Current Developments in Parallel Computation

Oliver A. McBryan

CU-CS-433-89 April 1989

Department of Computer Science
Campus Box 430
University of Colorado
Boulder, Colorado 80309

Current Developments in Parallel Computation[†]

Oliver A. McBryan [*]

Director,
Center for Applied Parallel Processing,
Dept. of Computer Science,
University of Colorado,
Boulder, CO 80309-0430.

ABSTRACT

Recently highly parallel machines have taken the lead as the fastest supercomputers, a trend that is likely to accelerate in the future. We describe recent developments in parallel computers and in associated system software. We describe some of these new computers in more detail, focusing on their novel aspects. Finally we present comparisons of conventional supercomputer performance, as exemplified by the CRAY-XMP, to that on a massively parallel computer, in areas as diverse as PDE solution, oil reservoir simulation and atmospheric modeling.

† To appear in GMD-Spiegel, April 1989.
* Research supported in part by DOE contract DE-ACO2-76ER03077, by NSF grant DMS-8619856 and by NSF Cooperative Agreement DCR-84200944.

1. INTRODUCTION

Supercomputers are the key to the simulation of a wide range of important physical problems. Such simulations typically require large numbers of degrees of freedom to provide sufficient resolution, particularly when engineering accuracy, rather than simple qualitative behavior, is required. In many cases one is currently limited by available computer resources, rather than by an understanding of the underlying physics.

A prime example of the needs for massive computer simulation is exhibited by the case of weather and climate forecasting. Global weather models represent the atmosphere as a three dimensional grid laid out along latitude, longitude and vertical directions on the sphere. The most detailed weather simulations performed to date use about 200 grid points in each horizontal coordinate direction and perhaps 20 in the vertical. Such resolution is woefully inadequate. To place it in context, note that with such a resolution the FRG would be represented by perhaps a single grid-point, allowing for no variation whatever in the temperature and pressure from Hamburg to Munich. Similarly the State of Colorado would be represented by a single point, completely ignoring the presence of over seven hundred 4,000 meter mountains within the state.

Even more challenging than the weather modeling problem is the prediction of flow in the ocean. The importance of oceanographic simulation is borne out by the wide-spread disruption caused by the return in 1986 of El Nino, a cyclical tropical current in the Pacific Ocean. El Nino is believed to have dominated world weather conditions for close to two years before it finally decayed. The primary difficulty with ocean modeling is caused by the enormous range of scales required to effectively simulate an ocean. Ocean eddies may range in scale from the diameter of oceans to centimeters or less. While the solution undoubtedly lies in the direction of better turbulence models to describe the flow, the use of massive supercomputer simulations may well be essential to discovering such better models.

As another example, it is very desirable to simulate accurately the flow of air over a plane. Current aircraft design strategy involves the use of wind tunnels. However wind tunnel testing is limited with respect to aircraft size and Mach number, although extrapolations from smaller scale models can overcome some of the limitations. Planned wind tunnel testing for the Boeing 7J7 was greatly reduced thanks to advances in computational aerodynamics, substantially curtailing 7J7 development time and, consequently, costs. But the computational techniques now in use do not simulate the complete physics for the flow past the entire aircraft; they model various aspects of the flow that, when combined, give guidance to the design, but not answers. The major limitation is that as more of the plane is included in the simulation, the numerical grids become larger, requiring more processing power and memory.

The same phenomenon is seen in oil reservoir simulation, in combustion studies, and wherever quantitative computations in three dimensions are performed. A non-inclusive list of application areas that would benefit substantially from increased computer power is given in table 1.

