Parallel Computers: Current Systems and Capabilities

Oliver A. McBryan

CU-CS-635-92 December 1992

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **DEC 1992**	2. REPORT TYPE	3. DATES COVERED **00-12-1992 to 00-12-1992**
4. TITLE AND SUBTITLE **Parallel Computers: Current Systems and Capabilities**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of Colorado at Boulder,Department of Computer Science,Boulder,CO,80309-0430**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**		**52**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

Parallel Computers: Current Systems and Capabilities[*]

Oliver A. McBryan
Dept Of Computer Science
University of Colorado
Boulder, CO 80309.

Abstract

The needs of scientific and engineering grand challenge computations are driving the design of current high performance computing systems. We review the background for this development and the essential role played by massively parallel computers (section 1).

We describe the various major classifications of massively parallel systems and describe the advantages of each approach (section 2). Finally we survey in detail most of the recent advanced systems, discussing both their hardware and software (sections 3-6).

[*]Research supported in part by AFOSR grant AFOSR-89-0422 and by NSF Grand Challenges Applications Grroup grant ASC-9217394.

TABLE OF CONTENTS

TABLE OF CONTENTS

Parallel Computers: Current Systems and Capabilities*

Oliver A. McBryan
Dept Of Computer Science
University of Colorado
Boulder, CO 80309.

1. Parallel Computing: Origins and Applicability

An analysis of current computer usage in government, industry and academic environments indicates several distinct patterns of use:

1. Individual use of personal computers and workstations.
2. Shared networks of personal computers and workstations.
3. Supercomputers used in commercial environments.
4. Supercomputers used in scientific and engineering applications.

All of these patterns of use are of great economic importance, but they are quite distinct in terms of the needs for both hardware and software features.

Individual computing (including local networks) has the highest requirement for software quality and hardware reliability, and low requirements for computational and I/O performance. This area also represents by far the major installed base of systems, whether counted in numbers of units or in dollar value, and consequently can deliver economies of scale on software and hardware development that are not available elsewhere.

Shared networks of PC's and workstations are currently widely distributed in office and academic settings. Typically the networks are used to provide access to shared resources such as printers and file servers, and to allow limited exchanges between users (usually of mail and files). For the most part shared network computing use is essentially identical to the previous individual computing case. However, there is an enormous potential for utilizing such networks as coordinated computing clusters, and it is this pattern of use that we single out here. We will refer to it as networked clusters in the sequel. Networked clusters provide opportunities to benefit from economies of scale not usually available to supercomputer vendors.

* Research supported in part by AFOSR grant AFOSR-89-0422 and by NSF Grand Challenges Applications Grroup grant ASC-9217394.

Commercial supercomputing is characterized by the need for extreme reliability and fault tolerance, an intermediate quality of software and (by current standards) very modest performance requirements, with the exception that in certain specialized areas, near real-time computing may be involved (e.g. airplane reservations, credit card checks). Note that in discussing commercial computing, we omit from that designation all scientific and engineering commercial applications.

Scientific and engineering supercomputing is characterized by the need for high performance - in Mflops, in Gbytes of memory and in I/O bandwidth. This is the area that has been labeled High Performance Computing (HPC) in recent years. In almost an exact inverse relationship, one finds the lowest quality standards required for scientific software, and relatively minimal attention to fault tolerance and reliability of systems. The driving force in the scientific and engineering supercomputing (SES) area that differentiates it from the other areas, is the nearly unlimited expansion in computer performance required to solve even the simplest problems, as we now illustrate with an example.

<u>**HPC Example:**</u>

> In order to simulate the simplest turbulent flow in a fluid, scientists would like to model the fluid with a 1000^3 grid of points, with perhaps 20 fluid variables (8 byte real numbers) stored at each point. Such a computation requires close to 100,000 *time steps*, and each time step involves performing 100 or more numeric operations per grid point. Thus a complete computation requires at least 10^{16} floating point operations and needs 160 Gbytes of main memory to store the data. Even on a gigaflops (Gflops) computer (delivered performance), such a computation will require 10^7 seconds, or over 100 days of dedicated runtime - and we have ignored completely the cost of I/O operations. Only a teraflops (Tflops) computer offers the hope of running such a computation to conclusion within an 8-hour day. The I/O needs may be estimated by imagining that a snapshot consisting of 10 variables per grid point needs to be written to mass storage every 100 time steps. Each snapshot will consume 80 Gbytes for a total of 80 TeraBytes (Tbytes) by the end of the run. If the data is to be written during the 8-hour run, the I/O rate will have to be 80 Tbytes in 8 hours or about 3 Gbytes/sec.

Commercial applications by contrast are confined more to the finite scale of human activities, with the largest applications such as bank transactions or IRS return processing required to deal with numbers of items roughly proportional to the U.S population, and frequently significantly smaller. The real-time access requirement to databases tends to put a premium on I/O bandwidth in relation to Mips (Mips is usually more relevant than Mflops for these non-numeric applications), at least as compared to scientific applications.

Because of these different requirements, and the near limitless demands of SES computing, the latter is currently driving almost all of the hardware developments in high performance computing. Indeed while I/O may be relatively less important than Mips/Mflops for scientific applications, the absolute I/O performance requirements are in fact way larger than for commercial computing. Therefore we find that all of the major computing performance breakthroughs are occurring in the scientific arena. For this reason we will restrict our discussion from here on exclusively to the domain of scientific and engineering supercomputing.

Unfortunately basic limitations of physics, such as energy dissipation and the finite speed of light, appear to prevent the development of processors capable of delivering the computational and I/O rates discussed above - at least using current technologies. Massively parallel computing is believed to be the only effective route to achieving the performance levels (measured in Tflops, and Gbytes/second) required to solve the grand challenge problems of science and engineering. For this reason we will concentrate on massively parallel computer design for the rest of this discussion.

We describe the various major classifications of massively parallel systems and describe the advantages of each approach in section 2. Then we survey in detail most of the recent advanced systems in sections 3-6, discussing both their hardware and software. The systems are categorized as SIMD (Section 3), MIMD Message Passing (section 4) and Shared Memory (Section 5). Details of performance measurements of many of these systems are in references [2-11].

2. Design of Parallel Systems

In this section we will provide a general discussion of design issues for massively parallel systems, with some discussion of the historical developments and systems that have resulted in the current massively parallel supercomputer architectures. We will present a detailed look at a selection of typical systems in sections 3-6.

2.1. Scalable Systems and Algorithms

In order that parallel supercomputers can make a serious attack on the SES Grand Challenge applications, it is essential that these systems have the property of scalability. Scalability essentially means that performance increases linearly with the number of processors P

$$Perf(P) = O(P).$$

In practice if performance drops below linearity by not more than a logarithmic factor, it is also considered to be scalable:

$$Perf(P) = O(P / (\ln P)^c).$$

Scalability in parallel computer architecture is a critical issue for massively parallel computer designers. This is a major departure from conventional supercomputer design where at most eight or sixteen processors are involved. In such systems the design emphasis is entirely on node performance and I/O to disk.

A second critical aspect of SES applications is the need to design scalable algorithms. An algorithm is scalable on a scalable architecture if performance scales linearly with the number of processors, provided that the problem size is scaled to need that many processors. The need for a particular minimum number of processors is usually driven by the memory needs of the application. Again a reduction from linearity by a logarithmic factor is still considered scalable.

The design of scalable machines and the search for scalable algorithms for SES applications on such machines are the two dominant forces controlling HPC activities today.

2.2. Classification of Parallel Computers

Parallel computers may be broadly categorized in two types - SIMD or MIMD [1]. SIMD and MIMD are acronyms for Single Instruction stream Multiple Data stream, and Multiple Instruction stream - Multiple Data stream, respectively. In SIMD computers, every processor executes the same instruction at every cycle, whereas in a MIMD machine, each processor executes instructions independently of the others. The vector unit of a CRAY computer is an example of SIMD parallelism - the same operation must be performed on all components of a vector. Most of the interesting new parallel computers are of MIMD type which greatly increases the range of computations in which parallelism may be effectively exploited using these machines. However, this occurs at the expense of programming ease - MIMD computers are much more difficult to program than SIMD machines. Many current designs incorporate both MIMD and SIMD aspects - typically each node of an MIMD system is itself a vector processor. A frequent programming paradigm for MIMD machines is the SPMD model, an acronym for Single Program - Multiple Data. In this model the same program text is run on all processors of a system, but the execution may follow different paths through the program on different processors.

Another easy categorization is between machines with global or local memories. In local memory machines, communication between processors is entirely handled by a communication network, whereas in global memory machines a single high-speed memory is accessible to all processors. Beyond this, it becomes difficult to categorize parallel machines. There is an enormous variety in the current designs, particularly in the interconnection networks.

While many interesting parallel machines involve only a few processors, we will restrict consideration to those machines which have moderate to large numbers of processors, since they represent the path to true massive parallelism. Furthermore, we will emphasize those machines which have a distributed memory architecture, including virtual memory systems, because at this point such systems are the only ones that appear to be truly scalable. Important classes of machines such as the CRAY Y-MP, CRAY-2, CRAY C-90 and the Japanese supercomputers are briefly discussed, but are not the focus of our remarks because such systems do not appear to offer true scalability using any current approach.

2.3. Approaches to Multicomputing

There are over 100 recent or current parallel computer projects worldwide. Table 1 lists a selection of such projects, a mix of university and industrial enterprises. This is just a sample of the diverse projects but covers a wide range of different architectures chosen more or less at random. While some of these projects are unlikely to lead to practical machines, a substantial number have already, or will, lead to useful prototypes. Many commercial parallel computers are included in the table and these have been or are already in production (e.g., Connection Machine CM-2, CM-5, Denelcor HEP-1, Evans and Sutherland ES-1, FPS T-Series, ICL DAP, Intel iPSC2, Intel Paragon, Alliant FX/8, Alliant FX/2800, Alliant CAMPUS/800, Meiko CS2, Myrias SPS-3, NCUBE-2, Parsytec GC(T805), SUPRENUM-1, Symult 2010) and more are under development. Some of these products have been commercial failures (e.g. Denelcor HEP, ETA-10, ES-1, FPS T-Series, Symult 2010), yet they have provided important insights into parallelism. One should also remember that the latest CRAY computers, (e.g. CRAY-2, CRAY Y-

MP and CRAY Y-MP/C90) involve multiple processors, and other vector computer manufacturers such as NEC, Fujitsu and Hitachi have similar strategies.

Beyond the simple classification into SIMD or MIMD computers we recognize a vast array of different approaches to the task of building a parallel architecture. We will now look at the reasons for this broad range by discussing some of the possibilities encountered for both node and communication facilities.

