REPRINTED FROM IBM SYSTEMS JOURNAL, VOL34, NO 2, 1995; © 1995, 1999

SP2 system architecture

by T. Agerwala
J. L. Martin
J. H. Mirza
D. C. Sadler
D. M. Dias
M. Snir

Scalable parallel systems are increasingly being used today to address existing and emerging application areas that require performance levels significantly beyond what symmetric multiprocessors are capable of providing. These areas include traditional technical computing applications, commercial computing applications such as decision support and transaction processing, and emerging areas such as "grand challenge" applications, digital libraries, and video production and distribution. The IBM SP2™ is a general-purpose scalable parallel system designed to address a wide range of these applications. This paper gives an overview of the architecture and structure of SP2, discusses the rationale for the significant system design decisions that were made, indicates the extent to which key objectives were met, and identifies future system challenges and advanced technology development areas.

The IBM SP2* is a general-purpose scalable parallel system based on a distributed memory message-passing architecture. Generally available SP2 systems range from 2 to 128 nodes (or processing elements), although much larger systems of up to 512 nodes have been delivered and are successfully being used today. The latest POWER2* technology RISC System/6000* processors are used for SP2 nodes, interconnected by a high-performance, multistage, packet-switched network for interprocessor communication. Each node contains its own copy of the standard AIX* operating system and other standard RISC System/6000 system software. A set of new software products designed specifically for the SP2 allows the parallel capabilities of the SP2 to be effectively exploited.

Today, SP2 systems are used productively in a wide range of application areas and environments in the high-end UNIX** technical and commercial computing market. This broad-based success is attributable to the highly flexible and general-purpose nature of the system. This paper gives an overview of the architecture and structure of SP2, discusses the rationale for the significant system design decisions that were made, indicates the extent to which key objectives were met, and identifies system challenges and advanced technology development areas for the future.

We first discuss the overall goal of the SP2 system and the key focus areas. Next we discuss our rationale for the systems approach we have selected and which we will refine over time to meet these requirements. This is followed by a discussion of the overall SP2 system architecture, some of the major system components, and the SP2 performance. We conclude with our views on the key challenges facing system architects of scalable parallel systems and areas in which we need to focus in the future, and a summary of how the SP2 systems are being used today.

 0018-8670/99/$5.00 © 1999 IBM

Design goals

Massively parallel processors (MPPs) have been around for a number of years. These systems have typically been designed to apply the combined capacity of hundreds and even thousands of low-cost, low-performance processing elements for solving single large problems. However, until recently these systems were not adopted for mainstream supercomputing applications. Since the individual processors gave very low performance, considerable effort was required up front to *parallelize* an application code sufficiently (that is, divide the code into multiple parts that can execute in parallel) even to get performance equivalent to the mainstream uniprocessors. That was a major inhibitor. In addition, limited processor memory, limited input/output, poor reliability, primitive nonstandard software development environments and tools, and programming models that were closely tied to the underlying hardware (such as the interconnection structure) all contributed to their failure to be generally accepted. MPPs remained, at best, special-purpose machines for very narrow niche applications.

From the inception of the SP2 project, our goal was to design general-purpose scalable parallel systems. We realized (as did others[1]) that for massively parallel systems to succeed they must be more general-purpose and less intimidating to use than they have been in the past. They must also provide all the capabilities available on a traditional system, at similar or lower price/performance. The basic nodes must be powerful enough and the underlying operating system must have full function so that users can move their current work over to the system with little effort and run their current applications in serial mode with acceptable performance. The systems must support familiar interfaces, tools, and environments, support existing standards and languages, and have common applications available. In this way, users can begin productive use of the system with little upfront effort and gradually parallelize and optimize the code over time. In addition, the system must also provide support in key areas to enable customers to grow (or *scale*) their applications in size and performance beyond what can be achieved on conventional systems.

Consequently, we have designed our systems to be used in a variety of environments. These include very large, stand-alone configurations dedicated to solving extremely complex and large single applications, smaller systems that coexist with mainframes and traditional supercomputers and that are used to offload some of the work for price/performance reasons, and consolidated servers for midrange local area network server environments.

The scalable parallel capabilities of the SP2 system allow customers to scale their applications, both in computation and data, much beyond what is possible with conventional systems. Our initial focus with the earlier SP1* was on high-performance scientific and technical computing in areas such as computational chemistry, petroleum exploration and production, engineering analysis, research, and "grand challenge" problems (those important for national interest). Today, SP2 systems address those areas and are also being used increasingly for commercial computing—primarily for complex query, decision support, business management applications, and on-line transaction processing. Over time we expect SP2 systems to be used for emerging applications such as large information servers, digital libraries, personal communications, video-on-demand, and interactive television, as well as for mission-critical applications for business operations such as airline reservations and point-of-sales.

In order to properly address these diverse applications and environments, we realized that we needed to focus the design on three key areas: programming models, flexible architecture, and system availability.

- Programming models—The SP2 must support key programming models prevalent in the technical and commercial computing environment so that existing applications can be readily ported (or migrated from another processor) to the SP2. These models are discussed in more detail later in this section.
- Flexible architecture—The SP2 must be flexible in how it can be configured and how it can be used. The system must be scalable from a very low entry point to a very large system, and be able to do this in small increments. The nodes must be individually configurable for hardware and software to meet the specific requirements of the customer's application and environment. The system must support a multiuser environment with a mix of serial and parallel, and batch and interactive jobs, and must accommodate a

mix of throughput and job turnaround time requirements.

- System availability—In order to succeed commercially, the SP2 cannot merely be a research machine; it must exhibit good reliability and availability characteristics so that customers can run their production codes on it. Points of catastrophic failures must be removed and failures must be isolated to the failing component and not be allowed to propagate. The system must support concurrent and deferred maintenance for the most common service situations and must support concurrent upgrade for the most common system upgrade situations. Finally, critical hardware and software resources must be designed for transparent recovery from failures.

Programming models in technical computing. The availability of software applications from vendors is critical to the success of any technical computing system. There is a large number and variety of such applications, and it is important to make it as easy as possible for software vendors and customers to port their applications to the SP2.

There is a significant number of applications available today for the RISC System/6000, and we must preserve the execution environment for these applications so that they can continue to run serially on an SP2 node without requiring any modifications.

In addition, key technical applications must be able to execute in parallel. To facilitate this, the system must provide support for prevalent parallel programming models and styles, and provide a comprehensive set of tools and environments (for both FORTRAN and C) for the development of new parallel applications, the porting of existing parallel applications, and the conversion of existing serial applications.

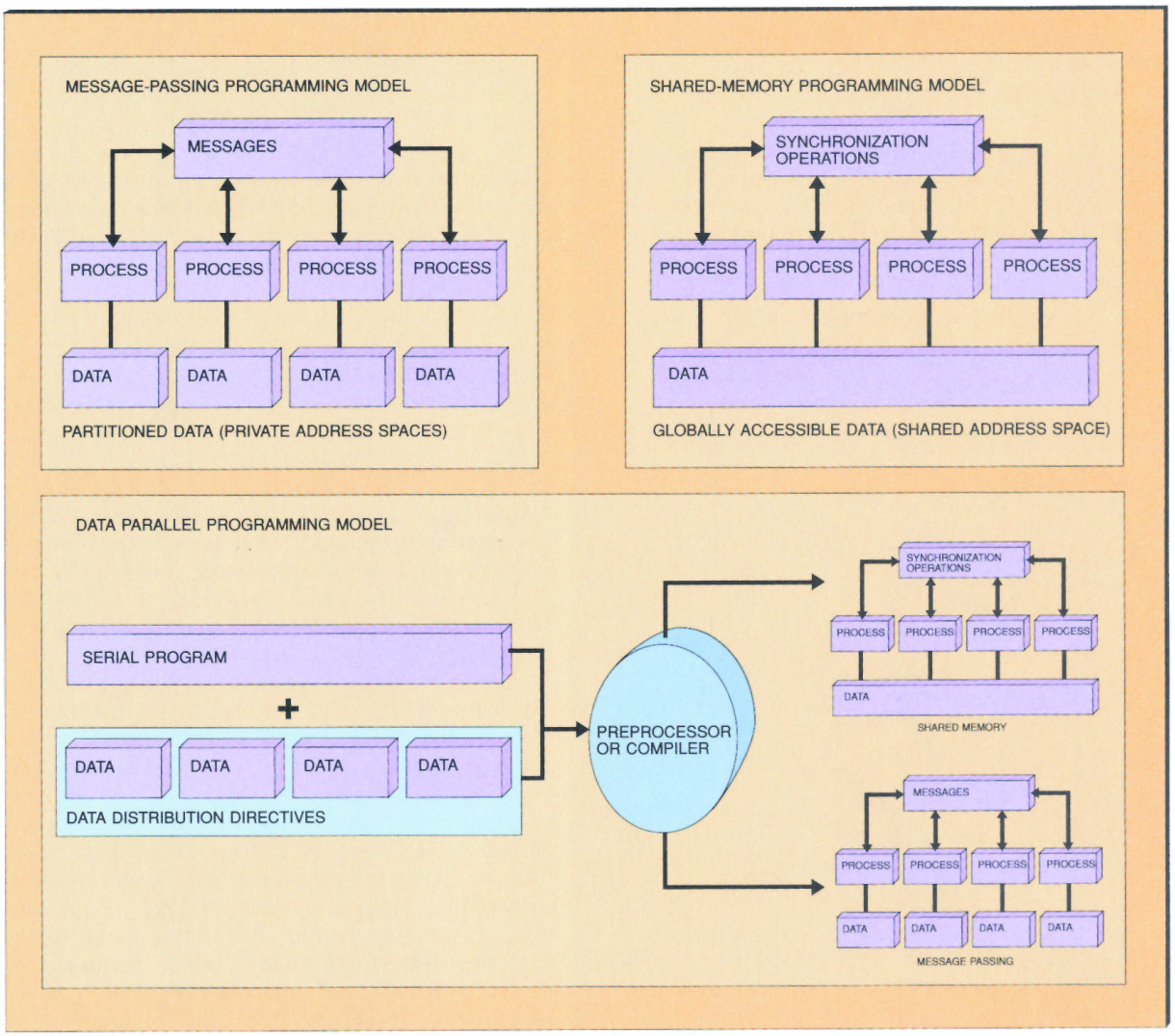
There are essentially three parallel programming models that are being used in large scalable systems (see Figure 1), the message-passing programming model, the shared-memory programming model, and the data parallel programming model.

Message-passing programming model. With the explicit message-passing model, processes in a parallel application have their own private address spaces and share data via explicit messages; the source process explicitly sends a message and the target process explicitly receives a message. However, since data are shared by explicit action on the part of the processes involved, synchronization is implicit in the act of sending and receiving of messages. The programs are generally written with a single-program, multiple-data (SPMD) stream structure, where the same basic code executes against partitioned data. Such programs execute in a loosely synchronous style with computation phases alternating with communication phases. During the computation phase, each process computes on its own portion of the data; during the communication phase, the processes exchange data using a message-passing library.

Shared-memory programming model. With the shared-memory model, processes in a parallel application share a common address space, and data are shared by a process directly referencing that address space. No explicit action is required for data to be shared. However, process synchronization is explicit; since there are no restrictions on referencing shared data, a programmer must identify when and what data are being shared and must properly synchronize the processes using special synchronization constructs. This ensures the proper ordering of accesses to shared variables by the different processes. The shared-memory programming model is often associated with dynamic control parallelism, where logically independent threads of execution are spawned at the level of functional tasks or at the level of loop iterations.

Data parallel programming model. The data parallel model is supported by a data parallel language such as High Performance FORTRAN.[2] Programs are written using sequential FORTRAN to specify the computations on the data (using either iterative constructs or the vector operations provided by FORTRAN 90), and data mapping directives to specify how large arrays should be distributed across processes. A High Performance FORTRAN preprocessor or compiler then translates the High Performance FORTRAN source code into an equivalent SPMD program with message-passing calls (if the target is a system with an underlying message-passing architecture), or with proper synchronizations (when the target system has a shared-memory architecture). The computation is distributed to the parallel processes to match the specified data distributions. This approach has the advantage of freeing the user from the need for explicitly distributing global arrays onto local arrays and changing names and indices accordingly, allocating buffers for data that must be communicated from one node to another, and inserting the required com-

Figure 1 Dominant programming models in parallel technical computing



munication calls or the required synchronizations. Another advantage is that High Performance FORTRAN source code is compatible with regular FORTRAN (since syntactically, directives are comments), so that code development can occur on ordinary workstations and porting code from one processor to another is easier.

To date, the primary focus for programming large scalable parallel systems has been on the explicit message-passing and data parallel models, and our current emphasis is on the efficient support of these models. For the explicit message-passing model, we must support the prevalent and emerging message-passing libraries efficiently. For the data parallel model, we must provide High Performance FORTRAN language support. Support of these programming models and an easy-to-use program development and execution environment are critical to encourage software vendors and users to invest the effort necessary to exploit the parallel capabilities of scalable parallel systems such as SP2.

Both explicit message-passing and data parallel models encourage a fairly static (declarative) distribution of data. As programmers become more sophisticated in the use of large scalable systems, we expect that parallel numerical algorithms in many disciplines will increasingly focus on sparse, irregular data structures, and dynamic distribution of data and computation to nodes. In the future, SP2 must also support a shared-memory programming model to enable this evolution. Improvements in compiler technology and in communications hardware and software will be necessary to enable support of this model on a system with an underlying distributed memory message-passing architecture. (Note that we are making a distinction here between the underlying system architecture and the supported programming style or programming model. It should be fairly evident that with the correct software and hardware support, any of the programming models can be supported on a system with either of the underlying architectures.)

Support for function shipping and data sharing models enable commercial applications.

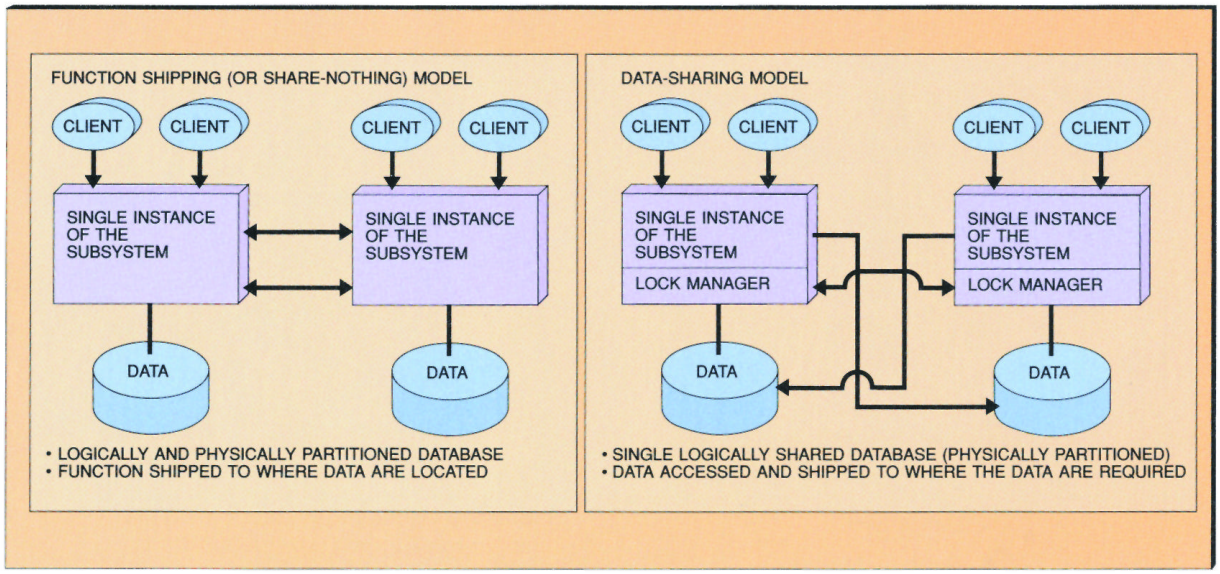
Programming models in commercial computing. "Commercial computing" is a broad term that has many different connotations. For our purposes, by commercial computing we will refer largely to online transaction processing (OLTP), database query processing, and related emerging applications such as data mining and very large information servers. Such commercial applications in the UNIX environment are largely based on a few key subsystems. Database subsystems include DB2/6000*,[3,4] Oracle,[5] Sybase,[6] Ingres, and Informix.[7] Transaction monitors include CICS/6000*,[8] Encina,[9] and Tuxedo.[10]

Porting these few primary subsystems to run in parallel on a scalable parallel system provides the basis for enabling a host of applications that utilize these subsystems. Many commercial applications do not need to be modified to run in a parallel environment since they utilize and request services from a few key subsystems. Instead it is these subsystems that need to be enabled and optimized for parallel execution or for throughput. The reason is that for many applications, the bulk of the processor time is spent executing function in the subsystems; thus optimizing the subsystem performance is the key aspect. Only sophisticated applications that provide considerable functionality over and beyond the underlying subsystems need to be specifically modified and tuned for parallel execution on scalable parallel systems.