While there are no absolute standards for comparing computer performance, the concept of peak floating point performance is one widely used criterion. This performance measure is generally described by the number of floating point operations per second ("flops") that the computer can deliver. For this purpose additions and multiplications are usually treated equally (as well as subtraction or division), whereas operations such as copying floating point data, or initializing data are ignored. A rate of one gigaflops (abbreviated Gflops) represents a processing power of one billion floating point operations per second. We will also use the names megaflops (or Mflops) for a million operations per second, and teraflops (or Tflops) for a trillion operations per second. Several current supercomputers have passed

Table 1: Major Supercomputer Applications
Weather Modeling
Global Change and the "Greenhouse Effect"
Aerodynamic Design
VLSI Design
Automobile Design
High Temperature Superconductivity
Pharmaceutical Industry
Molecular Modeling and "Designer Chemicals"
The Human Genome project
Computer Vision and Image Processing
Seismic Processing
Oil Reservoir Simulation
Stability of Large Structures
Quantum Chromo-Dynamics
Transport Phenomena
Linear and Non-linear Optimization
Cryptography

the one Gflops barrier. Major advances in many of the listed application areas are expected as soon as computer power increases to about 100 Gflops, and especially if one can reach the Tflops range. This would correspond to an increase of close to an order of magnitude in resolution in each of the coordinate directions of typical fluid simulations, compared to current vector machine capabilities.

Conventional supercomputers with one or a few processors are limited by various factors, including the need to dissipate energy in a small volume, effects of the finite speed of light, and bottlenecks related to memory access. It is widely believed that parallel computers provide the only hope of reaching this range of computer power in the near future. Furthermore, in most applications the cost per megaflop is a relevant issue. Massively parallel computers provide economies of scale not available to conventional computers larger than a personal computer. Parallel computers may in addition be built from lower cost technologies, because the individual processors need not be particularly powerful.

Because of these factors, parallel computers have been widely studied in recent years. Substantial research has been accomplished related to these machines, including both theoretical advances, involving algorithm design, and computational experiments. Hardware advances have reached the point where the fastest available supercomputers are now highly parallel machines[1], as we demonstrate in section 4. Furthermore, the combined efforts of many researchers have demonstrated that parallel computing is feasible.

The one great disadvantage of a parallel computer, is that it is much harder to program than a serial machine. Each processor must be assigned a distinct component of the work to be performed, and substantial synchronization of the processors is then required in order to ensure that the results from individual processors are merged appropriately. The difficulties of programming parallel machines have spawned a whole range of new research areas for computer science and are a primary reason why this area has been so dynamic in recent years.

In section 2 we give an overview of the architectural and software approaches used by current parallel computers. In section 3 we review in more detail some of the specific parallel architectures that are currently available or are that under development. For further details on several of these architectures, and for examples of applications such as partial differential equation solution on these machines, we refer to our papers[1-7]. In Section 4 we provide examples of the extraordinary performance attainable with current parallel machines applied to both model problems such as PDE solution, as well as to more complete physical simulations.

2. OVERVIEW OF PARALLEL SYSTEMS

2.1. Classification of Parallel Computers

Parallel computers may be broadly categorized in two types - SIMD or MIMD[8]. 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 an 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.

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 inter-connection networks. For a taxonomy of current designs, see the paper of Schwartz[9].

While many interesting parallel machines involve only a few processors, we will concentrate in this paper on those machines which have moderate to large numbers of processors. Important classes of machines such as the CRAY X-MP, CRAY-2 and ETA-10 are therefore omitted from the subsequent discussions.

2.2. A Partial List of Multi-processors

There are at least 50 to 100 parallel computer projects underway at this time worldwide. While some of these projects are unlikely to lead to practical machines, a substantial number will probably lead to useful prototypes. In addition, several commercial parallel computers have been or are already in production (e.g., ICL DAP, Denelcor HEP, Intel iPSC, NCUBE, FPS T-Series, Connection Machine, Symult, Meiko, Parsatec) and more are under development. One should also remember that the latest CRAY computers, (e.g. CRAY X-MP and CRAY-2) involve multiple processors, and other vector computer manufacturers such as ETA Systems, NEC, Fujitsu and Hitachi have similar strategies.

