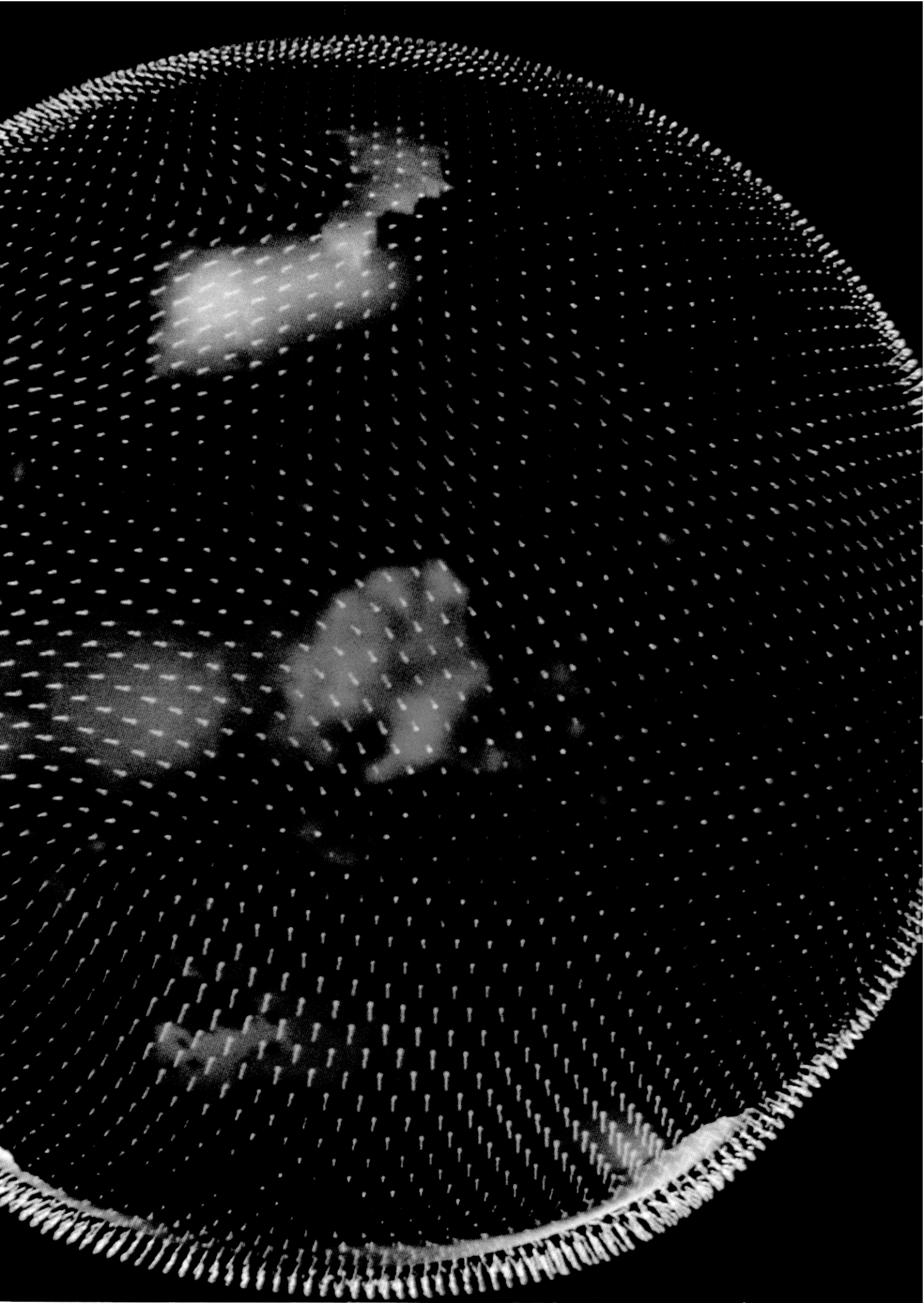


SUPERCOMPUTING

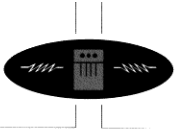
W. Daniel Hillis and Lewis W. Tucker

THE CM-5 CONNECTION MACHINE: A SCALABLE SUPERCOMPUTER

The CM-5 Connection Machine is a scalable homogeneous multiprocessor designed for large-scale scientific and business applications. In this article we describe its architecture and implementation from the standpoint of the programmer or user of parallel machines. In particular, we emphasize three features of the Connection Machine architecture: scalability, distributed memory/global addressing, and distributed execution/global synchronization. We believe that architectures of this type will replace most other forms of supercomputing in the foreseeable future. • Examples of the current applications of the machine are included, focusing particularly on the machine's ability to support a variety of programming models. The article is intended to be a general overview appropriate to a user or programmer of parallel machines, as opposed to a hardware designer. The latter may prefer more detailed descriptions elsewhere [16, 23, 24].

Architecture vs. Implementation

In describing the CM-5, it is useful to distinguish between issues of architecture and implementation. While there is no hard line between the two, the architectural features of the machine are those features that are intended to remain constant across multiple implementations. In a microprocessor, for example, one of the most important features of the architecture is usually the instruction set. Object-code compatibility from one generation of the machine to the next is an important issue for many microprocessor applications. Supercomputer applications, on the other hand, are usually written in higher-level languages, so the instruction set is not a necessary part of the architectural specification of the CM-5. The architecture is defined by other issues, such as the addressability of memory, the *user-visible* mechanisms of synchronization, and the function-



ality of the communication and I/O systems.

We will describe both the architecture of the CM-5 and its current implementation. Figure 1, for example, is the architectural block diagram of the machine that is unlikely to change. The specifications shown in Table 1, on the other hand, are descriptions of the current implementation, which are likely to change from year to year as new versions of the machine are introduced and faster, denser chips become available.

