POWER CHALLENGEarray

1

1.1 Introducing *POWER CHALLENGEarray*

Rapidly advancing integrated circuit technology and computer architec-tures are driving microprocessors to performance levels that rival tradition-al supercomputers at a fraction of the price. These advances, combined with sophisticated memory hierarchies, let powerful RISC-based shared-memory multiprocessor machines achieve supercomputing-class perform-ance on a wide range of scientific and engineering applications.

Shared-memory multiprocessing refers to the singularity of the machine address space and gives an intuitive, efficient way for users to write para-llel programs. Shared-memory multiprocessor systems such as *Silicon Graphics*® *POWER CHALLENGE*™ and *POWER Onyx*™ SMPs (Symmetric Multiprocessing) have large memory capacities and high I/O bandwidth, necessities for many supercomputing applications. Further, these support high-speed connect options such as FDDI, HiPPI, and ATM. Such character-istics make shared-memory multiprocessors more powerful than traditional workstations. These systems, based on the same technology, are designed to provide a moderate amount of parallelism at the high end, while scaling down to affordable low-end multiprocessors.

There is a demand for larger (greater number of CPUs) parallel machines to solve problems faster; to solve problems previously only attempted on special-purpose computers; to solve unique and untested problems. This is particularly true for the grand challenge class of problems, spanning areas such as computational fluid dynamics, materials design, weather modeling, medicine, and quantum chemistry.

These problems, amenable to large-scale parallel processing, can exploit computational power offered by para-llel machines having hundreds of processors. A modular approach for buil-ding parallel systems scaleable to hundreds of processors is to connect multiple shared-memory multiprocessors, such as POWER CHALLENGE systems, by a high bandwidth interconnect, such as HiPPI, in an optim-ized topology. In addition to delivering performance as a large-scale para-llel processor, this architecture can also serve as a powerful throughput en-gine for running applications that exploit moderate levels of parallelism. At the same time, it provides a single system image to users and operators.

A Modular Approach to Distributed Parallel Processing

POWER CHALLENGEarray is a distributed parallel processing system that scales up to 144 MIPS R8000 or up to 288 MIPS R10000 microprocessors to serve as a powerful distributed throughput engine in production envir-onments, and to solve grand challenge-class problems in research and pro-duction environments. POWER CHALLENGEarray consists of up to eight POWER CHALLENGE or POWER Onyx (***POWERnode***) Supercomputing systems connected by a high-performance HiPPI interconnect. Using the POWERnode as a building block, POWER CHALLENGEarray exploits the ultra-high-performance *POWERpath-2*™ interconnect to form a low-cost, modular, scaleable system. Using this unique modular approach, POWER CHALLENGEarray creates a highly scalable system, *providing more than 115 GFLOPS of peak performance, up to 128GB of main memory, more than 4GB/sec of sustained disk transfer capacity, and more than 28TB of disk space.*

POWER CHALLENGEarray therefore offers a two-level communication hierarchy, whereas CPUs within a POWERnode communicate via a fast shared-bus interconnect, and CPUs across POWERnodes communicate via a high-bandwidth HiPPI interconnect. Each POWERnode comes with a suite of parallel programming software tools already available on POWER CHALLENGE (for example, the MIPSpro compiler, CHALLENGEcomplib, and ProDev/Workshop Application Development Tools). Additionally, POWER CHALLENGEarray offers other software tools to aid in developing distributed parallel programs, as well as to manage the parallel computer from a single point of control. Figure 1-1 illustrates a four-node POWERnode POWER CHALLENGEarray processing system.

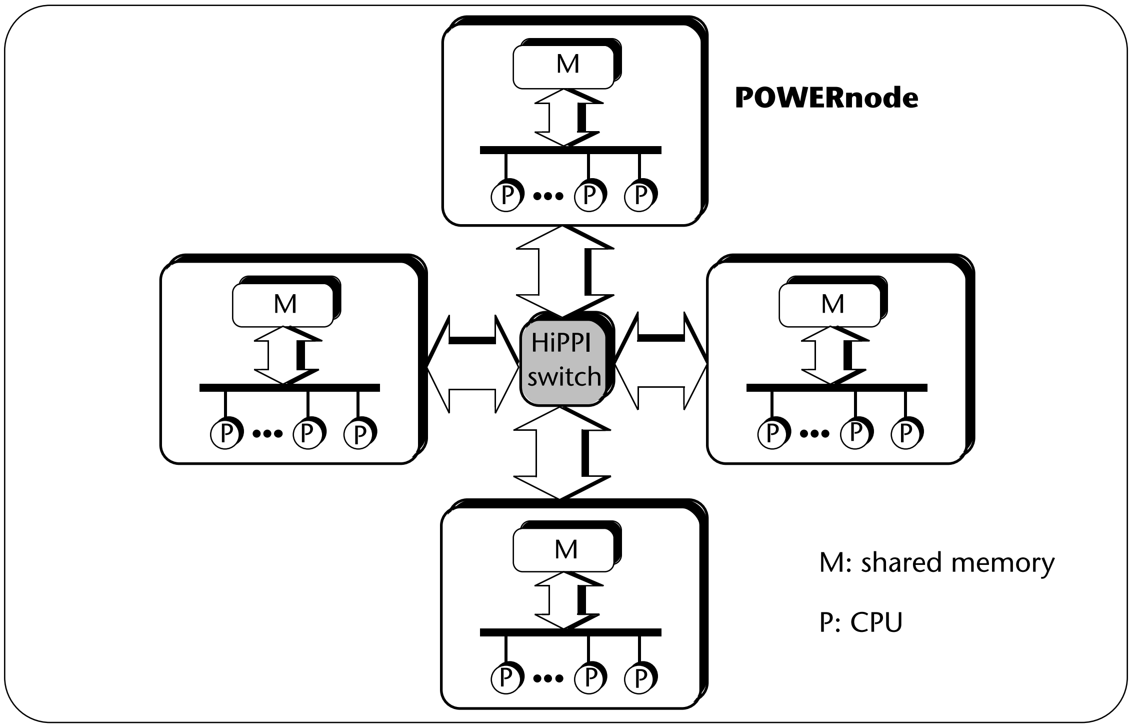


Figure 1-1 Four POWERnode POWER CHALLENGEarray systems

In contrast to traditional distributed-memory private-address-space “shared nothing” architectures, the hierarchical POWER CHALLENGEarray approach offers several advantages:

- ❑ Less work to program
- ❑ Improves computation-to-communication ratio of parallel applications
- ❑ Reduces physical memory requirements of applications
- ❑ Provides better load balancing for irregular problems
- ❑ Improves latency tolerance for coarse-grained, message-level parallelism
- ❑ Offers a flexible parallel programming environment, rich in several types of parallel programming paradigms

- ❑ Provides a gradual learning path involving shared, distributed, and distributed-shared-memory (hybrid) parallel programming for program-mers relatively new to parallel programming

Combining Shared Memory/Message-Passing Techniques

The POWER CHALLENGEarray approach is unique among distributed com-puting models because each individual system comprising the array is itself a parallel supercomputer. POWER CHALLENGEarray combines the effic-iency, flexibility, and ease of programmability features of shared-memory multiprocessing with the upward scalability of message-passing architec-tures. This leads to a better computation-to-communication ratio because messages are sent between systems for larger blocks of parallel processing.

Powerful Shared-Memory POWERnode

Each POWERnode of POWER CHALLENGEarray is a Silicon Graphics POWER CHALLENGE shared-memory multiprocessor system, supporting up to 18 MIPS R8000/90MHz or 36 MIPS R10000 supercomputing micro-processors and providing up to 14.4 GFLOPS of peak performance. An interleaved memory subsystem accepts up to 16GB of main memory, and a high-speed I/O subsystem scales up to four 320MB/second I/O channels. Each POWERnode system also supports a wide range of connectivity options, including HiPPI, FDDI, and Ethernet™.

If the number of tasks required in a parallel program is less than or equal to the number of CPUs on a POWERnode, the shared-memory, single-address-space programming model can be used. This provides an intuitive and efficient way of parallel programming to the user and avoids the over-head of message-passing between processes. Figure 1-2 on page 5 demon-strates this point. For moderately parallel jobs, SMP-level performance can be obtained. Since there are a large number of such jobs in a typical para-llel environment, POWER CHALLENGEarray can be used as an efficient throughput engine for such a job mix. Additionally, users can access the entire set of array tools and resources for POWER CHALLENGEarray.

Applications with large computational, memory, and I/O requirements that cannot be accommodated on individual workstation-class machines (that form the basic building blocks of traditional distributed-memory parallel machines), and that can exploit a moderate level of parallelism are particularly well-suited for POWER CHALLENGEarray.

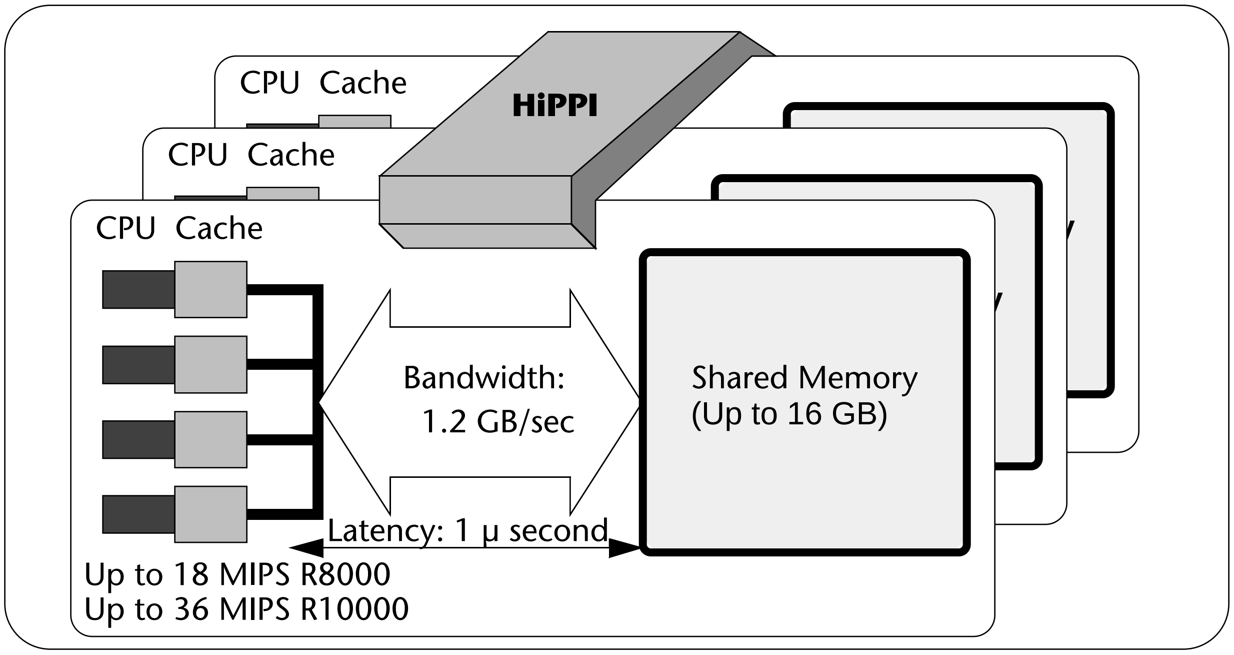


Figure 1-2 Efficiency of Shared-Memory Interprocess Communications

Low Communication-to-Computation Ratio

Applications having scalability or memory requirements beyond the capa-bilities of a single POWERnode can be restructured using hierarchical pro-gramming techniques to span multiple POWERnodes. Under this model, the tasks within a POWERnode still communicate via shared memory and tasks between POWERnodes communicate via message-passing. The message-passing overhead can be optimized, compared to the computation for each message sent, since parallel tasks within a POWERnode can use global shared memory to communicate shared data.

For many applications, domain decomposition results in maximum data locality and data reuse, resulting in reduced intertask communication.

The programmer has the flexibility to use a single message-passing library for communicating within and across POWERnodes. This is because message-passing libraries use low-latency, high-bandwidth shared-memory techniques to communicate data within POWERnodes and high-bandwidth networking protocols to communicate across POWERnodes. Thus, POWER CHALLENGEarray offers the great advantage of low communication-to-computation ratios for many large-scale problems while involving no more work than traditional distributed-memory, private address-space systems.

A Powerful Distributed Throughput Engine

Many applications, particularly in industrial computing environments, can exploit only moderate levels of parallelism: 10-20 CPUs. Amdahl's law, which relates the speedup achievable by a parallel program as a function of its parallelizable portion and the number of processors used, illustrates this point. It states:

$$\boxed{\text{Speedup} = \frac{1}{(1-f) + \left(\frac{f}{\rho}\right)}}$$

Figure 1-3 Amdahl's Law Equation

where *f* is the parallelizable fraction of the total program, and *p* is the number of processors that the program uses. Assuming an infinite number of processors, the speedup becomes *1/(1-f*). Table 1-1 on page 7 plots the value of this maximum achievable speedup for a given parallel fraction, *f*.

% Parallelism (*f*)	Maximum Speedup
50%	2.0
60%	2.5
70%	3.33
80%	5.0
90%	10.0
95%	20.0
96%	25.0
97%	33.0
98%	50.0
99%	100.0

Table 1-1 Amdahl's Law of Parallelism

Most real applications have some amount (in the order of 5 percent or more) of non-parallelizable code. Using more than a moderate number of processors (10-20) for them does not yield additional performance bene-fits. For such an application, running on a single POWERnode with a mod-erate number of processors may be sufficient to realize any potential bene-fit from parallelization. Hence, a typical parallel environment will consist of some large-scale parallel applications mixed with a large number of modestly parallel applications.

For a workload consisting of moderately parallel applications, POWER CHALLENGEarray systems deliver very high throughput. This is because most of these applications can be parallelized using shared-memory tech-niques and run within a POWERnode. Combining this with Silicon Graphics parallelizing tools, the IRIX™ operating systems, and the batch processing tools available, POWER CHALLENGEarray serves as a powerful distributed parallel throughput environment.

High-Performance Graphics Support: Interactive Supercomputing

One or more POWERnodes in POWER CHALLENGEarray can be a POWER Onyx supercomputing graphics system for the ultimate graphics applica-tion. POWER Onyx is the world's most powerful supercomputing graphics system, combining advanced *RealityEngine* graphics of the Onyx architec-ture and floating point performance of the

MIPS R10000 or R8000 CPU. The RealityEngine graphics subsystems offer the highest performance and most advanced features of any computer graphics system, based on a scaleable and expandable graphics architecture containing 1.2 GFLOPS of floating-point processing power dedicated solely to the task of accelerating geo-metric and image processing functions. POWER Onyx scales up to 12 MIPS R8000 processors or 24 MIPS R10000 with 4MB of secondary cache, deliv-ering up to 9.6 GFLOPS of peak performance. It meets wide-ranging needs of users in such diverse fields as computational chemistry, oil and gas, molecular modeling, weather analysis and modeling, structural and fluid dynamics, image processing, animation rendering, and more. This combi-nation of powerful supercomputing and powerful graphics allows users to visualize simulations immediately and interactively steer them.

A Hierarchy of Programming Models

POWER CHALLENGEarray supports fine-grained, medium-grained, and coarse-grained parallelism. It also supports both shared-memory and message-passing programming models, and several hybrid combinations of those two models. Shared-memory programming typically involves the usage of the parallelizing FORTRAN and C compilers, whereas message-passing programming typically involves the usage of popular message-passing communication libraries such as the Message-Passing Interface (MPI) standard and Parallel Virtual Machine (PVM). Applications may alternatively use High Performance FORTRAN (HPF) as their parallel library.

Most applications with fine and medium-grained parallelism can be effic-iently parallelized using easy-to-use shared-memory techniques available on each POWERnode system. This system enables a wide range of scien-tific, engineering and commercial applications to take advantage of shared-memory parallelism by using parallelizing FORTRAN 77, FORTRAN 90, and C compilers.

Applications with medium and coarse-grained message-level parallelism can use the efficient communication characteristics of shared-memory within each POWERnode and resort to conventional message-passing between POWERnode systems. Also, many message-passing applications previously running on workstation clusters or pure message-passing architectures can run efficiently on a single POWERnode with little or no modification. Common message-passing libraries such as MPI and PVM on POWER CHALLENGEarray exploit the fast shared-memory communication path between tasks on a POWERnode.

For applications with large scalability and memory requirements, the algo-rithm can be restructured to use the hierarchical programming model to combine the benefits of shared memory with the upward scalability of message-passing techniques. POWER CHALLENGEarray therefore supports several parallel programming models, including:

- Shared-memory with *n* processes inside a POWERnode
- Message-passing with *n* processes inside a POWERnode
- Hybrid model with *n* processes inside a POWERnode, using a combination of shared-memory and message-passing
- Message-passing with *n* processes over *p* POWERnodes
- Hybrid model with *n* processes over *p* POWERnodes, using a combina-tion of shared-memory within a POWERnode system and message-passing between POWERnodes

Tools Suite: Distributed Software Development & System Management

POWER CHALLENGEarray gives you a variety of software tools, enabling you to use it as a distributed parallel processing system and as a distributed parallel-throughput processing engine. Each POWERnode is loaded with the original tools suite from the POWER CHALLENGE platform, including:

- ❑ *XFS*™ high-performance, journaled filesystem
- ❑ *NFS*™ Version 3 network filesystem
- ❑ *MIPSpro Power FORTRAN 77*, *MIPSpro Power FORTRAN 90* and *MIPSpro Power C* compilers support automatic/user-directed parallelization of FORTRAN 77, FORTRAN 90, and C applications for shared memory multiprocessing
- ❑ *CHALLENGEcomplib*, a comprehensive collection of scientific and math subroutine libraries that provide support for mathematical and numer-ical algorithms used in scientific computing
- ❑ *ProDev/Workshop*, a suite of software development tools that includes a parallel program development tool, a parallel debugger, a parallel pro-gram profiler, and performance-tuning tools

POWER CHALLENGEarray provides additional tools to aid in distributed program development and distributed system management, including:

- ❑ HiPPI high-performance networking software
- ❑ Array services, array management software
- ❑ IRISconsole centralized server administration software
- ❑ Message-Passing Interface (MPI) communication software
- ❑ Parallel Virtual Machine (PVM) communication software
- ❑ Array diagnostics

Array services on POWER CHALLENGEarray provides the basis for distribu-ted processing and system management by managing key global resources of POWER CHALLENGEarray, and sending this information to message-passing libraries and to other system management tools. IRISconsole pro-vides a single point of control for administering the POWERnodes that constitute POWER CHALLENGEarray, and can be used with system man-agement and administration tools such as IRIXPro and Performance Co-Pilot. The array diagnostics package aids in fault diagnosis and recovery.

The most commonly used message-passing libraries, MPI and PVM, are tuned for POWER CHALLENGEarray and are available as supported Silicon Graphics products. These libraries provide a uniform message-passing ab-straction, but exploit the shared-memory communication path for intra-POWERnode tasks. There is also support for distributed memory debuggers and visualization tools. Finally, load balancing and batch processing soft-ware tools are available to exploit POWER CHALLENGEarray as a through-put engine and to best allocate resources to parallel programs with varying resource requirements. Figure 1-4 on page 12 gives an overview of the main software solutions available on POWER CHALLENGEarray.

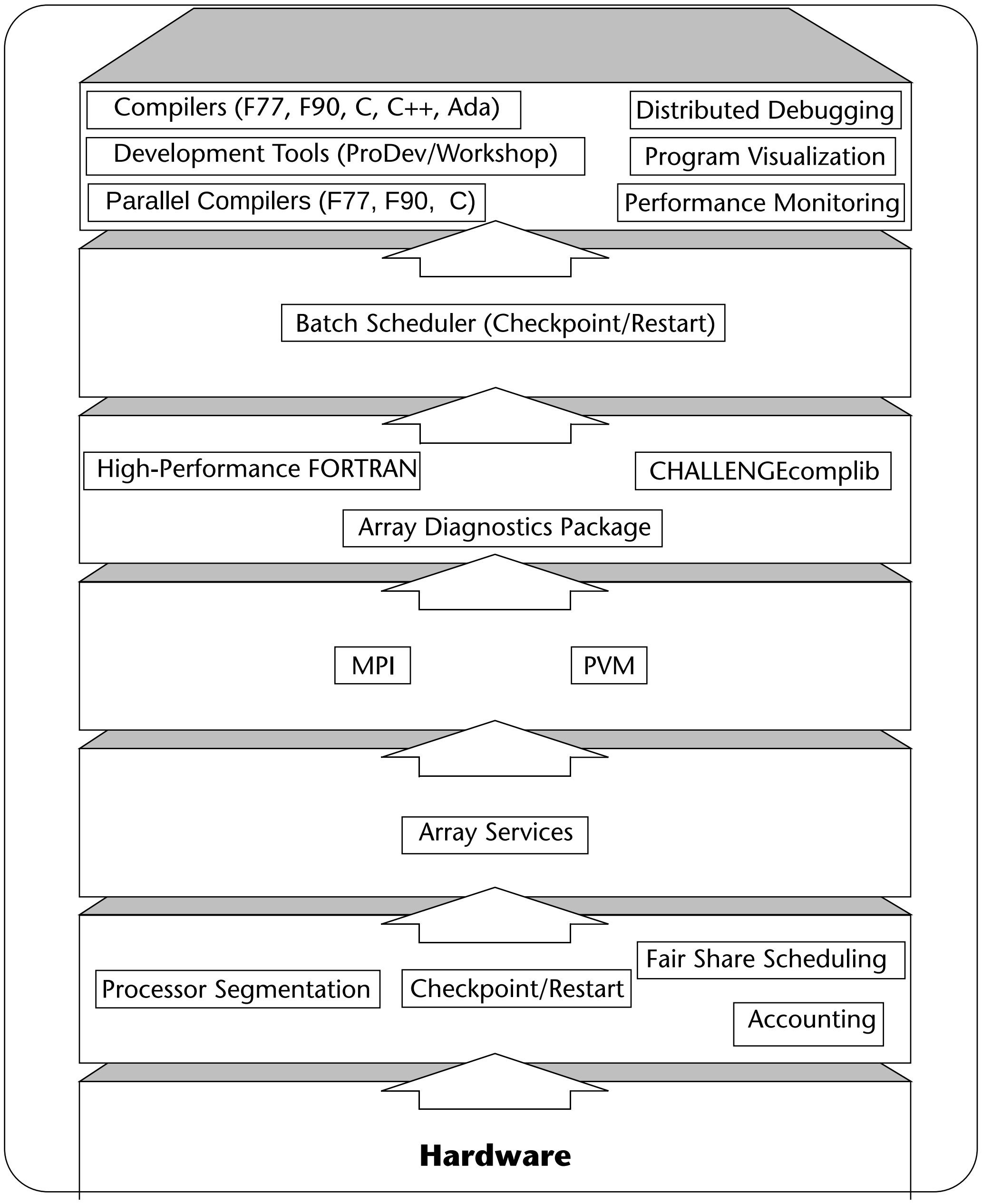


Figure 1-4 Main Software Solutions on POWER CHALLENGEarray

Hardware Architecture

2

This chapter describes:

- ❑ POWERnode
- ❑ HiPPI Interconnect
- ❑ IRISconsole

POWER CHALLENGEarray consists of up to eight POWERnodes connected by high-performance interconnection technology in ways that can be customized to suit communication requirements of varied problems. For POWER CHALLENGEarray, HiPPI is the recommended interconnection technology. The high-bandwidth characteristics of multiple HiPPI chann-els (100 MB/sec per channel) make HiPPI apt for the relaying of large amounts of data between POWERnodes. Also, HiPPI uses ANSI standard- conforming protocol networking technology, assuring interoperability with other HiPPI equipment.

A switch-based interconnection topology is recommended to connect POWERnodes. A switch-based system can dynamically shift between dif-ferent topologies (1D ring, 2D mesh, 3D Torus) to conform to the applica-tion's communication requirements.

POWER CHALLENGEarray also comes with IRISconsole, providing a cen-tral controlling point for system management purposes. The core of IRISconsole consists of Silicon Graphics Indy™ workstation with an ST-1600, the Serial Port Multiplexer, and software for managing serial connections.

POWERnodes based on POWER Onyx supercomputing graphics systems are suitable for applications requiring leading-edge graphic performance.

POWER CHALLENGEarray has a highly flexible and scalable interconnec-tion architecture which need not be restricted to a single topology, tech-nology, or predefined interconnect bandwidth. The interconnection topol-ogy can scale incrementally with additional POWERnodes, and is easily replaced when new interconnect technology, such as ATM, becomes avail-able. All POWERnodes run Silicon Graphics IRIX 6.1 enhanced 64-bit UNIX^{â} operating system which includes a multithreaded kernel, the XFS high-performance journaled filesystem, and NFS version 3 networking software. Table 2-1 summarizes maximum system configurations.

