmo**, takes effect at varyonvg time)

pervg_blocked_io_count

number of io's that were blocked due to lack of free pbufs for this vg. (displayed only, not tunable)

The lvmo also displays the following systemwide items:

global_pbuf_count min # of pbufs that are added when a pv is added to any vg. (tunable w/**ioo**, not **lvmo**, because this is systemwide. Takes effect at varyonvg time).

global_blocked_io_count

systemwide # of io's that were blocked due to lack of free pbufs

2.9.5 Partition Load Manager

Partition Load Manager (PLM) for AIX 5L is a load manager that provides automated CPU 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 and shared processor environment partitions. The only restriction is that all partitions in a group must be of the same type.

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 several events on every required LPAR node. In order for PLM to get system information and dynamically reconfigure resources, it will require an SSH network connection from the managed AIX 5L partitions to the HMC. The Resource Management and Control (RMC) services are responsible to gather all the status information. The RMC daemon exports system status attributes and processes the reconfiguration 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.

Figure 2-32 PLM view using PTX

PLM uses a Micro-Partitioning entitlement model with a guaranteed or entitled amount of resource, a shares amount and an optional minimum and maximum amounts. The guaranteed amount of resources is the amount allocated to a partition that has a demand for it. It can get the resource from free pool if available and group not over its maximum, take underutilized resource from other partitions or take utilized resource from partitions over their guaranteed resource. The allocated resource will vary between minimum and maximum values based on demand. For a partition to allocate resources above the guaranteed amount, it needs to know the shares amount. This amount has a unit less factor between 0 and 255 with a zero preventing the allocation of resources above guaranteed amount. The formula to calculate the resource allocated to partitions is (shares) / (sum of shares from competing partitions).

PLM has manages partitions in groups which means that all partitions must be a member of a group. At least there must be one group defined in the PLM policy. One PLM server can manage independent groups of partitions but it cannot share resources between 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: either they are of the shared

processor type or of the dedicated processor type. In spite of that, one group may contain both capped and uncapped partitions. PLM manages the entitled CPU capacity, memory, and number of virtual processors for both types.

System administrators must set up HMC partition definitions compatible with the PLM policy. The XLPLM is not able to decrease a partition's minimum below the HMCs minimum nor is the XLPLM 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.

Example 2-14 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
```

To manage the PLM you can use two commands: **xlpstat** and **xlplm**. With these two commands, you can monitor partition statistics and start, stop, modify, reserve, and query a cross-logical PLM server.

Example 2-15 xlplm query

```
# xlplm -Q default
PLM Instance: default

GROUP: example
                  CUR       MAX      AVAIL      RESVD      MNGD
       CPU:      0.80      2.00       1.20       0.00       Yes
       MEM:      1536         0          0          0        No


  p5_1test1

    RESOURCES:
                  CUR       MIN       GUAR       MAX       SHR
       CPU:      0.30      0.10       0.30       0.50         2
       MEM:       512       128        512       1024         1