Architectural Overview

The CM-5 is a coordinated homogeneous array of RISC microprocessors. We note that the acronym for this description is CHARM. CHARM architectures take advantage of high-volume microprocessors and memory components to build high-performance supercomputers that are capable of supporting a wide range of applications and programming styles. By CHARM we mean any homogeneous collection of RISC microprocessors that has the following coordination mechanisms:

1. A low-latency, high-bandwidth communications mechanism that allows each processor to access data stored in other processors
2. A fast global synchronization mechanism that allows the entire machine, including the network, to be brought to a defined state at specific points in the course of a computation

These coordinating mechanisms support the efficient execution of sequentially deterministic parallel programs. Both mechanisms are important for supporting the data-parallel model of programming [14]. The CM-5 is a CHARM architecture and we believe most of the massively parallel machines currently on the drawing boards in the U.S., Europe, and Japan are also CHARMS.

The CM-5 consists of a large number of processing nodes connected by a pair of networks, implementing the coordination functions described earlier. The number of processors may vary from a few dozen to tens of thousands. The number of processors in the machines currently installed ranges from 32 to 1,024. The networks are similar to the backplanes of conventional computers in that the machine's I/O and processing can be expanded by plugging more modules into the networks. Unlike a bus in a conventional computer, the bandwidths of the networks increase in proportion to the number of processors (see Figure 1). The memory of the machine is locally distributed across the machine for fast local access. Each processor accesses data stored in remote memories of the processors via the network.

One of the most important features of the CM-5 is its scalability. The current implementation, including networks, clocking, I/O system, and software are designed to scale reliably up to 16,384 processors. The same application software runs all of these machines. Because the machine is constructed from modular components, the number of processors, amount of memory, and I/O capacity can be expanded on-site, without requiring any change in application software.

The Processor Node

Microprocessor technology changes very rapidly and the CM-5 was designed with the assumption that the processing node would therefore change from implementation to implementation. This implies that the system software, including compilers and operating system, be written in a way that is largely processor-independent. It also requires, at the hardware level, that the clocking system of the network is decoupled from that of the processor, so the processor and network can be upgraded independently.

The current implementation of the CM-5 uses a SPARC-based processing node, operating at a 32- or 40MHz clock rate, with 32MB of memory per node. The SPARC is augmented with auxiliary chips that increase floating-point execution rate and memory bandwidth. The result is a 5-chip microprocessor with performance characteristics similar to that of single-chip microprocessors that will be widely available in a few more years. The SPARC was chosen primarily for its wide base of existing software. One drawback in using a standard microprocessor, SPARC included, is that standard RISC processors generally depend on a data cache to increase the effective memory bandwidth. These caching techniques do not perform as well in a large parallel machine, because many applications reference most of the data in memory on each pass of an iteration. This results in relatively little reuse of data cache references, reducing the effectiveness of the cache. In a parallel machine, the absolute uncached bandwidth between the memory and processor is an important parameter for performance.

In the current implementation of the CM-5 we addressed the memory bandwidth problem by adding four floating-point data paths, each with direct parallel access to main memory, to the standard SPARC microprocessor. The four data paths have an aggregate bandwidth to main memory of 500MB/sec per processing node, and 128MFLOPS peak floating-point rate. This is enough bandwidth to allow a single memory reference for each multiply and add when all of the floating-point units are operating at peak rate. A block diagram of the node is shown in Figure 2. The data paths are capable of performing scalar or vector operations on IEEE single- or double-precision floating-point numbers, including 64-bit integers. They also support specialized operations such as bit count. While the use of additional floating-point units with their own paths to memory is an important implementation feature of the current CM-5, it is not generally visible to users at the programming level.

The measured rate in a single node is currently 64MFLOPS for matrix multiply and 100MFLOPS on matrix-vector multiplication. The rate for the 500×500 Linpack benchmark is 50MFLOPS. The sustained rate for the node 8K-point FFTs is 90MFLOPS. In a typical application, since much of the time is spent in interprocessor communications, the peak execution rate of the node is not usually a reliable predic-

tor of average performance on applications. In data parallel applications, where the opportunities for parallelism increase with the size of data, performance increases almost linearly with an increasing number of processors and dataset size. This near-linear scaling is possible because network bandwidth scales in proportion to the number of processing nodes. Benchmarks such as the large matrix Linpack [5], in which the problem size increases with the amount of available memory, as shown in Figure 3, are an example of a problem with near-linear speedup. Measurements on various sizes of CM-5's show Linpack benchmark running with near-linear speedups at rates of 45MFLOPS/node. A 16,000 processor CM-5 is expected to exceed 700GFLOP/sec in performance on this benchmark.

Choosing the right balance between memory bandwidth, communication bandwidth, and floating-point capacity is one of the major design issues of a supercomputer. In the past, when floating point units were very expensive, designers mea-

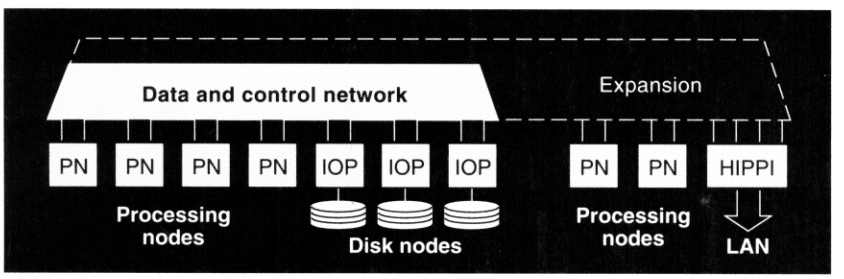


Figure 1.
CM-5 Communications networks
The CM-5 consists of processing, control, and I/O nodes connected by two scalable networks which handle communication of data and control information. Unlike a bus on a conventional computer, the communications network is scalable in the sense that the bandwidth of the network grows in proportion to the number of nodes. Shown is the expansion of the network that results when additional processing and I/O nodes are added.