CalTech hypercube	Intel iPSC2 hypercube
NCUBE2 hypercube	Intel iPSC/860 hypercube
ICL DAP	Goodyear MPP
Connection Machine CM-2	Masspar MP1000
FPS T-Series	Symult 2010
Intel Touchstone	Intel Paragon
Meiko MK860	Parsytec
SUPRENUM-1	IBM GF-11 and TF1
CRAY Y-MP and CRAY-2	ETA-10
Denelcor HEP-1	NYU/IBM Ultra-computer/RP3
BBN Butterfly	Cedar Project
Myrias SPS-3	Kendall Square KSR1
Illiac IV	Wisconsin Database Machine
Connection Machine CM-5	Fujitsu AP-1000
Intel DELTA	Intel Paragon
Flex 1	Alliant FX/8
Alliant FX/2800	Alliant Campus/800
Sequent Balance	Encore Multimax
CCI Navier-Stokes Machine	TERA
Multiflow	Data-flow Machines
Intel iWarp	Evans and Sutherland ES-1

Table 1. Some Parallel Computer Projects

2.4. Node design

By *node* we mean the individual computational processor, along with its associated communications hardware, and local memory if available. Node design tends to be far less variable than other aspects of parallel computers. The main reason for this is that most architects have relied on off-the-shelf products for the node - standard microprocessors, floating point accelerators and memory chips. The advantage is that startup time for a project may be substantially reduced. Additionally, usually there is a substantial body of low-level software available for such processors - such as compilers, assemblers and debuggers. Thus we find that an enormous number of the recent or current parallel computer products are based on one or more of the Intel 80386, Motorola 68020, INMOS T800 transputer, the Weitek floating point accelerators, the Intel i860, and the Sparc processors. Typically one of these microprocessors will be combined on a board with a floating point coprocessor (e.g. 80387 or 68881), possibly a Weitek processor and several megabytes of memory. The most recent processors (e.g. i860) tend to have built-in floating point processing, and sometimes a graphics processing capability. Despite these general comments, it should be mentioned that some manufacturers have developed custom processors specifically for parallel computers. In the list above we would point to the DAP, NCUBE, HEP-1, CM-2, ES-1, iWarp, Navier-Stokes and KSR1 machines as examples.

Memory consumes substantial space, and current systems have in the range of 1 to 32 Mbytes per node, although up to 128 Mbytes has been announced for some systems (e.g. Intel Paragon). Most systems support several levels of memory on a node: frequently main memory, cache memory and registers. Effective management of cach memory is often critical for good node performance.

Very recently there has been a trend towards complexifying the physical node concept in order to increase packaging density and reduce cost. Early examples were the CM-1 and CM-2 machines which have 16 processors on a single chip. In the CM-2 this is further confused by the fact that a single Weitek vector processor is shared by the 32 processors on each pair of adjacent chips. The recent CM-2 and CM-200 slicewise operating systems in fact recognize this in that they present a view of the system as consisting of 2048 vector processing nodes, essentially ignoring the view of the system as composed of 65,536 bit-serial processors. The new CM-5 node consists of a Sparc processor and 4 vector processors, which may only be treated as a unit. Intel has announced that sometime in 1993 they will switch from the current Paragon node design of two i860 processors (one computational and one communicational), to a highly integrated chip containing 4 i860 processors - and in fact these processors share memory.

The increasing complexity of the node concept is a major challenge for any simulation system for massively parallel computers. It is also a major challenge for users and software designers who would prefer not know about these details. One resolution of this dichotomy is to separate the concepts of logical and physical node. A logical node is a unit that supports a single thread of computation and memory access at any time. A physical node is a level in the physical hierarchy constituting a massively parallel system, and is the smallest hierarchical level that contains at least one CPU.

2.5. Communication Features

The range of interprocessor communication facilities is what really characterizes the differences in architecture among various parallel machines. While we have previously distinguished the shared memory and distributed memory classes, one should observe that this distinction should not be taken too seriously. A distributed memory computer can certainly simulate a global shared memory and vice versa.

Communication pathways are typically built either from direct point-to-point connections, or from busses. Busses have the advantage that many processors may be serviced by one communication path, but have the disadvantage of slower bandwidth performance as the number of processors increases. With point-to-point connections, processors that are directly connected will have very efficient communication, but indirectly connected processors will likely incur substantial extra overheads including increased latency as well as lower bandwidth. Over the last seven years, interconnection performance has improved almost as fast as processor performance. For example, the Intel systems have moved from performance in the range of hundreds of Kbytes/sec on the iPSC1 to 200 Mbytes/sec on the Paragon. Communication latency has also improved substantially although to a lesser extent over this period, from about 5000 microsecs on the iPSC1 to 60 microseconds on the Intel DELTA for example.

The most popular interconnection strategies involve simple symmetric arrangements including rings, meshes, hypercubes, trees and complete connections or crossbars. The prevalence of hypercube designs is explained by the fact that the architecture supplies substantial parallel bandwidth for many standard algorithms, for example the Fast Fourier Transform, while at the same time incurring only relatively modest fan-in and fan-out of connections which grow in number only logarithmically with the processors. On the other hand, hypercube wiring complexity grows with the number of processors and the likelihood increases that most wires are unused most of the time. Table 2 compares several simple topologies as a function of processor number P from the point of view of amount of wiring (difficulty of building), connectivity (ease of programming) and maximal path (efficiency of long-range communication).

Network	Wires	Connectivity	Max Path
Cross Bar	P^2	P	1
1D Grid	P	2	P
2D Grid	2P	4	$2\sqrt{P}$
Binary Tree	P	1-3	$2\log_2 P$
Hypercube	$\frac{1}{2}P\log_2 P$	$\log_2 P$	$\log_2 P$

Table 2. Properties of Interconnection Networks

While cross bar switches are extremely difficult to build for large numbers of processors, they have tremendous flexibility in terms of efficiency and ease of use. It is conceivable that a technological breakthrough such as optical switching might allow cross bars to be built that would connect thousands of processors. For the time being, crossbars are restricted to small systems of at most 64 processors, or to providing interconnects among the processors of sub-clusters within larger machines (e.g. the ES-1 and Alliant FX/2800).

Bus based connection networks are attractive for moderate numbers of processors, for example 16 to 32. Beyond this point bandwidth begins to suffer intolerably. Architectures based on busses therefore tend to be hierarchical beyond that number of processors. As an example, the SUPRENUM-1 computer uses a fast local bus to connect within a cluster of 16 processors. Clusters are arranged in a rectangular grid and connected by row and column busses, which has the added attraction of providing redundancy and double bandwidth. The Myrias SPS-3 is similar, utilizing three levels of bus: 4 processors connected on a single board by a bus, cages of 16 boards connected by a pair of backplane busses, and finally cages connected by third-level busses. The KSR1 connects 32 processors in a ring as a basic cluster, with rings of clusters used to scale the system past 1,000 nodes.

New configurations of processors continue to be proposed. One recent trend is the move towards "worm-hole" routing in distributed systems. The basic idea here is to allow virtual circuits to be established between remote processors, without the necessity of interrupting any intermediate nodes. While there may be a small overhead for circuit creation, subsequently all data traverses the circuit without overheads such as multiple startup costs at intermediate nodes. Once a circuit is established, communication proceeds essentially in pipelined fashion. Frequently it suffices to create logical rather than physical connections. These allow messages to proceed on virtual worm-hole channels, but with the possibility that physically the channels are multiplexed. This is particularly convenient as a means for preventing dead-locks and blocking of small messages by large ones. The resulting communication performance tends to be essentially independent of distance.

Worm-hole routing is utilized in the CM-2, the iPSC2, iPSC/860, the Intel Paragon and the Intel iWarp among others. The CM-2 and the Symult 2020 were the first systems to emphasize wormhole routing. In the case of Symult, the designers were so confident of the advantages of wormhole routing that they abandoned a hypercube architecture from their first generation in favor of a simple two-dimensional rectangular grid. The Intel iPSC2 hypercube has similarly given way recently to a rectangular-grid based Intel Paragon, similar in spirit to the Symult.

2.6. Parallel Software

Software for currently available parallel computers is extremely limited. In all cases manufacturers provide Fortran and C compilers, which are frequently just a single-node processor compiler. These compilers usually have no concept of parallelism or of communication capability. Typical examples are the systems supplied by Intel, Meiko and NCUBE. In these systems, all communication and process control is initiated explicitly by the user, resulting in substantial code modification as well as a loss of portability of software.Typically libraries of low-level communication primitives are supplied with these systems to allow the user to initiate communication operations. The resulting software is best described as "programming in communication assembly language".

A few manufacturers have gone beyond this by providing language extensions that capture aspects of the parallel hardware. Thinking Machines provides a parallel Fortran for their Connection Machine CM-2 and CM-5 computers. The compiler supports the Fortran 90 array extensions to Fortran 77, and the convention is that objects used as arrays are understood to be distributed across the parallel processors. Communication among processors is supported by the F90 shift operations, as well as the various reduction operators such as vector sum. While the Connection Machine programming environment is remarkably elegant and user-friendly, one should point out that the task is much simplified by the SIMD nature of the CM-2 hardware onto which array operations map extremely well.

Myrias Corporation (SPS-2, SPS-3) and Evans and Sutherland (ES-1, no longer in production) both support a virtual address space across processors. If a processor attempts to access a memory location not in its physical memory, then a page fault occurs and the appropriate memory page is fetched from the processor which has it. Myrias in particular have implemented a sophisticated mechanism for load balancing and rapid access to memory. The system attempts to localize page table information and to provide access to it in a distributed fashion. The Myrias system is the first to provide virtual shared memory on a distributed memory architecture. The ES-1 was actually a cross-bar based shared memory computer, and here again virtual memory was provided to make the memory system more transparent.

The most recent system of this type is the KSR1 which combines a two-level hierarchical memory based on rings, with real supercomputer performance (1088 nodes, 40 Mflops/node peak). The system implements a global 40-bit address space using hardware assistance for virtual shared memory operations. The KSR-1 system supports interprocessor communication at rates of 34 Mbytes/sec with a latency of only 6 microsecs.

SUPRENUM supports extensions to Fortran for task control, and to assist in communication operations. In addition SUPRENUM is unique in providing a sophisticated high-level library interface to the communication system. The library supports a range of grid-oriented operations that largely shield a numerical user from dealing with the communication system directly. In addition to providing powerful programming tools, such systems deliver the possibility of substantial program portability across architectures that support the common set of primitives.

One should also note the tendency to support virtual processes. This is an important aid to software development as it allows an application to simulate a larger number of processors than are physically present. Virtual processing actually can increase system throughput by allowing nodes to remain computationally active while a process is suspended waiting for memory or messages. Virtual processing is supported by the majority of systems in one form or another. Examples include iPSC, SUPRENUM, Myrias and CM-2.

Another recent trend is towards providing a complete UNIX capability on every node. For example the Alliant CAMPUS/800, the KSR1 and the Intel Paragon provide this feature. In some cases (e.g. Paragon) this is accomplished by supporting a microkernel on computational nodes which relies on services provided by special server and I/O nodes. In addition to UNIX, other standard software products are being offered by a wide range of vendors: for example NFS file systems, TCP/IP networking to nodes, and support for graphics systems such as AVS and Explorer.

3. Single Instruction Multiple Data (SIMD) Machines

3.1. Thinking Machines CM-2

The Connection Machine CM-1 designed by Thinking Machines, Inc., has 65,536 1-bit processors. Processors are arranged with 16 on a chip along with a communications router and associated local memory. While initially designed primarily for artificial intelligence work, this machine proved to have even greater potential application to scientific computing applications. The more recent CM-2 computer adds 2,048 Weitek floating point processors and 2 Gbytes of memory, to provide a powerful computer for numerical as well as symbolic computing. Peak computational rate at 7 MHz (the current clock rate) is theoretically about 24 Gflops (32-bit). A later version, the CM-200, differs by having a 10 MHz clock and peak rates of 32 Gflops.

The CM1 and CM-2 computers are SIMD machines. Logic is supported by allowing individual processors to skip the execution of any instruction, based on the setting of a flag in their local memory. The CM communications facilities are based on a hypercube network, with a total communication bandwidth of order 3 Gbytes/sec. Communication is by worm-hole type routing. The system supports I/O to disks at up to 320 Mbyte/sec, and to graphics frame buffers at 40 Mbyte/sec.