  p5_1test3

    RESOURCES:
                  CUR       MIN       GUAR       MAX       SHR
       CPU:      0.50      0.10       0.30       0.50         2
       MEM:      1024       512       1024       2048         1
```

You can also use Web System Management tool (WebSM) to manage PLM.

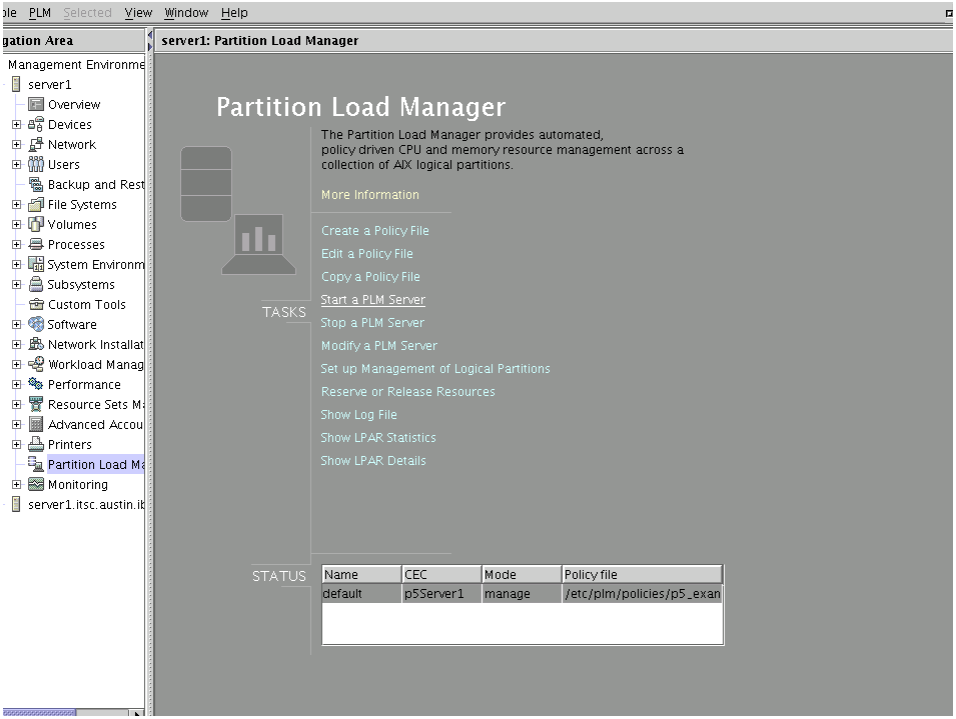


Figure 2-33 PLM management using WebSM

With this tool you can view partitions cpu statistics and and memory statistics.

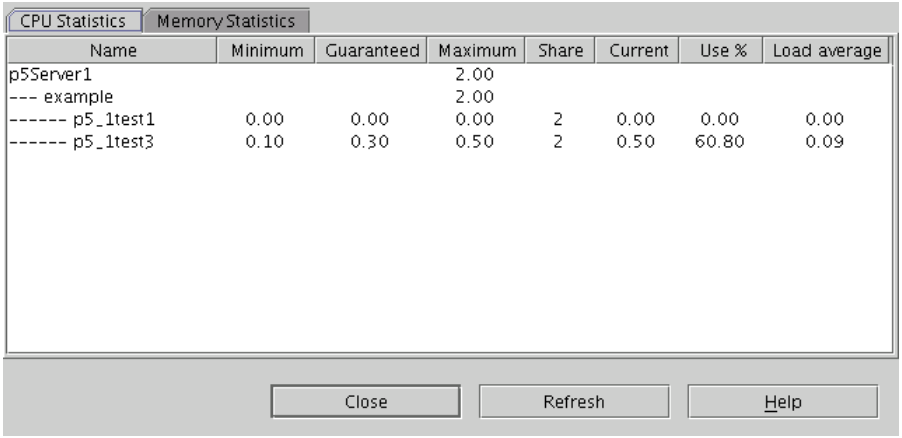


Figure 2-34 PLM cpu statistics using WebSM

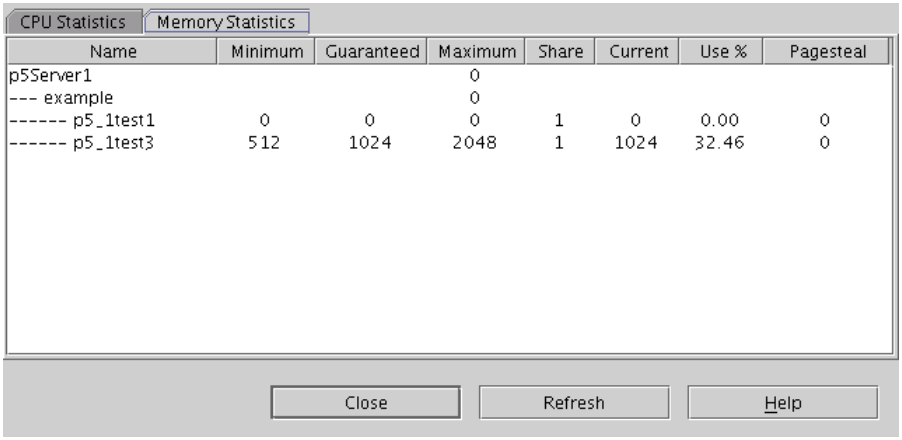


Figure 2-35 PLM memory statistics using WebSM

3

Simultaneous Multi-Threading

SMT was briefly introduced in chapter 1. This chapter talks about the software considerations and optimizations to effectively utilize SMT.

3.1 Idea behind SMT

As processor fabrication technologies evolve, transistor densities are on the rise, which means chip designers can put in more resources into the chip. Processor designers can now put larger on chip caches, integrate more functional units that were carried out traditionally by separate chips (like memory controllers etc.). Just having large amounts of cache and higher levels of systems integration alone won't help improve the processor performance. Ultimately, the processor's performance will be measured by the amount of instructions executed over a period of time. Hence, it makes sense to have more execution resources on the processors. Modern superscalar processors do just that. They provide for more execution resources and can execute multiple instructions at the same time[1] in parallel. This means that more execution units are available on the chip, and if the executing program (a sequence of instructions) can be split into instructions that can be executed in parallel by these execution units, then the number of instructions executed over a period of time increases, and hence processor performance goes up. But, the processor cannot arbitrarily split a program into parallel instructions -- the processor should take care as to not break the sequential programming model assumed by the software, and honour instruction dependencies. If a sequential program can be split up into parallel executable instructions, such a program is termed to exhibit instruction level parallelism. The degree of instruction level parallelism is program dependent, or rather, workload dependent. Hence, putting in more execution resources will help better performance, but it is limited by the instruction level parallelism of the workload. This is a significant drawback in traditional superscalar processors, since all workloads (and commercial workloads in particular) don't exhibit higher instruction level parallelism, resulting in processor execution resources going under utilized.

Now, what if, instead of one program executing on a processor at a time, if two or more programs could run on the processor at the same time -- then execution resources not utilized by one program could be utilized by another program ready to run on the same processor. So by just providing for different 'threads' of execution on the processor (hardware contexts), the superscalar is no longer limited by the degree of instruction level parallelism on one thread. The processor exploits thread level parallelism to compensate for the low instruction level parallelism of individual threads of execution. A Hardware context just provides the processor architected software model to programs. That is, the architected processor registers are replicated to create more hardware contexts. The technique of sharing resources on a single processor core among many execution threads (contexts) is known as multithreading. Note that this is different from the term multithreading used in software. Each hardware context is

[1] By definition, a super scalar processor can issue a number of instructions each cycle. The capability to issue n instructions per cycle is also termed as the issue width.

seen by the software (operating system) as a separate processor (although it is not a separate physical processor). These hardware execution contexts are sometimes termed as logical processors or logical cpus.

There are different kinds of multithreading in computer architecture literature. They can be broadly classified into coarse grained multi threading, fine grain multi threading and simultaneous multi threading.

In coarse grain multithreading, one thread known as the active thread, executes on the processor at one time while the other thread[2] 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. For such threading mechanisms 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 effectively becomes difficult as the processor pipeline depth increases.

In fine grain multithreading, 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 multi 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. 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.

[2] Assuming two way multithreading. A processor is said to be n-way multi threaded if n-hardware contexts are provided for the software.

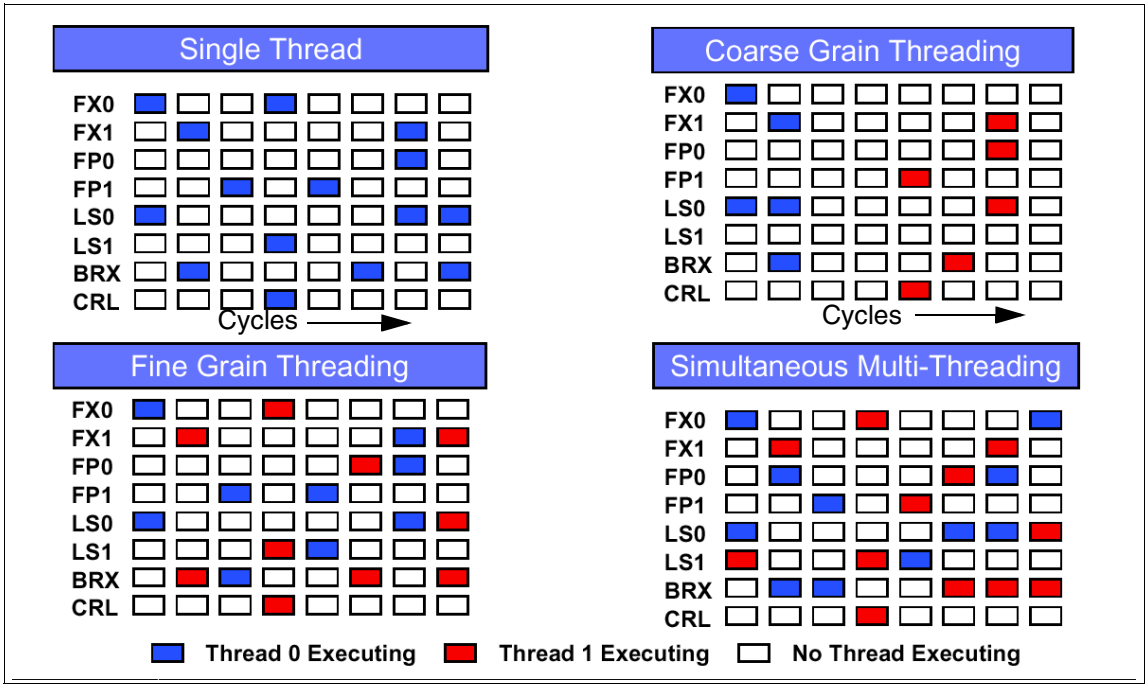


Figure 3-1 Different multithreading models

Figure 3-1 on page 92 shows the processor execution resource utilization under different threading environments every cycle. 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.

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

3.2 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 cpu SMP for 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 and adjustable thread priorities for efficient resource utilization of the resources shared by both the threads.

Dynamic resource balancing logic helps the two hardware contexts on the core execute smoothly. If one of the threads starts hogging the shared execution resources, such as GCT entries, issue queue slots etc., starving out the other thread, the DRB logic kicks in and throttles down the thread hogging resources by reducing the its priority or holding the thread from decoding instructions or flushing all the instructions waiting for dispatch. The throttling method used is dependent on the source of the stall -- if GCT entries were being hogged by one thread, then the thread priority would have been reduced. DRB has been discussed in detail on section xxx

Add reference to drb logic section in chapter 2 above

Adjustable thread priorities provides for the software to throttle down or throttle up execution of either thread on the processor core. The POWER5 provides for eight levels of thread priorities, 0-7. A thread priority of zero implies ST mode, and the hardware does not allocate any architected registers to the thread. Architected registers are maintained for all other priority levels in the hardware. A thread priority of 7 is the highest. Software (phyp/operating system/applications) can specify certain thread priorities based on their execution privilege level. Refer to table xxx and section xxx for a detailed discussion of adjustable thread priorities. Software can change the thread priorities by executing or x,x,x noops or using mtspr instruction to write to the thread's TSR (thread status register).

Add cross reference to table showing thread priorities in chapter 2

The POWER5 provides for the software to dynamically switch from SMT mode to ST 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 ST 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 a SMT thread executing the idle loop needs at least the

architected registers from the rename resource pool (GPRs, FPRs etc.), the performance difference of a task when it is run in ST mode to when it is run in SMT mode when the other hardware thread is running the OS idle loop could be as significant as 10-15%. This is mainly due to the execution resources consumed by the idle thread.

The threads on a POWER5 can be in just two states -- live or dead. Hardware maintains the architected state of the thread for a live thread, and the execution resources are shared as determined by the adjustable thread priority and DRB mechanisms. For a dead thread, there is no architected state maintained by the hardware. This means more rename resources (32 + 4 GPRs more etc.) for the 'other' thread. The switch from SMT to ST can be done by the software (hypervisor or operating system). The operating system has to use the appropriate hypervisor call for that. The power hypervisor in turn uses a special mtctrl instruction to kill an active thread in the hardware. The operating system could revive a dead thread (switch the core back to SMT from ST) by using a hypervisor call which again uses the mtctrl instruction to do that. The service processor can also switch a processor core to SMT. Decrementor interrupts and external interrupts could also awaken threads, based on how the system is set up (special register settings).

3.3 Software considerations for SMT

In the hardware, a smt hardware context (thread) could just have two states -- live and 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 in when the thread is dead both in the hardware and software. The software does not maintain any per-processor structures/data and the dead thread is set not to wake up on DECR or external interrupts. To revive a dead thread, the sibling thread is expected to execute the mtctrl 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-cpu data etc. 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.

3.3.1 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 ST 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_CEDE hypervisor call (refer to section xxx on chapter 2). The thread then goes to the dormant state. A snoozed thread is brought alive when a Decrementor, external interrupt or a H_PROD is received for the thread (refer hypervisor section xxx chapter2). 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 ST mode to SMT mode through any of the means mentioned earlier. This involves the snoozed thread beginning life at its SRI (system reset interrupt) vector for the thread, and having the power hypervisor restore the operating system state, and then returning the original H_CEDE hypervisor call made by the thread to snooze. This means several thousand cycles of thread startup latency. Hence, it doesn't make sense to snooze a thread as soon as idle condition is detected -- there could be another task ready in the runqueue by the time you snooze 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.

3.3.2 Process accounting

With ST 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 cpu resources with its sibling. POWER5 provides for Processor Resource Utilization Register (PURR) to provide proper process/system accounting. PURR for system accounting on AIX has been discussed in section xxx (2.9.2 chapter2).

PURR is a new per thread register introduced in POWER5. It is an incrementing 64 bit counter just like the TB. It gets incremented once in eight 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 CPU time in the SMT environment, since the sum of the two PURRS is close to the TB ticks over the same period of time.

3.3.3 CPU utilization

There are different approaches to reporting CPU utilization in an SMTenvironment.

One approach is to treat each hardware (SMT) thread (logical cpu) as a separate processing engine. Under this approach, the CPU utilization metric represents the proportion of time that work was dispatched on the logical cpu. The CPU utilization reported using this approach does not necessarily reflect the logical processing engine's utilization of the processor's physical execution resourcessince it does not account for factors such as the relative hardware priorities of the two SMT threads etc.[3]

The second approach, used by AIX, is to report CPU utilization from the perspective of the actual physical processor resource utilization. Under this approach, the CPU 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 distrubition of physical processor resources to both the hardware threads).

[3] 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

3.3.4 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 cpus (four logical threads) and two runnable tasks, the scheduler could schedule tasks on two sibling threads of the same cpu and keep the other core (cpu) totally idle. Since the os is not SMT aware, there is no way the scheduler can distinguish between threads on the same cpu and different cpus. 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 primary[4] 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 TLB, L1 etc., 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.

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.

3.3.5 Interrupts

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

3.3.6 Effective use of adjustable thread priorities

POWER5 features Adjustable thread priorities for better processor resource utilization. The feature has been explained in detail in section xxx 2.3.2 of

[4] 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 smt threads enjoy equal access to the execution resources with other factors like thread priorities being equal.

chapter 2. To summarize, POWER5 provides for eight levels of thread priorities -- 0-7, Please refer to table xxx 2-3 for all the supported priorities. Ratio of decode slots allocated to a hardware thread is dependent on the thread priorities of the sibling threads. table xxx 2-4 depicts the effect of thread priorities on execution resource sharing. The operating system can set priorities from 1 to 6, which correspond to 'very low' to 'high' priorities. Application programs (user space) can set thread priorities from 2 to 4, which correspond to 'low' to 'normal' priorities.

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

AIX

AIX will lower the thread priority for idle threads, so that the other working sibling thread can execute faster -- or power savings is achieved at least. 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 period before snoozing the idle thread by means of a H_CEDE hypervisor call.

For AIX instances in a micropartitioning environment, AIX always invokes H_CEDE hcall.

If a task in the kernel is waiting for a spinlock, AIX changes the thread priority of the hardware context executing that task to '2' -- 'low' priority; so that the spinning thread, which is not doing any useful work yields processor resources to it's sibling. For instances of AIX running in micropartitioning environment, AIX waits for a spin delay after lowering priority to 2 and then invokes H_CONFER hcall. POWER hypervisor will control priority management and redispatch the physical cpu if the sibling thread also cedes or confers.

AIX also boosts priority of threads which are executing tasks holding certain critical and hot (contended) spinlocks. AIX will boost the priorities of the threads executing these tasks to '5' -- 'medium high' so that lock holders execute faster thus reducing lock hold time and paving way for increased application throughput. The priority of the thread is restored to '4' -- 'normal' in the corresponding unlock function. If a boosted thread is interrupted, the thread priority will be boosted to '4' -- 'normal', if the current thread priority is less than '4'. Otherwise, current thread priority is preserved. Hence, the priority boost for these hot locks also boosts priorities of interrupts and exception handlers running on the thread which was holding the hot lock.

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 micropartitioning 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.

3.4 Cache effects due to SMT

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 icache and dcache 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.

3.5 Performance benefits due to POWER5 SMT

Performance measurements for various standard industrial benchmarks were made with AIX 5.3L on 4way p-series 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.

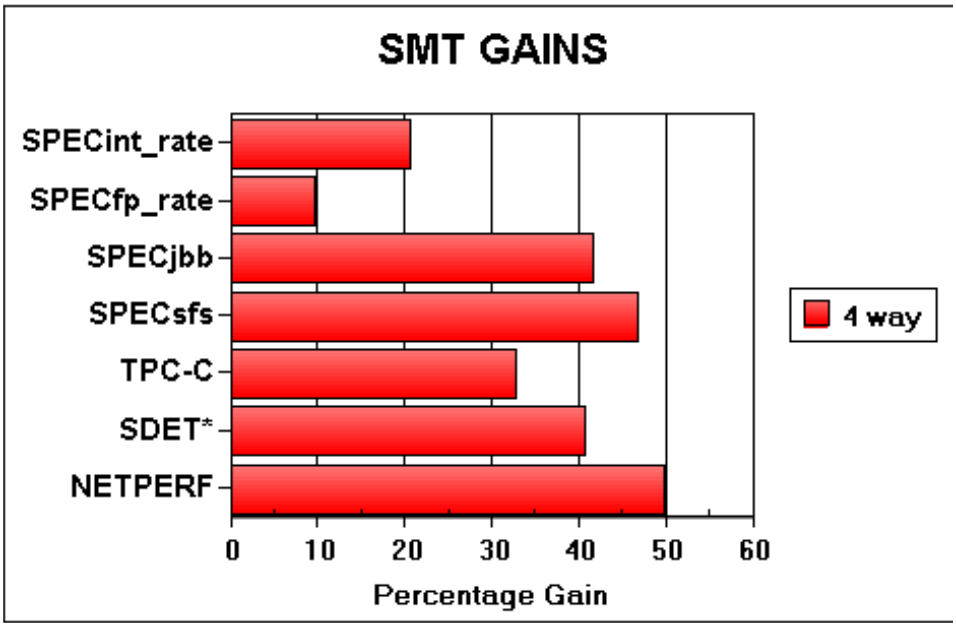


Figure 3-2 SMT gains for various workloads

Figure 3-2 on page 100 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.

3.6 Conclusion

To summarise, the POWER5 SMT implementation is more than just SMT. Unlike other commercial SMT implementations, POWER5 SMT combines features such as dynamic resource balancing, adjustable thread priorities and hardware support for dynamic SMT switching which help the software get more out of the silicon. It has found to be a good ROI for the extra silicon area -- that is, performance gain out of SMT is greater than the area increase due to SMT. There is a 24% area growth per core for SMT, but the performance gains due to POWER5 SMT can vary from 10-50% or even more depending on the workload and machine configurations.

4

Virtualization

In this chapter we discuss performance considerations when using micro-partitioning, as well as the Virtual I/O Server, Virtual Ethernet and Virtual SCSI. We discuss some effects that the partitions may have based on the configuration, and suggest some guidelines when configuring a system with these components.

4.1 Micro-partitioning considerations for performance

There are some considerations that must be noted when using shared processor partitions in order to obtain a good performance and system utilization. There are some options when configuring a partition that may affect performance, and some guidelines that are recommended in order to get the maximum benefit from the technology.

4.1.1 Micro-partitioning overhead

Micro-partitioning adds an additional layer of abstraction by virtualizing the physical processor into a virtual processor. The virtualization of the processors adds great flexibility in using the system and fractional processing power, but may result in consumption of more cycles for every instruction completed.

There is an impact of running an application inside a shared processor partition, but in most cases the impact is negligible.