These key subsystems mentioned above have all been enabled to run under the UNIX operating system. They were initially developed for high-volume single-processor systems, but most have been modified to run in a multiprocessor environment. To provide performance, capacity, and availability beyond symmetric multiprocessors, these subsystems are also being enabled to run in a clustered systems environment; in this environment a separate instance of the subsystem runs on each of the systems in the cluster, and a layer of software ties these instances together to provide a single system image to higher level application software.

There are two principal clustered system programming models for parallel transaction and query processing, as illustrated in Figure 2. In the function shipping model[11] (also referred to as the shared-nothing model[12]) the data are physically partitioned among the nodes in the cluster, and remote function calls are made to access remote data. In the data-sharing model,[13–15] the data are shared among the nodes of the cluster. One option is to provide a direct physical connection from all nodes to all devices storing the database (for example via multi-tailed devices). Alternatively, the data may be logically shared among the nodes, but are physically partitioned; in this case the remote data are shipped to a requesting node either at the database level, referred to as data shipping,[16] or at the input/output device driver level, which we refer to as virtual shared disk (VSD)[17] and further describe in a later section. The software that ties the instances together routes transactions to provide load balancing and affinity routing in the data-sharing case, or routes them based on the locality of data in the function shipping case. Complex queries are divided into individual steps that can be executed in parallel to reduce the turn-around time of a query. For the data-sharing model, a fully distributed "global lock" manager must also be provided.

Figure 2 Dominant programming models in parallel commercial computing



The critical step to enable commercial applications to run on a scalable parallel system is to support both of these fundamental programming models efficiently and to optimize the execution of the various parallel subsystems. In this sense, the solution is actually less complex than the technical computing area in which most of the individual applications have to be individually enabled for the scalable parallel environment.

System strategy

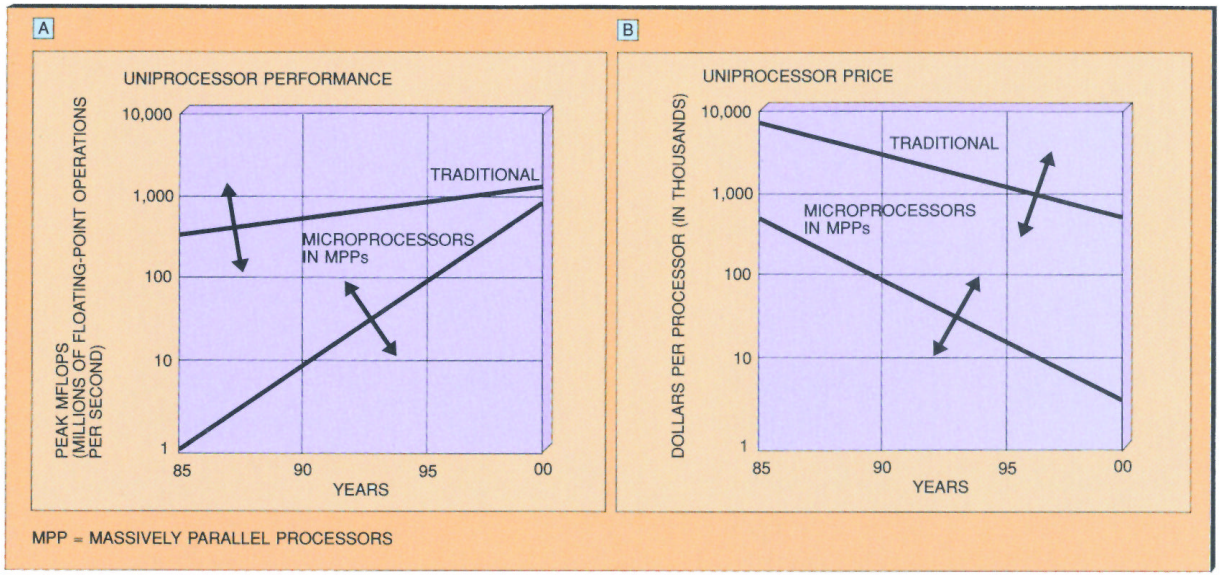
Various system approaches to scalable computing are being pursued commercially and are being investigated in academic environments. Scalable systems available today include the AT&T 3600[18] (formerly Teradata Corporation) and the Tandem Computers Inc. Himalaya**[19] in the commercial computing arena, and the Cray Research, Inc. T3D,[20] the Convex Computer Corporation SPP1000,[21] and the Intel Corporation Paragon[22] for scientific and technical computing. Academic research covers a broad range of different areas of investigation; these include how to improve the scalability of shared-memory multiprocessors,[23] what architecture support is required for low-latency communication and fine-grain computing,[24–27] and how to support efficient parallel programming over networked workstations.[28,29]

In designing the SP2 as a flexible, general-purpose scalable parallel system, we followed a set of guiding principles that are discussed below. We arrived at these principles after analyzing the current technology trends in both hardware and software, and understanding the requirements in the different application areas and customer environments we expected to address.

Principle 1. A high-performance scalable parallel system must utilize standard microprocessors, packaging, and operating systems.

Major technology advances in recent years have primarily come from the workstation and distributed systems marketplace. High volumes and competitive pressures in that marketplace have prompted significant investments, resulting in significant advances being made in all aspects of the technology—processors, input/output technology, communications technology, compilers, system software, tools, and applications. It is generally accepted that microprocessor performance is doubling roughly every 18 to 24 months. This is being accomplished through a combination of superscalar designs, faster and more dense CMOS (complementary metal oxide semiconductor) technologies, architecture improvements that take advantage of the increased gate counts, and improved compiler

Figure 3 Processor technology trends



optimizations that use these improvements. No fundamental limitations are expected over the immediate future, and processors with speeds in the hundreds of megahertz are being designed in the community. Furthermore, tightly packaged symmetric multiprocessors offer the opportunity for even greater improvements in node price/performance.

Figure 3 shows a least squares fit through the performance and price (over the past 10 years and extrapolated over the next few years) of microprocessor-technology-based processors used in MPP systems and custom-designed processors used in traditional vector supercomputers and mainframes. As the figure shows, the performance of the two is rapidly converging, while the price is diverging. It is our contention that specialized microprocessors for scalable, high-performance computing will be unable to keep up with the rate and pace at which the "commodity" microprocessors will evolve and improve. Therefore, our design approach is to "ride" the microprocessor technology curve; we will use standard components (both hardware and software) from the workstation environment as much as possible, and develop custom hardware and software only where standard technology cannot meet some unique requirements of a scalable parallel system at the desired performance levels.

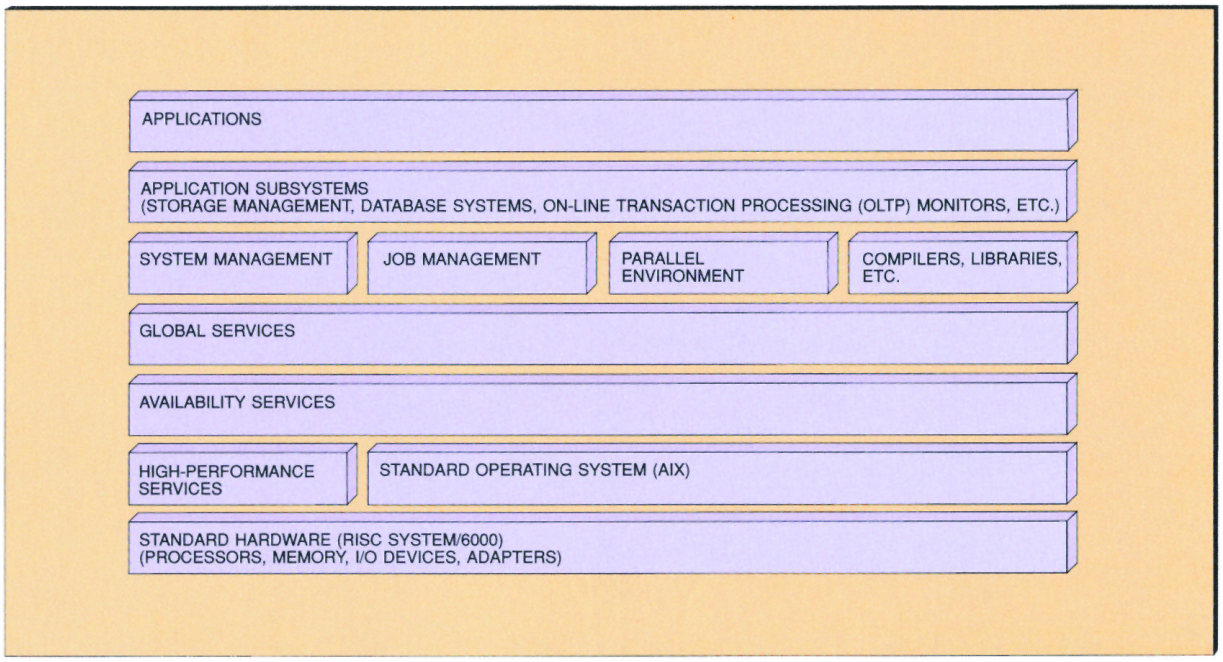
Principle 2. Time-to-market with the latest technology is critical to achieving leadership performance and price performance.

The rate of technology improvements mentioned above creates both an opportunity and a challenge. Since performance and time can essentially be traded, it is imperative that the SP2 systems be able to incorporate the latest microprocessor technology very rapidly. This emphasizes the need to exploit this technology essentially as is, and place as few dependencies as possible on our technology suppliers for special features to support parallel processing. It also has an implication for the underlying system architecture; it must be flexible enough to allow rapid exploitation of the latest hardware and software technologies without requiring time-consuming enhancements or modifications. This implies a relatively loose coupling between the nodes at the operating system level.

Principle 3. Required levels of latency (small multiples of memory access time) and bandwidth (small submultiples of memory bandwidth) will require custom interconnected networks and communication subsystems over the next few years.

For parallel applications, a key determinant of performance is the process-to-process communication

Figure 4 Software structure required for a scalable parallel system



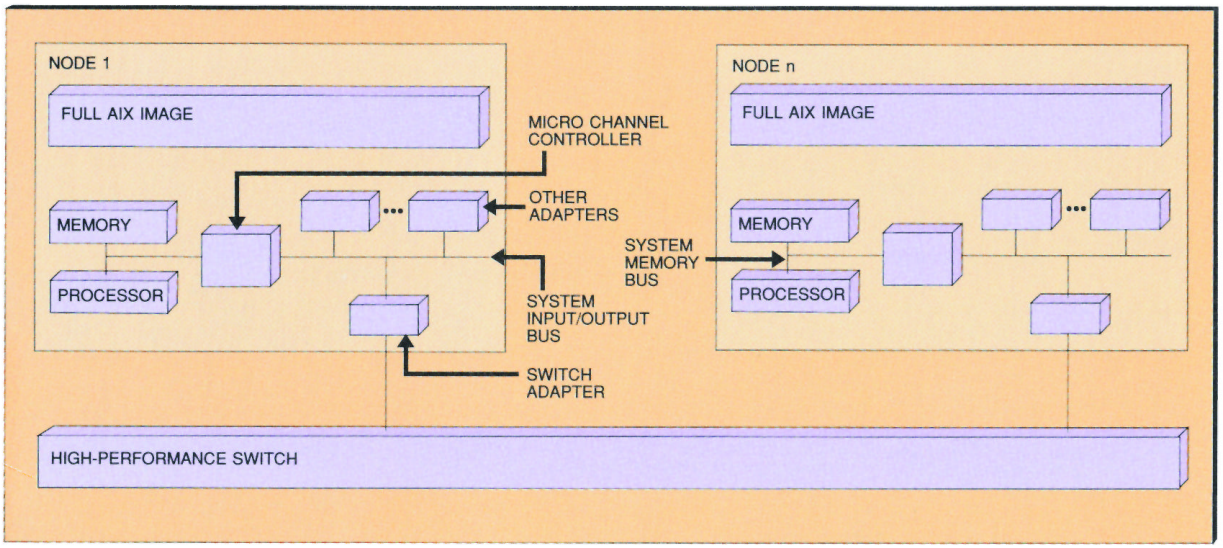
latency and bandwidth and the corresponding overhead on the processor for executing the communications protocol. In typical scalable parallel systems today, the sustainable pair-wise interprocessor communication bandwidth for large messages is typically several tens of megabytes per second, and the latency for short messages is of the order of a few tens of microseconds. Systems with global real shared-memory architecture can typically transfer a cache line amount of data from remote memory at even lower latencies. Significant improvements of up to an order of magnitude will be required in the future as the individual nodes improve in performance.

While several interesting "commodity" network technologies (such as Fiber Channel Standard [FCS] and Asynchronous Transfer Mode [ATM]) have recently emerged, these alternatives are optimized for a very different environment and do not provide the correct levels of latency, bandwidth, and processor overhead to meet the stringent performance requirements of parallel systems. For example, ATM networks are different from networks for scalable parallel systems in that the technology is optimized for communication between heterogeneous systems across great geographic distances, there is no guaranteed delivery or flow control in the low-level protocols, and there is no protection implemented at low levels. High-level protocols provide these functions and imply higher latencies. Interconnection networks in scalable parallel systems optimize for these functions at the lowest levels, and we believe that Principle 3 will therefore continue to be valid for low-latency as well as for high-bandwidth environments. In either case, standard network technologies with special software will support high-bandwidth communications to external devices at tens-of-microsecond latencies. This will allow scalable parallel systems to more effectively utilize network resources for a variety of tasks such as input/output, storage management, and some forms of computation.

Principle 4. The system must support a programming and execution environment identical to a standard open, distributed UNIX environment.

Figure 4 shows the full stack of software (explained in the rest of this section) that is required for enabling various technical and commercial applications to run. It is not feasible to develop unique

Figure 5 High-level system structure for the IBM SP2



new software for all or even the bulk of the components in the stack specifically for a scalable parallel system. Much of the software for systems management, job management, storage management, databases, and message-passing libraries exists for distributed UNIX environments. Our goal is to accommodate and depend on this software. This support provides one of the dominant "personalities" of the system and allows software written for a distributed UNIX environment and available for the underlying base node to run on the SP2 machine.

Principle 5. The system should provide a judiciously chosen set of high-performance services in areas such as the communications system, high-performance file systems, parallel libraries, parallel databases, and high-performance input/output to provide state-of-the-art execution support for supercomputing, parallel query, and high-performance transaction solutions.