Table 2 lists a selection of the parallel computers under development. This is just a sample of the projects mentioned above, but covers a wide range of different architectures chosen more or less at random. 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

Table 2: Some Parallel Computer Projects

ICL DAP	Caltech Hyper-Cube
Intel iPSC hypercube	NCUBE hypercube
Denelcor HEP-1	NYU/IBM Ultra-computer/RP3
Connection Machine CM-2	FPS T-Series
CRAY X-MP and CRAY-2	ETA-10
IBM 3096	Multiflow
Goodyear MPP	MIT Data-flow Machines
BBN Butterfly	Wisconsin Database Machine
SUPRENUM-1	IBM GF-11 and TF1
Paralex Pegasus	Symult 2010
Myrias SPS-2	Cedar Project
Meiko	Parsetec
Evans & Sutherland ES-1	CMU iWarp
Flex	Alliant FX-8
Sequent Balance	Encore Multimax
CCI Navier-Stokes Machine	TERA

this broad array by discussing some of the possibilities encountered for both node and communication facilities.

2.3. Node Design

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 there is a usually a substantial body of low-level software available for such processors - software such as compilers, assemblers and debuggers. Thus we find that an enormous number of the current parallel computer products are based on one or more of the Intel 80386, Motorola 68020, INMOS T800 transputer and the Weitek floating point accelerators. Typically one of these microprocessors will be combined on a board with a floating point co-processor (e.g. 80387 or 68881), possibly a Weitek processor and several megabytes of memory. Memory consumes substantial space, and current systems have in the range of 1 to 20 MBytes per node. 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 and Navier-Stokes machines as examples. The iWarp is of particular interest here on account of the high level of integration used in the design of the custom processing element.

2.4. Communication Features

The range of inter-processor communication facilities is what really characterizes the differences in architecture among the 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.

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.

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 that 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. Table 3 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).

Table 3: Properties of Interconnection Networks

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	$2\log P$
Hypercube	$.5P\log P$	$\log P$	$\log P$

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.

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 hierarchial 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. New configurations of processors continue to be proposed. Of particular interest are Giloi and Montenegro's TICNET architecture[10], and Faber's vertex-symmetric minimal path networks[11].

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, and 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 bit-serial 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, iWarp and the Symult among others. In the case of the Symult, the designers were so confident of the advantages of worm-hole routing that they abandoned a hypercube architecture from their first generation in favor of a simple two-dimensional rectangular grid.

2.5. 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 have no concept of parallelism or of communication capability. Typical examples are the systems supplied by Intel, Symult 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 communications operations. The resulting software is best described as "programming in communication assembly language".

A few manufacturers have gone beyond this step by providing language extensions that capture aspects of the parallel hardware. Thinking Machines provides a parallel Fortran for their Connection Machine CM-2 computer. The compiler supports the Fortran 8X array extensions to Fortran 77, and the convention is that objects declared as arrays are understood to be distributed across the parallel processors. Communication among processors is supported by the 8X 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 hardware which maps extremely well onto array operations.

Myrias Corporation and Evans and Sutherland 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 who 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.

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 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 is supported by the majority of systems in one form or another. Examples include iPSC, SUPRENUM, Symult and CM-2.

3. SOME REPRESENTATIVE PARALLEL SYSTEMS

In this section we will look briefly at the characteristics of a number of these machines. The machines currently under development have processor numbers ranging between 2 and 65,536. The machines listed above vary greatly in local processing power, ranging from a few megaflops up to 20 Gflops.

Evans and Sutherland ES-1

The ES-1 is a new (1989) parallel architecture based on a hierarchial crossbar structure. The basic "processor" is a complex package consisting of a cluster of 16 processing elements called computational units, along with 256 Mbytes of shared memory. Each processor supports an I/O subsystem with a bandwith of 160 Mbytes/sec over eight full duplex channels. Up to 8 processors and 8 I/O subsystems are currently supported. As mentioned previously, the ES-1 supports a virtual memory address space of 32 bits, which greatly simplifies program development for the system.

