DECwest/SDT Agenda

Digital Equipment Corporation
Confidential and Proprietary

David N. Cutler
DECwest Engineering
May 30, 1988

What will be Presented

Introduction

- Why are we here?
- Problem Statement
- Brief Statement of Alternatives
- Assumptions and Background

Technical Evaluation of MIPS Hardware and Software Architecture

- Hardware
- PRISM Software Reliance on Privileged Architecture
- Software
- Fortran compiler study

Analysis of Alternatives

- MIPS only
- PRISM only
- Both

Recommendations

Introduction
Problem Statement
Assumptions

What do we hope to accomplish today?

- Discuss the MIPS Plan B Proposal.
- Discuss the alternative strategies that might be pursued.
- Assess the impact on current projects and future products.
- Formulate a decision or at least a process by which a decision can be reached.

Problem Statement

- We are losing a significant business opportunity in the lowend, high performance, workstation market.
- VAXes are not perceived to be price performance competitive in this market.
- PRISM offers a price performance competitive architecture, but will not provide a product in the critical market window.
- Therefore, we are considering building workstation products to attack this market based on the MIPS microprocessor chips because they are available, competitive, and offer a time to market advantage.

Alternative Strategies

- Build our entry workstation products based on MIPS and continue our investment in PRISM as the long term solution.

- Build our entry workstation products based on MIPS, stop the development of the PRISM software and hardware architecture and switch all development to MIPS based products.

- Continue the execution of the current strategy which provides an entry workstation based on PRISM chips and server products for high performance computation and database/OLTP systems.

Assumptions and Background

- Enterprise computing requires a highly compatible family of systems that address the complete set of customer requirements.

- The PRISM effort was begun to provide a long term architecture on which to build a complete line of compatible systems that exhibit a high degree of VAX compatiblity and fit into the Digital Computing Environment.

- The PRISM product line must be phased into Digital's business carefully to avoid impingement on the current VAX business until such time as we have a complete set of software in place and are able to do the whole job for the customer.

- The VAX architecture has been a positive and profitable experience for Digital; a well defined architecture with many compatible implementations to protect customer investment.

- If Digital wants to remain a major innovator in the computer industry, we must control our destiny and have control over our software and hardware architectures.

A Quick Analysis of the MIPS Hardware Architecture

Attractive Features of the MIPS Hardware Architecture

- It's better than the Sun Sparc architecture.
- There are competitive chip implementations from several vendors available today.
- MIPSco's chip plans and semiconductor recruitment promise a steady flow of competitive chip implementations in the future.

BUT

- It was not designed with strong VAX compatibility in mind and therefore contains several features that are explicitly incompatible with VAX.

- It was not designed with the goal of being compatibly extendable to a 64-bit architecture in the future.

- It was not designed to be able to support high performance vector computing.

- It was not designed to allow a single set of software to run on multiple generations of implementation without change.

- It was not designed to have inherent high performance multiprocessing support.

Major Architectural Problems

- The architecture is not well specified in several areas especially cache coherency and exception processing (e.g. what happens when the I-stream is written into).

- There is no support for multiprocessing either in the form of interlocked instructions or processor identification (i.e. something similar to WHAMI).

- There is no support for vector instructions. Ardent was able to implement vector capabilities via a coprocessor with much difficulty especially with respect to memory management.

- Floating point capabilities are incompatible with VAX both computationally as well as value format.

- Memory management capabilities are inadequate for a 1990's processor architecture. Execute only and noexecute protection are not supported even though they are simple to implement.

Other Architectural Problems

- Multiply and divide use special hardware registers to hold their results. This prohibits pipelining and requires special instructions to move results to general registers.

- Integer overflow checking is supported for add and subtract operations, but not for multiply and divide.

- There are no defined rules for data sharing, atomicity, or cache coherency.

- Memory mapping does not allow user mode execution of code contained within the executive. This is something that has proven very useful in VMS and Mica.

- The interrupt system is single level with no interrupt priority. Every Digital architecture has provided prioritized interrupts because they have been found to be so useful.

Other Architectural Problems Cont.

- The various rounding modes of the floating point coprocessor are programmed via a status and control register rather than being part of the instruction set. This is the archaic technology that we abandoned with the PDP–11/45.

- The floating point coprocessor uses a separate register file which causes compilers to manage another nonhomogeneous resource and causes additional movement of operands between the general registers and the floating point registers.

- The result of a floating point comparison operation is stored in a single bit in the floating status register. This serializes floating comparison operations and prevents pipelining.

- Certain floating point operations require software to wait a specified number of cycles before attempting to read the result.

Other Architectural Problems Cont.

- The floating point coprocessor generates interrupts when an exception is discovered. This either prevents overlap of integer instructions or results in floating point interrupts at inappropriate times (same problem we had with the PDP–11/45).

- The mapping of memory is very unusual in that one quarter of the virtual address space is permanently mapped to the first 512mb of physical memory. If done compatibly, this will cause a very inefficient migration to a large address space.

- There is no architected interval clock, cycle counter, or any other way in which time or accounting information can be implemented. This capability must be implemented via an external device.

- A dedicated general purpose register is used for subroutine linkage. This prevents the generation of optimized procedures linkages for locally compiled leaf procedures.

- There is no support for ASTs in hardware. Besides VAX/VMS and Mica even VAX/ULTRIX (nee Berkeley UN*X) has found this useful.

Other Architectural Problems Cont.

- There is no architected software interface to privileged hardware functions.

- The coprocessor architecture forces floating point, vectors, and interlock operations to be appended to the processor like a bag on the side rather than being integrated into the architecture and instruction set.

- Exception/interrupt vectors are fixed in physical memory which will cause availability problems if that particular memory fails.

- In certain exception cases, continuation of the program requires interpretation of the instruction stream to effect a delayed branch.

- Although the floating point coprocessor is able to directly pick up instructions from the I-cache bus, it is not clear additional coprocessors will be able to do so without causing serious signal integrity problems or a slowdown in chip speed.

Hardware Architecture Conclusions

- The MIPS architecture is well-suited for UN*X based workstations. It is, however, not adequate to build a family of system products.

- The MIPS architecture leaves too much unspecified or ambiguously specified. This will produce both development and support problems. (Currently planned implementation are all slightly incompatible with each other.)

- Without a major overhaul, the MIPS architecture would yield a family of prodcuts with weak compatibility, even at the application level (i.e. no binary compatibility).

- Incompatibility with VAX would cause serious erosion of the customer base.

PRISM Software Dependence on PRISM Hardware Architectural Features Not Present in MIPS Hardware Architecture

Dependence on Priviliged Architecture

- PRISM software is designed to take maximum advantage of the PRISM privileged hardware architecture and expects it to remain absolutely compatible from implementation to implementation. (Epicode is provided to make it easy for hardware implementers to do this.)

- Interlocked instructions - RMAxl/CMPSWxl, for fine grained parallelism, multiprocessor synchronization, and support of Digital I/O controllers.

- ASTs - I/O completion, suspend/resume, force exit, enter context of target thread to obtain system management and performance information.

- Priority interrupts - software interrupts for scheduling, ASTs, external entry to system debugger.

- Cycle count register - Performance data collection (very important in vector code analysis) and accounting.

Dependence on Privilged Architecture Cont.

- FOE, FOR, and FOW bits in TB - provide state of the art memory protection for proprietary code (execute only) and greater fault isolation (no execute on data).

- Full memory mapping - enable user mode execution of code in the system part of address space for condition handling, instruction emulation (vector emulation), AST delivery, AST routine to force exit, thread startup, initial image activation, and raise condition in target thread.

- Transparent caches with well defined data sharing rules that don't force the buffering of all I/O through noncached data areas.

- Interprocessor interrupts - thread scheduling, TB maintenance, AST delivery, system shutdown, ICache flushes, and system debugger synchronization in multiprocessor configurations.

Dependence on Privilged Architecture Cont.

- Powerfail - the ability to increase availability by riding through power transients.
- Interval timer - quantum runout, timers, and system time.
- Multiprocessor console and boot architecture - implementation independent support of console, processor self test and configuration, console terminal, restart parameter block, and boot coordinator.

The MIPS Privileged Architecture Impact

- Three months to fully analyze the MIPS architecture and seek alternate solutions to the privileged architecture problems.

- Three months to implement privileged architecture solutions and design changes caused by lack of capabilities in MIPS hardware architecture.

- Full impact on privileged part of software is six months and does not include any time to architect system independent solutions to the above problems that would prevent the same problem from reoccurring in the next implementation of the MIPS hardware architecture.

A Quick Analysis of the MIPS Software Architecture

Attractive Features of the MIPS Software Architecture

- State of the art compiler technology.
- Delayed branch filling, common subexpression elimination, code motion, invariant removal from loops, strength reduction, and code scheduling optimizations, etc.
- A single code generator shared by several compiler frontends including C, Fortran, Pascal, ADA, PL/I, and Cobol.
- Optimized calling conventions for leaf procedures.
- Berkeley based, System V compatible, UN*X implementation.
- All the components exist today.

BUT

- Maybe compilers are not so state of the art after all. No decomposition or vectorization support. They require an assembler phase as last phase of compilation (archaic technology but it may help them with their implementation to implementation hardware incompatibilities).

- Little evidence of state of the art features such as code scheduling and loop unrolling in study of MIPS Fortran compiler (Grove). (A future version does unrolling; the assembler does simple load and constraint scheduling.) Digital PRISM ULTRIX V1 compilers will equal or exceed the MIPS compilers.

- The debugging, performance coverage, and language sensitive editor environment is far from state of the art and is inferior to that of VAX VMS and what is planned for PRISM ULTRIX.

- Compiler frontends bought from third parties and not developed by MIPS which begs the question of their real quality (e.g. Fortran not 100% VAX compatible as claimed).

More BUTs

- The MIPS UN*X implementation requires big endian addressing which is incompatible with VAX and will require byte swapping for network communication with VAXes (it is not clear how this problem will be solved).

- The MIPS calling conventions do not adequately support modern languages such as ADA. Specifically, frame based condition handling is not provided which makes it extremely difficult to support a multilanguage environment , let alone structured condition handling which improves software reliability.

- The MIPS calling conventions do not take into account the manipulation of the floating point status register and use a very peculiar register allocation scheme (one register fix allocated to the assembler, two others to the operating system!).

- There is no evidence that the MIPS software architecture is designed to support multithreading.

And Yet More BUTs

- The MIPS calling conventions are not nearly as powerful as the PRISM calling standard and do not allow register frames as well as stack frames for storing procedure context. They also do not allow the passing of arguments in registers when external procedures are called.

- The MIPS software and hardware architectures are not designed to easily migrate to a 64-bit machine architecture.

- The MIPS math library is most likely the same poor quality library that is supplied with most UN*X systems which is far inferior to the VAX VMS library.

- Mixed language programs have never worked well on UN*X systems. Although the MIPS conventions show improvement over other UN*X systems, they are not as powerful as the VAX VMS conventions. The PRISM Calling Standard makes further improvements over the VAX VMS conventions in the areas of strings, multi-dimensional arrays, and procedure values.

While

- The PRISM software and hardware architectures were designed together to provide maximum VAX VMS compatibility. Application code can be run on VAX VMS, Mica, and PRISM ULTRIX by simply recompiling.

- The PRISM software architecture provides for multithreading and integration with a state of the art debugger and development environment.

- The PRISM software architecture provides separate compilation of source modules across all languages.

- The PRISM compilers will do loop unrolling, inline procedure expansion, vectorization, automatic decomposition, state of the art register allocation, sophisticated code scheduling, and interprocedural analysis and optimization across source modules. (MIPS doesn't believe in vectors at all.)

- The PRISM software architecture has been designed to ensure an easy and transparent migration to the 64-bit PRISM architecture.

While Cont.

- The PRISM architecture presents the opportunity for performance far in excess of the MIPS performance. This is a combination of hardware and software architecture.

- The PRISM architecture makes the conversion of the VAX math library to PRISM easy since the same data types and rounding modes are provided.

- The PRISM architecture provides the foundation to build state of the art database and transaction processing systems.

- The PRISM architecture provides a compatible growth and migration path for VAX VMS as well as VAX ULTRIX.

- The PRISM architecture gives Digital an opportunity to lead the market with the eventual introduction of a 64-bit architecture.

Software Architecture Conclusions

- MIPS has an impressive set of compilers for their UN*X system.
- The MIPS language environment is not state of the art and lacks several key components required of such an environment.
- The PRISM software environment will be far superior to anything MIPS is likely to have in the future and represents a far more compatible growth path for current Digital customers.

Strategy
Alternative Analysis

MIPS Only Strategy

Build our entry workstation product based on MIPS, stop the development of the PRISM software and hardware architecture, and switch all development to MIPS based products.

MIPS Only - Pros

- Timely introduction of a competitive UN*X based workstation into the market.
- Early sign up of ISV's on MIPS provided development systems.
- Large selection of available good compilers.
- Steady flow of competitive parts from several semi-conductor vendors.
- Resources working on PRISM can be utilized in MIPS based products.

MIPS Only - Cons

- Architecture unsuitable for a complete family of compatible products and will be difficult to extend to 64-bits. Questionable multiprocessing and vector performance.

- Cheyenne and Glacier products delayed one year with no promise of increased performance. Shrike and Osprey will also have to be restarted resulting in inevitable delay.