Scalable parallel systems must provide a second dominant personality for the high-performance supercomputing environment. This consists of a set of high-performance services, and a development environment with tools to enable, develop, and execute new parallel applications and subsystems that cannot execute efficiently in conventional distributed system environments.

The combination of Principles 4 and 5 allows us to overcome a significant limitation of prior highly parallel solutions and dispel a commonly held misconception that massively parallel machines can provide only niche solutions. In fact, it is our contention that scalable parallel systems can provide the most general-purpose solutions. Principle 4 supports the execution of all distributed open systems software and Principle 5 at the same time provides competitive solutions for traditional MPP grand challenge (national interest) and high-performance commercial applications.

The first five principles lead us to the high-level system structure shown in Figure 5. The nodes consist of robust, high-function, high-performance RISC System/6000 processors, each running a full AIX operating system. The nodes are interconnected by a High-Performance Switch through communication adapters attached to the node input/output bus (the microchannel). For the current systems, using the microchannel as the interface to the switch subsystem was a practical decision; the standard microchannel interface allows us to rapidly introduce new node technologies into the system while achieving the target goals for latency and bandwidth.

A full AIX image on each node, together with support for standard communication protocols on the

switch (i.e., Internet Protocol), provide full logical support of all standard distributed services. The core of the high-performance services on the SP2 is provided by a high-performance interconnection network, an optimized communications subsystem software and a parallel file system implemented as kernel extensions to AIX, and a parallel program development and execution environment. This system architecture allows us to achieve state-of-the-art performance, price/performance, and scalability in supercomputing environments.

Principle 6. Desired system availability can be cost-effectively achieved with standard commodity components by systematically removing single points of failure that make the entire system unusable, and by providing very fast recovery from all failures.

The structure described so far consists to a large extent of commodity components that are produced for workstations rather than large system environments. In a very large system with hundreds or thousands of commodity parts, failures in the node hardware, node software, and switch will occur frequently enough so that the system must be designed to continue functioning in the presence of failures.

The distributed operating system architecture has some inherent advantages over symmetric multiprocessors. The failure of an operating system image does not have to disable the entire system since the other operating system images can continue to function. Our system approach to high availability relies on this.

This approach requires that the system be configured with sufficient replication (of hardware and software components and data), and a software infrastructure for availability is provided. This infrastructure consists of a set of availability services for failure detection, failure diagnosis, reconfiguration of the system, and invocation of recovery action. The goal of these services is to allow a system to gracefully degrade from N resources to M resources (where $M < N$), and reintegrate the $N - M$ resources later in a nondisruptive manner.

It should be noted that this is merely an infrastructure. To provide real benefit to an end user, all higher level subsystems such as the program development and execution environment, job scheduling, and database and transaction subsystems must exploit the $N \rightarrow M \rightarrow N$ infrastructure and take the appropriate recovery actions nondisruptively.

Principle 7. Selected support for a single-system image through the globalization of key resources and commands, together with a single point of control for systems management and administration, is preferred compared to a true single-system image.

At the level just above the high-availability services, the software system view is that of N AIX images, each of which manages a set of local resources and provides a set of local services. A critical design decision is the level of a single-system image to be supported. Two extreme approaches are possible: the first is to stay with the totally distributed view; the second is to implement a layer of software that makes the N images appear to be one in all respects (a true single-system image).

This is a complex decision since different environments need different views. An interactive user would generally prefer a true single-system image. On the other hand, database subsystems have been written for a distributed environment and expect to see the totally distributed view; these subsystems explicitly manage the different images for performance, and provide a single-system image at the database subsystem level. Finally, for a technical computing user, a single-system image at the source code level and at the UNIX shell level is desirable.

Since providing a true (or complete) single-system image in an efficient manner is a complex undertaking and not a critical requirement in all environments, we have taken a more pragmatic approach. There are clearly key resources (such as disks, tapes, and directory services) that need to be globally known and accessible. Similarly, there are key commands that should be globalized for ease of use. Our approach is to stage in the globalization of these selective resources and commands over time, based on the critical requirements of the applications and subsystems we expect to support. Furthermore, our approach is to provide hardware and software support for the controlling, administering, and managing of N AIX images and nodes in an SP2 system from a single point (i.e., we will also provide a single-system image at the system management level). Similarly,

we will provide a single-system image at the job management level so that a user can submit a job to the system as a whole, and the job management software can automatically select and allocate the required resources to the job from the set of resources available in the system at that time.

Above the global services layer in Figure 4 are a set of subsystems that are primarily built from standard distributed systems technology and tools, with extensions or modifications where necessary.

System overview

In this section we give a brief overview of the SP2 system architecture. We focus on high-level design choices that were made and, where appropriate, the rationale behind them or the implications of those choices.

System architecture. One of the fundamental decisions in the design of a parallel system is the underlying architecture. It is generally understood that symmetric multiprocessors with centralized memory and a single copy of the operating system are not scalable to beyond a small number of processors (typically up to a maximum of around 20 today). Furthermore, the single, system-wide operating system image is a critical single point of failure; an operating system failure can result in the loss of the total system.

In order to scale to hundreds of processors today (and thousands in the future), the SP2 is structured as a distributed memory machine. In such systems, a portion of the total system memory is packaged close to each processor. Access to local memory is fast and remains constant with the size of the system, while access to remote memory is slower. Scalable distributed memory machines can have one of two underlying architectures based on how data are shared—distributed shared-memory architecture or distributed memory message-passing architecture (Figure 6).

With distributed shared-memory architecture, a single global real address space exists across the whole system. All of the physical memory is directly addressable from any node, and a node can perform a load or a store instruction to any part of the real address space. This underlying architecture has the advantage that it generally makes it easier to efficiently support a shared-memory programming model (discussed earlier in the section on design goals). Typically there is a separate operating system (or micro-kernel) image on each node, but they are not independent; the different images are tightly connected at least at the virtual memory manager level so as to present a single global real address space. In such systems, address and data coherence must be maintained in hardware (which makes the hardware complex and costly, and is a fundamental limit to performance and scalability), or in software (which adds to programming or compiler complexity and program correctness exposures, and can potentially affect performance because of conservative coherence management actions).

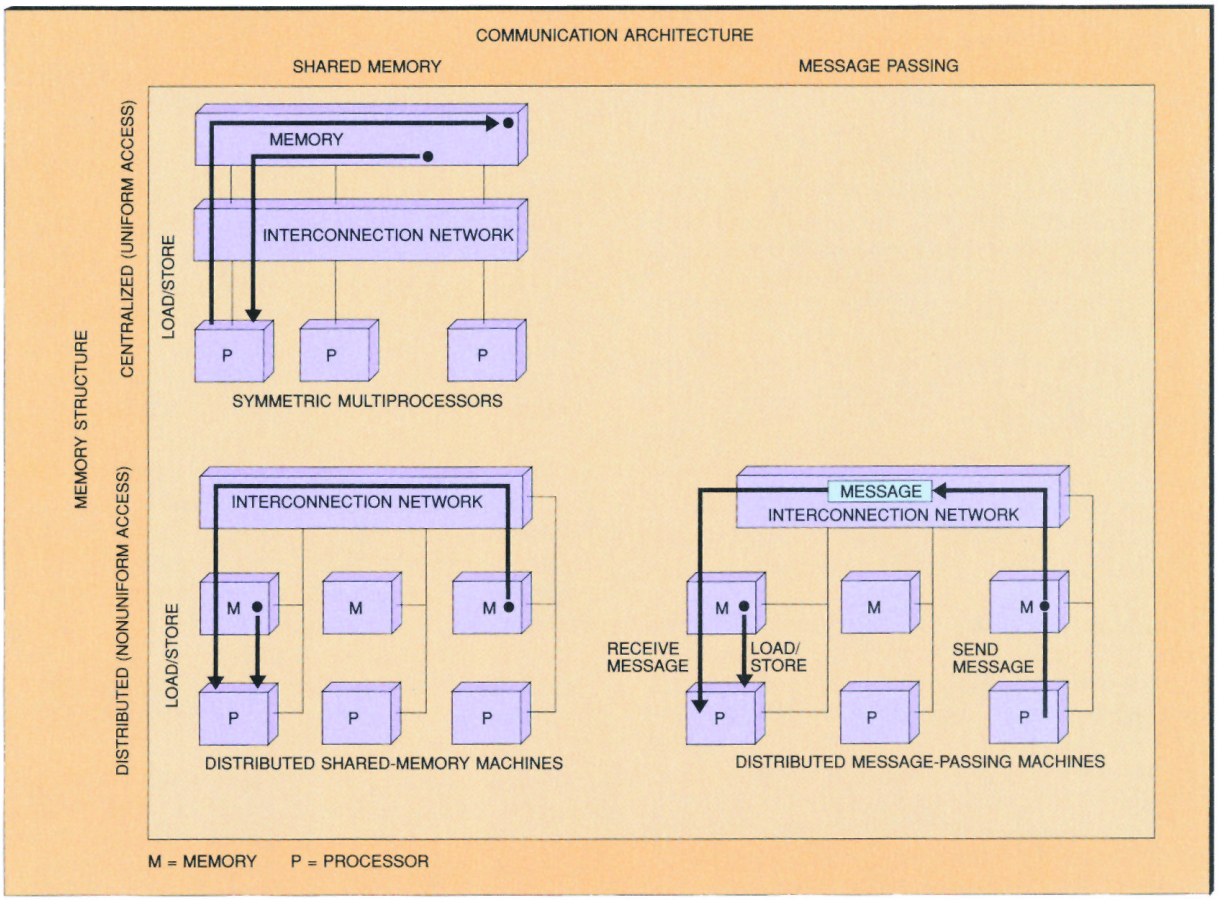
Alternatively, with a distributed message-passing architecture, a processor has direct access (i.e., can perform load or store operations) to only its local memory. Remote memory is not directly addressable and data are shared by explicitly sending and receiving messages. Address and data coherence across nodes is not an issue here.

The SP2 is a distributed memory message-passing machine. Two primary reasons, and a host of secondary reasons, led us to select this architecture as opposed to the alternative distributed shared-memory architecture.

In the alternative distributed shared-memory architecture, a globally shared real (and we mean *real* as opposed to *virtual*) address space implies fundamental changes in the operating system running on these nodes—primarily in the virtual memory management area, but affecting other areas of the operating system. Requirement of such fundamental changes in the operating system would have been contrary to our guiding Principles 1 and 2.

Even more important is our contention that an underlying distributed memory architecture without global real memory addressability is the correct choice for cost-effective scalable parallel systems. Such systems are inherently more scalable at the system level because they do not require tight coordination between the operating system images on the different nodes to provide common address space management and maintain address coherence; nor do they require tight coordination at the hardware level to maintain data coherence. Further, message-passing structures with loose coupling between the operating systems have inherently more availability since it is easier to localize failures to the failing node. Finally, a message-pass-

Figure 6 System architecture alternatives for scalable parallel systems



ing architecture, if properly designed, allows for significant offloading of the overhead associated with communication and overlaps it with computation, especially when one can aggregate the data to be communicated into a few large messages.

In a system with a distributed shared-memory architecture, data are shared implicitly by merely referencing them, and the referenced data from a remote memory are accessed on a cache-line-by-cache-line basis. Although a single line miss may result in a latency and overhead that is very small compared to message-passing architecture, many such line misses would be required to share the data that could have been passed with just one large message.

Programming models. Our goal with the SP2 is to support as many as possible of the dominant programming models being used today in technical and commercial parallel processing, and continue to add others over time. Although introduced earlier in the section on design goals, here we discuss what support is provided for these models in the SP2 system.

For technical computing, our initial focus is on explicit message-passing and data parallel models. Because of the underlying message-passing architecture, clearly a message-passing programming style is the preferred one for performance on the SP2. Several message-passing libraries callable from FORTRAN and C are supported on the SP2. We provide MPL,[30] which is an advanced Message-

Passing Library that is tuned and optimized for the underlying communication hardware of the SP2, and will soon support the emerging Message-Passing Interface (MPI)[31] standard. We also provide the PVMe, which is IBM's optimized version of the message-passing library used in Parallel Virtual Machines (PVM).[32]

The SP2 also supports the data parallel programming model with High Performance FORTRAN.[2] The Forge90 High Performance FORTRAN preprocessor from Applied Parallel Research is available today for the SP2, where the preprocessor converts a High Performance FORTRAN program into an SPMD structure using standard FORTRAN 77 and calls to the MPL. A High Performance FORTRAN compiler for the SP2 is expected to be available in the future. As mentioned earlier, High Performance FORTRAN relieves the programmer from many of the details of data partitioning and communications. In effect, it allows the user to program within a single global name space.

For commercial computing, the function shipping model (previously described in the section on design goals) fits the underlying distributed memory message-passing architecture of the SP2. The other dominant model is the data-sharing model and is supported on the SP2 via the virtual shared disk support (that will be described in the section on system components). Many of the popular databases, on-line transaction processing, and business application subsystems have been enabled to run on the SP2. The typical user does not have to do anything to exploit the underlying parallel capabilities of the system; instead, the application subsystem takes care of decomposing the individual queries to run on multiple nodes or of routing the transaction to the correct server.

Flexible architecture. The SP2 was designed to address a diverse set of application areas and customer environments. Consequently, it was of fundamental importance that the SP2 have a flexible architecture that allowed a customer to customize the system to site-specific requirements and situations.

Figure 7 shows two views of an SP2 system, illustrating its configuration flexibility and operational flexibility.

Configuration flexibility. The SP2 system consists of from 2 to 512 POWER2 Architecture*[33] RISC System/6000 processor nodes, each with its own private memory and its own copy of the AIX operating system, interconnected by a High-Performance Switch (Figure 7A). Each SP2 system also requires a control workstation that is a separate RISC System/6000 workstation that serves as the SP2 system console.

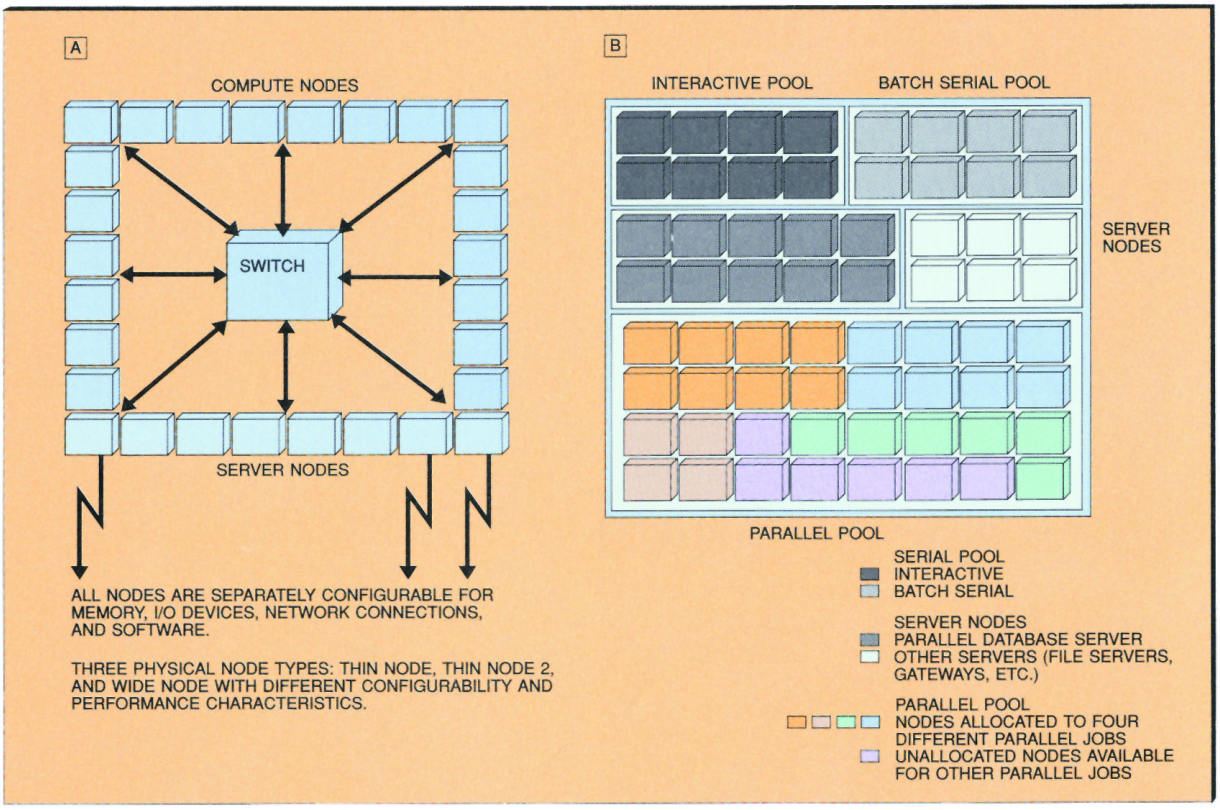
As per Principles 1, 2, and 4, we have made no fundamental change to the base RISC System/6000 processor and the AIX operating system. This means that any of the major hardware or software options available on the base RISC System/6000 workstations can be installed on an SP2 node. Similarly, several thousand RISC System/6000 applications are available immediately to an SP2 customer.