Connection Machine software consists of parallel versions of Fortran, C and Lisp. In each case it is possible to declare parallel variables, which are automatically allocated on the hypercube. Programs execute on a front end machine, but when instructions are encountered involving parallel variables, they are executed as parallel instructions on the hypercube. The system supports the concept of virtual processors. A user can specify that he would like to compute with a million (or more) virtual processors, and such processors are then similar to physical processors in all respects except speed and memory size. A typical use is to assign one virtual processor per grid point in a discretization application. This provides a very convenient programming model. Parallel global memory reference is supported using both regular multi-dimensional grid notations (NEWS communication) and random access (hypercube) modes.

The Connection Machine is one of the few examples where a program from a serial machine (e.g. work-station) or from a CRAY may be moved to a parallel machine and run essentially without change. The CM-2 Fortran is Fortran-77 with the addition of the array extensions of Fortran 90. All array data types and operators are implemented as parallel objects or operators on the CM-2. In the case of Fortran-77 programs, a vectorizer is available that produces Fortran 90 as output. Because of the SIMD architecture no synchronization instructions are required.

It is extremely rare to approach the 24 Gflops peak rate of the CM-2. In practice one attains about 10% to 20% of that rate. In part this is because in addition to normal hypercube communication (e.g. to nearest neighbors in a grid) there is also extra communication required for every floating point operation. Since the floating point processors are off-chip, each of the 32 bit-serial processors that share a Weitek, must send its data to the Weitek for processing. Furthermore since the data arrives bit-serially, it needs to be "transposed" so as to be presented as floating point data to the Weitek. These two steps between the processors and the Weitek units account for much of the performance loss. Only in situations where numbers can be deposited in the 64 Weitek registers (shared by 32 processors), and then computed on for a substantial time without leaving the Weitek, can the theoretical speed be approached. For example, parallel polynomial evaluation proceeds at up to 20 Gflops - which ensures that transcendental functions are extremely fast on the system. Standard numerical algorithms such as relaxation or conjugate gradient iterations perform at from 2 to 4 Gflops, which is also typical of performance on regular-grid evolution problems. Despite the low efficiency, these performance numbers exceed those of a CRAY Y-MP/8, and with a price tag around $6M, the 65,536 processor system is cost-effective for classes of problems that exhibit massive SIMD data parallelism.

The CM-2 computer has established a real commercial success in the following sense. Most highly parallel machines sold to date have been purchased for experimental research at universities or major research laboratories, not for production supercomputing use. A substantial number of CM-2 machines have however been sold for production uses, including database management (Dow Jones), seismic processing (Mobil), and aircraft design (United Technologies and Lockheed) to mention a few examples. In comparing the CM-2 computer to other systems one should keep in mind that the CM-2 is (in late 1992) a four year old system.

3.2. MasPar MP-1 and MP-2

The MasPar MP-1 series (1101, 1104, .. , 1116) are SIMD array machines featuring both a grid connectivity and a routing system for messages. The individual processors are 4-bit, with floating point performed in emulation mode. The systems range in size from 1K to 16K processors. The MP-1 is intermediate in design between the DAP and the CM-2. The system is priced substantially lower than a CM-2, but this is likely an advantage only for those applications not requiring floating point. Because of its intermediate position between the DAP and the CM-2, it is not clear how effective MasPar can be in capturing market share.

MasPar is a technology spin-off of Digital Equipment Corporation (DEC). A similar 16K system was developed in-house at DEC, and several key developers subsequently left DEC to found MasPar. The 1K system lists at $170,000, and is being offered in collaboration with DEC for $94,350 under joint research agreements.

MasPar software includes MPF, a parallel Fortran based on Fortran 90. MasPar also supports VAST-2 for converting Fortran 77 to Fortran 90 code by recognizing parallelism within Fortran code. The MasPar programming environment is Motif based and includes the AVS visualization system from AVS

Inc. MasPar also supports a parallel C, MPL, based on Gnu C, and with extensions for parallel data-types, communication routing and function prototyping.

In late 1992 MasPar introduced a second generation SIMD system, the MP-2. The MP-2 is 5 times faster than the MP-1 series with which it is binary compatible. Peak rates are in the range of 2.4 Gflops (64-bit). The new system integrates 32 32-bit CPU's onto a single chip. As with the MP-1, the MP-2 provides connectivity to eight nearest neighbors.

3.3. AMT DAP

The DAP was the first massively parallel single-bit computer, and has been widely used for a range of scientific applications and embedded systems. Its current incarnation as the AMT 510 attached processor, provides the capability to attach a 1024 processor DAP array to any VAX or SUN computer. The 510 is a 32x32 array of processors, arranged as a two-dimensional grid and is implemented in VLSI on 16 chips. Additional busses connect all processors on each row and column and are used for broadcasts and other non-local operations. Up to 1 Mbit of memory may be installed per processor, for a combined total of 128 Mbytes. The computer is SIMD, and can execute at up to 60 Mflops, although boolean operations perform at up to 10 Gips.

4. Multiple Instruction Multiple Data (MIMD) Machines

4.1. Thinking Machines CM-5

The CM-5 is the first MIMD computer from Thinking Machines Corporation (TMC). The system is based on a complex interconnection network involving 3 sub-components - the Data Network, the Control Network and the Diagnostics Network. Individual processing nodes (PM) consist of a SPARC microprocessor with 4 optional vector units and 32 Mbytes of memory (4 Mbit DRAM). Peak computation rate is 128 Mflops (64-bit) although in practice this will be difficult to achieve due to memory bandwidth limitations and because this high rate is supported only for the multiply-add operation, with other operations running at half that rate. Additional control processors (SP) are used to execute system administration tasks and serial user tasks. Currently the SP are top end SUN workstations. The CM-5 is designed as a scalable architecture supporting from 32 to 16,384 processing nodes with a constant data bandwidth per node.

4.1.1. Networks

The Data Network (DN) provides high performance data communications among all system components. Typically the Data Network handles bulk transfers of data where each item has one source and one destination. The DN is a high-bandwidth message-based point-to-point network optimized for common access patterns. It provides guaranteed message delivery, fair conflict arbitration and freedom from deadlocks. Message routing may be overlapped with communication. The DN guarantees 5 Mbytes/sec communication between any pair of points in the system, and this design criterion is scalable to tens of thousands of ports (the DN scales logically to a million ports). If the destination node is within the same group of 16, then messages are transferred at 10 Mbytes/sec, while within the same group of 4 nodes, messages are delivered at 20 Mbytes/sec. There is therefore only a factor of four range between the best and worst communication cases.

The DN network is based on a fat-tree structure. Each tree node has four children, with a Network Interface (NI) at each leaf. While there are fewer nodes higher in a tree, the bandwidth of each is made correspondingly larger, maintaining the guaranteed bandwidth of 5 Mbytes/sec between any pair of nodes.

The Control Network (CN) handles operations that require cooperation of many, or all, processors. The Control Network accelerates cooperative operations, e.g. broadcast and reduction operations, and provides system management operations such as error reporting. The Control Network supports synchronization primitives including "soft" barrier synchronization for loosely synchronous programs, and asynchronous transition signals. The CN also has hardware support for cooperative arithmetic operations, including arithmetic reduction (sum, max, min, and, or, xor), parallel prefix and segmented parallel prefix. The CN has a bandwidth of 20 Mbytes/sec with a latency in the range 2-5 msecs, depending on the partition size. The CN architecture allows for scaling to 1 million network addresses.

The Diagnostic Network finds and isolates errors throughout the system.

Every processor is connected to each of these networks through the CM-5 Network Interface. Both the DN and the CN networks are connected at 20 Mbytes/sec and are directly accessible to user code through memory mapping.

4.1.2. Partitions

The CM-5 may be configured as multiple user partitions. Each partition has dedicated processor and network resources while the system as a whole maintains shared data, control network and I/O resources. Each partition has a separate control processor (SP) running UNIX, while the PE run a microkernel. Each partition may run in either batch or timesharing mode, independently of other partitions.

4.1.3. Processing Nodes

Each processing node includes a Sparc processor, a Network Interface (NI) processor and 4 vector units, all connected on a 64-bit bus. Each vector unit in turn connects to 8 Mbytes of DRAM via a 64-bit path. The vector units dual function as memory controllers.

The current implementation uses a 33 MHz Sparc processor with 64 Kbytes of cache. The processor can execute at 22 Mips and 5 Mflops and also acts as control and scalar processing unit for the vector processors. The processor overlaps scalar control computations with vector processing.

The vector units operate at 16 MHz and allow one memory and one arithmetic operation per cycle. A one-cycle multiply-add operation is supported, and runs at 32 Mflops; multiply and add operations run at 16 Mflops. Divide and square-root operations require 5-8 cycles per operation. There are 64 64-bit registers (or 128 32-bit registers). The register file may be organized in terms of vectors of any length from 1 to 16. Total bandwidth to the vector processors is 512 Mbytes/sec. Vector processors generally require a minimal vector length for full performance. In the case of the CM-5 vector units, this length is only 4, guaranteeing very good performance in most applications.

4.1.4. Control Processors

Each partition of a CM-5 system is under the control of a Control Processor (CP). The CP processor is a standard Sparc workstation with the addition of a CM-5 Network Interface (NI) board.

The CP runs a full UNIX system. The CP and processing nodes are time-sliced together. All processing nodes of a partition work on the same user task at the same time. The processing nodes run a simpler microkernel.

The control network is used to relay interrupts from nodes, and to broadcast interrupts to nodes. To switch users, the control processor interrupts the processing nodes, each of which then saves its user state. All user state may be restored at a later time, and the same mechanism is used to support checkpointing.

The system provides security between users a) within nodes (by memory address translation) and b) between nodes (by network address translation).

The data network and control network divide up cleanly among partitions. Each partition has its own internal communication; user traffic within one partition cannot impede system or user traffic on other partitions. Partitions are themselves joined by the two networks. I/O transfers are supported between a partition and I/O devices. Interprocess UNIX pipes are supported between partitions. Network address translation in each NI enforces security between partitions.

4.1.5. I/O Capabilities

I/O bandwidth is independent of the number of processing nodes. I/O is accessed via the Data Network so that bandwidth scales with the amount of attached hardware. Standard external hardware and software interfaces and protocols supported include HIPPI, VME, CMIO, FDDI. The CM Data Vault is also supported.

The file system uses UNIX file system commands and data formats and supports the NFS (Network File System) standard. The file system flexibly stripes individual devices of any type. One important feature is that the control and data aspects of I/O operations take separate data paths, allowing full-bandwidth transfers of data among parallel devices and groups of processors.

4.1.6. Reliability

Most system components are off the shelf components as used in workstations. The system supports a fully synchronous clocking design. Validity checks include: 8-bit ECC on 64-bit memory words, parity checking or better on I/O transfers, and use of a CRC on every wire of the internal networks.

The third network, the Diagnostic Network, has access to signals throughout the system. All proprietary components in the system may be fully tested in parallel, with greater than 99% fault coverage. The system also supports full testability of wires connecting such components. Any broken component may be isolated logically and electrically. Backplanes can be electrically isolated under software control and then independently powered down for maintenance.