- Possible customer perception that Digital has abandoned its proprietary architecture and VAX customers are left with nowhere to migrate.

- No control over hardware or software architecture unless we buy MIPS.

- No long term competitive advantage - anyone can produce the same products.

- Long term compatibility problem with VAX.

PRISM Only Strategy

Continue the execution of the current strategy which provides an entry workstation based on PRISM chips and server products for high performance computation and database/OLTP systems.

PRISM Only - Pros

- Digital proprietary architecture providing the best base for future system products with strong VAX compatibility, integrated vector and multiprocessing support, and extensible to 64-bits.

- Strong compatibility and migration path for VAX VMS customer base with application transportability across VAX VMS, PRISM ULTRIX, and Mica.

- Glacier, Cheyenne, Shrike, and Osprey delivered according to current schedules.

- Truly state of the art compilers and software architecture to support database, OLTP, and parallel processing.

PRISM Only - Cons

- No plans for a lower cost workstation that can meet time to market constraints.
- Not as many compilers available at FRS.
- Dependent on Digital semiconductor design and manufacturing as sole source of parts.
- Current level of funding inadequate to maintain competitive flow of chip and software products.

Do Both Strategy

Build our entry workstation products based on MIPS chips and continue our investment in PRISM as the long term solution.

Do Both - Pros

- Everybody at Digital is happy except maybe the financial people!
- MIPS workstation, PRISM servers with a switch to an all PRISM strategy in one or two generations.
- Get workstation now, proprietary architecture and VMS compatibility too.
- Hedge our bets on risky programs, an egg in every basket.
- Product quality acceptable; MIPS for today's UN*X customers, PRISM for traditional Digital customers.

Do Both - Cons

- Possible customer perception that MIPS workstation is a point product. Digital is buying time until PRISM is ready; don't commit to Digital workstations.

- Will be difficult to build a single set of point products and switch to PRISM. Most likely will have to build full line of MIPS based workstations.

- Not enough resources now; adding another product family will not get timely application compatibility for MIPS based products.

- Bewildering array of products, architectures, and operating systems.

- More pressure on funding for both programs; may not be able to end up winning in either.

Recommendations

- Execute the PRISM only strategy and hold it stable long enough for products to be built, PLEASE.
- IMMEDIATELY staff and fund a PRISM based PVAX product to complement Shrike and Osprey.
- Fund and staff CMOS III and CMOS IV PRISM chip teams NOW for scalar, cache control, and vector parts.
- Put team together NOW to produce custom Bipolar chip set - today's effort is inadequate.
- Consider letting other semiconductor vendors produce PRISM chips; leverage their design resources and processes to get parallel development.
- Provide additional directed funding for more compiler and layered products earlier.
- Consider licensing Mica and Pillar to establish a real operating system standard that is built on state of the art principles (we are planning to file upwards of about 50 patents on Mica).

MIPS Computer Systems, Inc.

PERFORMANCE BRIEF
PART 1: CPU BENCHMARKS

MIPS M/120-5, M/1000, M/800 and M/500 Systems
UMIPS-BSD (2.1), UMIPS-V (3.0)
Compiler Release 1.31

Revision History:

Issue 3.2, May 1988

Issue 3.0, October 1987
Issue 2.2, April 1987
Issue 1.0, October 1986
Issue 0, April 1986

1 Introduction 1

2 Benchmark Summary 2
2.1 Choice of Benchmarks 2
2.2 Benchmark Summary Data 2

3 Methodology 5

4 Integer Benchmarks 8
4.1 MIPS UNIX Benchmarks (MIPS UNIX) 8
4.2 Dhrystone (DHRY 1.1) 10
4.3 Stanford Small Integer Benchmarks (STAN INT) 14

5 Floating Point Benchmarks 16
5.1 Livermore Fortran Kernels (LLNL DP) 16
5.2 LINPACK (LNPK DP and LNPK SP) 20
5.3 Spice Benchmarks (SPCE 2G6) 23
5.4 Digital Review (DIG REV) 25
5.5 Doduc Benchmark (DDUC) 26
5.6 Whetstone 28
5.7 Stanford Floating Point Benchmarks 29

6 Acknowledgements 30

7 References 31

1. Introduction

New Features of This Issue
Added to this issue are performance numbers for our new M/120-5 system, and new 1.30 compiler numbers for a few benchmarks (in particular, fortran blas linpack mflops are now almost equal to coded blas mflops).

Benchmarking - Caveats and Comments
While no one benchmark can fully characterize overall system performance, the results of a variety of benchmarks can give *some* insight into expected real performance. A more important benchmarking methodology is a side-by-side comparison of two systems running the same real application.

We don't believe in characterizing a processor with just a single number, but we follow (what seems to be) standard industry practice of using a mips-rating that essentially describes overall integer performance. Thus, we label a 5-mips machine to be one that is about 5X (i.e., anywhere from 4X to 6X!) faster than a VAX 11/780 (UNIX 4.3BSD, unless we can get Ultrix or VAX/VMS numbers) on integer performance, since this seems to be how most people intuitively compute mips-ratings. Even within the same computer family, performance ratios between processors vary widely. For example, [McInnis 87] characterizes a "6 mips" VAX 8700 as anywhere from 3X to 7X faster than the 11/780. Floating point speed often varies more than, and scales up slower than integer speed versus the 11/780. In particular, microprocessor FP performance has lagged integer speed.

We try to be as straightforward and detailed as possible, so you can make your own judgements about the claims here. We include the raw data, so that you can catch any mistakes in data reduction. We include benchmarks that are popular, but that we don't believe are very meaningful, and we say so.

This paper analyzes one important aspect of overall computer system performance - user-level CPU performance.

Performance Summary
Our new 16.7 MHz MIPS M/120-5 (our "12 vax-mips" machine) runs:

- on realistic user-level integer programs:
  - about 10-15X faster than the VAX 11/780 (4.3BSD UNIX) on UNIX commands.
  - about 1.5-1.8X faster than a Sun-4/200 with SunOS 3.2L
- on realistic FORTRAN floating point tests:
  - 9-14X faster than the 11/780 under VAX/VMS
  - 1.6 - 2.5X faster than a a VAX 8650 or 8700 under VAX/VMS
  - about 2.5X faster than a Sun-4/200 (SunOS 3.2L) on most tests.

This *Brief* offers comparative data that places MIPS' computational performance, not at the microprocessor level, but at or above high-end superminis, even on floating point performance. M/120-5s even act like mini-supers on some tests. *We believe these machines are defining a new class of system: the micro-super: faster than a super-mini, with performance balanced between integer and FP, but at microprocessor prices.*

If you like our approach, please let us know. If you've got suggestions for improvement, we'd be glad to look at them, especially if you have more accurate numbers. We are trying to give an accurate picture, rather than hype meaningless mips- and flops-ratings that have little to do with real performance. The next few pages give overall benchmark charts and data, followed by more detailed information.

MIPS Computer Systems does not warrant or represent that the performance data stated in this document will be achieved by any particular application. (We have to say that, sorry.)

2. Benchmark Summary

2.1. Choice of Benchmarks

This brief offers both public-domain and MIPS-created benchmarks. We prefer public domain ones, but some of the most popular ones are inadequate for accurately characterizing performance. In this section, we give an overview of the importance we attach to the various benchmarks, whose results are summarized on the next page.

Dhrystone [DHRY 1.1] and Stanford [STAN INT] are two popular small integer benchmarks. Compared with the *fastest* VAX 11/780 systems, the M/120-5 is 15-17X faster than the VAX on these tests, and yet, we rate the M/120-5 as a 12-vax-mips machine. *In fact, if we chose different VAX software to compare against, we could call the M/120-5 17-19 mips, right now.* However, our mips-ratings are derived from the performance of real programs, and we conclude the artificial benchmarks are *not* representative. We observe that many vendors claim mips-ratings based on the most favorable choice of benchmarks ("Dhrystone mips", for example) or performance estimates for machines not built. If you're comparing an M/120-5 against such claims, *it is a 19-mips machine.*

While we present Dhrystone and Stanford, we feel that the performance of large UNIX utilities, such as *grep, yacc, diff,* and *nroff* is a better (but not perfect!) guide to the performance customers will receive. These four, which make up our [MIPS UNIX] benchmark, demonstrate that performance ratios are not single numbers, but range here from 10X to 15X faster than the VAX.

Even these UNIX utilities tend to overstate performance relative to large applications, such as CAD applications. Our own vax-mips ratings are based on a proprietary set of larger and more stressful real programs, such as our compiler, assembler, debugger, and various CAD programs.

For floating point, the public domain benchmarks are much better. We're still careful not to use a single benchmark to characterize all floating point applications.

The Livermore Fortran kernels [LLNL DP] give insight into both vector and non-vector performance for scientific applications. Linpack [LNPK DP and LNPK SP] tests vector performance on a single scientific application, and stresses cache performance. Spice [SPCE 2G6] and Doduc [DDUC] test a different part of the floating point application spectrum. The codes are large and thus test both instruction fetch bandwidth and scalar floating point. Digital Review Magazines benchmark [DIG REV] is a compendium of FORTRAN tests that measure a wide variety of behavior, and seem to correlate well with some classes of real programs.

2.2. Benchmark Summary Data This section summarizes the most important benchmark results described in more detail throughout this document. The numbers show performance relative to the VAX 11/780, i.e., larger numbers are better/faster.

- A few numbers have been estimated by interpolations from closely-related benchmarks and/or closely-related machines. The methods are given in great detail in the individual sections.
- Several of the columns represent summaries of multiple benchmarks. For example, the MIPS UNIX column represents 4 benchmarks, the SPICE 2G6 column 3, and LLNL DP represents 24.
- In the Integer section, MIPS UNIX is the most indicative of real performance.
- For Floating Point, we especially like LLNL DP (Livermore FORTRAN kernels), but all of these are useful, non-toy benchmarks.
- In the following table, "Pub mips" gives the manufacturer-published mips-ratings. As in all tables in this document, the machines are listed in increasing order of performance according to the benchmarks, in this case, by Integer performance.
- The summary includes only those machines for which we could get measured results on almost all the benchmarks and good estimates on the the results for the few missing data items.
- The next few pages contain a summary table and graph.

Summary of Benchmark Results

(VAX 11/780 = 1.0, Bigger is Faster)

Integer (C)			Floating Point (FORTRAN)							
MIPS UNIX	DHRY 1.1	STAN INT	LLNL DP	LNPK DP	LNPK SP	SPCE 2G6	DIG REV	DDUC	Publ mips	System
1	1	1	1	1	1	1	1	1	1	VAX 11/780#
2.1	1.9	1.8	1.9	2.9	2.5	1.6	*1.5	*1.3	2	Sun-3/160 FPA
*4	4.1	4.7	2.8	3.3	3.4	2.4	*2	1.7	4	Sun-3/260 FPA
5.5	**7.4**	**7.2**	**2.5**	**4.3**	**3.7**	**3.4**	**3.6**	**3.8**	**5**	**MIPS M/500**
*6	5.9	6.5	5.9	6.9	5.6	*5.3	4.6	5.2	6	VAX 8700
8.4	10.8	7.3	4.5	7.9	6.4	4.1	3.2	3.5	10	Sun-4/200
9.2	**11.3**	**11.8**	**8.1**	**7.1**	**7.6**	**6.6**	**5.6**	**7.3**	**8**	**MIPS M/800**
11.3	**13.5**	**14.1**	**10.8**	**10.4**	**14.0**	**8.0**	**7.2**	**8.8**	**10**	**MIPS M/1000**
13.1	**15.6**	**15.9**	**12.1**	**15.0**	**16.0**	**9.7**	**8.6**	**11.3**	**12**	**MIPS M/120-5**

VAX 11/780 runs 4.3BSD for MIPS UNIX, Ultrix 2.0 (vcc) for Stanford, VAX/VMS for all others. Use of 4.3BSD (no global optimizer) probably inflates the MIPS UNIX column by about 10%.

* Although it is nontrivial to gather full set of numbers, it is important to avoid holes in benchmark tables, as it is too easy to be misleading. Thus, we had to make reasoned guesses at these numbers. The MIPS UNIX values for VAX 8700 and Sun-3/260 were taken from the Published mips-ratings, which are consistent (+/- 10%) with experience with these machines. DIG REV and DDUC were guessed by noting that most machines do somewhat better on DIG REV than on SPCE, and than a Sun-3/260 is usually 1.5X faster than a Sun-3/160 on floating-point benchmarks.

Benchmark Descriptions:

MIPS UNIX
MIPS UNIX benchmarks: *grep, diff, yacc, nroff*, same 4.2BSD C source compiled and run on all machines. The summary number is the geometric mean of the 4 relative performance numbers.

DHRY 1.1
Dhrystone 1.1, any optimization except inlining.

STAN INT
Stanford Integer.

LLNL DP
Lawrence Livermore Fortran Kernels, 64-bit. The summary number is the given as the relative performance based on the geometric mean, i.e., the "middle" of the 3 means.

LNPK DP
Linpack Double Precision, FORTRAN.

LNPK SP
Linpack Single Precision, FORTRAN.

SPCE 2G6
Spice 2G6, 3 public-domain circuits, for which the geometric mean is shown.

DIG REV
Digital Review magazine, combination of 33 benchmarks.

DDOC
Doduc Monte Carlo benchmark.