In an NFS test, throughput was measured on 4 dedicated processor partitions, each one with one physical processor. The result was then compared to the throughput of 4 shared processor partitions with 1.0 entitlement per partition. The throughput was the same in both cases. Processor usage was about 2% higher in the case of shared processor partitions. The same overhead was observed in another test conducted running a Java-based application server with Websphere and DB2 in an uncapped partition.

Note that 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 overhead.

These measurements are indicative of the minimum overheads associated with the most simplistic micro-partitioning environment.

4.1.2 Simultaneous Multithreading and micro-partitioning

Simultaneous Multithreading (SMT) is a function implemented in the processor, that the operating system must be aware of in order to exploit. Hardware threads are seen by the operating system and applications as two distinct processors. SMT behaves the same way either on dedicated processor partitions or shared processor partitions, with a few differences explained here.

Dynamic switching between SMT and Single Threading

When working on a partiton with SMT enabled, work is dispatched to all primary hardware threads before the secondary threads. This helps optimal performance

in case of single threads running on a processor. Since the secondary threads of not get work to be done, they get into a *snooze* state and the primary thread runs on almost single thread performance.

In dedicated processor partitions, the POWER Hypervisor can dynamically transition the processor from SMT to ST, depending on the workload running. In case a thread running on a processor turns idle, the processor is changed to Single Threading (ST) mode, and the running thread benefits of full singe thread performance. When the other thread is runnable again, the processor returns to SMT mode and run both threads.

This is not supported on shared processor partitions however, and these must be explicitly changed by the **`smtclt`** command. On shared processor 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 <xref to SMT>, SMT enables two hardware threads to run simultaneously. For a processor to cede its idle cycles to the POWER Hypervisor, 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 give back to the POWER Hypervisor.

This behavior is more perceptible when partition 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 CPU utilization (that is seen as processing capacity consumed by the threads inside the partition). Figure 4-1 illustrates this effect observed when running a Java-based application server with Websphere and DB2.

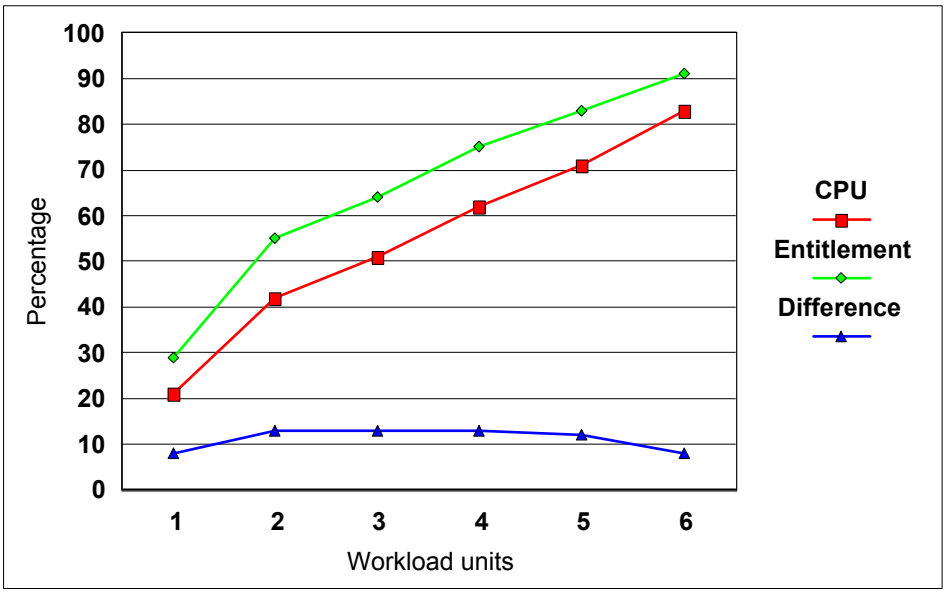


Figure 4-1 Effect of an SMT idle thread on shared processor partitions

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 ST mode.

4.1.3 Cache architecture and number of virtual processors

On POWER5 systems, two CPU cores on the same chip share the both 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. Therefore, the cache usage ratio depends on the workloads on each partition, specifically on the memory access patterns from the applications on each of the two partitions.

In shared processor 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 POWER Hypervisor is responsible for maintaining affinity between virtual and physical resources in a shared processor partition environment. When dispatching a virtual processor onto a physical processor, the POWER Hypervisor tries to redispatch a partition 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 shared processor 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 POWER 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 4-2depicts the flow of actions described.

Even in the case where 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. Depending 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 can be affected when running in a shared processor partition along with other partitions that do intensive memory access. High performance computing (HPC) applications are more likely to have intense memory access behavior.

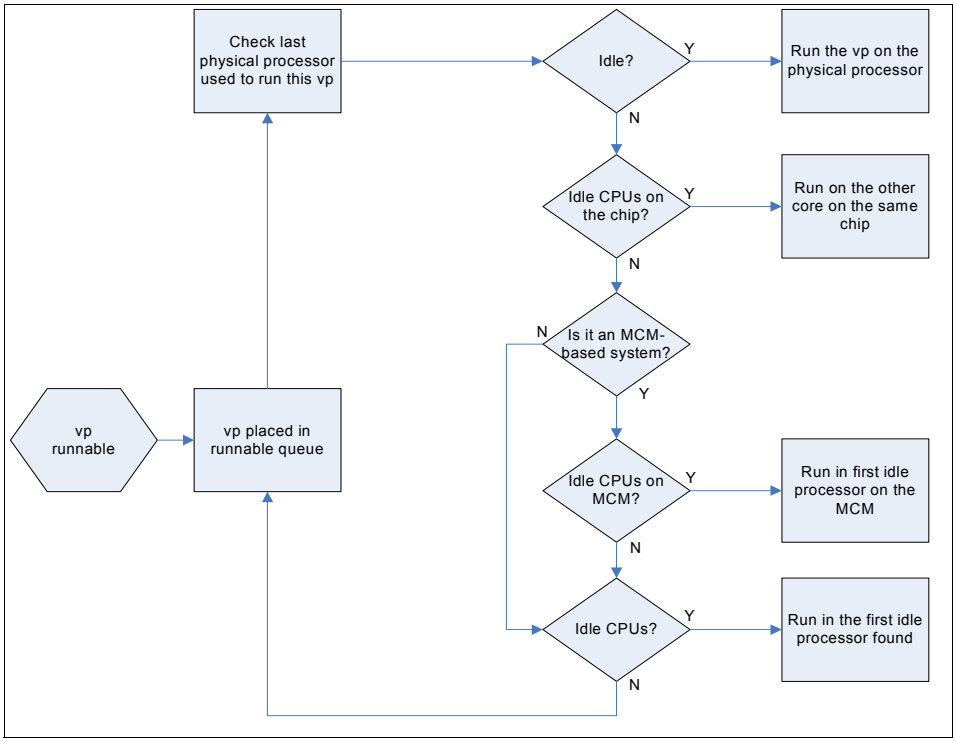


Figure 4-2 Affinity relation between virtual and physical processors

Number of virtual processors

In the case where the number of virtual processors is much larger than the number of physical processors, the time slice given to a virtual processor gets smaller as the number of virtual processors increase. This also contributes to diminishing efficiency in cache, since for each virtual processor that is dispatched, the memory context changes, and data must be reloaded from memory.

When virtual processor capacity is small, the overhead 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 4-3 shows a case where two partitions, each one with one virtual processor and same processor capacity, are running on a system with one physical processor. Partitions A and B use the physical processor 5 ms each. Even if data is loaded from memory into the cache, it remains in the same

memory context for the duration of half of the dispatch wheel. 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 processor capacity. In this case however, each partition only takes 2 ms of the dispatch wheel. Not only 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 becomes completely invalidated between the time a partition is dispatched twice.

.

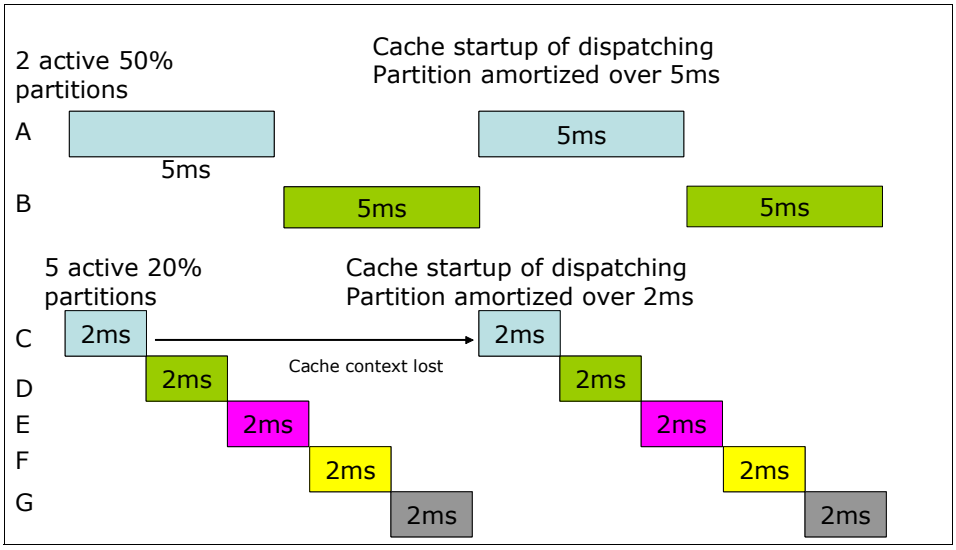


Figure 4-3 Impact on cache due to number of virtual processors

Effects of cache-friendly and cache-unfriendly applications

The effect of the cache reload in the partition performance depends on the size of the partition, number of virtual processors and type of application. A worst-case scenario is a partition that uses the cache moderately running on the same physical processor of another partition that uses memory and cache intensely. This is represented on for a case where a benchmark application A (composed by small but numerous tasks, involving process creation and termination) sensible to cache efficiency is run on the same processor as other partition running a benchmark application B that uses memory heavily for reading and writing large blocks of data.

In all cases, each partition has 2 virtual processors, each with 0.1 processor capacity. The partition running benchmark A is uncapped, while the partitions running benchmark B are capped.

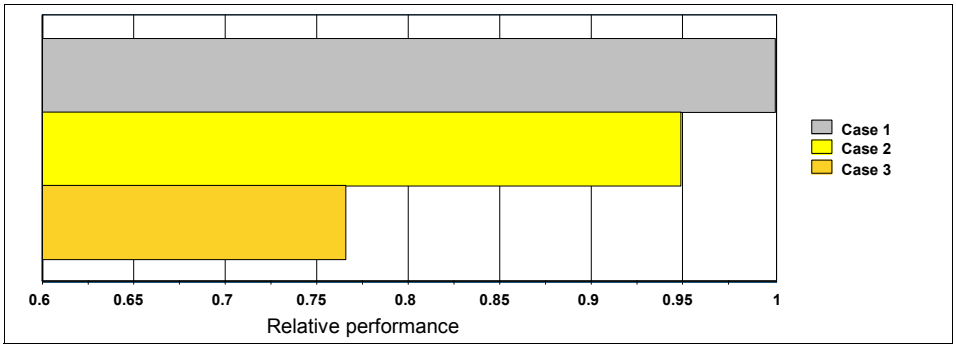


Figure 4-4 Measurements of cache effects in different partitions

Case 1 shows the throughput for the reference benchmark when A runs without other partitions running at the same time, and servers 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 aggressive to the cache, 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 have a smaller impact in performance.

4.1.4 SMP locking and number of virtual processors

In a shared processor partition, virtual processors are dispatched into physical processors by the POWER Hypervisor. Ths dispatch wheel assures that every virtual processor is dispatches in a 10 ms interval. It does not garantee, however, that they are dispatched simultaneously in case you have more than one physical processor. Because of this, additional locking contention is possible. SMP locks need special handling in shared processor partitions, since a virtual processor can be waiting for a lock release from another virtual processor. If the virtual processor owning the lock is not running (because it has ceded its cycles, or

have finished the entitled time slice), then the other virtual processor will spin waiting for the lock. To effectively solve this situation without spending unnecessary cycles, the virtual processor waiting for lock calls the h_confer() hypervisor call to give its cycles to the virtual processor owning the lock, so that it can 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 ST mode may reduce the impact.

Figure 4-5 shows the relative performance of various configurations when running 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 SMP partition to 4 1-way partitions, aggregate throughput is increased by a small margin.

For the same amount of virtual processors, it is better to have more partitions with less virtual processors inside, for aggregate throughput.

As we increase the number of virtual processors, the relative performance is more impacted, because of the SMP scaling inside the partition, and also because of the time used by the POWER Hypervisor to dispatch the multiple virtual processors in the system.

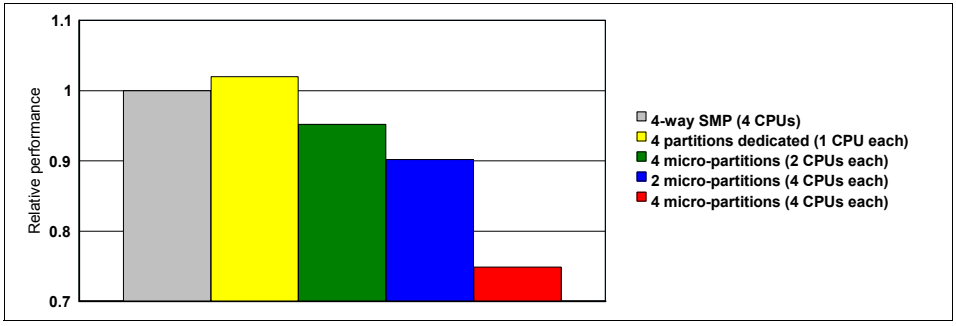


Figure 4-5 The effect of multiple virtual processors in overall performance

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

4.1.5 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 shared processor 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 shared processor 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 shared processor partitions.

4.1.6 Idle partition overhead

In a system running shared processor partitions, the POWER 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 idle (no work being done), the unused processing cycles are then dispatched to other partitions by the POWER Hypervisor, therefore leading to a more efficient usage of the 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 this, an idle partition still present some load to the physical processor. Moreover, the POWER 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 t he work that has to be done). AIX version 5.3 implements some timer-management functions to minimize resource consumption by the idle partitions. This amount is less than 1%. Figure 4-6 shows the impact of adding idle partitions to a system running a workload in one uncapped partition.

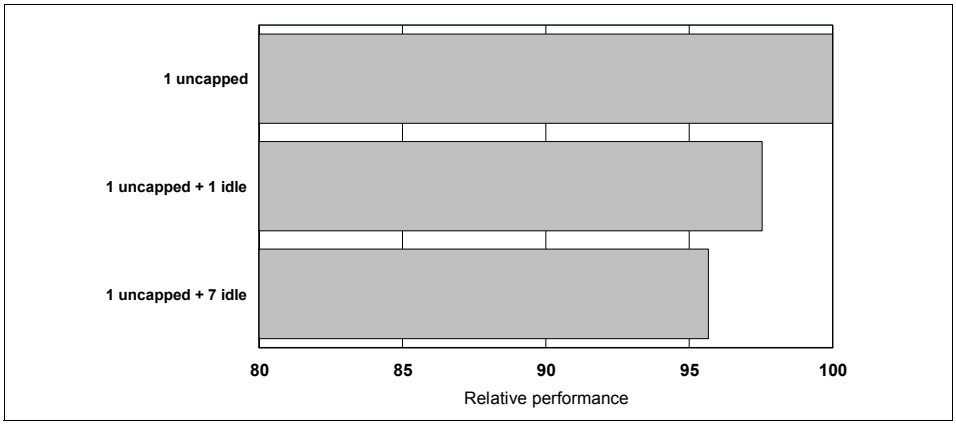


Figure 4-6 Performance of an uncapped partition when adding multiple idle partitions

Since idle partitions are not doing any productive work, in order to reduce further the overhead 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 ticks get 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 cpu 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).

4.1.7 Partition size and overhead

As described in the other sections there is an overhead associated with the implementation of shared processor partitions. If a partition is defined with a small capacity entitlement, the effect of the overhead becomes more significant. While it is possible to configure partitions with entitlements of 0.1 physical processors, they can experience a performance penalty when running in a system with other active partitions, specially if the other partitions are with a moderate to high processor usage. The impact on the small partition can be as high as 40% of its processing capacity.

Larger partitions may be less sensitive to this effect, and should not perceive significant overhead relative to their processing capacity. It is important to note that if a partition is running less than 100% processor utilization, it is giving

cycles back to the POWER Hypervisor. This means that there are more context switches for virtual processors, for underutilized partitions.

4.1.8 Interactions between partitions with high processor usage

Partitioned systems running high performance applications tend to have the partitions competing against each other for shared resources (like cache, system buses, memory access, I/O). This is true for any partitioning mechanism utilized, but care must be taken when running shared processor partitions, since even the physical processors are shared by the partitions.

As stated in the previous sections, there are some factors that can affect the performance of an application running in a shared partition.The behavior of the application itself, partition size, number of virtual processors and system utilization can all impact the performance. The impact varies based on the listed factors.

On a test running a Java-based application server with Websphere and DB2, the performance of a single partition running in the system (consuming 20% of the processor resources) was 25 units. When four additional partitions were added to the system (consuming all the processor resources), performance measured in one partition (but with all partitions running) was reduced by 20%. The aggregate throughput of all partitions was also 20% below the theoretical linear scalability. Figure 4-7 illustrates the test scenario.

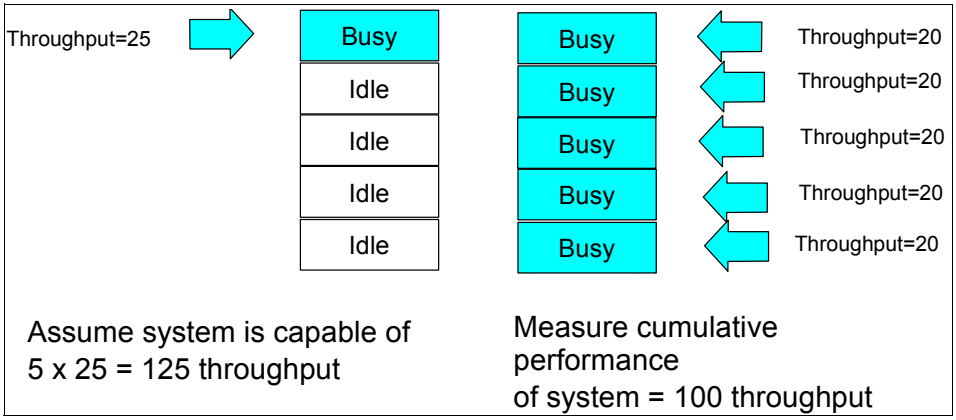


Figure 4-7 Effect of multiple partitions in a high system utilization

As stated before, the overhead is dependant on the workload being run in the partitions. On another test the CPU intensive applications running, the impact was less significant when running the same configuration was less than 8%.

When planning for several shared processor partitions running at high utilization, it is recommended to consider the required capacity from 20% to 25% higher than the actual (or measured) performance requirements. Upon deployment, you can monitor the system and adjust the partition entitlements to the correct needs. If the environment requires running a very large number of partitions, it may be necessary to increase the allocation of entitlement further to accommodate overhead.

4.1.9 Application considerations for shared processor 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 shared processor partitions, and which are not.

Applications with response time requirements

The shared processor partition environment is a dynamic environment, specially when capped and uncapped partitions are running on the same system.

As stated in Chapter 2.6.2, “POWER Hypervisor Design” on page 46, the dispatch wheel assures that all virtual processors are dispatched in an interval of 10 ms. It does not guarantee, however, that the *elapsed time* between one dispatch and the next one is fixed. Virtual processors can therefore been 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 4-8 illustrates the case for the smallest capacity (10% of a physical processor), where the time slice is 1 ms.

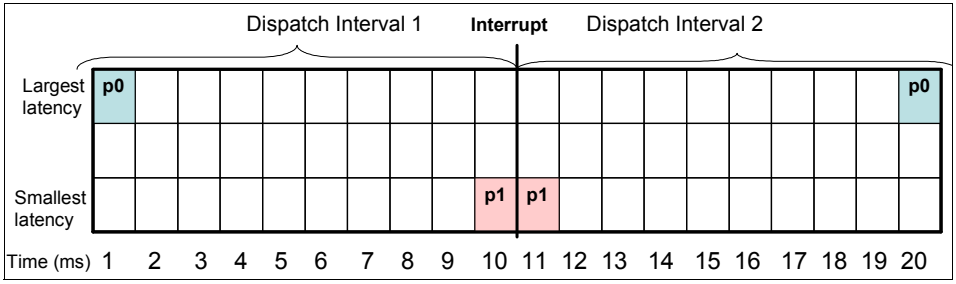


Figure 4-8 Dispatch latencies for virtual processors

Applications that require a specific response time for transactions may also not be good candidates for shared processor partitions. You can configure the processing capacity inside a partition using different ways, and different number 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 may experience performance impact when running on a partition with smaller (but more) virtual processors. Batch applications typically fall into this category, and run as fast as the individual capacity of a processor. Therefore, care must be taken when configuring a shared processor partitionto run similar workloads, in order not to impact performance.

Therefore, applications where constant response time is a requirement, or real time applications where the response time is on the same order of magnitude as the dispatch latencies may not be optimal candidates for shared partitions. 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 that present polling behavior

Applications that rely on polling to execute their processing may not be suitable for shared processor 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 shared processor partition environment you can use the spare cycles for other partitions, by having the virtual processors calling the *h_cede()* system call. 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 use of processor resources are low, but have peaks of usage during a short period of time are good candidates for a shared processor 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 number of users accessing a system tend to follow a pattern based on the user day of work. This way, it shows two distinct 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 4-9 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.

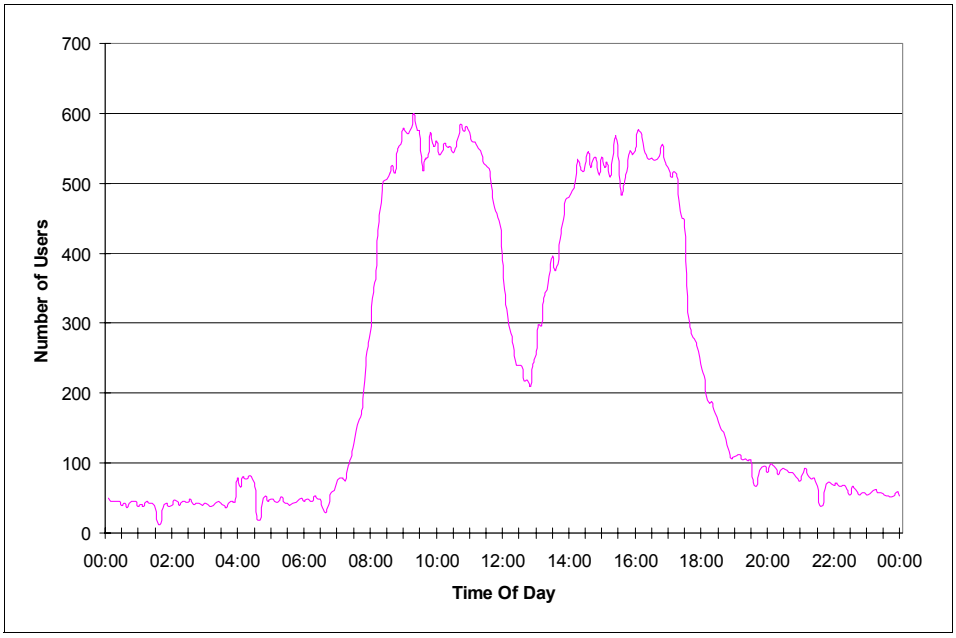


Figure 4-9 User distribution during the day in an ERP application server

For OLTP applications, the processor usage usually follows a similar distribution, as shown on Figure 4-10 for the same system.

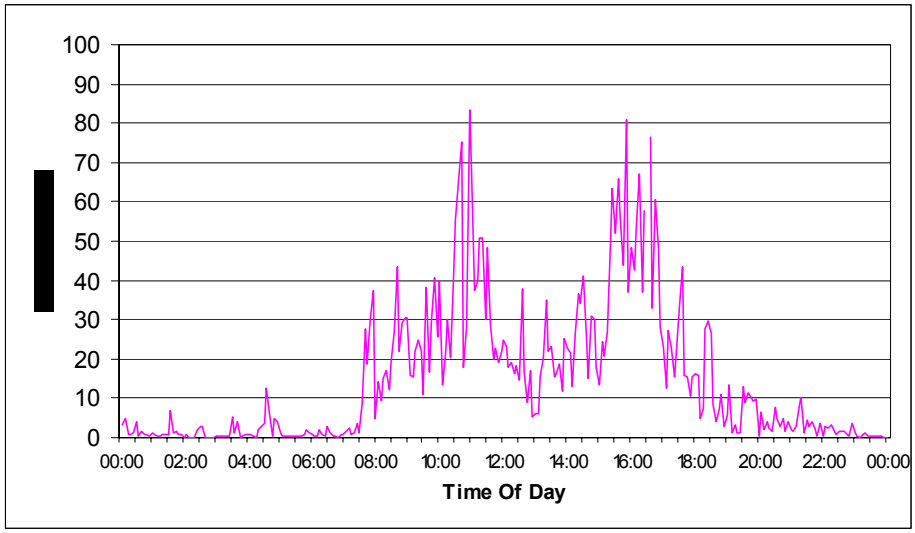


Figure 4-10 Processor utilization by the ERP application server

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 4-11 illustrates a typical scenario where different applications are running on shared processor 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.

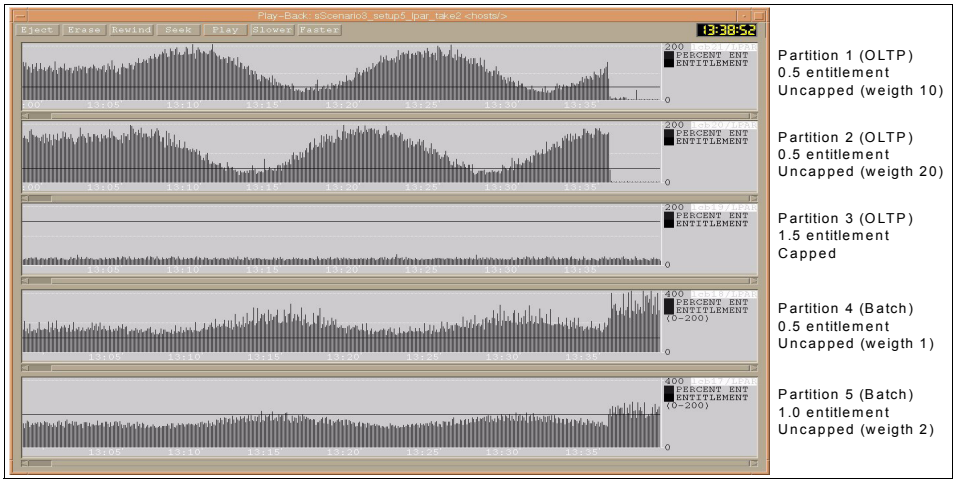


Figure 4-11 Processor utilization between five partitions with different workloads

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 the processing capacity during its execution, 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 received the processing capacity it needs, and it is less suitable of interference from other workloads running in the system.

It may be useful to run these applications in an uncapped 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.

4.1.10 Guidelines for planning shared processor partitions

When planning partitions to run applications using micro-partitioning, it is important to identify the application requirements and behavior, in order to proper 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 shared processor 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 4-1, 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 4-1 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 system, in order to accommodate the requirements in a single system with shared processor partitions, we sum the peak performance requirements for each function. We also consider an additional capacity for the micro-partitioning operation and activities. We consider 20% of the required capacity as an additional for overhead.

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. Also, the DB Server usually requires room for growth, that translates into using a server with scalability. In this example, we would have:

Table 4-2 Implementation with separate servers

Function	Capacity requirement in rPerf	Server	Capacity provided in rPerf
DB Server	4.4	p630 2-way 1.45 GHz	4.41
App Server 1	3.7	p615 2-way 1.2 GHz	4.0