Component	Description
Peak Performance	115.2 GFLOPS
POWERnodes	8
Processor	288 MIPS R10000 or 144 MIPS R8000
Main Memory	128GB
Bisection BW	1.6GB/sec
Disk I/O BW	4GB/sec
RAID Storage Capacity	139.2 Terabytes

Table 2-1 POWER CHALLENGEarray Maximum System Configurations

Figure 2-1 POWER CHALLENGEarray with eight POWERnodes

POWERnode

Each POWERnode is a shared-memory supercomputer, consisting of multi-ple R8000 CPU boards, interleaved memory cards, and POWER Channel™-2 I/O boards, together with a wide variety of I/O controllers and peripheral devices. These boards are interconnected by the POWERpath-2 bus, which provides high-bandwidth, low-latency, cache-coherent communication be-tween processors, memory and I/O graphics subsystems. Table 2-2 on page 16 summarizes maximum system configurations for a POWERnode.

Component	Description
Processors	18 MIPS R8000 CPUs 36 MIPS R10000 CPUs
Peak Performance	14.4 GFLOPS
Main Memory	16 gigabytes, 8-way interleaving
I/O bus	6 POWER Channel-2 buses, each providing up to 320MB/sec I/O bandwidth
SCSI channels	40 fast-wide independent SCSI-2 channels
Disk	17.4TB disk (RAID) or 5.6TB non-RAID disks
Connectivity	6 HiPPI channels, 8 Ethernet channels
VME slots	5 VME64 expansion buses provide 25 VME64 slots

Table 2-2 POWERnode Maximum System Configurations

The RealityEngine Graphics Subsystem

For applications requiring extreme graphics performance, one or more POWERnodes of POWER CHALLENGEarray can be a POWER Onyx super-computing graphics system. POWER Onyx creates a new paradigm for affordable, visual supercomputing.

HiPPI Interconnect

The High-Performance Parallel Interface (HiPPI) is the recommended interconnection technology for POWER CHALLENGEarray. Each POWERnode is required to have one or more bidirectional HiPPI interfaces. HiPPI is the industry standard for high-bandwidth networking today in both system-to-system and system-to-peripheral environments. Standard-ized by the American National Standards Institute (ANSI), HiPPI is widely adopted by research, higher education, and engineering organizations worldwide. It is a simplex point-to-point interface for transferring data at peak data rates of 100 or 200MB/sec over distances of up to 25 meters. IRIS® HiPPI, which provides HiPPI connectivity for Silicon Graphics mach-ines, supports the 100MB/sec option.

The HiPPI physical layer specifies 50-pair, twisted-pair cables for distances up to 25 meters, with the 100MB/sec option using one cable and the 200 MB/sec option using two cables. HiPPI signal lines are unidirectional to accommodate fiber-optic implementations and crossbar switches. Control and data signals are timed with respect to the continuous 25MHz clock signal. IRIS HiPPI contains two 32-bit parallel channels clocked at 25MHz. In addition to the Framing Protocol (HiPPI-FP), which is used by the Silicon Graphics implementation of the MPI library, TCP/IP can be layered over HiPPI, providing a fast communication fabric for TCP/IP applications while retaining naming, reliability, and internetworking flexibility.

The basic organization of the HiPPI information or data framing is as shown in Figure 2-2. Connections are made in a way similar to dialing a telephone. Once a connection is established, one or more packets can be sent from the source to the destination. Each packet contains one or more bursts, each burst contains up to 256 words, and each word is composed of 32 bits with odd parity on each byte. Bursts containing less than 256 words may occur only as the first or last burst of a packet.

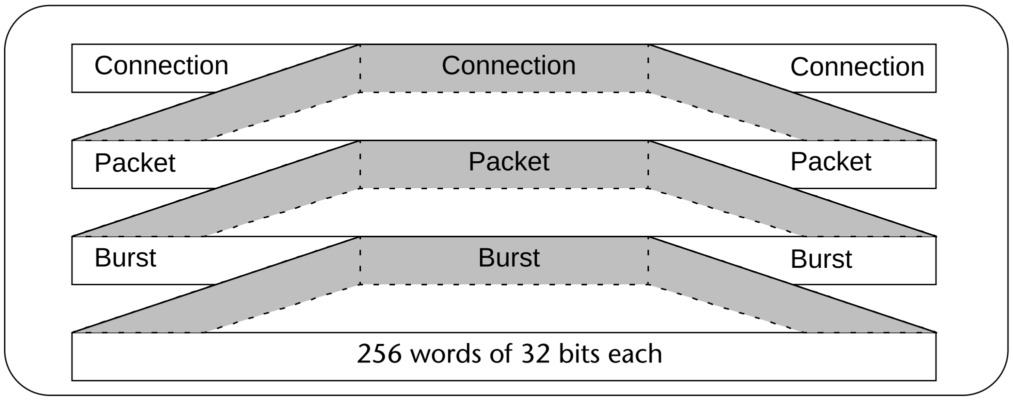


Figure 2-2 HiPPI Data Framings

HiPPI signal lines are unidirectional to accommodate fiber-optic implem-entations and crossbar switches. Control and data signals use differential ECL drivers and receivers. The signal set includes:

- *REQUEST*: (1-bit) source requests a connection
- *CONNECT*: (1-bit) destination accepts the connection
- *READY*: (1-bit) destination permits the source to send a burst
- *PACKET*: (1-bit) brackets one or more bursts into a packet
- *BURST*: (1-bit) brackets 256 data words on contiguous clocks
- *DATA*: (32-bits) data
- *PARITY*: (4-bits) DATA BUS odd byte parity
- *CLOCK*: (1-bit) continuous 25MHz, 40 nanoseconds period
- *INTERCONNECT*: (1-bit) cables connected and power ON

Examples of HiPPI waveforms are shown in Figure 2-3 on page 19. In this example, the source requests a connection by asserting the *REQUEST* sig-nal, supplying the I-Field on the data bus. If the destination can accept the request, it asserts the *CONNECT* signal completing the connection. Once a connection is established, single or multiple packets may be transmitted from the source to the destination; in this example, a packet containing two bursts is sent. Packets are delimited by the *PACKET* signal being true. Packets are composed of one or more bursts, each burst having an even-bit checksum. Either the source or the destination can break the connection by dropping the *REQUEST* or *CONNECT* respectively; in this example, the source breaks the connection.

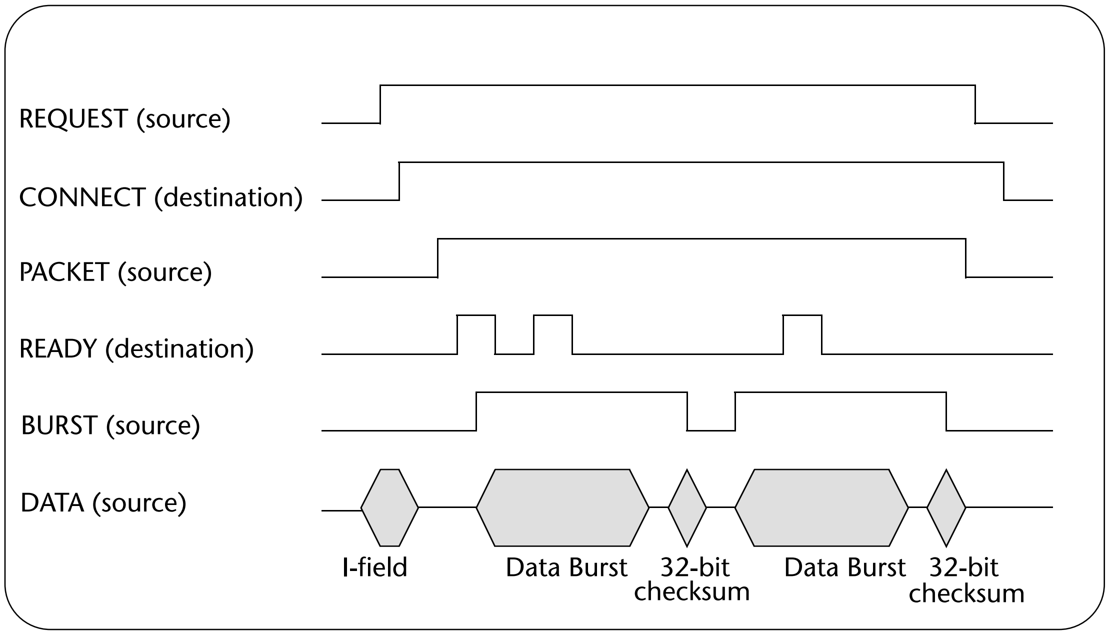


Figure 2-3 Typical HiPPI Waveforms

HiPPI flow control is designed to accommodate the longer distances affor-ded by future fiber-optic-based systems. A HiPPI destination generates a *READY* signal to give the source permission to send a burst of up to 256 words (1,024 bytes). The destination can issue multiple READY signals according to its current buffering capability. These READY signals are queued by the source so that when data transmission is desired, round-trip handshake delays do not occur. All HiPPI source endpoints are required to be capable of enqueuing a minimum of 63 READYs. There is no minimum requirement for a destination's ability to generate READYs. The source channel on the IRIS HiPPI board can enqueue up to 65,535 READYs and the destination channel can generate up to 255 outstanding READYs. By sending ahead and queuing READYs, the two endpoints can optimize the throughput on their connection.

The HiPPI physical layer specifies a point-to-point link, and crossbar switches are the most common interconnection mechanism used with HiPPI to connect multiple devices. The HiPPI crossbar switch is an inter-connection matrix with HiPPI interfaces, as shown in Figure 2-4. It has an aggregate bandwidth that is a multiple of the peak data rate of any single interface. Multiple simultaneous connections can exist through it, and since connections do not share switch resources, they can pass data con-currently at the full HiPPI rate. Connection switching times are typically less than 1 microsecond.

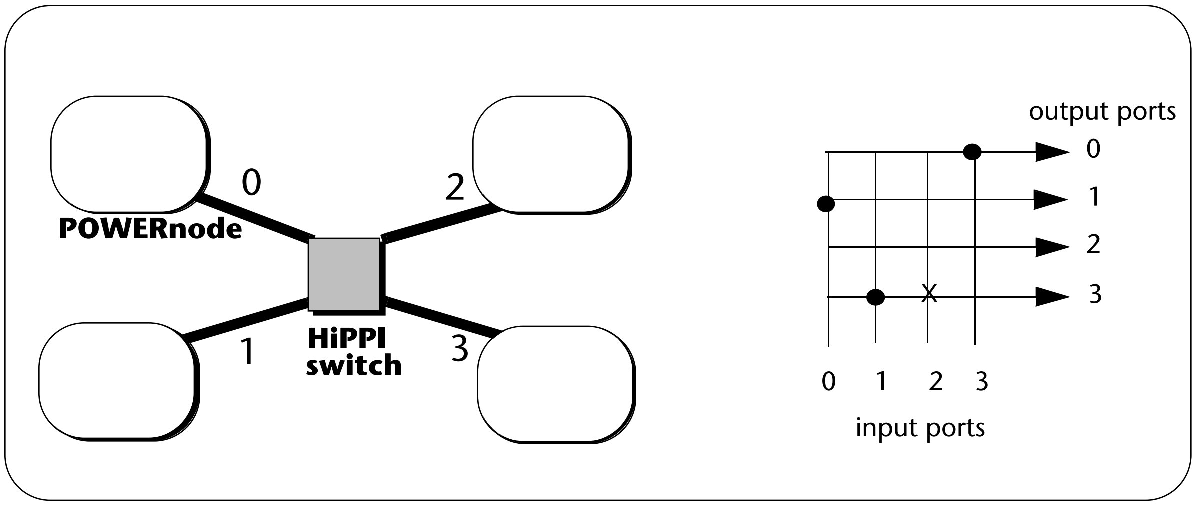


Figure 2-4 Non-Blocking Crossbar Switch

Each simplex HiPPI coming into the switch can be connected to one going out. The switch connection is made electrically at the time the HiPPI con-nection is negotiated. If the requested destination is already connected to another source, the switch can either reject the new request or delay satis-fying the new request. The requester can time-out and withdraw an unsat-isfied request and then try a different destination. In the example shown in Figure 2-4 on page 20, 0 sends to 1, 1 sends to 3, 2 sends to 3, and 3 sends to 0. There are two requests for output port 3, and one of the con-nection requests to 3 (2 to 3) fails, and is either rejected or queued.

IRISconsole

IRISconsole, part of POWER CHALLENGEarray, provides a central control-ling point for the collection of POWERnodes. It is a powerful, easy-to-use central control point which continually monitors the activities of the POWERnodes. It functions as a console server, managing serial connectiv-ity to the POWERnodes; it also functions as a data collection center, gathers status information, and perform intelligent actions based on that information. In doing so, IRISconsole ties in many disjoint functionalities that already exist to manage a collection of POWERnodes.

The IRISconsole architecture is shown in Figure 2-5 on page 22. The core of IRISconsole comprises:

- ❑ Indy workstation with an ST-1600
- ❑ Serial Port Multiplexor
- ❑ Serial and SCSI cables
- ❑ Software to manage serial port connectivity
- ❑ Text-based interface to manage a cluster remotely

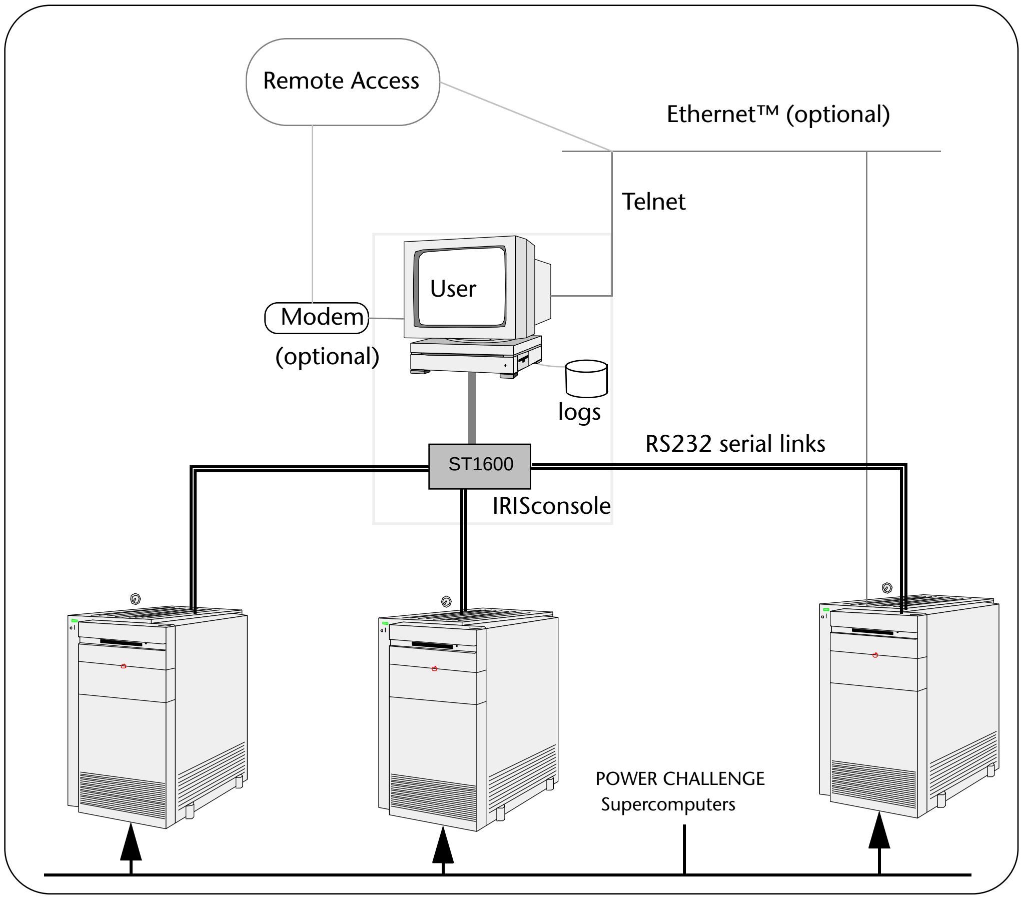


Figure 2-5 IRISconsole Architectures

Users have the option of connecting a modem to the IRISconsole Indy Workstation and/or connecting it to Ethernet. IRISconsole uses an easy-to-use configurable graphical interface, allowing an operator to click a button to perform tasks such as resetting a system or generating a Non-Maskable Interrupt (NMI), which forces a system to generate a corefile for debugging purposes.

"Intelligent" Console for Managing Multiserver System Activity

IRISconsole monitors up to 16 POWERnodes and provides information such as:

- Voltage levels of the power supply
- Operating temperature
- Speed of the internal blowers
- Availability report of the servers
- System log
- Console activity by other users
- Hardware inventory of any machine

Alarms are set off if any data points are outside of established threshold values. These alarms may be audible and/or indicated by changes in the color of a graph. IRISconsole can notify remote operators (via e-mail or pager, for example) of the occurrence of an anomaly and/or store a graph of it for later troubleshooting. If ESCALL is installed as part of the service contract, then IRISconsole, upon detecting an anomaly, will automatically start a program that facilitates the notification of the service provider.

Each configured system is presented as a window on the screen. Transac-tions in the window can be optionally logged; each window has scrollable history. All viewable graphs can be saved to a file or a PostScript™ format for later inspection, printing, debugging, or interchange across a network.

Remote Management and Diagnostics

With a modem, IRISconsole can remotely monitor and manage POWER CHALLENGEarray off-site through a text-based or graphical interface (using a SLIP or PPP connection). This helps system administrators watch for system conditions such as a disk becoming full or unusual interrupts.

Independent Security System

IRISconsole provides its own password-based security system (separate from the IRIX "/and more./passwd" password mechanism). It gives you the ability to control access to any entry point and supports security check-points at arbitrary points. You may choose between login-based access (log-in plus password) and simple password-based access (password only) for any checkpoint. This allows the System Administrator to securely configure the array (to add or delete machines, for example). Also, the text-based IRISconsole interface used for remote access restricts the user to only IRISconsole functionality.

Software Overview

3

3.1 POWER CHALLENGEarray Software

POWER CHALLENGEarray can be used as a single large-scale central com-puting resource or a distributed shared resource. In a production environ-ment, a single POWER CHALLENGEarray system may be shared between different teams, users, groups and projects. Similarly, the system may be used for running various types of applications with different resource requirements. Managing and monitoring the system in such an environ-ment can be a cumbersome task for the system administrator. To fit into such an environment, POWER CHALLENGEarray is equipped with a rich assortment of centralized system management tools that together provide a scalable environment for efficient utilization, monitoring, and manage-ment of the system resources. This is in addition to the suite of software development tools that are provided to support development, debugging and performance monitoring of shared memory, message-passing and data parallel programs on POWER CHALLENGEarray.

Thus, tools on POWER CHALLENGEarray can be classified as follows:

- ❑ Native POWERnode tools
- ❑ Array services
- ❑ Distributed program development tools
- ❑ Distributed batch processing tools
- ❑ Distributed system management tools

These tools are discussed below. Further details about POWER CHALLENGEarray can be obtained from the World Wide Web at:

`http://www.sgi.com/Products/PowerChallengeArray/`

Native POWERnode Tools

The software environment on a POWERnode includes the following:

- ❑ 64-bit operating system
- ❑ 64-bit fast filesystem, XFS
- ❑ 64-bit NFS Version 3
- ❑ 64-bit MIPSpro compilers, supporting FORTRAN 77, FORTRAN 90, C, and C++
- ❑ High-performance, optimized scientific and math libraries
- ❑ 64-bit development environment

MIPSpro Compilers

MIPSpro compilers are the Silicon Graphics third-generation family of optimizing and parallelizing compilers. They have comprehensive support for parallel application development. The compilers perform a range of general-purpose and architecture-specific optimizations to improve applica-tion performance by reducing the number of instructions executed. This better use of the CPU’s instruction set maximizes register use, minimizes memory references, and eliminates unused or redundant code.

A rich assortment of command-line options can leverage different combi-nations of optimizations. In general, optimizations are spread across the compilation system for better efficiency. The key optimizations include architecture-specific optimizations such as software pipelining, instruction scheduling, and automatic blocking; statement level optimizations such as array expansion, common subexpression elimination, and global constant propagation; loop-level optimizations such as loop unrolling, loop inter-change, and unroll-and-jam; and procedure-level optimizations such as procedure inlining and interprocedural analysis (IPA).

MIPSpro Power compilers (MIPSpro Power FORTRAN 77, MIPSpro Power FORTRAN 90, and MIPSpro Power C) support automatic and user-directed parallelization of FORTRAN and C applications for multiprocessing execution. The compilers employ automatic parallelization techniques to analyze and restructure user applications for parallel execution. Automatic parallelization is invoked via the *-pfa*/-*pca* flags for FORTRAN 77, FORTRAN 90, and C, respectively.

MIPSpro compilers also provide a comprehensive set of standards-based comment directives that enable users to assist the compiler in the parallel-ization process. Users can use these directives to provide additional infor-mation to the compiler to boost parallel performance. User-assisted para-llelization is enabled by specifying the *-mp* flag for both FORTRAN and C. The directives provide comprehensive support for specifying and control-ling the degree and dynamics of parallel execution.

The parallelization technology is fine-tuned to take advantage of the POWER CHALLENGE system architecture. A combination of automatic and user-assisted parallelization can lead to substantial improvements in the performance of many programs.

dbx, pixie, and prof

The MIPSpro compiler family also includes basic debugging and program runtime analysis tools including *dbx, pixie* and *prof. T*he source level de-bugger, *dbx*, facilitates debugging of parallel programs written using FORTRAN 77, FORTRAN 90, C, and C++. The standard profiling tool, *prof*, provides “program counter sampling” of an application’s execution. Another profiling tool, *pixie*, provides statement-level execution profiles by using a basic-block counting technique which provides much finer resolu-tion than *prof*.

High-Performance Scientific and Math Libraries

The compilers are complemented by *CHALLENGEcomplib*, a comprehensive collection of scientific and math subroutine libraries that provide support for mathematical and numerical algorithms used in scientific computing. The key motivation for creating *CHALLENGEcomplib* is to provide standard library functionality and to improve the runtime performance of applica-tions. By incorporating *CHALLENGEcomplib* routines in compute-intensive

portions of scientific and engineering applications, users can take advantage of the performance capabilities of the underlying system without having to rewrite their applications.

CHALLENGEcomplib is a parallel implementation of the scientific and math libraries, employing the *sproc* facility available on the POWER CHALLENGE platform to create new parallel processes. *CHALLENGEcomplib* is similar to scientific libraries provided by other supercomputing vendors such as the *Cray SCILIB™*, *IBM ESSL*, and *Convex VECLIB*. The library consists mainly of the subcomponent, *complib.sgimath*, which is the hand-tuned portion of *CHALLENGEcomplib* and includes the following routines:

- Basic Linear Algebra Subprograms (BLAS), levels 1, 2, and 3
- 1D, 2D, and 3D Fast Fourier Transforms (FFT)
- Convolutions and correlation routines
- LAPACK, LINPACK, and EISPACK
- SCIPORT (Portable version of SCILIB)
- SOLVERS: *pcg* sparse solvers, *direct* sparse solvers, *symmetric* iterative solvers, and solvers for special linear systems.

ProDev/Workshop: Application Development Tools

ProDev/Workshop is a programming environment specifically designed to facilitate the development of parallel programs. *ProDev* has the following tools to assist the advanced developer:

ProDev/Workshop Static Analyzer

This visual source code navigation and analysis tool provides the ability to visualize program structure and allows easy navigation through code— vital for restructuring and re-engineering of existing software. It is helpful in porting situations, when code being ported from other platforms will not run or compile. It provides multiple queries into code structure, such as queries on functions, variables, and common blocks.

ProDev/Workshop Debugger (cvd)

The Workshop debugger, *cvd*, is a state-of-the-art, source-level, parallel de-bugger, featuring multiple graphical views that are dynamically updated during program execution. It is tightly integrated with the performance analyzer, providing increased efficiency for overall program analysis. It pro-vides 15 different views into a program and the views are dynamically updated as the user steps through the program. *cvd* also provides debug-ging support for programs that have multiple processes or that have been parallelized using shared memory (MIPSpro compilers) or message-passing (MPI) techniques. Finally, it has three views that provide machine-level debugging: Register View, Memory View, and Disassembly View.

ProDev/Workshop Performance Analyzer

The Performance Analyzer is an integrated collection of tools that measure, analyze, and help to improve application performance. Tightly integrated with *cvd*, it allows the user to visualize a program's performance over sepa-rate phases of execution, and correlate the information back to the source code. All the views show performance statistics on a per-thread basis, and provide the ability to correlate the performance of all threads.

ProDev/Workshop Pro MPF

Workshop Pro MPF provides a powerful visual interface into the transfor-mations of MIPSpro Power FORTRAN 77 and MIPSpro Power FORTRAN 90 to show which loops were parallelized, which were not, and why they were not. In all cases where a loop could not be parallelized, it will show the user the obstacles to parallelization and allow the user to rearrange the algorithm to circumvent those obstacles.

Array Services

POWER CHALLENGEarray supports a suite of software features known as array services to manage and administer the array as a single system. The array services revolve around the concept of an array session, which is a set of processes running on different POWERnodes in an array, that are con-ceptually related as a single job.