The charts on the next page shows several machine combinations.

100ns
180ns
~~Sorry~~

M120-5: R2000 @ 16.7 MHz * 12 VUPS
64KB I-cache (16K x 4's)
64KB D-cache (16K x 4's)
cache miss penalty 5 cycles (1 word)
write time 4 cycles
1 word refill
100ns DRAMs
parity

---

workstation: R2000 @ 16.7MHz 12 VUPS
32KB I-cache (8K x 8's)
32KB D-cache
cache miss penalty for 1 word 4 cycles
write time 3 cycles (2 in page mode)
optional 2or4 word refill (6 cycles, 10 cycles)
100ns DRAMs
parity

M/1000:

R2000 @ 15.0 MHz 9.7 VUPS
64KB I-cache
64KB D-cache
cache miss penalty ~~8~~ 8 or 9 (1 word)
write time 8 cycles same bank
4 cycles different bank
1 word refill
ECC

R3000-based solution:
R3000 @ 25.0 MHz
64KB I-cache, 64KB D-cache (20ns ~~8~~ 16K x 4's)
cache miss penalty for 16 word refill 12 + 16 = 28 cycles
write time 6 random, 2 page mode

3. Methodology

Tested Configurations

When we report measured results, rather than numbers published elsewhere, the configurations were as shown below. These system configurations do not necessarily reflect optimal configurations, but rather the in-house systems to which we had repeatable access. While we think that the VAX-11/780 configuration is fairly representative of typical "real world" configurations. Also, newer Suns have faster (16.7MHz) 68881s, and Sun sells a Weitek-based FPA board that provides substantially better performance than the MC68881. When we've had the faster results available, we've quoted them in place of our own system's numbers.

DEC VAX-11/780

Main Memory:	8 Mbytes
Floating Point:	Configured with FPA board.
Operating System:	4.3 BSD UNIX.

Sun-3/160M

CPU:	(16.67 MHz MC68020)
Main Memory:	8 Mbytes
Floating Point:	12.5 MHz MC68881 coprocessor (compiled -f68881).
Operating System:	SunOS 3.2 (4.2BSD)

MIPS M/500

CPU:	8MHz R2000, in R2300 CPU board, 16K I-cache, 8K D-cache
Floating Point:	R2010 FPA chip (8MHz)
Main Memory:	8 Mbytes (2 R2350 memory boards)
Operating System:	UMIPS-BSD 2.1 (4.3BSD UNIX with NFS)

MIPS M/800

CPU:	12.5 MHz R2000, in R2600 CPU board, 64K I-cache, 64K D-cache
Floating Point:	R2010 FPA chip (12.5MHz)
Main Memory:	8 Mbytes (2 R2350 memory boards)
Operating System:	UMIPS-BSD 2.1

MIPS M/1000

CPU:	15 MHz R2000, in R2600 CPU board, 64K I-cache, 64K D-cache
Floating Point:	R2010 FPA chip (15 MHz)
Main Memory:	16 Mbytes (4 R2350 memory boards)
Operating System:	UMIPS-BSD 2.1

MIPS M/120-5

CPU:	16.7 MHz R2000, 64K I-cache, 64K D-cache
Floating Point:	R2010 FPA chip (16.7 MHz)
Main Memory:	16 Mbytes (2 memory boards)
Operating System:	UMIPS-V.3 3.0

Test Conditions

All programs were compiled with -O (optimize), unless otherwise noted.

C is used for all benchmarks except Whetstone, LINPACK, Doduc, Spice 2g.6, Hspice, and the Livermore Fortran Kernels, which use FORTRAN. When possible, we've obtained numbers for VAX/VMS, and use them in place of UNIX numbers. The MIPS compilers are version 1.21 or 1.31.

User time was measured for all benchmarks using the /bin/time command.

Systems were tested in normal multi-user development environment, with load factor <0.2 (as measured by *uptime* command). Note that this occasionally makes them run longer, due to slight interference from background daemons and clock handling, even on an otherwise empty system. Benchmarks were run at least 3 times and averaged. The intent is to show numbers that can be reproduced on live systems.

MIPS R2300, R2600, and R2800 CPU Boards and M120-5 system

The MIPS R2300 VME CPU board is based on the 8 MHz (125 ns cycle time) version of the MIPS R2000 CPU chip. The R2300 also includes the MIPS R2010 Floating Point Accelerator chip, a 16 Kbyte instruction cache, an 8 Kbyte data cache, and a four-stage write buffer.

The MIPS R2600 CPU board is similar to the R2300, but runs at 12.5MHz (80ns cycle time), and has larger caches (64KB each for instruction and data).

The R2800 CPU board has the same cache size as the R2600, but runs at 15MHz (67ns cycle).

All 3 boards are 3-high, 3-deep Eurocard format, i.e., 366.7mm high by 280mm deep. The R2600/R2800 core CPU complex [CPU, FPA, write buffers, latches, and caches] is a 6" square.

The M120-5 system has a small plug-in cpu card with a R2000, R2010, 64KB of instruction cache, 64KB of data cache, the 4-deep R2020 write buffer, and a fast memory system interface, all running at 16.7MHz.

MIPS R2010 FPA Chip

The R2010 has the following cycle counts:

Operation	Single Precision	Double Precision
Add, Subtract	2 cycles	2 cycles
Multiply	4 cycles	5 cycles
Divide	12 cycles	19 cycles

This yields 250ns for an Add @ 8 MHz, 160ns @ 12.5MHz or 120ns @ 16.7 MHz. In addition, the R2010 overlaps operations extensively, i.e., load and store can be overlapped with arithmetic, while multiply, divide, and add are mostly independent.

MIPS offers 3 levels of floating point: kernel software emulation, R2360 board, and R2010 chip, *which all use identical object code, i.e., no compiler options or differing libraries.* R2010 performance is typically 2-3X that with the R2360 board, just the opposite of many micro systems, whose single-chip FPAs are much slower than board-level designs.

Peak Performance Numbers

We don't believe these mean much, but people ask. Note that VAX-Relative and Peak mips are not the same!

System	Rated VAX-Mips	Peak (Burst) Mips	Peak DP MegaFlops
M/500	5	8.0	4.0
M/800	8	12.5	6.25
M/1000	10	15.0	7.5
M/120-5	12	16.7	8.3

How to Interpret the Numbers

Times (or rates, such as for Dhrystones, Whetstones, and LINPACK KFlops) are shown for the VAX 11/780. Other machines' times or rates are shown, and their relative performance ("Rel." column) normalized to the 11/780 treated as 1.0. VAX/VMS is used whenever possible as the base.

Compilers and Operating Systems

Unless otherwise specified, The M-series benchmark numbers use Release 1.21 of the MIPS compilers and UMIPS-BSD 2.1. The latter is 4.3BSD UNIX port, with NFS, compiled at optimization level -O2, with a modest amount of tuning.

UMIPS-V 3.0 is a System V, Release 3.0 port, with TCP/IP, NFS, a Berkeley Fast File System, and other Berkeley features. Most user-level programs run at about the same speed on UMIPS-BSD and UMIPS-V.

Optimization Levels

Unless otherwise specified, all benchmarks were compiled -O, i.e., with optimization. UMIPS compilers call this level -O2, and it includes global intra-procedural optimization. In a few cases, we show numbers for -O3 and -O4 optimization levels, which do inter-procedural register allocation and procedure merging. -O3 is now generally available.

Simulation Process

Although we no longer cite simulated numbers, a few notes on our process are worthwhile, because simulation and performance analysis are very important to us.

MIPS has developed a sophisticated simulation environment to help predict CPU Board performance in compute-bound applications. While simulations can never predict the exact performance that will eventually be measured on actual systems, our simulations usually come within a few percent, as can be seen by comparisons of simulations (in earlier *Briefs*) and currently available actual results.

The simulation process begins by using our optimizing compiler system to generate MIPS object code. This code is run through *PIXIE*, which instruments the code with counters. Executing the code yields cycle count information and a memory reference trace. The reference trace is then run through an extensive cache/TLB simulator giving a degradation factor which is applied to the cycle count. Time (in seconds) for the benchmark results from converting the degraded cycle count into elapsed time based on the CPU clock frequency.

We often use these detailed results to understand odd program behavior, to analyze the realism of a benchmark (see Dhrystone, for example), to tune our software, and to design new systems.

Now, let's look at the benchmarks. Each section title includes the (CODE NAME) that relates it back to the earlier Summary, if it is included there.

4. Integer Benchmarks

4.1. MIPS UNIX Benchmarks (MIPS UNIX)

The MIPS UNIX Benchmarks described below are fairly typical of nontrivial UNIX programs. This benchmark suite provides the opportunity to execute the same code across several different machines, in contrast to the compilers and linkers for each machine, which have substantially different code. *User time is shown*; kernel time is typically 10-15% of the user time (on the 780), so these are good indications of integer/character compute-intensive programs. The first 3 benchmarks were running too fast to be meaningful on our faster machines, so we modified the input files to get larger times. The grep and nroff benchmarks are still too fast for accurate benchmarking. The VAX 8600 ran consistently around 3.8X faster than the 11/780 on these tests, but we sold it, so it has dropped out as we've changed benchmarks. These benchmarks contain UNIX source code, and are thus not generally distributable.

For better statistical properties, we now report the Geometric Mean of the Relative performance numbers, because it does *not* ignore the performance contributions of the shorter benchmarks.

Note: the Geometric Mean of N numbers is the Nth root of the product of those numbers. It is necessarily used in place of the arithmetic mean when computing the mean of performance ratios, or of benchmarks whose runtimes are quite different. See [Fleming 86] for a detailed discussion.

MIPS UNIX Benchmarks Results

grep		diff		yacc		nroff		Geom	System
Secs	**Rel.**	**Secs**	**Rel.**	**Secs**	**Rel.**	**Secs**	**Rel.**	**Mean**	
11.2	1.0	246.4	1.0	101.1	1.0	18.8	1.0	1.0	11/780 4.3BSD
5.6	2.0	105.3	2.3	48.1	2.1	9.0	2.1	2.1	Sun-3/160M
2.4	4.7	35.8	6.9	19.5	5.2	3.3	5.7	**5.5**	**MIPS M/500**
1.6	7.0	25.1	9.7	11.8	8.6	2.2	8.6	8.4	Sun-4/200 -O3
1.6	7.0	21.6	11.4	11.2	9.0	1.9	9.9	**9.2**	**MIPS M/800**
1.3	8.6	18.0	13.7	9.3	10.9	1.5	12.5	**11.3**	**MIPS M/1000**
1.1	10.2	15.7	15.7	8.0	12.6	1.3	14.5	**13.1**	**MIPS M/120-5**

* These numbers derived as shown on next page.

Benchmark Descriptions

grep Regular expression pattern matcher.
grep DATAN grepinput >grepoutput, where grepinput is the spice2G6 source, 575,500 bytes, 18,323 lines, and DATAN occurs 3 times.

diff File comparison.
diff f1 f2, where f1 is the spice2G6 source, and f2 is the same, but with one line changed and 2 others switched.

yacc "yet another compiler compiler". General purpose parser generator.
yacc yaccinput, where yaccinput is a 1372-line ADA grammar (due to Fischer, Fisher, and Charles). (This requires BSD-sized table limits).

nroff UNIX document processor.
nroff file, where file is 700 lines long.

Note: in order to assure "apples-to-apples" comparisons, we moved the *same* copies of the (4.2BSD) sources for these to the various machines, compiled them there, and ran them, to avoid surprises from different binary versions of commands resident on these machines. Thus, this method tests the performance of compiler code generation, libraries, and CPU, not of algorithmic tunings that may have uncontrollably appeared in the delivered releases. For example, *grep* has many variants.

Note that the granularity here is at the edge of UNIX timing, i.e., tenths of seconds make differences, especially on the faster machines. *The performance ratios seen here seem typical of large UNIX commands on MIPS systems.*

Finally, note that this benchmark set is running versus 4.3BSD, *not* versus Ultrix 2.0 with *vcc*. Hence, the relative performance numbers are inflated somewhat relative to VAX/VMS or VAX-Ultrix numbers. From other experience, we'd guess that subtracting 10% from most of the computed mips-ratings would give a good estimate of the Ultrix 2.0 (vcc)-relative mips-ratings.

4.2. Dhrystone (DHRY 1.1)

Dhrystone is a synthetic programming benchmark that measures processor and compiler efficiency in executing a "typical" benchmark. The "typical" benchmark was constructed by Reinhold P. Weicker from measured statistics of "real" programs [Weicker 84]. Dhrystone does not use floating point, I/O, or operating system calls, and contains little code that could be optimized by vector processors.

The original Dhrystone benchmark, written in Ada, was re-written in C and posted to Usenet by Rick Richardson [Richardson 86]. The Dhrystone results shown below are measured in Dhrystones / second, using the 1.1 version of the benchmark.

We include Dhrystone because it is popular. MIPS systems do extremely well on it. *However, comparisons of systems based on Dhrystone and especially, only on Dhrystone, are unreliable and should be avoided.* More details are given at the end of this section.

Please be careful to differentiate between results based on the 1.1 and 1.0 versions of Dhrystone. According to [Richardson 87], 1.1 cleans up a bug, and is the correct version to use, even though results for a given machine are typically about 15% less for 1.1 than with 1.0. Dhrystone 2.0 now exists, and we will report its results in future briefs.