If a processor fails, the OS reconfigures the network translation map in each NI so that messages sent to the failed node are forwarded to a spare. Each user task is restarted from its most recent checkpoint. The system degrades gracefully in terms of worst case consequences of failures as follows. Loss of a processing node can mean that 1/64th of the nodes in that partition must be mapped out (one can substitute spares). Loss of a control processor affects only one partition, and the nodes can be added to another partition. Loss of a control network chip can result in 1/64th of the nodes being mapped out. Loss of a Data Network chip can result in either 1/64th of the nodes being mapped out, or a loss of bandwidth (< 6%) with no node loss.

4.2. Intel iPSC Series

The Intel Corporation, a major manufacturer of microprocessors and computer chips, produced the first commercial realization of the hypercube design, based largely on the CalTech Cosmic Cube. We will compare the performance and characteristics of the four generations of Intel systems as it is illustrative of the rapid rate of progress in parallel computing hardware.

4.2.1. iPSC1

The first Intel hypercube was developed in 1985. The machine, known as the iPSC/1, came in three models - the d5, d6 and d7. These had 32, 64 and 128 processors respectively. The individual processors were the Intel 80286/80287 with up to 512Kb of memory, and the interconnections were provided by an Ethernet, again using an Intel Ethernet chip. The intermediate host machine, which was both the control processor and the user interface, was an Intel 310 microcomputer running a UNIX system (Xenix). In addition to the Ethernets along cube edges, a global communication channel was provided from the intermediate host machine to the individual processors. This feature was extremely useful for debugging and to a limited extent for control purposes. Besides the UNIX system on the host, software for the system consisted of a node-resident kernel providing for process creation and debugging along with appropriate communications software for inter-processor exchanges, and for host to processor direct communication. Combined computing power of a 128-node system with all processors 80% used was about 100 Mips or 8 Mflops, which along with the 64 Mbytes of memory available, provided a relatively powerful computer, although certainly not a supercomputer, at that time.

4.2.2. iPSC2

The more recently (1987) developed iPSC/2 computer is a second generation machine that provides greatly increased processing power and communication throughput, and which is still in use as an experimental machine in many locations. Each node contains an 80386 microprocessor with up to 8 Mbytes of memory (extendible to 16 Mbytes per node for up to 64 nodes). There are three available numeric coprocessors: an Intel 80387 coprocessor (300 Kflops), a Weitek 1167 scalar processor (900 Kflops) and a VX vector board (6 Mflops double precision, maximum of 64 nodes). Thus the top-rated system has 64 nodes capable of 424 Mflops double precision and 1280 Mflops single precision. Special communication processors allow message circuits to be established between remote processors without intervention from intermediate processors. Thus the iPSC/2 implements worm-hole routing rather than the store and forward protocol of previous generations.

4.2.3. iPSC/860

In fall 1989 Intel announced an i860-based version of the older iPSC/2 hypercube system. These iPSC/860 systems are basically iPSC/2 hypercubes with the node processors replaced by Intel i860 processors. In terms of raw floating point performance the peak rate is thereby increased to 60 Mflops per node (40 Mflops for most operations). In practice it is rare that more than 5 Mflops can be realized due to the memory model used by the i860. Some simple vector type kernels, hand-coded in assembler, run at from 28 to 38 Mflops/node. Well-designed Fortran programs are currently yielding about 3-4 Mflops/node due to the poor state of the i860 Fortran compilers. This huge variation between peak rate and realized performance is a serious design flaw in the iPSC/860 and is attributable in large part to deficiencies of the i860 processor which make it hard to compile for and also difficult to deliver data to at rates needed to sustain 40-60 Mflops. Several major i860 compiler efforts are underway and will undoubtedly improve substantially on the early results. Because the communication facilities of the iPSC/860 are those of the iPSC/2, the system is constrained to a maximum of 128 nodes.

While the iPSC/860 utilizes the slow iPSC/2 communication hardware and software, communication proves to be much faster on the i860 system than on the iPSC/2. This is because most of the message startup communication overhead is software overhead involved in negotiating the communication protocol. Because the i860 is so much faster than the 80386, the software overhead is correspondingly decreased. The effect is to reduce messaging startup time by about a factor of three.

The iPSC/860 actually supports heterogeneous boards - a mixture of i860 and 80386 boards is allowed. This permits special 80386 nodes to take advantage of the flexible interfaces to graphics, disk and other peripherals available to that processor. For example 780 Mbyte disks may be attached to such nodes via 4Mbyte/sec SCSI interfaces. Frame buffers, VMEbus devices and Ethernet interfaces also plug into these boards.

4.2.4. Intel DELTA

In 1990 Intel developed a prototype system, the Intel DELTA (Touchstone). This system is a rectangular grid version of the iPSC/860, using the same node processors but an entirely new communication network. There are 8 communication paths per node, allowing 4 bidirectional channels as required for a two-dimensional grid. With the new communication structure, the iPSC is freed from the constraint of a maximum of 128 nodes. Indeed Intel has delivered a 512 processor prototype version of the system to CalTech. With a peak processing rate of 20 Gflops this is a formidable system in the 1992 time frame. This system will not be available commercially.

The DELTA shows enormous improvement over the iPSC/860 in terms of communication performance. For example while the iPSC/860 delivers a node-to-node bandwidth of 2 Mbytes/sec, the DELTA delivers 17 Mbytes/sec. Both systems have a comparable message latency of about 60 microseconds.

4.2.5. Intel Paragon

In late 1992, Intel began shipping a commercial upgrade of the DELTA, called Paragon. The Paragon uses the same rectangular grid structure as the DELTA, but faster processing nodes. The system is designed with scalability in mind from the outset. Initial systems are designed to have up to 2048 nodes with a peak rate of 300 Gflops. The largest systems will have 128 Gbytes of main memory, with an aggregate 500 Gbyte/sec aggregate bandwidth, and access to over 1 Tbyte of internal disk with an aggregate bandwidth of 6.4 Gbyte/sec. Communication bandwidth between nodes is 200 Mbytes/sec full duplex

Paragon software plans indicate a substantial divergence from previous Intel systems. The basic software environment is the same as on the iPSC2 - a library of message passing routines. However the Paragon also supports a full UNIX (Mach) kernel in each node, along with a node-level virtual memory. Finally Intel has indicated that a virtual shared memory capability will also be available across nodes.

The Paragon node contains two identical Intel i860XP processors, an improved 50MHz version of the 40 MHz i860 used in previous Intel systems. This processor has peak rates of 75 Mflops (64-bit) and 42 Mips and can support from 16-128 Mbytes with a 400 Mbyte/sec processor-memory bandwidth and an 800 Mbyte/sec processor-cache bandwidth. The second processor on a node is dedicated entirely to communications processing.

Paragon nodes are organized into three partitions: the Compute partition, the Service partition and the I/O partition. Parallel applications and a UNIX micro-kernel reside on the Compute partition. The Compute partition can be subdivided into sub-partitions allocated to either interactive or batch processing, and there may be any number of each kind. Partition sizes and shapes may be changed at any time. Batch processing is provided through the standard NQS system. The Service partition provides full operating system facilities such as shells, editors and compilers. This partition can grow or shrink in time with the system running, according to user needs. Compute partition and Service partition nodes are identical, allowing re-partitioning between these partitions at any time. The I/O partition provides disk, tape and network connections. I/O nodes include SCSI nodes for disks and tapes, VME nodes for specialized devices, and HiPPI nodes for connection to disk arrays and frame buffers. These nodes can also be used as Service partition nodes, but are never allocated to the Compute partition. By increasing the I/O partition size as the system grows, I/O capabilities can scale to match the computational capabilities. Applications can avail of both UNIX OSF/1 facilities and Intel NX/2 operating system facilities for interaction between nodes, or with Service partition nodes.

For interactive use, users can login directly or remote applications can perform remote procedure calls to the service partition. Logins may use standard UNIX tools such as xterm, telnet or rlogin and can use the UNIX shell of their choice on a service node. When compilations and other services are performed, the OS load balances the service partition by distributing tasks among the service nodes. Users and applications see the system as a single UNIX system. Process ID's are unique across the system, and a process can signal any other process. Similarly the file system is accessible from all nodes and file path-names are independent of the node accessing the file. Users are also known across the system (e.g. the UNIX who command).

Paragon supports both X11 Release 5 and Silicon Graphics Distributed Graphics Library (DGL), allowing applications to direct output to a graphics workstation. The X implementation includes PHIGS 2D and 3D capability in the form of the PEX extension of X. The OSF/Motif system is also supported. Silicon Graphics and IBM RS/6000 workstations may obtain enhanced performance using the DGL system. The AVS and Explorer interactive visualizers are also supported, allowing users to develop AVS or Explorer models that run on Paragon compute nodes, which are linked to AVS or Explorer visualizers on work-stations. For highest graphics performance, the Paragon can be configured with one or more HiPPI frame buffers, each connected by a 100 Mbytes/sec HiPPI channel. In this mode, full-resolution graphics with animation rates of 60 frames per second can be achieved.

The Paragon interconnection network is based on a specialized Paragon Mesh Routing Chip (PMRC) which can simultaneously route traffic moving in the four network directions, and to and from the node processor, at speeds exceeding 200 Mbytes/sec. The PMRC latency is 40 ns. to make the routing decision and close the required switches. Message traffic moves across multiple router switches and is pipelined along the wires so that speed is essentially independent of distance.

Paragon is configured to support communication at HiPPI (800 Mbits/sec), FDDI (100 Mbits/sec) and Ethernet (10 Mbits/sec) rates. The Paragon can support multiple HiPPI channels, and the number can scale with the number of nodes, with each connection maintaining its full band-width. HiPPI, FDDI and Ethernet connections are direct to the nodes, and thereby avoid any host interference (there is in fact no host processor for the system).

The Paragon system supports several file systems. The basic file access is to the Virtual File system from Berkeley 4.4BSD UNIX. Access is also available to a standard UNIX 4.3BSD file system and to a UNIX System V system. Finally access to a Network File System (NFS) compatible file system is provided. Libraries are available providing access to UniTree file servers, which allow access to remotely connected disk and tape devices, as well as automatic migration of files through a hierarchy of I/O devices. Features supported include transparent archiving, automatic backup and restore, large file management and shared peripherals.

The support for virtual shared memory on every node means that large applications can be compiled even on a single node and later parallelized to allow them to run on multiple nodes. In cases where an application fits entirely within physical memory all page faults are avoided, allowing high efficiency for such applications while retaining the flexibility of a virtual memory for over-sized applications. The Shared Virtual Memory system allows users to avoid message passing primitives altogether, and instead view the system as a single memory address space. The system automatically maps memory references, one page at a time, between nodes so that the single address space is visible to all nodes. This allows applications to share variables across node boundaries without explicit message passing. All types of data reference are supported, while ensuring data consistency across nodes.

4.3. SUPRENUM-1

The German SUPRENUM-1 project involves coupling up to 16 processor clusters with a network of 200 Mbit/sec busses. The busses are arranged as a rectangular grid with 4 horizontal and 4 vertical busses. Each cluster consists of 16 processors connected by a fast bus, along with I/O devices for communication to the global bus grid and to disk and host computers. There is a dedicated disk for each cluster. Individual processors can deliver up to 20 Mflops (64-bit) of computing power in chained operations, or 10 Mflops unchained, and support 8 Mbytes of memory. The high speed and versatility of the bus network makes this an interesting machine for a wide range of applications, including those requiring long-range communication. No more than three communication steps are ever required between remote nodes.

SUPRENUM has the best support for programmimg grid-based scientific applications to be found among the various distributed memory MIMD vendors. The effort invested to develop libraries of high-level grid and communication primitives will greatly ease the effort of moving applications to the computer, and also provides substantial high-level portability to other systems, since the communication library can be implemented efficiently in terms of low level primitives on almost any distributed system.