Additional services are provided by the array services daemon, which is aware of the configuration of the array and provides functions for describ-ing and administering it. Array services store and manage key information about the POWER CHALLENGEarray. The array services information can be used both by the message-passing libraries as well as by resource manage-ment and accounting tools that fall under the system management class of tools. Users may also access this information via commands provided by array services.

The chief component of array services is *arrayd,* the array services daemon, which runs on each POWERnode in the array. The array services daemon manages array-related information residing in the *array configuration data-base* located in the local filespace. Each daemon has knowledge about one or more arrays (for cases where a POWERnode is part of more than one POWER CHALLENGEarray system) and the machines that comprise them.

An array services daemon running on each POWERnode performs the following tasks:

- Maintaining information about the current array configuration, and providing this information to other commands and programs
- Determining which processes belong to a particular array session and giving that information to other commands and programs
- Allocating global Array Session Handles (ASH)
- Forwarding commands issued by a user on a single POWERnode to all other POWERnodes of the POWER CHALLENGEarray system

Array Sessions

Mechanisms typically used to manage multiple related processes (for exam-ple, process groups and terminal sessions) are limited in scope to a single POWERnode, and cannot be used for jobs with tasks running on two or more POWERnodes. Array services provide array sessions, which correlates single-job processes running across several POWERnodes.

An array session is a set of processes, possibly running across several POWERnodes of a POWER CHALLENGEarray, that are related to another by a single, unique identifier called the Array Session Handle (ASH). An ASH is a 64-bit value and can be either local or global. A local ASH is assigned by the kernel and is guaranteed to be unique within a single POWERnode, whereas a global ASH is assigned by the array services daemon, and is guaranteed to be unique across the entire POWER CHALLENGEarray. A child process ordinarily inherits the ASH of its parent when it is created, thus becoming a member of its parent’s array session. However, it is possible for a process to leave its parent’s array session and start a new one. This would be done by programs such as login or rshd so that logging on to a system will effectively start a new array session. This would also be done by batch-queuing systems so that work done on behalf of another user will be done in its own array session. Figure 3-1 on page 31 illustrates array services for two distinct arrays, ARRAY1 (consisting of POWERnodes 1, 2, 3, and 4) and ARRAY2 (consisting of POWERnodes 4, 5, and so on), with POWERnode4 belonging to both the arrays. Processes 1 and 2 on POWERnode1 and process 3 on POWERnode3 belong to an array session, identified by a global ASH.

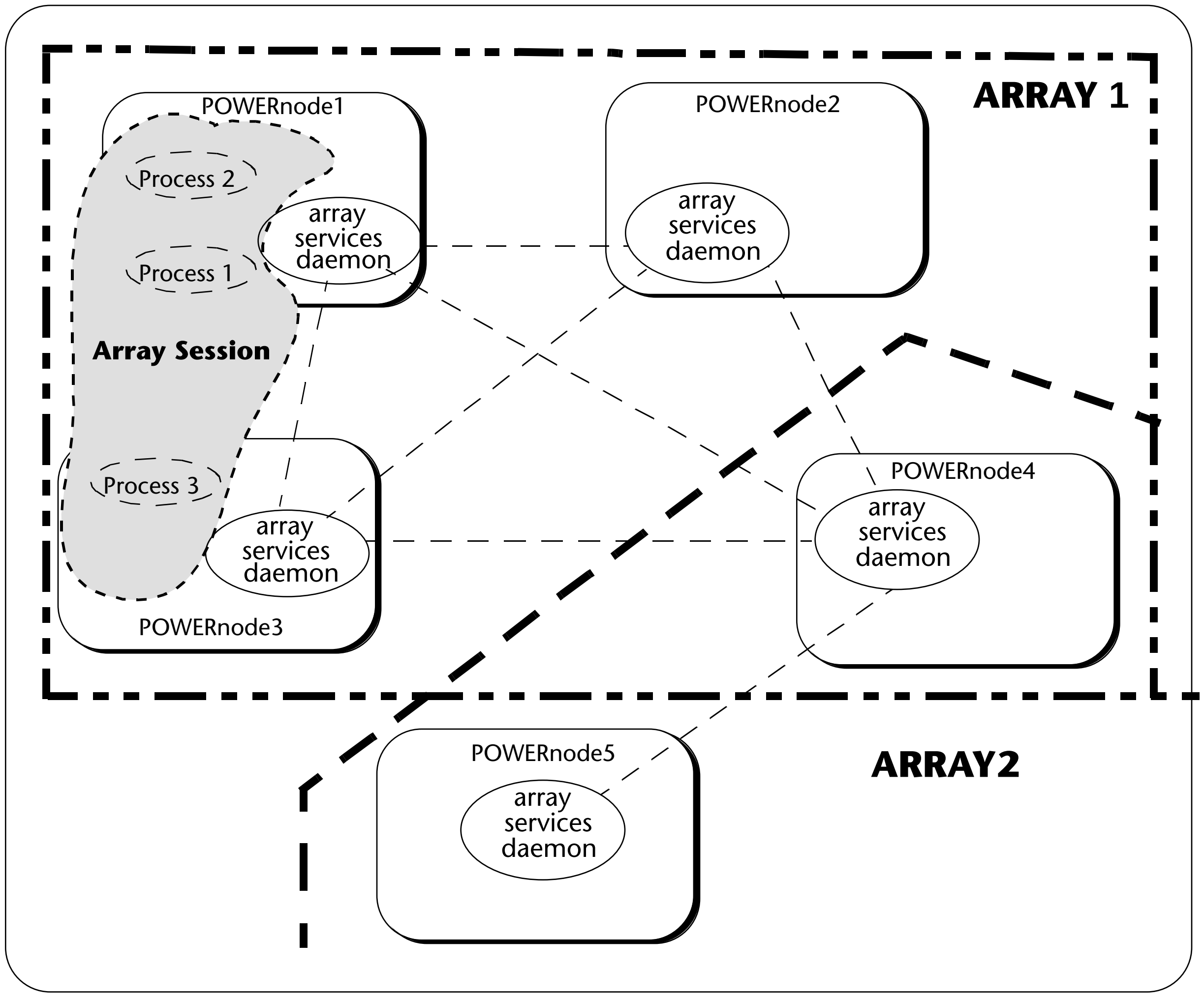


Figure 3-1 Array Services for Two POWERCHALLENGEarrays

A parallel job having tasks on different POWERnodes is thus an array session. All its tasks can be identified via the session's global ASH. This in-formation can be used by a message-passing library or by system manage-ment programs to provide resource management information to the users, and to generally control the parallel job. For example, it can be used for killing a job or accounting for a job's resource usage. When the last task on a POWERnode with a given ASH exits, a session accounting record con-taining accumulated statistics for all of the processes that ran in the array session (or all the tasks) on that POWERnode is written and the array session ends.

ASH

The kernel assigns a unique ASH to each new array session as its handle. This type of ASH is referred to as a local ASH; though it is guaranteed to be unique on the local POWERnode, it may be in use by a different array sess-ion on another POWERnode in the same array. Hence, a local ASH is not appropriate for identifying parallel jobs spanning more than one POWERnode. For such jobs, the array services daemon assigns a global ASH, which is unique across an array. By arranging the same global ASH to be associated with each process in a job, it is possible to treat the set of processes as a single entity. When a new array session is started, it only has a local ASH; it upgrades its handle to a global ASH either by obtaining a new global ASH from the array services daemon (if it is the first process of the job) or via information passed to it from its parent (if it is not the first process of the job). Once an appropriate global ASH is established for a job, the processes on each POWERnode that started the new array session on that node can fork off any number of children to do the required work. These children will inherit the global ASH of this session, thus all the tasks of the job get correlated.

Array services consist of five main components, namely the array services daemon, the array configuration database, the array command, the ainfo command, and the array library. Once the array services have been set up, they can be accessed by users and programs in one of three ways: via the array or ainfo commands or via libarray library calls. Each of the five com-ponents of array services is discussed below.

Array Configuration Database

Each array services daemon maintains information about one or more arrays, POWERnodes making these arrays, commands executable by the daemon, and various local options, in configuration files in its local file-space. This information is given to other programs and commands, thus other array-oriented programs do not need separate configuration data.

array Command

The array command is one of two main array services interfaces which lets users specify commands for execution on a given array. It passes array commands to the array services daemon and reports the results. While an array services daemon requires IRIX 6.1 or later, the array command can also run under IRIX 5.3. Thus, the array command can be used on non-POWERnodes serving as the central console interface for the array. The array command allows users to: specify the name of the array to direct the command to; indicate a local request; set the verbose mode on.

Arguments to the array command are the user-command and its argu-ments. The user-command refers to an entry in each machine's array con-figuration file, which in turn specifies the command to execute and other information. This gives flexibility in handling commands across an array. Additionally, since the array configuration file is under the system admini-strator's control, only a set of safe commands can be executed by a user.

ainfo Command

The *ainfo* command is the second of two main interfaces to array services. It displays information about arrays known to the array services daemon, and array functions. This information, useful for interactive users and shell scripts, is displayable in formats appropriate for either. The type of infor-mation displayed is set by the request argument: Information about all arrays known to the array services daemon, the ASH of a given process, information about each machine in the array specified, process identities of processes on the local machine running in the array session, and array session handles of global array sessions in the array specified. You can also use it to obtain a new global ASH for the array specified.

libarray Library

libarray is the main interface to array services and comes in o32, n32, and 64-bit versions. A program includes the array services library by using the option *larray* during compilation. Library features for interacting with the array services daemon include functions to:

- Allocate a global ASH
- Indicate whether an ASH is global or local

- Request all global ASHs in a specified array
- Request information on all known arrays
- Request information on all POWERnodes in a specified array
- Describe hardware/network configurations of POWERnodes on an array
- Execute an array command from within a program.

Figure 3-2 shows you various components of array services on a POWERnode and an Indy workstation.

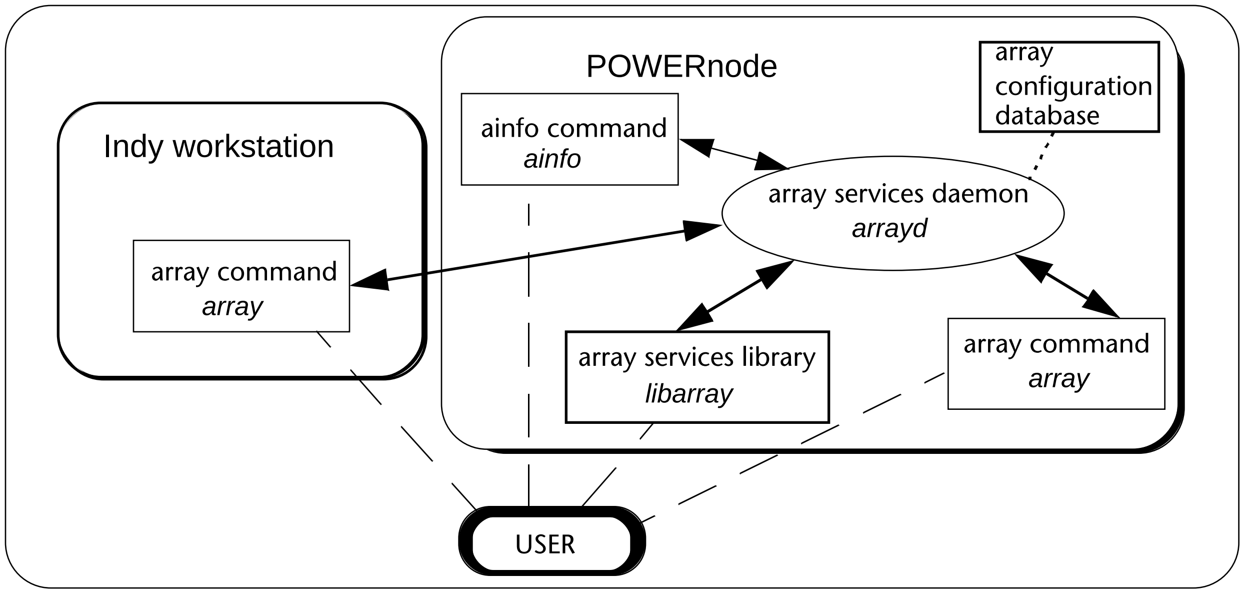


Figure 3-2 Components of Array Services

Distributed Program Development Tools

Software tools to aid the programmer in effectively using POWER CHALLENGEarray as a large parallel machine include array services, message-passing libraries MPI and PVM, the Single Program Multiple Data (SPMD) processing language HPF, and tools for distributed program visual-ization and debugging. The message-passing libraries are optimized for POWER CHALLENGEarray, and take advantage of the benefits of shared-memory multiprocessing for interprocess communication within each

POWERnode. The libraries use fast message-passing protocols to commun-icate between POWERnodes. In addition to these optimized libraries being available from their respective public domain sites, Silicon Graphics pro-vides its own highly-tuned implementation of the MPI and PVM libraries.

Message-Passing Interface (MPI)

MPI is a standard message-passing library interface developed by the MPI Forum, a broadly-based group of parallel computing vendors, parallel lib-rary writers, and application scientists. MPI works to assimilate the most attractive features of a number of existing message-passing systems, rather than selecting and adopting one as the standard. The main advantages of establishing a message-passing standard such as MPI are low-overhead, portability, and ease of use. MPI specifies a binding for FORTRAN 77 and C. Thus MPI can serve as the lower-level message-passing infrastructure for a wide range of higher-level distributed parallel applications across different parallel platforms.

The MPI library consists of routines for:

- Point-to-point communication
- Collective communication
- Group management
- Communicator management
- Process topologies
- Environment management
- Profiling

Silicon Graphics has adopted MPI as the primary message-passing model for POWER CHALLENGEarray. A highly optimized, native version of MPI is available from Silicon Graphics for POWER CHALLENGEarray.

In addition to employing an optimized shared-memory mechanism for intra-POWERnode communication and general TCP/IP support for inter-POWERnode communication, the Silicon Graphics optimized MPI will exploit a customized mechanism to achieve low latency for inter-POWERnode communication. This mechanism, called the HiPPI bypass protocol, employs several techniques to achieve these objectives. Geared toward reducing the latency of short messages (less than 16K), this mech-anism bypasses the kernel for message exchanges across the HiPPI network. It involves a dedicated memory area in an application that is shared with the network device and the device driver; data movement through the memory area is managed in user space without help from the kernel, except when initializing the area. HiPPI firmware manages flow control and security issues, which is usually handled by the kernel. Latency for short messages is further reduced by pre-allocating send and receive buffers and pinning them in memory. The Silicon Graphics MPI library allows you to specify the number of such pre-allocated buffers, since this requirement may vary.

The MPI standard does not provide support for dynamic spawning of tasks. It also does not provide mechanisms to specify the initial allocation of processes to an MPI computation and their binding to physical processors. The runtime environment must therefore provide a means for the user to specify where to start the tasks of a parallel job. Additionally, there is often a need to start jobs remotely. Silicon Graphics' MPI provides the command *mpirun,* which takes host names and numbers as arguments, among others, to start off the user's tasks on the machines specified. This is a flexible MPI program launcher that provides a user with a wide variety of options for starting an MPI program on POWER CHALLENGEarray. The Silicon Graphics MPI can support several parallel programming models on POWER CHALLENGEarray, including:

- ❑ Shared-Memory MPI Model
- ❑ Message-Passing MPI Model
- ❑ Hiearchical MPI Communication Model
- ❑ MPI Hybrid Programming Model

Shared-Memory MPI Model

All tasks of an MPI program run on the CPUs of a single POWERnode, employing shared memory for communication between tasks. The number of MPI tasks may be greater than the number of CPUs in the POWERnode.

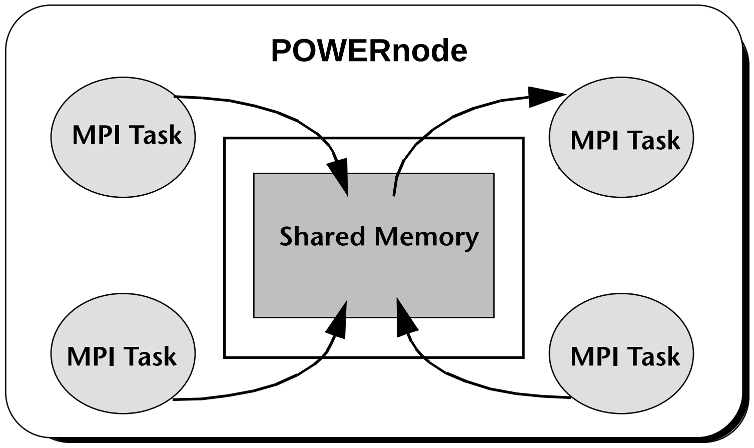


Figure 3-3 Support for MultiParallel Memory Sharing

Message-Passing MPI Model

The tasks of an MPI program run on CPUs across several POWERnodes, with one or more tasks running on each POWERnode. Tasks within a POWERnode employ shared-memory for communication with each other, and sockets for communicating with tasks on other POWERnodes. This is the most general-purpose model for MPI programs that can be run with no modifications across different architectures.

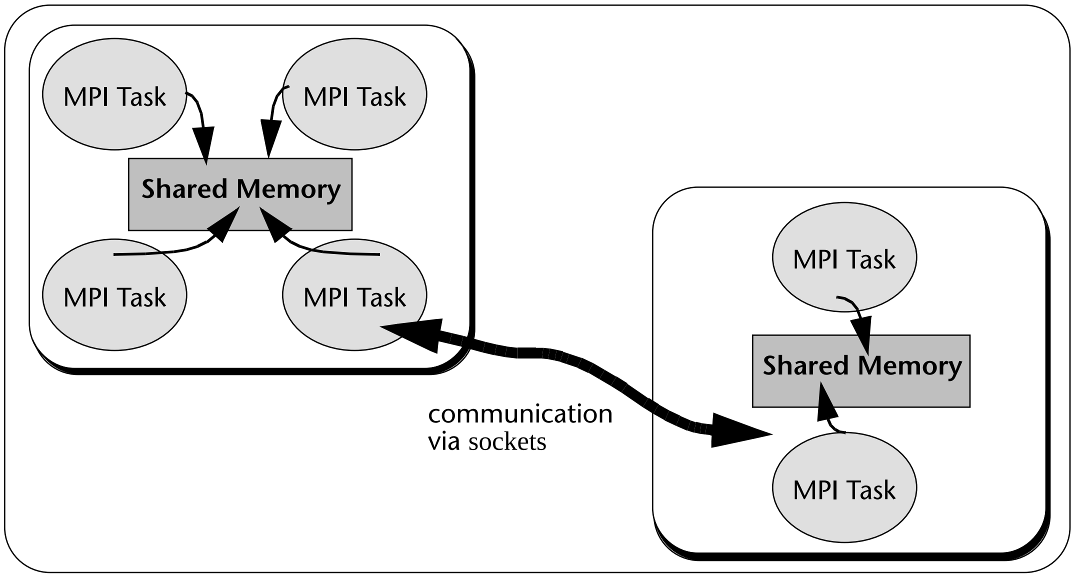


Figure 3-4 Message-Passing MPI Model

Hierarchical MPI Communication Model

In this case, tasks within a POWERnode may be divided into several groups, with shared-memory communication between tasks within a group, and sockets between groups within the same POWERnode. As in other cases, tasks between POWERnodes communicate via sockets. This model allows debugging array applications on a single node.

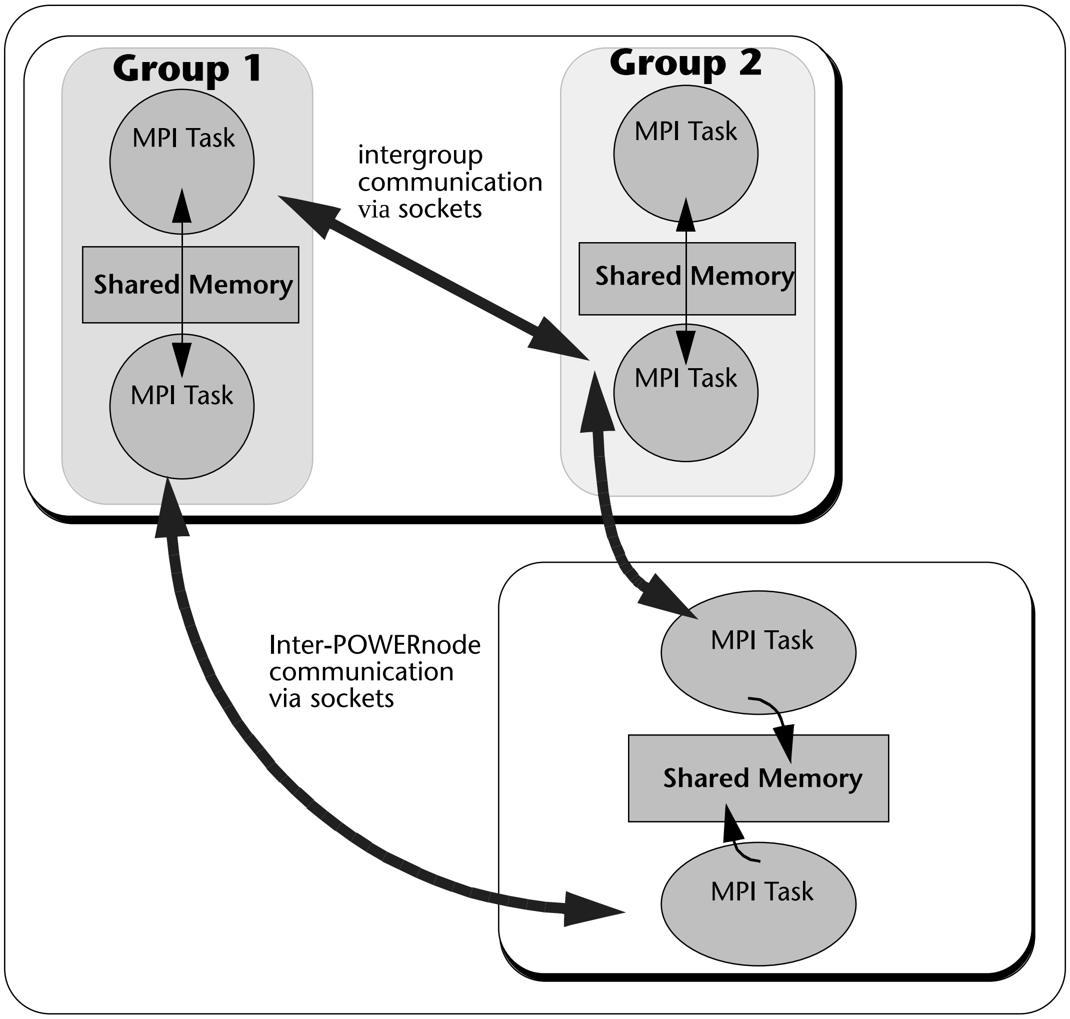


Figure 3-5 Hierarchical MPI Communication Model

MPI Hybrid Programming Model

Here the MPI library can be combined with the native shared-memory par-allelization techniques (both compiler-assisted and explicit sproc'ing) available on POWERnode. This model restricts the number of MPI tasks per POWERnode per job to one. Therefore, an MPI program can start with *n* tasks, one on each POWERnode, where n is the number of POWERnodes. Then each MPI task can spawn multiple threads within each POWERnode using the native POWERnode shared-memory parallel programming prim-itives. An MPI task (one on each POWERnode) communicates with another MPI task via sockets, and with the threads on its POWERnode via shared memory. This is the most general-purpose hybrid parallel programming model that combines the benefits of the shared-memory programming model with the benefits of the message-passing programming model.