Advice for running Dhrystone has changed over time with regard to optimization. It used to ask that people turn off optimizers that were more than peephole optimizers, because the benchmark contained a modest amount of "dead" code that optimizers were eliminating. (This is one of the things that Dhrystone 2.0 attempts to fix.) However, it turned out that many people were submitting optimized results, often unlabeled, confusing everyone. Currently, any numbers can be submitted, as long as they're appropriately labeled, except that procedure inlining (done by only a few very advanced compilers) must be avoided.

We continue to include a range of numbers to show the difference optimization technology makes on this particular benchmark, and to provide a range for comparison when others' cited Dhrystone figures are not clearly defined by optimization levels. For example, -O3 does interprocedural register allocation, and -O4 does procedure inlining, and we know -O4 is beyond the spirit of the benchmark. Hence, we now cite the -O3 numbers. We're not sure what the Sun-4's -O3 level does, but we do not believe that it does inlining either.

In the table below, it is interesting to compare the performance of the two Ultrix compilers. Also, examination of the MIPS and Sun-4 numbers shows the performance gained by the high-powered optimizers available on these machines.

The numbers are ordered in what we *think* is the overall integer performance of the processors.

Dhrystone Benchmark Results - Optimization Effects

No Opt		-O		-O3	-O4	
NoReg	Regs	NoReg	Regs	Regs	Regs	
Dhry's /Sec	Dhry's /Sec	Dhry's /Sec	Dhry's /Sec	Dhry's /Sec	Dhry's /Sec	System
1,442	1,474	1,559	1,571			DEC VAX 11/780, 4.3BSD
2,800	3,025	3,030	3,325			Sun-3/160M
4,896	5,130	5,154	5,235			DEC VAX 8600, Ultrix 1.2
8,800	10,200	12,300	12,300	**13,000**	14,200	**MIPS M/500**
8,000	8,000	8,700	8,700			DEC VAX 8550, Ultrix 2.0 cc
9,600	9,600	9,600	9,700			DEC VAX 8550, Ultrix 2.0 vcc
10,550	12,750	17,700	17,700	19,000		Sun-4/200, SunOS 3.2L
12,800	15,300	18,500	18,500	**19,800**	21,300	**MIPS M/800**
15,100	18,300	22,000	22,000	**23,700**	25,000	**MIPS M/1000**
18,700	21,500	25,800	25,800	**27,400**	29,200	**MIPS M/120-5**

Some other published numbers of interest include the following, all of which are taken from [Richardson 87], unless otherwise noted. Items marked * are those that we know (or have good reason to believe) use optimizing compilers. These are the "register" versions of the numbers, i.e., the highest ones reported by people.

Dhrystone Benchmark Results

Dhry's /Sec	Rel.	System
1571	0.9	VAX 11/780, 4.3BSD [in-house]
1757	1.0	VAX 11/780, VAX/VMS 4.2 [Intergraph 86]*
3325	1.9	Sun-3/160, SunOS 3.2 [in-house]
3856	2.2	Pyramid 98X, OSx 3.1, CLE 3.2.0
4433	2.5	MASSCOMP MC-5700, 16.7MHz 68020, RTU 3.1*
4716	2.7	Celerity 1230, 4.2BSD, v3.2
6374	3.6	Sun-3/260, 25MHz 68020, SunOS 3.2
6423	3.7	VAX 8600, 4.3BSD
6440	3.7	IBM 4381-2, UTS V, cc 1.11
6896	3.9	Intergraph InterPro 32C, SYSV R3 3.0.0, Greenhills, -O*
7109	4.0	Apollo DN4000 -O
7142	4.1	Sun-3/200 [Sun 87] *
7249	4.2	Convex C-1 XP 6.0, vc 1.1
7409	4.2	VAX 8600, VAX/VMS in [Intergraph 86]*
7655	4.4	Alliant FX/8 [Multiflow]
8300	4.7	DG MV20000-I and MV15000-20 [Stahlman 87]
8309	4.7	InterPro-32C,30MHz Clipper,Green Hills[Intergraph 86]*
9436	5.4	Convergent Server PC, 20MHz 80386, GreenHills*
9920	5.6	HP 9000/840S [HP 87]
10416	5.9	VAX 8550, VAX/VMS 4.5, cc 2.2*
10787	6.1	VAX 8650, VAX/VMS, [Intergraph 86]*
11215	6.4	HP 9000/840, HP-UX, full optimization*
12639	7.2	HP 9000/825S [HP 87]*
13000	**7.4**	**MIPS M/500, 8MHz R2000, -O3***
13157	7.5	HP 825SRX [Sun 87]*
14195	8.1	Multiflow Trace 7/200 [Multiflow]
14820	8.4	CRAY 1S
15007	8.5	IBM 3081, UTS SVR2.5, cc 1.5
15576	8.9	HP 9000/850S [HP 87]
18530	10.5	CRAY X-MP
19000	10.8	Sun-4/200* [Sun 87]
19800	**11.3**	**MIPS M/800, 12.5MHz R2000, -O3***
23700	**13.5**	**MIPS M/1000, 15MHz R2000, -O3***
27400	**15.6**	**MIPS M/120-5, 16.7MHz R2000, -O3***
28846	16.4	Amdahl 5860, UTS-V, cc1.22
31250	17.8	IBM 3090/200
43668	24.9	Amdahl 5890/300E, cc -O

Numbers in [HP 87] were often lower than those reported for the same machines elsewhere. We suspect HP was being conservative on optimization.

Unusual Dhrystone Attributes

We've calibrated this benchmark against many more realistic ones, and we believe that its results must be treated with care, because the detailed program statistics are unusual in some ways. It has an unusually low number of instructions per function call (35-40 on our machines), where most C programs fall in in the 50-60 range or higher. Stated another way, Dhrystone does more function calls than usual, which especially penalizes the DEC VAX, making this a favored benchmark for inflating one's "VAX-mips" rating. Any machine with a lean function call sequence looks a little better on Dhrystone that it does on others.

The dynamic nesting depth of function calls inside the timed part of Dhrystone is low (3-4). This means that most register-window RISC machines would never even once overflow/underflow their register windows and be required to save/restore registers.

This is not to say fast function calls or register windows are bad (they're not!), merely that this benchmark overstates their performance effects.

Dhrystone can spend 30-40% of the time in the *strcpy* function, copying atypically long (30-character) strings, which happen to be alignable on word boundaries. More realistic programs don't spend this much time in this sort of code, and when they do, they handle more shorter strings: 6 characters would be much more typical.

On our machines, Dhrystone uses 0-offset addressing for 50% of memory data references (dynamic). Most real programs use 0-offsets 10-15% of the time. This, and the previous effect, make some machines look better on Dhrystone than they would on more typical programs.

Of course, Dhrystone is a fairly small benchmark, and thus fits into almost any reasonable instruction cache.

In conclusion, Dhrystone gives some indication of user-level integer performance, but is susceptible to surprises when comparing amongst architectures that differ strongly. Unfortunately, the industry seems to lack a good set of widely-available integer benchmarks that are as representative as are some of the popular floating point ones.

4.3. Stanford Small Integer Benchmarks (STAN INT)

The Computer Systems Laboratory at Stanford University, has collected a set of programs to compare the performance of various systems. These benchmarks are popular in some circles as they are small enough to simulate, and are responsive to compiler optimizations.

It is well known that small benchmarks can be misleading. We would **not** claim that the M/1000 performance is 19 times that of a VAX-11/780 based on these benchmarks. If you see claims that machine X is up to N times a VAX on some (unspecified) benchmarks, these benchmarks are probably the sort they're talking about.

Stanford Small Integer Benchmark Results

Perm	Tower	Queen	Intmm	Puzzle	Quick	Bubble	Tree	Aggr	Rel.	
Secs	Secs	Secs	Secs	Secs	Secs	Secs	Secs	Secs*	Perf+	System
2.34	2.30	.94	1.67	11.23	1.12	1.51	2.72	3.08	.84	VAX 11/780 4.3BSD
								2.60	1.0	VAX 11/780@
.72	1.07	.50	.93	5.53	.58	.97	1.05	1.42	1.8	Sun-3/160M [ours]
.63	.63	.27	.73	2.96	.31	.44	.69	.86	3.0	VAX 8600 Ultrix1.2
.28	.35	.17	.42	2.22	.18	.25	.35	.50	5.2	VAX 8550#
.28	.35	.13	.15	.88	.13	.17	.50	.40	6.5	VAX 8550##
								.65	4.7	Sun-3/200 [Sun 87]
.18	.24	.15	.23	1.15	.17	.19	.34	.36	**7.2**	**MIPS M/500**
.11	.17	.09	.26	.62	.10	.13	.75	.37	7.3	Sun-4/200 -O3
.12	.16	.11	.13	.61	.10	.12	.22	.22	**11.8**	**MIPS M/800**
.10	.13	.10	.11	.51	.08	.10	.17	.18	**14.1**	**MIPS M/1000**
.09	.12	.07	.09	.48	.07	.09	.16	.16	**15.9**	**MIPS M/120-5**

* As weighted by the Stanford Benchmark Suite

\+ Ratios of the Aggregate times

@ Estimated VAX 11/780 Ultrix 2.0 vcc -O time. We get this by 3.08 * (.40+.02)/.50 = 2.60, i.e., using the VAX 8550 numbers to estimate the effect of optimization. The ".02" is a guess that optimization helps the 8550 a little more than it does the 11/780, because the former's cache is big enough to hold the whole program and data, whereas the latter's does not. Another way to put it is that the 8550 is not cache-missing very much, and so optimization pays off more in removing what's left, whereas the 11/780 will cache-miss more, and the nature of these particular tests is that the optimizations won't fix cache-misses. (None of this is very scientific, but it's probably within 10%!)

\# Ultrix 2.0 cc -O

\## Ultrix 2.0 vcc -O. The quick and bubble tests actually had errors; however, the times were in line with expectations (these two optimize well), so we used them. All 8550 numbers thanks to Greg Pavlov (ames!harvard!hscvax!pavlov, of Amherst, NY).

The Sun numbers are from [Sun 87]. The published Sun-4/200 number is .356, for SunOS 3.2L software, i.e., it is slightly faster than the M/500.

Benchmark Descriptions

Perm Computes permutations of 7 elements 5 times. Heavy use of arrays and procedure calls.

Towers Solves Towers of Hanoi for fourteen disks. Heavy use of recursive procedures.

Queen Solves the eight queens problem fifty times. Extensive use of both loops and recursion with backtracking.

Intmm Multiplies two 40x40 integer matrices. Entirely limited by integer multiply time.

Puzzle Forest Baskett's program solves a Soma Cube type problem. Heavy use of small, tight loops.

Quick Performs a quick sort of 5000 elements. Tests recursion and array indexing.

Bubble Reads a file and does a bubble sort of 500 elements. Heavy use of array manipulation.

Tree Performs binary tree sort of 5000 items. Heavy use of pointers, dynamic data structures.

Note that the Stanford Floating Point benchmarks are given in a later section.

5. Floating Point Benchmarks

5.1. Livermore Fortran Kernels (LLNL DP)

Lawrence Livermore National Labs' workload is dominated by large scientific calculations that are largely vectorizable. The workload is primarily served by expensive supercomputers. This benchmark was designed for evaluation of such machines, although it has been run on a wide variety of hardware, including workstations and PCs [McMahon86].

The Livermore Fortran Kernels are 24 pieces of code abstracted from the applications at Lawrence Livermore Labs. These kernels are embedded in a large, carefully engineered benchmark driver. The driver runs the kernels multiple times on different data sets, checks for correct results, verifies timing accuracy, reports execution rates for all 24 kernels, and summarizes the results with several statistics.

Unlike many other benchmarks, there is no attempt to distill the benchmark results down to a single number. Instead all 24 kernel rates, measured in mflops (million floating point operations per second) are presented individually for three different vector lengths (a total of 72 results). The minimum and maximum rates define the performance range of the hardware. Various statistics of the 24 or 72 rates, such as the harmonic, geometric, and arithmetic means give insight into general behavior. Any one of these statistics might suffice for comparisons of scalar machines, but multiple statistics are necessary for comparisons involving machines with vector or parallel features. These machines have unbalanced, bimodal performance, and a single statistic is insufficient characterization. McMahon asserts:

> "When the computer performance range is very large the net Mflops rate of many Fortran programs and workloads will be in the sub-range between the equi-weighted harmonic and arithmetic means depending on the degree of code parallelism and optimization. More accurate estimates of cpu workload rates depend on assigning appropriate weights for each kernel.

McMahon's analysis goes on to suggest that the harmonic mean corresponds to approximately 40% vectorization, the geometric mean to approximately 70% vectorization, and the arithmetic mean to 90%+ vectorization. These three statistics can be interpreted as different benchmarks that each characterize certain applications. For example, there is fair agreement between the kernels' harmonic mean and Spice performance. LINPACK, on the other hand, is better characterized by the geometric mean.

On the next two pages are shown a summary of results from McMahon's report, followed by the complete M/120-5 results. (Given the volume of data, we've only done this on M/120-5s.)