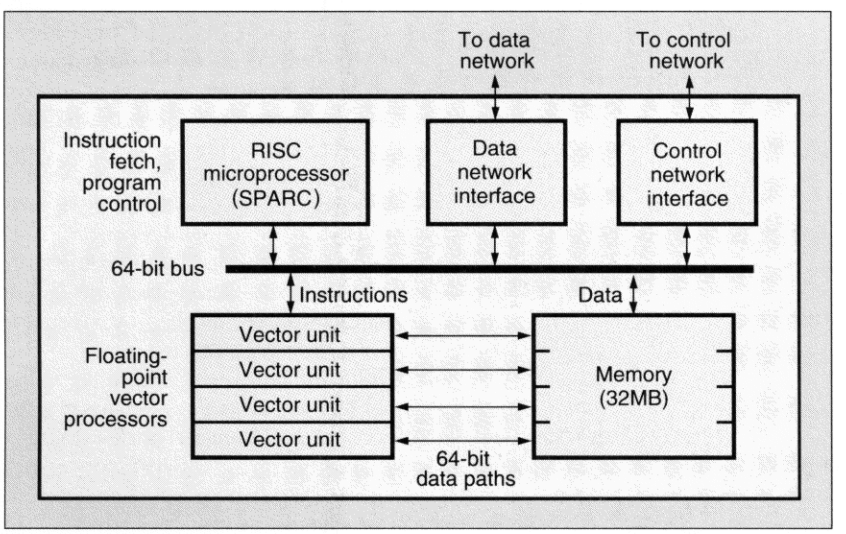


Figure 2.
CM-5 processing node with vector units
Each processing node of the current implementation of the CM-5 consists of a RISC microprocessor, 32MB of memory, and an interface to the control and data interconnection networks. The processor also includes four independent vector units, each with a direct 64-bit path to an 8MB bank of memory. This gives a processing node with a double-precision floating-point rate of 128MFLOPS, and a memory bandwidth of 512MB/sec.

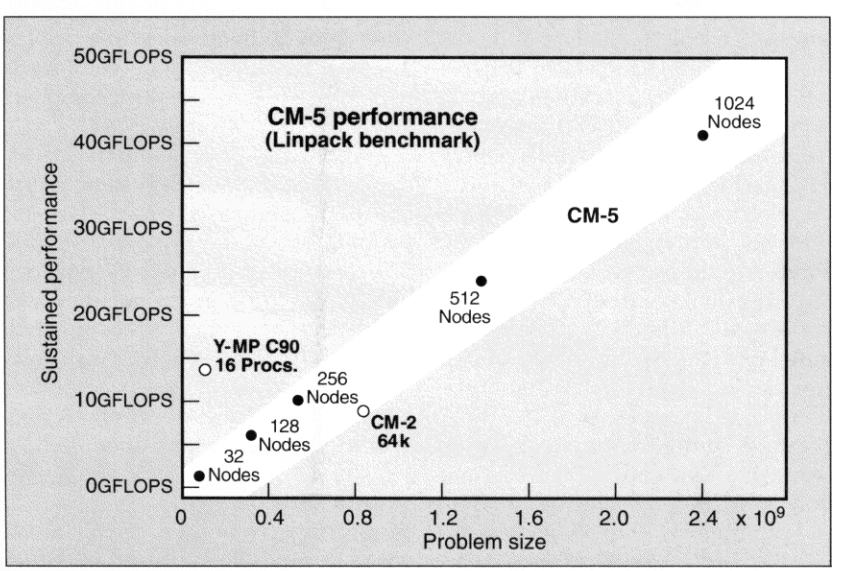
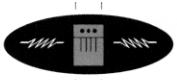


Figure 3.
Scalable performance
The performance of data parallel applications increases with the size of data and available processing resources. Shown is the measured performance of the Linpack benchmark on the CM-5 for increasing problem size and number of processors.



sured the efficiency of their machine by the percentage utilization of the floating-point units in a typical application. This sometimes led to designs that were inefficient at using an even more precious resource, such as memory bandwidth. Since applications vary in their floating-point-to-memory reference ratio, no architecture uses both of these resources at 100% capacity in every application. Today, floating-point units represent a relatively small portion of the total cost, so optimizing the design around their utilization makes little sense. Instead, the design point chosen in the CM-5 was based on the balance point calculated by the equal ratios method [12], which takes the relative cost and performance aspect into account.

Table 1. CM-5 specs

The specifications of the current implementation of the CM-5 are shown here. The largest machine shipped to date is 1,024 processors, but the software networks clocking, power distribution, and so forth are all designed to scale to the 16,000 processor system.

Processors	32	1024	16384
Network address	64	2048	32768
Number of data paths	128	4096	65536
Peak MFLOPS	4GFLOPS	128GFLOPS	2048GFLOPS
Memory	1GB	32GB	512GB
Memory bandwidth	16GB/sec	512GB/sec	8192GB/sec
Bisection bandwidth	320MB/sec	10GB/sec	160GB/sec
I/O bandwidth	320MB/sec	10GB/sec	160GB/sec
Global synch time	1.5 microsecs	3 microsecs	4 microsecs

The Data Network

Like the CM-2, the CM-5 incorporates two important types of communication networks: one for point-to-point communication of data, and a second for broadcast and synchronization. As in the CM-2, the data network is implemented as a packet-switched logarithmic network. The CM-5 uses a fat-tree topology (see Figure 4) [15], which is a more general and flexible structure than the grid and hypercube used in the CM-1 and CM-2. This topology offers better scalability properties than the hypercubes or grids. Specifically, the network expands naturally as new nodes are added, and the capacity of network traffic across the machine (bisection bandwidth) is proportional to the number of nodes in the machine. This is not the case with grid-based networks. The fat-tree topology also has good fault tolerance properties.