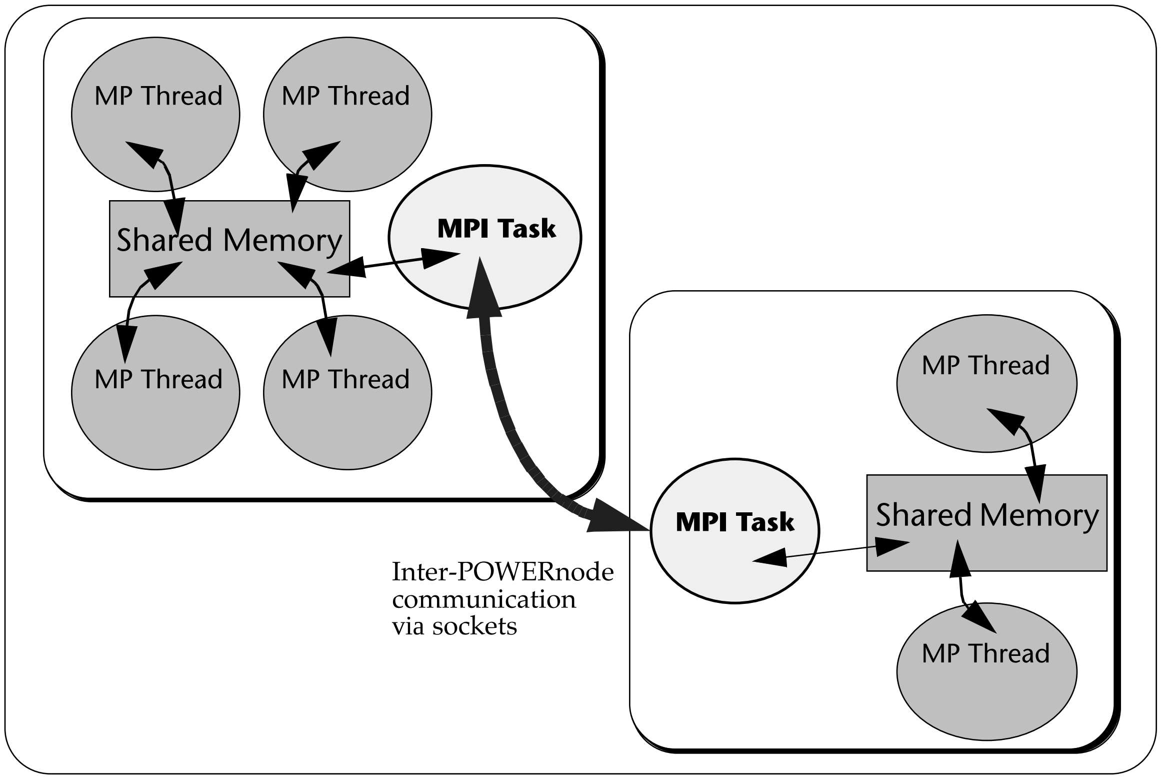


Figure 3-6 MPI Hybrid Programming Model

A 64-bit shared memory port of MPICH, the public domain implementa-tion of MPI, is also available for POWER CHALLENGEarray from Argonne National Laboratories. For POWER CHALLENGEarray, this implementation employs shared-memory for intra-POWERnode communication, and sock-ets for inter-POWERnode communication.

Parallel Virtual Machine (PVM)

PVM, originating from the Oak Ridge National Laboratory and the University of Tennessee at Knoxville, is another message passing library used for parallel processing across a heterogeneous collection of compu-ters. PVM allows applications to be partitioned into multiple processes for concurrent execution on different hosts in an environment. Each host can itself be a parallel computer, with multiple processors connected by a pro-prietary network or shared memory.

Silicon Graphics also provides an optimized version of PVM for the POWER CHALLENGE and POWER CHALLENGEarray systems. The Silicon Graphics implementation of PVM is compatible with PVM 3.3.9. and uses a combination of message-passing (socket-based TCP/IP) and shared-memory techniques to communicate between PVM tasks. Figure 3-7 on page 42 illustrates three different intertask communication methods used by SGI PVM 1.Ø. These methods are known as the Non-Route-direct, Route-direct and Shared Memory methods of communication. Each host is a POWERnode.

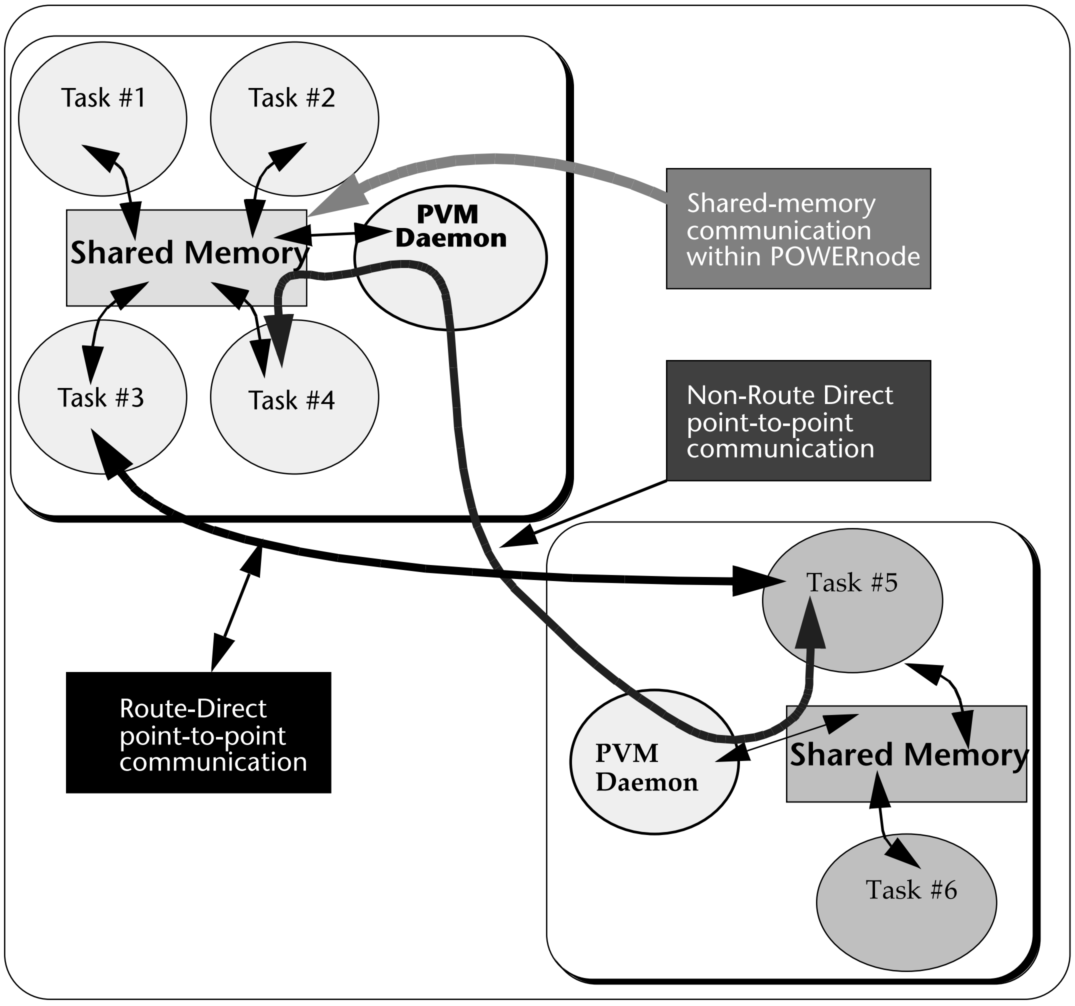

Task #1
Task #2
Shared Memory
PVM Daemon
Task #3
Task #4
Shared-memory communication within POWERnode
Non-Route Direct point-to-point communication
Task #5
Route-Direct point-to-point communication
PVM Daemon
Shared Memory
Task #6

Figure 3-7 Communication Methods in SGI PVM 1.0

Non-Route-direct Communication

By default, PVM tasks use the Non-Route-direct mode of intertask comm-unication by exchanging data through the local PVM daemon on each host. This is illustrated by the communication path between task #4 on POWERnode #1 and task #6 on POWERnode

#2 in Figure 3-7. Routing messages through the local PVM daemon can increase communication costs as a result of single-point congestion if multiple tasks within one host need to communicate with one or more tasks on other hosts.

Route-Direct Communication

The Route-Direct method overcomes the overhead of the non-Route-Direct method by allowing PVM tasks to communicate directly with one another without going through the host PVM daemon. This is illus-trated by the communication path between task #3 on POWERnode #1 and task #5 on POWERnode #2 in Figure 3-7 on page 42.

Shared-Memory Communication

PVM tasks can use message-passing, route-direct methods for interhost communication (between different hosts) and shared-memory for intra-host communication (within the same host). PVM speeds up interproc-ess communication within a single host by allowing parallel processes executing on different processors to communicate with each other by reading and writing to shared memory. Shared-memory communication has advantages over traditional message-passing communication:

- Global shared-memory access is several orders of magnitude faster than memory-to-memory communication in a distributed memory architecture
- By using shared memory, processes can overcome the inefficiencies of packing and unpacking data at the source and destination respectively that is inherited by a pure message-passing model of communication

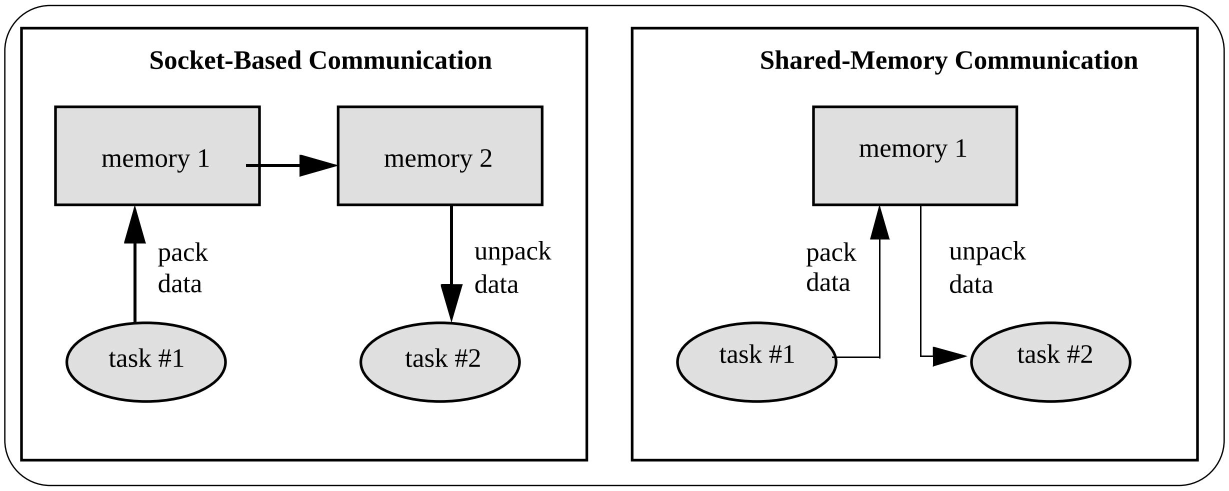


Figure 3-8 Efficiency of PVM InterProcess Communication Using Shared Memory

The combination of the above two advantages can result in speeding up PVM intertask communication by several orders of magnitude. By using PVM in a distributed shared-memory environment, applications can be partitioned to take advantage of shared-memory communication within each POWERnode and high-bandwidth message-passing communication between POWERnodes over sockets. Thus, existing PVM applications that have run on traditional message-passing systems can execute more efficien-tly in a distributed shared-memory multiprocessing environment such as POWER CHALLENGEarray.

High-Performance FORTRAN (HPF)

Although parallel computing has been widely available for more than half a decade, scientists and engineers are still reluctant to use it. This is mainly because parallel machines traditionally lack software systems that make them easy to use. Application developers want to program in a standard language which is portable across a broad range of platforms. Compilers for these standard languages should deliver high performance consistently, so that programmers can avoid the low-level details of managing the parallelism and the memory hierarchy unless it is absolutely necessary for achieving the desired level of performance.

High-Performance FORTRAN (HPF) compiler development is a step toward providing a standard, portable, high-level parallel programming model that is effective on a large segment of parallel applications running on shared-memory MP systems, MPP systems, clusters, and shared and distrib-uted memory hybrid machines such as POWER CHALLENGEarray. HPF is an extended version of FORTRAN 90 that is emerging as a standard for pro-gramming of shared and distributed-memory systems in the data parallel style. HPF incorporates a data-mapping model and associated directives that allow a programmer to specify how data is logically distributed in an application. An HPF compiler interprets these directives to generate SPMD code that minimizes interprocessor communication in distributed systems and maximizes data re-use in all types of systems. This makes HPF an “enabling” technology with a number of advantages, including:

- The HPF programming model makes it possible to port parallel programs and/or write them with much more ease than with traditional non-port-able message-passing or explicit thread calls
- HPF programs generally have a much higher assurance of being correct because the burden of parallelization and communication between proc-esses is shifted to the compiler. Machine-specific details such as cache sizes, number of CPUs, and message-passing techniques are all recog-nized and used by the compiler with little or no input from the user
- HPF programs are much easier to read and debug than explicitly written message-passing or threaded programs
- Change in the underlying architecture can be accommodated by recom-piling the HPF programs

As an active member of the HPF standards committee, Silicon Graphics is involved in the HPF language development and standardization efforts. Two prominent HPF solutions available on Silicon Graphics POWER CHALLENGEarray systems are xHPFä from Applied Parallel Research and PGHPFä from the Portland Group, Inc. The HPF compilers are tightly integrated with Silicon Graphics MIPSpro 64-bit FORTRAN compilers.

Figure 3-9 illustrates the process of converting an HPF source code into a parallel program.

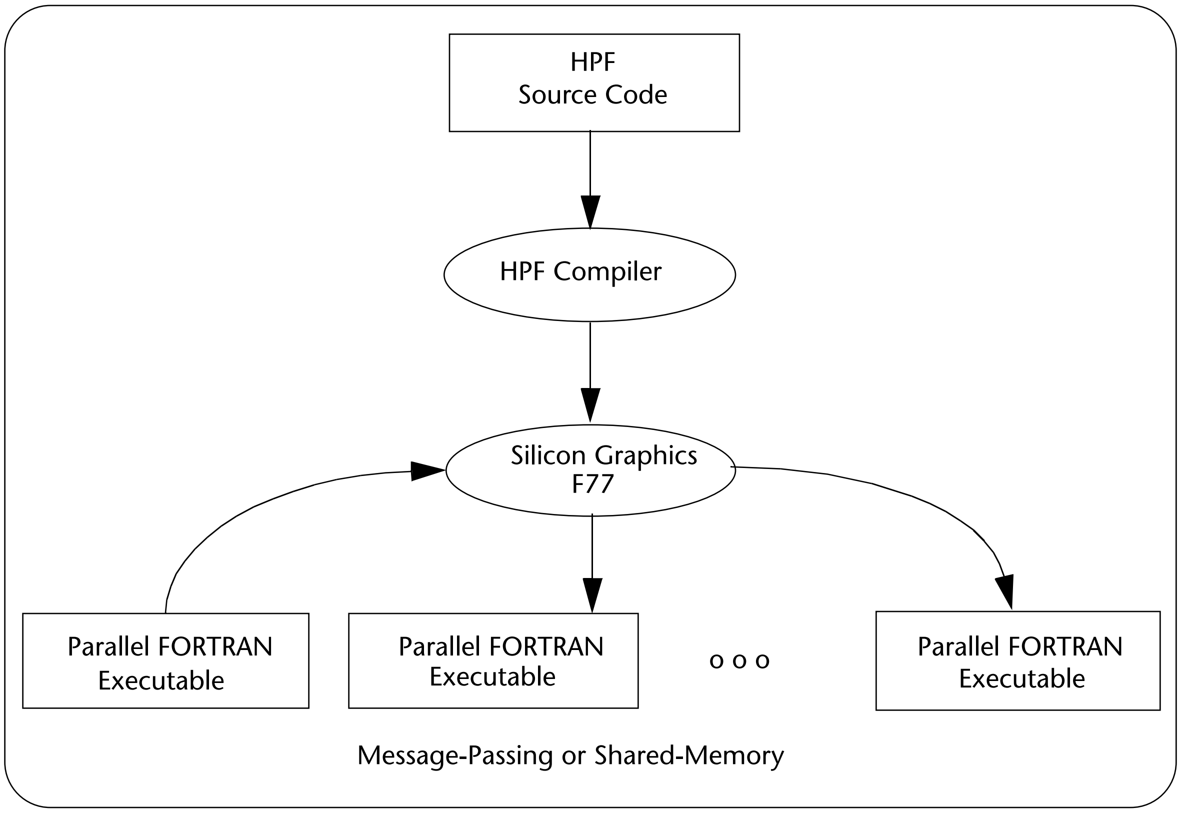


Figure 3-9 HPF Source —Parallel FORTRAN Executables

PGHPF

An HPF compiler for Silicon Graphics systems is available from The Portland Group, Inc. (PGI). Features supported by the Portland Group's HPF compiler, *PGHPF*, include the following:

- ❑ Integration with the Silicon Graphics MIPSpro FORTRAN compilation system for easy compile-and-go usage
- ❑ Fully compliant to ANSI FORTRAN 77 standard
- ❑ MIL-STD-1753 features (DO WHILE, ENDDO, INCLUDE, and more)
- ❑ Full FORTRAN 90 array syntax
- ❑ Allocatable arrays
- ❑ Modules and the MODULE and USE statements
- ❑ Array constructors
- ❑ KIND parameters in type declarations
- ❑ KIND specifier in literal constants
- ❑ Non-advancing and namelist I/O
- ❑ Free-form source
- ❑ All FORTRAN 90 intrinsics, millisecond resolution on SYSTEM_CLOCK
- ❑ All HPF intrinsics
- ❑ Multi-D ALIGN, DISTRIBUTE, TEMPLATE, and PROCESSORS support
- ❑ Fully general BLOCK, CYCLIC, and CYCLIC(K) distributions
- ❑ All of the HPF_LIBRARY module
- ❑ The HPF FORALL construct
- ❑ Calls to FORTRAN 77 local routines using EXTRINSIC (F77_LOCAL)
- ❑ Full support for the INHERIT directive
- ❑ Executables which run on an arbitrary number of processors

Interprocessor communications and synchronizations are performed using a runtime library optimized for hybrid-shared/distributed-memory parallel systems. Communications between CPUs inside a POWERnode use the shared-memory MPI model, and communications between CPUs on diff-erent POWERnodes are based on the Message-Passing MPI model.

xHPF

Applied Parallel Research's HPF pre-compiler, xHPF, is available for the Silicon Graphics POWER CHALLENGEarray systems. Its main features include:

- APR's FORGE® Magic™ technology to automatically parallelize FORTRAN 77 into an HPF program
- HPF Program consistency checking
- Parallelization of array assignments, FORALL, and DO loops
- Parallel runtime performance analysis for locating interprocessor communication bottlenecks and load balancing problems

The xHPF precompiler utilizes a preprocessor to lower the FORTRAN 90 constructs in an HPF Subset program to standard FORTRAN 77. The second pass of xHPF converts this into a SPMD parallelized FORTRAN 77 program with calls to APR's runtime parallel library that acts as an interface to a number of common message-passing libraries.

Because local and global consistency of HPF directives versus program con-text is critical, xHPF includes a special pass that checks directives against the static analysis of the program and issues diagnostics for illegally or in-consistently partitioned arrays. It ensures that HPF directives it finds are valid. In its automatic parallelization mode, xHPF converts a serial FORTRAN 77 program into an HPF program, utilizing APR's Magic tech-nology. For POWER CHALLENGEarray, the generated code facilitates the distribution of data across the clusters of processors while using shared-memory directives for the best performance of each cluster.

Interprocessor communications and synchronizations are performed using a runtime library optimized for hybrid-shared/distributed-memory parallel systems. xHPF uses the Hybrid communication model that combines both shared-memory and message-passing programming models.

Program Visualization: Upshot & XPVM

There are two popular parallel program visualization tools available for the POWER CHALLENGEarray systems:

- Upshot for parallel programs written using MPI
- XPVM for parallel programs written using PVM

Upshot

Upshot is a trace analysis and visualization package developed at Argonne National Laboratory for message-passing systems: In particular, trace events can be generated automatically by using an instrumented version of MPI. Alternatively, the programmer can insert event-logging calls man-ually. Upshot's display tools are designed for the visualization and analysis of state data derived from logged events. A state is defined by starting and ending events. For instance, an instrumented collective communication routine can generate two separate events on each processor to indicate when the processor entered and exited the routine. The Upshot Gantt chart display shows the state of each processor as a function of time. States can be nested, thereby allowing multiple levels of detail to be captured in a single display. States can be defined either in an input file or interactively during visualization.

XPVM

XPVM is a graphical console and monitor for PVM. It provides a graphical interface to the PVM console commands and information, along with sev-eral animated views to monitor the execution of PVM programs. These views provide information about the interactions among tasks in a parallel PVM program to assist in performance tuning and debugging. The XPVM monitor views include the Network View, Space Time View, and the Utilization View. XPVM provides basic debugging assistance via the Call Trace View and the Task Output View. XPVM works with PVM 3.3 or later, which is instrumented to capture tracing information at runtime. Then, any task spawned from XPVM will return trace event information for anal-ysis in real time or for postmortem playback from saved trace files.

Additionally, Silicon Graphics Performance Co-Pilot (PCP) graphical sys-tem-monitoring tool provides program visualization through its *procvis* view, which displays the entire set (or a subset) of processes that can be identified by a global ASH. The section on distributed system adminis-tration discusses PCP in detail.

Distributed Batch Processing and Load Balancing Tools

Distributed batch processing allows users to efficiently and transparently utilize all the computational resources in a distributed parallel-throughput processing environment such as POWER CHALLENGEarray. These tools match the computational requirements of different applications with the capabilities of different resources in the environment. Thus, applications that were previously run in a truly “waiting-for-my-turn” batch mode on traditional supercomputers can now be submitted from workstations, X terminals, or other clients to the POWER CHALLENGEarray environment for immediate execution, depending on available resources. This results in a great increase in overall job throughput. POWER CHALLENGEarray can be used as a powerful distributed throughput engine for serial jobs and for parallel jobs employing different levels of parallelism, while providing a single system image to users and operators.

Figure 3-10 on page 52 illustrates the batch-queuing paradigm on POWER CHALLENGEarray.

Features of a Distributed Batch Processing System

A distributed parallel-throughput environment supports a mixture of se-quential and parallel jobs. This means that the distributed batch process-ing systems should have the capacity to facilitate transparent, dynamic, and intelligent distribution of batch, interactive, parallel jobs for maxi-mum throughput performance. Balancing computation of large parallel jobs across different processors in the environment is especially important to ensure that all parallel tasks are completed at the same time. This feature is crucial for good parallel speedup.

Features of a versatile batch processing system include:

- ❑ A scalable, distributed batch queueing system for spawning jobs on all available processor resources
- ❑ A load-balanced interactive environment
- ❑ Support for single and multiprocessor systems
- ❑ Dynamic, real-time load balancing
- ❑ Capacity to monitor a wide variety of system parameters, including CPU load, paging rate, swap space, memory, interactive I/O, number of logins, and disk space
- ❑ Fault tolerance
- ❑ No modification to the operating system or existing applications required
- ❑ Efficient interactive I/O when interactive jobs are executed remotely
- ❑ Graphical user interface (GUI) for configuration management
- ❑ Job checkpointing and restarting
- ❑ Process migration facility
- ❑ Exclusive use of processing resources
- ❑ Resource allocation on a per-user or group basis
- ❑ Priority scheduling of batch jobs
- ❑ Time-of-day-sensitive host usage
- ❑ Run-time limits for batch jobs
- ❑ A secure environment

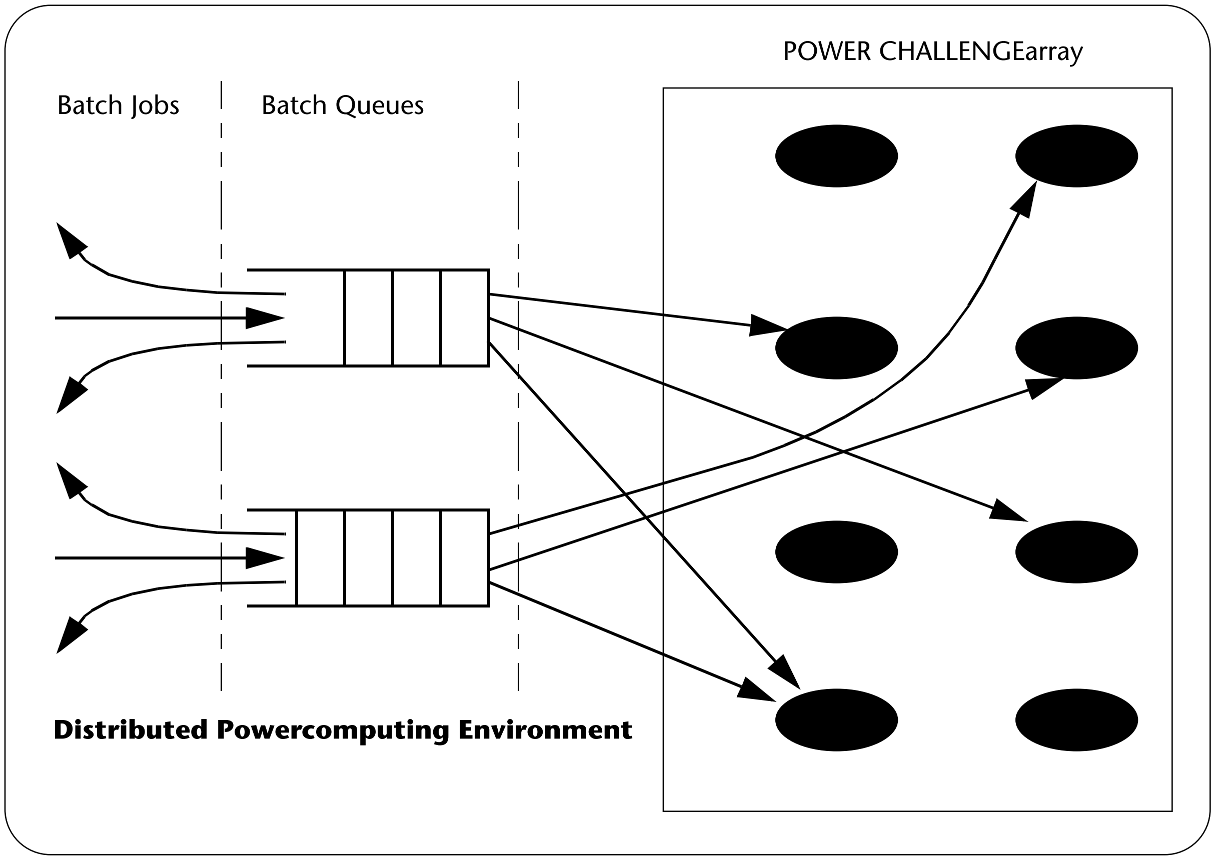


Figure 3-10 Distributed Batch Processing on POWER CHALLENGEarray

Load Balancing and Distributed Batch Processing Tools

Load Sharing Facility (LSF) and CODINE

Load Sharing Facility (LSF) from Platform Computing Corporation, Toronto, Canada, and CODINE from Genias Software GmbH, Germany, are two prominent tools that support batch queuing of serial and parallel jobs on Silicon Graphics POWER CHALLENGEarray and POWER CHALLENGE systems. These also provide load sharing across multiple POWER CHALLENGE systems in the network. They seamlessly integrate all existing and new systems, workstations, and servers, thus easing the introduction of new systems. Their parallel applications and multiprocessor support en-sure optimal utilization of distributed computing resources. The sophisti-cated scheduling and control built into these tools enable the implemen-tation of site-specific policies for resource sharing.