The first 64-processor system was delivered in December 1989 and was upgraded to 256 processors in 1991. SUPRENUM-1 has a 5 Gflops peak rating and high realizable efficiency in suitable applications - performance as high as 1300 Mflops has been measured. We have recently benchmarked the Shallow Water Equations on the SUPRENUM-1 obtaining 5.33 Mflops per node, which equals or slightly exceeds typical iPSC2/860 performance.

4.4. NCUBE2

NCUBE introduced the NCUBE hypercube at around the same time that Intel developed the iPSC1. The NCUBE is characterized by having a custom designed scalar processor, and allowing up to 1024 processors as compared to 128 for the iPSC1.

Recently NCUBE has introduced the second generation NCUBE2. This system is extensible up to an 8192 node hypercube with a peak rating of 60 Gips, 27 Gflops (32-bit) and 19 Gflops (64-bit). The full size system costs over $20M, and supports 512 Gbytes of memory and 16 Tbytes of disk. 1024 processor systems have been shipped to Shell Oil and to Sandia National Laboratory. The high cost per Mflops of the NCUBE2 compared to the Intel iPSC/860 may prove to be an obstacle to marketing success.

The key advance in the NCUBE2 is the development of a 64-bit 20MHz custom processor providing high scalar performance. The communication processor supports 28 DMA channels for communication to neighboring processors, allowing 14 bidirectional connections to nearest neighbors in a hypercube.

5. Physical and Virtual Shared Memory Architectures

5.1. Myrias SPS-2

The Myrias SPS-2 computer, built by Myrias Research Corp. of Edmonton, Alberta, has up to 1024 processing elements. The architecture is a hierarchical bus design, utilizing 33 Mbyte/sec busses to interconnect processors within clusters and clusters to each other. Each processor is a 32-bit Motorola 68020 microprocessor with 4 Mbytes of local memory. The architecture is a three-level hierarchical system. Processors are assembled in groups of four on a board, connected among each other by a bus, along with an I/O port controller. At the second level in the hierarchy is the card cage, containing 16 processor boards and thus 64 processors, as well as one or two off-cage communication boards. Each communication board supports four off-cage links which can be connected to other cages or to the front end computer.

The SPS-2 supports a global 32-bit virtual address space. There is no concept of shared access to memory locations. Simple extensions of Fortran77 support parallel do and join operations. The PARDO model used by Myrias is somewhat unusual in that there are no possibilities for explicit sharing of data. A PARDO is executed by specifying a code segment to be executed and the number of child tasks to be run. Each thread of execution performs completely independently in its own address space, starting with a copy of the parents memory. Execution of a child proceeds in normal sequential mode, except that PARDO's may be nested recursively. On completion of all children, the memory states of the children are merged to form the new memory state of the parent. Thus a child can never affect the memory of another child, but can affect the memory state of its parent, but only after all children merge.

The rules for merging of child memories at an address on task completion are:

- If no child stored a value at the address, the location in the parent memory retains its original value.
- If exactly one child changed a value at the address, the location in the parent receives the last value from the child.
- If more than one child stores a value at the address the result is unpredictable unless all values stored are the same.

Efficiency is maintained throughout the process by using a copy-on-write approach which ensures that most of the global address space is never really replicated.

Myrias later modified the software environment of the SPS-2 to provide for more flexibility in programming than is afforded by the PARDO model. Initially reduction variables were provided for through compiler directives. Later, facilities for sharing variables and for explicit message passing were added to the system. A

1040 node SPS-2 was constructed for a short time from smalle systems and operated successfully.

5.1.1. Myrias SPS-3

The SPS-3 is essentially similar to the SPS-2, but incoporates a much faster Motorola 68030 processor instead of the 68020 used on the SPS-2. Myrias has also discussed the possibility of a 68040-based SPS-4.

5.2. Kendall Square KSR1

The KSR1 is a highly parallel computer system designed to be scalable to thousands of processors while preserving the simplicity and familiarity of a shared memory programming model. Each processor is a RISC-style superscalar 64-bit unit, operating at 20 Mips and 40 Mflops (peak). A KSR1 system contains from eight to 1088 processors with a peak performance range from 320 to 43,520 Mflops, all sharing a common virtual address space of one million megabytes (2**40 bytes).

5.2.1. KSR1 Software

The KSR Operating System is an extension of OSF/1 (Mach). As such, it is a very complete implementation of all of Unix. KSR OS is fully compatible with 4.3BSD UNIX (which has no official validation suite). In addition, it will pass the validation suites for ATT System V Release 3, base and kernel extensions, X/Open XPG3, and POSIX.

KSR OS does not use any front end machines and there is no distinguished processor. Thus, there are no OS bottlenecks and no reason to limit the traditional Unix flexibility. In particular, KSR OS supports an arbitrarily large number of multi-threaded processes timesharing a large number of processors. This ability to timeshare is crucial in many interactive applications, in which periods of intense computing are followed by human time scale periods of thought. Interactive applications spanning the entire range between state of the art numeric processing guided by a user involving scientific visualization all the way to traditional transaction processing as practiced by banks and airlines are all efficiently supported in the KSR OS environment.

The KSR1 environment is what a sophisticated Unix user would expect to find. At the user interface level, there is X11 and Motif. At the language level, there is Fortran, with automatic parallelization, C (both the ANSI and PCC dialects), and IBM-compatible COBOL. At the database level there is the ORACLE relational database management system (RDBMS), including application development tools. Kendall Square Research is extending ORACLE's features for the parallel environment. At the transaction processing level, there is AT&T's Tuxedo/T and Tuxedo/D, fast non-relational file access methods, and fourth generation languages.

For decision support applications, ORACLE for KSR1 will provide automatic parallel processing for complex queries. Kendall Square Research has developed a general purpose technique called Query Decomposition which automatically parallelizes SQL queries generated by ORACLE-based applications. Future third party RDBMS software ported to the KSR1 will also take advantage of the Query Decomposition tool.

Parallel Fortran programming on the KSR1 can be fully-automatic, semi-automatic or manual. The parallel programming environment of the KSR1 is based on a proprietary parallel run-time system (PRESTO), that dynamically executes run-time decisions based on compiler-generated or programmer-specified directives. The functioning of the runtime system is one of the keys to KSR's dramatically improved performance. The system dynamically decides the level of resources it will devote to a particular parallel task at runtime based on the amount of calculation required at this particular time and the resources available at that time, rather than by making a static decision about the resource allocation question at compile time. The result of this policy is that real world problems that have significant variations in their processing requirements can be run together, taking advantage of all the cycles on the machine rather than running them one at a time, wasting cycles in those parts of the program that don't exhibit maximum parallelism.

While offering a highly parallel applications development environment, Kendall Square Research all provides scientific and mathematical subroutine libraries, and important third-party software packages for computational fluid dynamics, quantum chemistry, mathematical algorithms for engineering applications, molecular dynamic modeling for computational chemistry, and finite element analysis for engineering applications.

5.2.2. Shared vs. Distributed Memory

The KSR1's shared memory programming model is made possible by a new architectural technique called ALLCACHE memory. The KSR1 memory system is designed to do for distributed memory what virtual memory did for hierarchical memory. It replaces the complexity and rigidity of the physical mechanism with a uniform address space, now shared by a set of processors. System hardware and software maps this space into physical devices. The KSR1 ALLCACHE memory system, achieves this programming simplicity without sacrificing the benefit of distributed memory (i.e. scalability) - in fact its performance continues to be good even as the number of processors grows very large.

The memory models of today's highly parallel computer architectures raise problems for programmers which are reminiscent of storage management in the 1960s. Twenty-five years ago, storage management via overlay structures was an integral part of the job of writing a program. Necessarily, programmers attacked the task with a static analysis of the memory requirements of a single program.

Advances in programming practice and system architectures, however, gradually rendered static storage management unnecessary. The goals of machine independence, re-use of modular program elements, and algorithms of high complexity characterized by data structures of widely varying size and shape were inconsistent with static, programmer controlled storage management. In addition, the introduction of system environments in which computers were organized for simultaneous use by several programs made it impossible for the author of a single program to predict accurately the time-varying storage requirements of the entire system. Ultimately, these factors led to the adoption of virtual memory as a near-universal feature of storage management in modern computer architectures. Virtual memory makes storage management dynamic and largely automatic. It permits programmers to write applications with a storage abstraction which is simple and powerful - a single uniform address space. System hardware and software maps this space into physical devices.

Highly parallel computer architectures revisit these early storage management issues, with a new twist. All of the highly parallel systems that have been introduced have distributed memories. That is, the physical memory comprises a set of memory units, each connected to a unique processor. The processor-memory pairs are interconnected by a network. Distributed memories have been universal among highly parallel machines because they provide the only known means of providing completely scalable access to memory. That is to say access whose bandwidth increases in direct proportion to the number of processors. In most of today's parallel systems, the job of managing the movement of code and data among these distributed memory units belongs to the programmer. The job is similar in style to the task of managing the migration of data back and forth between primary and secondary storage prior to the introduction of virtual memory, but it is much more complex. As before, programmers need to be concerned about exactly what will fit where and what to remove to make room for something new. Now however there are thousands of memory units to deal with instead of just two or three.

5.2.3. The KSR ALLCACHE Shared Memory

ALLCACHE memory provides programmers with a uniform 2^{40} byte (1 million Mbyte) address space for instructions and data. This space is called system virtual address space or SVA space. The contents of SVA locations are physically stored in a distributed fashion. ALLCACHE memory physically comprises a set of memory arrays called local caches, each capable of storing 32 Mbytes. There is one local cache for each processor in the system. Hardware mechanisms (the search engine described later) cause SVA addresses and their contents to materialize in the local cache of a processor when the address is referenced by that processor. The address and data remain at that local cache until the space is required for other purposes.

As the name suggests, ALLCACHE memory behavior is like that of familiar caches: data moves to the point of reference on demand. However, unlike the typical cache architecture (which we might call SOMECACHE memory), the source for the data which materializes in a local cache is not main memory but rather another local cache. In fact, all of the memory in the machine consists of large, communicating, local caches. The main memory of the machine is identical to the collection of local caches. The address and data that materialize in local cache A in response to a reference by processor A may continue to reside simultaneously in other local caches. Consistency is maintained by distinguishing the type of reference made by processor A:

- If the data in the location will be modified by A, the local cache will receive the one and only instance of an address and its data.
- If the data will be read but not modified by A, the local cache will receive a copy of the address and its data.

When processor A first references the address X, the ALLCACHE memory searches that processor's local cache to see if the requested location is already stored there. If not, a hardware search engine locates another local cache (say, local cache B) where the address and data exist.

If the processor request being serviced is a read request (for example, to load the value into a register) then the search engine will copy the address and data from local cache B into local cache A. The amount of data copied will be 128 bytes, called a sub-page. At the end of this operation the sub-page will reside at both A and B.

If the processor request is a write request (for example, to store the contents of a register into this location) then the search engine will remove the copy of the sub-page from local cache B as well as from any other local caches where it may exist before copying it into local cache A. Thus the search engine is responsible for finding and copying sub-pages stored in local caches and for maintaining consistency by eliminating old copies when new contents are stored.

In order to maintain consistency, each local cache records state information about the sub-pages it has stored. These states are specific to the physical instance of a sub-page within a particular local cache. Thus a single sub-page in SVA space may be in Invalid state in one local cache and in Copy state in another. Some sub-page states are used and maintained exclusively by hardware as part of the operation of the search engine. Others can be manipulated indirectly by the operation of software.