SP2 nodes can be configured to have one of two fundamental logical personalities. Some nodes are configured as *compute nodes* and are used for executing user jobs. Other nodes are configured as *server nodes* that provide various services required to support the execution of user jobs on compute nodes; these could be integrated file servers, gateway servers for external connectivity, database servers, backup and archival servers, etc.

The requirements for node performance and configurability can be very different for compute and server nodes, and for different application areas. As a result, SP2 provides three different physical node types—thin node, thin node 2, and wide node. These nodes differ in their configurability (memory and microchannel slots), performance characteristics, price, and physical size. Each node provides an optimal price/performance point in specific environments.

SP2 is designed for maximum configuration flexibility so that a system can be tuned in a cost-effective manner to a customer's particular requirements. The system can scale up over a very wide range (2 to 512 nodes) in very small increments (one or two nodes). There can be any mix of compute and server nodes within the system. Similarly, there can be any mix of thin and wide nodes (within limits of frame configuration options). Furthermore, each SP2 node can be individually configured for memory, adapters, internal hard disk, and software. This configuration flexibility is an important distinguishing characteristic of the SP2 system compared to other MPP systems.

Figure 7 Flexible architecture features of the SP2 system



Operational flexibility. SP2 systems will commonly be used as enterprise-wide or department-wide servers and therefore must accommodate a mix of job types (serial and parallel, interactive and batch), and accommodate throughput as well as response time requirements of applications. With a large user community sharing the system, the system must function as a throughput engine, concurrently processing unrelated user jobs; however, when response time is important, it must be possible to usurp required system resources and apply them to one or a few large, long-running jobs.

Figure 7B shows the operational flexibility of the SP2 systems in such an environment. Using the job management software, SP2 nodes can be partitioned into different pools for different classes of work—interactive, batch serial, and parallel. Jobs of different classes are channeled to the corresponding pool by the job manager. The nodes within a pool may be shared by several jobs, or they may be dedicated to a single application. In the figure, each small square represents a processor node. Using the job management system software, an administrator can identify different sets of processors for different types of work. For example, a set of eight can be reserved for doing interactive work. Another set of eight can be kept aside for batch serial production jobs.

The figure also shows that some nodes may be set aside as server nodes; server nodes provide specific services that are required by applications running on the compute nodes. In the figure, there is a set of six general server nodes that could be used for specific services such as file servers and/or gateway services; another set of 10 nodes has been configured as a parallel database server.

Similarly, a number of nodes may be set aside for parallel work and form the parallel pool; when a parallel job is submitted, it requests a parallel job

partition with the required number of nodes. The system software locates these nodes from the parallel pool and, if the requested number of nodes is available, they are allocated to the job and the job begins execution in that partition. There can be several parallel job partitions created as long as there are nodes available in the parallel pool. In the figure there are four hypothetical partitions of eight, four, six, and eight nodes each and there are still six unallocated nodes in the parallel pool available for another parallel job. As parallel jobs complete, they return the nodes they were using to the available pool.

Parallel programs generally execute in a dedicated partition of processors; this allows the individual processes to remain loosely synchronized and thus achieve good parallel speedup. It also allows the processes to use direct user-mode communication that bypasses the operating system, thus resulting in better performance. Multiple parallel jobs can be running on the machine simultaneously, each controlling a disjoint set of the nodes. Because of the topology of the interconnection network, there are no restrictions on the size of a parallel job partition or on the topological location of the nodes in the partition; this provides significant flexibility in application design as well as in job management and resource management. This is discussed further in the section on the communication subsystem.

Input/output architecture. The SP2 system provides an input/output (I/O) subsystem that scales in performance and capacity with the computation capability of the system.

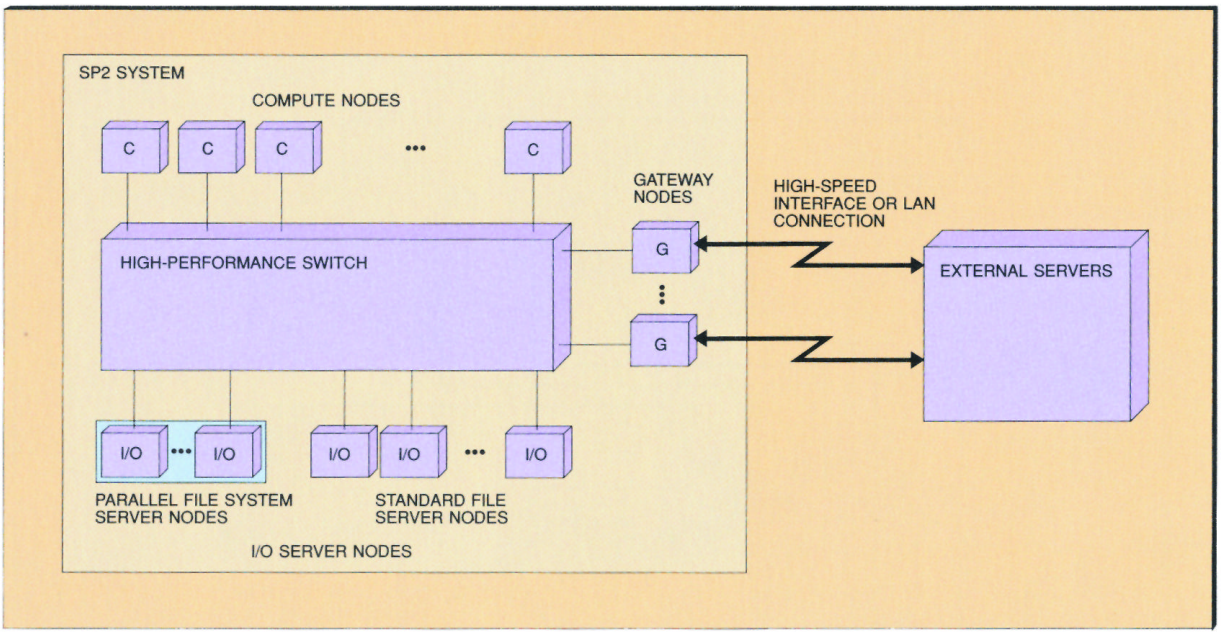
Typically, an SP2 system will be part of an enterprise-wide network, and therefore must have accessibility to network data. In fact, data required by many technical applications are so large that the data are typically stored on less expensive archival storage somewhere in the customer network, and retrieved into the on-line storage only when required by an application. Similarly, because of high-availability requirements, commercial relational databases are typically copied periodically on to similar network backup and archival systems.

External servers connect to the SP2 via intermediate gateway nodes (G in Figure 8). The connection between the gateway nodes and the enterprise servers can be via a local area network, or LAN (Ethernet, token ring, or the fiber distributed data interface FDDI), or a high-speed interface such as high-performance parallel interface (HiPPI), Fiber Channel Standard (FCS), or ATM switches. The external servers provide I/O and file service in response to requests from SP2 compute nodes through a LAN-based distributed file system such as Network File System** (NFS**), Andrew File System (AFS), or Distributed File System (DFS). By using multiple servers and multiple gateway nodes, the aggregate I/O bandwidth can be scaled.

However, the scalability of such a solution is limited and it cannot meet the more stringent requirements of parallel applications for high-bandwidth, low-latency, and low-overhead access to a single shared data object. For high-performance I/O requirements, the SP2 allows I/O and file servers to be integrated into the system by configuring some of the nodes as I/O and file servers (I/O server nodes in Figure 8). Raw I/O capacity and bandwidth can be arbitrarily increased simply by adding more I/O server nodes. With the I/O server nodes directly on the same "attachment fabric" as compute nodes, significantly higher bandwidths are possible than over LANs. More importantly, with clients and servers connected via a reliable switch subsystem rather than LANs, there is the opportunity in the future to move to a tuned, lightweight communications protocol for I/O and file system operations; this would considerably reduce processor overhead compared to the overhead associated with protocols (such as Transmission Control Protocol/Internet Protocol, or TCP/IP, and User Datagram Protocol/Internet Protocol, or UDP/IP) normally used over LANs. Finally, the I/O server node memory can provide sufficient buffer caching for improved I/O performance.

When applications are parallelized for speed improvements, their I/O bandwidth requirement from a single file also increases proportionately. Such bandwidths cannot be satisfied by standard network file systems, even with integrated I/O server nodes; the single server and the single link to the switch ultimately limit the maximum bandwidth that can be supported to a single file. SP2 expects to provide a parallel file system (discussed later) that allows files to be distributed (optionally under user control) across multiple I/O nodes to increase the bandwidth to that file from a parallel application.

Figure 8 The SP2 I/O architecture



Note that the integrated I/O server node architecture logically supports the function shipping model for commercial parallel computing discussed earlier; the clients on compute nodes ship the file requests to the servers on I/O server nodes. The data-sharing model is also supported by providing global access to disks connected to individual nodes as discussed later in the section on global services.

System components

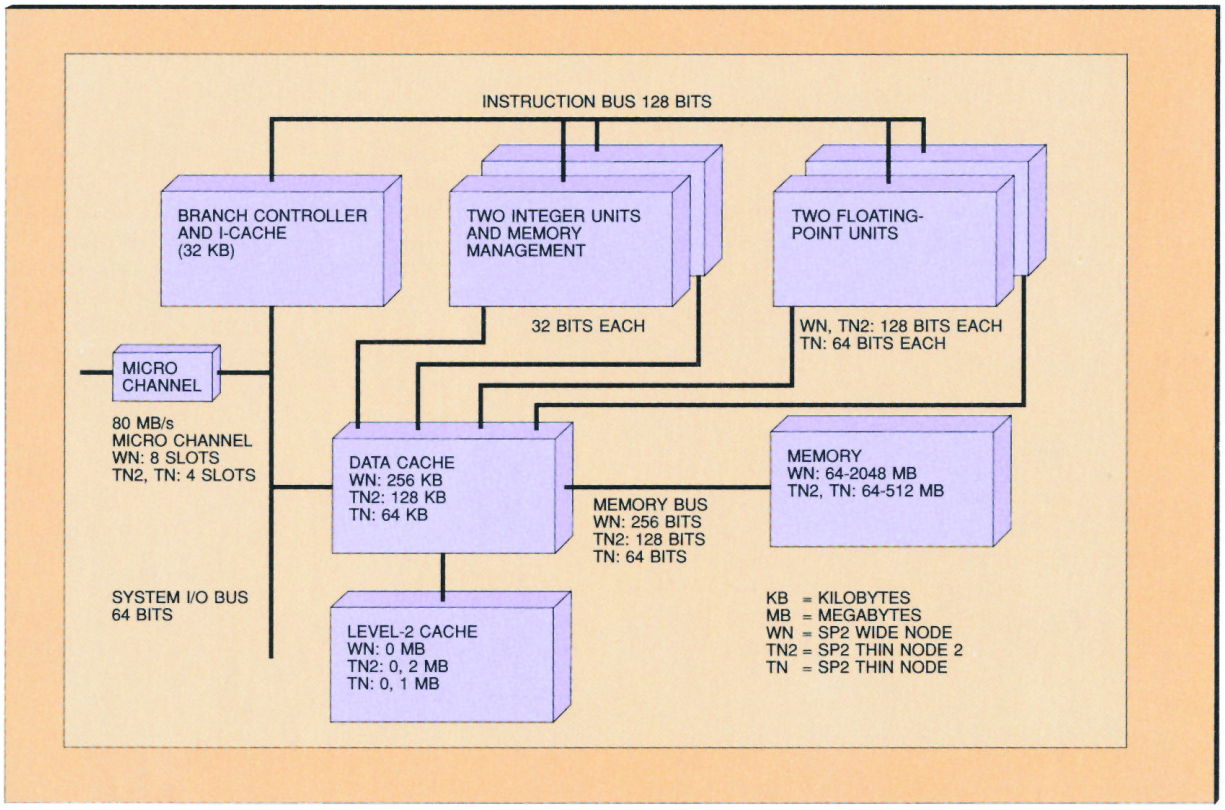
Given the overall system architecture and features described in the previous section, we now look at some of the primary system components in more detail. These components have a direct impact on the essential character of the system—scalability, flexibility, its general-purpose nature, performance and price/performance, system availability, and usability.

Processor nodes. A first-order determinant of performance for both serial and parallel jobs is the individual node performance. Our decision was to use the fastest and most robust processors available to us. This way users can get reasonable performance on their existing serial applications without ever modifying them to run in parallel.

The SP2 nodes are standard POWER2 Architecture RISC System/6000 processors.[33] They are superscalar (issuing and executing multiple instructions concurrently every cycle) rather than merely superpipelined (running the processor at very high frequency), and incorporate many sophisticated organizational techniques to achieve high sustainable performance. As previously discussed in the section on configuration flexibility, the SP2 provides three physical node types—wide node, thin node, and thin node 2. The structure for these is shown in Figure 9.

Both the wide and the thin nodes have two fixed-point units, two floating-point units (each capable of a multiply-and-add every cycle) and an instruction and branch control unit. A 66.7 megahertz (MHz) clock speed gives the processors a peak performance of 267 million floating-point operations per second (MFLOPS). The instruction unit can decode and issue multiple instructions every cycle to keep all the units busy. This is matched with a large-capacity high-performance memory hierar-

Figure 9 The SP2 processor node structure for thin and wide nodes



chy. For example, in the SP2 wide node, the memory can be up to 2 gigabytes (GB) and has a bandwidth of 2.1 GB per second; a 256 kilobyte (KB) four-way set associative cache can supply data at the rate of four 64-bit operands per cycle to the floating-point units. As a result, within a tight loop one can effectively sustain performance equivalent to a two-pipe vector processor (two loads/stores, two floating-point multiply-and-add, index increment and conditional branch every cycle). Features such as short instruction pipelines, sophisticated branch prediction techniques, register renaming techniques, large cache, large memory, and high cache and memory bandwidth all add up to a very robust processor capable of good performance on both vectorizable as well as scalar code. As a result, the processor consistently sustains a very high percent of its peak capability. The SP2 thin nodes are similar to the wide nodes but have a less robust memory hierarchy and I/O configurability.