Interactive, Batch, and Throughput Processing Support

These batch schedulers log detailed resource consumption data for each interactive and batch job across POWER CHALLENGEarray, including all UNIX resource info: CPU time, memory size, I/O, swap space, and more. Such records also give a complete list of jobs processing in the system—by whom, when, job name and parameters, start and end time, number of processors used, and more. These files can be used for accounting and aud-iting purposes. These can be configured to set resource consumption limits on jobs submitted to each queue to prevent abuse or run away jobs. Limits can be set by the user as well. Access to the queues can be restricted to cer-tain users or machines to discriminate among users and groups.

NQS Interoperability

These tools provide an NQS-compatible command interface. The NQS user commands *qsub*, *qdel*, and *qstat*, are provided. Thus, NQS batch script files can be submitted in order to submit, control and check jobs. These three commands are the user commands in NQS. The administrative commands provided by the batch tools are more than compatible with NQS...they are much more extensive—most things done in NQS can be accomplished by using these tools, but sometimes in different ways. Both tools interoperate with any NQS systems. Jobs can be submitted to these and automatically routed to NQS systems for execution. The tools also extend NQS capabil-ities to distributed systems.

Configuration and Authentication Options

The batch tools support, through configuration options, a variety of auth-entication mechanisms, including UNIX setuid, identd daemon, and Kerberos. Table 3-1 compares the feature list of both LSF and CODINE.

Features	Load Sharing Facility (LSF)	CODINE
Batch Processing	YES	YES
Load Balancing	YES	YES
Interactive Support	YES	YES
Transparent Interactive Remote Execution	YES	YES
NQS Interoperability	YES	YES
Shared-Memory Parallel Job Scheduling	YES	YES
PVM Application Support	YES	YES
MPI Application Support	YES	YES
Non-Shared File System Support	YES	YES
CheckPointing Support	YES	YES
Process Migration	YES	YES
Job Run-Time Limits	YES	YES
Job Relinking Required?	NO	NO
Fault Tolerant (No Single Point of Failure)	YES	YES
Set Execution Time and Date	YES	YES
Job Submission File	YES	YES
Withdraw Machine at Will	YES	YES
Return of Resources Used	YES	YES
User Account Required	YES	YES
Time of Day Sensitive	YES	YES
Input/Output Redirection	YES	YES
Calendar-Driven Job Scheduling	YES	YES
File Event Detection for Scheduling	YES	YES

Table 3-1 Load Balancing and Batch Processing Tools: Feature Comparison Table

Features	Load Sharing Facility (LSF)	CODINE
Support for Job Dependencies	YES	YES
Support for IRIX Processor Sets (pset)	YES	NO
Processor Limit for Jobs	YES	YES
Load Thresholds for Preemptive Job Scheduling	YES	YES
Job Limits per Queue, Processor, User	YES	YES
Fully Configurable Load Indices for Scheduling	YES	YES
Job Resource Requirements	YES	YES
WAN Support	YES	YES
Job Accounting with Analysis Tool	YES	YES
Project ID & Name for Job Accounting	YES	YES
ARRAY Session Support	YES	YES
Job Notification Mail	YES	YES
Access to Files Where Submitted	YES	YES
Query Job Status	YES	YES
FairShare Policy Scheduling	YES	YES
DCE Support	YES	YES
Andrew File System (AFS) Support	YES	YES
Distributed File System (DFS) Support	NO	YES
Kerberos Support	YES	YES
POSIX P1003.15/D12 Compliant GUI	NO	YES
Japanese Language Support	YES	NO
Floating Software License Checking	YES	YES

Table 3-1 Load Balancing and Batch Processing Tools: Feature Comparison Table

Distributed System Management Tools

On a large distributed system such as POWER CHALLENGEarray, adminis-tering and managing system resources can be a difficult task, and solutions are necessary to help the system administrator. Several solutions are provi-ded on POWER CHALLENGEarray for this purpose.

IRISconsole

POWER CHALLENGEarray comes with IRISconsole, comprising an Indy workstation, a serial port multiplexor, serial and SCSI cables, software to manage serial port connectivity, and a text-based interface to manage a cluster remotely. The Indy workstation manages and monitors the activity of POWER CHALLENGEarray, then stores this information. IRISconsole uses an easy-to-use configurable graphical interface, allowing you to simply click a button to perform tasks such as resetting a system or generating a Non-Maskable Interrupt (NMI), which forces a system to generate a corefile for debugging purposes.

IRISconsole can monitor up to 16 POWERnodes and provide information such as:

- Voltage levels of the power supply
- Operating temperature
- Speed of the internal blowers
- Availability report of the servers
- System log
- Console activity by other users
- Hardware inventory of any machine

Each configured system is presented as a window on the screen. Transact-ions in the window can be optionally logged; each window has scrollable history. All viewable graphs can be saved to a file or a PostScript format for interchange across a network. IRISconsole employs its own password-based security system, allowing the system administrator to securely configure POWER CHALLENGEarray.

IRIXPro

Silicon Graphics IRIXPro is a layered product that simplifies the administration of POWER CHALLENGEarray from an IRISconsole system. IRIXPro contains two tools that are useful for administering an array of systems, Propel and Provision.

Propel

Propel is a configuration file management system. It allows an administra-tor to centrally maintain any file, then distribute it in regular intervals. It also supports multiple-administrator administration by moving standard IRIX configuration file information into a database system with *locking* and *infinite undo*. Propel allows for systems to be arbitrarily grouped into collections for the ease of administration.

Provision

Provision is a remote-monitoring facility which allows collection of vir-tually any data source available, including SNMP, Sun RPC, ICMP, Silicon Graphics Performance Co-Pilot, and the Silicon Graphics Objectserver. Information may be graphed in real-time on the IRISconsole system, then logged for later playback, or used to drive a configurable set of actions. Collection can be done for a host or an arbitrary collection of systems.

The interfaces to both of these tools are written entirely in the scripting language *Tcl* so that they can be customized to any site. Addition of data-base classes or attributes can be simply added to the Propel editors, and alternative protocols or actions can be added to Provision.

IRIXPro also includes a ProDev/Tracker problem tracking system, and a Dynamic Host Configuration Protocol server. These are useful when man-aging large numbers of workstations and users.

Performance Co-Pilot (PCP)

Performance Co-Pilot (PCP) is a system visualization graphical tool from Silicon Graphics for monitoring, visualizing, and managing systems perfor-mance. PCP is designed for the in-depth analysis and sophisticated control mechanisms that are needed to understand and manage the hardest performance problems in most complex systems, including POWER CHALLENGEarray.

PCP provides a systems-level suite of tools that cooperate to deliver distrib-uted, integrated performance management services. It has a distributed client-server architecture, with performance data collected and exported from multiple sources, most notably the IRIX kernel, DBMS products, layered services (such as WWW and NNTP servers, print spoolers, mail agents), and end-user applications.

PCP is targeted at the performance analyst, benchmarker, engineering dev-eloper, database administrator, capacity planner, or system administrator with an interest in overall system performance. It provides the capability to quickly isolate and understand performance behavior, resource utiliz-ation, activity levels, and performance bottlenecks.

Dealing efficiently with the dynamic behavior of complex systems requires services to filter noise from the stream of performance data, allowing the performance manager to concentrate on exceptional scenarios. The ability to review previous performance data, performance visualization, and the automated reasoning about performance data, is a key technique suppor-ted in PCP to provide the necessary high-bandwidth filtering.

From the PCP enduser's perspective, PCP presents an integrated suite of tools, user interfaces and services that support real-time and retrospective performance analysis. PCP focuses attention on the exceptional and extra-ordinary performance behavior. The user can concentrate upon in-depth analysis or target management procedures for the critical system perform-ance problems.

PCP has been customized for POWER CHALLENGEarray to provide visuali-zation of system-level and job-level statistics for the POWERnodes across the array. The client portion of PCP can be run on IRIX 5.3 as well as IRIX 6.1, so an IRISconsole can be used as a visualization client across the array. PCP can be used both for playing back trace data or for on-line perform-ance monitoring.

PCP Utilities

An array user can view a variety of relevant performance metrics on the array via the following PCP utilities:

- *arrayvis*: to visualize aggregate POWER CHALLENGEarray performance
- *procvis*: to visualize CPU utilization across an array for tasks belonging to a particular global ASH
- *mpvis*: to visualize CPU utilization of a POWERnode
- *dkvis*: to visualize disk I/O rates on a POWERnode

- *nfsvis*: to visualize NSF statistics on a POWERnode
- *pmchart*: to plot general performance metrics vs. time on a POWERnode

Array Diagnostics

POWER CHALLENGEarray comprises many layered hardware and software components. If any of these layered components are misconfigured or faulty, the functionality of the array can be significantly reduced. In some cases, the array can be rendered completely inoperable. POWERnode soft-ware includes an array diagnostics package that eases the process of fault diagnosis and recovery by verifying the integrity of a variety of crucial hardware and software components. Although the primary purpose of array diagnostics is to ease the installation and configuration of new POWER CHALLENGEarray systems, these may also be helpful in diag-nosing problems on existing arrays.

The array components tested by the diagnostics package include:

- ❑ HiPPI
- ❑ IP network configuration (primary network and HiPPI)
- ❑ Overall system configuration
- ❑ Array services
- ❑ MPI functionality
- ❑ PVM functionality

Processor Segmentation

Processor segmentation, a value-added feature of the IRIX operating sys-tem, provides a simple and flexible way to partition the available process-ors in a host to ensure inter/intra-departmental sharing of processor re-sources in a fair way. When segmented, each processor set can be used as a shared or dedicated resource among different groups, users, departments, or projects. Figure 3-11 illustrates partitioning of a 10-processor system in-to four processor sets. Each set can be used as a shared or dedicated resource among different users, groups, projects, or departments.

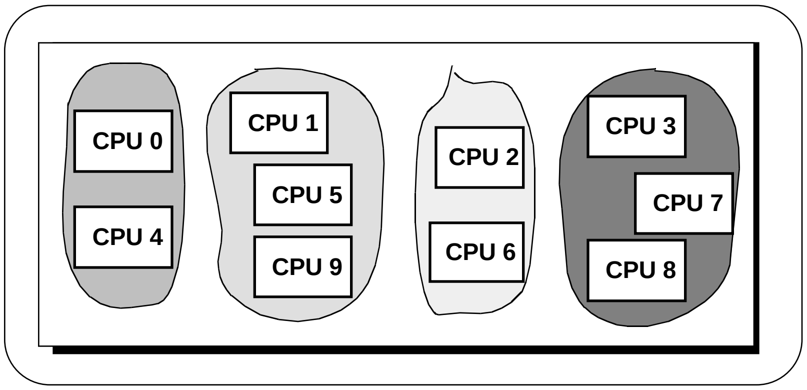


Figure 3-11 Processor Segmentation

Job and Session Accounting

Chargeback accounting is an integral tool for managing resource usage in a high-throughput computing environment. The chargeback accounting and resource control feature for POWER CHALLENGEarray systems supports re-source control and accounting at the process level, session level, POWERnode level, and POWER CHALLENGEarray level. Chargeback accounting is based on "actual-cost" accounting for each array session, user, group, project, and cost center, or "proportional" chargeback and billing on a project or department level.

The IRIX system accounting package is called PerfAcct. It collects system accounting data for all resources utilized on a POWER CHALLENGEarray, and brings it to a central location where it automatically summarizes the data and creates reports and bills. These may be created by a variety of keys, including machine, group, user, project, session, and shift. A new "session accounting" feature provides true job accounting, allowing auto-matic chargeback of usage by any kind of job. PerfAcct features low-over-head data collection on the systems being monitored. It also has a graphic-al user interface for ad hoc on-line queries and for designing automatic re-ports and bills.

This accounting tool is ideally suited for a high-throughput environment characterized by a diversity of jobs executing on the system. Resources tracked include memory, CPU usage (user and system usage can be tracked at global schedule intervals), disk usage, file usage, detailed file I/O (buf-fered, direct, and count of requests), nice value, buffered and direct I/O wait times, system usage and priority, number of processes (tracked at each request), and total system CPU allocation. Figure 3-12 demonstrates the basic components of the accounting tool.

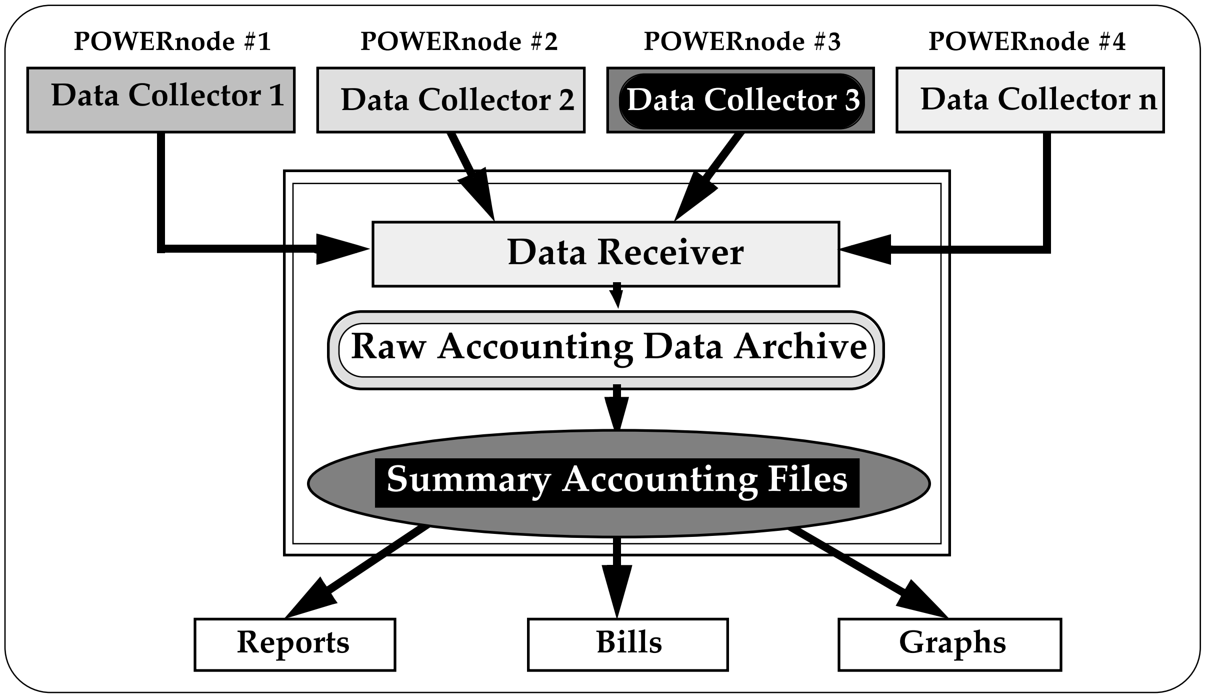


Figure 3-12 Chargeback Accounting on POWER CHALLENGEarray

Fair-Share Scheduling

SHARE II™ is an optional scheduler that allows users to be grouped into an arbitrarily defined resource allocation and charging hierarchy. Within this structure, resource usage policy can be set according to organizational pri-orities. Resources that can be controlled, distributed and accurately monit-ored using SHARE II include CPU time, disk space, processes, system mem-ory, connect time logins, printer/plotter usage, and other user-definable resources. Renewable resources, such as CPU time and printer pages, are shared among the competing user population as the resources become available. Fixed quantity resources such as system memory and disk space are constrained to defined limits regardless of competition. In any event, when the “hard limit” is reached, further allocation of that resource is de-nied until the user releases some of it. Disk resources allow a further “soft limit” to warn users of their approaching “hard limit.”

SHARE II augments the coarse-grained resource usage information available from UNIX to accurately record the accumulated usage of all resources. This makes preparation of fine-grained, detailed reports on resource con-sumption possible. SHARE II actually refines the standard UNIX security mechanisms by allowing partial delegation of administrative power, while maintaining overall control. It can also manage user or group access to specified resources or applications under its control, even automatically varying access at different times. Under SHARE II, the system administrator (the super-user) can delegate policy setting and control of a group’s resource allocation to a subadministrator without granting full super-user privileges. In turn, these subadministrators may appoint other subadmin-istrators within their own groups so that resource allocation can be fine-tuned by those closest to work requirements. Individual subadministrators can only reassign resources up to the limit of their own group’s allocation. Overall system security and policy still resides with the super-user, always.

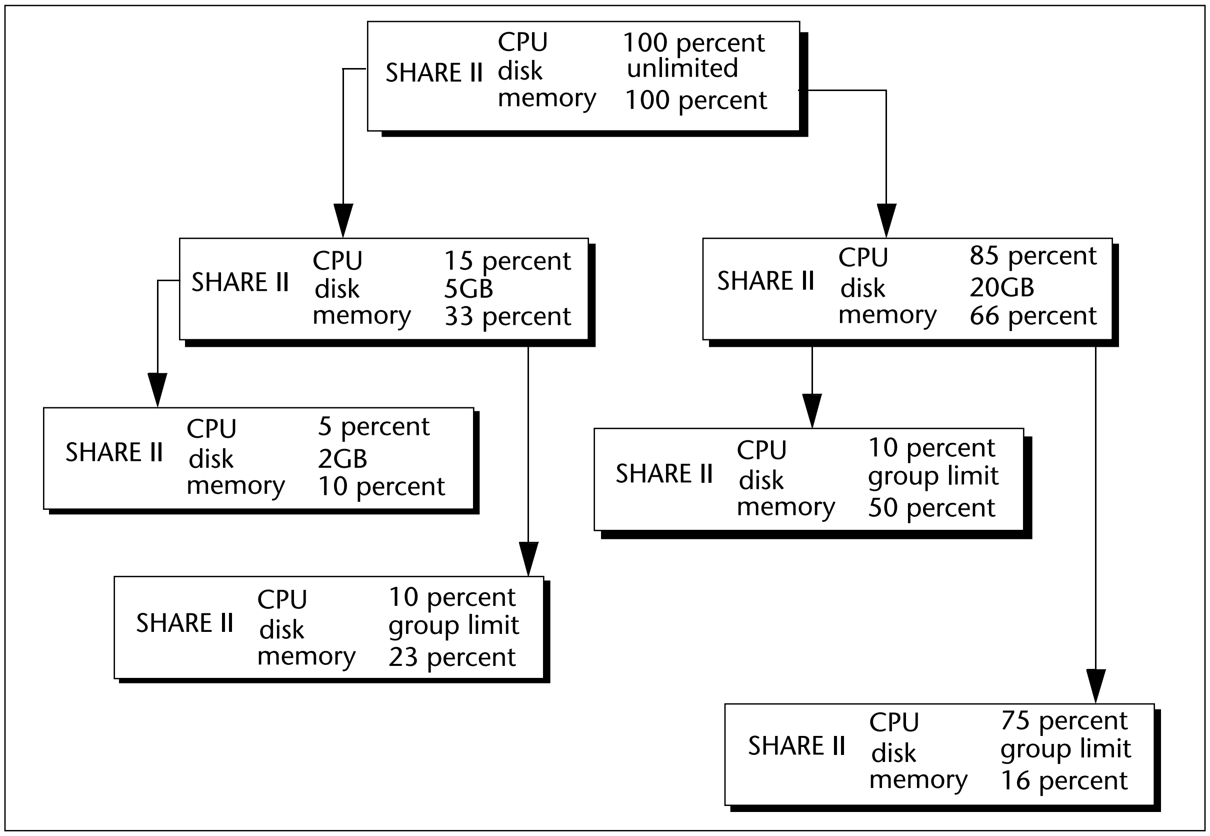


Figure 3-13 A SHARE II Configuration to Enforce Resource Usage Policy

Application Mapping & Case Studies

4

4.1 POWER CHALLENGEarray: Mapping Applications & Case Studies

Algorithmic Issues

Many applications have been successfully parallelized for POWER CHALLENGEarray platforms. POWER CHALLENGEarray exploits a comm-unication hierarchy, with a fast, shared-bus interconnect for intra-POWERnode communication, a high-bandwidth HiPPI interconnect for inter-POWERnode communication, and a final TCP/IP network layer for wide-area metacomputing. As with memory hierarchies, resultant appli-cation performance depends on sustained bandwidth performance, latency avoidance, latency tolerance, and reduced intertask communication. By maximizing data locality, processors can frequently exchange data through the fast layers of the communication hierarchy, avoiding the slower layers. Remaining communication latency can be tolerated by overlapping useful computation with data transfer and by moving large volumes of data at high bandwidth.

The combined benefits of shared and distributed memory provided by this architecture make it a good candidate for a wide range of parallel applica-tions. Parallel tasks residing within the same POWERnode can communi-cate via shared memory, exploiting the high bandwidth and low latency of the POWERpath-2 system bus. Such shared-memory communication can be explicit with direct references to shared-memory locations, or implicit using industry-standard communication libraries such as MPI, PVM, or HPF. Parallel tasks residing on distinct POWERnodes can communicate via the high-bandwidth HiPPI network. Inter-POWERnode communication can be implemented using sockets directly or via communication routines provided by MPI, PVM, or HPF.

Many general-purpose scientific and engineering applications require a moderate number of processors and are good candidates for running in parallel on a single POWERnode of POWER CHALLENGEarray. Since a typ-ical customer environment will consist of several such applications, each with varying resource requirements, POWER CHALLENGEarray serves as a good throughput engine that maximizes the throughput for such a set of applications.

However, an interesting set of applications, typically from the grand challenge-class of problems, can scale to a large number of processors. These applications can span multiple POWERnodes, and need to be archi-tected to take advantage of the most general-purpose hybrid communi-cation model. The following list discusses some of the issues that are important in designing multiple-POWERnode parallel applications for POWER CHALLENGEarray, with multiple threads or tasks within each POWERnode. Many existing parallel algorithms will need to be increment-ally modified to exploit POWER CHALLENGEarray architecture to achieve good speedups. Several approaches to modify these algorithms are discuss-ed below. Ultimately, the method of choice for any particular application will depend on the computation and communication characteristics of the algorithms used.

Task Granularity

The communication-to-computation ratio is one of the most important factors in determining the performance of a parallel algorithm. Communi-cation is characterized by the frequency and amount of data communica-ted between the parallel tasks on distinct POWERnodes. A task’s granular-ity is determined by the amount of computation it executes between com-munication steps, with a large granularity referring to large amounts of computation between tasks. Applications that can be decomposed into tasks with a large granularity are good candidates for the POWER CHALLENGEarray platform. Often, the frequency of intertask communi-cations in an algorithm can be reduced by lumping two or more of these exchanges together and sending a few big messages. This amortizes the communication overhead associated with data transfer between POWERnodes, thus tolerating a higher network latency.

Furthermore, the resulting larger messages also make better use of the effective bandwidth of the underlying communication layer. Figure 4-1 illustrates avoiding latency through message aggregation.

Figure 4-1 Avoiding Latency Through Message Aggregation

Additionally, the number of inter-POWERnode data exchanges can be re-duced by creating overlapping buffer zones on each POWERnode and per-forming redundant computation as an alternative to more frequent message exchange. That is, the problem is partitioned into overlapping subproblems, so that the work divided among the tasks is not disjoint— part of the work done on a POWERnode is the same task that other POWERnodes would have needed to do. This allows a POWERnode to communicate less frequently with other POWERnodes, at the expense of some extra computation.