On the next two charts, note that the M/120-5:

- Strictly outperforms high-end superminis in the list.
- On applications characterized by the harmonic mean, it is faster than the 2 minisupers in vector mode, and is even a healthy fraction of supercomputer performance (40% of a Cray-1!).
- On applications characterized by the geometric mean, it is +-10% of the performance of the minisupers.
- On those characterized by the arithmetic mean, it offers 45-70% of minisuper performance.
- Only for highly-vectorizable code does the M/120-5 fall well behind the minisupers. Falls far behind the minisupers only for extremely vectorizable code.

The complete M/120-5 data shows that MIPS performance is insensitive to vector length. The minimum to maximum variation is also small for this benchmark. Both characteristics are typical of scalar machines with mature compilers. Performance of vector and parallel machines, on the other hand, may span two orders of magnitude on this benchmark, or more, depending on the kernel and the vector length.

64-Bit Livermore FORTRAN Kernels

MegaFlops, L = 167, Sorted by Geometric Mean

Min	Harm. Mean	Geom. Mean	Arith. Mean	Max	Rel.* Geom.	System
.05	.12	.12	.13	.24	.7	VAX 780 w/FPA 4.3BSD f77 [ours]
.06	.16	.17	.18	.28	1.0	VAX 780 w/FPA VMS 4.1
.11	.30	.33	.37	.87	1.9	SUN 3/160 w/FPA
.20	**.42**	**.46**	**.50**	**1.42**	**2.5**	**MIPS M/500**
.17	.43	.48	.53	1.13	2.8	SUN 3/260 w/FPA [our numbers]
.29	.58	.64	.70	1.21	3.8	Alliant FX/1 FX 2.0.2 Scalar
.38	.72	.77	.83	1.57	4.5	SUN 4/200 w/FPA [Hough 87]
.39	.94	1.00	1.04	1.64	5.9	VAX 8700 w/FPA VMS 4.1
.10	.76	1.06	1.50	5.23	6.2	Alliant FX/1 FX 2.0.2 Vector
.33	.92	1.06	1.20	2.88	6.2	Convex C-1 F77 V2.1 Scalar
.52	1.09	1.19	1.30	2.74	7.0	ELXSI 6420 EMBOS F77 MP=1
.51	1.26	1.37	1.48	2.70	8.1	MIPS M/800, f771.21
.61	1.51	1.65	1.78	3.24	9.7	MIPS M/1000, f771.21
.65	**1.63**	**1.83**	**2.03**	**3.50**	**10.8**	**MIPS M/1000, f771.30**
.11	1.06	1.94	3.33	12.79	11.4	Convex C-1 F77 V2.1 Vector
.80	**1.84**	**2.06**	**2.27**	**3.89**	**12.1**	**MIPS M/120-5, f771.31**
.28	1.24	2.32	5.11	29.20	13.7	Alliant FX/8 FX 2.0.2 MP=8*Vec
1.51	4.93	5.86	7.00	17.43	34.5	Cray-1S CFT 1.4 scalar
1.23	4.74	6.09	7.67	21.64	35.8	FPS 264 SJE APFTN64
3.43	9.29	10.68	12.15	25.89	62.8	Cray-XMP/1 COS CFT77.12 scalar
0.97	6.47	11.94	22.20	82.05	70.2	Cray-1S CFT 1.4 vector
4.47	11.35	13.08	15.20	45.07	76.9	NEC SX-2 SXOS1.21 F77/SX24 scalar
1.47	12.33	24.84	50.18	188	146	Cray-XMP/1 COS CFT77.12 vector
4.47	19.07	43.94	140	1042	258	NEC SX-2 SXOS1.21 F77/SX24 vector

* Relative Performance, as ratio of the Geometric Mean numbers. This is a simplistic attempt to extract a single figure-of-merit. We admit this goes against the grain of this benchmark.

32-Bit Livermore FORTRAN Kernels

MegaFlops, L = 167, Sorted by Geometric Mean

Min	Harm. Mean	Geom. Mean	Arith. Mean	Max	Rel.* Geom.	System
.05	.18	.20	.23	.48	.7	VAX 780 4.3BSD f77 [ours]
.10	.28	.30	.32	.58	1.0	VAX 780 w/FPA VMS 4.1
.19	.46	.50	.56	1.26	1.7	SUN 3/160 w/FPA
.30	.65	.71	.77	1.55	2.4	SUN 3/260 w/FPA [ours]
.30	.66	.74	.83	1.60	2.5	Alliant FX/1 FX 2.0.2 Scalar
.10	.60	.90	1.31	4.23	3.0	Alliant FX/1 FX 2.0.2 Vector
.40	**.97**	**1.05**	**1.14**	**2.08**	**3.5**	**MIPS M/500**
.55	1.04	1.12	1.20	2.21	3.7	SUN 4/200 w/FPA [Hough 87]
.36	1.11	1.27	1.42	3.61	4.2	Convex C-1 F77 V2.1 Scalar
.46	1.26	1.36	1.45	2.41	4.5	VAX 8700 w/FPA VMS 4.1
.68	1.31	1.46	1.61	3.19	4.9	ELXSI 6420 EMBOS F77 MP=1
.93	2.02	2.19	2.36	3.96	7.3	MIPS M/1000, f771.21
.94	**2.29**	**2.57**	**2.81**	**4.48**	**8.6**	**MIPS M/1000, f771.30**
.28	1.30	2.47	5.59	33.52	8.2	Alliant FX/8 FX 2.0.2 MP=8*Vec
.12	1.27	2.73	5.44	23.60	9.1	Convex C-1 F77 V2.1 Vector
1.05	**2.58**	**2.89**	**3.15**	**5.01**	**9.6**	**MIPS M/120-5, f771.31**

The next table gives the complete M/120-5 output, in the form used by McMahon.

Livermore FORTRAN Kernels - Complete MIPS M/120-5 Output

Vendor	MIPS	MIPS	MIPS	MIPS	MIPS	MIPS	MIPS	MIPS
Model	M/120-5	M/120-5	M/120-5	M/120-5	M/120-5	M/120-5	M/120-5	M/120-5
OSystem	V.3 3.0	V.3 3.0	V.3 3.0	V.3 3.0	V.3 3.0	V.3 3.0	V.3 3.0	V.3 3.0
Compiler	1.31	1.31	1.31	1.31	1.31	1.31	1.31	1.31
OptLevel	O2	O2	O2	O2	O2	O2	O2	O2
Samples	72	24	24	24	72	24	24	24
WordSize	64	64	64	64	32	32	32	32
DO Span	167	19	90	471	167	19	90	471
Year	1988	1988	1988	1988	1988	1988	1988	1988
Kernel	------	------	------	------	------	------	------	------
1	2.8800	2.8800	2.9535	2.9459	3.9122	3.9122	3.9142	3.8487
2	2.2009	2.2009	2.5339	2.5451	3.6809	3.6809	3.6518	2.9828
3	2.8506	2.8506	2.9680	2.9677	4.1781	4.1781	4.0510	3.8582
4	1.8240	1.8240	2.6133	2.9772	3.5978	3.5978	3.1571	2.2205
5	2.0083	2.0083	2.0533	2.0438	3.2797	3.2797	3.2766	3.1695
6	1.3938	1.3938	1.9006	1.9267	3.0934	3.0934	2.8727	1.9807
7	3.8400	3.8400	3.8885	3.8846	5.0121	5.0121	4.9773	4.9192
8	3.5009	3.5009	3.5273	3.5325	4.6659	4.6659	4.6961	4.6136
9	3.5529	3.5529	3.5801	3.5833	4.4751	4.4751	4.4476	4.3809
10	1.4000	1.4000	1.4017	1.4071	2.8119	2.8119	2.8116	2.8000
11	1.4250	1.4250	1.4749	1.4808	2.7811	2.7811	2.6947	2.4806
12	1.4410	1.4410	1.4760	1.5000	2.7827	2.7827	2.7007	2.6127
13	0.8034	0.8034	0.8515	0.8684	1.0682	1.0682	1.0604	1.0510
14	1.3824	1.3824	1.3555	0.9176	1.5660	1.5660	1.9144	1.8807
15	1.0735	1.0735	1.0452	1.0476	1.4139	1.4139	1.4085	1.4494
16	1.5332	1.5332	1.4803	1.5143	1.6219	1.6219	1.6097	1.6585
17	2.6562	2.6562	2.5389	2.5452	3.2781	3.2781	3.2841	3.4185
18	3.2112	3.2112	3.1598	3.1598	4.5896	4.5896	4.6200	4.4800
19	2.8224	2.8224	2.9124	2.8772	3.5246	3.5246	3.5479	3.4005
20	3.3757	3.3757	3.3471	2.6667	4.5008	4.5008	4.5147	4.5298
21	2.0880	2.0880	2.2436	2.2955	3.5501	3.5501	3.4517	3.1764
22	1.5131	1.5131	1.5228	1.5196	2.1875	2.1875	2.1782	2.1454
23	3.1670	3.1670	3.3730	3.3600	4.3913	4.3913	4.4064	4.1833
24	0.9238	0.9238	0.9283	0.9396	1.0874	1.0874	1.1057	1.0631
----------	------	------	------	------	------	------	------	------
Standard Dev.	0.9189	0.9157	0.9180	0.9208	1.1609	1.1514	1.1537	1.1741
Median Dev.	1.1275	1.1205	1.0009	1.0429	1.3376	1.4080	1.3476	1.2294
Maximum Rate	3.8885*	3.8400	3.8885	3.8846	5.0121*	4.9192	4.9773	5.0121
Average Rate	2.2670*	2.2028	2.2971	2.2711	3.1465*	3.0127	3.1814	3.2104
Geometric Mean	2.0628*	2.0030	2.0943	2.0608	2.8840*	2.7590	2.9219	2.9371
Median Rate	2.2222	2.0481	2.3887	2.4203	3.2766	3.0762	3.2804	3.4022
Harmonic Mean	1.8545*	1.8070	1.8856	1.8420	2.5773*	2.4784	2.6135	2.6092
Minimum Rate	0.8034*	0.8034	0.8515	0.8684	1.0510*	1.0510	1.0604	1.0682
Maximum Ratio	1.0000	0.9875	1.0000	0.9989	1.0000	0.9814	0.9930	1.0000
Average Ratio	1.0000	0.9716	1.0132	1.0018	1.0000	0.9574	1.0110	1.0203
Geometric Ratio	1.0000	0.9710	1.0152	0.9990	1.0000	0.9566	1.0131	1.0184
Harmonic Mean	1.0000	0.9743	1.0167	0.9932	1.0000	0.9616	1.0140	1.0123
Minimum Rate	1.0000	1.0000	1.0598	1.0809	1.0000	1.0000	1.0089	1.0163

* These are the numbers brought forward into the summary section.

5.2. LINPACK (LNPK DP and LNPK SP)

The LINPACK benchmark has become one of the most widely used single benchmarks to predict relative performance in scientific and engineering environments. The benchmark is considered a good one because it measures the overall performance of the hardware and compiler in a straight forward manner, on a computation typical of some kinds of work. LINPACK is a linear equations package that particularly emphasizes floating point addition and multiplication. The usual LINPACK benchmark measures the time required to solve a 100x100 system of linear equations using the LINPACK package. LINPACK results are measured in MFlops, millions of floating point operations per second. All numbers are from [Dongarra 87], unless otherwise noted.

The LINPACK benchmark is so large that the data does not entirely fit into the data cache of most CPUs, including ours. Consequently, for many cached computer systems, the LINPACK benchmark is as much a memory system benchmark as it is a floating point benchmark. Today's caches, however, are on the brink of being able to hold the entire working data of the benchmark. This should be taken into account when comparing results between machines with different cache sizes.

The effect of caching is readily apparent even on two slightly different runs on the same machine. The LINPACK benchmark operates on 100x100 submatrix of both a 201x200 matrix and a 200x200 matrix. Usually the two results are close and the higher is used. However, for the M/120-5 the FORTRAN results are 1.7 and 2.1 mflops for leading dimension 200 and 201 respectively. Modifying the benchmark to use a 202x200 array gives 2.3 mflops. Other array sizes give still other results.

Why should this sensitivity exist? Using a submatrix of a larger matrix spreads out the data so that it is no longer sequential. After 100 doublewords of data, 101 or 100 doublewords are unused. The cache is therefore only half used until the data reaches the end. The way in which the data wraps back around and overlays either the used or unused cache locations determines how much of the cache is used in solving the problem. As it turns out, leading dimensions 200, 201, and 202 result in 56%, 85%, and 97% of a 64Kb data cache being used respectively. Following Dongarra's report, we use the leading dimension 201 results, which is a median result. This effect is not as pronounced on other machines (including the M/500) because their data caches are either much smaller or much larger, or because their floating point is slow enough to dominate memory access. Only the combination of a cache close to the LINPACK problem size and fast floating point reveals the effect.

The LINPACK benchmark is easily vectorized, and thus vector machines perform relatively better here than on benchmarks such as Doduc. However, the 100x100 system is usually not large enough for vector machines to achieve their peak performance.

The LINPACK package calls on a set of general-purpose utility routines called BLAS -- Basic Linear Algebra Subroutines -- to do most of the actual computation. A FORTRAN version of the BLAS is available, and the appropriate routines are included in the benchmark. However, vendors often provide hand-coded versions of the BLAS as a library package. Thus LINPACK results are usually cited in two forms: FORTRAN BLAS and Coded BLAS. The FORTRAN BLAS actually come in two forms as well, depending on whether the loops are 4X unrolled in the FORTRAN source (the usual) or whether the unrolling is undone to facilitate recognition of the loop as a vector instruction. According to the ground rules of the benchmark, either may be used when citing FORTRAN BLAS results, although it is typical to note rolled loops with the annotation "(Rolled BLAS)."