There are times when two or more processors must synchronize their access to SVA locations. The ALLCACHE memory supports this requirement through instructions which lock and unlock sub-pages. These instructions can be used to implement any multiprocessor synchronization function including data locks, barriers, critical regions, and condition variables. These forms of synchronization and others are available via KSR compilers, libraries, and OS calls.

A "lock" in ALLCACHE memory is achieved by setting a sub-page to the Atomic state. A program does that by issuing a GET instruction on the address of a byte within the desired sub-page. This instruction will cause the search engine to find the sub-page and, if the page is not in Atomic state, return it to the requesting processor in Atomic state. In the process the search engine will ensure that all other copies of the sub-page are set Invalid. If the sub-page is already Atomic it will not be returned to the requester immediately. Instead the request packet will return to the requester with an indicator that the sub-page was found in the Atomic state. A program removes Atomic state from a sub-page by issuing the RELEASE instruction.

5.2.4. The KSR Search Engine

In addition to the basic functional roles of the search engine (finding sub-pages within the set of local caches and maintaining consistency), the search engine must be scalable. It must be implemented in such a way that good performance continues to be delivered as the number of processors grows. This objective is achieved in the KSR1 by implementing the search engine as a hierarchy. The KSR1 search engine is a two-level hierarchy of uni-directional rings. Each ring is a sequence of point-to-point connections among a set of units, with the last unit in the set being connected back to the first. Each unit is a combination of a router for request/response packets and a directory. The router can move a packet farther along the ring or send it up or down in the hierarchy. All of the units on all rings can operate simultaneously, so the search engine is a highly parallel mechanism.

The lowest level rings are called Search-Engine:0s (or SE:0). Each SE:0 can be configured to contain from eight to 32 processor/local cache pairs. Each processor/local cache pair is connected to exactly one SE:0 via a unit which contains a directory for that local cache. There is one entry in the directory for each page allocated in the local cache. The entry gives the SVA address of the page and the state of each of its sub-pages. When a packet passes such a unit, it can determine whether the sub-page the packet is seeking can be found in the desired state in the local cache. If so, the unit routes the packet there, if not it moves the packet on to the next unit on the ring. The unit on a SE:0 which connects upward to the next higher level is called an ALLCACHE Routing and Directory cell or ARD. It contains a directory covering the entire SE:0 - there is an entry in its directory for every page allocated on every local cache on the ring.

When a packet reaches an ARD it will be moved to the next unit on the SE:0 if the directory in the ARD indicates that the data sought is on the SE:0. If not, the packet is routed up to the next higher level in the hierarchy. The ring at the top level of a KSR1 is called Search-Engine-l (or SE:1). SE:l becomes involved in a search operation when a processor requests a sub-page which is stored (for the moment) in a local cache on a different SE:0. A SE:1 can be configured to connect two to 34 SE:0s. Hence the maximum system size in a KSR1 is 1088 (i.e. 34×32) processors with 34 Gbytes of ALLCACHE memory. SE:l is composed of ARDs, each containing a directory for the SE:0 to which it is connected. This directory is essentially a duplicate of the one stored in the ARD on the corresponding SE:0. When a packet reaches an ARD on SE:l, it will be moved to the next ARD on the ring if the directory in the ARD indicates that the data sought is not on the corresponding SE:0. Otherwise, the packet is routed down to the ARD on SE:0.

In the KSR1 the packet passing speed of an SE:0 is 8 million packets per second. An SE:l can be configured to handle 8, 16, or 32 million packets per second. Each packet contains 128 bytes of data. Hence the SE:0 bandwidth is 1 Gbyte/sec and the SE:1 bandwidth ranges from 1 to 4 Gbytes/sec.

5.2.5. The KSR1 Processor

The KSR1 processor is a four chip set implemented in 1.2 micron CMOS. One of these chips, called the Cell Execution Unit or CEU, is the basic control unit of the processor. On each clock cycle it fetches two instructions from memory. Certain instructions (loads, stores, branches, address arithmetic) will be executed directly by the CEU; others will be forwarded to a coprocessor for execution. The CEU is responsible for all instructions dealing with memory.

These instructions operate on 40 bit addresses. This design characteristic of the processor architecture is fundamental to the system design. In order to build a shared memory multi-processor with large numbers of processors, a large address is essential, 32 bits is not sufficient to address the amount of memory required. The KSR1 architecture actually envisions a 64 bit address (pointers are stored as 64 bit quantities) but, due to implementation constraints, the first generation address size is 40 bits, and that is clearly sufficient for the 1088 processor systems being built at this time. The CEU has 32 address registers, each 40 bits wide.

The CEU operates with three co-processors:

FPU (Floating Point Unit)

This chip executes arithmetic operations on IEEE floating point format values. It has 64 registers each 64 bits wide. It supports linked triad instructions in which two floating point operations are initiated from a single instruction, giving a peak floating point rate of 40 Mflops. Sustained floating point performance depends on the application, of course. Examples include: 6.6 Mflops (Livermore Loops harmonic mean), 15 Mflops (100×100 Linpack), 28 Mflops (FFT), and 32 Mflops (Matrix Multiply).

IPU (Integer and Logical Operations Unit)

This chip performs arithmetic and logical operations on 64 bit integers stored in 32 registers (each 64 bits wide).

XIU (I/O Channel Unit)

This chip provides a 30 Mbytes/sec pathway to peripheral devices. Since there is an XIU on every cell, large systems can be configured with very high aggregate bandwidth to disk drives and networks.

5.2.6. KSR1 Networking

KSR1 supports an extensive set of connectivity technology and standard protocols including:

- TCP/IP, NFS, DCE, SNA-3270, 3770, LU6.2/PU2.1, ISO/OSI X.25, X.29, X.28, X.3 and supports standard protocols including:
- Ethernet, Token Ring, HiPPI, and FDDI transports and;
- Industry standard buses, the first of which is VME, to facilitate the integration of third-party communication products.

5.2.7. KSR1 Performance

The system is so new that detailed performance measurements are only now becoming available. The standard 1000x1000 LINPACK algorithm has been ported to the KSR1 at Oak Ridge National Laboratory and yields 500 Mflops on 32 nodes as compared to 300 Mflops for the Intel DELTA and 150 Mflops for the iPSC/860. The three comparison systems each had 40 Mflops processors and each had 32 nodes. A key reason for the better KSR performance was the communication performance as exhibited in the table below.

System	Bandwidth	Communication Latency
KSR1	34 MB/sec	6 microsecs
Intel DELTA	17 MB/sec	60 microsecs
iPSC/860	2 MB/sec	60 microsecs

5.3. Alliant FX/2800 and CAMPUS/800

Alliant Corporation has developed a series of progressively more powerful parallel shared memory systems, beginning with the eight processor FX/8, leading to the FX/80 and FX/2800 systems, and culminating in the 800 processor CAMPUS/800. We will discuss the two most recent systems in detail. Unfortunately Alliant entered bankruptcy proceedings in 1992 leaving the future of these systems in doubt.

5.3.1. Alliant FX/2800

The FX/2800 was, on introduction, the first general-purpose, shared memory, parallel supercomputer that used standard VLSI processors rather than a proprietary processor architecture. It was also the first truly open supercomputer because it was standard at the processor/instruction set level as well as at the usual UNIX level (UNIX, NFS, NQS, compilers, etc.). The FX/2800 supports parallelism at multiple levels - instruction-level, loop-level, and task level - in an open supercomputing environment. The system supports a specialized high-performance graphics subsystem.

The FX/2800 series has up to 28 i860 processors with up to 1 Gbytes of shared memory and 4 Mbytes of global cache. The CPUs are connected to the cache via a 16×8 1.28 Gbyte/sec crossbar. The bus from memory to cache is 72 bits wide and runs at 640 Mbyte/sec. The system can be configured with up to 8 modules, at least one of which must be an I/O Module (IOM) containing an i860 and two 40 Mbyte/sec peak I/O channels. The channels can support either a VME bus or a closely-coupled frame buffer for graphics. Multiple 25 Mbyte/sec VME channels are supported with disk striping, UltraNet, and other services. The remaining modules are Processor Modules (PM) containing four i860 processors each. Each i860 has its own on-board caches. In addition to the standard i860 instruction set, Alliant has added custom VLSI implementing the parallel instructions from the FX/8 and FX/80 series.

The FX/2800 OS is Concentrix, a parallel UNIX, and it is very similar to the operating system on the older machines. The system supports Parallel FX/Fortran, FX/C and FX/ADA compilers. Also there is a real-time FX/RT executive which is priority-driven with preemptive scheduling, and is co-resident with UNIX. The FX/2800 is source-code compatible with the FX/80 Series.

The graphics subsystem is basically a frame buffer; transformations and rendering are done on one or more of the CPUs. The ability to have multiple processors writing simultaneously into the frame buffer is relatively unique. The frame buffer is supported with visualization capabilities including X11, PHIGS/PHIGS+, and several visualization toolkits.

The FX/2800 is compatible with the PAX (Parallel Architecture eXtension) ABI jointly defined by Intel and Alliant. Any vendor who wants to build a binary-compatible system can buy the i860 processors from Intel, buy Alliant's concurrency control architecture and parallelizing compilers (which Alliant has licensed to Intel) and any software vendor can produce a single binary version of his application that runs on a variety of machines, from workstations to parallel supercomputers like Alliant's.

The FX/2800 system is rated at 720 Mflops on the 1000×1000 LINPACK benchmark. In comparison, the single-processor Cray Y-MP/832 is rated at 308 Mflops, the C240 at 166 Mflops, and the VAX 9000/440VP at 312 Mflops. Other performance metrics include: over 1.12 peak Gflops (DP), 1148 VAX Mips (aggregate, based on Dhrystone V1.1) and 672 Whetstone Mips (non-inlined, aggregate).

The processors in the FX/2800 can be used as parallel processors or as multiprocessors. Up to six parallel clusters are supported. The scheduler automatically "breaks up" a cluster into independent multiprocessors if there are no parallel jobs waiting to execute, or automatically breaks clusters up in user-defined time-slices.

Each cluster consists of up to 14 processors controlled via hardware-based, concurrency control instructions that are automatically generated by the compilers.

The compilers detect opportunities for fine-grained parallelism, typically at the loop-level. (Up to 28 processors in the cluster are supported in certain situations, such as in the 1000×1000 LINPACK test).

Explicit parallelism via compiler directives or UNIX tasking is also supported. Note that UNIX itself runs directly on the i860 processors in an SMP implementation. There is no "front-end" or host processor for the system.

The i860 also has some interesting instruction-level parallelism features. It supports "superscalar" operations (up to three instructions per clock cycle - RISC integer/control, FP MUL and FP ADD). This requires sophisticated instruction scheduling in the compiler. The chip also supports pipelined floating operations, which allows compilers to produce code that has been optimized for both vectorization and concurrency.

5.3.2. Alliant CAMPUS/800

In Fall 1991, Alliant introduced a supercomputer based on clusters of FX/2800 systems. The CAMPUS/800 MPP system provides up to 32 peak Gflops of computing power and 128 Gbytes of memory. A fully configured CAMPUS/800 system is composed of 32 supercomputer ClusterNodes with each ClusterNode consisting of a cluster of 25 Intel 64-bit i860 RISC processors and 4 Gbytes of high-speed shared memory. For rapid data sharing and synchronization, the ClusterNodes communicate over a 2.56 Gbyte/sec High-speed Memory Interconnect (HMI).