The robustness of the nodes, both in configurability and performance, is a primary advantage of the SP2 system. In fact, on a processor-to-processor basis, we believe that the SP2 has the most powerful node in any contemporary scalable parallel system; this contributes greatly to the overall SP2 system performance.

Communication subsystem. Because of the high request rate, communication between processes in a parallel job must be high bandwidth and low latency. Traditional AIX (or UNIX) interfaces cannot provide the required performance. As per Principles 3 and 5, SP2 provides a communication subsystem as one of the high-performance services to match the performance requirements for communications within a parallel job. Further, since the communication subsystem is the heart of the system, we paid special attention to making it reliable

and able to recover from most failures automatically (and transparently to the applications).

The SP2 nodes are interconnected by a High-Performance Switch[34] designed for scalability, low latency, high bandwidth, low processor overhead, and reliable and flexible communication between the nodes.

The topology of the switch is an any-to-any packet-switched, multistage or indirect network similar to an Omega network.[35] This allows the bisection bandwidth to scale linearly with the size of the system, which is critical for system scalability. In contrast, the bisection bandwidth of direct networks (such as rings, meshes, or multidimensional toroids) increases much more slowly (or not at all, as in the case of simple rings).

A consequence of the High-Performance Switch topology is that the available bandwidth between any pair of communicating nodes remains constant irrespective of where in the topology the two nodes lie. This is not the case with direct networks. In general, the bandwidth available to a node is equal to $l \times k/h$, where l is the link bandwidth, k is the number of links in the switch per node, and h is the average number of hops through the switch required for a communication operation. In a multistage network such as the High-Performance Switch, the average number of hops, h, and the number of links per node, k, both scale logarithmically with the number of nodes. Thus the bandwidth between any communicating pair of nodes remains constant. This lends considerable flexibility in programming and managing such systems.

In contrast, in a direct network the number of hops, h (for a random communication pattern), increases with the number of nodes, but the number of links per node, k, stays constant. As a result, the average bandwidth per pair of communicating nodes decreases. Because of this problem, programmers have an additional optimization parameter to worry about; they must carefully design the application so as to minimize h by trying to limit internode communications to close neighbors. Additionally, the resource manager and job scheduler must be designed to allocate nodes for a parallel application that are topologically in close proximity with each other, and carefully map the processes to the nodes so as to minimize the average number of hops required by the internode communication in the application. These optimizations are not critical with a topology such as that of the High-Performance Switch.

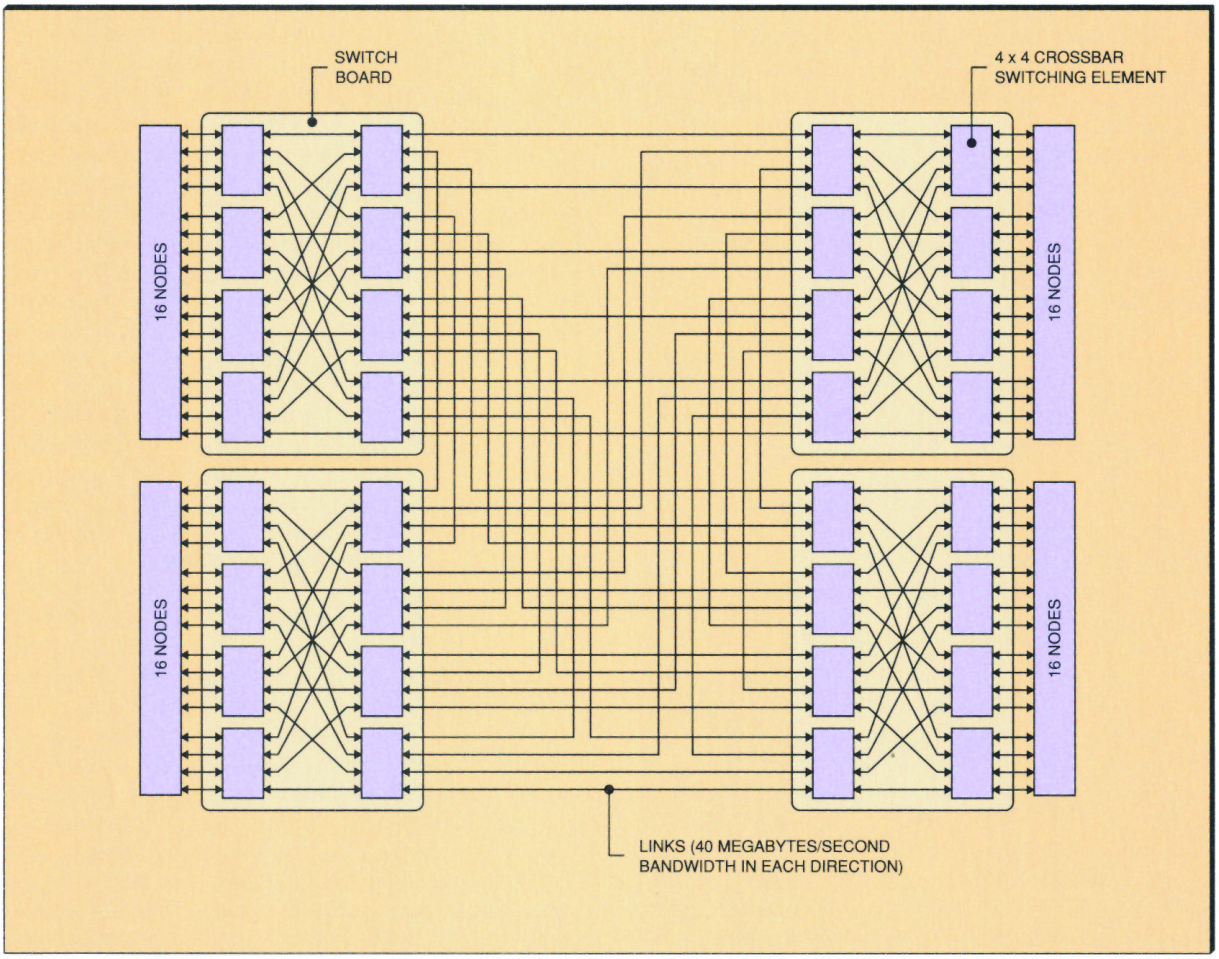
Figure 10 shows the switch structure for a 64-way SP2. The switch is designed to be scalable, with the building block being a two-staged 16 × 16 switch board, made up of 4 × 4 bidirectional crossbar switching elements. Each link is bidirectional and has a 40-megabytes-per-second bandwidth in each direction. The switch uses buffered cut-through worm-hole routing for maximizing performance.[34] In small systems (up to 64-way) only one switch board is required per 16 nodes. For the 64-way system shown in the figure, the required switch boards are packaged within the processor frames (one per frame) and connected via interframe cables to get a four-stage switch network. A *frame* is the "box" that houses up to 16 nodes and one switch board. Additional switch stages are required for larger systems. The extra switch boards for these additional stages are packaged in a special switch frame with up to eight switch boards per frame.

An SP2 node connects to the switch board through an intelligent Micro Channel* adapter. The adapter has an onboard microprocessor that offloads some of the work associated with moving messages between nodes. The adapter can move messages to and from processor memory directly via direct memory access (DMA), thus reducing the overhead on the processor node for message processing and significantly improving the sustainable bandwidth. It also provides protection support for secure user-mode communication between nodes within a parallel partition, without requiring system calls. This allows lower application-to-application message latency by avoiding kernel calls. Message cyclic redundancy check (CRC) code generation and checking is also done by the adapter, further reducing the overhead on the SP2 node. Normal message passing between the processor node and the adapter is driven by polling to avoid the overhead of interrupt processing. The communication subsystem allows shared use of the communications system by both user and kernel tasks; both user-space and Internet Protocol traffic is concurrently supported over the switch.

The communication subsystem hardware and software are designed for reliability and transparent recovery from hard and soft failures.

The switch hardware is fully checked. Each switching element is in fact shadowed by a duplicate

Figure 10 SP2 Switch structure for a 64-node system



switching element so that any error in the switching elements is detected. Similarly, packets on all links carry CRC codes; the code is generated at one end and checked at the other to detect errors in the links.

The switch always contains at least one stage more than necessary for full connectivity. Since the basic switching element is a 4 × 4 bidirectional crossbar, this extra stage guarantees that there are at least four different paths between every pair of nodes. The redundant paths provide for recovery in the presence of failures (as well as reduce congestion in the switch). In the presence of hard errors in the switch (due to a failing switching element or link), the switch can be reinitialized and the routes between nodes regenerated to avoid the failing components.

The communication subsystem software complements the hardware capability to provide transparent recovery of lost or corrupted messages. The communication protocol supports end-to-end packet acknowledgment. For every packet sent by a source node, there is a returned acknowledgment after the packet has reached and been received by the destination node. Thus the loss of a packet will be detected by the source node. The communication subsystem software automatically resends packets if an acknowledgment is not received within a preset interval of time.

When an error is detected by the switch subsystem hardware, the switch enters the diagnostic mode, during which the cause of the error is identified. Thereafter new internode routes are generated, avoiding any failing component or link that was detected, and then the switch is brought back into run mode. At this time, the communication subsystem retransmits the packets that were lost in transit when the failing event occurred. Thus the full error-detection support of the hardware, coupled with the error recovery capability of the software, results in a very reliable and robust communication subsystem for the SP2.

Parallel environment. The SP2 AIX Parallel Environment[36] is an integrated set of components that allow a user to develop, debug, and tune parallel FORTRAN or C programs, and to initiate, monitor, and control their execution.

The application of Principles 5 and 6, and to some extent Principle 2, resulted in a decision to base the Parallel Environment as much as possible on standard AIX tools and techniques and conform to established standards. This allows us to reduce the learning time and maximize ease-of-use for customers. Most commands use familiar UNIX syntax and various AIX tools are made available for use with the Parallel Environment. Program compilation, scheduling, execution, and monitoring are done in manners familiar to UNIX programmers.

The Parallel Environment has four primary components: the Message-Passing Library, the Parallel Operating Environment, the Visualization Tool, and the Parallel Debugger.

Message-Passing Library. The parallel Message-Passing Library (MPL)[29] is an advanced communications library that supports the explicit message-passing model for FORTRAN or C programs. It provides a rich and comprehensive set of functions and subroutines for implementing simple pair-wise communication between processes, as well as more powerful collective communications operations involving user-definable groups of processes.

The MPL is implemented to exploit the SP2 High-Performance Switch using an optimized, lightweight user-space protocol that does not require a kernel call. Alternatively, a user can elect to have the MPL run using the TCP/IP protocol over a local area network connection or over the switch.

Parallel Operating Environment. The Parallel Operating Environment (POE) provides a user environment for initiating, monitoring, and controlling the execution of a parallel application. It can be used to (1) compile and link parallel code with message-passing libraries, (2) create a parallel partition with the required nodes (nodes can be either explicitly specified by the user or selected by the SP2 resource manager based on user-specified job requirements), (3) load the parallel job on the nodes in the parallel partition, and (4) communicate with and monitor the job while it is executing. The parallel application is controlled by a Partition Manager process created by POE on the node or workstation that was used to initiate the application.

The Visualization Tool. The Visualization Tool (VT) provides performance monitoring and trace visualization for a parallel application. Performance monitoring displays system activity in real time while the application is running. Trace visualization is a postmortem process that allows the user to view in detail the interactions between parallel processes, using traces collected during run time. VT can be used to debug an application by identifying deadlock situations and analyzing interprocess communications, and it can be used to analyze and tune a parallel application by identifying performance bottlenecks and load imbalances.

The Parallel Debugger. The Parallel Debugger is a source-level debugger with both a command line and a graphical interface. It is an enhancement of the familiar UNIX dbx debugging tool and incorporates additional functions specific to parallel program debugging.

The AIX Parallel Environment is soon expected to provide support for an optimized version of the MPI message-passing library.

Parallel file system. Standard UNIX distributed file systems (e.g., NSF, ASF, and DFS) do not satisfy the extremely high bandwidth to file data required by some data-intensive applications. A parallel file system (PFS)[37] is one of the high-performance services created for SP2 to address the unique requirements of supercomputing applications that require scalable high bandwidth to an individual file from a parallel application.

PFS supports parallel access (from an application running in parallel on multiple compute nodes) to a file that is striped (i.e., the data are distributed)

across multiple PFS server nodes. It allows the user to control the physical layout of parallel files when they are created—that is, the user can specify how the data in the file are to be distributed into physical partitions across the PFS server nodes. In addition, it allows each process within a parallel application to open a different subfile (or logical file partition) within a parallel file. Thus, processes may open disjoint partitions of a parallel file, and this partitioning may be changed during program execution without changing the physical layout of the file. Processes may also share access to the same subfile (or to the entire file). Accesses to shared files are guaranteed to be atomic and serializable. A scalable serialization protocol is used that does not require locking of the file or parts of it whenever it is accessed. PFS supports files greater than 2 GB in size, and asynchronous I/O. Standard UNIX file system calls (vnode interface) are supported by PFS, and additional capabilities of PFS are made available to users via the UNIX ioctl operation.

Access to the PFS is provided by a PFS client that runs on each compute node. In addition, each PFS server node runs a PFS server. These server processes communicate among themselves and collectively form a parallel server that satisfies requests of PFS clients.

When a PFS (sub)file is opened, the PFS client is provided information about the physical layout of this file. Any subsequent access to this file results in direct communication from the client to any servers that hold parts of the accessed file, with no additional communication to a manager holding meta-data. Each write access results in exactly two messages between the client and each involved server (data are sent and an acknowledgment is received); each read access results in exactly three messages. Data are cached (buffered) only at the PFS server nodes so that no communication is needed for cache coherence. Atomicity and serializability are guaranteed by a protocol executed by the PFS server nodes; this guarantees that accesses that belong to the same transaction are seen in the same order at each server node.

Availability services. The availability services provided by the SP2 system and described below are today used by a limited and controlled set of subsystems. In the future, as the application programming interfaces and the function of these services stabilize, our intention is to make them more generally available. These services would then form a scalable infrastructure that can be used by software developers to build recoverable subsystems and servers for the SP2. As discussed earlier in the section on Principle 6, by using these services, critical subsystems can be designed to gracefully degrade from N resources to M resources (where $M < N$) without disruption of service; later, the $N - M$ resources can be reintegrated into the system in a nondisruptive manner after they have been serviced.

Today, SP2 availability services provide for "heartbeat," membership, notification, and recovery coordination.

Heartbeat services allow processors in the system to be monitored for normal operation.

Membership services allow processors and client processes to be identified as belonging to some defined group; a group is generally formed to include members providing some known and related application service.

Notification services allow members of the group to be notified when membership in the group changes. Membership within a group changes as processors join or leave the group as a result of various node events, such as node restart, shutdown, or a failure.

Recovery coordination services provide a mechanism for initiating recovery procedures within the active group in reaction to node events that change the membership. These services are used to coordinate the running of recovery procedures across the members of the currently active group in response to membership changes.

These services are implemented via *daemons* (unattended programs set up to execute periodically or by the occurrence of an event) running on the nodes. The heartbeat daemons exchange periodic heartbeat messages to determine which nodes are up.

In the event of a failure, the surviving members of the group are notified of the event, and the specific recovery action or procedure is initiated. The recovery procedure is specified by the subsystem that is to be recovered.

Currently these services are used to varying degrees by some of the SP2 system software compo-

nents (such as the virtual shared disk described later) as well as one of the third-party (supporting vendor) application subsystems. As the application programming interfaces and the functions for these services stabilize, we intend to make them more generally available. Over time we expect that all of the critical SP2 subsystems and many other third-party software will use them to provide transparent recovery.