For our own numbers, we've corrected a few to follow Dongarra more closely than we have in the past. LINPACK output produces quite a few MFlops numbers, and we've tended to use the fourth one in each group, which uses more iterations, and thus is more immune to clock randomness. Dongarra uses the highest MFlops number that appears, then rounds to two digits.

Note that relative ordering even within families is not particularly consistent, illustrating the extreme sensitivity of these benchmarks to memory system design.

100x100 LINPACK Results - FORTRAN and Coded BLAS

From [Dongarra 87], Unless Noted Otherwise

DP Fortran	DP Coded	SP Fortran	SP Coded	System
.10	.10	.11	.11	Sun-3/160, 16.7MHz (Rolled BLAS)+
.11	.11	.13	.11	Sun-3/260,25MHz 68020+20MHz 68881 (Rolled BLAS)+
.14	-	-	-	Apollo DN4000, 25MHz (68020 + 68881) [ENEWS 87]
.14	-	.24	-	VAX 11/780, 4.3BSD, LLL Fortran [ours]
.14	.17	.25	.34	VAX 11/780, VAX/VMS
.20	-	.24	-	80386+80387, 20MHz, 64K cache, GreenHills
.20	.23	.40	.51	VAX 11/785, VAX/VMS
.29	.49	.45	.69	Intergraph IP-32C,30Mz Clipper[Intergraph 86]
.30	-	-	-	IBM RT PC, optional FPA [IBM 87]
.33	-	.57	-	OPUS 300PM, Greenhills, 30MHz Clipper
.36	.59	.51	.72	Celerity C1230, 4.2BSD f77
.38	-	.67	-	80386+Weitek 1167,20MHz,64K cache, GreenHills
.41	.41	.62	.62	Sun-3/160, Weitek FPA (Rolled BLAS)+
.45	.54	.60	.74	HP9000 Model 840S [HP 87]
.46	.46	.86	.86	Sun-3/260, Weitek FPA (Rolled BLAS)+
.47	.81	.69	1.30	Gould PN9000
.49	.66	.84	1.20	VAX 8600, VAX/VMS 4.5
.49	.54	.62	.68	HP 9000/825S [HP 87]
.57	.72	.86	.87	HP9000 Model 850S [HP 87]
.60	.72	.93	1.2	MIPS M/500, f771.21
.61	-	.84	-	DG MV20000-I, MV15000-20 [Stahlman, 87]
.65	.76	.80	.96	VAX 8500, VAX/VMS
.70	.96	1.3	1.9	VAX 8650, VAX/VMS
.78	-	1.1	-	IBM 9370-90, VS FORT 1.3.0
.97	1.1	1.4	1.7	VAX 8550/8700/8800, VAX/VMS
1.0	1.3	1.9	3.6	MIPS M/800, f771.21
1.1	1.1	1.6	1.6	SUN 4/200 (Rolled BLAS)+
1.2	1.7	1.3	1.6	ELXSI 6420
1.2	1.6*	2.3*	4.3	MIPS M/1000, f771.21
1.5	**1.5**	**3.5**	**4.3**	**MIPS M/1000, f771.30**
1.6	2.0	1.6	2.0	Alliant FX-1 (1 CE)
2.1	-	2.4	-	IBM 3081K H enhanced opt=3
2.1	**2.2**	**4.0**	**4.8**	**MIPS M/120-5, f771.31**
3.0	3.3	4.3	4.9	CONVEX C-1/XP, Fort 2.0 (Rolled BLAS)
6.0	-	-	-	Multiflow Trace 7/200 Fortran 1.4 (Rolled BLAS)
7.0	11.0	7.6	9.8	Alliant FX-8, 8 CEs, FX Fortran, v2.0.1.9
12	23	n.a.	n.a.	CRAY 1S CFT (Rolled BLAS)
39	57	n.a.	n.a.	CRAY X-MP CFT (Rolled BLAS)
43	-	44	-	NEC SX-2, Fortran 77/SX (Rolled BLAS)

+ The Sun FORTRAN Rolled BLAS code appears to be optimal, so we used the same numbers for Coded BLAS. The 4X unrolled numbers for Sun-4/200 are .86 (DP) and 1.25 (SP) [Hough 87].

* These numbers are as reported by Dongarra. We prefer the typical results, which are slightly lower, viz. 1.2, 1.5, 2.2, and 4.3.

100x100 LINPACK Results - FORTRAN and Coded BLAS

VAX/VMS Relative Performance

For A Subset of the Systems

Rel. DP Fortran	Rel. DP Coded	Rel. SP Fortran	Rel. SP Coded	System
.8	.6	.5	.3	Sun-3/260,25MHz 68020+20MHz 68881 (Rolled)
1.0	1.0	1.0	1.0	VAX 11/780, VAX/VMS
1.4	-	1.0	-	80386+80387, 20MHz, 64K cache, GreenHills
2.0	2.9	1.8	2.0	Intergraph IP-32C,30Mz Clipper[Intergraph 86]
2.7	-	2.7	-	80386+Weitek 1167,20MHz,64K cache, GreenHills
2.9	2.4	2.5	1.8	Sun-3/160, Weitek FPA (Rolled BLAS)
3.3	2.7	3.4	2.5	Sun-3/260, Weitek FPA (Rolled BLAS)
3.5	3.9	3.4	3.5	VAX 8600, VAX/VMS 4.5
4.1	4.2	3.4	2.6	HP9000 Model 850S [HP 87]
4.3	**4.2**	**3.7**	**3.5**	**MIPS M/500, f771.21**
6.9	6.6	5.6	5.0	VAX 8550/8700/8800, VAX/VMS
7.1	**7.6**	**7.6**	**10.6**	**MIPS M/800, f771.21**
7.9	6.5	6.4	4.7	SUN 4/200 (Rolled BLAS)
8.6	9.0	9.2	12.6	MIPS M/1000, f771.21
10.4	**9.0**	**14.0**	**12.6**	**MIPS M/1000, f771.30**
11.4	11.8	6.4	5.9	Alliant FX-1 (1 CE)
15.0	**12.9**	**16.0**	**14.1**	**MIPS M/120-5, f771.31**
21.4	19.4	17.2	14.4	CONVEX C-1/XP, Fort 2.0 (Rolled BLAS)
50	65	30	28.8	Alliant FX-8, 8 CEs, FX Fortran, v2.0.1.9
307	-	176	-	NEC SX-2, Fortran 77/SX (Rolled BLAS)

5.3. Spice Benchmarks (SPCE 2G6)

Spice [UCB 87] is a general-purpose circuit simulator written at U.C. Berkeley. Spice and its derivatives are widely used in the semiconductor industry. It is a valuable benchmark because it shares many characteristics with other real-world programs that are not represented in popular small benchmarks. It uses both integer and floating-point computation heavily. The floating-point calculations are not vector oriented, as in LINPACK. Also, the program itself is very large and therefore tests both instruction and data cache performance.

We have chosen to benchmark Spice version 2g.6 because of its general availability. This is one of the later and more popular Fortran versions of Spice distributed by Berkeley. We felt that the circuits distributed with the Berkeley distribution for testing and benchmarking were not sufficiently large and modern to serve as benchmarks. In previous version of this brief, we presented results on circuits we felt were representative, but which contained proprietary data. This time, we gathered and produced appropriate benchmark circuits that can be distributed, and have since been posted as public domain on *Usenet*. The Spice group at Berkeley found these circuits to be up-to-date and good candidates for Spice benchmarking. In the table below, "Geom Mean" is the geometric mean of the 3 "Rel." columns.

Spice2G6 Benchmarks Results

digsr		bipole		comparator		Geom	
Secs	Rel.	Secs	Rel.	Secs	Rel.	Mean	System
1354.0	0.60	439.6	0.68	460.3	0.63	.6	VAX 11/780 4.3BSD, f77 V2.0
993.5	0.81	394.3	0.76	366.9	0.80	.8	Microvax-II Ultrix 1.1, fortrel
901.9	0.90	285.1	1.0	328.6	0.89	.9	SUN 3/160 SunOS 3.2 f77 -O -f68881
848.0	0.95	312.6	0.96	302.9	0.96	1.0	VAX 11/780 4.3BSD, fortrel -opt
808.1	1.0	299.1	1.0	291.7	1.0	1.0	VAX 11/780 VMS 4.4 /optimize
744.8	1.1	221.7	1.3	266.0	1.1	1.2	SUN 3/260 SunOS 3.2 f77 -O -f68881
506.5	1.6	170.0	1.8	189.1	1.5	1.6	SUN 3/160 SunOS 3.2 f77 -O -ffpa
361.2	2.2	112.0	2.7	129.4	2.3	2.4	SUN 3/260 SunOS 3.2 f77 -O -ffpa
296.5	**2.7**	**73.4**	**4.1**	**83.0**	**3.5**	**3.4**	**MIPS M/500**
225.9	3.6	63.7	4.7	73.4	4.0	4.1	SUN 4/200 f77 -O3 -Qoption as -Ff0[+]
-	-	-	-	-	-	5.3	VAX 8700 (estimate)
136.5	**5.9**	**42.6**	**7.0**	**41.4**	**7.0**	**6.6**	**MIPS M/800**
125.5	6.4	39.5	7.6	39.3	7.4	7.1	AMDAHL 470V7 VMSP FORTVS4.1
114.3	**7.1**	**35.4**	**8.4**	**34.5**	**8.5**	**8.0**	**MIPS M/1000**
92.4	**8.7**	**28.5**	**10.5**	**29.7**	**9.8**	**9.7**	**MIPS M/120-5**
48.0	16.8	12.5	23.9	17.5	16.7	18.9	FPS 20/64 VSPICE (2g6 derivative)

\+ Sun numbers are from [Hough 87], who notes that the Sun-4 number was beta software, and that a few modules did not optimize.

Benchmark descriptions:

digsr CMOS 9 bit Dynamic shift register with parallel load capability, i.e., SISO (Serial Input Serial Output) and PISO (Parallel Input Serial Output), widely used in microprocessors. Clock period is 10 ns. Channel length = 2 um, Gate Oxide = 400 Angstrom. Uses MOS LEVEL=2.

bipole Schottky TTL edge-triggered register used as a synchronizer.

comparator
Analog CMOS auto-zeroed comparator, composed of Input, Differential Amplifier and Latch. Input signal is 10 microvolts. Channel Length = 3 um, Gate Oxide = 500 Angstrom. Uses MOS LEVEL=3. Each part is connected by capacitive coupling, which is often used for the offset cancellation. (Sometimes called Toronto, in honor of its source).

Hspice is a commercial version of Spice offered by Meta-Software, which recently published benchmark results for a variety of machines [Meta-software 87]. (Note that the M/800 number cited there was before the UMIPS-BSD 2.1 and f77 1.21 releases, and the numbers have improved). The VAX 8700 Spice number (5.3X) was estimated by using the Hspice numbers below for 8700 and M/800, and the M/800 Spice number:
(5.5: 8700 Hspice) / (6.9: M/800 Hspice) X (6.6: M/800 Spice) yields 5.3X.

This section indicates that the performance ratios seem to hold for at least one important commercial version as well.

Hspice Benchmarks Results

HSPICE-8601K

ST230		
Secs	**Rel.**	**System**
166.5	.6	VAX 11/780, 4.2BSD
92.2	1.0	VAX 11/780 VMS
91.5	1.0	Microvax-II VMS
29.2	3.2	ELXSI 6400
29.1	3.2	Alliant FX/1
25.3	3.6	HyperSPICE (EDGE)
16.8	5.5	VAX 8700 VMS
16.3	5.7	IBM 4381-12
13.4	**6.9**	**MIPS M/800 [ours]**
11.3	**8.2**	**MIPS M/1000 [ours]**
8.7	**10.6**	**MIPS M/120-5 [ours]**
3.27	28.2	IBM 3090
2.71	34.0	CRAY-1S

Again, as in the less-vectorizable Livermore Kernels, the M/120-5 performs about 30% as fast as a CRAY-1S.

5.4. Digital Review (DIG REV)

The Digital Review magazine benchmark [DR 87] is a 3300-line FORTRAN program that includes 33 separate tests, mostly floating-point, some integer. The magazine reports the times for all tests, and summarizes them with the geometric mean seconds shown below. All numbers below are from [DR 87], except the M/500 and M/800 figures. Note that Digital Review gives relative performance using the microVax II as a basis for comparison (mVUPS). For consistency with the rest of this performance brief, we use the VAX 11/780, which significantly affects the ratios.

Digital Review Benchmarks Results

Secs	Rel.	System
9.17	0.7	VAXstation II/GPX, VMS 4.5
6.75	1.0	VAX 11/780, VMS [DEC]
2.90	2.3	VAXstation 3200
2.32	2.9	VAX 8600, VMS 4.5
2.09	3.2	Sun-4/200, SunOS 3.2L
1.86	3.6	MIPS M/500, f771.21 [ours]
1.584	4.2	VAX 8650
1.480	4.6	Alliant FX/8, 1 CE
1.469	4.6	VAX 8700
1.200	5.6	MIPS M/800, f771.21 [ours]
1.193	5.7	ELXSI 6420
.990	6.8	MIPS M/1000, f771.21*
.940	**7.2**	**MIPS M/1000, f771.30** [ours]
.783	**8.6**	**MIPS M/120-5** [ours]
0.487	18.8	Convex C-1 XP

* The actual run number was .99, which [DR 87] reported as 1.00.