Figure 4-2 on page 68 illustrates this with a problem where tasks need to communicate only boundary region data with their neighboring tasks. There are nine tasks, with the solid lines showing a disjoint work partition between tasks, the dotted lines showing an overlapping work partition, and the shaded region showing the overlapping regions for one of the tasks.

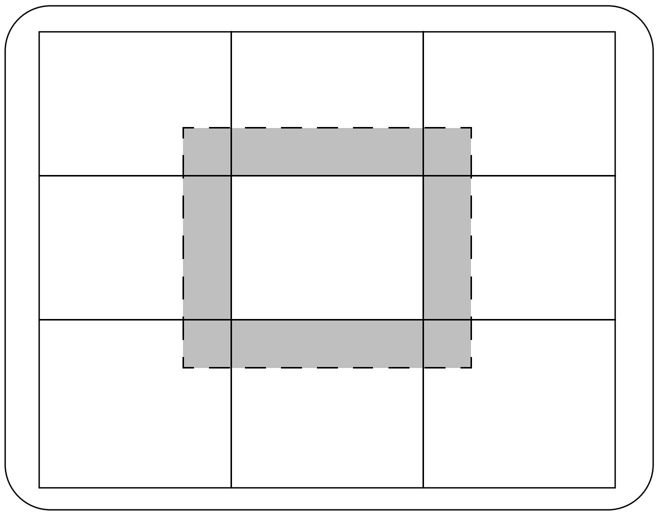


Figure 4-2 Overlapping Buffer Zones

Many applications that require little intertask communication are ideal candidates to be solved on POWER CHALLENGEarray. A classic example of such a computation is a problem that involves many independent simu-lations. POWER CHALLENGEarray architecture offers several advantages over other architectures, such as a single CPU/node message-passing mach-ine, even for these problems. These advantages include better load-balan-cing capabilities, higher I/O performance, and less stringent memory requirements.

Problem Decomposition

Decomposition of a problem into parallel tasks to be executed on potent-ially distinct POWERnodes is another key factor in determining its per-formance, and is closely related to task granularity. Since HiPPI network latency is higher than the memory latency within a single POWERnode, and the HiPPI network bandwidth is lower than the intra-POWERnode bandwidth, tasks must reside semipermanently on a POWERnode. These cannot be assigned to the first node to become available, but they can only migrate incrementally. This requirement is satisfied by domain decomp-osition, where the problem domain is divided among the tasks so that each

POWERnode works on one submatrix, subgrid, cluster of data ele-ments, or subarray. Domain decomposition typically provides tasks with very large granularity and allows tasks to have maximum data locality and data reuse.

In applications for which the result at a point is dependent upon data values at neighboring points, domain decomposition results in the majority of intertask communications being local to tasks on a POWERnode.

Locality of computation resulting from domain decomposition is necessary to limit the amount of communication required to support the parallel computation. This locality of computation also results in better locality of reference to the data, thus using the memory hierarchy more efficiently.

The low network bandwidth is accommodated because tasks on distinct POWERnodes only need to exchange data from the boundary regions of the subdomains. The required bandwidth is therefore reduced by the sur-face-to-volume factor. High network latency is accommodated by making these exchanges of boundary data infrequent, leading to large-grain tasks.

Overlapping Communication with Computation

Network latency can also be tolerated by overlapping communication with computation via asynchronous message-passing. Communication can be overlapped with computation by sending data from a task to a task on another POWERnode as soon as the data is ready. While underlying layers handle this data, the sender can resume its computation. Similarly, if the data that a task needs from another POWERnode has already been sent and received by the time it needs this data, it does not have to waste time waiting for this data, and can continue with its computations. Figure 4-3 on page 70 illustrates overlapping computation with communication.

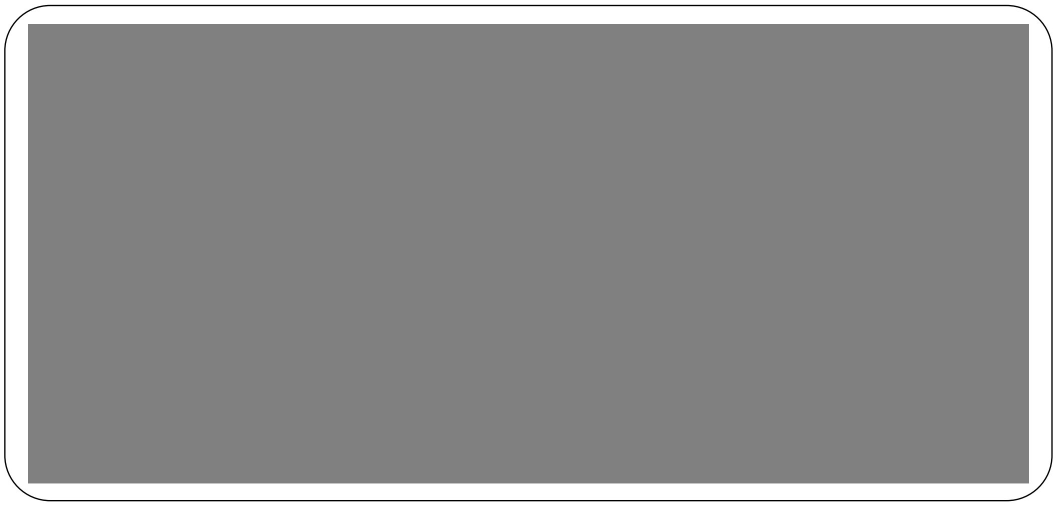


Figure 4-3 A Pipelined Algorithm

Intra-POWERnode Tasks

Intra-POWERnode tasks can either use the multiprocessing shared-memory parallel programming model directives provided by the compilers on the POWERnode architecture, or be coded as explicit message-passing tasks using either of the two message-passing libraries provided on POWERnode, MPI or PVM. These libraries exploit shared-memory primitives for inter-task communication within a POWERnode, at the same time providing the users with a more flexible programming paradigm. Alternatively, these applications could use HPF for which the details of intertask communi-cation are transparent to the application programmer.

Load Balancing

The combination of shared and distributed memory paradigms in POWER CHALLENGEarray provides maximum flexibility for irregular problems, both in terms of computational efficiency of the basic algorithm and for load balancing of component tasks. Load balancing is easier in an architec-ture such as POWER CHALLENGEarray because the number of nodes on the network is much less (currently a maximum of 16 POWERnodes) than a corresponding distributed-memory machine. Statistically, loads are more evenly balanced between these POWERnodes. Additionally, the overhead of load balancing is amortized much faster in the architecture of POWER CHALLENGEarray, since this overhead is spread across multiple CPUs (therefore, a larger computation chunk) of a POWERnode.

Memory Requirements

Memory requirements of an application tend to be lower in POWER CHALLENGEarray architecture as opposed to a pure distributed-memory machine, since processor-common data can be shared between the CPUs in a POWERnode.

Input/Output Requirements

Many applications have large I/O requirements, which can be satisfied by the combination of high performance XFS and NFS, version 3 filesystems provided on POWER CHALLENGEarray. The XFS file system is the local journaled filesystem on a POWERnode and can coexist with the EFS file-system. It provides extremely high I/O performance that scales well on the POWERnode multiprocessor systems. It is compatible with existing appli-cations and with NFS and can provide throughput in the range of 150-300MB per second. This, coupled with the augmented NFS Version 3, (which can provide throughput in the range of 10-15MB per second over the HiPPI network between POWERnodes), can provide significantly high I/O rates to applications.

In addition to all the issues discussed above, all single POWERnode optimizations will be very helpful in general. For example, data locality within a task can be further improved by rearranging the algorithmic steps such that nearby data elements are accessed together. Depending on the amount of data needed in a computation step, this can enable the entire data space needed for a computation step to all fit in the secondary cache of a POWERnode. Thus each POWERnode will need to reference main memory only infrequently. Reorganizing the algorithm so that a compu-tation step uses only stride-one arrays is an example of this technique. For a comprehensive discussion on single POWERnode and cache-based opti-mizations, please refer to the POWER CHALLENGE Tech Report.

Suitable Algorithm Characteristics

In addition to embarrassingly parallel algorithms, which require very little intertask communication, many other kinds of applications can be ported to POWER CHALLENGEarray platform successfully. In general, a larger amount of communication can be tolerated in a hierarchical communica-tion scheme provided by POWER CHALLENGEarray than a pure distrib-uted memory scheme, since the number of such distributed nodes is lower and the amount of computation per node is higher in the array. The algo-rithm characteristics that will facilitate this effort (of tolerating a higher network latency and a lower network bandwidth for inter-POWERnode communication) for an application include:

- The ability to divide the problem based on domain decomposition, such that tasks have maximum data locality and data reuse
- Resulting tasks have large granularity, either because of the decompo-sition method, or because the number of communications can be re-duced by lumping communication steps together
- The amount of data exchanged in a communication step is large
- Data exchange is only between boundary regions of subdomains, result-ing in a surface-to-volume ratio for communication to computation
- Data locality can be improved by rearranging the algorithmic steps
- Communication can be overlapped with computation to some extent

Applications on POWER CHALLENGEarray

Several applications from various fields have been parallelized for POWER CHALLENGEarray architecture, including those in computational fluid dynamics, computational structural mechanics, seismic modeling, chem-istry, operations research, and particle simulation. The following sections describe applications that have been run on the array from various fields.

PPM Hydrodynamics Code

One of the earliest examples of combining the benefits of shared and dis-tributed memory schemes offered by POWER CHALLENGEarray is the work performed at the University of Minnesota to solve a grand challenge problem in computational fluid dynamics on a cluster of Silicon Graphics CHALLENGE machines in September 1993. The goal was to perform the largest simulation of compressible fluid turbulence to date using the Piecewise-Parabolic Method hydrodynamics code. The simulation was per-formed on a grid of $1{,}024^3$ computational zones. The hardware consisted of 16 Silicon Graphics CHALLENGE XL servers, each with 20 100MHz, R4400™ CPUs, 1.75GB of memory, 12GB of local disk space and 3 FDDI interfaces to a 3-D toroidal network.

With 1,024^3 turbulence simulation and five 32-bit fluid state variables per computational zone, the memory required to store the primary data was 20GB. Factoring in local scratch storage and buffers for interprocessor communication, the amount of memory required for the problem was 28GB.

The problem was decomposed into 16 512 x 512 x 256 grid tasks, one each on the 16 nodes, with each task employing the MIPS compiler multiproc-essing directives within a node. The nodes were arranged in a 2 x 2 x 4 3D toroidal array, for a total of 20 toroidal rings.

Within a node, the 1.2GB/sec system bus provided a fast interconnection scheme. The communication bandwidth needed between the nodes was significantly reduced owing to a surface-to-volume effect; only data along the surfaces of the 3D subdomains updated by each POWERnode needed to be communicated to other machines. Additionally, the communication in each of the three coordinate dimensions could proceed simultaneously. Finally, the problem was structured in such a way that network commu-nications occurred only thrice in each computation time step, thus reducing the effects of communication latency.

The entire work space needed for updating a strip of 512 zones in a single 1D sweep consisted of stride-one arrays, all of which fit into the 1MB sec-ondary cache. Thus each main system memory was referenced only rarely, and these references were made to be predominantly stride-one to enhance cache performance. The global shared-memory of each node was exploited to perform relatively efficient transposes of the local data, and this allowed each CPU to operate on vectors of 512 or 256 zones.

The data filled 350 4GB Exabyte tapes. Including the 20 percent of the time spent in network communications, 4.9 GFLOPS sustained perform-ance was achieved. This was the first time general-purpose computers were used to solve grand challenge-class problems that were until then software only on special-purpose hardware.

Computational Structural Mechanics (CSM)/Computational Fluid Dynamics (CFD)

Much progress has been made in the parallelization of application codes in the fields of Computational Structural Mechanics (CSM) and Computation-al Fluid Dynamics (CFD). Many top commercial CSM application packages offer parallel solver technology on the POWER CHALLENGE platform. Similarly, many of the top CFD application packages are also available in parallel versions for POWER CHALLENGE. Until recently, most of the parallelization effort has been done using a fine-grain parallel approach. Much of the current state of the art concerning parallelization of these application types in the research community has focused on decompo-sition across processors in using a coarser grained approach, most often spatial domain decompositions.

On POWER CHALLENGEarray, the domain decomposition approach is more appropriate as it usually results in higher computation-to-communi-cation ratios. Fine-grain parallel approaches, relying only on very low com-munication latency, would be better suited for running on individual POWERnodes of POWER CHALLENGEarray. The more subdomains can be made independent of one another in the numerical technique employed, the greater the expected parallel efficiency will be. Many commercial CSM and CFD software packages are moving toward this coarser-grained parallel approach and there are already examples of packages using such tech-niques. For example, the RAMPANT™ product from Fluent®, Inc., utilizes a domain decomposition approach, and high parallel efficiency has been demonstrated across multiple nodes in POWER CHALLENGEarray. In RAMPANT, decomposition is performed by the application to maximize load balance across processors and to minimize communication between processors.

Seismic Modeling

Seismic modeling, preprocessing, and imaging algorithms typically contain parallelism at a variety of different levels. For example, in elastic modeling, processors on a per-source location basis or domain decomposition tech-niques allow all processors to work on the same problem. In preprocessing, processors can be assigned traces, gathers, records, or other groupings of data depending on the algorithm. In imaging, processors can be assigned frequency planes, time slices, blocks of image, or various types of gather.

The vast amount of parallelism available in these algorithms, coupled with the superior performance of MIPS R8000 on FFT’s, convolutions, and more, clearly indicates that POWER CHALLENGEarray is the ideal environment for production seismic processing as well as research. This environment is simple to work in as well as maintain, since the number of separate systems on the array is quite small. And, for the very large prestack imaging problems, or large-scale simulations, the full power of the array is available to simple distributed applications which need only concern themselves with communication among just a few very powerful nodes. The significant I/O requirements of such applications can be easily accom-modated with the powerful combination of the 64-bit XFS filesystem and extended NFS 3.

In May 1994, Silicon Graphics collaborated with Texaco to solve the world's largest 3D pre-stack depth migration problem on a 200-processor CHALLENGEarray system consisting of 10 CHALLENGE XL systems. Each CHALLENGE XL system consisted of 20 150MHz MIPS R4400 processors and 2GB of main memory. 3D pre-stack migration provides an accurate and highly resolved 3D image of the earth's interior derived from seismic recordings taken at the earth's surface. Improved imaging allows geoscien-tists at Texaco to identify smaller exploration targets with greater confi-dence, while improving the success of oil and gas exploration in difficult-to-find areas.

At Supercomputing ‘94, Silicon Graphics also demonstrated a large 3D poststack migration problem using the Hale-McClellan 3D poststack depth migration algorithm on POWER CHALLENGEarray. 3D poststack depth migration is used to analyze seismic data that is acquired on the earth's surface above an exploration target in order to improve the economics of oil and gas exploration and production.

The Hale-McClellan algorithm is the most popular algorithm for 3D post-stack depth migration because of its ability to accurately handle rapid lat-eral velocity changes. The initial data on the surface is extrapolated down-ward in depth using a one-way scalar wave equation, and imaged at each depth level corresponding to its energy at zero time. Hale-McClellan extra-polation is the recursive application of a 2D convolutional operator to the complex wave field followed by an interpolation using tabled coefficients dependent on the velocity model. This is applied independently to each temporal frequency. The imaging step is the summation of the real com-ponent of this field over all frequencies.

Each POWERnode of POWER CHALLENGEarray was assigned a group of frequencies, and migrated these frequencies in parallel. The algorithm used asynchronous disk I/O and asynchronous network communication between POWERnodes to completely hide the communication behind the computation. Network communication was performed using a simple socket-based message-passing library. As the migration proceeded and imaged new depth levels, the results were updated and viewed with an interactive volume renderer running on POWER Onyx. The migration was applied to a spatial volume of size 256 X 256 X 256. The relatively small spatial dimensions were chosen to demonstrate the ability to interactively interrogate the migration output, although the performance scales linearly for larger data sets. Furthermore, a very large number of frequencies (420) were chosen so that the total problem size was reasonably large; a single processor required 31,011 seconds to complete the job.

The POWER CHALLENGEarray system consisted of 11 POWERnodes, each with 1GB or 2GB of memory, up to 16GB of disk, and eight to 16 MIPS R8000/75Mhz processors, all connected through a HiPPI switch. The application was run on 84 processors and a speedup of 82.6 was observed, resulting in a sustained performance of more than 12.6 GFLOPS. The same problem when run on 60 MIPS R8000/75Mhz processors across eight POWERnodes realized a speedup of 58.9, resulting in 9 GFLOPS of sustained performance.

Operations Research

Many classes of operations research problems, such as the Traveling Sales-man Problem (TSP), have long captured the imagination of researchers in integer programming and discrete optimizations. It is easy to describe, yet exceedingly difficult to solve. Computational progress on the TSP has been responsible for much of the progress in the solution of general Mixed-Integer Programming (MIP), of which TSP is an example. Applications for this more general MIP model are unlimited, ranging from high-level capital budgeting through production planning and control to the design of computer chip layouts.

Mixed Integer Programming (MIP) problems are ideal candidates for POWER CHALLENGEarray. The main computation in MIP is typically ex-tremely coarse-grained, so the cost of exchanging data between nodes is minimal. At the same time, an effective MIP code requires some policy decisions, such as decisions about how to search a tree, to be made in a centralized fashion. The POWER CHALLENGEarray allows the distribution of the fine grained communication associated with these policy decisions thereby reducing the costs of centralized control.

In May 1994, Silicon Graphics collaborated with the Center for Research in Parallel Computing (CRPC) at Rice University, Rutgers University, Bellcore, and Bell Labs—known collectively as RCRBB—to solve the world's largest Traveling Salesman Problem ever (7,397 cities), on a CHALLENGEarray system consisting of 10 CHALLENGE XL systems. Each system consisted of 20 150MHz MIPS R4400 processors and 2GB of RAM. This is the most difficult discrete optimization problem ever solved.

Particle Simulation

PSiCM is a 3D Direct Simulation Monte Carlo (DSMC) code originally written in CM FORTRAN at NASA Ames Research Center, and has been converted to High-Performance FORTRAN and compiled using the pghpf HPF compiler from PGI. This HPF version of PSiCM has been successfully demonstrated on POWER CHALLENGEarray.

PSiCM presents several challenges to both a compiler and the underlying parallel hardware. It requires efficient integer sorting, permutation of sev-eral large 1K vectors using indirect array accesses, general data scatter oper-ations, segmented scan reduction operations on 1D vectors, and parallel random number generation. HPF supports many of these operations in the form of HPF library routines. Operations such as the permutations can be supported directly in the compiler given the capability for efficient parallel indirect array accesses.

PSiCM is used to simulate the flow of molecular nitrogen from a nozzle. Particle simulations are of great interest to space station designers, and can be used to analyze the effects on solar panels of plumes emitted from maneuvering jets during space shuttle docking. Complete simulations involving millions of particles typically require many hours of computation.

High-Throughput Environment 5

5.1 Introduction

Engineers and managers often base their selection of large, high-perform-ance computing systems on the performance of a few jobs. Many factors contribute to the proper configuration. Among these are:

- ❑ Performance of key large applications
- ❑ Basic system capacities:
  - ◦ Memory
  - ◦ I/O capacity
  - ◦ Disk space
- ❑ User productivity features
- ❑ Average batch system job turnaround time
- ❑ Interactive workload capacities

The first three factors are typical measurements for success. Compared with the first three elements, how to properly configure for job throughput and interactive workloads are often much less systematically addressed. These five areas often compete for the same budget resources and lead to differ-ent configuration needs. The relative importance of these categories will also vary from computer site to computer site. Here we explore some of these issues and how they relate to the large computer site with many applications, many users, and expensive, heavily-used systems.

Historically, and under controlled conditions, system-wide measurements of throughput have been difficult to obtain. It is often too expensive to dedicate large compute resources for an extended time to perform such an evaluation. Similarly, porting and tuning a large number of codes from scratch is labor-intensive and expensive. Understanding the costs and ben-efits of moving applications from one computer to another will be helpful in guiding the procurement process.

Below are the results of a real-world porting and tuning exercise involving a large number of codes to be installed at multiple large-scale compute facilities in North America. These results indicate several unique advan-tages of the POWER CHALLENGEarray in large sites. In addition, exam-ining throughput as a function of architecture reveals powerful economic advantages to using shared-memory building blocks (POWERnodes) in the high-throughput environment, as well as identifying practical recipes to optimally configure a system.

The Benchmark Suite

This paper is based on runs performed on a POWER CHALLENGEarray system. Executions were performed to obtain single-job timing as well as large throughput time mixes. All test cases reflect real-world scenarios typical of job mixes found at existing sites.

The benchmark suite consisted of 28 different codes and/or data sets. Run mixes were generated from this list by choosing subsets of bench-marks and an iteration count for each. Timing exercises were included in the compilers. The benchmarks are described in Table 5-1 on page 81. As can be seen, the applications range across a wide span of scientific and engineering disciplines.

Virtually all benchmarks were developed for the Cray C-90 architecture. Some benchmarks included PVM versions of the application as an alter-native starting point. Altogether, the codes consist of about 800,000 lines of source code. The bulk of the applications are in Fortran 77, with a small amount of C code. Each code is substantially different from the others in CPU requirements, memory, and I/O requirements.

ID	Type	Memory(MB)	PVM
B-1	CEM	11	NO
B-2	CEM	12	NO
B-3	CEM	538	NO
B-4	CFD	620	YES
B-5	CFD	64	NO
B-6	CFD	551	NO
B-7	CFD	222	YES
B-8	CFD	482	NO
B-9	CFD	4000	NO
B-10	Chemistry	17	NO
B-11	Chemistry	8	NO
B-12	Chemistry	8	NO
B-13	Chemistry	48	NO
B-14	Chemistry	8	NO
B-15	Chemistry	88	NO
B-16	CFD	2000	YES
B-17	Image Processing	19	NO
B-18	Image Processing	19	NO
B-19	Image Processing	39	NO
B-20	Miscellaneous	8	NO
B-21	Reservoir Modelling	4	NO
B-22	Signal Processing	412	NO
B-23	Structural Explicit FEA	36	NO
B-24	Structural Explicit FEA	1400	NO
B-25	Structural Explicit FEA	83	NO
B-26	Structural Explicit FEA	70	NO
B-27	Structural Implicit FEA	391	NO
B-28	Structural Implicit FEA	168	NO

Table 5-1 Benchmark Codes/Datasets

The benchmarking process was completed in three stages. The first stage was a simple porting of all applications to get to a simple level of correct-ness. This effort focused on obtaining single-job timings on a single MIPS R8000/90MHz processor. For some codes this was not practical because these codes were written using PVM and had to be run in parallel. In these cases, single job timing was obtained by using four or eight CPUs.

The second stage was to tune the individual benchmarks for performance. The general performance goal was to maximize throughput in processor-sec for each benchmark. In general, the benchmarks were not parallelized deliberately, since parallelization would be detrimental to throughput, based on Amdahl’s Law. Nevertheless, some benchmarks were parallelized to meet minimum execution time targets.

In the third stage, the codes were combined into essentially arbitrary mixes. These were executed on various-sized POWER CHALLENGEarray systems to determine the array throughput. Each throughput test was constructed by executing a subset of benchmarks picked from the list of real world codes. Each test executed a specific number of copies of each code selected. Vendors were free to schedule the job mixes in any order desired.

Porting Team

The porting team was a pool of 10 programmers, with five active members at any given time. The group, most of whom were outside contractors, had limited Silicon Graphics and POWER CHALLENGEarray or POWER CHALLENGE experience. Only four had previous POWER CHALLENGE experience at all. Of these four, only two were available for the full term of the effort. In general, the porting team had a strong scientific and high-performance computing background. The team was typical of an in-house team organized for porting production codes to a new platform.

The team went from novices of Silicon Graphics compiler and operating systems technology to mature users. This knowledge transfer also included extensive work with various scientific libraries. In particular, they devel-oped extensive expertise in MPI and PVM message-passing applications.

System Configurations

The test array was a POWER CHALLENGEarray system configured with eight POWERnodes. Each was configured with 8GB of system memory, between 12 and 16 MIPS R8000/90MHz CPUs, and 64GB of user disk space. In general, enough memory was chosen for the job mix to avoid paging.

The I/O subsystem was a moderate configuration on each POWERnode, consisting of sixteen SCSI disks of 4.3GB capacity stripped 16 ways. The filesystem gave a sustained I/O transfer rate of more than 50MB/sec on single file I/O transactions using standard FORTRAN binary I/O.

Systems were interconnected via Ethernet and HiPPI interfaces. The HiPPI connection sustained raw I/O point-to-point performance in excess of 89 MB/sec. FTP transfers over TCP/IP were approximately 40MB/sec. Substan-tial network transactions were not a part of the test suite.

The Porting Experience

On average, approximately two days were required to port a code. This was followed by tuning that required an average of one week per code. Some codes required as many as three weeks. Some required no changes at all. Many of these changes resulted in several-fold speedups over unoptimized timings. Tuning efforts were cut short due to time restrictions. It is clear that there is still significant room for further benchmark performance improvement.