Each ClusterNode is based on the FX/2800 supercomputer, enhanced via hardware and software for high-performance MPP operation. CAMPUS can be used as a single large MPP system for extremely large problems or as multiple autonomous supercomputers for individual groups. Advanced operating system software dynamically partitions the CAMPUS system based on user requirements and changing workloads. The ClusterNodes are connected together by HiPPI connections at 100 Mbytes/sec.

Ease-of-programming is provided by a sophisticated set of development tools, including Alliant's automatic parallel compilers and libraries, and the high-level EXPRESS environment from ParaSoft Corporation. Other standard software tools for CAMPUS include TCGMSG and P4 (from Argonne National Laboratory) and PICL (from Oak Ridge National Laboratory).

CAMPUS was the first MPP system with a full-function UNIX operating system executing on each node of the MPP system itself. Other MPP vendors rely on a separate workstation or *front-end* to execute the operating system and provide services such as time-sharing access, UNIX file I/O, network connectivity and software development facilities. This limits the number of simultaneous users, since access to the system is limited by the power of the front-end, and increases the overall cost of the system. In contrast, the CAMPUS system supports simultaneous execution of computational tasks with UNIX file I/O, network activity and development on each of the ClusterNodes, thus providing a significantly more versatile and robust operating environment and supporting a greater number of users.

The CAMPUS/800 system is priced at \$2.5M for a typical configuration with 100 processors. An 800-processor CAMPUS system is priced at \$16M. FX/2800 Family systems start at under \$200,000 and can be used as fully-compatible CAMPUS starter nodes.

5.4. The Cray Research MPP

Recently Cray Research Inc. (CRI) has announced plans to develop a massively parallel system which they refer to as the CRAY MPP. Little released information is currently available about the CRAY MPP hardware. Most of this report [1] will therefore focus on the software model for the MPP. This report describes the initial MPP Fortran programming model which will be supported on the first phase MPP systems. Based on existing and proposed standards, it is a work sharing model which combines features from existing models in a way that is both efficiently implementable and useful.

5.4.1. Hardware

MPP Systems will come in sizes ranging from 128, 256, 512 to 1024 nodes (and possibly larger). Each node, referred to below as a PE, will be based on the new Digital Equipment Corporation Alpha micro-processor, which has a peak rate of approximately 240 Mflops. It appears that the processors will be connected in a three-dimensional grid or torus, although this is not certain. The amount of memory per node is not currently available. The system will apparently require a CRAY C-90 as a front end processor, although this might be relaxed to a CRAY Y-MP for the smallest systems.

5.4.2. Programming Model

5.4.2.1. Overview

The CRAY MPP programming model is a work sharing model rather than a fork-join model. A work sharing model distributes parallel work across existing tasks, while fork-join models create new tasks to execute parallel work. The model includes features that allow a user to define a program in terms of the behavior of the system as a whole, where the behavior of individual tasks is implicit from this systemic definition. In general, features described as shared support this perspective.

---

[1]. Information in this report has been abstracted from discussions with and papers received from: Brian Dodd, Tom MacDonald, Andrew Meltzer, Douglas M. Pase, all of Cray Research, Inc.

The model also supports an opposite perspective, where a program may be defined in terms of the behaviors of individual tasks, and a program is implicitly the sum of the behaviors of all tasks. Features described as private are designed to support this perspective.

Users can exploit any combination of either set of features without ambiguity, and thus are free to define a program from whatever perspective is most appropriate to the problem at hand. The work distribution directives are adapted from a wide variety of sources, including Cray Autotasking and various workshops on parallel programming. The data distribution declarations were adapted in part from Rice University's FortranD project.

The programming model allows users to write programs which execute in a data-parallel fashion. It also allows the user to control processor element PE execution more explicitly, as occurs in a single-program, multiple data SPMD model. The major elements of this programming model include the access and placement of data, parallel execution, the distribution of loop iterations, synchronization primitives, distributed and sequential I/O, subroutine interfaces, and special intrinsic functions that support parallel reductions, parallel prefix operations, and segmented scan operations. Additional intrinsic functions allow the user to access low level detail about array distributions as well as more abstract concepts like the virtual machine topology used to distribute one or more arrays.

This programming model distinguishes between data objects that are private to a task and those that are shared among all tasks. There is a one to one relationship between tasks and PEs. Private data objects in this model, whether scalars or arrays, are not accessible to any other task. They are not distributed across PEs, but instead each private object resides entirely on only one PE. Thus each task that references a private object references its own private version of that object; the storage for the object is replicated across the PEs. Shared data objects, in contrast, are accessible to all tasks, are not replicated, and if the object is an array may be distributed across multiple PEs.

Programs initially execute as a single task on a single PE, and parallelism is initiated explicitly by the user. Tasks are not created dynamically, but rather one task is created per processor and parked until parallel execution is invoked. Each task also has an identity which it can use to distinguish itself from other tasks. Loops do not create parallelism in a program. Rather they may execute serially, one loop in each task, or they may share the work, distributing it across all available processors. Distributed loops, called shared loops in this model, are work sharing constructs rather than task creating constructs. A rich set of iteration distribution mechanisms is provided in order to support highly efficient execution under a variety of conditions. Local loops, or loops that are not distributed, called private loops in this model, are included as well. They allow a user to write programs by defining what each individual task will accomplish within each loop.

The standard shared memory synchronization primitives are supported in this model. A user can place barriers, locks, critical sections, and events within a program. It is anticipated that barriers, locks, and critical sections will be supported by extremely fast hardware mechanisms. This hardware support will greatly reduce the impact on performance that using these primitives has entailed on other systems.

Input and output also have distributed and non-distributed versions. Distributed I/O statements, also called shared I/O, allow the user to specify what all tasks will read or write as a whole, and allow the system to distribute the work as it sees fit. This works in the same way as shared loops in that the user specifies what work is to be done by all tasks and allows the system to divide up the work among the tasks. Sequential I/O is also supported, and it allows the user to specify the work to be done by each individual task. Sequential I/O, also called private I/O, is similar in concept to local loops in that each task must do all of the work specified by the statement.

Subroutine interfaces are extended to accommodate distributed data. While it is tempting to require that the distribution attributes of arguments in function calls exactly match the corresponding formal arguments in function definitions, it is perceived that such a restriction would cause undue hardship on the programmer in many circumstances. On the other hand, supporting such a restriction holds the potential of producing functions that execute significantly faster than their more general counterparts. This model offers a compromise by allowing a user to specify or not specify the distribution attributes of dummy arguments. When the attributes are given the compiler generates the more efficient code for those references. When they are not, the more general and less efficient code is generated. When different distributions are specified, redistribution is done implicitly by the compiler.

Two categories of intrinsic functions are supported in this model: high level reduction and reduction-like functions, and low level functions that give information about array distributions. The high level functions perform several similar computations. They perform reduction operations, parallel prefix operations, and segmented scan operations. These three types of operations work in general for any associative binary operator. The low level intrinsic functions provide usable information about how an array is distributed across the machine.

The model supports message passing primitives based on the PVM model Parallel Virtual Machine, a public domain set of portable message passing primitives originally from the Oak Ridge National Laboratory. These primitives allow an explicit message passing style of programming. Directives were chosen to allow codes written using this programming model to run correctly on machines that do not support the directives. Code written using this model produces mathematically identical results modulo hardware limitation considerations on a sequential machine if the directives are ignored, so long as there is no non-deterministic behavior in the users program and the program does not use any of the MPP specific intrinsic functions.

5.4.2.2. Data Objects

A data object is any program data storage area, whether it is a common block, an array, or a scalar variable. The programming model supports two basic sets of data object attributes. The first set of attributes are called PE PRIVATE, because data objects with this set of attributes are private to every processing element; they are accessible only to the task that owns them. The second are called shared, because such objects are accessible to all tasks.

a. Private Objects

Private data storage is replicated. Each declaration of a private array or scalar object causes one such object with the specified name to be created for each task that executes. Private data is not distributed. An object is always allocated completely on the task and PE that is able to reference it. Private data objects are intended to support, among other things, a user's ability to control the execution of individual tasks at an arbitrarily fine level of detail. Data can be specified to be private using the PE PRIVATE directive. The default distribution attribute is PE PRIVATE: all data objects are assumed to be private unless explicitly stated otherwise. Initial values for private data objects are undefined, but private objects may be explicitly initialized by DATA statements when it is permitted in cf77 to do so. If it is permitted in cf77 to initialize a variable with a DATA statement in a sequential program, it is permitted to initialize the same variable declared with the PE PRIVATE attribute in this model. This means that all private data objects may be DATA initialized except those that occur in blank or unnamed common, dummy arguments, and automatic arrays.

b. Shared Objects

Shared data objects are accessible to all tasks. Only one data object exists for each declaration of a shared data object. Scalar variables are always allocated on a single PE, although not all shared scalars are necessarily allocated on the same PE. Blank common blocks may not be shared, nor may objects in blank common be shared, but objects local to a subroutine, including automatic arrays, can be shared. Character data may not be shared. A shared object is considered distributed across the program's PEs. Shared data objects' distributions fall generally into two categories: canonical distributions and dimensional distributions.

Canonical distributions spread the elements across the machine independently of the dimensionality or rank of the object being distributed. The first few elements are allocated on PE 0, the second few elements are allocated on PE 1, etc., until a block of elements is allocated on the last PE. The allocation then wraps around, putting the next block on PE 0, then PE 1, etc. This process continues until all elements in the object are allocated. The block size used in the distribution is implementation dependent and may be different from one generation of machine to the next. Multidimensional array references are first linearized before distribution, and canonically distributed common blocks retain their storage associations as defined in Fortran.

Dimensional distributions may be applied to any shared array. They may not be applied to scalar objects or common blocks, although shared arrays in named common blocks can be dimensionally distributed. With dimensional distributions each array dimension is distributed as if it were independent of all other dimensions. For this to occur the number of available processors is factored and each array dimension is assigned some factor appropriate to the dimension size and distribution. Thus a three-dimensional array mapped to a 64 processor machine might have 4 processors mapped to each dimension. This works because $4^3 = 64$. Alternatively it might have 8 processors mapped to the first dimension, 4 mapped to the second, and 2 to the third. The user may specify a preference for one factorization over another by assigning weights to each of the dimensions. The first factorization would be chosen if all dimensions were given the same weight. The second factorization would be chosen if the first dimension weight were 4 because it is 4 times larger than the last dimension, the second dimension weight were 2, and the last dimension weight were 1.

c. Geometry

The concept of geometry is an abstraction of the dimensional distribution that gives two fairly diverse advantages. The first is that it simplifies the maintenance and declaration of arrays with similar dimensional distributions. One can think of it as providing a shorthand for declaring dimensionally distributed arrays. The second advantage is that it creates an abstract object that can be used to specify and determine the relationship between tasks. For example, a three-dimensional geometry represents a three-dimensional torus or mesh, depending on how it is used, with exactly as many nodes in the torus as there are tasks available. A geometry can be used to determine tasks that are neighbors in a computation. The information on neighboring tasks is available through the GEOMMAP and TASKMAP intrinsic functions.

d. Array Redistribution

Some applications can efficiently execute with all data being only statically distributed, but not all applications are like that. It is sometimes the case that a given data layout may yield efficient execution for some phase of the computation, but yield poor efficiency for some other part of the computation. If the two sections of code have sufficient work in them it might be desirable to redistribute the arrays dynamically in order to maximize reference locality. This can be done by declaring additional arrays with the desired distributions, then copying data into the appropriate array just before executing the section of code in question. Arrays may also be redistributed across subroutine boundaries. A dummy argument which is distributed differently than its actual argument in the calling routine is automatically redistributed upon entry to the subroutine by the compiler, and automatically redistributed back to its original distribution at the subroutine exits. If the distributions are identical, or the UNKNOWN or UNKNOWN SHARED directive is used, no redistribution occurs.