5.5. Doduc Benchmark (DDUC)

This benchmark [Doduc 86] is a 5300-line FORTRAN program that simulates aspects of nuclear reactors, has little vectorizable code, and is thought to be representative of Monte-Carlo simulations. It uses mostly double precision floating point, and is often viewed as a "nasty" benchmark, i.e., it breaks things, and makes machines underperform their usual VAX-mips ratings.

This simulation iterates until certain conditions are met. The number of bits in the floating point format, the rounding algorithm, and the accuracy of math libraries on different machines will all affect the number of iterations. More accurate machines do not necessarily require fewer iterations. Therefore runtimes on this benchmark only grossly reflect machine performance, and only gross comparisons are possible.

Performance is given as a number R normalized to 100 for an IBM 370/168-3 or 170 for an IBM 3033-U, [R = 48671/(cpu time in seconds)], so that larger R's are better.

In order of increasing performance, following are numbers for various machines. All are from [Doduc 87] unless otherwise specified.

Double Precision Doduc Benchmark Results

DoDuc R Factor	Relative Perf.	System
17	0.7	Sun-3/110, 16.7MHz
19	0.7	Intel 80386+80387, 16MHz, iRMX
22	0.8	Sun-3/260, 25MHz 68020, 20MHz 68881
26	1.0	VAX 11/780, VMS
33	1.3	Fairchild Clipper, 30MHz, Green Hills
43	1.7	Sun-3/260, 25MHz, Weitek FPA
48	1.8	Celerity C1260
50	1.9	CCI Power 6/32
53	2.0	Edge 1
64	2.5	Harris HCX-7
85	3.3	Alliant FX/1
88	3.4	MIPS M/500, f771.21 -O2, runs 553 seconds
90	3.5	IBM 4381-2
90	3.5	Sun-4/200 [Hough 1987], SunOS 3.2L, runs 540 seconds
91	3.5	DEC VAX 8600, VAX/VMS
97	3.7	ELXSI 6400
99	3.8	DG MV/20000
100	3.8	MIPS M/500, f771.21 -O3, runs 488 seconds
101	3.9	Alliant FX/8
113	4.3	FPSystems 164
119	4.6	Gould 32/8750
129	5.0	DEC VAX 8650
136	5.2	DEC VAX 8700, VAX/VMS
150	5.7	Amdahl 470 V8, VM/UTS
178	6.8	MIPS M/800, f77 -O2, runs 273 secs
181	7.0	IBM 3081-G, F4H ext, opt=2
190	7.3	MIPS M/800, f771.21 -O3, runs 256 secs
218	8.4	MIPS M/1000, f771.21 -O2, runs 223 secs
229	8.8	MIPS M/1000, f771.21 -O3, runs 213 secs
236	9.1	IBM 3081-K
259	**10.0**	**M120-5, UMIPS V.3 3.0, f77 1.31 -O2**
295	**11.3**	**M120-5, UMIPS V.3 3.0, f77 1.31 -O3**
475	18.3	Amdahl 5860
714	27.5	IBM 3090-200, scalar mode
1080	41.6	Cray X/MP [for perspective: we have a long way to go yet!]

As can be seen here, the MIPS systems act like large superminis, leaving a large gap between them and even quite competent microprocessor implementations.

5.6. Whetstone

Whetstone is a synthetic mix of floating point and integer arithmetic, function calls, array indexing, conditional jumps, and transcendental functions [Curnow 76]. Whetstone has been carefully arranged to defeat vectorizing and many compiler optimizations. It is less memory intensive than LINPACK: caches are actually useful. Thus, it and LINPACK correlate with different kinds of applications.

Whetstone results are measured in KWips, thousands of Whetstone interpreter instructions per second. In this case, some of our numbers actually went down, although compiled code has generally improved. First, the accuracy of several library routines was improved, at a slight cost in performance. Second, on machines this fast, relatively few clock ticks are actually counted, and UNIX timing includes some variance. We've been running many runs and averaging. We've now increased the loop counts from 10 to 1000 to increase the total running time to the point where the variance is reduced. This changed the benchmark slightly. Our experiences show some general uncertainty about the numbers reported by anybody: we've heard that various different source programs are being used.

Whetstone Benchmark Results

DP KWips	DP Rel.	SP Kwips	SP Rel.	System
410	0.5	500	0.4	VAX 11/780, 4.3BSD, f77 [ours]
715	0.9	1,083	0.9	VAX 11/780, LLL compiler [ours]
830	1.0	1,250	1.0	VAX 11/780 VAX/VMS [Intergraph 86]
960	1.2	1,040	0.8	Sun-3/160, 68881 [Intergraph 86]
1,110	1.3	1,670	1.3	VAX 11/785, VAX/VMS [Intergraph 86]
1,230	1.5	1,250	1.0	Sun-3/260, 25MHz 68020, 20MHz 68881
1,400	1.7	1,600	1.3	IBM RT PC, optional FPA [IBM 87]
1,730	2.1	1,860	1.5	Intel 80386+80387, 20MHz, 64K cache, GreenHills
1,740	2.1	2,980	2.4	Intergraph InterPro-32C,30MHz Clipper[Intergraph86]
1,744	2.1	2,170	1.7	Apollo DN4000, 25MHz 68020, 25MHz 68881 [ENEWS 87]
1,860	2.2	2,400	1.9	Sun-3/160, FPA
2,092	2.5	3,115	2.5	HP 9000/840S [HP 87]
2,433	2.9	3,521	2.8	HP 9000/825S [HP 87]
2,590	3.1	4,170	3.3	Intel 80386+Weitek 1167, 20MHz, Green Hills
2,600	3.1	3,400	2.7	Sun-3/260, Weitek FPA [measured elsewhere]
2,670	3.2	4,590	3.7	VAX 8600, VAX/VMS [Intergraph 86]
2,907	3.5	4,202	3.4	HP 9000 Model 850S [HP 87]
3,540	4.3	5,290	4.2	Sun-4/200 (reported secondhand, not confirmed)
3,950	4.8	6,670	5.3	VAX 8700, VAX/VMS, Pascal(?) [McInnis, 1987]
4,000	4.8	6,900	5.5	VAX 8650, VAX/VMS [Intergraph 86]
4,120	5.0	4,930	3.9	Alliant FX/8 (1 CE) [Alliant 86]
4,200	5.1	-	-	Convex C-1 XP [Multiflow]
4,220	**5.1**	**5,430**	**4.3**	**MIPS M/500**
6,930	**8.0**	**8,570**	**6.9**	**MIPS M/800**
7,960	**9.6**	**10,280**	**8.2**	**MIPS M/1000**
9,100	**11.0**	**11,400**	**9.1**	**MIPS M/120-5**
12,605	15	-	-	Multiflow Trace 7/200 [Multiflow]
25,000	30	-	-	IBM 3090-200 [Multiflow]
35,000	42	-	-	Cray X-MP/12

5.7. Stanford Floating Point Benchmarks

The following two benchmarks from the Stanford Benchmark Suite emphasize floating point performance. They exemplify that class of programs that use tight loops and a high proportion of actual FP code, unlike, for example Linpack, which depends as much on the speed of accessing main memory as on FP performance itself. These benchmarks are also quite responsive to good optimizing compiler technology. The ratios here are much higher than usual. First, the code is very susceptible to high-quality optimization. Second, the code uses a high proportion of floating-point code, on which we do well. Third, the comparison is with the 4.3BSD UNIX C compiler, which does not perform global optimization, unlike the LLL-FORTRAN compiler or VAX/VMS FORTRAN. We'd guess that use of VAX/VMS C would improve the VAX 11/780 time by nearly a factor of two (see the Doduc 4.3BSD versus VMS comparison, for example.) This would drop the relative performance ratios back where they belong.

Stanford Floating Point Benchmark Results

FFT Secs	Matrix Mult Secs	Aggregate Secs	Rel. Perf.*	System
3.97	2.31	5.52	1.0	VAX 11/780
3.12	2.14	3.41	1.6	Sun-3/160M#
.57	.30	.72	7.7	Sun-4/200 -O3
.34	**.26**	**.59**	**9.4**	**MIPS M/500**
.20	**.13**	**.37**	**14.9**	**MIPS M/800**
.17	**.12**	**.29**	**19.0**	**MIPS M/1000**
.12	.08	.24	23.0	MIPS M/120-5

* As weighted by the Stanford Benchmark Suite
(includes some contribution from the integer results)

Performance would be substantially improved with the respective FPA boards. Also, recall that the in-house Sun has only a 12.5MHz 68881, so it's slower than the more recent Suns would be.

Benchmark Descriptions

FFT Computes a 256-point Fast Fourier Transform (FFT) twenty times.

Matrix Multiply Multiplies two 40x40 single-precision matrices.

6. Acknowledgements

Some people have noted that they seldom believe the numbers that come from corporations unless accompanied by names of people who take responsibility for the numbers. Many people at MIPS have contributed to this document, which was originally created by Web Augustine. Particular contributors to this issue include Mark Johnson (much Spice work, including creation of public-domain Spice benchmarks), and especially Earl Killian (a great deal of work in various areas, particularly floating-point). *Final responsibility for the numbers in this Brief is taken by the editor, John Mashey.*

We thank David Hough of Sun Microsystems, who kindly supplied numbers for some of the Sun configurations, even fixing a few of our numbers that were incorrectly high, and who has also offered good comments on joint efforts looking for higher-quality benchmarks.

We also thank Cliff Purkiser of Intel, who posted the Intel 80386 Whetstone and LINPACK numbers on Usenet.

We also thank Greg Pavlov, who ran hordes of Stanford and Dhrystone benchmarks for us on a VAX 8550, Ultrix 2.0 system.

7. References

[Alliant 86]
Alliant Computer Systems Corp, "FX/Series Product Summary", October 1986.

[Curnow 76]
Curnow, H. J., and Wichman, B. A., "A Synthetic Benchmark", *Computing Journal*, Vol. 19, No. 1, February 1976, pp. 43-49.

[Doduc 87]
Doduc, N., FORTRAN Central Processor Time Benchmark, Framentec, June 1986, Version 13. Newer numbers were received 03/17/87, and we used them where different.
E-mail: seismo!mcvax!ftcsun3!ndoduc

[Dongarra 87]
Dongarra, J., "Performance of Various Computers Using Standard Linear Equations in a Fortran Environment", Argonne National Laboratory, August 10, 1987.

[Dongarra 87b]
Dongarra, J., Marin, J., Worlton, J., "Computer Benchmarking: paths and pitfalls", IEEE Spectrum, July 1987, 38-43.

[DR 87]
"A New Twist: Vectors in Parallel", June 29, 1987, "The M/1000: VAX 8800 Power for Price of a MicroVAX II", August 24, 1987, and "VAXstation 3200 Benchmarks: CVAX Eclipses MicroVAX II", September 14, 1987. Digital Review, One Park Ave., NY, NY 10016.

[ENEWS 87]
Electronic News, "Apollo Cuts Prices on Low-End Stations", July 6, 1987, p. 16.

[Fleming 86]
Fleming, P.J. and Wallace, J.J.,"How Not to Lie With Statistics: The Correct Way to Summarize Benchmark Results", *Communications of the ACM*, Vol. 29, No. 3, March 1986, 218-221.

[HP 87]
Hewlett Packard, "HP 9000 Series 800 Performance Brief", 5954-9903, 5/87. (A comprehensive 40-page characterization of 825S, 840S, 850S).

[Hough 86,1]
Hough, D., "Weitek 1164/5 Floating Point Accelerators", *Usenet*, January 1986.

[Hough 86,2]
Hough, D., "Benchmarking and the 68020 Cache", *Usenet*, January 1986.

[Hough 86,3]
Hough, D., "Floating-Point Programmer's Guide for the Sun Workstation", Sun Microsystems, September 1986. [an excellent document, including a good set of references on IEEE floating point, especially on micros, and good notes on benchmarking hazards]. Sun-3/260 Spice numbers are from later mail.

[Hough 87]
Hough, D., "Sun-4 Floating-Point Performance", *Usenet*, 08/04/87.

[IBM 87]
IBM, "IBM RT Personal Computer (RT PC) New Models, Features, and Software Overview, February 17, 1987.

[Intergraph 86]
Intergraph Corporation, "Benchmarks for the InterPro 32C", December 1986.

[Meta-Software 87]
Meta-Software, "HSPICE Performance Benchmarks", June 1987. 50 Curtner Avenue, Suite 16, Campbell, CA 95008.

[McInnis 87]
McInnis, D., Kusik, R., Bhandarkar, D., "VAX 8800 System Overview", Proc. IEEE

COMPCON, March 1987, San Francisco, 316-321.

[McMahon 86]
"The Livermore Fortran Kernels: A Computer Test of the Numerical Performance Range", December 1986, Lawrence Livermore National Labs.

[MIPS 87]
MIPS Computer Systems, "A Sun-4 Benchmark Analysis", and "RISC System Benchmark Comparison: Sun-4 vs MIPS", July 23, 1987.