App Server 2	3.7	p615 2-way 1.2 GHz	4.0
Development	2.0	p615 2-way 1.2 GHz	2.5
Test	1.5	p615 2-way 1.2 GHz	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 partitioned server, we will have more flexibility in moving extra resources among partitions, but still need to provide extra capacity than can be underutilized. Table 4-3 shows the same example using a partitioned 1.45 GHz pSeries 650 server:

Table 4-3 Implementation with a partitioned POWER4-based pSeries 650 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 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 using 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 Appendix 4-4, “Implementation with micro-partitioning” on page 121:

Table 4-4 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

Function	Capacity requirement in rPerf	Recommended capacity for micro-partitioning	Percentage of physical processor requirements
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 provisioning capacity for the peak processing of each partition. You can provision 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, on 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 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 maintained, with maximum efficiency in resource usage.

Figure 4-12 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.

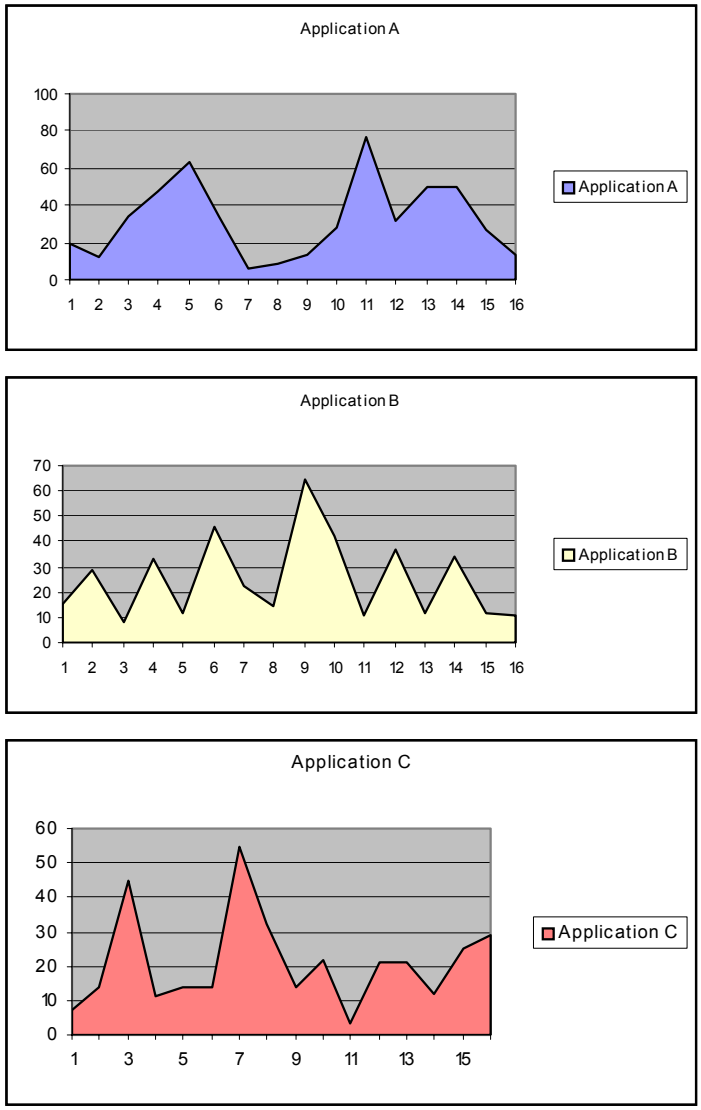


Figure 4-12 Processing resource consumption for three different applications

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 4-5 show the peak consumption for each partition, and the sum of the peaks.

Table 4-5

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 4-13 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.

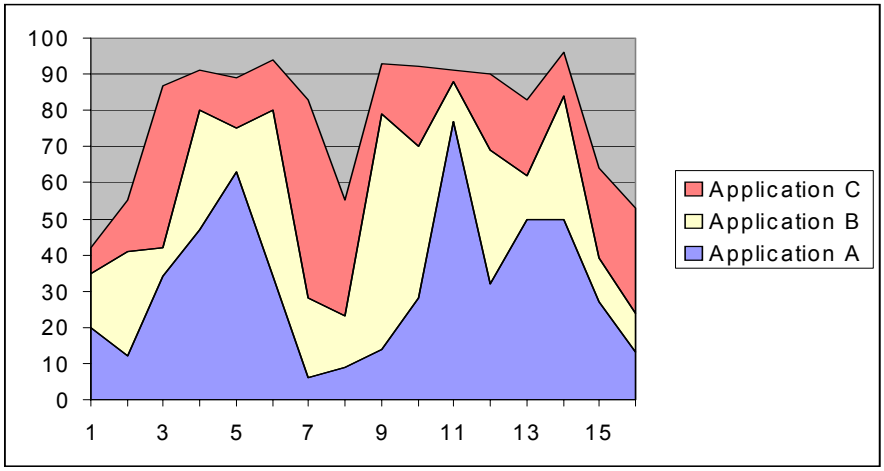


Figure 4-13 Sum of resource consumption for the three applications

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 that they can get the resources needed for peak processing. It

is also important to note that if by 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 degraded.

4.2 Virtualized Input/Output

This chapter gives after short introduction to virtualized I/O first, and after that a deeper view to that and how the POWER5 Hypervisor handles the transactions between the partitions.

4.2.1 Introduction

With the usage of virtual partitions in POWER5-based Servers, the number of possible partitions on many systems can be greater than the number of I/O slots in this systems. POWER5 processor-based servers support 254 LPARS (2004), which is more than the supported number of I/O-Slots per CEC (Central Electronic Complex). There are up to about 160 I/O Slots in 2004. Typically you need one slot per Network Interface Connector and one slot per Host Bus Adapter.

To be able to get a higher amount of devices, IBM introduced virtualized Input/Output technology for POWER5-based servers. Virtualized I/O is an optional feature and can be used on above hardware in conjunction with AIX 5L v5.3. In addition, virtual devices are more convient and it's a cost saving option if logical partitions communicate with one another via Virtual Ethernet then via dedicated adapters and network hardware.

Virtualized I/O as shown in Figure 4-14 is intended as a complement to dedicated or local I/O. It can also be referred to as physical I/O.

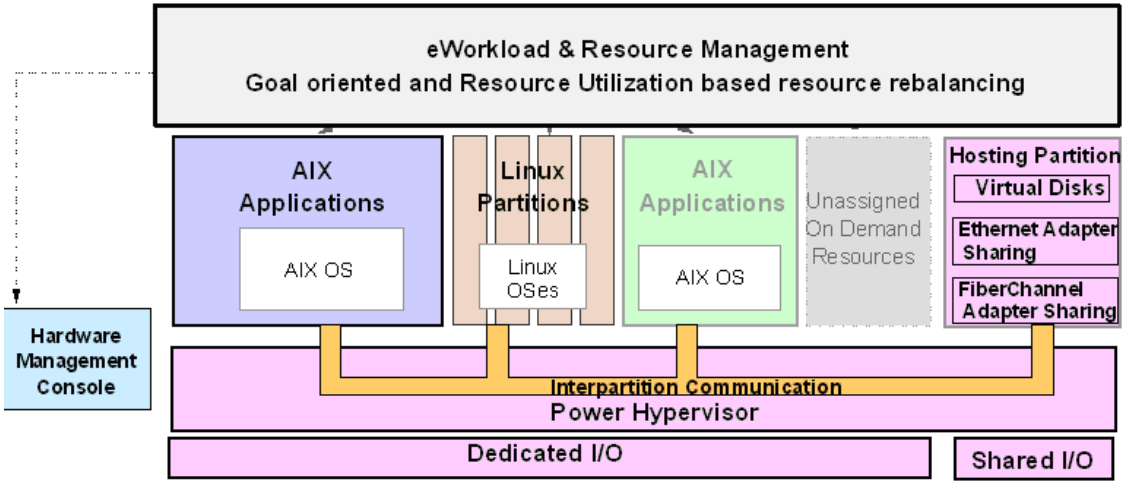


Figure 4-14 Virtual I/O provided by POWER Hypervisor

A partition can have any combination of local and virtualized I/O adapters.

Virtualized I/O includes Virtual SCSI, Virtual Ethernet and Virtual Serial. For performance issues we will concentrate on Virtual SCSI and Virtual Ethernet. It is also called *interpartition communication* and is handled by the POWER Hypervisor..

4.2.2 Virtualized I/O and the POWER Hypervisor

The following topic provides detail on how the POWER Hypervisor handles the inquiries from the virtualized I/O devices.

The POWER Hypervisor uses one of three techniques to realize virtualized I/O:

- *Hypervisor simulated class.* This class is shown in Example 4-15 and illustrates how the POWER Hypervisor may totally simulate the adapter. For example, this is used in the Virtual Ethernet support (see Chapter 4.3, “Virtual Ethernet” on page 137). This technique is applicable to communications between partitions that are created by a single hypervisor instance.

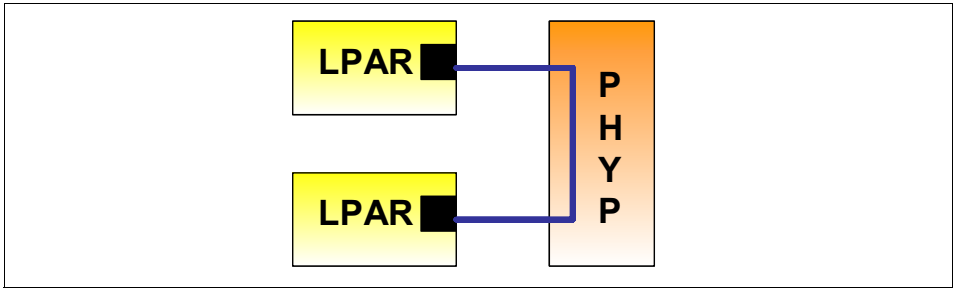


Figure 4-15 Hypervisor simulated class

- *Partition managed class*. This class is shown in Figure 4-16 on page 129 and is a class where the server partition provides the services of one of its virtual I/O adapters to a partner partition(s). A 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 partner partition’s DMA buffer areas using the Remote DMA (RDMA, see “Remote DMA” on page 134) facilities, and passing I/O responses back to the partner partition. This type of class is used by Virtual SCSI as decribed in Chapter 4.5, “Virtual SCSI” on page 169.

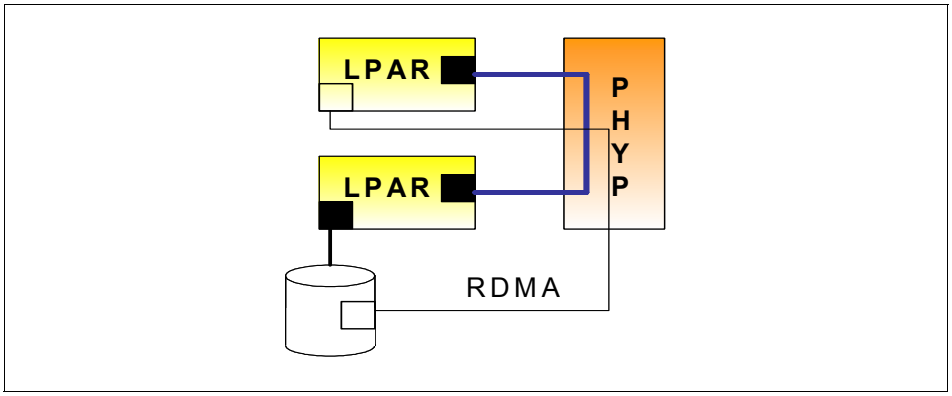


Figure 4-16 Partition managed Class

- *Hypervisor managed class*. A class where the POWER Hypervisor may provide low level hardware management (error and subchannel allocation) so that partition level code may directly manage its assigned sub-channels. This model is used with shared devices such as the InfiniBand Host Channel Adapter (HCA). This is just mentioned for the sake of completeness and will not be further handled in this chapter.

In addition to the above general classes, there are also two general characteristics that can be associated with most Virtual I/O:

- *Synchronous type virtualized I/O* is where the communicating partitions are under the control of the *same* hypervisor and within the same CEC (Central Electronic Complex). Command/Response Queue (CRQ) operations (see “Types of Connections” on page 133) and some memory to memory moves between the partitions is via synchronous hcall() operations.
- *Asynchronous type virtualized I/O* is where the communicating partitions may or may not be under the control of the same hypervisor and may or may not be within the same CEC. CRQ operations and memory to memory moves between the partitions is an *asynchronous queued operation*.

4.2.3 Virtualized I/O architectural infrastructure

Virtualized I/O is used in conjunction with the logical patitioning option. Now we will have a look at the virtualized I/O infrastructure and will explain the most common terms for virtualized I/O.

Client Virtual I/O model

This terminology is mainly used with the partition managed class of virtualized I/O. The client or client partition is an entity which generally makes requests of a server partition for access to I/O to which it does not have direct access. Unlike the server, the client does not provide services to other partitions in order to share the I/O which resides in their partition. However, it is possible to have the same partition be both a server and client partition, but under different virtual I/O adapters.

The Client Virtual I/O model is one where the client partition maps part of its local memory into an RTCE table (see “Remote Translation Control Entry (RTCE)” on page 134), so that the server partition can get access to that client’s local memory. An example of this is the VSCSI client.

Server Virtual I/O model

This terminology is mainly used with the partition managed class of virtualized I/O. The server, or server partition is an entity which provides a method of sharing the resources under its direct control with another partition, virtualizing those resources in the process. There are two versions of the Server Virtual I/O model:

- *The server is a server to a client*. An example of this is the VSCSI client. In this case, the server gets access to the client partition’s local memory via what the client mapped into an RTCE table. This access is done through the second widow pane of the server’s “ibm,my-dma-window” property, which is linked to the first window pane of the client’s “ibm,my-dma-window” property. Depending on whether asynchronous operation is required or not (as denoted by the virtual I/O Adapter type), the server uses either LRDMA or ALRDMA (see “Asynchronous Logical Remote DMA (ALRDMA) Option” and “Remote Direct Memory Access is DMA transfer from the server to its client or from the server to its partner partition. DMA refers to both physical I/O to and from memory operations and to memory to memory move) operations.” on page 134) operations to service the client.
- *The server is in a peer relationship to another server.* An example of this is the Shared Memory Cluster (SMC) virtual IOA. In this case, the Server Virtual I/O model is one where the server gets access to the other server (called the partner partition) partition’s local memory via what the partner mapped into an RTCE table (via the third pane of the “ibm,my-dma-window” property). Depending on whether asynchronous operation is required or not (as denoted

by the virtual IOA type), the server uses either ALRDMA operations for asynchronous operations, or LRDMA operations for synchronous operations, to communicate with its partner partition.

Attention: Don't mix up the Server Virtual I/O, which is a logical part of the Virtual Client/Server model and the Virtual I/O Server, which is a product, as described later.

The /vdevice Open Firmware Tree Node

The platform defines for each of its partitions the number and type of Virtual I/O Adapters with the associated interpartition communications paths (if any). These definitions take the architectural form of Virtual I/O Adapters and are communicated to the partitions as device nodes in their Open Firmware device tree.

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.

Most vital I/O adapters are represented in the Open Firmware device tree as children of the /vdevice node (child of the root node). While the vdevice sub-tree is the preferred architectural home for virtual I/O adapters, selected devices for historical reasons, are housed outside of the vdevice sub-tree.

The platform's /vdevice node must contain the properties as defined in Table 4-6.

Table 4-6 Properties of the 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

Property Name	Req?	Definition
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.
interrupt map	N/A	Property not present
interrupt-controller	Yes	The /vdevice node appears to contain an interrupt controller.
ibm,drc-indexes	for DR	For *Dynamic Reconfiguration (DR)* 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 POWER 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

4.2.4 Types of Connections

The virtualized I/O infrastructure provides several primitives that are then used to build connections between partitions for various purposes. These primitives include (for further descriptions see below):

- 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 TCE* (Translation Control Entry) table called the *RTCE (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 an virtualized I/O adapter in the server partition with its partner 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.

Reliable Command/Response Transport (RCRQ) Option

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)

TCE (Translation Control Entry) 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 have a little more information in it (as placed there by the POWER Hypervisor), so it can, among other things, allow the POWER 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 partition's software, only though hypervisor hcall()s.

Table 4-7 Comparing TCE and RTCE

TCE (Translation Control Entry)	RTCE (Remote TCE)
In POWER4 based pSeries Servers	In POWER5 based pSeries Servers
Translation Table for logical to dedicated I/O Bus Addresses	Necessary for Remote DMA
Managed by the Hypervisor	Managed by the POWER Hypervisor
Addressed by the Operating System	Never addressed directly by the Operating System. Addressed only through Hypervisor hcall()s

Window pane ("ibm,my-dma-window" property)

The RTCE tables for virtualized I/O DMA are pointed to by the "ibm,my-dma-window" property in the device tree for each virtual device. This property can have one, two, or three triples, each consisting of a Logical I/O Bus Number (LIOBN), phys which is 0, and size.

The LIOBN essentially points to a unique RTCE table (or a unique entry point into a single table).

Remote DMA

Remote Direct Memory Access is DMA transfer from the server to its client or from the server to its partner partition. DMA refers to both physical I/O to and from memory operations and to memory to memory move) operations.

The following options are available for RDMA:

Logical Remote Direct Memory Access (LRDMA) Option

The Logical Remote Direct Memory Access (LRDMA) option allows a server partition to securely target memory pages within a partner partition for virtual I/O operations. This option is defined for *synchronous* type Virtual I/O.

Asynchronous Logical Remote DMA (ALRDMA) Option

For the *asynchronous* infrastructure, the CRQ facility is implemented along with the ALRDMA option. The asynchronous nature of this infrastructure does not allow for the capability to immediately (synchronously) notify the sender of the message whether the message was delivered successfully or not, and therefore an asynchronous queueing structure is used to deliver entries to the CRQ.

Redirected RDMA

RDMA refers to when the TCE(s) for a dedicated I/O Adapter are set up through the use of the RTCE table manipulation hcall()s such that the client or partner's partition's RTCE table is used by the hypervisor during the processing of the hcall() to setup the TCE(s) for the physical IOA, and then the physical IOA DMAs directly to or from the client or partner partition's memory.

Hypervisor calls hcall()

The calls, used by the POWER Hypervisor, are described in Chapter 2.6.2, "POWER Hypervisor Design" on page 46

4.2.5 Shared Logical Resources

Owners of resources can grant, to one or more client partitions, access to any of its resources. A client partition being defined as a partition with which the resource owner is authorized to register a CRQ, as denoted via an Open Firmware device tree virtual I/O node. Granting access is accomplished by requesting that the hypervisor generate a specific cookie for that resource for a specific sharing partition. The cookie value thus generated is unique only within the context of the partition being granted the resource and is unusable for gaining access to the resource by any other partition.

This unique cookie is then communicated via some inter partition communications channel, most likely the authorized Command Response Queue. The partner partition then accepts the logical resource (mapping it into the accepting partition's logical address space). The owning partition may grant shared access of the same logical resource to several clients (by generating separate cookies for each client). During the time the resource is shared, both the owner and the sharer(s) have access to the logical resource, the software running in these partitions use private protocols to synchronize control access. Once the resource has been accepted into the client's logical address space, the

resource can be used by the client in any way it wishes, including granting it to one of its own clients. When the client no longer needs access to the shared logical resource, it destroys any virtual mappings it may have created for the logical resource and returns the logical resource thus unmapping it from its logical address space. The client program could, subsequently accept the logical resource again (given that the cookie is still valid).

To complete the termination of sharing, the owner partition rescinds the cookie describing the shared resource. Normally a rescind operation succeeds only if the client has returned the resource, however, the owner can force the rescind in cases where it suspects that the client is incapable of gracefully returning the resource.

4.2.6 The Virtual I/O-Server

The IBM Virtual I/O-Server is the link between the virtual and the real world. It is an AIX-based appliance and is supported on POWER5-based servers only.

Main purpose is to provide two functions:

- Server part for the Virtual SCSI devices (VSCI target)
- Functionality of shared Ethernet adapter for Virtual Ethernet.

The Virtual I/O Server will be shipped as a mksysb image. As described before, it is AIX5L V5.3 based and not accessible as a standard partition. Administrative access to the I/O-Server partition is only possible as user **padmin** and *not* as **root**. After login, user pdamin 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. Additionally, this product supports Linux and AIX 5L V5.3, but no specific OS skill is required for administration the I/O-Server.

The performance considerations of the Virtual I/O-Server will be addressed in later topics. For example, issues related to performance with the I/O-Server as a Virtual SCSI target device see Chapter 4.5, “Virtual SCSI” on page 169.

A performance measurement of Virtual I/O using the *Shared Ethernet Adapter* (SEA) functionality will be discussed in “Shared Ethernet Adapter functionality” on page 160

4.3 Virtual Ethernet

This chapter handles a short description and performance issues that come along with the usage of the 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.

4.3.1 Introduction

Virtual Ethernet creates logical ethernet connections between one or more partitions. There are no physical Adapter needed for implementing a Virtual Ethernet. The POWER5 Hypervisor copies frames from and to the virtual interfaces between the (maybe virtual) LPARs. It acts as a virtual 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 160.

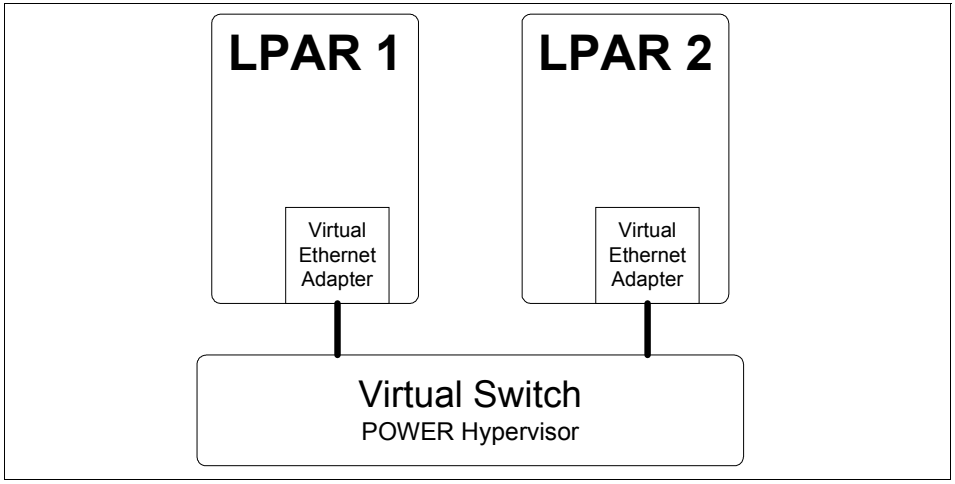


Figure 4-17 Virtual Ethernet

A virtual ethernet adapter can be configured like a standard ethernet adapter.

In every partition virtual and dedicated network devices can be used for communication as shown in Figure 4-18. Up to 256 adapters (sum of virtual and local) are supported per LPAR.