The data network uses a packet-switched router which guarantees deadlock-free delivery. The message-routing system uses an adaptive load-balancing algorithm to minimize network congestion. Packets are delivered at a rate of 20MB/sec to nearby nodes. The bisection bandwidth of a 1,024-node CM-5, that is, the minimum bandwidth between one-half of the machine and the other, is 5GB/sec in each direction. Network latencies across a 1k-processor machine are between 3 and 7 microseconds. This bandwidth is achieved on regular or irregular communications patterns. Bounds checking hardware, error checking, end-to-end counting of messages, and rapid context switching of network state protects users from interference in a time-shared, multiple-partition configuration. As discussed later, the data network, in conjunction with the run-time system software, supports either a message-passing or a global address space programming model.

The Control Network

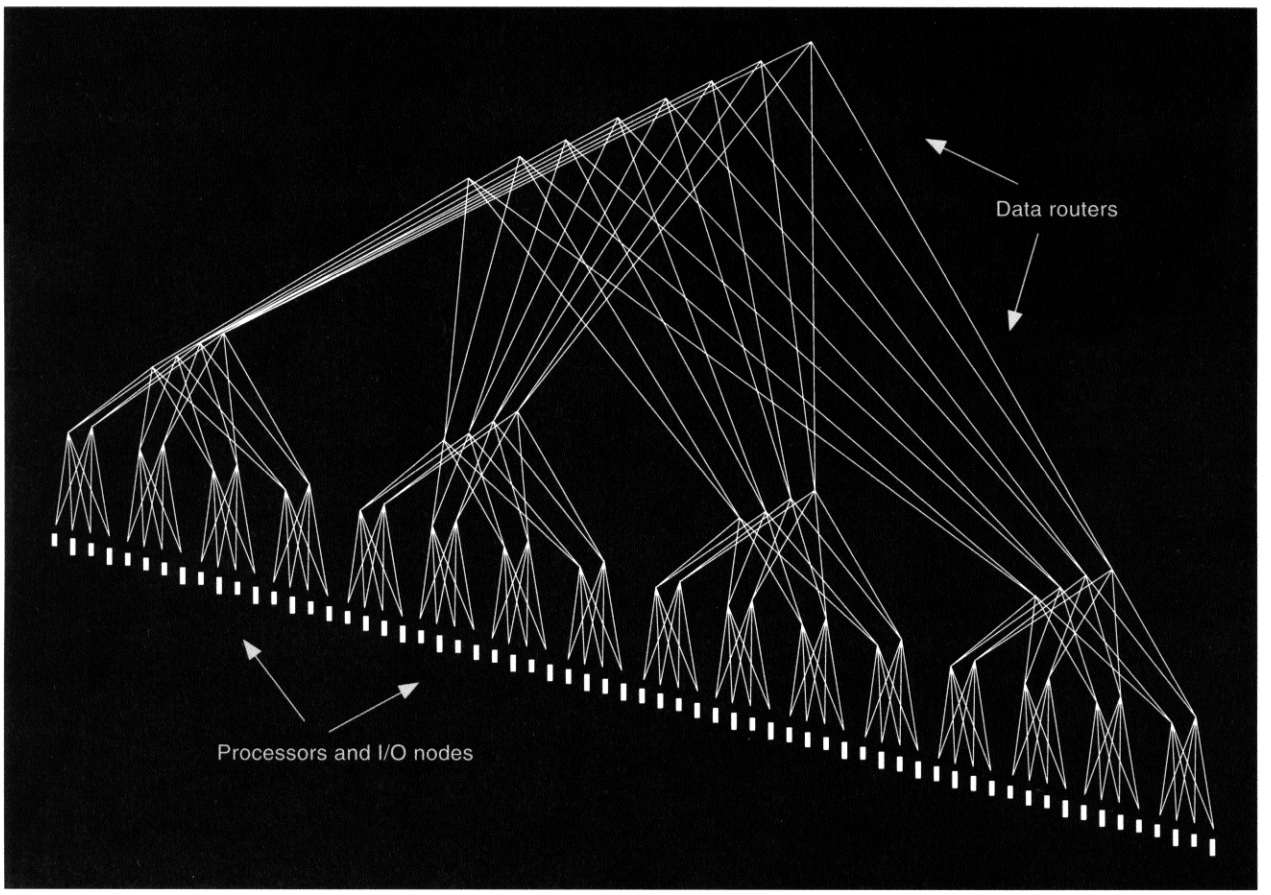
The CM-5's control network supports the broadcast, synchronization, and data reduction operations necessary to coordinate the processors. The network can broadcast information, such as blocks of instructions, to all processors simultaneously or within programmable segments. This control network also implements global reduction operations, such as computing the total of numbers posted by all processors, computing global logical AND, global MIN, and so forth. Also, the control network implements global and segmented reduction and parallel prefix functions that have been shown to be important building blocks for parallel programming [3, 14].

Unlike the CM-2, the operations of the individual CM-5 instructions are not necessarily synchronized. Each processor has its own program counter, and fetches instructions from a local instruction cache. Rapid synchronization, however, is provided by the control network employing a barrier synchronization mechanism. Each processor may elect to wait for all the other processors to reach a defined point in the program before proceeding. This mechanism allows processors to detect whether data network messages are still in transit between processors before establishing synchronization. The details of this mechanism are not normally visible to the application programmer, since the appropriate synchronization commands are typically generated by compilers and run-time libraries.

From the programmer's standpoint, the fast synchronization supports the sequential control flow required by the data parallel model. The automatically inserted synchronization barriers guarantee that each data parallel operation is completed, before the next begins. Such synchronization events occur whenever there are potential dependencies between the two parallel operations.

Input/Output

Many supercomputers spend much of their time accessing and updating



data stored externally to the processor. The most challenging part of input/output is generally the transfer of data to and from disk subsystems. The CM-5 was designed with the assumption that mass storage devices, such as disk arrays, will continue to increase in capacity and bandwidth. One implication is that no fixed I/O channel capacity will remain appropriate for very long. The CM-5 I/O system is constructed in such a way as to be connected directly into the processor's communication network. The network bandwidth available to a given I/O device is determined by the number of taps into the network. For each I/O device, multiple network taps are combined to form a single I/O channel that operates at whatever rate the device will allow.