The processors and I/O subsystems are connected together by a crossbar. Each node (computational unit) is a 20 MIPS 10 MFLOPS (64-bit precision) scalar processor with six pipelined functional units. The 16 nodes in a processor are connected by a 1 Gbyte/sec interconnect crossbar. The memory is 64-way interleaved and supports a bandwidth of 640 Mbytes/sec. The ES-1 operating system is a full UNIX system and runs in every computational unit. Unlike most distributed processors, there is no front end processor. Since each cluster delivers 160 Mflops double precision, the most powerful current system delivers a peak rate of 1280 Mflops scalar and supports 2 Gbytes of memory.

CMU iWarp

The iWarp computer[12] is a follow-on to the 100 Mflop Warp processor 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 general purpose distributed memory computers, or special purpose systolic arrays. The initial iWarp is an 8×8 array of processors delivering 1.2 Gflops, and expected to be available in 1990, but there are plans to extend this up to 1,024 processors.

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.

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.

Connection Machine

The Connection Machine CM-1 designed by Thinking Machines, Inc., of Cambridge, MA, has 65,536 1-bit processors, though this may be regarded as a prototype for a machine that might have 1,000,000 processors. While designed primarily for artificial intelligence work, this machine has proved to have even greater potential applications to scientific computing applications[1,5]. The more recent CM-2 computer adds 2,048 Weitek floating point processors and 512 Mbytes of memory, to provide a powerful computer for numerical as well as symbolic computing. The CM 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 machines are based on a hypercube communication 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 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.

Myrias

The Myrias SPS-2 computer, built by Myrias Research Corp. of Edmonton, Alberta, has up to 1024 processing elements. The architecture is a hierarchial bus design, utilizing 33 Mbytes/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 hierarchial 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 Fortran support parallel do and a join operation. The Par Do model used by Myrias is somewhat unusual in that there are no possibilities for sharing data. A Par Do 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 Par Do'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 on 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 unpredicatable 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.

The programming approach to the system is thus functional in nature and is based on parallel recursion. As an example, we describe an implementation of a global vector sum, assuming the vector is initially of length N. The parent creates 2 tasks, each intended to sum half of the elements, storing the results in variables *left* and *right.* Each child task similarly spawns two more children, storing results into 2 local variables of the principal child, and so on. On completion of a pair of child tasks, its parent has the two partial sums in separate variables, and therefore it proceeds to add these, providing its value to the next parent above. In this way the result is obtained at the global parent in time $log(N)$without ever sharing memory. All assignment of work to processors is handled automatically.

SUPRENUM

The German SUPRENUM-1 project involves coupling up to 16 processor clusters with a network of 200 Mbyte 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 16 Mflops of computing power and support 8 Mbytes of memory. The very high speed 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. A prototype cluster containing 16 nodes is already in operation, and a full machine with up to 16 clusters will be available in late 1989.

SUPRENUM is characterized by the best support for scientific applications to be found among the various vendors. The effort invested in development of 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 in terms of low level primitives on any distributed system.

Intel iPSC

The Intel iPSC was the first commercial hypercube computer, and has been the most widely available highly parallel computer in recent years. Built from 128 Intel 80286 processors, peak computer power was under 10 Mflops, yet the iPSC was the basis for a large number of useful experiments in parallel computing. The recently developed iPSC/2 computer is a second generation machine that provides greatly increased processing power and communication throughput. Each node contains an 80386 microprocessor with up to 8 Mbytes of memory (extendible to 16Mbytes with 64 processors). There are three available numeric co-processors: an Intel 80387 co-processor (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 now implements worm-hole routing rather than the store and forward protocol of previous generations.