```
# lsdev -Cc adapter
ent0   Available       Virtual I/O Ethernet Adapter (l-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 General Concepts
```

Figure 4-18 Virtual and local adapters on one partition

IEEE Support

The Virtual Ethernet, shipped with AIX 5L V5.3 supports the IEEE (The Institute of Electrical and Electronics Engineers, Inc.) 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. Up to 18 VIDs can be configured per Virtual LAN port. The following topic describes how the POWER5 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) Adress of physical Ethernet is coded on the (hardware) adapter, the MAC-Address of the virtual adapter is generated by the HMC.

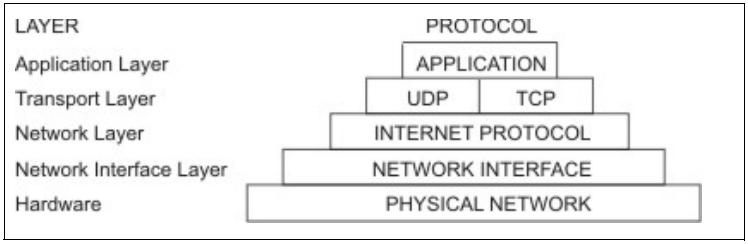


Figure 4-19 TCP/IP Suite of protocols

Figure 4-19 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 *Ethernet adapter*. In Virtual Ethernet, the *POWER Hypervisor* is responsible for that.

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.

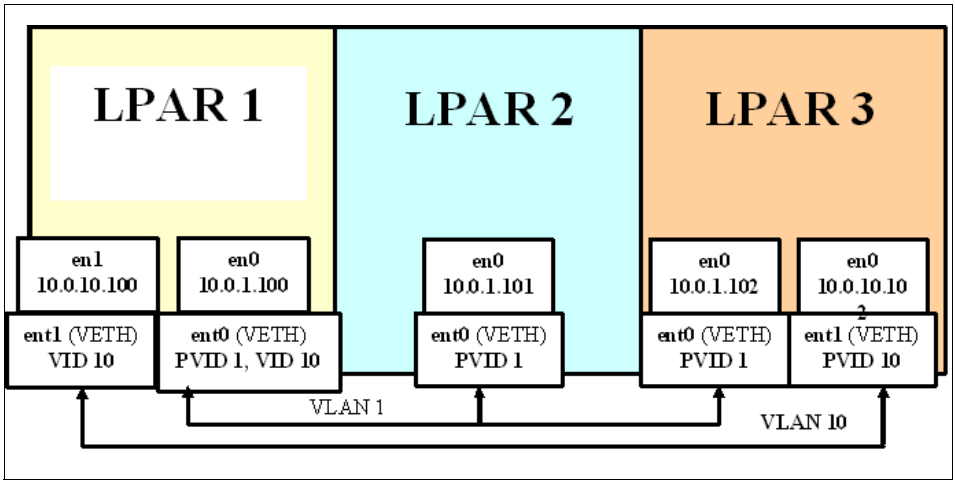


Figure 4-20 Example of two VLANs in a Virtual Ethernet Environment

4.3.2 Virtual Switch of the POWER5 Hypervisor

The POWER 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 POWER Hypervisor caches the message's source MAC address (2nd 6 bytes) to use as a filter for future messages to the Adapter. Then the POWER Hypervisor 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, then 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, then 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

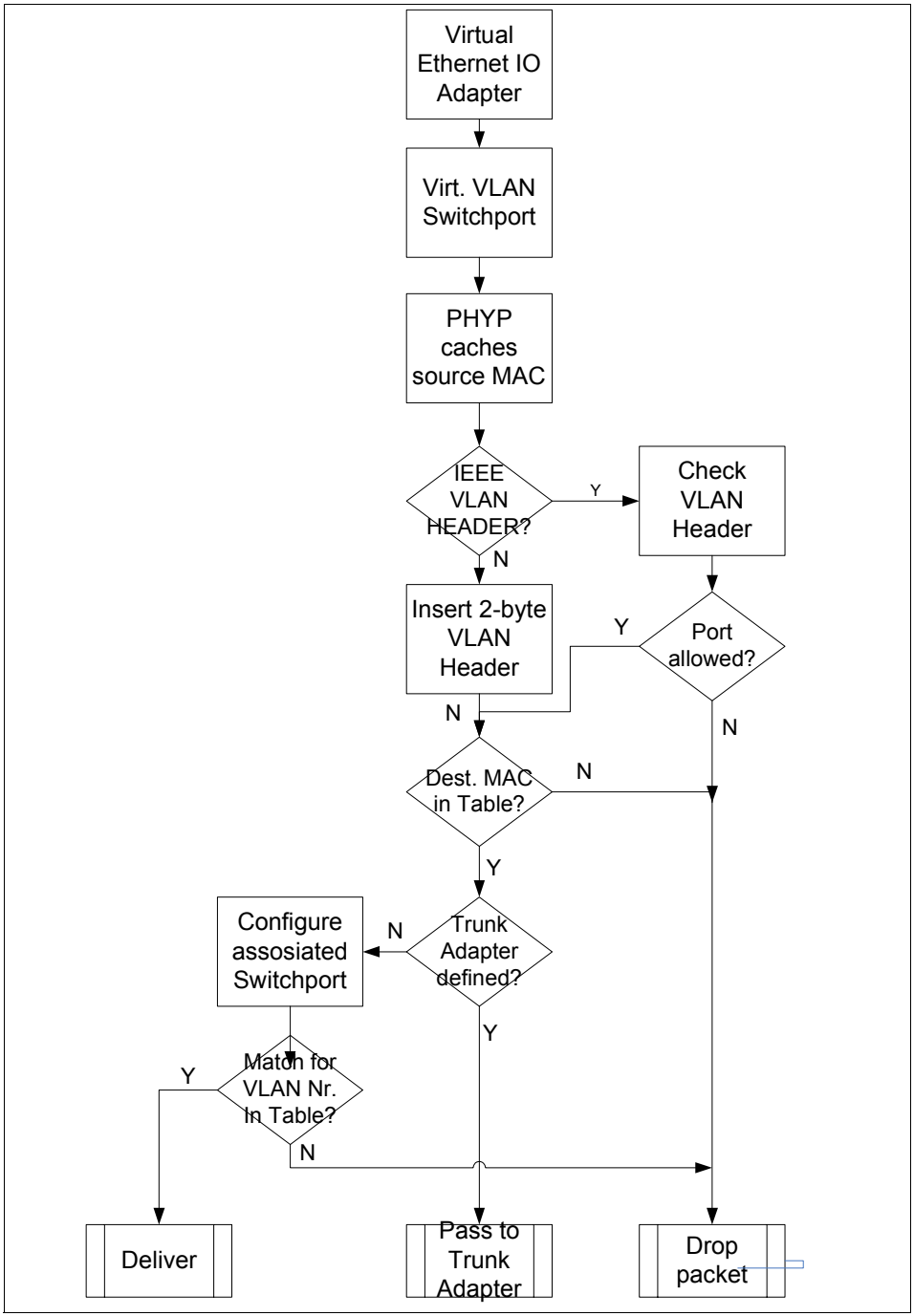


Figure 4-21 Flow chart of Virtual Ethernet

The Logical LAN adapters device tree entry includes Unit Address, and “ibm,my-dma-window” properties. The “ibm,my-dma-window” property contains a LIOBN field that represents the RTCE table used by the Logical adapter. The Logical LAN hcall()s use the Unit Address field to imply the LIOBN and, therefore, the RTCE table to reference. When the logical IOA is opened, the device driver registers, with the hypervisor, as the “Buffer List”, a TCE mapped page of partition I/O mapped memory that contains the receive buffer descriptors. These receive buffers are mapped via a TCE mechanism from partition memory into contiguous I/O DMA space. The first descriptor in the buffer list page is that of the receive queue buffer. The rest of the descriptors are for a number of buffer pools organized by increasing size of receive buffer. The format of the descriptor is a 1 byte control field, 3 byte buffer length, followed by a 4 byte I/O address. The number of buffer pools is determined by the device driver (up to an architected maximum of 254), the control field in all unused descriptors is 0h00. The last 8 bytes is reserved for statistics.

When a new message is received by the logical adapter, the list of buffer pools is scanned starting from the second descriptor in the buffer list looking for the first available buffer that is equal to or greater than the received message. That buffer is removed from the pool, filled with the incoming message, and an entry is placed on the receive queue noting the buffer status, message length, starting data offset, and the buffer correlator.

The sender of a logical LAN message uses an hcall() that takes as parameters the Unit Address and a list of up to 6 buffer descriptors (length, starting I/O address pairs). The sending hcall(), after verifying the sender owns the Unit Address, correlates the Unit Address with its associated Logical LAN Switch port and copies the message from the send buffer(s) into a receive buffer, as described above, for each target logical LAN IOA that is a member of the specified VLAN. If a given logical IOA does not have a suitable receive buffer, the message is dropped for that logical IOA (a return code indicates that one or more destinations did not receive a message allowing for a reliable datagram service).

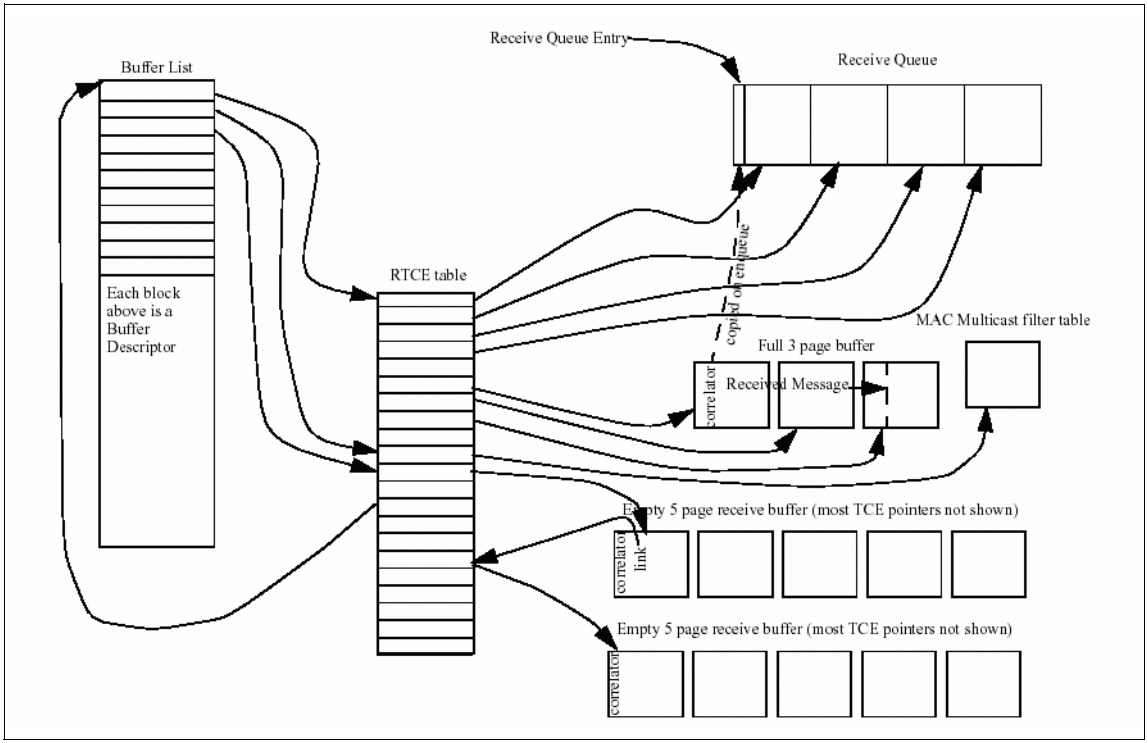


Figure 4-22 Virtual LAN IO-Adress Structures

4.3.3 Performance Considerations and Measurements

General comments to the measurements

The Operating System running on all partitions is AIX 5L v5.3. The code of AIX 5L v5.3 was not yet announced at the time this book was written (pre GA code). Because this also includes the drivers of the Gigabit Ethernet adapter and the Virtual Lan adapter, the results of the measurements may vary if they will be repeated at a later time.

The used hardware for all the test is, as not other mentioned, a 4 way POWER5 based server. This hardware, including the firmware is also not yet announced, so the performance may also vary with a later version.

But because the results are in the range as expected from the developers, the numbers in this chapter give quite a good feeling, what performance can be expected in a Virtual LAN environment.

If not other mentioned, SMT (Simultaneous Multi Threading) is turned on on POWER5 systems.

The settings of the Virtual LAN adapters and the gigabit ethernet adapter are at default, which is specially for the gigabit adapter:

- tcp_segmentation_offload=enabled (also known as large_send)
- tcp_sendspace=131072
- tcp_receivespace=131072
- rcf1323=1, for all MTUs except 1500
- checksum offload =enabled
- interrupt coalesing =enabled

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, UDP, or 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.

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.

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 one session test, how the response time and latency is growing with more load.

Overview of the following benchmark measurements

First, there will be a measurements, that shows, how throughput is growing by adding more *entitlements to a virtual CPU*.

Next benchmark test is comparing parameters as CPU consumption, transaction rate and latency from *dedicated* ethernet (gigabit adapter) to *Virtual Ethernet*.

The last series of measurements is about changing performance with *ST and SMT mode* of the POWER5 processor. This is again done with a gigabit ethernet adapter and Virtual Ethernet.

Virtual LAN throughput at different CPU Entitlements

This measurement shows what throughput can be expected in a Virtual LAN. Because of the throughput varies on CPU entitlements and MTU size, these parameters are variable in the following measurement.

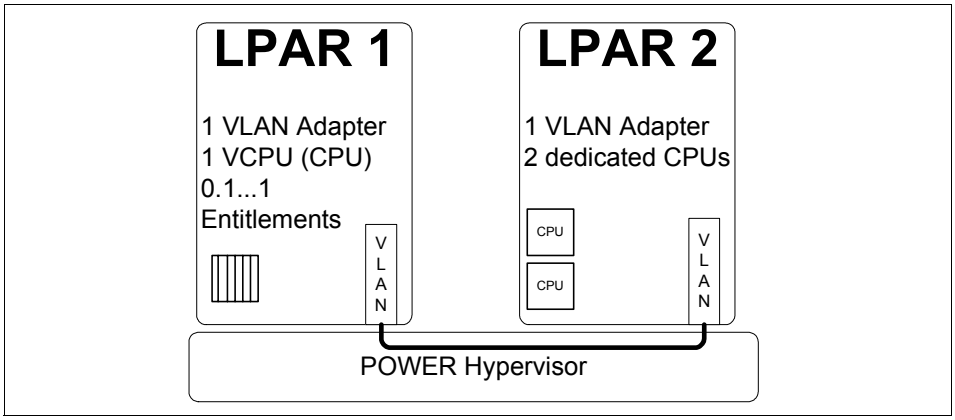


Figure 4-23 Throughput at variable CPU entitlements and MTU sizes

Both LPARS have one Virtual LAN adapter and there are multiple sessions running between both adapters. The benchmark used for this is *netperf TCP_STREAM* as described before.

LPAR 1 (with varied cpu entitlements) is sending a simplex stream, LPAR 2 (2 way dedicated) receives it.

Because LPAR 2 is a dedicated two-way partition, there is no bottleneck on the receiving side and the troughput of the Virtual LAN interface of LPAR 1 can be determined.

The following figures show the summary of the measured datas.

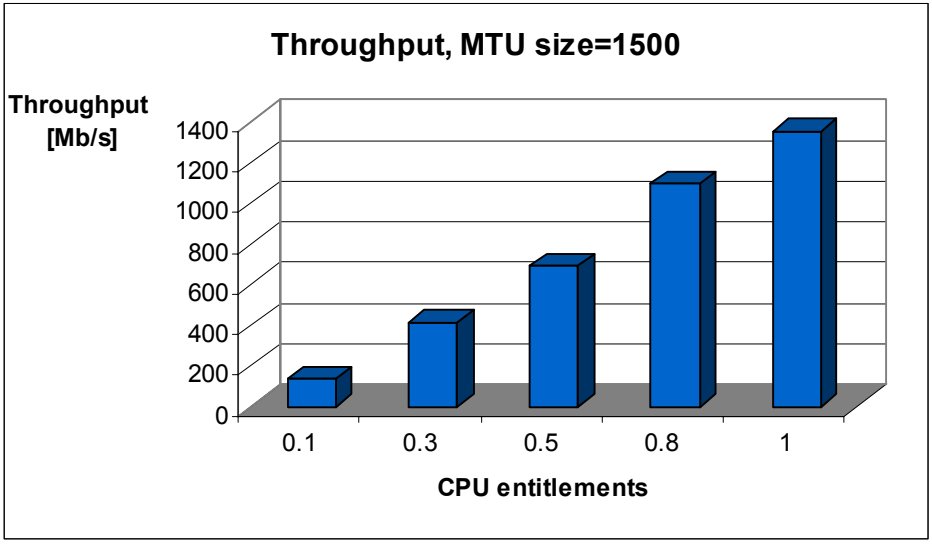


Figure 4-24 Throughput with diff. CPU entitlements, MTU size=1500

Figure 4-24 to Figure 4-26 show the measured throughput at different CPU entitlements and MTU sizes.

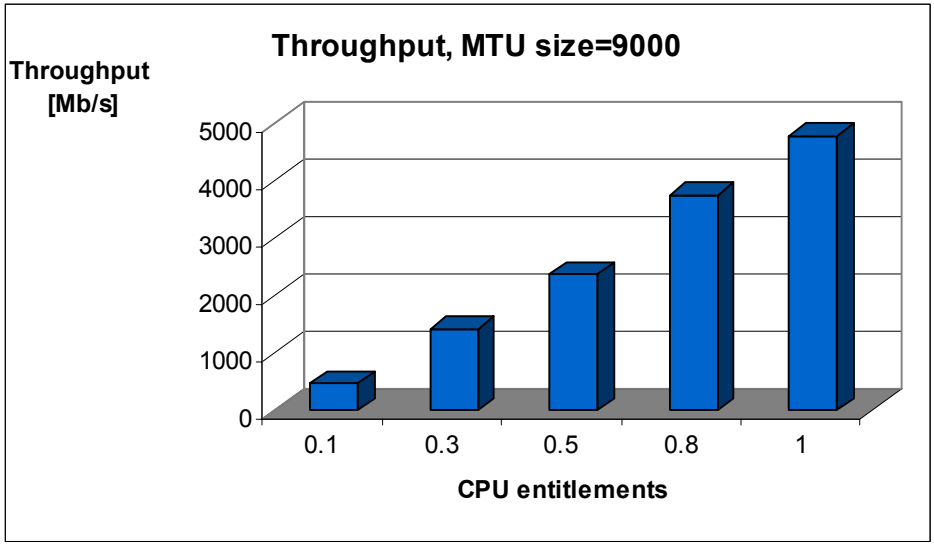


Figure 4-25 Throughput with diff. CPU entitlements, MTU size=9000

There is one chart for every MTU size: 1500, 9000 and 65394.

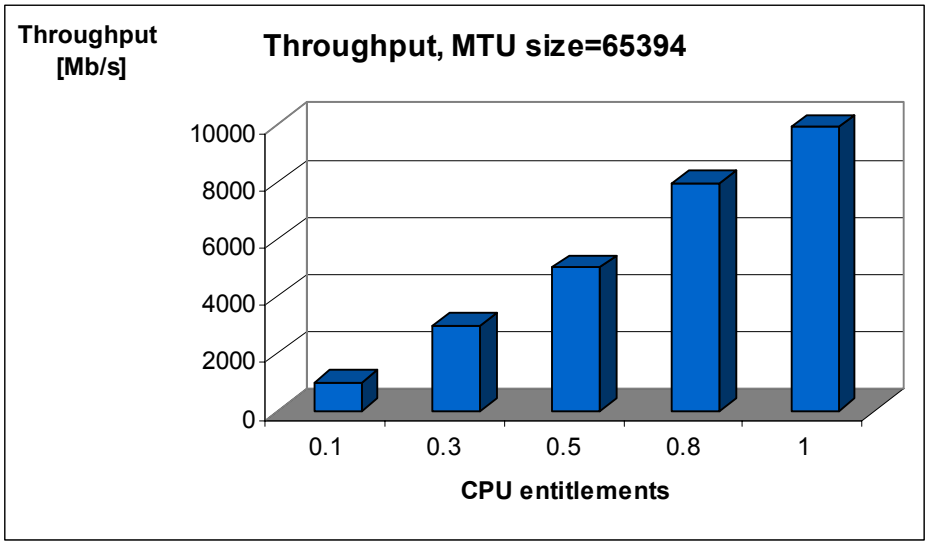


Figure 4-26 Throughput with diff. CPU entitlements, MTU size=65394

Findings of the measurements Virtual Ethernet performance

- The throughput of the Virtual Ethernet scales nearly linear with the allocated cpu entitlements. Because the interval of the CPU entitlements is not equal in the charts above, Figure 4-27 on page 149 shows the linearity of the throughput. For better comparison, all above measured data are *normalized to 0.1 CPU* entitlement: ([Throughput x 0.1]/Entitlements)