Since all code work had to be completed within two months, codes were selected for optimization based on their total expected load on the system. Thus, codes that consumed large CPU and memory resources were exam-ined closely. Codes that required few system resources were simply ported with no optimization effort.

Factors which entered into the allocation of tuning resources were based on overall weight of the benchmark in the expected mixes, the number of lines of code, and the perceived ease or difficulty of tuning. This approach is typical of porting efforts that might be done at a large, heavily loaded compute center. Time can be short, and key applications will get dispro-portionately large tuning resources based on various criteria.

Since the emphasis was on throughput, efforts at parallelization to reduce any individual job's execution time were generally not warranted. Parallel executions often suffer some inefficiencies in execution due to scalar sec-tions and parallel overhead and, as mentioned previously, leave through-put performance unchanged or even degraded. Parallelization was only performed to (a) get job run times below critical values and (b) actually improve code throughput performance because parallelizing led to a super-linear speedup (that is, two processors ran more than twice as fast due to better cache locality of reference).

Initially, optimized codes were compiled and executed one at a time. Wall clock timings for the compile and execution phases were obtained. Because of the trivial ease of using *parallel make*, the compile stages were often broadcast across more than one CPU to reduce the elapsed time. The maxi-mum number of CPUs was set to 8. This was normally reserved for the benchmarks with very large source trees. Most codes were compiled on one to four CPUs. No special coding was required to take advantage of this technique. The standard makefiles used under a normal UNIX *make* were executed under Silicon Graphics smake utility with no changes.

Early results of the study showed that POWER CHALLENGE is an excellent compile engine.The largest code of 250,000 lines of source compiled in just 383 seconds of elapsed wall clock time on eight CPUs. The same code compiled in just 259 seconds on 16 CPUs. This compile invoked all of the optimization flags available on the system, and was reflective of how a user would use the system in a real-world environment.

Single-Job Test Results

A summary of these first two stages appear in Table 5-2 on page 85. The third and fourth column are the initial execution times and the final opti-mized execution times, respectively; the fifth column shows the number of CPUs required for the optimized application; Cray C-90 single-CPU execu-tion times are presented in the sixth column; the last column provides the relative performance of the MIPS R8000/90MHz to the C-90 on a processor to processor basis. This C-90 equivalent performance is defined as:

$$\text{C-90 Equivalents} = \frac{\text{C-90 Time}}{\text{Optimized Time} \bullet \text{CPU per job}}$$

The tuning efforts of Silicon Graphics resulted in significant speedups. Typically, the team focuses on taking multiple three-dimensional DO loops in critical routines and combining them to form a single, larger 3D loop. This philosophy led to most improvements, including speedups ranging from 2-10x. Further improvement is possible.

Id	Type	Original Time	Tuned Time	CPUs	C-90 Time	C-90 Equivalents
B-1	CEM	103319	10217	1	3526	0.35
B-2	CEM	450	404	1	494	1.22
B-3	CEM	1690	877	1	536	0.61
B-4	CFD	8213	803	8	866	0.13
B-5	CFD	397	346	4	539	0.39
B-6	CFD	16866	3764	1	822	0.22
B-7	CFD	99098	6029	16	6333	0.07
B-8	CFD	32777	7157	1	1697	0.24
B-9	CFD	26851	515	1	1107	2.15
B-10	Chemistry	29898	352	2	195	0.28
B-11	Chemistry	20410	3963	1	7504	1.89
B-12	Chemistry	2516	542	1	420	0.78
B-13	Chemistry	17659	3699	1	5044	1.36
B-14	Chemistry	689	791	1	863	1.09
B-15	Chemistry	692	170	1	193	1.14
B-16	CFD	5952	1711	8	4619	0.34
B-17	Image Processing	271	52	1	40	0.76
B-18	Image Processing	4894	1998	1	501	0.25
B-19	Image Processing	293	135	1	192	1.43
B-20	Miscellaneous	35813	35813	12	1922	0.00
B-21	Reservoir Modelling	442	328	1	859	2.62
B-22	Signal Processing	15937	6523	1	805	0.12
B-23	Structural Explicit FEA	1164	326	1	430	1.32

Table 5-2 Single Job Test Results

Id	Type	Original Time	Tuned Time	CPUs	C-90 Time	C-90 Equivalents
B-24	Structural Explicit FEA	19196	3068	1	939	0.31
B-25	Structural Explicit FEA	15236	8597	1	3504	0.41
B-26	Structural Explicit FEA	3638	3257	1	780	0.24
B-27	Structural Explicit FEA	27044	2248	1	1604	0.71
B-28	Structural Explicit FEA	11030	5708	1	3011	0.53

Table 5-2 Single Job Test Results

Figure 5-1 on page 87 shows a histogram of the relative performance of the MIPS R8000 system with respect to a single processor C-90. In summary, POWERnode with the 90 MHz R8000 processor is a tremendous performer relative to the Cray C-90 CPU. Most codes execute at a significant fraction of the C-90 performance on a CPU to CPU basis. Several even exceed C-90 performance numbers.

Figure 5-1 also shows that the R8000 exceeds the performance of the C-90 for about 30 percent of the benchmarks. This is due to a substantial scalar fraction in some codes coupled with cache reuse on the R8000 processor. This behavior is common with many high-performance scientific appli-cations. The high levels of sustained performance in so many of these cases is clear evidence of the excellent architectural design of the POWERnode system. In particular, the compiler's ability to effectively exploit the super-scalar features of the R8000 processor contributed strongly to the high POWERnode performance relative to the C-90.

Figure 5-1 Relative Performance: C-90: MIPS R8000

In single-job performance, this histogram shows the excellent performance of a POWERnode over a broad application range. Even more, it offers excellent price/performance value. The average performance for this set of benchmarks shows the R8000/90MHz processor to be 70 percent of the C-90 on a single CPU to single CPU basis. Factor in the 20x price differential between the CPUs and the price/performance value is compelling.

Throughput Test Results

The team completed 24 throughput tests. These test cases were each comprised of an almost random selection of benchmarks from the list above, coupled to an almost random number of iterations of each.

The order of execution of the jobs was scheduled to maximize the total system throughput. This was accomplished by employing a job scheduler that used the single-job runtimes coupled with knowledge of disk and memory utilization of each job to most efficiently launch them into the system. This system worked well and total CPU use was more than 95 percent for all mixes tested. An example of how system resources are loaded for a typical run is shown in Figure 5-2 on page 90. These results were generated with the Performance Co-Pilot (PCP) profiling tool. Since the goal of the exercise was to provide maximum throughput for mini-mum budget, many key system resources were heavily loaded: processors, memory, and disk.

Table 5-3 on page 91 summarizes the results of the throughput tests. The first column contains the actual elapsed wall clock seconds for each mix. The second column contains the predicted elapsed wall clock time as de-termined by the scheduler. The scheduler included no model for multiple job overhead and memory contention on the system memory bus. The third column shows the efficiency obtained for real runs compared to the predicted times. A histogram of relative efficiency is shown in Figure 5-3 on page 92 for both the 12 processor runs and the 16 processor runs. The results show that the overall interference overhead for all cases is quite small. Worst case numbers are only a few percent from perfect.

Real-world throughput mixes are typically made up from many diverse codes. The test cases under consideration are a reflection of such an envir-onment. Based on actual run results we can say that such mixes will exe-cute extremely well on POWER CHALLENGEarray.

In summary, Figure 5-1 on page 87 shows dramatic proof that POWER CHALLENGE is a powerful performer in a heavily-loaded throughput environment. Aside from the extensive memory and CPU requirements, the codes executed also typically exhibit large I/O requirements as well. Many of the codes require multiple reads and writes to files in excess of 200MB. Some codes wrote files larger than 2GB. Some wrote files repeat-edly for total I/O counts in excess of 6GB.

Finally, Figure 5-2 on page 90 shows the excellent balanced architecture of POWER CHALLENGE in the throughput environment. Under heavy loads with all the processors very busy, the memory subsystem capacity is well matched to the memory demands of multiple simultaneous jobs. In no case did the overall performance of the system degrade by more than seven percent over the timing that would have been obtained on an ideal system with no code interference, despite tremendous activity on the system bus. In general, the measured result is within 2 percent of the predicted times. This is within the noise of the measurements as the actual run times of any code can vary on the order of 2 percent from run to run.

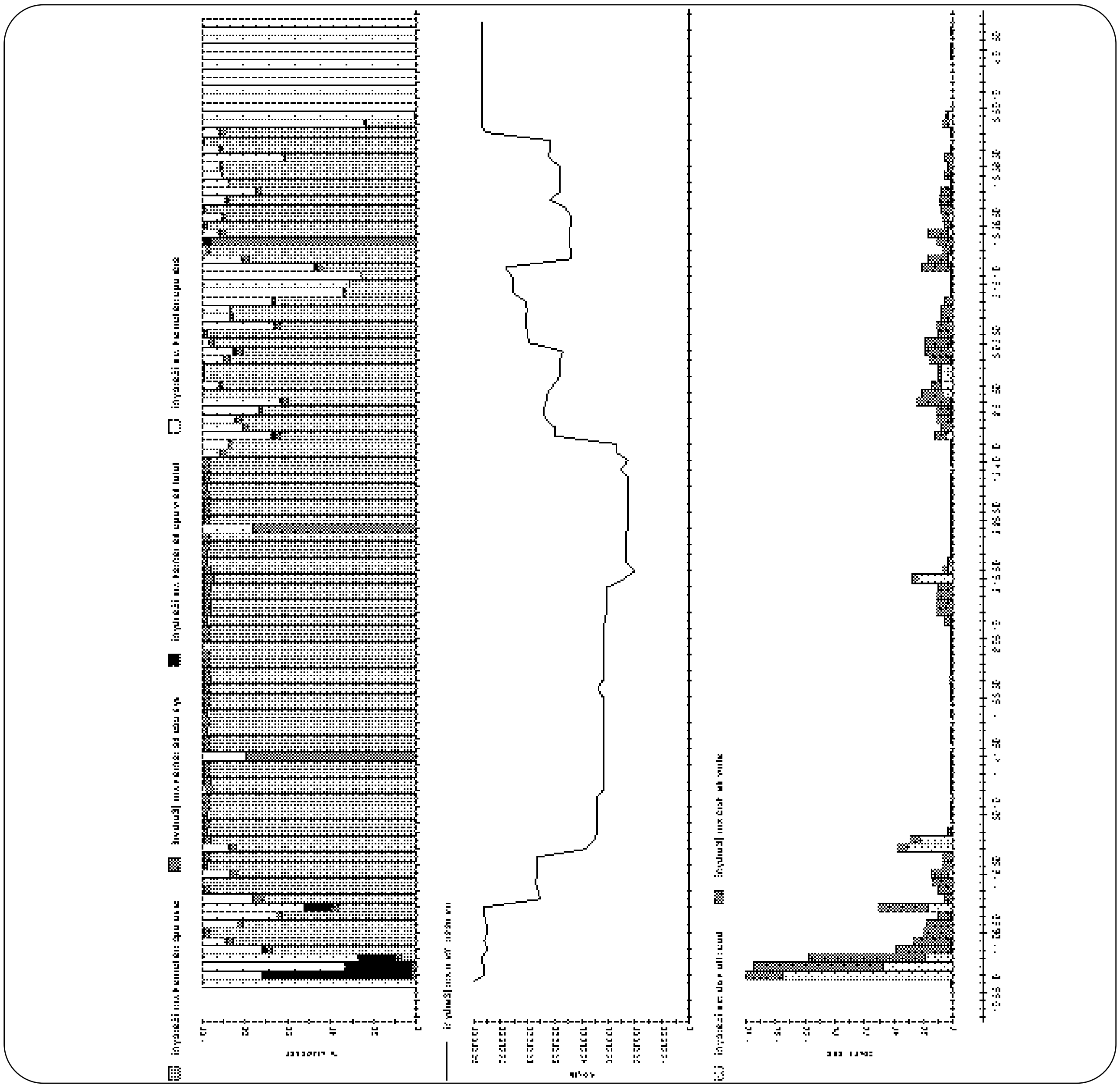


Figure 5-2 Loading System Resources: A Typical Run

Actual	Predicted	Efficiency	CPU Count	Memory (GB)
14014	14023	1.00	12	4
18429	18145	0.98	12	4
15893	16086	1.01	12	4
18243	18337	1.01	12	4
15633	15671	1.00	12	4
13915	13873	1.00	12	4
11325	11329	1.00	12	4
16309	16156	0.99	12	4
16855	16786	1.00	12	4
17979	17919	1.00	12	4
18558	18375	0.99	12	4
13674	13665	1.00	16	4
15083	15094	1.00	16	4
14299	13755	0.96	16	4
18824	18521	0.98	16	4
21831	21536	0.99	16	4
17496	17564	1.00	16	4
16920	16897	1.00	16	4
14485	14303	0.99	16	4
11258	10787	0.96	16	4
13079	12203	0.93	16	8
11104	10734	0.97	16	8
13777	12960	0.94	16	4
10783	10436	0.97	16	4

Table 5-3 Predicted and Measured Job-Mix Times

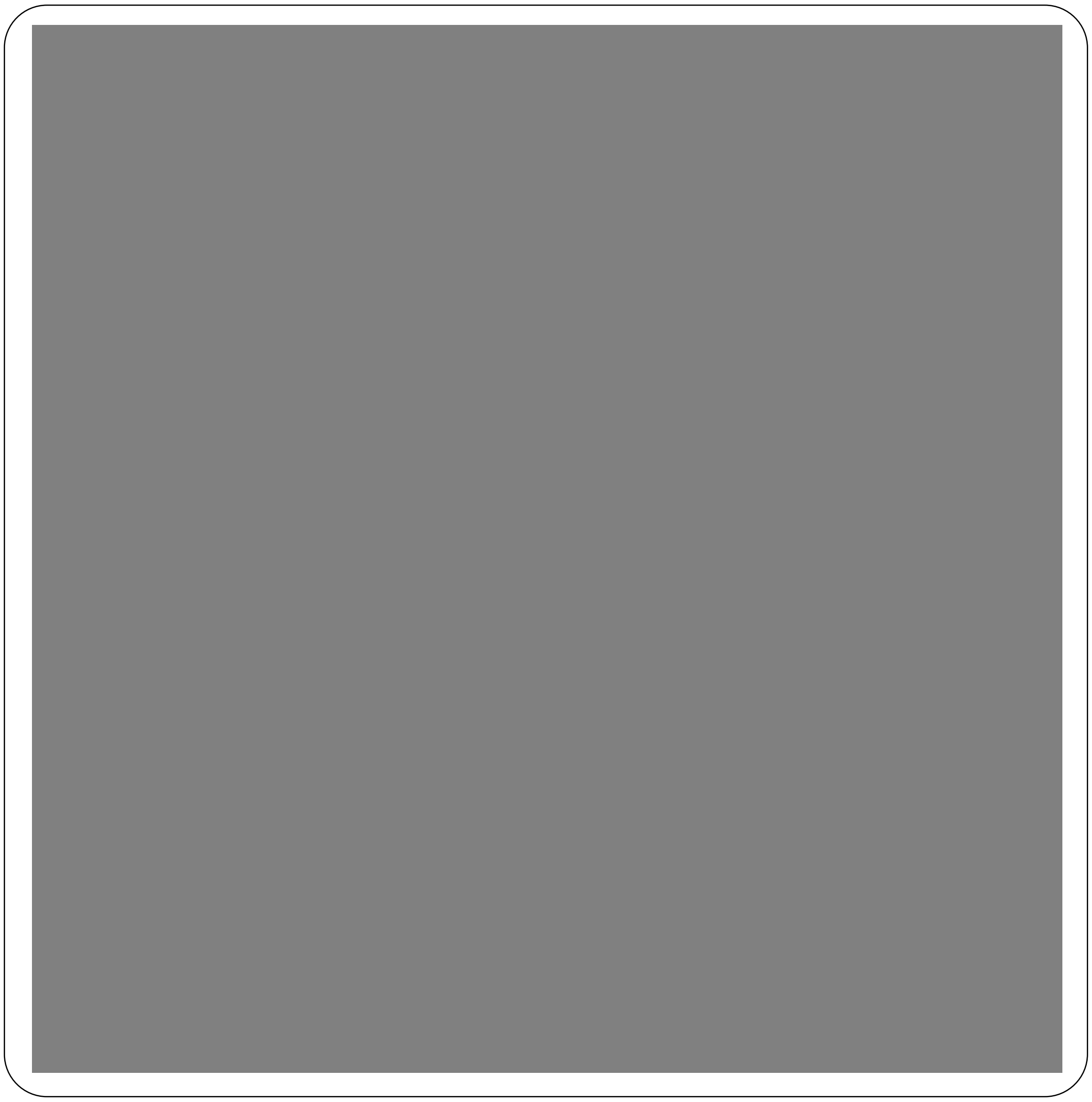


Figure 5-3 Histogram of Multi-Job Efficiency

Leveraging Resources: POWER CHALLENGEarray

For large sites, POWER CHALLENGEarray offers an attractive alternative to the established mainframe. With respect to the individual, the system offers the excellent single CPU floating-point performance of POWER CHALLENGE, only now the total number of CPUs can grow to 288 per system (8 POWERnodes with 36 R10000 CPUs per POWERnode).The system continues to support multiple CPU parallelism via automatic com-pilation and/or PVM/MPI.

Other supercomputer features are supported as well. Some very important features include:

- ❑ A flat 64-bit address space—allowing single jobs to directly access all 16GB of memory in a single node
- ❑ Transparent large-file support for single files larger than 2GB is standard

The Silicon Graphics high-performance product line also provides high-performance I/O features, including HIPPI (sustained point-to-point file transfers over TCP/IP in excess of 90MB/sec).

For system administrators needing to oversee the throughput needs of an entire department, POWER CHALLENGE and POWER CHALLENGEarray products are a perfect match. Large numbers of CPUs attached to large globally-addressable memories ensure the best possible use of each CPU in a heavy throughput environment. The single-system image with single points of control also dramatically reduces the burden to management. This means that the system administrator has a much better chance of using all of his memory and all of his CPUs for his workload, thus better serving the user community.

Simple-minded, distributed-memory, “shared nothing” traditional message-passing cluster systems insure that CPU and memory resources cannot possibly be used efficiently. Jobs cannot be migrated in such a system in any practical sense. The result is a severe under-use of expensive system resources for an average site.

Throughput Economics

We consider here the relative economic value of shared-memory versus dis-tributed-memory. Typically, the high-throughput environment will have a significant batch workload. It also has a wide variety of jobs which need to be run and which stress different aspects of system capabilities:

- Compute performance
- Memory capacity
- Disk performance
- Disk capacity
- Network performance.

We consider here the economics of throughput based on memory and processor configuration for the shared memory environment (SMP), the POWER CHALLENGEarray environment, and the usual distributed mem-ory environment (MPP). In the batch environment, jobs are not scheduled until resources are available: processors, memory, disk space, licenses, and more. In the distributed environment, there is an additional constraint: for MPP, memory for a job must be local to the processor used for that job thread. It is intuitively clear that this additional constraint will lead to less efficient use of memory than for SMP. It is also clear that the economics for POWER CHALLENGEarray will fall between the SMP and the MPP extremes. We explore these economics below.

Consider MPP first. Any computer site can characterize its workload accor-ding to a mix of jobs, each job*i* having requirements for a number of proc-essorsP_i and some amount of memorym_i. Each job also requires a certain amount of timet_i to run, and is run with a relative frequencyf_i to the other jobs that are in the mix. To be able to run any job in the job mix requires that the system have the minimum capacities for that job. In particular, the critical resource will often be memory per processor. Many jobs are not performance sensitive, but nevertheless require a large memory reserve to run. These jobs will run on very few processors (typically one), and a small number of processors required for the job will tend to create large memory per processor requirements. To run the job at all requires some number of processors to be configured with the largest memory per processor that any job in the job mix might require. It is desirable in an MPP machine to have balanced memory. This gives the possibility of running jobs across the whole machine which can use the whole memory. We have made this additional assumption in looking at throughput economics. While this assumption can be relaxed, it is not without penalty. Uneven amounts of memory per processor will lead to greater scheduling difficulties and will reduce the effective memory available for large multiprocessor runs. Simi-larly, the flexibility to grow workload to include jobs with large memory per processor requirements will be reduced. Assuming balanced memory, the memory required for the whole system is simply:

$$M = [m/p] \bullet P$$

where M is the memory of the computer system, P is the number of proc-essors in the system, and $[m/p]$ is the largest value of memory per processor required by any job.

We now consider the SMP case. Here, memory is a shared resource and assuming that the largest memory requirement is met, system memory requirements are something like:

$$M = \langle m/p \rangle \bullet P$$

where m/p is the expectation value of memory per processor required for any job. The expectation value for any variable x is:

$$M = \frac{\sum \chi_i f_i t_i}{\sum f_i t_i}$$

We also define σ according to the usual convention for future reference.

$$\sigma_x = \sqrt{(x - x)^2}$$

We examine this intuition in more detail. The problem of scheduling work in a real heavily-loaded batch environment is somewhat different than the previous method of throughput scheduling. In the typical environment, it is difficult to know *a priori* what the real values of memory, processors, and execution time that any one job will require. Also, scheduling is not gen-erally a function of optimizing the order of execution of jobs to maximize throughput. While there is rescheduling, it is generally due to site policies and not directly to throughput optimization. To give some idea of what the raw throughput capacity of a system will be, we propose a simple model of throughput which is close to a typical sites use of throughput. We assume for a given configuration of memory and processors, that the next job is only allowed to execute when enough processors and memory are available to run the job. We assume that there is always a job waiting to be executed, and that the jobs are randomly distributed according to the various frequenciesf_i.

There are a number of ways to measure throughput, and we choose processor-sec equivalents. In general, determining a closed form for throughput is difficult, so we turn to modelling the throughput via a Monte Carlo simulation over a long and fixed time interval. The results of such a run based on a job mix typical of the previous sessions and with varying processors and memory for an ideal SMP machine is shown in Figure 5-4 on page 97 as a greyscale plot. Lighter values of gray indicate larger throughputs. Hyperbolic contours represent lines of constant throughput. The solid sloped line is the line of constant system cost assuming a fixed ratio between the cost of processors and the cost of memory. In this case, we used current processor and memory costs for POWER CHALLENGE. To buy the most throughput for a certain budget, simply search along the line until the line is tangent to an isobar of throughput. The locus of such points for different budgets, assuming constant ratios of processor cost to memory cost, is shown as the upper dashed line. The lower dashed line shows the appropriate choice of proc-essors to memory based on the constraints of the job which requires the largest amount of memory per processor under the principle of balance.

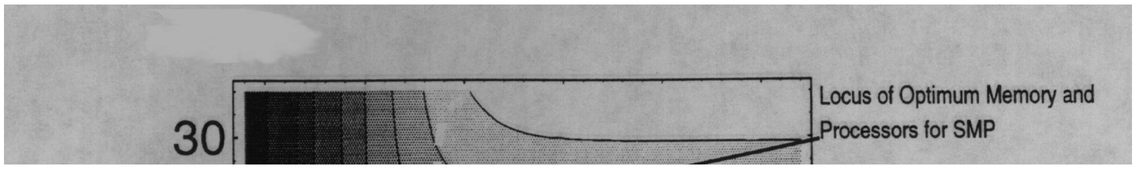


Figure 5-4 Plot of a Typical Job Mix for an Ideal SMP Machine

To understand throughput capacity for POWER CHALLENGEarray, we ex-tend the principle of balance that we have for MPP. The number of proc-essors per POWERnode are equal across the array and less than or equal to 18 processors per POWERnode. Memory per POWERnode is divided evenly. We measure performance with a similar Monte Carlo simulation. For job mixes similar to the mix used in Figure 5-4 and under the same costing assumptions, the locus of best processor-memory points is plotted in Figure 5-5 on page 98 for both POWER CHALLENGEarray and MPP.