Symult

The Symult 2010 is a new commercial machine based on a grid architecture. Individual nodes consist of a Motorola 68020 (25 MHz) with a 68881 co-processor (150 Kflops). An optional upgrade to the 68882 processor (215 Kflops) is possible. A further option provides for a vector processing board based on Weitek chips, with a 20 Mflops performance. Peak performance of a 1024 node system may be as high as 20 Gflops. Memory per node ranges from 2 to 10 Mbytes. Maximum node memory is 8 Mbytes, with a further 10 Mbytes of memory on the vector board. The most interesting feature of the machine is the message routing system which establishes point-to-point communications between remote nodes. Each node has a routing device that can support simultaneous transmission on four links at 20Mbytes each, without interruption of intermediate computational processors. Communication is by "worm-hole" connectivity rather than the usual store-and-forward, resulting in far greater performance for long-range communication. In this communication mode, a connection circuit is first established between remote nodes, incurring a small startup cost, after which the message is transferred in bit-serial fashion in a single operation. Worm-hole communication provides extremely fast long-distance communication, whereas a standard store and forward model would incur large overheads due to the long path-lengths on a grid.

AMT DAP

The DAP was the first massively parallel single-bit computer, and has been widely used for a range of scientific applications. 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 32×32 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.

Paralex Gemini and Pegasus

Paralex Research Inc. is developing a line of highly parallel local memory systems in the supercomputer class. The initial Gemini product supports up to 1000 nodes with peak performance up to about 2 Gips and 500 Mflops. The Gemini uses a hypercube to provide connectivity, and also features a high performance UNIX front end. The second generation Pegasus machine, due in 1989, will support 512 nodes with 8 Gbytes of memory and will provide 25 Gips and 15 Gflops peak rate. This system will be based on the new SPARC technology being licensed by SUN Microsystems. The follow-on Genesis system, planned for 1990, will provide up to 2 Tflops (teraflops) of performance.

GF-11 and TF1

The GF-11 is an IBM parallel computer, designed to perform very specific scientific computations at Gflop rates. The GF-11 has 576 processors (including 64 backup processors), coupled through a three stage Benes network which can be reconfigured at every cycle in 1024 different ways by an IBM 3084 control processor. Peak processing power of 11 Gflops will allow previously uncharted computational regimes to be explored. The machine has been designed primarily for solving quantum field theory programs and is not a general purpose computer; in particular, very little software is available. It is an SIMD architecture but with some flexibility in that the settings of local registers may be used to control the behavior of individual processors.

CCI Navier-Stokes Machine

The NASA sponsored Navier-Stokes Machine being built at Princeton University involves an experiment with reconfigurable pipelines as well as parallelism. Up to 64 processors are supported with hypercube connections. Each node consists of a CPU, 32 arithmetic processing units and 2 Gbytes of memory. Each of the arithmetic units may be specified to be an adder, multiplier, etc., and connections can then be specified between them in order to represent efficiently a pipeline to evaluate an expression. Reconfiguring the connections takes only 50 nano-seconds. Since each arithmetic unit has a peak processing power of 20 Mflops, the combined processing power per node is 640 Mflops. CCI Corporation plans to market a commercial version of the Navier-Stokes Machine.

HEP and TERA

The Denelcor HEP was the first commercial parallel computer. The HEP featured a shared memory, with special *access* bits to provide for memory locking on every word. The processors were pipelined units, each capable of executing a large number of separate instruction streams simultaneously. Each processor was rated at 10 Mips. The new TERA computer, designed by HEP creator Burton Smith, will support 256 processors, each similar in many respects to the HEP, and will provide up to 256 Gflops of computational power in a shared memory, scalar processing environment.

Other Approaches

A variety of other important architectures are also under development. These include various dataflow machines (with bus, tree and grid structures), examples include the MIT Tagged Token machine, the NTT Dataflow grid machine and the Manchester Dataflow Machine. Another important class are the tree-structured machines (binary trees, trees with sibling or perfect shuffle connections), examples of which are the Columbia University DADO machine and the CMU Tree Machine. Because of the simplicity of the connections, nearest neighbor machines, such as the MPP, and ring architectures, such as the University of Maryland's ZMOB (256 processors on a ring), are also popular designs.