- The linear scaling of Virtual Ethernet with CPU entitlements shows, that there is no measurable overhead when using shared processors versus dedicated processors for the throughput between Virtual LANs.
- Throughput is increasing, as expected, with growing MTU-Sizes. From MTU-Size 1500 to 9000 with factor ca. >3 and from 1500 to 65394 with factor >7.

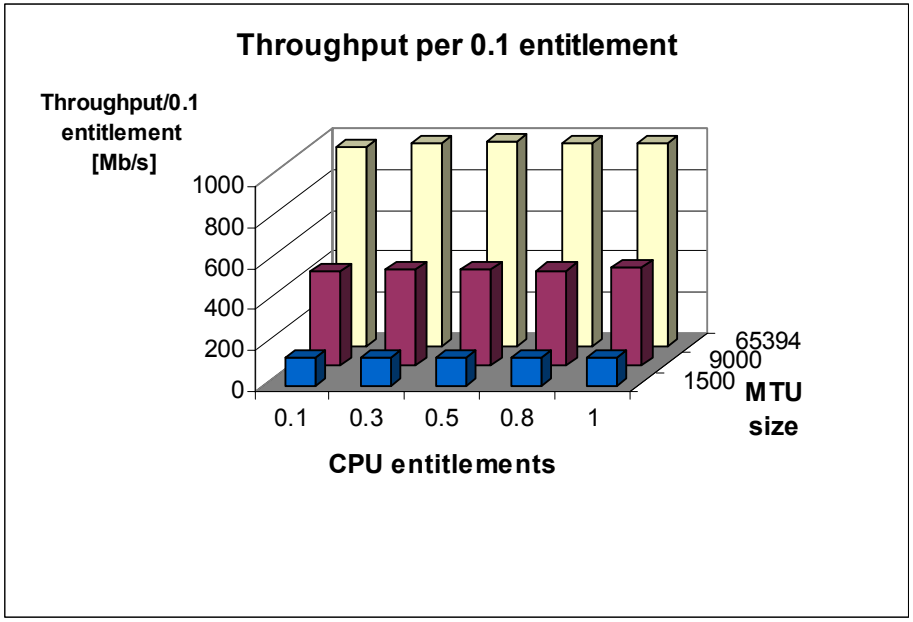


Figure 4-27 Throughput moniced to 0.1 Entitlement

Virtual LAN versus Gigabit Ethernet

With the next benchmark test we'll check performance of Virtual LAN versus Gigabit Ethernet (GbEN). For that, the following setups are chosen:

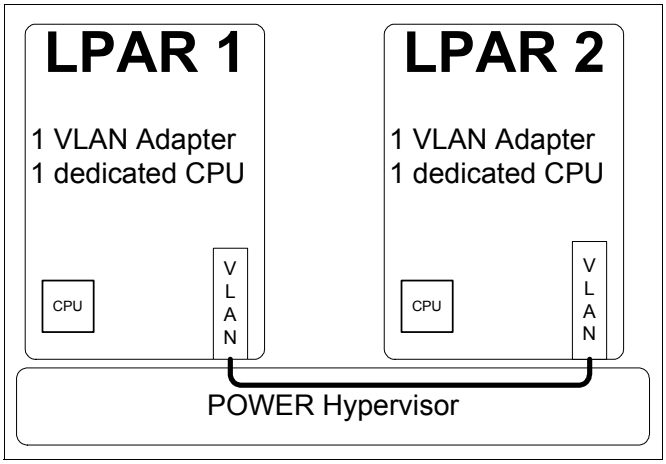


Figure 4-28 Setup VLAN to VLAN performance, 1 dedicated CPU per LPAR

Both LPARS have one dedicated POWER5 CPU assigned.

This figures show the two different types of connection between the LPARs: Virtual Ethernet via POWER Hypervisor and Gigabit ethernet via a Gigabit Ethernet Switch.

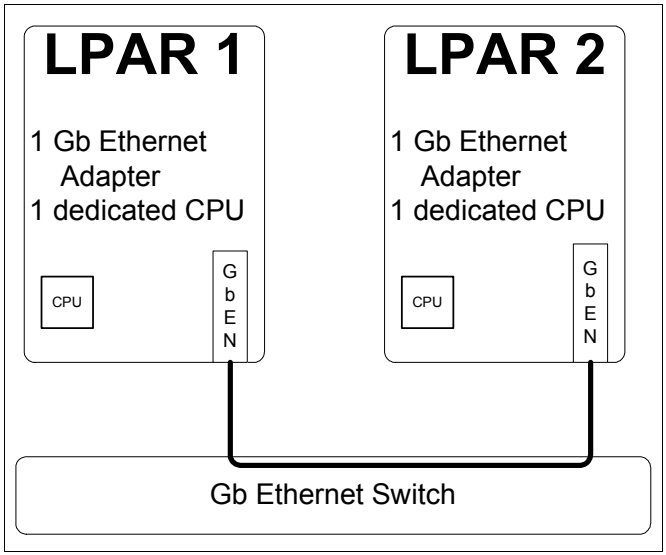


Figure 4-29 Setup GB Ethernet to GB Ethernet, 1 dedicated CPU per LPAR

The following measurements have been done:

The benchmark TCP_STREAM was running in simplex and duplex mode at different MTU sizes on both setups. During that, throughput and CPU consumption was determined.

Results of the measurements

The following chart shows the results of the measurements. Because the Gigabit ethernet adapter doesn't support MTU size of 65394, there is only data for Virtual ethernet for that.

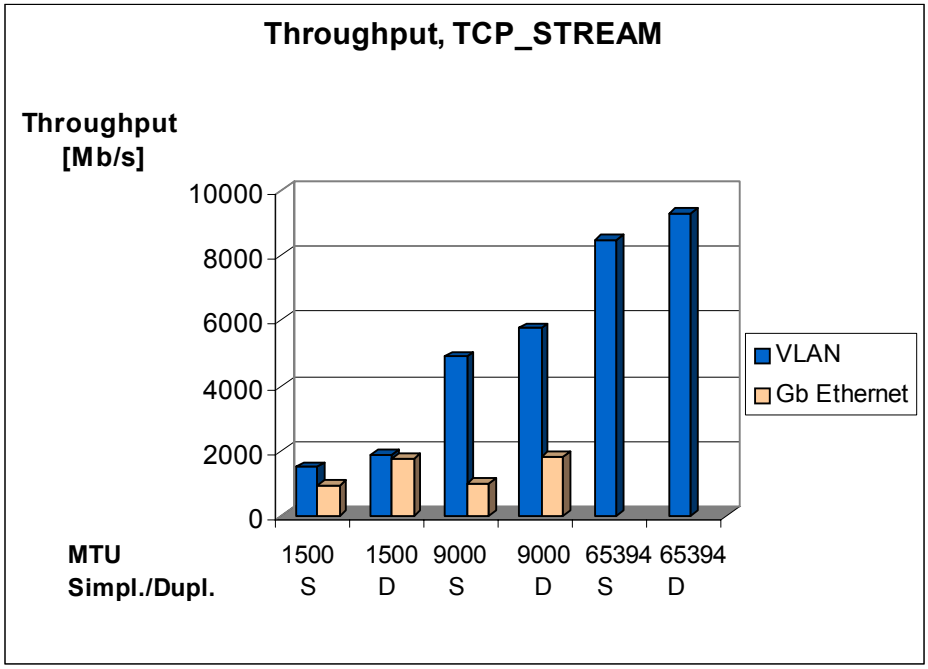


Figure 4-30 Throughput of Virtual LAN and gigabit ethernet with TCP_STREAM

Findings of: Throughput Virtual LAN and gigabit ethernet

- The Virtual Ethernet adapter has higher raw throughput at all MTU sizes.
- On MTU 9000, the difference in throughput between is very large due to the fact, that the in-memory copy, that Virtual Ethernet uses to transfer data is more efficient at larger MTU.

Comparing CPU consumption of VLAN versus GB Ethernet

For this measurements, the same setups as shown in Figure 4-28 and Figure 4-29 on page 151 are used. The used workload is TCP_STREAM again. Now, the CPU consumption is recorded at different MTU sizes. This test is also running in simplex and duplex mode.

Results of: Comparing CPU 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 CPU consumption, this is *normalized to 1 Gb throughput* for both, Virtual LAN and Gigabit ethernet.

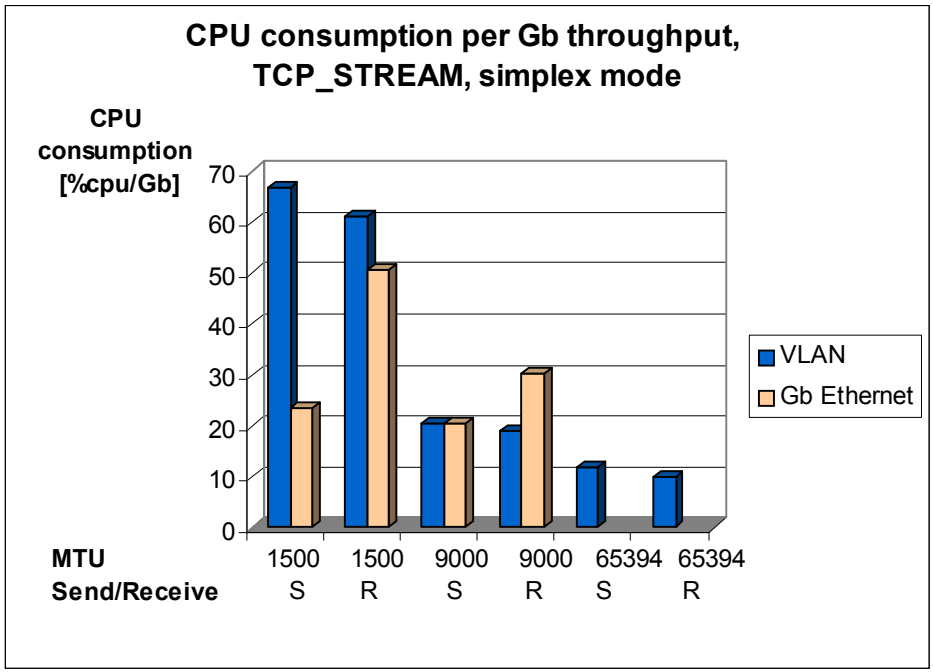


Figure 4-31 CPU consumption with TCP_STREAM, simplex mode

The results are split to two charts. One is simplex and one duplex mode.

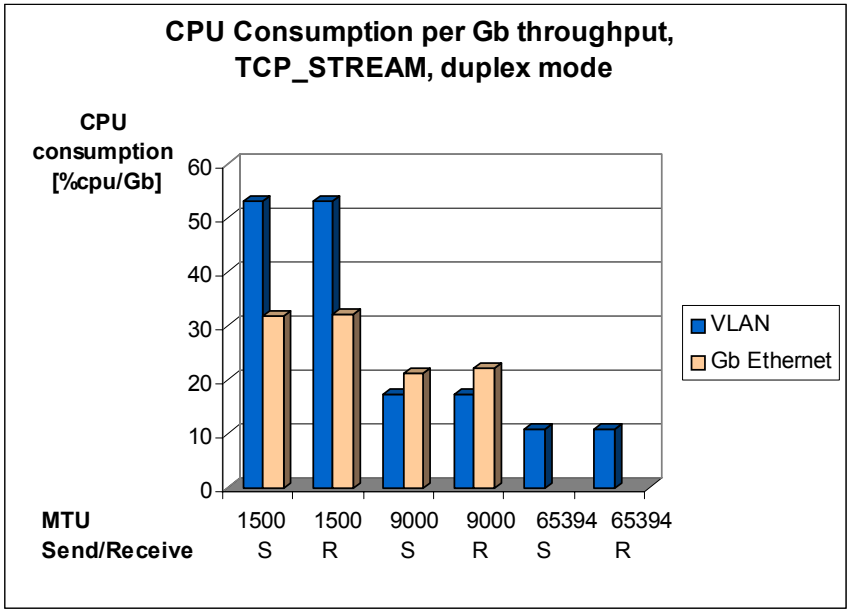


Figure 4-32 CPU consumption with TCP_STREAM, duplex mode

Findings of comparing CPU consumption

- The difference in CPU consumption between the Virtual Ethernet and the Gigabit Ethernet adapter when using MTU 1500 is the effect of having the attributes large_send and checksum_offload enabled on the Gigabit adapter.
- Additionally, the Virtual Ethernet device driver must recalculate the header checksum of the outgoing IP-packets and handle TCP segmentation. This affects CPU consumption more on smaller MTU sizes.

Comparing transaction rate and latency

For this measurements, again the same setups as shown in Figure 4-28 and Figure 4-29 on page 151 are used. 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. They show the transaction rate and latency for MTU size of 1500 and 9000 and for 1 and 20 sessions.

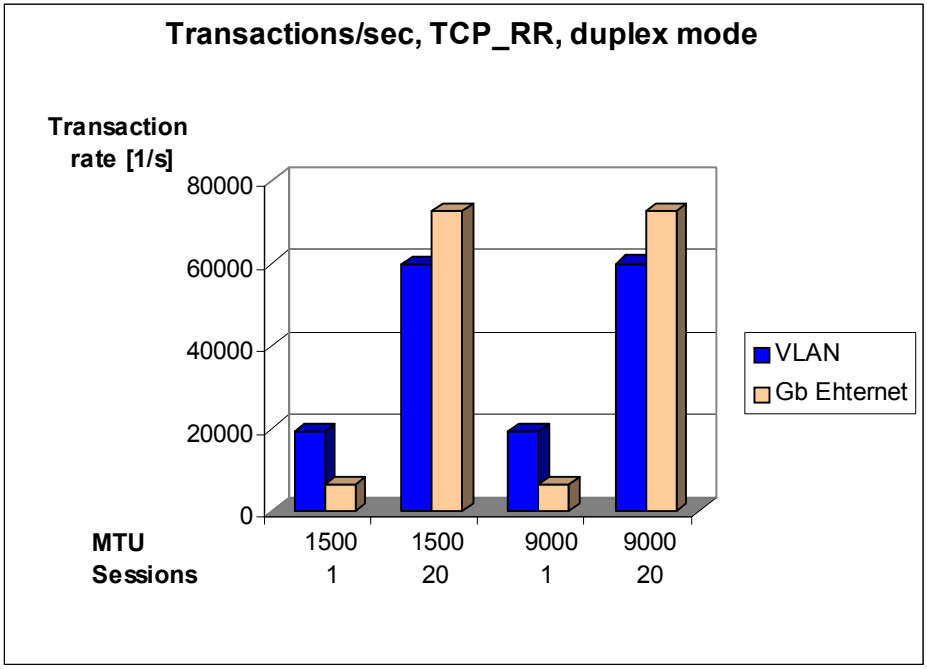


Figure 4-33 Transaction rate at different MTU sizes and 1/20 sessions

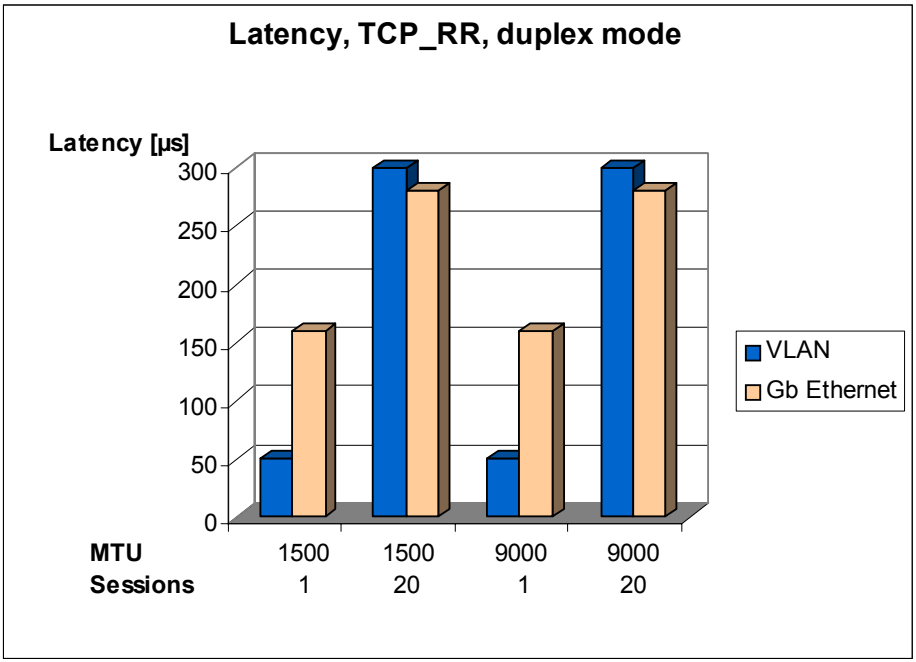


Figure 4-34 Latency at different MTU sizes and 1/20 sessions

Findings of transactions and latency

- The Virtual Ethernet adapter has lower latency for light workloads than the Gigabit ethernet adapter.
- The gigabit adapter has lower latency in heavy workloads due the coalesing interrupt feature on the adapter.

Performance of Virtual Ethernet in ST and SMT mode

A new feature of the POWER5 processor is Simultaneous Multi-Threading (SMT) which presents the kernel with two logical CPUs per virtual or dedicated CPU as described in XXXXXXXX. This measurement shows the performance gain of SMT for Virtual Ethernet. The setups as shown in Figure 4-28 on page 150 can be used again. For this comparison, both workloads TCP_STREAM and TCP_RR are used.

Comparison of Virtual Ethernet performance with ST and SMT

Because the absolute SMT data was shown before (see Figure 4-30 on page 152), the following charts show the *gain of throughput in percent,* comparing SMT to Single Threaded (ST) mode.

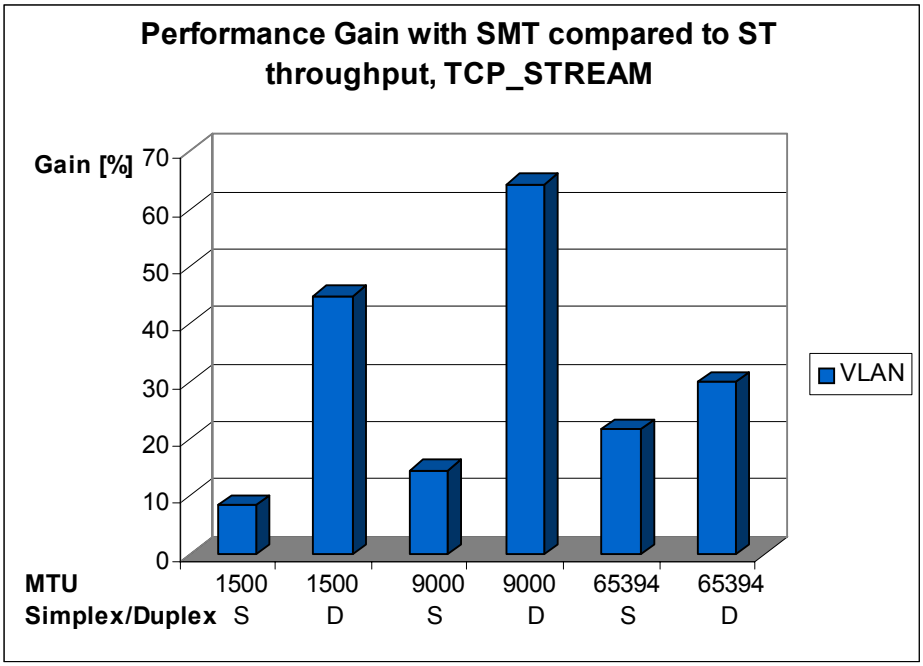


Figure 4-35 Performance gain with SMT, TCP_STREAM

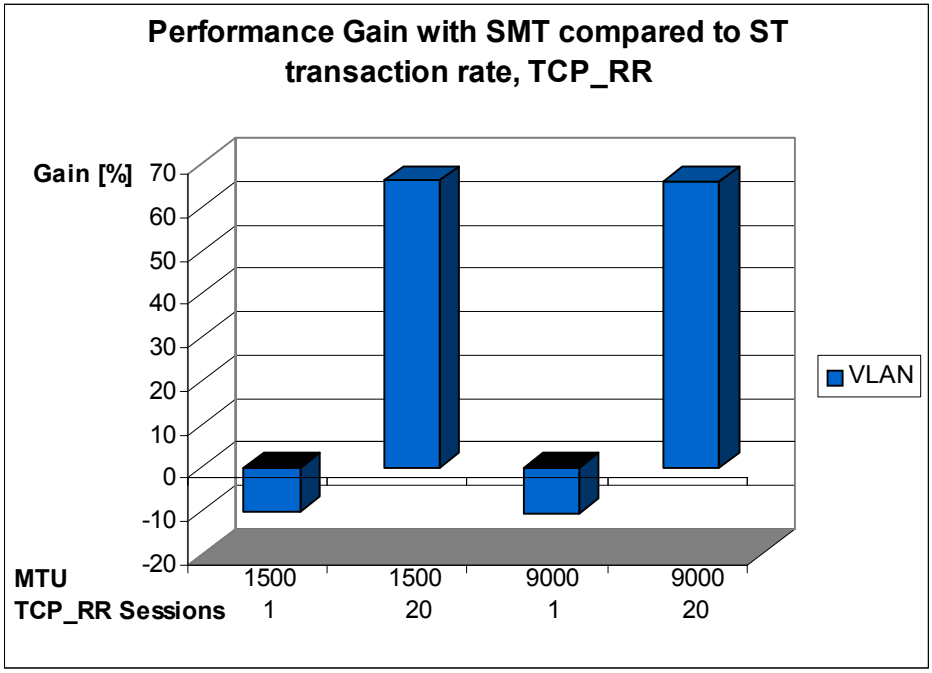


Figure 4-36 Performance gain with SMT, TCP_RR

Findings of comparing performance of ST and SMT mode

- The Virtual Ethernet adapter benefits from SMT because it is not limited by media speed and takes advantage of the extra available CPU cycles.
- The performance reduction of SMT with one session TCP_RR is because the thread on the logical CPUs go sleep and have to be wake up at every transaction.

4.3.4 Virtual Ethernet implementation guidelines

Because there was only a small amount of information available at the time for the development of this book, we can only present some rules of thumb for designing Virtual LANs.

Important: The following recommendations are no guarantee for good performance.

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 the MTU size 65394 if you expect a large amount of data to be copied inside your Virtual LAN
4. Enable tcp_pmtu_discover and udp_pmtu_discover in conjunction with MTU size 65394, if there is a communication to physical adapters.
5. Do not turn off SMT (Simultaneous Multi-Threading) unless your applications demand it.
6. The throughput in Virtual LANs scale linear with CPU entitlements, so there is no need for dedicated CPUs for partitions because of Virtual LAN performance

4.4 Shared Ethernet Adapter functionality

4.4.1 Introduction

For implementing a virtual LAN, no Virtual I/O-Server is necessarily needed. Virtual ethernet adapters can communicate among themselves via the POWER5 Hypervisor without the functionality of the Virtual I/O-Server. The Virtual I/O-Server is needed, if virtual Adapters should communicate with a physical LAN. It can logically connect one ore 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.

Implementation of the bridge

The following example in Figure 4-37 should shows, how the bridge functionality is implemented in the Virtual I/O-Server.

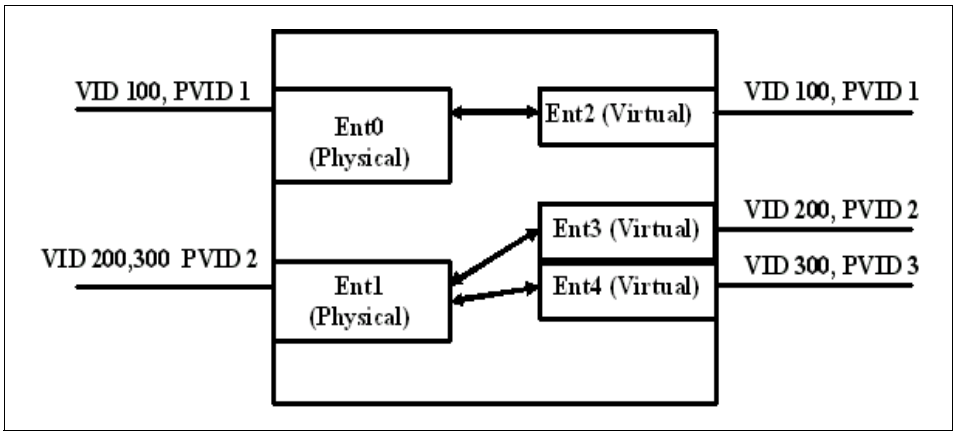


Figure 4-37 Example of a Virtual I/O-Server configuration for SEA

This I/O-Server bridges the virtual LAN with VID 100 to the Ent0 Adapter. The Virtual LANs 200 and 300 are bridged to Ent1 Adapter. So the physical adapters are *shared* by the Virtual LANs.

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, see Figure 4-38 on page 161), and is independent of any higher layer protocol.

The bridge is said to be transparent to IP. 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.

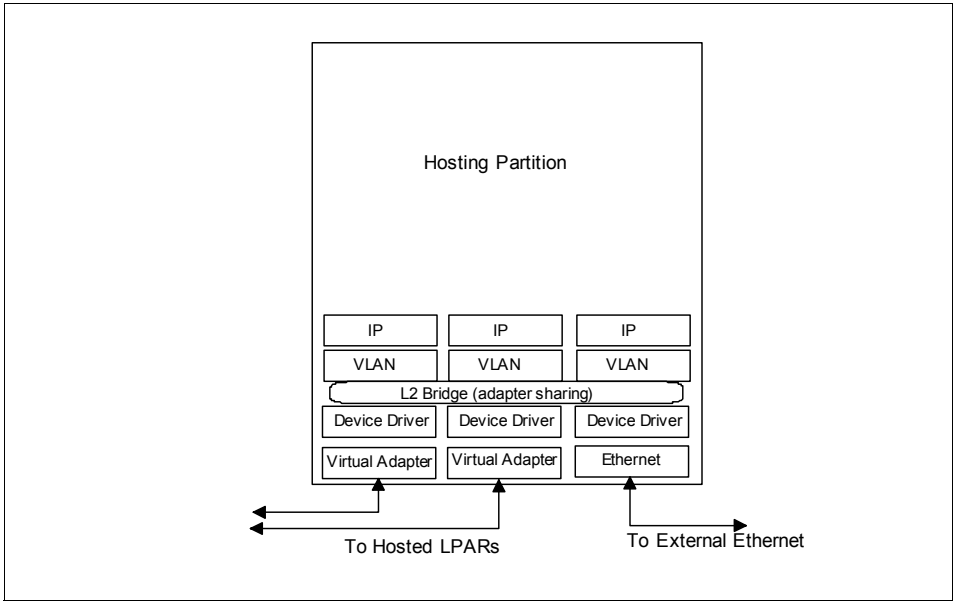


Figure 4-38 Sharing a (physical) Ethernet adapter on OSI-Layers

The I/O-Server offers broadcast and multicast support. ARP (Adress Resolution Protocol) and NDP (Neighbor Discovery Protocol) are also working across a shared ethernet adapter.

4.4.2 Performance measurements

This topic shows some measurements that were obtained with the Virtual I/O Server using the Shared Ethernet Adapter (SEA).

General comments about the measurements