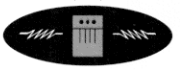
On CM-5 systems, in addition to processing nodes, there will typically be a number of independent nodes with attached disk drive units. These "disk nodes" are configured by software to form a completely scalable, high-bandwidth RAID file system—limited only by the aggregate bandwidth of its individual drives. To an application, the disks appear as a single very-high-bandwidth mass storage subsystem.

Unix Operating System

The CM-5 is normally operated as a multiuser machine. The system supports a Unix-based operating system capable of time-sharing multiple users and a priority-based job queuing system, compatible with NQS. Tasks are fully protected from one another through memory management units and bounds checking hardware in the network. The network state is always accessible to the operating system so messages in transit can be "swapped out" during the process switch from user to user.

In addition to normal time-sharing, the CM-5 supports a form of space sharing, called "partitioning." The CM-5 can be partitioned under software control into several independent sections. Each partition effectively works as an independent,

Figure 4.
Data network topology
The data network uses a fat tree topology. Each interior node connects four children to two or more parents (only two parents are shown). Processing nodes form the leaf nodes of the tree. As more processing nodes are added, the number of levels of the tree increases, allowing the total bandwidth across the network to increase proportionally to the number of processors.



For a 1-D array, compute the average of nearby neighbors.

Fortran 77 with automatic parallelization

```
integer size, i
parameter (size=1000)
double precision x(size), y(size)

do i = 2, size-1
    y(i) = (x(i-1)+x(i+1))/2.0
enddo
```

High-performance Fortran (Fortran 90)

```
forall (i=2:size-1)
    y(i) = (x(i-1)+x(i+1))/2.0
```

C*

```
where (pcoord(0)>0 && pcoord(0)<SIZE-1)
    y = ([.-1]x+[.+1]x)/2.0;
```

Message passing

```
int nodesize = size/nproc;

if (myself > 0)
    CMMD_send_noblock (myself-1,tag, &x[0], len);
if (myself < (nproc-1) {
    CMMD_send_noblock (myself+1,tag, &x[nodesize-1], len);
    CMMD_receive (myself+1,tag,&right,len);
    }
if (myself > 0)
    CMMD_receive(myself-1,tag,&left, len);

for (i=1; i < nodesize; i++) y[i] = (x [i-1] + x[i+1])/2.0;
y[0] = (left + x[1])/2.0;
y[nodesize-1] = (x[nodesize-2] + right)/2.0
```

***Lisp**

```
(*if (and (> (self-address!!) 0)
          (< (self-address!!) x size))
     (*set y (/ (+ (news!! x -1) (news!! = 1)) 2)))
```

Id

```
def relax x =
    { 1,u = bounds x;
        typeof x = vector *0;
         in { array(1,u) of
    | [i] = (x[i-1]+x[i+1])/2.0 || i <- 1+1 to u-1}
    };
```

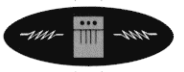


Figure 5.
Rosetta Stone
This example program fragment shows the same function written in different programming styles. All of these programs compile and run efficiently on the CM-5. The program computes the average of neighboring elements in a nondimensional array.

time-shared, machine. Hardware in the communication network isolates these partitions in such a way that the computations in one partition of the machine have no effect on the computations in another. Each partition appears to the software as a single linear address space, although it may map into physically discontinuous sections of the machine. It is even possible to power down a partition of the machine for maintenance while other partitions continue to operate.

Because the CM-5 is used in large-scale, long-running applications requiring very large datasets, the standard Unix file system has been extended to handle files with terabytes of data. Checkpointing facilities, invoked either automatically or under user control, have been added to the operating system [22].

Relation of CM-5 to CM-2

The CM-5 is the successor to the CM-2 and CM-200. The CM-5 shares with the CM-2 the two-network organization, the global address local memory structure, the ability to broadcast and synchronize instructions, and special hardware support for data parallel execution. Its architecture differs most dramatically from the CM-2 in the ability of the individual nodes to execute instructions independently and in the organization of the I/O system. Also, the CM-5 uses a larger-grained 64-bit processor, instead of the single-bit processors and 32-bit floating-point paths of the CM-2. Most programs written for the CM-2 only require recompilation to run on the CM-5.

The most important lesson learned from the CM-2 was the importance of the scalable data parallel model which is used in most commercial and scientific applications of massively parallel machines. As it turned out, the strict per-instruction synchronization of the CM-2 was not required to support this model. The CM-5 is a MIMD machine (multiple instruction/multiple data), although it incorporates many features that are normally found only in SIMD (single instruction/multiple data) machines. The machine incorporates several important architectural features from message-passing machines such as the Caltech Hypercube [21]. The data network of fat trees is based on work from MIT [15] and the New York University Ultracomputer [20].

Support for Multiple Programming Models

The CM-5 is particularly effective at executing data parallel programs, but it was designed with the assumption that no one model of parallel programming will solve all users' needs. The data parallel model, which is embodied in languages such as Fortran-90, *Lisp, and C*, has proved extremely successful for broad classes of scientific and engineering applications. On the other hand, this model does not seem to be the best way to express a control structured algorithm such as playing chess by searching the game tree [8]. In this case, a message-passing model is often more appropriate. Other applications may be better suited by process-based models, in which tasks communicate via pipes or shared variables, or by object-based models in which each processor executes a different program, or even dataflow models, such as those supported by the language Id. Figure 5 shows a "rosetta stone" of program fragments performing a similar operation in many different programming languages. All of these languages compile efficiently for the CM-5. For this particular example, the data parallel languages offer the most concise formulation, but in other cases message-passing-based approaches may offer a more natural expression of an algorithm.