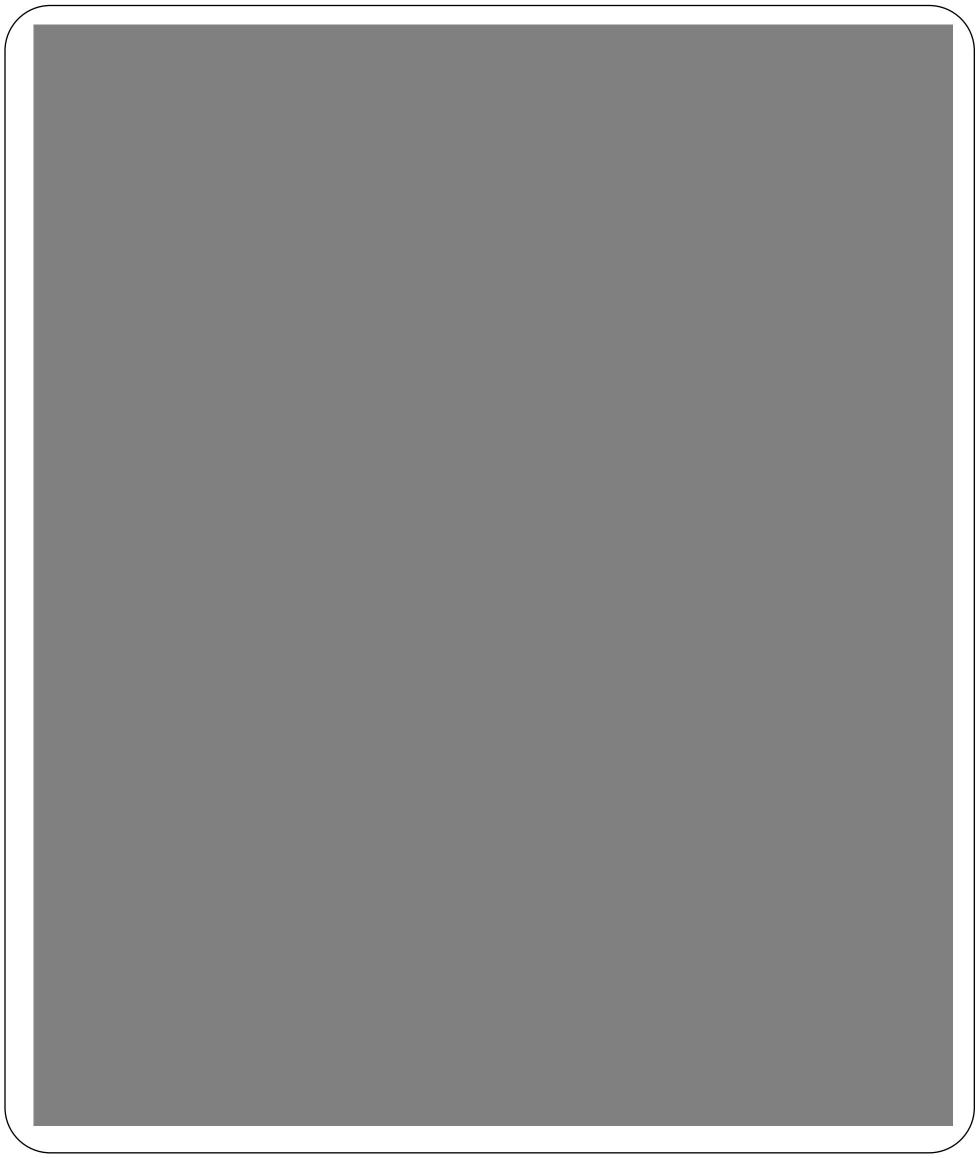


Figure 5-5 Locus of Optimal Processor-Memory Combinations

The team ran many experiments with varying job mixes and varying ratios of processor to memory cost. From these experiments, Silicon Graphics found an empirical approximation for finding the optimal ratio of memory to processors for a given budget *B($K)*, a given cost of memory C_m *($K/GB)*, and a given cost of processors C_p *($K/processor)*.

$$\frac{M}{P} = \frac{\eta \quad m \quad + \frac{1}{2}\sqrt{\frac{C_p}{C_m}\eta\sigma_m\sigma_p}}{\eta \quad p \quad + \frac{1}{2}\sqrt{\frac{C_m}{C_p}\eta\sigma_m\sigma_p}}$$

$$\eta = 0.0438 \quad \frac{B\sigma_{m/p}}{C_p \quad p \quad + C_m \quad m} \quad (1 - 0.9992^B)$$

The average error for the fit is about eight percent.

It is interesting to compare the relative throughput obtained for a partic-ular budget between MPP-style architecture and POWER CHALLENGEarray architecture. The results of such a comparison are shown in Figure 5-6 on page 100. In this figure, we have also looked at the case where the job re-quiring the largest memory/processor ratio was dropped from the mix. For constant budgets, shared memory provides factors of two or more in throughput capacity.

The performance of POWER CHALLENGEarray versus MPP is shown in Figure 5-7 on page 102 for the same conditions as Figure 5-6. You see that generally, coarse-grained distributed memory combined with SMP nodes leads to efficient use of memory, compared to the ideal case of pure SMP.

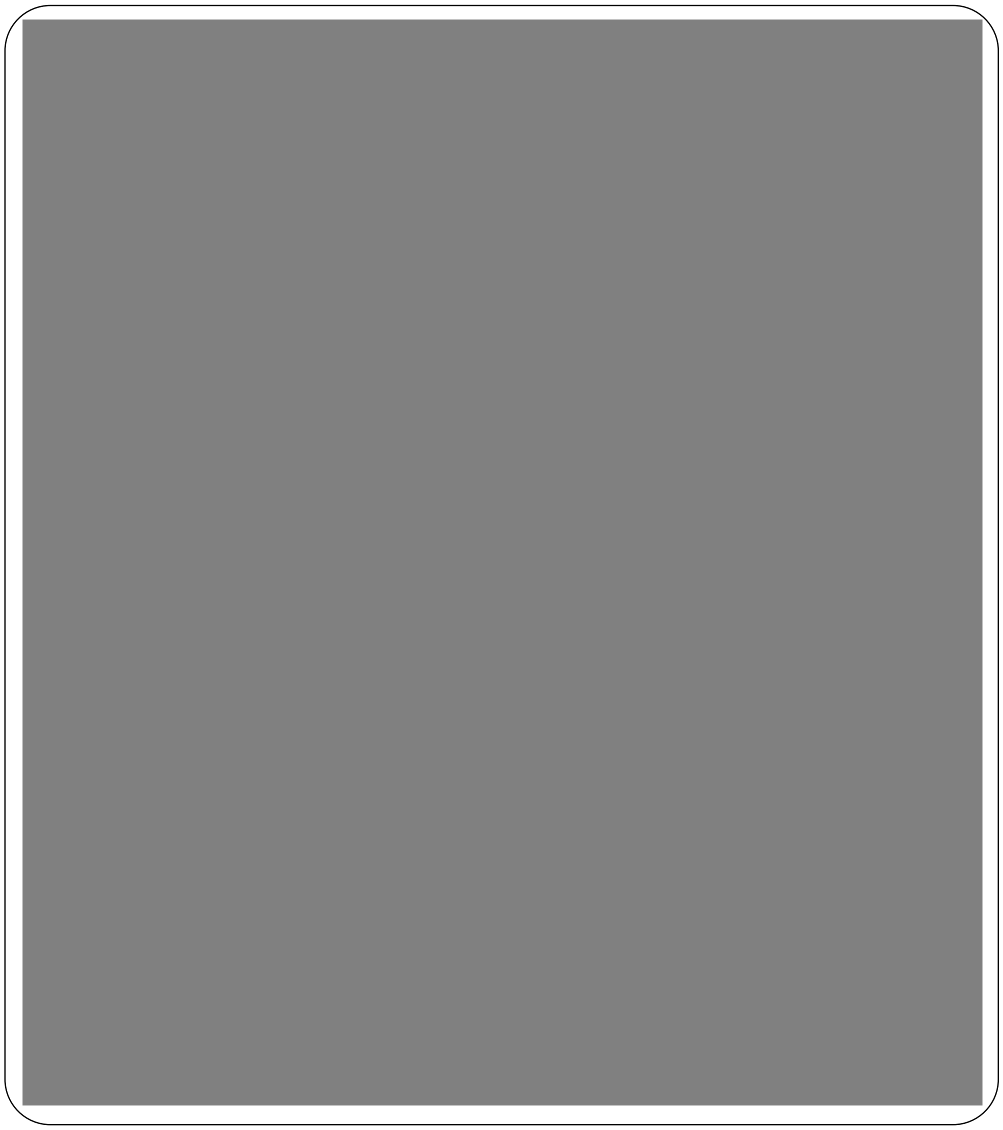


Figure 5-6 Throughput Comparison: MPP and POWER CHALLENGEarray

There are several intangible benefits of shared memory and the coarse-grained memory distribution of POWER CHALLENGEarray. It is important to not only have the minimum system capacities, but also the flexibility to handle unforeseen demands on memory capacity and memory per processor. The fundamental shared-resource approach underlying SMP and the semi-shared approach implemented in POWER CHALLENGEarray will both have excellent flexibility to handle larger memory demands up to the fundamental available memory. While it is possible to enhance through-put value for MPP with unbalanced memory mapping, total throughput value will not approach the SMP case, and, inevitably, compromises in flexibility will ensue. Net memory for jobs which run across the system will fall. The capacity to increase memory/processor will also be severely restricted. All follow from the fact that on an MPP machine, memory is additionally constrained to be local to the processor that is using that memory.

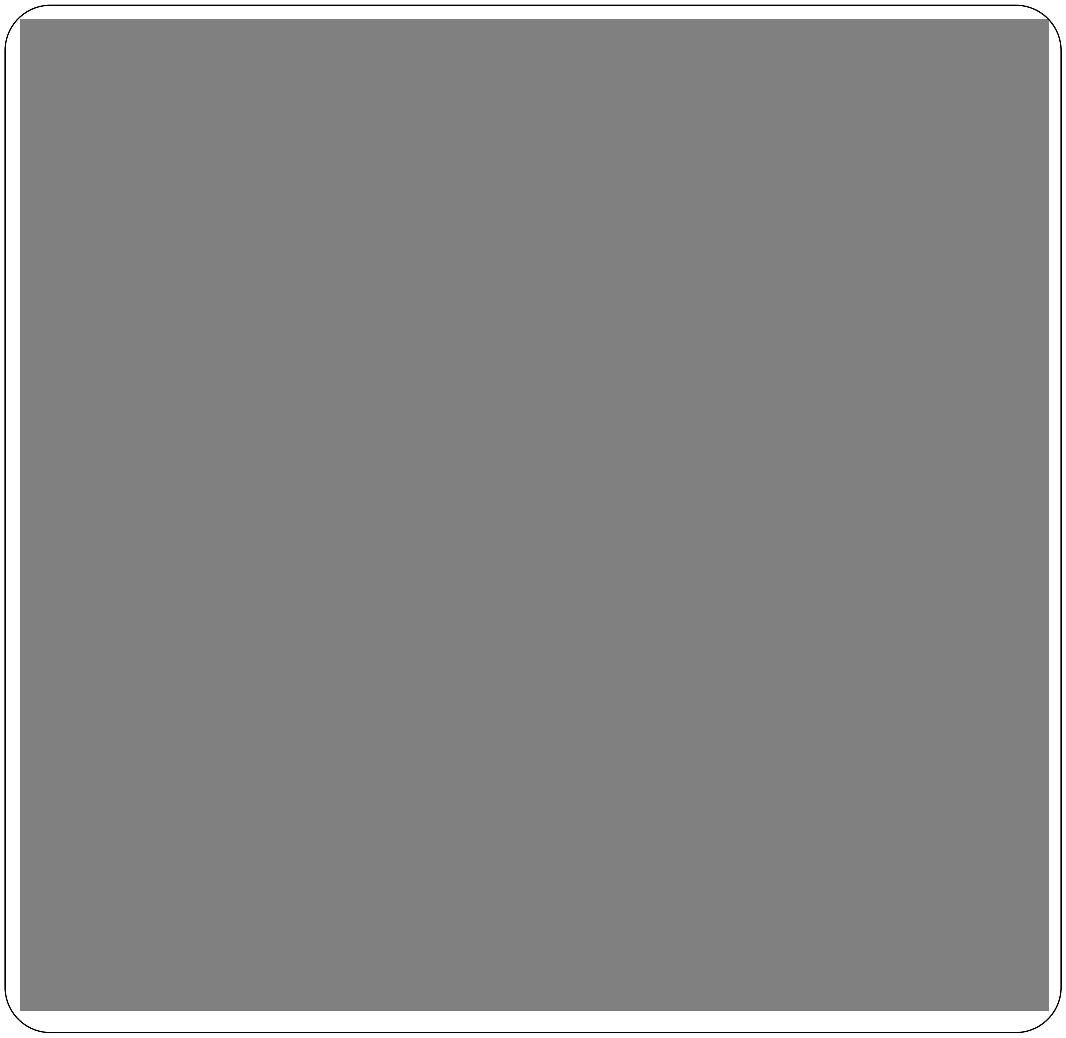


Figure 5-7 Throughput Comparison: SMP/POWER CHALLENGEarray

In the heavy throughput environment, POWER CHALLENGEarray offers powerful memory economies. The cost of memory for traditional high-end vector processors practically limits vector machines to small memory cap-acity per processor flop. Commodity pricing levels of POWER CHALLENGE memory allow large memory configurations for both raw memory capac-ity, and allow processors to run job mixes with much greater processor utilization. MPP machines, in turn, require far larger memory configura-tions to achieve similar throughput capabilities.

Summary

This study shows that POWER CHALLENGEarray is well-suited to provide high-performance compute capacity for your heavily-loaded computer center. These key features include:

- ❑ Excellent individual processor performance across a wide range of scientific applications
- ❑ Low programmer investment to port existing applications with excellent performance
- ❑ Balanced architecture to handle heavy concurrent job execution with excellent efficiency
- ❑ Large and inexpensive memory capacity
- ❑ Best-in-class throughput value

In the heavy throughput environment, memory is an often overlooked and underconfigured component of throughput capacity. With care, it is possible to maximize the total throughput capacity by matching memory and processors for the site’s real workload. Buying the most throughput capacity for available budget dollars will decrease average user turnaround time, an important business objective. It is also clear that the large and affordable memory capacity of POWER CHALLENGEarray will enable larger detailed scientific and engineering numerical models, which in turn will increase your overall enterprise competitiveness and productivity.

ThePOWERWALL Project

6

6.1 The POWERWALL Project

Background and Motivation

The original motivation for building a high-performance scientific visual-ization wall was to build a visualization and display system capable of coping with the high-resolution images and the high-bandwidth require-ments of supercomputing applications and to do this in a large format so that a group of researchers could display their data interactively and dis-cuss it together on a “digital” chalkboard. In this case, the chalkboard be-comes an ultra-high-resolution full-color window, also known as POWERWALL, on the virtual world of their supercomputing applications.

POWERWALL—Enabling the Power Workplace

The primary purpose of POWERWALL is to visualize and display very high-resolution data from large scientific simulations performed on super-computers or from high-resolution imaging applications. In addition to this high resolution, POWERWALL provides a large 6-foot-by-8-foot display area to facilitate collaborations of small researcher groups using the same data. All collaborators can see the display clearly and without obstruction, and the rear-projection technology makes it possible to walk up to the dis-play and point to features, just as one would while discussing work at a chalkboard. Thus POWERWALL could be a model for the digital movie theater of the future, since its display of 3200 X 2400 pixels has nearly the resolution of 35mm movie film. The POWERWALL has been used to anim-ate images drawn by computers. These images represent the results of supercomputer simulations of the behavior of gases under exotic con-ditions. Because these conditions prove difficult to achieve in the labor-atory, computers are used to simulate these environments instead.

An example of such an exotic fluid flow, simulated on a supercomputer and displayed on POWERWALL, is propagation of a gaseous jet at Mach 4 through an ambient gas 10 times denser. This simulation helps astron-omers to understand powerful jets that are observed shooting out of the nuclei of certain active galaxies. The same simulation is also helpful in understanding how jet aircraft engines generate noise at the airport. In either case, gas flow is subdivided into millions of tiny cells in which the behavior of the gas is treated in a simplified fashion. To see the results of such a calculation in their full complexity, a POWERWALL display is required, with its nearly 8 million pixel resolution.

POWERWALL can also be used as a virtual reality (VR) system by utilizing specialized software for navigating through data sets. These data sets could come from computer simulations or, for example, satellite observations of terrain and data archives, such as meteorological or geological archives. These data sets can be accessed by applications running on the Silicon Graphics POWER CHALLENGEarray system that drive POWERWALL. As the user explores the data sets, POWERWALL also becomes a window onto the virtual world of the simulation.

The University of Minnesota in collaboration with Silicon Graphics, Ciprico, Inc., and IBM Storage Products Division, successfully constructed and demonstrated the very first POWERWALL. On display in Silicon Graphics Booth #401 at Supercomputing '94, this system consisted of two POWER Onyx supercomputers, each equipped with eight MIPS R8000 processors, 2GB main memory, 15 Fast/Wide SCSI-2 I/O channels, two HiPPI channels, 12 Ciprico disk arrays (192GB total per system), 2 RealityEngine^2 graphics engines, and two Electrohome Marquee 8000 projection screen displays.

The Supercomputing ’94 POWERWALL was used to interactively explore a data set taken from the largest simulation to date of homogeneous, compressible turbulence, a simulation carried out a year ago by the University of Minnesota team using a Silicon Graphics 320 CPU CHALLENGEarray XL system.

Raw data representing the velocity field in the simulation was rendered into images with a POWER CHALLENGEarray system and displayed interactively on POWERWALL. This turbulence simulation produced a data set of half a terabyte. The POWERWALL enables scientists to put entire data sets of this size on line for fully interactive exploration. The ultimate intended result is scientific insight, which alone can be obtained by viewing all the data interactively from any angle using any desired method of visualization.

The Technology

Display Hardware

POWERWALL's display is a single 6-foot-by-8-foot screen illuminated from the rear by a 2-by-2 matrix of Electrohome video projectors. These projectors are driven by four RealityEngine^2 graphics engines. Each projector provides a resolution of 1600 x 1200 pixels (about 2 megapixels), making the entire POWERWALL resolution 3200 x 2400 pixels (~8 megapixels). Ciprico disk arrays supplied the RealityEngine^2 graphic engines with more than 300MB per second of data to display smooth-motion animation across the entire viewing area. POWERWALL does not consist solely of a high-resolution display system; it is in itself a super-computing system. In the configuration setup at Supercomputing '94 (see Figure 6-1 on page 108), POWERWALL was an integrated visualization system connected by a HiPPI network to a POWER CHALLENGEarray distributed parallel processing system, which included large and extremely fast disk storage systems for raw or image data and more than 100 MIPS R8000 processors.

POWERWALL Software

POWERWALL software was developed over a course of six years by a team of people at the University of Minnesota's Graphics and Visualization Laboratory (GVL) at the Army High-Performance Computing Research Center (AHPCRC) under the direction and supervision of Paul Woodward and Tom Ruwart. Based on graphics and visualization tools, the software is responsible for synchronization and control of processing, movement, and display of data on POWERWALL.

Figure 6-1 Supercomputing ‘94 POWERWALL Equipment Configuration

The POWERWALL Team

The University of Minnesota team is headed by Dr. Paul Woodward, a professor of astronomy at the University of Minnesota. Dr. Woodward is also a Fellow of the

ıtific visualization of

ı supercomputer

ormed on

Silicon Graphics

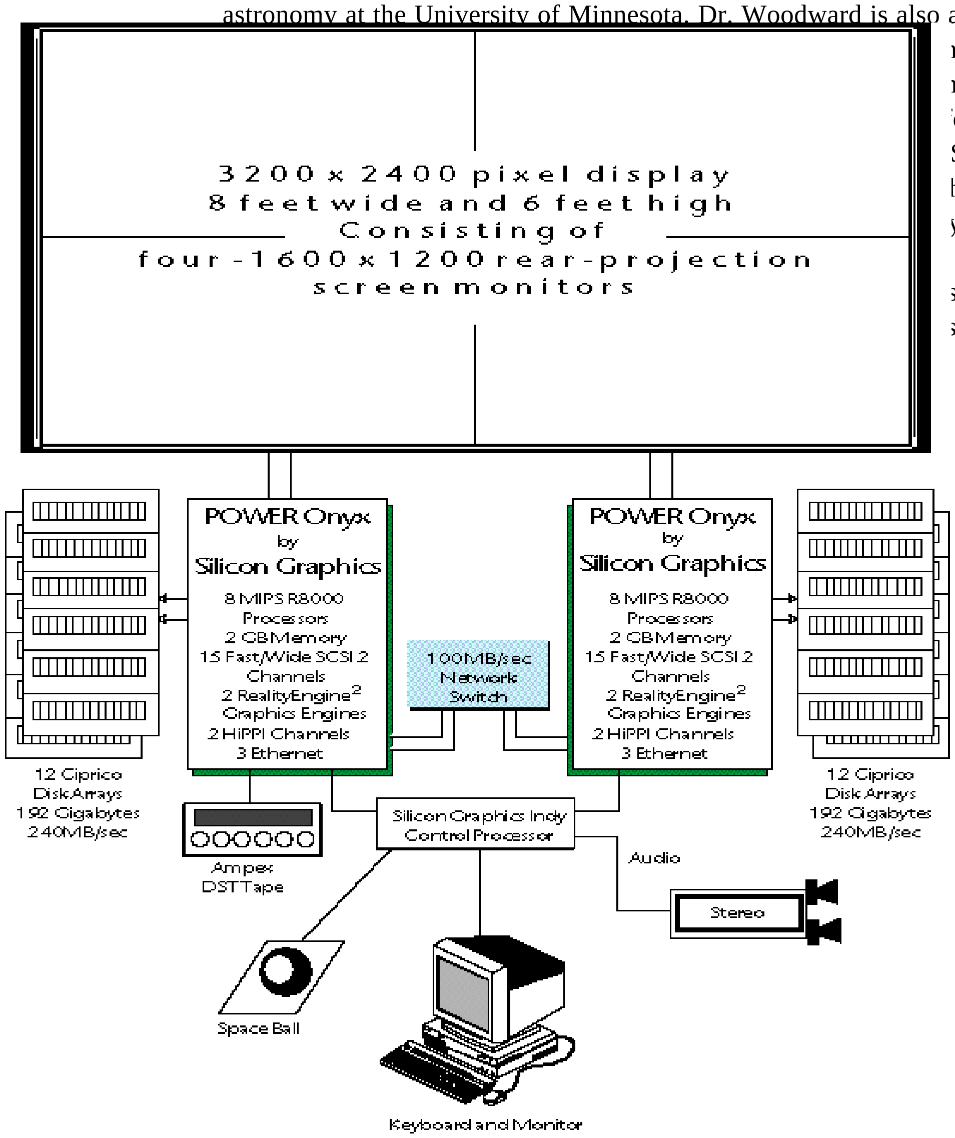
hich he developed

y and at the

by Thomas Ruwart.

son, Michael Palmer

Jacobson, and Jeff



References

A

POWER CHALLENGEarray References

Silicon Graphics publications:

POWER CHALLENGE™ Tech Report

POWER CHALLENGEarray™ Data Sheet

Onyx® and POWER Onyx™ Technical Report

Onyx Family Product Guide

Reality Station Data Sheet

IRISconsole™ Data Sheet

SHARE II™ Data Sheet

Other Publications:

Forest Baskett and John Hennessy, “Microprocessors: From Desktops to Supercomputers”, Science, August 1993, pp 864-871.

Don Tolmie and John Renwick, “HiPPI: Simplicity Yields Success”, IEEE Network, January 1993, pp 28-32.

EISA HiPPI Network Interface Card Data Sheet - Essential Communications

Serial HiPPI Media Interconnect Card Data Sheet - Essential Communications

“Using MPI”

“PVM: Parallel Virtual Machine: A User’s Guide and Tutorial for Networked Parallel Computing”

xHPF Parallelizer - Applied Parallel Research

PGHPF Writeup from Portland Group

TotalView Release Notes - BBN System and Technologies

PerfAcct Version 1.0 User’s Guide - Instrumental

David Porter et al, “Attacking a Grand Challenge In Computational Fluid Dynamics on a Cluster of Silicon Graphics Challenge Machines,”

Michael Berry et al, “Algorithmic Design on the Cedar Multiprocessor,”

Message-Passing Interface (MPI) References

Silicon Graphics man page:

man mpi

WWW home page for POWER CHALLENGEarray:

http:/www.sgi.com/Products/PowerChallengeArray

Book on MPIs:

Using MPI

WWW home page for public domain MPI:

http://www.mcs.anl.gov/mpi/index.html

Parallel Virtual Machine (PVM) References

WWW home page for POWER CHALLENGEarray:

http://www.sgi.com/Products/PowerChallengeArray

Getting Started with the POWER CHALLENGEarray

Book on PVM*:*

PVM

WWW home page for public domain PVM

http://www.epm.ornl.gov/pvm/pvm_home.html

High Performance FORTRAN (HPF) References

WWW home page for PGI*:*

http://www.pgroup.com

WWW home page for APR

http://www.infomall.org/apri

Distributed Batch Processing and Load Balancing Tools References

WWW home page for Platform Computing Corporation*:*

http://www.platform.com

WWW home page for Instrumental Inc.:

http://www.instrumental.comi

Reader Comment Sheet

Dear Customer,

At Silicon Graphics we want to provide you with the best possible documentation for our products. To this end, we solicit your comments on this report. We would appreciate your telling us about any errors in the content. Also, please tell us of any material that you feel should be there but isn't. Comments on *POWER CHALLENGEarray Tech Report can be E-mailed, faxed, or sent to:*

POWER CHALLENGEarray Marketing

Silicon Graphics
MS 8L-580
2011 N. Shoreline Blvd.
Mountain View, CA 94043
FAX: (415) 390-3562
E-Mail: krsik@asd.sgi.com