SP2 also supports AIX High-Availability Cluster Multi-Processing/6000 (HACMP/6000*), an alternative software subsystem that allows up to eight SP2 nodes to be configured in a highly available cluster. In the past, HACMP/6000 has been available only for a cluster of RISC System/6000 workstations.

Global services. As discussed earlier in the section on Principle 7, a full single-system image is not a critical requirement for SP2; instead, much of the benefit associated with a single-system image can be derived from providing elements of this functionality as global services. Global access to specific resources such as disks, files, and communication networks is the primary requirement.

In SP2, global access to files is provided today by networked file solutions such as NFS and AFS. These provide for concurrent shared access to file data and are the basis for the globalization of this resource.

Global network access is provided via normal network routing functions and TCP/IP and UDP/IP support over the switch. In this way, SP2 nodes that are not physically attached to an external network still have the ability to communicate through nodes that are physically attached (i.e., through gateway nodes).

Global access to disks is provided by the virtual shared disk[16] support introduced earlier in the section on programming models in commercial computing. Using VSD, an application running at any SP2 node can transparently access any disk, physically located on any other SP2 node, as if it were locally attached to that node. This is done by trapping a request for a remote shared disk at the disk driver level and shipping the request to the corresponding node. In effect, VSD is a device driver layer that sits on top of the AIX Logical Volume Manager and exports a raw device interface. If the access is to a shared disk that is locally connected, the VSD layer passes the request directly on to the Logical Volume Manager on that node. If, however, the access is to a shared disk attached to a remote node, the VSD layer sends the request to the VSD on that remote node (through the switch), which in turn passes it on to the remote Logical Volume Manager for access. The response is returned to the VSD on the originating node and to the requesting application.

The primary requirement is global access to resources such as disks, files, and networks.

VSD is highly optimized for performance, both in terms of remote disk access bandwidth, and in terms of the overhead on the processor to access remote data. For instance, when a request is shipped to a remote node, causing a communication adapter interrupt at the remote node, the corresponding disk access is initiated at the interrupt level. By contrast, network file systems such as NFS need to schedule a daemon at the remote server node, and consequently have a much higher processor overhead and lower bandwidth than VSD. The aggregate bandwidth supported at a server node is about an order of magnitude higher than that for remote access for typical network file systems. Remote disk access for a single client using VSD is close to the disk bandwidth for sequential access, and slightly lower than the local access rate for random access of small data blocks.

VSD provides transparent switching to an alternative server for recovery. For this feature, each VSD server is logically paired with an alternate secondary server. The disks must be twin-tail attached to both the primary and secondary server nodes. However, only the primary tail is active normally. The heartbeat service is used by the primary and secondary nodes to monitor one another. If the secondary node detects a primary node failure, control of the disks attached to the failing node is transparently switched over to the secondary server, and requests to those disks continue to be serviced

by the secondary server node transparently to the application. The SP2 availability services are used by VSD to implement this transparent recovery capability.

Currently, VSD is primarily required by database subsystems based on the data-sharing architecture that requires access to disks from any node.

Another global service is the system data repository (SDR). The SDR is a distributed data repository that provides data storage and retrieval across nodes of the SP2 and the control workstation. It contains system-wide information about the nodes and switches and their configuration, and about the jobs currently in the system. It is used by system management, hardware monitor, parallel file system, switch subsystem, and Resource Manager.

The SDR is the result of an important design decision to require that critical system data (such as job management and system management data) not only be local to the servers that implement the function; instead, the server data must be maintained externally in a system-wide repository (that being SDR). The rationale for this is twofold. First, it eliminates redundancies and inconsistencies between servers using the same data. But more importantly, it is the first step toward solving the server availability problem in large scalable parallel systems. The goal is that critical global servers (such as the Resource Manager and file systems) should be restartable. If a primary system server fails, then it should be possible to restart the server without affecting other servers and user jobs in the system in any catastrophic manner. During the period of time when the failing server is unavailable, requests to that server will cause the requester to be delayed, but there should be no other sympathetic failures caused by the server outage. The way that a failing server gets restarted is by inspecting the SDR objects that contain all the necessary information for it to bring itself back to its state prior to failure.

System management. The SP2 system management and system monitoring software provides the administrator with a single point of control for managing and monitoring the SP2 system. The underlying philosophy was to build upon many of the system management tools, commands, and interfaces already available for the AIX/6000* workstation environment. SP2 system management extensions facilitate performing traditional system management functions in a system with multiple nodes, each running its own independent copy of the operating system. These functions include system installation, system operation, user management, configuration management, file management, security management, job accounting, problem and change management, hardware monitoring and control, and print and mail services.

The single point of control is via the control workstation that acts as the system console. It is a separate RISC System/6000 workstation that connects to each SP2 frame (the nodes and switch board) in the system. By logging on to the control workstation from anywhere on the customer network, an administrator can remotely manage the entire SP2 system from a central location.

The system management software uses the UNIX distributed shell command dsh and the secure client/server system sysctl for initiating system management commands from a single point of control and have them execute in parallel on multiple nodes. Sysctl is used for executing remote commands in parallel and uses the UNIX security package Kerberos for authentication. Filters are provided to facilitate specifying which nodes to operate on and for modifying and consolidating the responses in order to make them more comprehensible.

Hardware monitoring allows the administrator to monitor and control the state of the SP2 hardware. Each node, switch board, and frame in the SP2 system has a supervisor card that provides sensing of environmental conditions and control over the hardware components. Through this facility, an administrator can power nodes and switches on and off, reset nodes or switches, change the key switch position of a node, and monitor the node displays, node and frame voltage levels, node temperature, and certain hardware failures. Information can be viewed at various levels of abstractions—single node, frame, switch or total system—all from the control workstation.

Job management. Users access the SP2 system through an external network, either in batch mode or interactive mode. In batch mode, jobs are submitted using network job scheduling software. Typically that will be LoadLeveler*[38] or NQS.[39] Currently, LoadLeveler is the only supported mechanism for submitting parallel batch jobs. For interactive access, a user logs on to the SP2 sys-

tem via the standard distributed UNIX rlogin command. The logon can be directed to a specific SP2 node by using the corresponding network address. Alternatively, the SP2 host name can be specified, in which case the user is given logon access to a lightly loaded SP2 node.

The SP2 Resource Manager is a parallel job server that runs on one of the SP2 nodes and manages the allocation of nodes to parallel jobs entering the SP2 system. The nodes of an SP2 system can be divided into a serial pool and one or more parallel pools. Nodes in a parallel pool are used exclusively for running the processes of parallel jobs. Nodes in the serial pool are used to run serial jobs and to run the Partition Manager for a parallel job.

When a parallel job is initiated (either interactively or as a batch job via LoadLeveler), a Partition Manager process for that job is started on a node in the serial pool. The Partition Manager connects to the Resource Manager and requests the required number of nodes; if available, the Resource Manager returns a list of the nodes that would form the parallel partition for that job.

Once the parallel partition is established, the Partition Manager loads a copy of the executable code for the job and starts a Partition Manager daemon on each of the nodes in the partition. The Partition Manager daemon on each node then starts execution of the process. When the application ends (either normally or abnormally), the Partition Manager is responsible for cleanup and orderly termination, and for returning the nodes in the partition to the Resource Manager.

If the node on which the Resource Manager is running fails, a backup Resource Manager is automatically started on another SP2 node. Transparent recovery is accomplished by the backup Resource Manager reinstating the original state by retrieving all job and node resource data from the system data repository.

Reliability and availability features. In the design of the system we paid particular attention to minimizing the probability of catastrophic failures. A catastrophic failure is defined as a complete system failure and restart is not possible, even in degraded mode, until service is applied. In particular, the system is designed to be tolerant of single node failures; the effect of a failing node is isolated to applications using that node. Independent operating system images on each node make this possible. As previously described in the section on the communication subsystem, the SP2 communication subsystem is designed for transparent recovery from most common hard and soft errors. Support for Redundant Array of Inexpensive Disks (RAIDs), multi-tailed devices, and mirroring is provided for storage media recovery.

In the most common situations, hardware and software service or upgrade can be applied to a node or a part of the system while the rest of the system remains in operational mode. A node in an SP2 system can be powered off and disconnected from the rest of the system to perform repairs on the node, and later replaced and "powered-up" while the rest of the system is operational.

The SP2 frames are designed for reliable, remotely controlled operations with concurrent maintenance and upgrade capability. Each frame has redundant main power supplies to reduce the chance of a system outage due to a power supply failure.

Support for transparent recovery of software subsystems and servers is provided by the SP2 availability services, as described earlier. Currently these services are used by the VSD and the Resource Manager as well as one of the database subsystems. As these services stabilize and are made more generally available, we expect many of the other SP2 system software components and other third-party software subsystems to use them to provide transparent recovery.

Similarly, the system data repository (SDR) provides a convenient mechanism for externalizing critical system data from the servers that use them. This way servers can be designed to be restartable after a failure using a backup copy on another node and re-establishing the original state by retrieving the data from the SDR. Currently the SP2 Resource Manager uses this capability to provide transparent recovery. Over time we expect other critical system servers to use this capability.

Performance

The primary determinant of system performance for a parallel system is the performance capability of the two primary building blocks—the individual nodes and the communication subsystem used to interconnect the nodes. We first discuss the ca-

Table 1 SP2 node performance where the data for Linpack 100 and Linpack TPP are in MFLOPS, and SPECint92 and SPECfp92 are in SPEC units

Benchmark	Thin Node	Thin Node 2	Wide Node
Linpack 100	55	132	132 (MFLOPS)
Linpack TPP	183	227	237 (MFLOPS)
SPECint92	114	122	122 (SPEC UNITS)
SPECfp92	205	251	260 (SPEC UNITS)

pabilities of these building blocks in the SP2 and then look at system-level performance.

Table 1 shows the performance of the three different SP2 node types for several common benchmarks. SPECint92 and SPECfp92[40] are, respectively, suites of integer and floating-point code fragments defined by the System Performance Evaluation Cooperative (SPEC); these fragments run entirely out of the processor cache on typical microprocessors and therefore do not stress the memory subsystem. Linpack 100 and Linpack TPP benchmarks[41] are solutions of relatively small systems of linear equations, but typically do not fit in the processor cache and therefore represent a different aspect of technical computing application performance than SPECfp92. Thus the Linpack and SPEC results should not be compared with each other. Linpack performance of SP2 nodes is significantly higher than that of nodes used in other scalable parallel systems available today.[41] The powerful nodes with robust configuration capability are a fundamental advantage of the SP2 system and are one of the primary contributors toward the good system-level performance of the SP2, as we show later.

The primary standard benchmarks in commercial processing are benchmarks defined by the Transaction Processing Council (TPC),[40] which also publishes the results of the benchmark measurements on different systems. There are several TPC benchmarks representing different types of commercial workloads. The TPC-C benchmark is representative of database transactions in a retail operation. Again, considering single processor performance, the SP2 nodes are considerably more powerful than the nodes used in other scalable parallel systems.[41] For example, the published TPC-C measurement for the SP2 wide node (which is equivalent to the RISC System/6000 Model 590) is 726.1 tpmC (TPC-C transactions per minute) with a corresponding $/tpmC of 1395.

Generally speaking, the SP2 wide node provides the highest performance in most environments; the SP2 thin node 2 is typically within 10 percent of the wide node performance, while the SP2 thin node is typically 20 to 30 percent lower. In environments where performance is gated by the amount of node memory and I/O configurability, the wide node generally provides the best solution; otherwise, the thin nodes generally provide better price/performance.

The other primary performance determinant is the communication subsystem. Table 2 shows latency and sustainable bandwidth measurements using different SP2 nodes. The latency measures the time to send a zero-byte message from one process space to another on a remote node; it accounts for all hardware and software path lengths between the sending application process and the receiving process. The point-to-point bandwidth measures the sustainable rate at which a process running on one node can transfer data to another application process running on another node; it assumes very large messages, one-way data transfer, and accounts for all hardware and software overheads involved in the transfer path. Finally, the exchange bandwidth measures the sustainable rate at which two application processes running on two different processors can exchange data in a simultaneous send and receive (two-way transfers). The performance is measured using the IBM Message-Passing Library (MPL), both with user-space communication over the switch and also through UDP/IP over the switch. Both paths are simultaneously available to users and, depending on the application, either or both may be used.

The sustainable pair-wise bandwidth of the SP2 is comparable to that of other scalable parallel systems available today. Many of the scalable parallel systems available today provide higher switch link bandwidths than SP2, but when the switch topology effects, message payload capability, and other system features are taken into consideration, the sustainable bandwidth between random pairs of processors in the system is comparable to that of the SP2 system.

However, compared to distributed shared-memory systems (such as the Cray T3D), the latency of

Figure 11 SP2 performance for the Linpack HPC benchmark

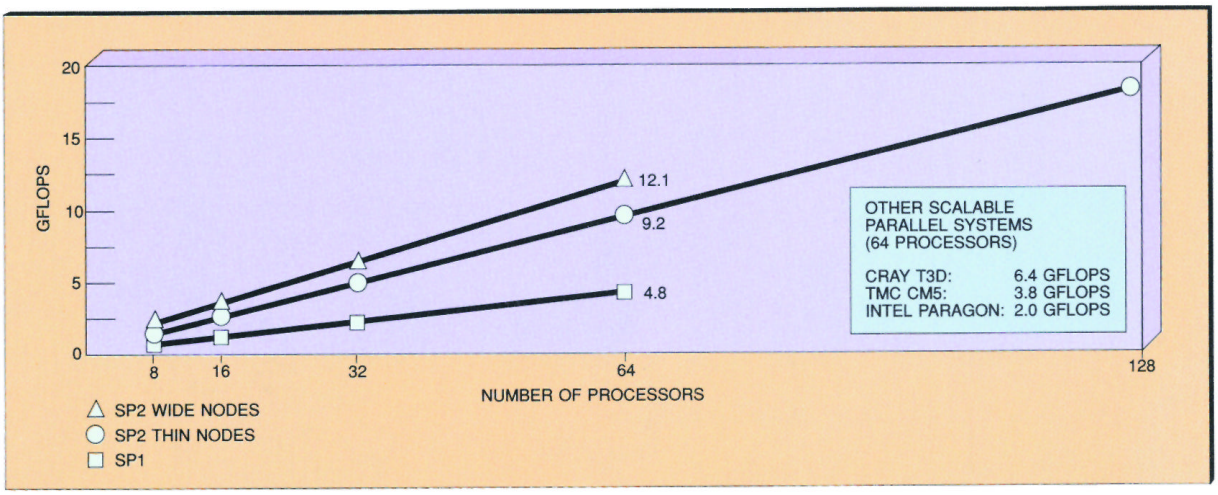


Table 2 SP2 communication subsystem performance

Node, Communication Mode	Latency (microseconds)	Point-to-Point Bandwidth (MB/s)	Exchange Bandwidth (MB/s)
Thin node, user space	40	35	41
Thin node 2, user space	39	36	48
Wide node, user space	40	36	48
Thin node, UDP/IP	312	10	13
Thin node 2, UDP/IP	270	12	16
Wide node, UDP/IP	269	12	16