The operating system running on all partitions is AIX 5L v5.3. Since this was a pre-GA release of the operating system, the results of the measurements may vary if repeated at a later time.

The system used for the test was a 4-way POWER5 based server.

Throughput and CPU utilization

Here is the description of the chosen measurement setup:

A single LPAR with a single dedicated CPU was connected through a Virtual LAN adapter to the POWER Hypervisor and to the Virtual LAN adapter of the Virtual I/O-Server. The Virtual I/O server “bridges” the Virtual Ethernet adapter to a Gigabit Ethernet adapter that is connected via a gigabit ethernet switch to a two-way Power4+ based server as shown in Figure 4-39.

The Virtual I/O-Server runs on a LPAR with a dedicated 1.65 GHz CPU. With a higher clock speed, the throughput of the I/O-Server will grow.

First, the workload TCP_STREAM as described in “Description of the performance tests and tools” on page 144 is used to examine the throughput.

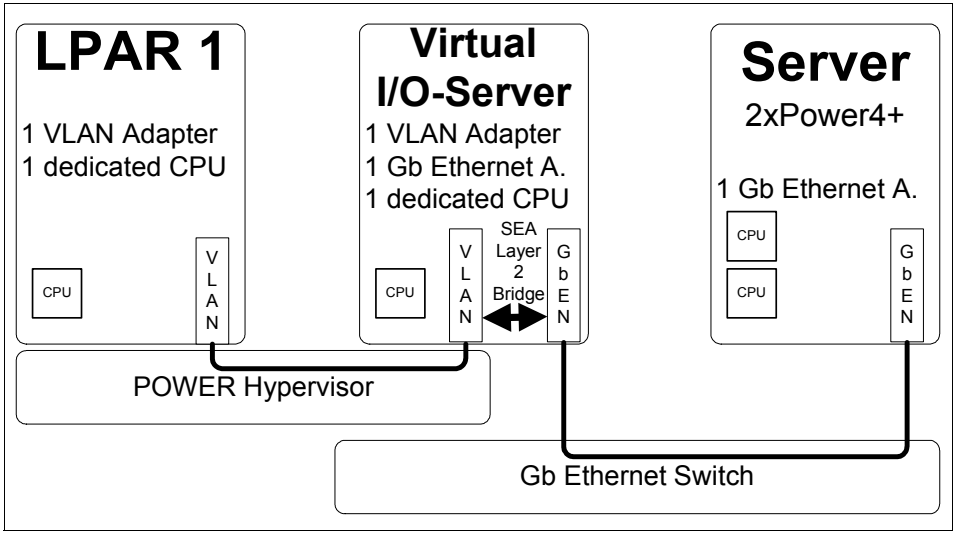


Figure 4-39 Setup for I/O-Server performance measurements

Note: The measurements are not done with a gigabit ethernet switch as shown on the figure above. A physical point to point connection was used instead. So there is no falsification of the measurement because of the internal values of the switch.

Performance results of the Virtual I/O-Server

The following charts show the results of the tests measured on the Virtual I/O-Server.

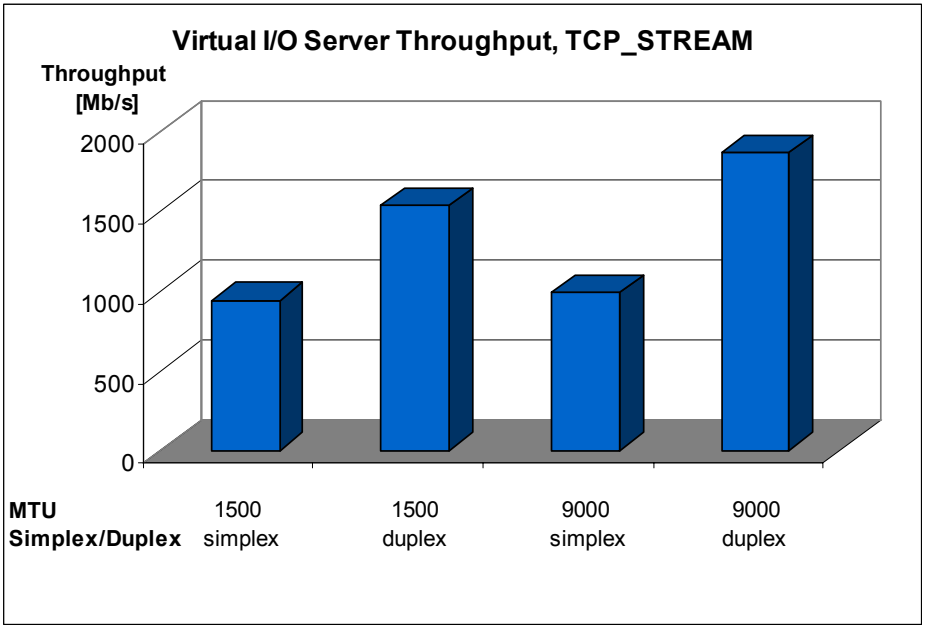


Figure 4-40 Throughput of the Virtual I/O-Server

The chart above 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 is hitting the line speed of the gigabit ethernet and is limited by the physical adapter.

Figure 4-41 on page 164 shows the utilization of the CPU in the Virtual I/O-Server. It has the same MTU parameters as the throughput measurement. Because of better comparison of the CPU utilization and throughput, the *utilization is normalized to 1Gb data throughput*.

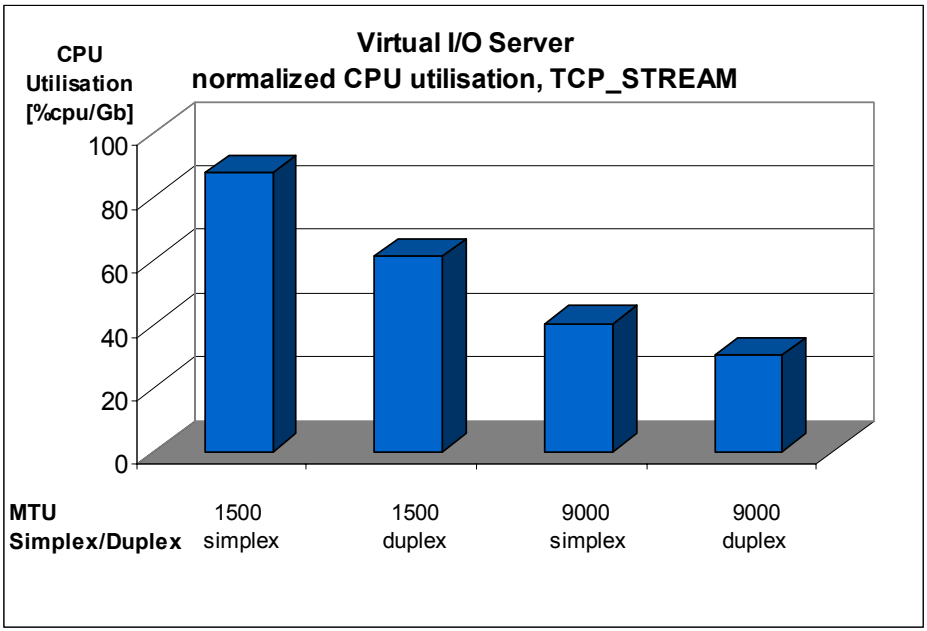


Figure 4-41 CPU utilization of the Virtual I/O-Server

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 CPU entitlements.
- CPU utilization per gigabit of throughput is higher with 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

Request/response time and latency

In this test, the workload TCP_RR is used to determine the transaction rate and the latency of the Shared Ethernet Adapter (SEA) and the gigabit ethernet adapter of the POWER4+ server.

The setup is the same as before and is shown in Figure 4-39 on page 162.

The data for the I/O-server are taken from the Shared Ethernet Adapter (SEA), the gigabit ethernet performance is measured at the two-way POWER4+ machine.

Results of Request/response time and latency

The next chart show the results of the TCP_RR benchmark.

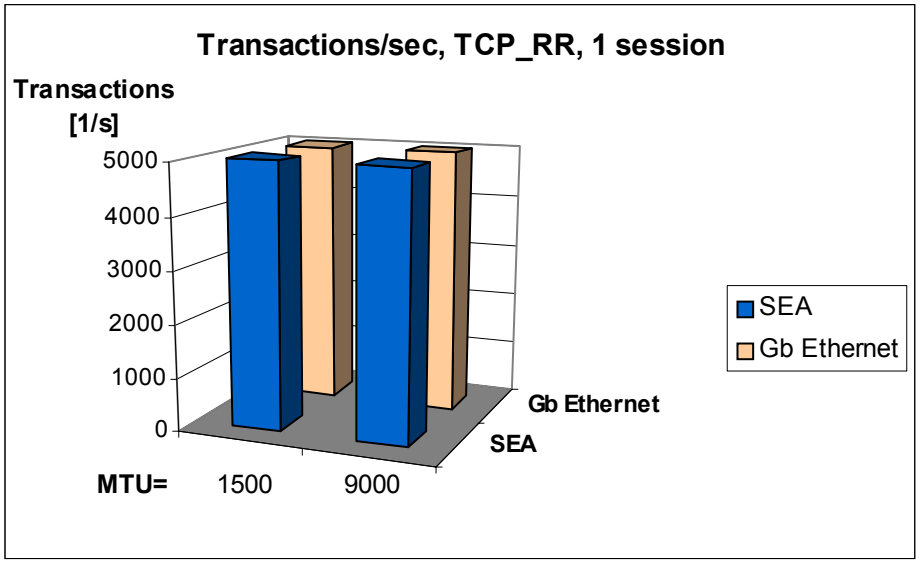


Figure 4-42 Transaction rates, TCP_RR, one session

Figure 4-42 and Figure 4-43 show the number of transactions that where done by Shared Ethernet Adapter and the Gigabit Ethernet for one session and also for twenty sessions. Note that the values shown for one session are both limited by the default setting of the gigabit ethernet adapters: INTR_RATE=10.000.

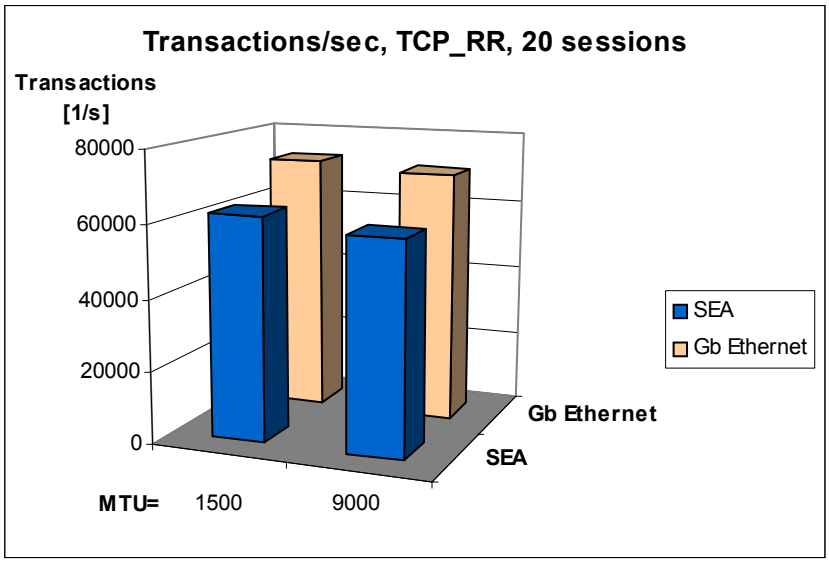


Figure 4-43 Transaction rates, TCP_RR, 20 sessions

Next is the examination of the latency in that environment. Latency is measured with the same parameters as the transaction rate.

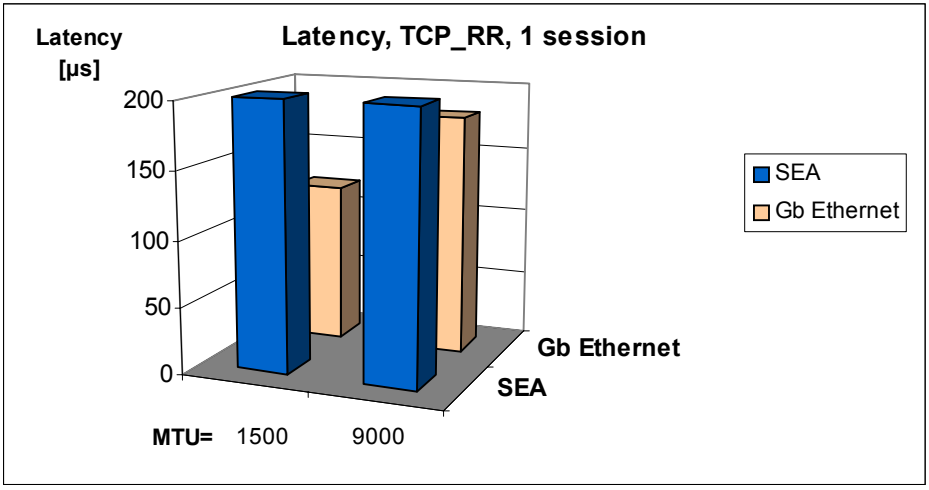


Figure 4-44 Latencies, TCP_RR, 1 session

Figure 4-44 and Figure 4-45 show the differences between the Shared Ethernet adapter and Gigabit Ethernet and the increasing latency, if the load grows (20sessions).

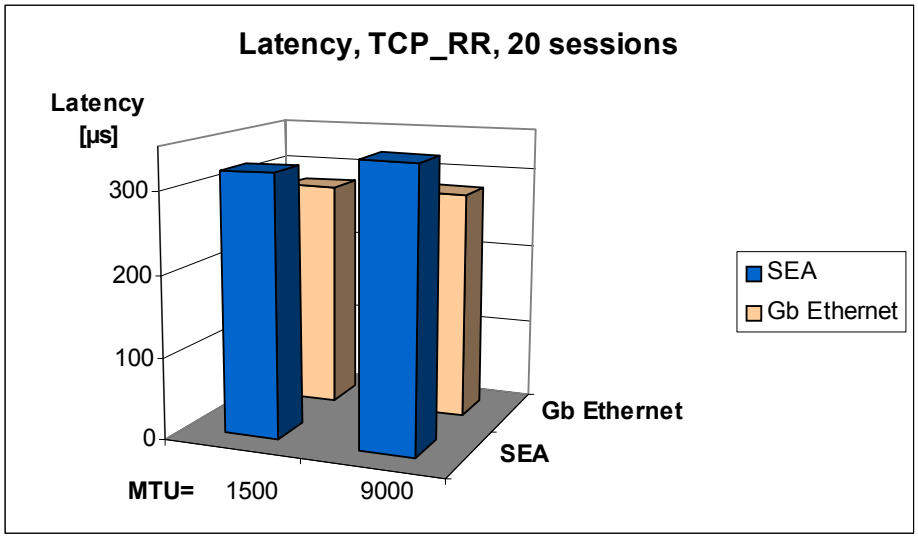


Figure 4-45 Latency, TCP_RR, 20 sessions

4.4.3 Implementation guidelines

Like Virtual LAN there is only a little experience with Virtual I/O Server and that results may vary in the future. However, here are some rules of thumb for designing a Virtual I/O-Server.

Important: The following recommendations are no guarantee for good performance.

- Know your environment and the network traffic
- Don't use the Shared Ethernet Adapter functionality of the Virtual I/O-Server if you expect heavy network traffic between Virtual LANs and local networks. Use a dedicated network adapter instead.
- If possible, use dedicated CPU's for the Virtual I/O-Server (no shared processors)
- Choose 9000 for MTU size, as this makes sense for your network traffic.
- Don't use the Shared Ethernet Adapter functionality of the Virtual I/O-Server for latency critical applications.
- With MTU size 1500, you need about one CPU per gigabit ethernet adapter streaming at media speed. With MTU size 9000, two Gigabit Ethernet adapters can stream at media speed per CPU.

4.5 Virtual SCSI

Virtual I/O allows the POWER5 based systems to support more partitions than it has slots for I/O devices by enabling the sharing of I/O adapters among 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 a hosting partition that uses two primitive functions Reliable Command / Response Transport and Logical Remote DMA to service I/O requests for a client running in a hosted partition, such that, the hosted partition appears to enjoy the services of its own SCSI adapter. The terms hosting and hosted partitions refer to platform partitions that are respectively servers and clients of requests, usually I/O operations, using the hosting partition's I/O adapters. This allows a platform to have more hosted partitions than it may have I/O adapters because the hosted partitions share I/O adapters via the hosting partition.

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 172, or single copy of the data is first moved to the memory space of the hosting partition before being DMAed to the hosted partition.

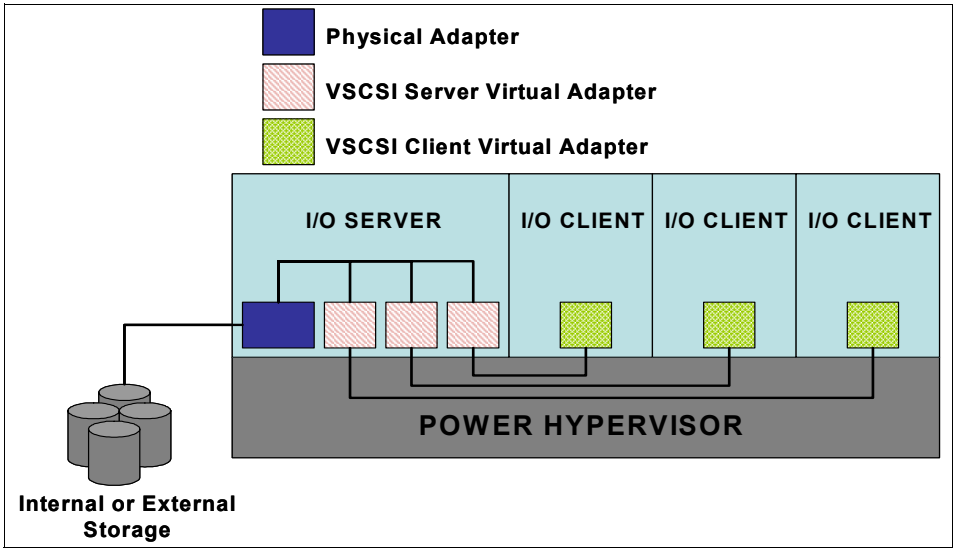


Figure 4-46 AIX 5L Server and Client Partitions

4.5.1 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 cdrom 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 4-47 on page 171.

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:

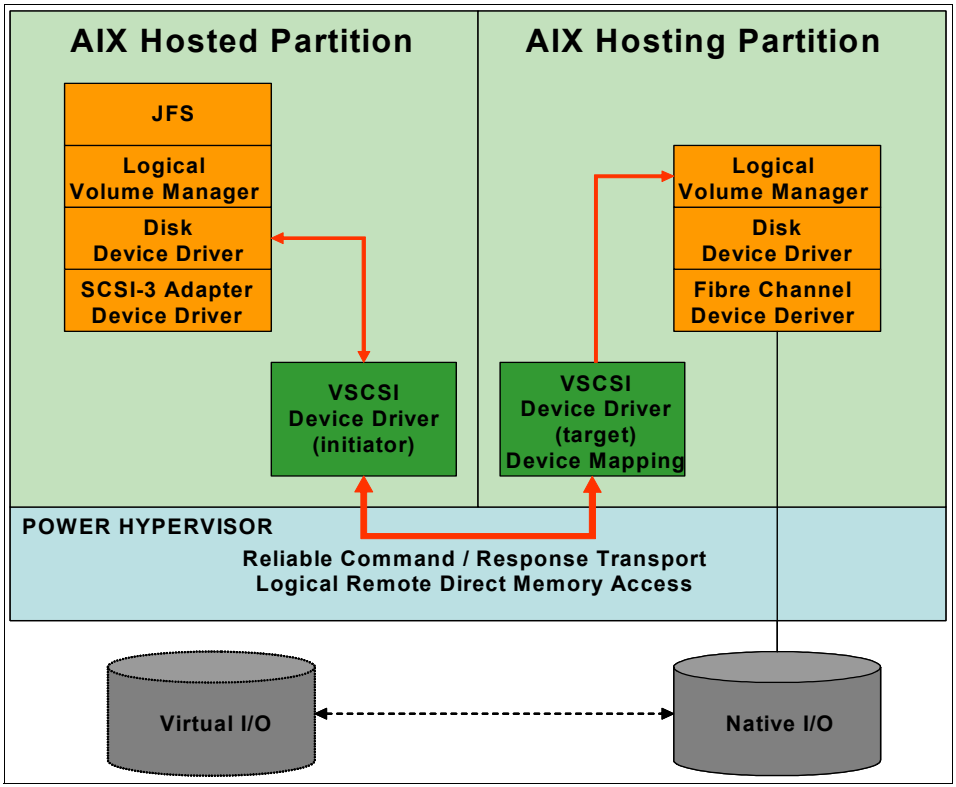


Figure 4-47 Reliable Command / Response Transport and LRDMA

Reliable Command / Response Transport

The transport over which SRP runs is the Reliable Command/Response Transport facility provided by POWER 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 detailed description of CRQ, refer to , “The Command/Response Queue (CRQ)” on page 133

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 a hosting partition to securely target memory pages within a hosted partition for virtual I/O operations. The hosting partition uses the hcall()s 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 hosting partition's I/O adapters targeting DMA buffers in the hosting partition's memory and having the hypervisor copy data between that DMA buffer and the hosted partition's memory.
- Redirected RDMA allows for a hosting partition to securely target its physical I/O adapter's DMA operations directly at the memory pages of the hosted partition.

The platform overhead of Copy RDMA is generally greater than Redirected RDMA, but this overhead may be offset if the hosting partition's DMA buffer is being used as a data cache for multiple virtual I/O operations.

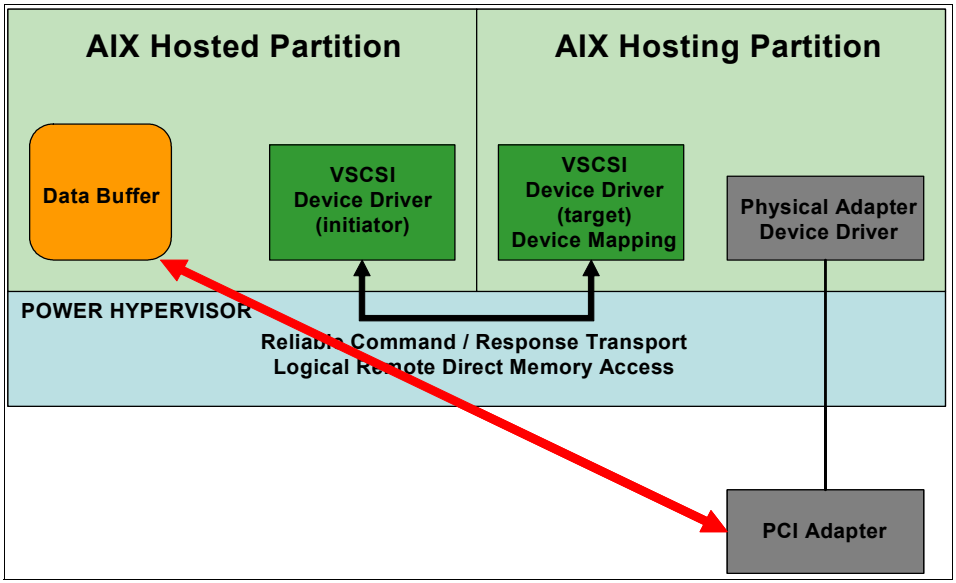


Figure 4-48 Logical Remote Direct Memory Access