4. PARALLEL PERFORMANCE ON SCIENTIFIC APPLICATIONS

In the previous sections we have given an overview of capabilities of some of the currently available parallel systems. In this section we turn to the question of what kind of performance one can expect from these systems. In order to maintain coherence we will discuss performance on a single parallel computer, the Connection Machine CM-2, and compare where appropriate with the CRAY-XMP or CRAY-2 computers. The CM-2 is the computer with the highest peak processing power (20 Gflops), and consistently delivers in the range of 1 to 3 Gflops on applications. Thus it sets a performance standard against which other potential supercomputers may be compared.

4.1. Solution of Elliptic Partial Differential Equations

Discretization of elliptic partial differential equations such as the equation

$$\vec{\nabla}\cdot\overleftrightarrow{k}(\vec{r})\vec{\nabla}u = f(\vec{r}) \quad ,$$

by finite element or finite difference methods, leads to systems of linear equations of the form $Ax = y$ with sparse coefficient matrices. The fill-in of the matrix tends to follow diagonals and the bandwidth is about $dN^{1/2}$ or $dN^{2/3}$, for two or three dimensional space respectively, where N is the dimension of the matrix and d is the degree of the finite elements used for the discretization. Furthermore, typically only $O(1)$ diagonals have nonzero elements. We have developed a parallel preconditioned conjugate gradient algorithm[13-15] on the Connection Machine to solve systems of equations with such coefficient matrix structures. A *preconditioning* operator can be effective in improving substantially the convergence rate of the algorithm[16].

We parallelize the algorithm by exploiting parallelism in every operation of the iteration. All of the vectors in the algorithm are allocated as CM parallel variables (pvars). For our Poisson-like test problem with a 5-point discretization on a rectangle, the operation $x \rightarrow Ax$ is easily written using the NEWS grid addressing modes of the CM. For simplicity we have chosen the pre-conditioning operator B to be the diagonal of the operator A. The other communication intensive operations in the conjugate gradient algorithm are the several inner products of vectors which are required. These inner products perform at very high speeds on the CM by taking advantage of the hypercube structure to evaluate the global sum. For full details on the implementation, we refer to our paper[5].

The performance of this algorithm for a two-dimensional PDE discretized with a five-point formula on a 65,536 processor CM-2 is presented in Table 4 where we have given results for solution of equations on grids up to size 4096×4096. These measurements were made with a simple diagonal scaling pre-conditioner. As can be seen, the highest performance is attained with the largest grid size, which corresponds to the highest virtual processor ratio.

Table 4: Performance of Conjugate Gradient on CM-2

Grid Size	Mflops
512×512	1412
1024×1024	2357
2048×2048	3123
4096×4096	3812

We have also developed a fast-solver for the Poisson equation which uses an FFT algorithm developed by Thinking Machines Corporation and supplied in the CM-2 mathematics library. The fast solver runs at 1.1 Gflops, and has been used as a preconditioner for the conjugate gradient solver described above. This results in far fewer iterations on fine grids, although the overall performance of the code in terms of Mflops is then only 1.2 Gflops. However these are substantially more "useful" flops than in the diagonally preconditioned case.

4.2. An example from Oil Reservoir Simulation

We have ported substantial segments of an oil reservoir simulator to the Connection Machine. In particular the slowest part, the solver for a set of non-symmetric linear equations, is currently running at 7 times its performance on a CRAY-XMP. Remarkably higher speedups over a CRAY are found in other parts of the code, and we illustrate this with the following example of a code fragment from the simulator:

```
          do 100 j = 1,nc
            do 970 kz = 1,nz
              do 971 jy = 1,ny
                do 972 ix = 1,nx
                 tden(ix,jy,kz,j) = avmw(ix,jy,kz,j)/gvolph(ix,jy,kz,j)
                 dumr(ix,jy,kz) = grk(ix,jy,kz,j)/
     &                         gvis(ix,jy,kz,j)/gvolph(ix,jy,kz,j)
972              continue
971            continue
970          continue
```

In the table below we present the performance of this code on a CRAY 2 processor, and on the CM-2. The CRAY-2 performance is better than the CRAY-XMP performance the oil company in question was attaining, and the grid sizes shown for the CRAY-2 represent the limits of the capability of the XMP to handle the overall problem in memory.