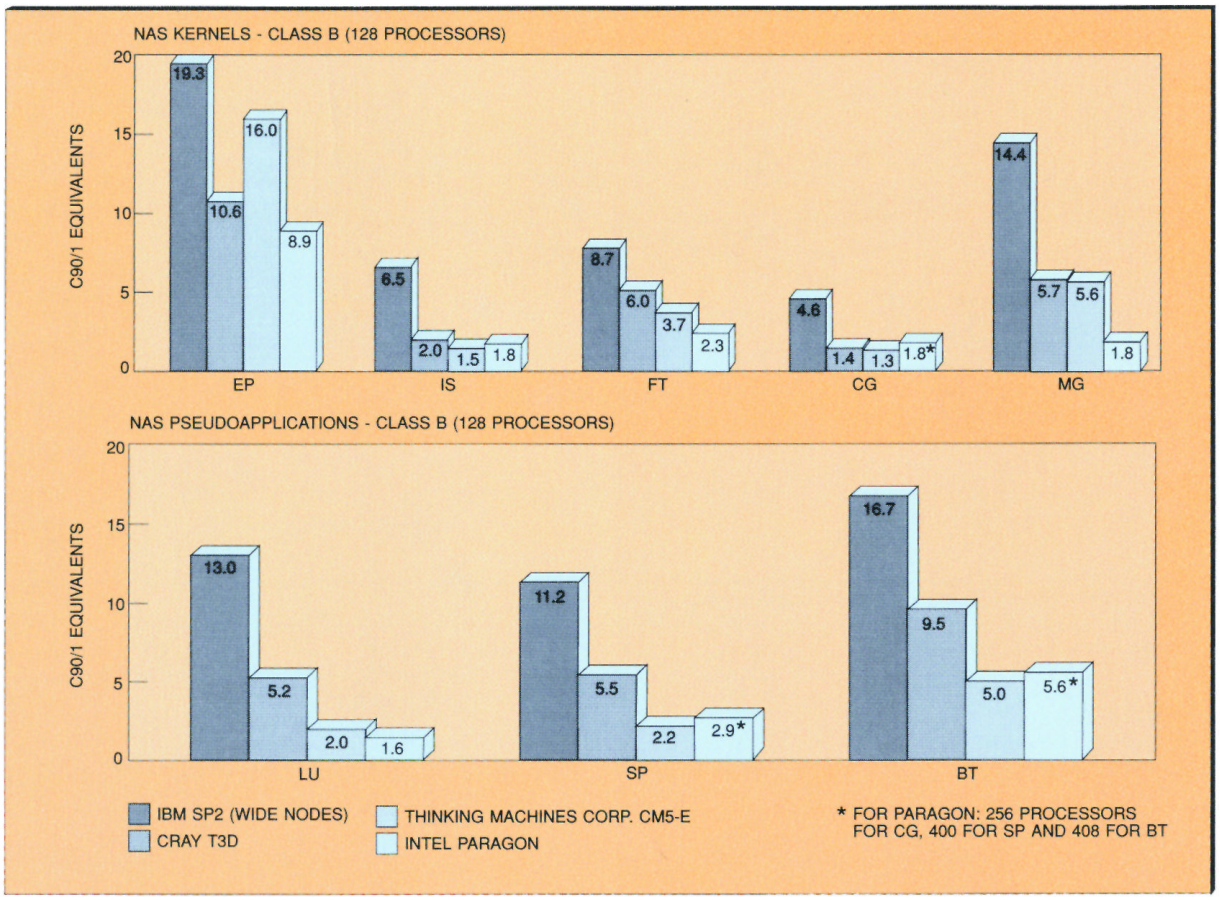
the SP2 system is higher. This can affect parallel application performance adversely if the application is structured for frequent fine-grained communication. But it is important to remember that distributed shared-memory systems access data synchronously and in units of cache line size. In contrast, a distributed message-passing system can transfer data asynchronously and in very large units. In situations where the data to be transferred can in fact be aggregated into one large message, the longer latency of the message-passing machine is not an issue, and the asynchronous nature of the transfer can be an advantage.

For parallel application execution capability, there are two standard benchmarks that are widely used today, namely the Linpack HPC and the NAS Parallel Benchmark suite.

The Linpack HPC is functionally similar to the other Linpack benchmarks discussed earlier, but it does not specify any problem size; instead, it allows the problem size to grow with the size of the system under test. Its performance is a function of the floating-point capability of the nodes, the memory size, and the interprocessor communication performance of the system. Figure 11 shows that for an equivalent number of nodes, the SP2 delivers significantly higher performance than other scalable parallel systems that have reported the Linpack HPC performance.[41] Other systems shown are the Cray T3D, the Thinking Machines CM5**, and the Intel Paragon.

The NAS Parallel Benchmark suite, developed by the Numerical Aerodynamic Simulation (NAS) project at the National Aeronautics and Space Admin-

Figure 12 NAS Parallel Benchmark performance for IBM SP2 and selected other scalable parallel systems

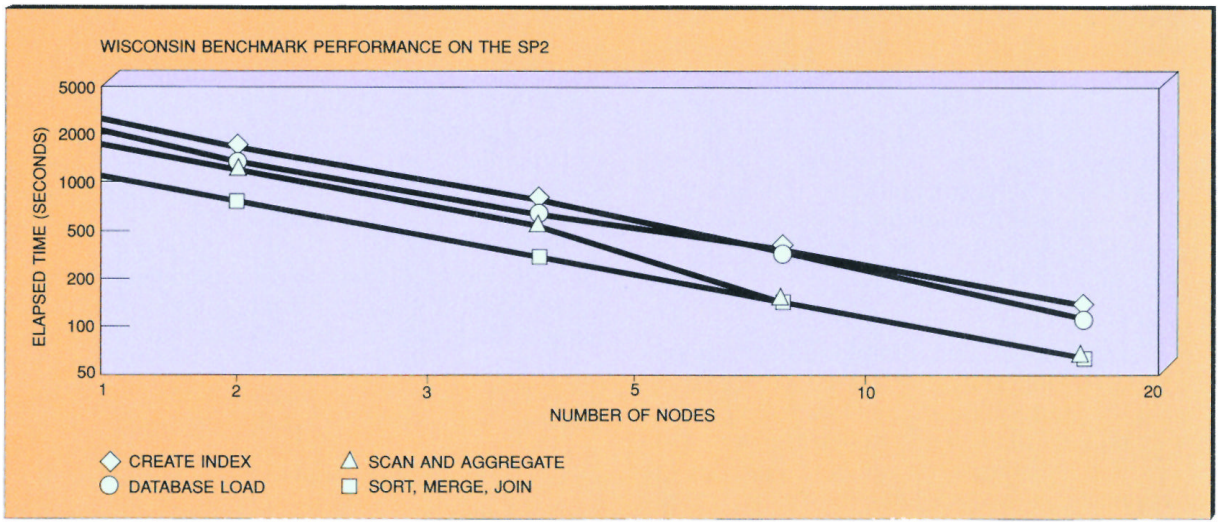


istration (NASA) Ames Research Center,[42] consists of five kernels and three pseudoapplications that collectively represent the NASA/Ames scientific computing workload. The kernels have simple data structures and each represents a fundamental computational structure in NASA's aerodynamic simulation applications. The simulated applications have multiple interacting data structures, and are more closely related to the full applications in use by NASA scientists. The kernels address specific components of system performance, while the simulated applications examine the overall system performance that is attained by combination of the components.

Figure 12 shows the performance of the SP2 system (using wide nodes with a fully configured memory subsystem) using the Class B NAS Parallel Benchmarks, compared to some other massively parallel systems, as reported in Reference 43. The results are shown in Cray C90 single processor equivalents, and are for 128-node systems (except for three cases on the Intel Paragon where data are only available for a different number of processors, as noted in the figure). The SP2 significantly outperforms other equivalent size systems, especially on the pseudoapplications that are closely related to full applications in use by NASA scientists. The implementations of the NAS benchmarks on the SP2 are described in depth in References 44 and 45.

To characterize the SP2 relative to commercial computing requirements, we started with intrinsic database operations for decision support environments and have tested their scalability on SP2 systems with increasing numbers of nodes. For

Figure 13 Performance of the SP2 using the Wisconsin Benchmark set (1 GB database size) for decision support environments, and using the IBM DB2 Parallel Edition relational database subsystem



these measurements we have used the Wisconsin Benchmark[40] that tests the following basic operations on a 1 GB database: (1) load database, (2) create index, (3) scan database and aggregate result, and (4) sort, merge, and join.

It is important to realize that when evaluating the performance of the SP2 system in this context, the performance of the database subsystem itself is as significant as that of the hardware platform. The two database subsystems that have been ported for parallel execution on the SP2 are the Oracle Parallel Server (Version 7.1) and the DB2* Parallel Edition. The elapsed time for these operations on SP2 with DB2 Parallel Edition is shown in Figure 13 and shows generally linear speedup with good parallel efficiency. Measurements for Oracle 7.1 will be available in the future. We also expect to do measurements on TPC-D (decision support) and TPC-C (on-line transaction processing) benchmarks defined by the Transaction Processing Council.

Key future challenges

We expect that the future evolution of SP2 systems will continue to be guided by the principles that we articulated in the section on system strategy. In particular, we will continue to exploit workstation technology as much as possible. However, as the market for the scalable parallel systems expands, we will continue to assess technologies that are specifically targeted to scalable parallel computing.

A symmetric multiprocessor as the basic node in a scalable parallel system is an attractive option for future systems. It can provide significant improvement in peak node performance at a cost that is not much higher than the cost of a uniprocessor with the same amount of storage. However, effective use of this performance is a challenging problem. In particular, one needs to manage two levels of parallelism, with big differences in performance and function between intranode communication and internode communication. Hiding the complexity of two levels of parallelism by providing transparent compiler exploitation will be crucial.

The communication subsystem in scalable parallel systems must continue to improve in performance and capability. Efficient support of a shared-memory programming model requires aggressive bandwidth increase and latency reduction. This precludes interfacing to the switch via the I/O bus in future SP2 systems. Direct connection to an internal memory bus will lead to improved performance, but will require a more specialized package and interlocking of adapter technology and processor technology.

Aggressive latency reduction will require a communication architecture that will allow a processor to access remote memory without involving the remote processor. Such functionality is already available in one form or another on some systems. However, current implementations either compromise on performance (latency), or require a global real address space. The challenge is to provide such functionality with very low latency on a virtual memory system with minimal impact on system interfaces, while still maintaining the distributed message-passing system model we have adopted for scalability and availability.

Very large and scalable information servers (including video and multimedia servers) require the sharing and movement of massive amounts of data among nodes. A key metric here is the processor overhead for moving data. Minimizing processor overhead and supporting large throughput, rather than low latency, is critical.

There is limited experience today with scalable storage systems and scalable file systems, and hence limited knowledge to guide system architects in the design of future parallel I/O subsystems. Yet low-cost, scalable I/O is key to the success of scalable parallel computing. The High-Performance Storage Subsystem (HPSS) initiative (a cooperative effort between key government labs and several vendors of high-performance systems) has generated solutions to some of the key issues in this area. The challenge is to quickly incorporate these solutions into mainline products.

Efficient utilization of future scalable parallel computers will require major advances in software technology. Efficient compiler technology is needed to support more convenient programming models while preserving portability to other platforms. Future systems will have to support efficient time sharing of processors by parallel jobs, and preemptive scheduling of parallel computations. This requires a mechanism for coordinated dynamic allocation of processors and memory to the processes that constitute a parallel job; in particular, support for gang (simultaneous) scheduling is required. Also, users should be able to control the execution of parallel jobs (e.g., via a parallel shell), and connect parallel computations (e.g., via parallel pipes), with the same convenience with which they control or connect individual processes today.

The availability of parallel application codes and subsystems is critical for the success of scalable parallel computing. The development of such codes is hampered not only by the difficulties of parallel programming, but also by the lack of accepted standards in this arena. High Performance FORTRAN and the Message-Passing Interface (MPI) are two successful examples of establishing such (*de facto*) standards. Much more is needed. In particular, standards are needed in the area of parallel system services, such as parallel I/O.

Another aspect, particularly in commercial applications, is that of providing coordinated and consistent recovery between a large number of subsystems running on any scalable parallel system. These could include multiple database systems, a transaction monitor, a parallel file system, a multimedia server, resource managers, the operating system, the global I/O subsystem, etc. Each subsystem may detect several software failure conditions on its own, and these need to be combined with other failure detection mechanisms (such as the heartbeat service), for overall diagnosis. Ordering, synchronization, and parallelism during recovery of the various subsystems is needed. Providing a high-availability infrastructure that addresses these issues is a challenging problem.

Meanwhile, the software technology used in sequential systems is progressing. Object-oriented (OO) programming and object-oriented storage (object-oriented databases, object-oriented file systems) are becoming more prevalent; operating system technology is evolving to better support OO. Future scalable parallel systems will have to cope with this evolution, both because of its impact on the basic uniprocessor software technology, and because of the importance of OO for future applications. Scalable OO technologies are yet to emerge from the research community. Yet such technologies are likely to have a major impact on the future of scalable parallel systems.

Summary

In this paper we described the seven guiding principles that form the basis for the SP2 system architecture, and how these influenced the system overview and system components. Our contention was that these principles should let us bring to market scalable parallel solutions in a timely manner for a wide range of applications. This has largely been borne out by the success and broad-based market

acceptance SP1 and SP2 systems have enjoyed since the SP1 systems became generally available. Around 400 SP1 and SP2 systems (with 2 nodes all the way up to 512 nodes) are today being used productively in many different areas, including computational chemistry, crash analysis, electronic design analysis, seismic analysis, reservoir modeling, decision support, data analysis, on-line transaction processing, local area network consolidation, and as workgroup servers. They are being used in diverse industry areas including manufacturing, process, distribution, transportation, petroleum, communications, utilities, education, government, finance, insurance, and travel.

Time-to-market with the latest technology has been one of the keys to the success of the SP1 and SP2. The SP1 system was introduced in the market roughly one year after the POWERparallel system* program was started. This was accomplished through Principles 1, 2, and 3 by using many standard RISC System/6000 hardware and software components and leveraging interprocessor communication technology developed in IBM Research. The SP2 was delivered less than a year after the SP1. The loosely coupled system structure allowed significant enhancements in the communication protocols and a very fast introduction of the new POWER2 microprocessor.

Another key to success has been the availability of key applications. A wide range of IBM and vendor applications and subsystems run in parallel on the SP2. These applications span a spectrum of areas: computational chemistry and pharmaceuticals, engineering analysis, electronics analysis, and petroleum explorations and production in the technical computing area; database management systems, transaction processing monitors, and business applications systems that run on top of database subsystems, for the commercial computing area; and general tools for system management, network management, and storage management. Most are currently available.

Principles 1, 4, 5, and 7 are the primary reasons behind the broad portfolio of parallel applications. Support of Principles 1 and 4 also contributes to the several thousands of RISC System/6000 applications available on the SP2. The support for standard distributed AIX program development and execution environment, key programming models, standard message-passing libraries, globalization of key resources, and availability services has allowed independent software vendors to modify key applications and subsystems to run on the SP2 without diverting from their mainstream development strategies.

Our strategy of using standard microprocessors and a mature operating system, complemented with a set of high-performance services has provided other benefits. The system provides industry-leading performance as shown by most key benchmark results. The system installs easily and exhibits good availability characteristics. The flexible system architecture and a variety of applications allow users to start doing real work immediately. For example, a large 400-node SP2 has been installed at the Maui High-Performance Computing Center. Within a few weeks of delivery, Professor Frank L. Gilfeather, one of the principal investigators, provided the following testimonial: "We are seeing availability (considering down time from all causes) of 98 percent and are achieving 70–90 percent CPU utilization. Compared to other MPP machines, the SP2 has set a new standard in ease of installation and use, availability, reliability, and cost/performance."

This broad-based acceptance attests to the flexible and general-purpose nature of the architecture and validates the principles we used to design the SP2 to effectively address the requirements of a wide range of applications and industries.

Acknowledgments

Ray Bryant, Daniel Frye, Kevin Gildea, and Don Grice were part of the SP2 systems team and made significant contributions to the SP2 systems architecture. Christos Polyzois and Jim Rymarczyk were instrumental in the definition of SP2 architecture enhancements for commercial computing.