Since the actual hardware required to support all of these programming models is very similar, it is likely that most parallel machines of the future will be designed to support all of these programming models. All that is required is a good communication system, an efficient global synchronization mechanism, and a fast hardware broadcast and reduction. With today's technology, this is relatively inexpensive. This is part of the charm of CHARM architectures.

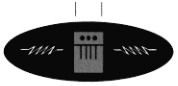
Shared Memory vs. Shared Address Space

Some of the most interesting issues in computer architecture arise in resolving the conflicts between the needs of the programmer and the constraints of the hardware. Sometimes apparent conflicts are resolved by careful separation of the actual requirements. For example, consider the shared vs. distributed memory issue. Programmers would like all of the data to be accessible from every processor, which suggests the use of a single central memory that is accessible by all the processors. Hardware constraints, on the other hand, make this central memory approach costly and inefficient. Local memory, distributed among the processors, is much faster and less expensive.

It would seem at first glance that the conflict is irresolvable, and that the user will be forced to choose between convenience and efficiency. Fortunately, the choice is unnecessary. It is possible to address both the hardware and the software needs simultaneously by distributing memories for fast local access and providing an additional mechanism for accessing all of the memory through a network. This allows the compiler and operating system to provide the user with a shared address space, without sacrificing the cost-effectiveness, speed and efficiency of distributed memory. The CM-5 adopts this combination of shared address space and distributed memory. Hennessy and Patterson have published a discussion of these two dimensions of architecture [10]. Figure 6 shows one of their tables with the addition of the CM-5.

SIMD vs. MIMD

A similar analysis can be applied to the SIMD/MIMD issue. Programmers often find it most convenient to specify programs in terms of a single



flow of control [2], using a sequential or a data-parallel programming style. This frees the programmer from issues of synchronization, deadlocking, stale access, nondeterministic ordering, and so forth. Parallel computers, on the other hand, can operate more efficiently executing independent instruction streams in each processor, since a sequential program overspecifies the ordering and synchronization. These two issues are resolved by providing a machine with both independent instruction streams, and the synchronization and broadcast mechanisms necessary to support a sequential programming style. The program is guaranteed to compute exactly the same results as it would operating on a single processor, although the result is generally computed more quickly.

If we take the classical SIMD machine to be one that is synchronized on every instruction, and the classical MIMD machine to be one in which synchronization is left to the programmer, then the CM-5 normally operates in the middle ground; that is, synchronization occurs only as often as necessary to guarantee getting the same answer as the globally synchronized machine. These invocations of the hardware synchronization operations are generated automatically by the compiler. The processors are free to operate independently whenever such synchronization is unnecessary. Even applications that were originally written for traditional MIMD machines have been able to take good advantage of these hardware synchronization mechanisms. Of course, it is also possible to program the machine as a standard MIMD machine, or for that matter to force a synchronization on every instruction execution, in which case it operates as a SIMD machine. This is illustrated in Figure 7.

Applications

When massively parallel computers were first introduced, many expected that they would only achieve good performance on a narrow range of applications. Experience has shown that, contrary to initial expectations, most applications involving large amounts of data can take advantage of the computational power of massively parallel machines [9]. This is not to say that most existing applications, as currently written, run well on massively parallel machines. They do not. Most existing supercomputer applications were designed with much smaller, nonparallel machines in mind. The average application on Cray Y-MP at the NCSA Supercomputer Center, for example, runs at 70MFLOP/sec in 25MB of memory. Such small-scale applications rarely have the inherent parallelism to take advantage of massive parallelism. On the other hand, large-scale applications that involve many GBs of data can almost always be written in a way that takes good advantage of large-scale parallelism. In a recent survey of the world's most powerful supercomputers, four out of the top five systems reported were Connection Machines [6].

Current applications of the Connection Machine system include high-energy physics, global climate modeling and geophysics, astrophysics, linear and nonlinear optimization, computational fluid dynamics and magnetohydrodynamics, electromagnetism, computational chemistry, computational electromagnetics, computational structural mechanics, materials modeling, evolutionary modeling, neural modeling, and database search [19]. There are many other examples. These applications use a variety of numerical methods, including finite difference and finite element schemes, direct methods, Monte Carlo calculations, particle-in-cell methods, n-body calculations, and spectral methods [4].

Many applications that were originally developed for the CM-2 achieve high rates of sustained performance on the CM-5. For example, a rarefied gas dynamics simulation, for modeling situations such as spacecraft reentry [17, 18], which runs at 8GFLOP/sec on the largest CM-2, runs at a sustained rate of 64GFLOP/sec on a 1,024 processor CM-5 (1GFLOP/sec is about the peak rate of a single processor on a Y-MP C90). Another example is the Global Climate Modeling software developed by Los Alamos National Laboratory (see Figure 8). This simulation is designed to model global climate warming due to the buildup of greenhouse gases. The algorithm relies on a dynamically varying mesh of over 250,000 points and 1,200,000 tetrahedra to cover the Earth's atmosphere. The application consists of over 30,000 lines of Fortran. Dynamic rearrangement of the mesh automatically adjusts local resolution to resolve important details of the flow, such as storm formations. Simulation of a six-week time period requires about one hour of simulation time on a 64K CM-2. When run on a 1,024 processor CM-5, this six-week simulation completes in less than five minutes.

Massively parallel machines are also becoming increasingly important in applications outside of science and engineering. Many of these commercial applications take advantage of the scalable input/output capability of the CM-5. For example, American Express corporation is using two CM-5's to process its large credit card databases. Another example is the oil industry, where companies such as Mobil and Schlumberger use CM-5's for large-scale seismic processing.

Because the CM-5 provides direct user-level access to the network hardware, it is also used as an experimental platform for testing new architectures. There is no operating system software overhead between the user and the network. Computer scientists at the University of California, Berkeley are using this property of the CM-5 to investigate a model of parallel computation based on what they term a "Threaded Abstract Machine" (TAM). This abstract machine is the compilation target for parallel languages such as ID90. This language, originally designed for dataflow computers, relies on a fine-grained, message-driven execution model. They have taken advantage of direct access to the data network to implement an ultralight form of remote procedure call, called active messages [7].