5.4.3. Parallel Execution

The MPP Fortran programming model is built primarily around the notion of work sharing. Constructs within this model provide access to mechanisms that distribute work among the available executing tasks. Shared data is distributed across PEs independently of executing tasks. The model supports both sequential regions (code segments executed by a single task) and parallel regions (code segments executed concurrently by one or more tasks). To simplify programming for some situations, each task is given a unique name. A variety of standard facilities are provided for synchronization of parallel tasks. These include Parallel Sections, Parallel Loops, Task Identity, Barriers, Locks, Critical Sections, and Events.

5.4.4. Input and Output

5.4.4.1. Private I/O

As with memory and loops, input and output may be either private or shared. Each task performs private read and write operations as if it were the only task looking at a file. When conflicts arise between two tasks, with one or both attempting to write to the same file, undefined behavior results just as it would between two UNICOS processes on a shared memory multiprocessor system like the Cray Y-MP. Private I/O is the default, so all read, write, open, close, and inquire operations will be private operations unless explicitly declared otherwise. Private I/O is useful when a programmer wishes to specify the I/O behavior from the perspective of what each task does, or when a task must write private data. No other task is required to participate in private I/O, so it may be used to achieve unsynchronized I/O as well. Although private I/O is easy enough to understand in theory, in practice it is difficult to use effectively for reading and writing shared arrays. It requires seeking to a point in a file that represents, because files are linear sequences of bytes or words, a linearization of what may be a multidimensional array.

5.4.4.2. Shared I/O

Shared I/O is most intuitive when used to read shared objects from or write them to a file, although it may also be used to read and write private data. All tasks must participate in shared I/O operations, so the I/O operations carry an implicit barrier to execute. Of course all tasks must have the same list of objects to be read or written, to. Shared I/O allows the programmer to specify the behavior, for the whole machine, of an I/O operation. Thus if a shared array X1:1024 is to be read using shared I/O, however it is done internally, the statement specifies that all processors together are to read 1024 elements and store them in X. Depending perhaps on the distribution of X, one task reads some subset of the elements of X, and another task reads a different subset. If this were specified as a private I/O statement, then each task would read 1024 elements. When X is a private array, the operation specifies that all copies of X be written or read, in the order specified by the processor number, starting from PE 0 and going through the last PE.

Scalar variables, whether shared or private, may also be written or read. Shared scalars cause one datum per scalar to be written or read; private scalars cause NP values per scalar to be written or read, where NP is the number of PE.

5.4.5. Message Passing

Many problems are naturally expressed using message passing and many codes are currently written for MPP machines which currently use message passing. The programming model supports the PVM Parallel Virtual Machine message passing model. PVM has been designed to be a general purpose model which can be used to communicate between heterogeneous systems, as well as within tightly coupled processors such as the CRI MPP, while utilizing the most efficient means of transferring data. The software consists of a library of PVM functions with Fortran and C interfaces. The library includes functions for task creation, message passing, and synchronization.

6. Other Machines

6.1. Fujitsu AM1000

The Fujitsu AM1000 is a scalable (to 1024 processors), single user, parallel computer with three communications networks: a point-point or point-row and or point-column Torus router (T-Net), a scatter-gather network (B-Net), and a synchronization network (S-Net).

The processor node characteristics include

- Sparc IU (25 MHz)
- Weitek Floating Point Processor
- 16 Mbytes DRAM
- DRAM controller [custom]
- 128 Kbytes Cache
- Router Controller (RTC) for T-Net [custom]
- Broadcast Interface (BIF) for B-Net and S-Net [custom]
- Message (Cache) Controller (MSC) [custom]

6.1.1. Network Interface (MSC)

The MSC chip provides both cache control for normal SPARC needs, and the interface to the RTC, BIF, and DRAM. Messages may be sent directly from a cache line to the RTC, and into the network. Messages not in cache are automatically retrieved from DRAM. The MSC also manages DMA reception of data into ring buffers allocated in DRAM.

The MSC chip is fairly critical in this node architecture. The fact that it is also the cache controller, a feature made use of in so-called "line sending", means that any enhancement of the processor speed will require a similar enhancement of the MSC. Perhaps more importantly, integration of the FPU, IU, and cache controller into a single chip will likely force Fujitsu out of this architecture, or to build their own highly integrated SPARC unit with an MSC instead of the vanilla cache controller.

6.1.2. T-Net

This is the network that gets the most use. It is a 2-d torus with a 25 Mbyte/sec per channel bandwidth. A "structured buffer pool algorithm" is implemented on top of worm-hole routing, designed to avoid deadlock. Buffer space in the router interface limits the maximum x and y dimensions of the torus to 32, putting a hard upper bound on the router address space of 1024.

The Torus network supports both point to point communication, and point to row, and/or point to column broadcasts. The point-row/column communication takes the same time as the longest (furthest away) point-point communication. The bandwidth of the network is 25 Mbyte/sec. Latency is described by the function:

$$160ns + 160ns \cdot (x - distance + y - distance) + 160ns \cdot message - length(words).$$

Messages are delimited by "END OF MESSAGE" markers, and appear to have no inherent maximum length. The router does not support message combining.

6.1.3. B-Net

The broadcast network is a hierarchical bus with rings of processors at the leaves. It uses a 32-bit bus with a transfer rate of 50 Mbytes/sec. It's primary function is instruction broadcast from the host to the PEs. However it can also be used to perform scatter-gather operations from the host. For example, if the host has 100 data items, it can use the broadcast network to distribute each item to 1 of 100 processors, or vice versa.

The broadcast network does not operate in any way on the data passed through it, even in the "gather" mode. The host is not connected to the T-Net, but only the B-Net and S-Net.

6.1.4. S-Net

The synchronization network is a tree connected network whose main ability is to provide a fast (1.6 microsec for a 1024 PE machine) global AND function. It does not support any data combining other than boolean AND. Actually, the S-net provides a global barrier synchronization mechanism. The S-net performs asynchronous global and barrier synchronization with global and.

6.1.5. Host IO and OS

Each one of the PEs runs a "mini-kernel" which supports a single active process. The host itself executes a single AM1000 job. There is no timesharing on the system.

A single host provides the interface to the PEs through the B-Net and S-Net. In the current implementation, the host also provides the IO for the system through the B-Net, although a PE based IO solution is in development.

6.1.6. Programming Environment

The current programming environment supports C and Fortran with data-parallel library calls. Future plans call for a parallelizing compiler. Fujitsu also supports a simulator which runs on a Sun workstation.

6.1.7. Performance

One application of note is the implementation of a parallel matrix LU decomposition algorithm. Performance results are in the range of 1 Mflop/PE for a 1000×1000 matrix on a 512 PE machine. The implementation of the parallel LU decomposition seems to have been chosen to provide a test case for the special features of the router (cache line sends, buffered receives, and broadcast).

6.1.8. Packaging

The system places two PE nodes on a roughly 8.5"x11" PC board. The 12 big chips are all ceramic PGA's. Fujitsu fits 256 PEs in a cabinet roughly 2'x5'x5'. Four cabinets make a 1024 processor system.

6.2. Intel iWARP

The iWarp computer is a follow-on to the 100 Mflop Warp system developed at Carnegie-Mellon University. The key advance in the iWarp is the development of a single chip processor combining the following functions: 20 Mflops computational power, 320 Mbyte/sec memory throughput and a communication engine with a latency of only 150 nanoseconds. The processor has been implemented as a 600,000 transistor custom VLSI chip fabricated by Intel Corporation, hence the 'i' in the name iWarp. Up to 64 Mbytes of memory is accessible per processor.

One important point is that the processor accomplishes 20 Mflops without pipelining. The adder unit delivers 5 Mflops (64-bit) or 10 Mflops (32-bit), non-pipelined, as does the multiplier unit. In addition the integer/logical unit delivers 20 Mips. All three units may perform simultaneously.

The system has been designed for flexibility from the start, and can be used efficiently to represent either a general purpose distributed memory computer, or special purpose systolic arrays. The initial iWarp is an 8x8 array of processors delivering 1.2 Gflops, but there are plans to extend this up to 1,024 processors.

The communication facilities of iWarp are based on four input and output ports, each running at 40 Mbytes/sec. An input port of one iWarp processor may be connected directly to the output port of another processor to form a point-to-point communication network. A natural arrangement is thus to create one and two dimensional grids of processors. Because the communication processor performs independently of the numeric processor, worm-hole routing can be supported. Logical channels are supported by multiplexing of the physical communication lines, allowing for deadlock to be broken, and for long messages to be interrupted in worm-hole routing.

One of the advances made in the iWarp project is the development of parallel program generators. These are tied to specific application domains - for example there is one for domain-based scientific computing, and another for image processing. iWarp associated research projects are developing a virtual shared memory system for the machine.

7. References

[1] Flynn, M., "*Some Computer Organizations and Their Effectiveness*", IEEE Transaction on Computer C-21, pp. 948-60.

[2] McBryan, O., "*Hypercube Algorithms and Implementations*", SISSC, 8, pp. 227-287, 1987.

[3] McBryan, O., "*Optimization of Connection Machine Performance*", International Journal of High Speed Computing, vol. 2, no. 1, pp. 23-48, 1990.

[4] McBryan, O., "*New Architectures: Performance Highlights and New Algorithms*", Parallel Computing, vol. 7, pp. 477-499, North-Holland, 1988.

[5]. O. McBryan and R. Pozo, *"Performance Evaluation of the Myrias SPS-2 Computer,"* CS Dept Technical Report CU-CS-505-90 (to appear in Concurrency: Practice and Experience), University of Colorado, Boulder, 1990.

[6]. O. McBryan and R. Pozo, *"Performance Evaluation of the Evans and Sutherland ES-1 Computer,"* CS Dept Technical Report CU-CS-506-90, University of Colorado, Boulder, 1990.

[7]. O. McBryan, *"A Comparison of the Intel iPSC860 and SUPRENUM-1 Parallel Computers,"* University of Colorado Tech. Report CU-CS-499-90 and Supercomputer, vol. 41, no. 1, pp. 6-17, 1991.

[8]. O. McBryan, *"Software Issues at the User Interface,"* in Frontiers of Supercomputing II: A National Reassessment, ed. W.L. Thompson, University of Colorado CS Dept. Tech Report CU-CS-527-91 and MIT Press, 1992, to appear.

[9]. O. McBryan, "*Optimization of Connection Machine Performance*", International Journal of High Speed Computing, vol. 2, no. 1, pp. 23-48, 1990.

[10]. O. McBryan, *"Scaling Behavior of the SUPRENUM-1 Parallel Computer"* University of Colorado Tech. Report CU-CS-637-92 and Supercomputer, to appear.

[11]. O. McBryan, *"Performance of the Shallow Water Equations on the CM-200 and CM-5 Parallel Supercomputers"*, University of Colorado Techical Report CU-CS-634-92, and Proceedings of the Fifth Workshop of the European Centre for Medium-Range Weather Forecasts on *"Uses of Parallel Processors in Meteorology"*, Nov. 23-27, 1992, to appear.