[Purkiser 87]
Purkiser, C., "Whetstone and LINPACK Numbers", *Usenet*, March 1987.

[Richardson 87]
Richardson, R., "9/20/87 Dhrystone Benchmark Results", *Usenet*, Sept. 1987.
Rick publishes the source several times a year. E-mail address: ...!seismo!uunet!pcrat!rick

[Serlin 87a]
Serlin, O., "MIPS, DHRYSTONES, AND OTHER TALES", Reprinted with revisions from SUPERMICRO Newsletter, April 1986, ITOM International, P.O. Box 1450, Los Altos, CA 94023.
Analyses on the perils of simplistic benchmark measures.

[Serlin 87b]
Serlin, O., SUPERMICRO #69, July 31, 1987. pp. 1-2.
Offers good list of attributes customers should demand of vendor benchmarking.

[Stahlman 87]
Stahlman, M., "The Myth of Price/performance", Sanford C. Bernstein & Co, Inc, NY, NY, March 17, 1987.

[Sun 86]
SUN Microsystems, "The SUN-3 Family: A Hardware Overview", August 1986.

[Sun 87]
SUN Microsystems, SUN-4 Product Introduction Material, July 7, 1987.

[UCB 87]
U. C. Berkeley, CAD/IC group, "SPICE2G.6", March 1987. Contact: Cindy Manly, EECS/ERL Industrial Liason Program, 479 Cory Hall, University of California, Berkeley, CA 94720.

[Weicker 84]
Weicker, R. P., "Dhrystone: A Synthetic Systems Programming Benchmark", *Communications of the ACM*, Vol. 27, No. 10, October 1984, pp. 1013-1030.

From: TLE::DECWET::CUTLER "02-Jun-1988 0803" 2-JUN-1988 11:23
To: PIPE::KEATING,TLE::GROVE,TLE::NYLANDER,KISWA::HEINEN,GILBERT,ORBITS
Subj: Yesterday's meeting

Thank you for speaking out yesterday at the summit meeting. I think we managed to get 3 points across:

1. to switch to MIPS would not provide the same family of products with a high degree of VAX compatibility

2. that a switch to MIPS would case at least a year schedule slip

3. engineering as a whole is not for the MIPS proposal

Appanrently this will all be hashed over again at the executive committee meeting. Jack is supposed to let me know the outcome of that meeting today - sometime he slips a day.

I will let you know hwat I know as soon as I know it!

The down side on all this is that Ken wants to do the MIPS proposal, Jack does not. I belive Ken will win and we will be in the MIPS workstation business. We'll see.

d

From: TLE::GROVE 25-MAY-1988 12:55
To: NYLANDER
Subj: MIPS correspondence w Dave 1/3

From: DECWET::CUTLER 25-MAY-1988 11:10
To: TLE::GROVE
CC:
Subj: register allocation

I have a MIPS architecture book that was published by Prentis Hall and describes both the hardware architecture and some of the software architecture.

It appears that they require an assembly phase in all compilations. Did you find this to be true? (real state of the art huh).

There is an explanation of register allocation that looks very funny. There is one register that is dedicated to the assembler. It is register 1. There are also two other registers that are dedicated to the operating system kernel. I think these were 26 and 27. Did you find any evidence to support this?

There is also a dscription of stack frames and register save areas and the such. They do not have frame based condition handling. They do no include the floating point status register as part of the frame context. By the way did you know that the floating point coprocessor is alla the PDP-11/45? The rounding modes are selected by a status register rather than being part of the instructions. Prodcuing a math library would indeed be fun - switching back and forth between rounding modes. And exceptions are interrupts which has the problem of receiving the exception when you are in the wrong context (i.e. just after an interrupt of system call).

I'm not impressed with the architecture.

d

From: TLE::GROVE 25-MAY-1988 12:56
To: NYLANDER
Subj: MIPS correspondence w Dave 2/3

From: TLE::GROVE 25-MAY-1988 11:46
To: DECWET::CUTLER
CC: GROVE
Subj: MIPS assembler and register allocation

Yes, the compiler(s) produce an assembler source file; and the assembly phase is a standard part of the compiler block diagram. They do this for a couple of reasons:

1. It's a UNIX system, and UNIX compilers always produce assembly code.

2. Acccording to the assembler manual, it's a fairly "smart" assembler.

   The assembler has built into it the knowledge of pipeline conflicts and scheduling rules. MIPS doesn't do resource interlocks, so someone has to insert NOPs if a delay is needed before using the result of the previous instruction.

   MIPS also isn't committed to binary compatibility between successive implementations. Perhaps the assembler interface helps to provide a more compatible interface.

   The assembler also has built into it a lot of macro instructions that may in fact expand to several real machine instructions. For example, there is an assembler pseudo-op LA (load address). You can give this an arbitrary address expression or constant and it will produce the code to load the address. That load might be LDL displaced using the global pointer register or it could be loading an address constant from somewhere else.

   So, I would guess R1 is reserved so the assembler has a scratch register for some of its multi-instruction sequences.

   I think the assembler probably does some peephole optimizations as well. The compiled code contained things like:

```
     jmp     1$
1$:
```

3. Apparently they also do code scheduling in the assembler. The assembler has some Reorder/Noreorder directives. It doesn't seem to me the assembler could be expected to do a very good job on scheduling, because I don't think it would have a very good data dependency model. It would be real hard to do a sophisticated (and correct!) job of reordering loads and stores at that level.

   Bob Supnik claimed they could do a sufficiently scheduling job given the particular characteristics of the pipeline. Still with processors and caches getting faster and main memory relatively slower, load scheduling seems important.

Of course, it was a little hard to figure out what the assembler had done because we didn't find a way to get a listing of the result. (UNIX isn't big on listings). So we weren't able to evaluate just what kinds of scheduling transformations they had done. The table of contents for the assembler manual listed an Appendix C on "reordering rules and constraints",

but the appendix was non-existent.

Hope this is of some help. If there is anything else we can do (technically, politically, etc) to help keep the PRISM program moving full speed ahead, please let me know!!!!

Rich

From: TLE::GROVE 25-MAY-1988 09:28
To: KLEIN,NYLANDER,NM%THESPN::DAVIS
Subj: Dileep's analysis of mipsco vs prism

From: DECEAT::BHANDARKAR "24-May-1988 1729" 24-MAY-1988 17:31
To: @DIS$:PRISM,MIST::SHORT
CC:
Subj: prism vs mipsco

+---+---+---+---+---+---+---+
| d | i | g | i | t | a | l |
+---+---+---+---+---+---+---+

Interoffice Memorandum

To: Bill Strecker

Date: 24 May 1988
From: Dileep Bhandarkar
Dept: Mid Range Systems
Technical Director
Ext: 293-5350
Loc: BXB1-1/E11
ENET: DECEAT::BHANDARKAR

cc: Dick Angel, Dave Cutler, Bill Demmer, Sam Fuller, Roger Heinen, Bill Keating, Dom LaCava, Cathy Learoyd, Dom McInnis, Jack Smith, Bob Supnik

Subject: My thoughts on our PRISM Strategy

Over the last 3 years I have had the opportunity to talk to several customers about their computing requirements, especially as they related to the price performance of VAXes. While several customers were very complimentary about our single consistent architecture, they did express interest in other architectures if there was a significant (at least a factor of 2) price/performance advantage. Our PRISM strategy is based on the belief that it will become increasingly difficult for us to compete with just VAXes against RISC based products from HP, Sun, and other system integrators of SPARC, MC88000, or MIPSCO chips. Our customer needs can be served best with an approach based on PRISM rather than commodity RISC chips.

1 THREE TYPES OF CUSTOMERS

Our customer base can be divided into 3 major categories:

1. VAX/VMS zealots who are extremely happy with the development environment under VAX/VMS and have very little interest in anything else. They don't particularly like UNIX, but find the price/performance of RISC machines to be a cause for concern. Such customers often ask, "Can you put VMS on a RISC machine?". To many of these customers, VMS means DCL, DECnet, system services, file formats, and their favorite layered products and utilities. Most of them do not write assembler code, but do share binary files across different machines. These customers are willing to consider companies like Convex that claim to be compatible with VMS Fortran and offer DCL and EDT.

2. UNIX bigots who want Unix because they think that it will make them vendor independent. Ironically, they also want us to support VMS features (e.g. clusters, or a particular set of

VMS layered products) under Ultrix. Many VMS users are considering UNIX because they are afraid to get tied down to a single vendor.

3. Fence sitters who don't have a strong preference but will use some factor such as price/performance, or software functionality to make a decision. If all things are equal, they would probably choose Unix because it gives them a feeling of vendor independence.

2 MEETING THE NEEDS OF CUSTOMERS

The needs of each type of customer are different. We can meet their needs with a set of VAX and PRISM products that provide 2 excellent Digital alternatives.

2.1 VAX Zealots

These customers are obviously happy with VMS. The best solution for them is to continue to build newer faster VAXes. When other machines with better price/performance become available, these customers will consider moving bounded applications to the new machines, especially if the new machines can be integrated relatively seamlessly into their existing environment. Such is the case with many VAX customers who have bought SUN or Apollo workstations for their CAD frontends, and Convex minisupercompouters for their backend compute servers. Loyal VAX customers are likely to feel betrayed when we offer PRISM based workstations that offer twice the performance of their VAX equivalents at the same price. This may lead them to conclude (and rightfully so) that RISC machines have much better price/performance. They will start re-evaluating their commitment to VAX and VMS. Even though they might like VMS, they will examine other alternatives. Ideally, they would prefer a VMS like environment over Unix. However, if the only thing we provide on PRISM is Unix, I would expect that these customers will consider Unix platforms from other vendors too.

We can keep these customers on our PRISM platform by providing VMS functionality under MICA. Obviously, we cannot port all or even most VMS layered products instantaneously. We need a phased strategy for moving layered products in groups to MICA. The members of each group would consist of products that work together (e.g. CMS and MMS) and form the nucleus for the needs of a particular market. Obviously, some basic set should be ported over at FRS (e.g. DCL, TPU, LSE, PCA, etc). Over time, more and more VAX users will migrate to PRISM. Users who still want VAX binary compatibility will continue to buy VAXes.

Our current PRISM plan does not call for a non-Ultrix user interface at FRS. I believe that we are making a big mistake and will cause some number of customers to go to other vendors machines instead of staying

with VAX/VMS or moving to PRISM/Ultrix.

2.2 Unix Bigots

For not entirely rational reasons, a lot of users want Unix. We should give them world class Unix with competitive price/performance as per our current strategy. To make our Unix more attractive that other vendors, we should migrate selected VMS layered products to PRISM/Ultrix. Our current implementation strategy for PRISM/MICA and PRISM/ULTRIX is based on a common software architecture. The VAX data type compatibility of PRISM and similarity of privileged architecture is a major factor that makes the common software architecture possible. The availability of VMS layered products under Ultrix will make our offering superior to other Unix offerings. We can make it easy for people to move from standard Unix to PRISM/ULTRIX and provide them a richer environment.

2.3 Fence Sitters

The fence sitters will have a choice between VAX/VMS with the richest software environment and broadest range of compatible systems at some price premium, PRISM/MICA systems with limited VAX functionality but better price/performance, and PRISM/Ultrix with Unix compliance and proprietary extensions. The three choices can be mixed in a network environment and will offer data type and source code compatibility.

3 PRISM VS MIPSCO

There are several aspects of the PRISM architecture that make it possible to support a VAX/VMS like environment on it. The most basic one is its data type compatibility (integer and floating point). PRISM also has mechanisms for operating on interlocked VAX queues that are used by our I/O adapters. The privileged architecture of PRISM is modeled after VAX. The PRISM vector architecture and VAX vector architecture provide identical features, allowing the sharing of common chips and compilation strategies.

MIPSCO has none of the above. A MIPSCO based strategy would lock us into the standard UNIX software domain and make it exteremely difficult to port existing VMS layered products to it. In addition, MIPSCO is not committed to full binary compatibility among its different chips. MIPSCO does not provide hardware synchronization for resource conflicts between instructions. MIPSCO does have a large number of semiconductor vendors lined up to produce their chips. Our CMOS parts appear to be competitive, but we lack custom bipolar expertise. There are 2 possible solutions to this problem. We can pay an outside semiconductor vendor a premium price (and large non-recurring engineering expenses) for a proprietary part just for our use or we can license semiconductor

manufacturers for bipolar implementations to be sold on the open market. Such an approach would provide us with state of the art bipolar chips that meet our architectural desires. Making our architecture public is much better than embracing someone else's public architecture. I believe that semiconductor vendors would be very receptive to building our chips if we let them sell those chips on the open market.

While a MIPSCO based product line may help bring in First Customer Ship dates by a couple of months, it will serve to defocus our PRISM strategy, contend for scarce resources in the company, confuse our product message in the short term and possible dilute our long-term strategy.

4 CONCLUSIONS

A PRISM based strategy allows us to migrate our installed VAX/VMS customer base in a controlled way to PRISM when they are ready to move. It also allows us to be a leader in the Unix marketplace. The common software architecture allows us to build from our existing VMS layered product set and evolve to two cost effective platforms without duplication of work.

We are basically on the right strategic path with PRISM. Let's exercise some discipline and execute our plan without what now appears to be an annual destabilization. Where there are holes in our current plan, let's strengthen them with additional complementary resources instead of embarking on a redundant parallel effort.