The overall success of SP2 is the result of contributions of many other people in all the various phases in the development and delivery of a system—system architecture and system design, hardware and software design and development of the different system components, system test, manufacturing, applications enabling, performance benchmarking, marketing support, and service. They are too numerous to acknowledge them all individually here but their contributions were critical.

Much of the success can also be attributed to the close working relationship between the IBM POWER Parallel Division and the IBM Research Division, and others from the RISC System/6000 hardware group, the former Federal Systems Marketing Division, IBM Software Solutions, and the European Center for Scientific/Engineering Computing. Joint partnerships with some of our key customers helped guide and validate many of our architecture and design decisions.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of X/Open Co. Ltd., Tandem Computers Incorporated, Sun Microsystems, Inc., or Thinking Machines Corporation.

Cited references

1. D. E. Nielsen, "A Strategy for Smoothly Transitioning to Massively Parallel Computing," *Energy and Technology Review*, University of California, Lawrence Livermore National Laboratory (November 1991).
2. C. H. Koelbel, D. B. Loveman, R. S. Schreiber, G. L. Steele, Jr., and M. E. Zosel, *The High Performance FORTRAN Handbook*, The MIT Press (1994).
3. *Database 2 AIX/6000, Programmer's Reference Manual*, SC09-1573-00, IBM Corporation; available through IBM branch offices.
4. *Database 2 Parallel Edition*, White Paper, Database Group, IBM Software Solutions Toronto Lab., 1150 Eglinton Ave. East, North York, Ontario M3C 1H7, Canada.
5. *Oracle for Massively Parallel Systems—Technical Overview*, Oracle White Paper, Oracle Corporation (March 1993).
6. M. Ferguson, "Parallel Query and Transaction Processing," *Info DB (USA)* **7**, No. 3, 18–27 (Summer 1993).
7. D. Clay, *Informix Parallel Data Query (PDQ)*, IEEE Computer Society Press (January 1993).
8. *Customer Information Control System/6000 Storage (CICS/6000): Technical Overview*, GC33-1225, IBM Corporation; available through IBM branch offices.
9. M. Sherman, *Distributed Transaction Processing with Encina*, IEEE Computer Society Press (January 1993).
10. J. M. Andrade, M. T. Carges, and M. R. MacBlane, "Open On-line Transaction Processing with the Tuxedo System," digest of papers from the IEEE 37th COMPCON (Spring 1992).
11. D. W. Cornell, D. M. Dias, and P. S. Yu, "On Multisystem Coupling Through Function Request Shipping," *IEEE Transactions on Software Engineering* **SE-12**, No. 1, 1006–1016 (October 1986).
12. M. Stonbraker, "The Case for Shared Nothing," *IEEE Database Engineering* **9**, No. 1 (1986).
13. N. P. Kronenberg, H. Levy, and W. D. Strecker, "VAXClusters: A Closely-Coupled Distributed System," *ACM Trans. Comput. Syst.* **4**, No. 2, 130–146 (May 1986).
14. A. Sekino, K. Moritani, T. Masai, and K. Goto, "DCS—A New Approach to Multi-System Data-Sharing," *Proceedings of the National Computer Conference*, Las Vegas, NV (July 1984), pp. 59–68.
15. P. S. Yu, D. M. Dias, D. W. Cornell, and A. Thomasian, "Performance Comparison of I/O Shipping and Database Call Shipping: Schemes in Multisystem Partitioned Databases," *Performance Evaluation* **10**, No. 1, 15–33 (1989).
16. P. S. Yu, D. M. Dias, J. T. Robinson, B. R. Iyer, and D. W. Cornell, "On Coupling Multi-Systems Through Data Sharing," *Proceedings of the IEEE* **75**, No. 5, 573–587 (1987).
17. D. Attanasio, M. Butrico, J. Peterson, C. Polyzois, and S. Smith, *Design and Implementation of a Recoverable Virtual Shared Disk*, IBM Research Report, in preparation.
18. F. Carino and P. Kostamaa, "Exegesis of DBC/1012 and P90: Industrial Supercomputer Database Machines," *Proceedings of the 4th International Parallel Architecture and Languages (Europe) Conference* (1992).
19. R. Horst, "Massively Parallel Systems You Can Trust," *Proceedings of Spring COMPCON 94* (February 1994).
20. Cray Research, Inc., *Cray T3D Technical Summary* (September 1993).
21. Convex Computer Corp., *Convex Exemplar Scalable Parallel Processing System: System Overview* (1994).
22. W. Groscup, "The Intel Paragon XP/S Supercomputer," *Proceedings of the 5th ECMWF Workshop on the Use of Parallel Processors in Meteorology* (November 1992).
23. D. Lenoski, J. Laudon, K. Gharachorloo, W. D. Weber, A. Gupta, J. Hennesey, and M. Lam, "The Stanford DASH Multiprocessor," *IEEE Computer* **25**, No. 3, 63–79 (March 1992).
24. A. Agarwal, J. Kubiatowicz, D. Kranz, B-H. Lim, D. Yeung, G. D'Souza, and M. Parkin, "Sparcle: An Evolutionary Processor Design for Large-scale Multiprocessors," *Micro* **13**, No. 3, 48–61 (June 1993).
25. B. S. Ang, Arvind, and D. Chiou, *StarT the Next Generation: Integrating Global Caches and Dataflow Architectures*, MIT CGS Memo 354 (February 1994).
26. W. J. Dally, J. A. S. Fiske, J. S. Keen, R. A. Lethin, M. D. Noakes, P. R. Nuth, R. E. Davison, and G. A. Flyer, "The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms," *IEEE Micro* **12**, No. 2, 23–39 (April 1992).
27. S. K. Reinhart, J. R. Larus, and D. Wood, "Tempest and Typhoon: User-level Shared Memory," *Proceedings of the 21st Annual International Symposium on Computer Architecture* (April 1994), 325–336.
28. T. E. Anderson, D. E. Culler, and D. Patterson, "A Case for NOW (Network of Workstations)," *IEEE Micro* **15**, No. 1, 54–64 (February 1995).
29. P. Keleher, S. Dwarkadas, A. Cox, and W. Zwaenepol, *Distributed Shared Memory on Standard Workstations and Operating Systems*, Technical Report RICE COMP TR93-206, Rice University (June 1993).
30. V. Bala et al., "IBM External User Interface for Scalable Parallel Systems," *Parallel Computing* **20**, No. 4, 445–462 (April 1994).
31. MPI Forum, *Document for a Message Passing Interface*, Technical Report CS-93-214, University of Tennessee (November 1993).
32. V. S. Sundcram, G. S. Geist, J. Dongarra, and R. Manchek, "PVM Concurrent Computing System: Evolution, Experiences and Trends," *Parallel Computing* **20**, No. 4, 531–545 (April 1994).
33. S. White and S. Dhawan, "POWER2: Next Generation of the RISC System/6000 Family," *PowerPC and POWER2: Technical Aspects of the New IBM RISC System/6000*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1994).
34. C. B. Stunkel, D. G. Shea, B. Abali, M. G. Atkins, C. A.

Bender, D. G. Grice, P. Hochschild, D. J. Joseph, B. J. Nathanson, R. A. Swetz, R. F. Stucke, M. Tsao, and P. R. Varker, "The SP2 High-Performance Switch," *IBM Systems Journal* **34**, No. 2, 185–204 (1995, this issue).
35. T. Feng, "A Survey of Interconnection Networks," *IEEE Computer* **14**, No. 12, 12–27 (December 1981).
36. M. Snir, P. Hochschild, D. D. Frye, and K. J. Gildea, "The Communication Software and Parallel Environment of the IBM SP2," *IBM Systems Journal* **34**, No. 2, 205–221 (1995, this issue).
37. P. F. Corbett, D. G. Feitelson, J.-P. Prost, G. S. Almasi, S. J. Baylor, A. S. Bolmarcich, Y. Hsu, J. Satran, M. Snir, R. Colao, B. D. Herr, J. Kavaky, T. R. Morgan, and A. Zlotek, "Parallel File Systems for the IBM SP Computers," *IBM Systems Journal* **34**, No. 2, 222–248 (1995, this issue).
38. S. Dewey and J. Banas, "LoadLeveler: A Solution for Job Management in the UNIX Environment," *AIXtra* (May, June 1994).
39. B. Beckenbach, "User Experience with Queuing Systems for UNIX Batch Processing," *Proceeding of SHARE Europe Spring Conference* (April 1994), pp. 15–22.
40. *The Benchmark Handbook for Database and Transaction Processing Systems, 2nd Edition*, J. Gray, Editor, Morgan Kaufmann Publishers, Inc. (1993).
41. J. Dongarra, *University of Tennessee Linpack Performance Report*, University of Tennessee (September 1994).
42. D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, V. Ver.katakrishnan, and S. Weeratunga, *The NAS Parallel Benchmarks*, Technical Report RNR-94-007, NASA Ames Research Center (March 1994).
43. D. Bailey, E. Barszcz, L. Dagum, and H. Simon, *NAS Parallel Benchmark Results*, Technical Report RNR-94-001, NASA Ames Research Center (October 1994), and supplements.
44. R. C. Agarwal, B. Alpern, L. Carter, F. G. Gustavson, D. J. Klepacki, R. Lawrence, and M. Zubair, "High-Performance Parallel Implementations of the NAS Kernel Benchmarks on the IBM SP2," *IBM Systems Journal* **34**, No. 2, 263–272 (1995, this issue).
45. V. K. Naik, "A Scalable Implementation of the NAS Parallel Benchmark BT on Distributed Memory Systems," *IBM Systems Journal* **34**, No. 2, 273–291 (1995, this issue).

Accepted for publication February 21, 1995.

Tilak Agerwala *IBM POWER Parallel Division, Highly Parallel Supercomputing Systems Laboratory, 522 South Road, Poughkeepsie, New York 12601-5400 (electronic mail: tilak@vnet.ibm.com).* Dr. Agerwala is the Director of Parallel Architecture and System Design in the POWER Parallel Division and is responsible for system architecture, technology strategy, and performance evaluation. His area of expertise is computer architecture with a focus on high-performance computing, superscalar designs, and parallel processing. Dr. Agerwala received his Ph.D. in electrical engineering from The Johns Hopkins University, Baltimore, Maryland, in 1975 and his B. Tech. degree from the Indian Institute of Technology, Kanpur, India, in 1971. From 1975 to 1979 he was an assistant professor at the University of Texas at Austin. He joined IBM in 1979 as a research staff member at the Thomas J. Watson Research Center. From 1984 to 1987 he established and managed broad research programs in parallel processing, supercomputing, and artificial intelligence. He was appointed to the Corporate Technical Committee in 1987 and was Director of Future Systems Technology in the RISC System/6000 division prior to assuming his present position. Dr. Agerwala has published several papers and given numerous invited technical presentations worldwide in his area of expertise. He is a member of the IBM Academy of Technology and was elected to its Technology Council in 1990. He has served on many professional panels and advisory committees. Dr. Agerwala is a member of ACM and a Fellow of the Institute of Electrical and Electronics Engineers.

Joanne L. Martin *IBM POWER Parallel Division, Highly Parallel Supercomputing Systems Laboratory, 522 South Road, Poughkeepsie, New York 12601-5400 (electronic mail: jmartin@vnet.ibm.com).* Dr. Martin joined IBM as a research staff member at the Thomas J. Watson Research Center in November, 1984. She received her Ph.D. in mathematics from The Johns Hopkins University in 1981 and conducted research in performance evaluation for supercomputers at Los Alamos Partnerships and Performance Studies. Her area of expertise is in the evaluation of performance of supercomputing systems, an area in which she has published a number of papers and edited books and journals. In May 1991, Dr. Martin was appointed manager of her present department, which is responsible for performance and modeling for the POWER Parallel Division, and in January 1993, she was appointed a Senior Technical Staff Member in recognition of her work in the area of analysis of high-performance systems. She has maintained a connection to the external scientific community, editing a journal that is published by The MIT Press, serving as the general chair of Supercomputing 90, and serving as an advisor to the Department of Energy and the National Science Foundation (where she served as the chair of the program advisory committee for advanced scientific computing in 1991). Additionally, she was named to *Who's Who in Science and Engineering* for 1992–1993.

Jamshed H. Mirza *IBM POWER Parallel Division, Highly Parallel Supercomputing Systems Laboratory, 522 South Road, Poughkeepsie, New York 12601-5400 (electronic mail: mirza@vnet.ibm.com).* Dr. Mirza is currently a system architect in the POWER Parallel Division and works on various aspects of the architecture definition and system design for future SP2 systems. Since joining IBM in 1982, he has held several technical and management positions in the design and development of system products for the technical computing market. Prior to that, he was an assistant professor of computer science at the Polytechnic Institute of New York, Brooklyn. Dr. Mirza has a B. Tech. degree from the Indian Institute of Technology, Kharagpur, India, and M.S. and Ph.D. degrees in computer science from the Polytechnic Institute of New York, Brooklyn.

David C. Sadler *IBM POWER Parallel Division, Highly Parallel Supercomputing Systems Laboratory, 522 South Road, Poughkeepsie, New York 12601-5400 (electronic mail: dsadler@vnet.ibm.com).* Mr. Sadler is currently a member of the Parallel Architecture and System Design Department where he has been working on the definition of languages, architecture, and system design for scalable parallel RISC-based systems. Prior to joining the Parallel Architecture and System Design Department he managed the initial software development effort for the SP1 for the communication subsystem and the parallel programming environment. Mr. Sadler received his B.S. in mathematics from the Pennsylvania State University in 1967. He joined IBM in 1967 and has held various technical

and management positions within IBM. He managed the development activities for IBM for 4700 COBOL, 4700 C, 4700 Assembler, IBM Clustered FORTRAN, Enhanced Clustered FORTRAN, 4700 Host Support programs, network management tools for distributed systems, and 4700 microcode development. He has held technical and management positions within IBM's supercomputer and parallel programming efforts since 1986.

Daniel M. Dias *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.* Dr. Dias received the B. Tech. degree from the Indian Institute of Technology, Bombay, India, and the M.S. and Ph.D. degrees from Rice University, Houston, Texas, all in electrical engineering. He currently manages the Parallel Commercial Systems group at the IBM Thomas J. Watson Research Center, which includes exploratory systems architecture, design, and analysis, in the areas listed below, with a focus on reducing these ideas to working prototypes and products. His current research includes highly available clustered systems, parallel and distributed systems, video server architectures, parallel transaction and query processing, reliable disk architectures, interconnection networks, and performance analysis. Dr. Dias has published more than 100 papers in refereed journals and conferences. He has won two best paper awards, IBM Outstanding Innovation and Technical Achievement Awards, five Invention Achievement Awards, and Research Division awards. He holds eight U.S. patents, with four additional patents pending.

Marc Snir *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (electronic mail: snir@watson.ibm.com).* Dr. Snir is senior manager at the IBM Thomas J. Watson Research Center, where he leads research on scalable parallel software and on scalable parallel architectures. He recently led the Vulcan software effort and the initial design and prototyping of parallel software for the IBM SP1. He received a Ph.D. in mathematics from the Hebrew University of Jerusalem in 1979. He worked at New York University (NYU) on the NYU Ultracomputer project from 1980–1982 and worked at the Hebrew University of Jerusalem from 1982–1986. He has published on computational complexity, parallel algorithms, parallel architectures, interconnection networks, and parallel programming environments. He recently contributed to High Performance FORTRAN and to the Message-Passing Interface. Dr. Snir is a member of the IBM Academy of Technology, a senior member of IEEE, and a member of ACM and SIAM.