At the University of Wisconsin, investigators are also using their CM-5 for virtual prototyping; that is, they use the machine to simulate other architectures to better under-

stand alternative architectural choices. The CM-5 Wisconsin Wind Tunnel prototyping system mimics the operations of a large-scale, cache-coherent, shared-memory machine. Almost all application instructions are directly executed by the hardware. References to shared data that are not locally cached, invoke communication to retrieve memory pages from the global address space. The scalability and flexibility of the CM-5 has allowed these researchers to run full-sized applications on a simulated architecture and study the effects of various cache coherency schemes [11].

Some of the applications of the CM-5 really require even greater computing power than is available today. For example, a consortium of particle physicists has formed to construct a special version of the CM-5 for teraflops calculations in quantum

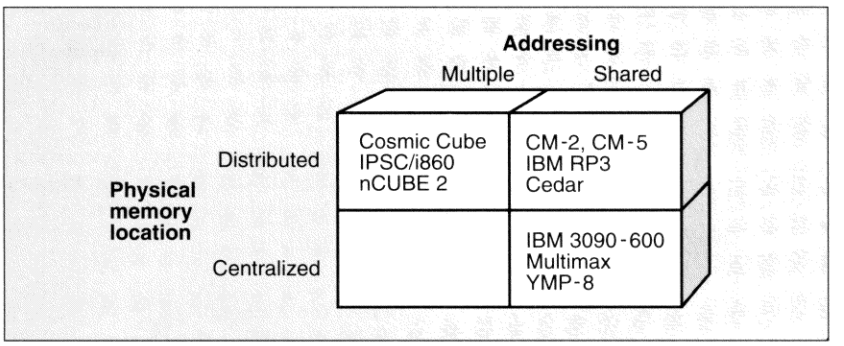


Figure 6.
Parallel processors according to centralized versus distributed memory and shared vs. multiple addressing
Source: Patterson and Hennessy
The hardware notion of shared memory is often confused with the software notion of shared addressing. Machines with locally distributed memory can be designed to support a program model on which the user sees all data in a single address space. This allows the machine to combine the hardware efficiency of distributed memory with the software convenience of a single address space.

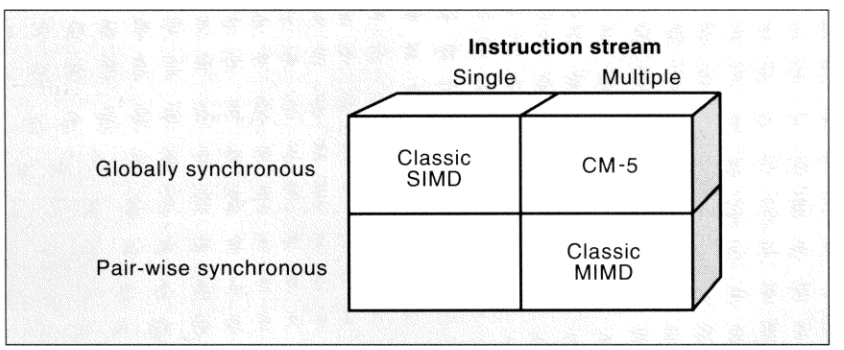


Figure 7.
Instruction stream
In a classical SIMD machine, global synchronization is provided by a single broadcast instruction stream. In a classical MIMD machine, multiple instruction streams are synchronized via pairwise interactions; for example, through messages or shared memory references. The CM-5 is an example of a coordinated MIMD machine, in which multiple instruction streams are synchronized whenever necessary through global synchronization hardware.

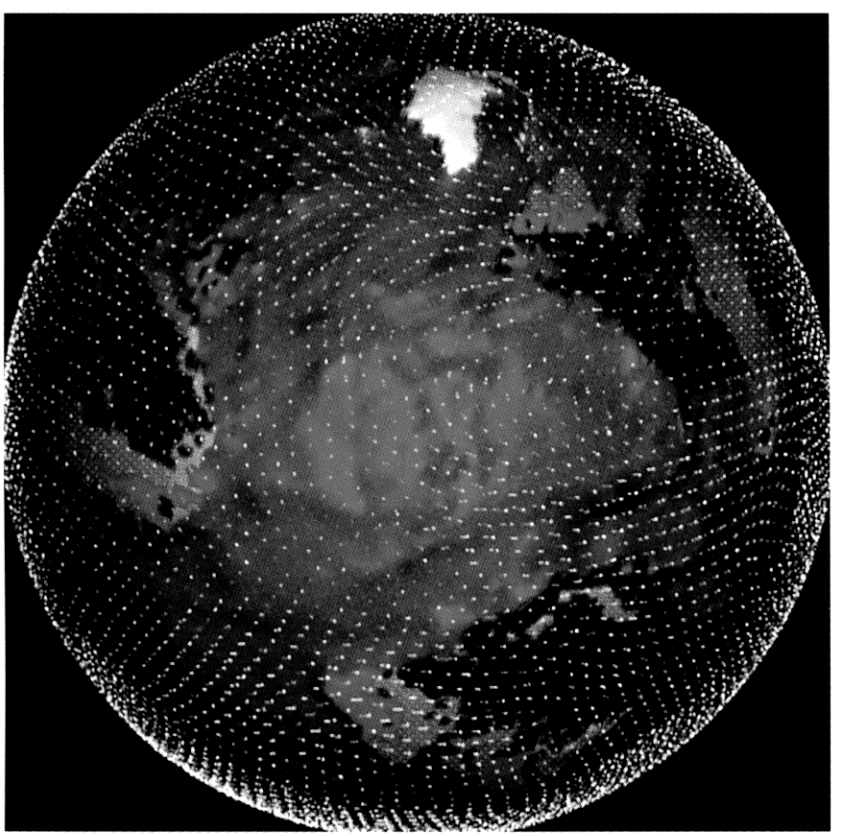


Figure 8.
X3D—Massively parallel adaptive advanced climate model
This global climate model simulation is an example of a scientific application of the CM-5. Lines represent an irregular computational mesh of approximately 1,200,000 tetrahedra and is used to compute Lagrangian hydrodynamics, Monte Carlo Transport, and thermal heat diffusion. Land masses are shown in yellow. The goal of this application is to better answer fundamental questions about changes in the earth's global climate through computer simulation. Credit: *Harold Trease (Los Alamos National Laboratory)*