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 hosted partition, and has no impact on the VSCSI architecture defined in this document. It is left entirely to the discretion of the hosting partition whether it first moves data from a physical device into it's own memory before moving (DMAing) the data to the hosted partition, or whether the hosting partition 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 hosted partition. The hosting partition 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 hosted partition's memory. This service is unidirectional, i.e. the hosted driver cannot use the service to write to, or read from, the hosting partition'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 and IBM @server i5 logical partition. Because virtual SCSI involves heterogeneous operating systems (AIX 5L, Linux, and i5/OS) 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 hosted partition (from the initiator port.) Since the hosting partition's RTCE tables are not authorized for access by the hosted partition, any attempt by the hosted partition to modify hosting partition 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 hosted partition 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

4.5.2 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 POWER 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 buf struct interface to the LVM. And it provides an SRP target interface to its partner initiator driver across the POWER 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 POWER 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 POWER 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 POWER 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 POWER 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 hosted partition 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_interrrupt 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_interrrupt. 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 4-49 on page 178 shows the possible partitioning of a physical disk on the hosting partition 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.

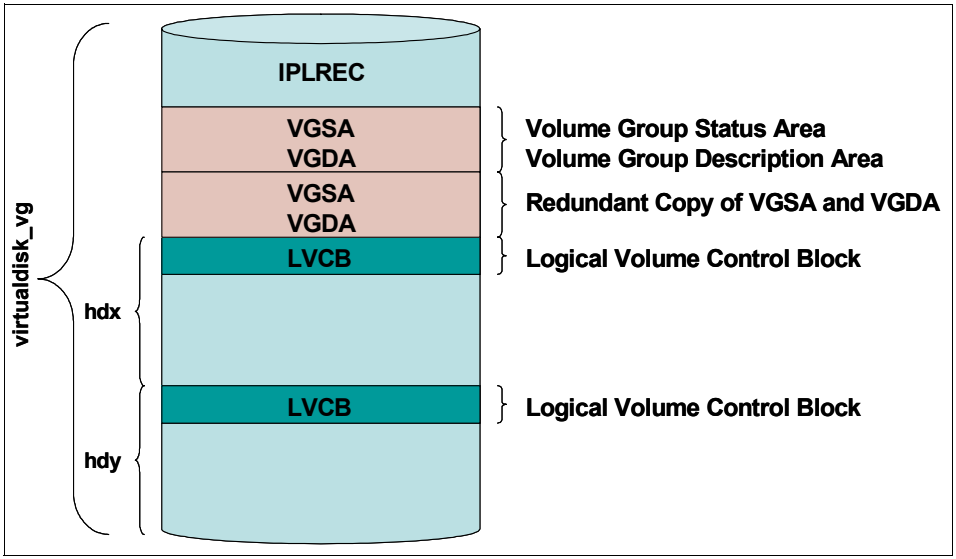


Figure 4-49 Volume group on Virtual I/O Server

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 hosting partition. When one hosted partition wins a reservation on a logical volume, the VSCSI virtual adapter target driver will have to refuse access by other hosted partitions to the logical volume. And when the hosted partition 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 hosting partition provides storage services for multiple hosted partitions. 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 hosting partition.

Command Tag Queuing

SCSI Command tag queuing refers to queuing commands to a SCSI device. Command tag queuing 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 queuing.

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 hosting partitions or two devices on the same hosting partition, 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.

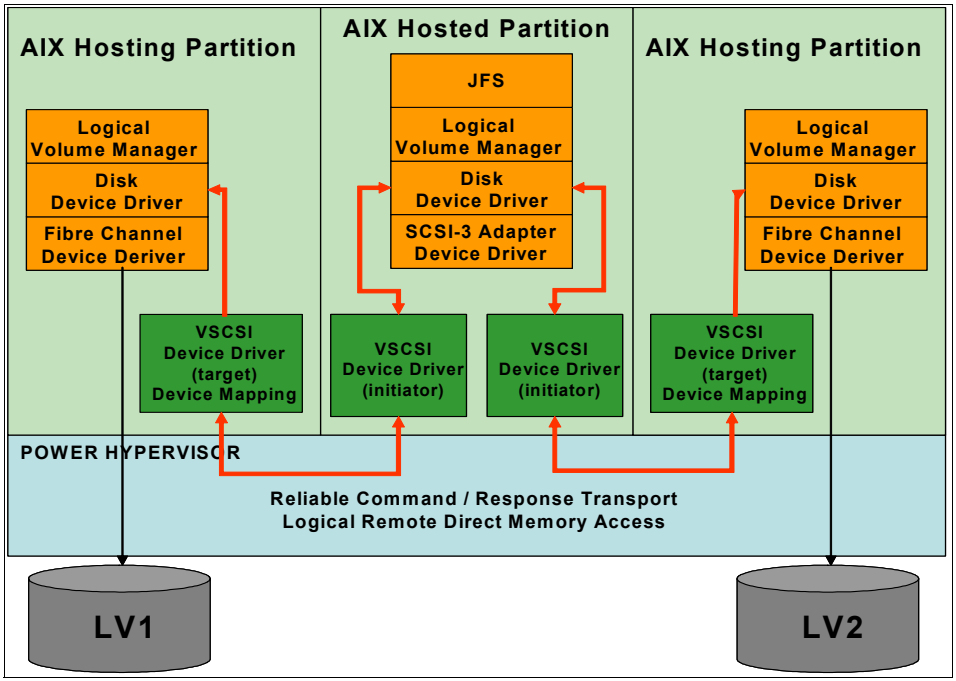


Figure 4-50 Using LVM mirroring for virtual SCSI

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.

4.5.3 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 CPU 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 CPU time to perform I/O as compared when using directly attached storage. This CPU 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 base-case configuration is when one physical disk is dedicated to a partition.

Virtual storage can still be manipulate using Logical Volume Manager just like an ordinary physical disk. Some performance considerations from dedicated storage are still application 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 an logical volume of a drive. If the entire drive is served to the cleint, then the rules and procudures apply on the client side, as if the drive were local. If the server partition provides the client, with a partition of a drive, an logical volume, then the server decides the area of the drive to serve to the client, when the logical volume was created), and setting 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 CPU utilization will always be higher than doing 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 (CPU, memory, and disk) are allocated to do the job.

- If not constrained by CPU 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. See iostat command output in Example 2-8 on page 72

A

Sample level 1 “a.” appendix heading (yHead0Appendix)

Note to Author: Describe the appendix contents here using these or similar words. Optionally add level 2 headings to a list using:
Special > Cross-Reference > Format: Head > Insert

This appendix provides/describes/discusses/contains ...

In this appendix we provide/describe/discuss ...

In this appendix:
In this appendix, the following are described:
This appendix provides/describes/discusses/contains the following:

- ...
- ...
- ...
- Sample level 2 appendix heading
  (created by **Special > Cross-Reference > Format: Head > Insert**)
- Sample next level 2 appendix heading

Sample level 2 heading (yHead1Appendix), new page

Note to Author: The first level 2 heading in an appendix should be the (yHead1Appendix) tag, skip to new page.

Add text here (Body0).

Sample level 2 heading (yHead2Appendix)

Add text here (Body0).

Sample level 3 heading (yHead3Appendix)

Add text here (Body0).

Sample level 4 heading (yHead4Appendix)

Add text here (Body0).

Sample level 5 heading (yHead5Appendix)

Add text here (Body0).

B

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG24????

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG24????.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
????????.zip	????Zipped Code Samples????

????????.zip	????Zipped HTML Documents????
????????.zip	????Zipped Presentations????

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	????MB minimum????
Operating System:	????Windows/Linux????
Processor:	???? or higher????
Memory:	????MB????

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder.

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 “How to get IBM Redbooks” on page 188. Note that some of the documents referenced here may be available in softcopy only.

- *????full title???????,* xxxx-xxxx

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.hpcaconf.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*

Online resources

These Web sites and URLs are also relevant as further information sources:

- Simultaneous Multithreading Project

http://www.cs.washington.edu/research/smt/

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

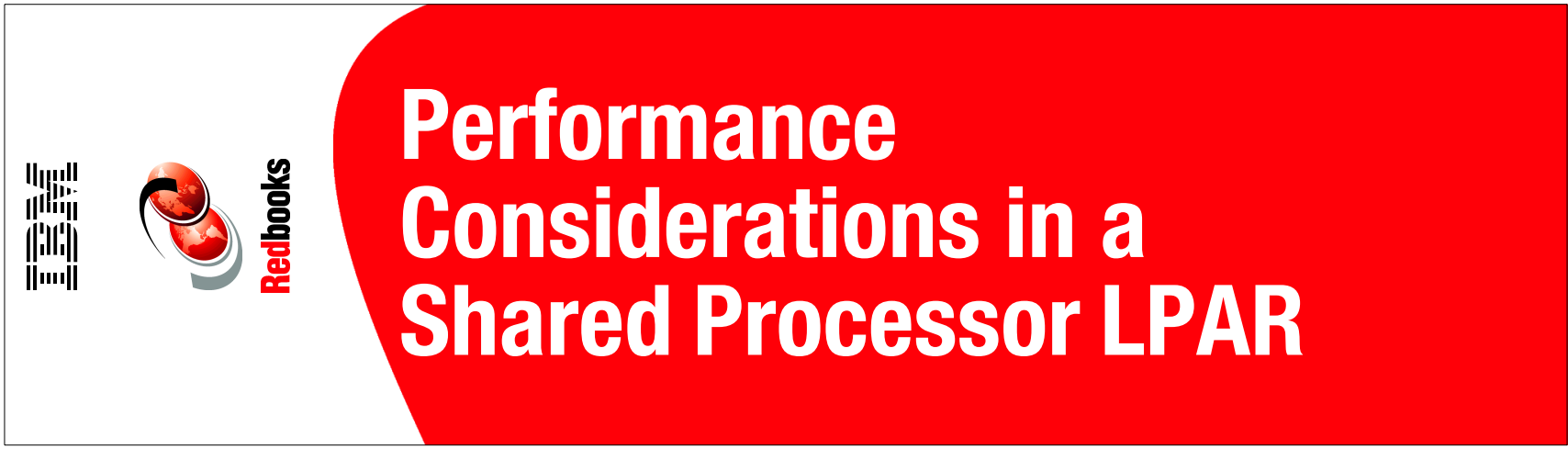
(1.5" spine)
1.5"<-> 1.998"
789 <->1051 pages

(1.0" spine)
0.875"<->1.498"
460 <-> 788 pages

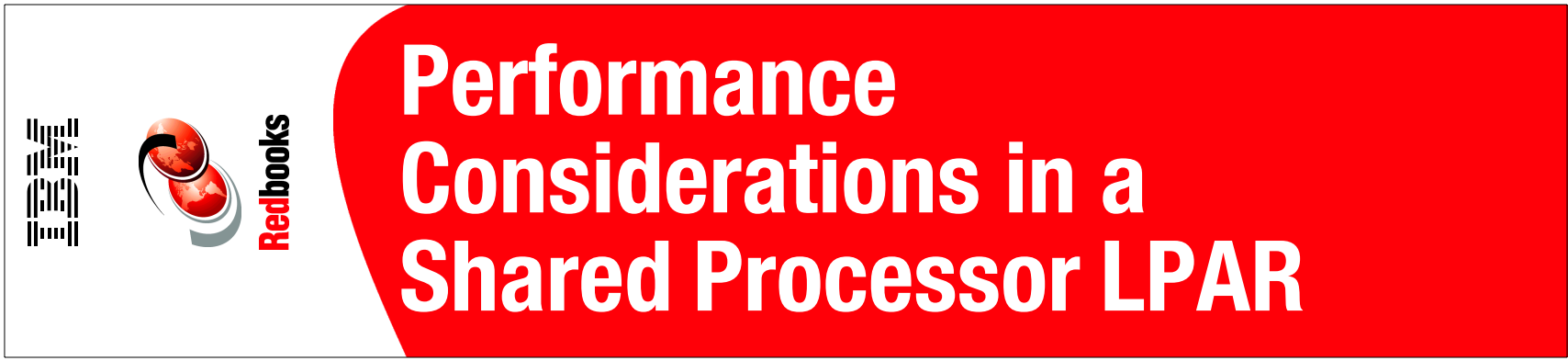
(0.5" spine)
0.475"<->0.875"
250 <-> 459 pages

(0.2"spine)
0.17"<->0.473"
90<->249 pages

(0.1"spine)
0.1"<->0.169"
53<->89 pages



(2.5” spine)
2.5”<->nnn.n”
1315<-> nnnn pages



(2.0” spine)
2.0” <-> 2.498”
1052 <-> 1314 pages

Draft Document for Review July 31, 2004 4:38 am

IBM @server p5 Virtualization Performance

Simultaneous Multi-Threaded (SMT) processor performance

Micro-partitioning performance

Virtual I/O performance

To deal with the performance management aspects, we will probably need
a fairly extensive Redbook to explain these changes. For example a title of:
"Performance considerations in a micro-partitioning environment"
might include the following chapters:
- Concepts and definitions <may be published separately as a whitepaper>
- Operating system considerations
 - AIX
 - Linux
- Simultaneous MultiThreaded (SMT) processor performance
- Micro-partitioning performance
- Virtual LAN performance
- Virtual SCSI performance
- Bringing it all together - using micro-partitioning, SMT, VLAN and VSCSI

This would likely be a large and complex Redbook. The Performance
team would like to expand their role to participate in the creation of this
Redbook as recognized authors. Hardware availability will probably gate this

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

SG24-5768-00 ISBN