Machine	Processors	nx	ny	nz	nc	Performance
CRAY 2	1	16	16	32	2	18 Mflops
CM-2	65536	64	64	64	2	741 Mflops

4.3. Atmospheric/Oceanographic Simulation

As another example of the current capabilities of massively parallel architectures, we describe the implementation of a standard two-dimensional atmospheric model - the Shallow Water Equations - on the Connection Machine. These equations provide a primitive but useful model of the dynamics of the atmosphere or of certain ocean systems. Because the model is simple, yet captures features typical of more complex codes, the model is frequently used in the atmospheric sciences community to benchmark computers[17]. Furthermore, the model has been extensively analyzed mathematically and numerically[18, 19]. We[*] have recently implemented the shallow water equations model on the Connection Machine, and compared the performance there with the CRAY-XMP We have used both explicit[7] and

* Joint work with R. Sato and P. Swarztrauber of the National Center for Atmospheric Research.

spectral[20] solution methods for the equations.

The shallow water equations, without a Coriolis force term, take the form

$$\frac{\partial u}{\partial t} - \zeta v + \frac{\partial H}{\partial x} = 0 ,$$
$$\frac{\partial v}{\partial t} - \zeta u + \frac{\partial H}{\partial y} = 0 ,$$
$$\frac{\partial P}{\partial t} + \frac{\partial Pu}{\partial x} + \frac{\partial Pv}{\partial y} = 0 ,$$

where u and v are the velocity components in the x and y directions, P is pressure, ζ is the vorticity: $\zeta = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$ and H, related to the height field, is given by: $H = P + (u^2 + v^2)/2$. It is required to solve these equations in a rectangle $a \le x \le b$, $c \le y \le d$. Periodic boundary conditions are imposed on u, v, and P, each of which satisfies $f(x+b,y) = f(x+a,y)$, $f(x,y+d) = f(x,y+c)$.

We have discretized the above equations on a rectangular grid with periodic boundary conditions. We time difference using the Leap-frog method. We then apply a time filter to avoid weak instabilities inherent in the leap-frog scheme:

$$F^{(n)} = f^{(n)} + \alpha\,(f^{(n+1)} - 2f^{(n)} + f^{(n-1)}) ,$$

where α is a filtering parameter. The filtered values of the variables at the previous time-step are used in computing new values at the next time-step. For a complete description of the discretization we refer to[17].

In the case of the spectral version, the physical variables are Fourier transformed to frequency space, and all spatial derivatives are computed in that space. An inverse Fourier transform is applied at each time to recover the derivatives in grid space in order to compute the time derivatives and the propagation.

In table 5 we list the results of representative performance of both the explicit and spectral codes on the CRAY-XMP and CM-2 computers. A four processor result for the XMP spectral code is not yet available as some significant work is required to multi-task the specialized assembly language FFT routines used on the XMP.

Table 5: Performance of Shallow Water Equations

Machine	Processors	Algorithm	Grid Size	Performance
CRAY-XMP	1	Explicit	256×256	148 Mflops
CRAY-XMP	4	Explicit	512×512	560 Mflops
CM-2	65536	Explicit	2048×2048	1714 Mflops
CRAY-XMP	1	Spectral	500×500	122 Mflops
CM-2	65536	Spectral	2048×2048	1167 Mflops

In each case we have solved the largest grid size that would fit in memory. The CRAY spectral code performed poorly on grids that were a multiple of 64 due to memory bank conflicts. The 500×500 performance was the best observed from among a large range of power-of-two and other grids. The Mflops being performed are essentially equivalent in "usefulness". This was measured by computing the actual processing time per grid-point, which is in fact a better measure of performance than Mflops. This quantity behaved in the same way as the Mflops, apart from an expected logarithmic term due to the inherent logarithmic time dependence of the FFT.

References

1. O. McBryan, ''Solving PDE at 3.8 Gigaflops,'' University of Colorado CS Dept Preprint, Sept 1987.
2. O. McBryan and E. Van de Velde, ''Parallel Algorithms for Elliptic Equations,'' *Commun. Pure and Appl. Math.*, vol. 38, pp. 769-795, 1985.
3. O. McBryan and E. Van de Velde, ''The Multigrid Method on Parallel Computers,'' in *Proceedings of 2nd European Multigrid Conference, Cologne, Oct. 1985*, ed. J. Linden, GMD Studie Nr. 110, GMD, July 1986.
4. O. McBryan and E. Van de Velde, *Hypercube Algorithms and Implementations*, SIAM J. Sci. Stat. Comput., 8, pp. 227-287, 1987.
5. O. McBryan, ''The Connection Machine: PDE Solution on 65536 Processors,'' *Parallel Computing*, vol. 9, pp. 1-24, North-Holland, 1988.
6. P. O. Frederickson and O. McBryan, ''Parallel Superconvergent Multigrid,'' in *Multigrid Methods: Theory, Applications and Supercomputing*, ed. S. McCormick, Math Applications Series, vol. 110, pp. 195-210, Marcel-Dekker Inc., New York, 1988.
7. O. McBryan, ''New Architectures: Performance Highlights and New Algorithms,'' *Parallel Computing*, vol. 7, pp. 477-499, North-Holland, 1988.
8. M. J. Flynn, ''Very high-speed computing,'' *Proc. IEEE*, vol. 54, pp. 1901-1909, 1966.
9. J. Schwartz, ''A Taxonomic Table of Parallel Computers, Based on 55 Designs,'' Ultracomputer Note #69, Courant Institute, New York, 1983.
10. W.K. Giloi and S. Montenegro, ''Super Interconnection Networks for Super Computers,'' GMD Technical Report, Berlin, 1988.
11. V. Faber and J. Moore, ''High-degree Low-diameter Interconnection Networks with Vertex Symmetry: The Directed Case,'' Los Alamos Technical Report LA-UR-88-1051, March 1988.
12. S. Borkar, R. Cohn, G. Cox, S. Gleason, T. Gross, H.T. Kung, M. Lam, B. Moore, C. Peterson, J. Pieper, L. Rankin, P. Tseng, J. Sutton, J. Urbanski, and J. Webb, ''iWarp: An Integrated Solution to High Speed Parallel Computing,'' *Proceedings of Supercomputing '88 Conference*, pp. 330-339, Orlando, Florida, Nov 1988.
13. C. Lanczos, ''An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators,'' *J. Res. Nat. Bur. Standards*, vol. 45, pp. 255-282, 1950.
14. M. R. Hestenes and E. Stiefel, ''Methods of conjugate gradients for solving linear systems,'' *J. Res. Nat. Bur. Standards*, vol. 49, pp. 409-436, 1952.

15. J. K. Reid, ''On the method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations,'' in *Large Sparse Sets of Linear Equations*, ed. J. K. Reid, pp. 231-54, Academic Press, New York, 1971.

16. M. Engeli, Th. Ginsburg, H. Rutishauser, and E. Stiefel, *Refined Iterative Methods for Computation of the Solution and the Eigenvalues of Self-Adjoint Boundary Value Problems,* Birkhauser Verlag, Basel/Stuttgart, 1959.

17. G.-R. Hoffman, P.N. Swarztrauber, and R.A. Sweet, ''Aspects of using multiprocessors for meteorological modeling,'' in *Multiprocessing in Meteorological Models*, ed. D. Snelling, pp. 126-195, Springer-Verlag, Berlin, 1988.

18. R. Sadourny, ''The dynamics of finite difference models of the shallow water equations,'' *JAS*, vol. 32, pp. 680-689, 1975.

19. G.L. Browning and H.-O. Kreiss, ''Reduced systems for the shallow water equations,'' *JAS*, to appear.

20. O. McBryan, ''Connection Machine Application Performance,'' in *Scientific Applications of the Connection Machine, Proceedings of the NASA-Ames Conference on Massively Parallel Computing*, ed. Horst Simon, 1989.