chromodynamics. These calculations could potentially advance our understanding of the structure of atomic particles [1]. Protein structure prediction and global climate modeling are other examples of problems that will ultimately require teraflops performance. These calculations have tremendously large potential economic impact. As a final example of an application that will require even greater performance, one of us (W.D. Hillis) is using a Connection Machine to design computer algorithms by a process analogous to biological evolution [13]. Perhaps some future version of the Connection Machine will use circuits designed by these evolutionary methods.

Summary

In summary, we believe the most important features of the Connection Machine are its scalability, its global access of locally distributed memory, and its global coordination of local executing processes. Together, these features allow the CM-5 to support a variety of parallel programming models and a wide range of applications with good performance. Because the cost of this universality is relatively low, we predict that the architectures of this type, using coordinated homogeneous arrays of RISC microprocessors (CHARM), will be the dominant form of large-scale computing in the 1990s. ◘

References

1. Aoki, A., et al. Physics goals of the QCD Teraflop project. *J. Mod. Phys. C, 2,* 4 (1991), 892–947.
2. Backus, J. Can programming be liberated from the von Neumann style? *Commun. ACM* 8 (1978).
3. Blelloch, G. Scan primitives and parallel vector models. Ph.D. thesis, Carnegie-Mellon University, Jan. 1989.
4. Boghosian, B. Computational physics on the Connection Machine. *Comput. Phys. 4,* 1 (1990).
5. Dongarra, J.J. Performance of various computers using standard linear equations software. University of Tennessee, Sept. 1992.
6. Dongarra, J.J. TOP500. Tech. Rep. 37831, Computer Science dept., University of Tennessee, July 1993.
7. Eicken, T., Culler, D., Goldstein, S. and Schauser, K. Active Messages: A mechanism for integrated communication and computation. In *Proceedings of the Nineteenth International Symposium on Computer Architecture.* ACM Press (May 1992).
8. Fox, G.C. Parallel computing comes of age: Supercomputer level parallel computations at Caltech. *Concurrency, 1,* 1 (Sept. 1982).
9. Fox, G.C. Achievements and prospects for parallel computing. *Concurrency: Pract. Exp., 3,* (1991), 725.
10. Hennessy, J.L. and Patterson, D.A. *Computer Architecture: A Quantitative Approach,* Morgan Kaufmann, 1990.
11. Hill, M., Larus, J., Reinhardt, S. and Wood, D. Cooperative shared memory: Software and hardware for scalable multiprocessors. Tech. Rep. 1096, Computer Science Dept., University of Wisconsin-Madison, July 1992.
12. Hillis, W.D. Balancing a Design. *IEEE Spectrum* (May 1987).
13. Hillis, W.D. Co-Evolving parasites improve simulated evolution as an optimization procedure. *Physica D* 42 (1990).
14. Hillis, W.D. and Steele, G.L. Data parallel algorithms. *Commun. ACM 29,* 12 (Dec. 1986).
15. Leiserson, C.E. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput. C-34, 10* (Oct. 1985).
16. Leiserson, C.E., et al. The network architecture of the connection machine CM-5. In the *Fourth Annual ACM Symposium on Parallel Algorithms and Architecture* (June 1992).
17. Long, L.N., Kamon, M., Chyczewski, T.S. and Myczkowski, J. A deterministic parallel algorithm to solve a model Boltzmann equation (BGK). *Comput. Syst. Eng. 3,* 1–4, (Dec. 1992), 337–345.
18. Long, L.N., Kamon, M., Myczkowski, J. A massively parallel algorithm to solve the Boltzmann (BGK) equation. AIAA Rep. 92-0563, Jan. 1992.
19. Sabot, G., Tennies, L., Vasilevsky, A. and Shapiro, R. In *Scientific Applications of the Connection Machine,* H.D. Simon, Ed. World Scientific, River Edge, N.J., Second ed., 1992, pp. 364–378.
20. Schwartz, J. Ultracomputers. *ACM Trans. Program. Lang. Syst. 2,* 4 (Oct. 1980).
21. Seitz, C.L. The cosmic cube. *Commun. ACM 28,* 1 (Jan. 1985).
22. Thinking Machines Corporation. CM-5 Software Sum., CMost Version 7.1, Jan. 1992.
23. Thinking Machines Corporation. The Connection Machine CM-5 Tech. Sum. (Oct. 1991).
24. Wade, J. The vector coprocessor unit (VU) for the CM-5. Symposium Record: Hot Chips IV, Stanford University, IEEE Computer Society, August 1992.

CR Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures—*connection machines*; C.5.1 [**Computer System Implementation**]: Large and Medium ("Mainframe") Computers—*super (very large) computers*

General Terms: Design, Performance

Additional Key Words and Phrases: CM-5, Massively Parallel Systems

About the Authors:

W. DANIEL HILLIS is chief scientist and cofounder of Thinking Machines Corp. He is the architect of the Connection Machine computer. His research interests include parallel programming, evolutionary biology, computer architecture, and the study of complex dynamical systems with emergent behavior. His current research is on evolution and parallel learning algorithms.

LEWIS W. TUCKER is director of programming models at Thinking Machines Corp., and is responsible for the CM-5's message passing and scientific visualization library development. His research interests include parallel architecture design, scientific visualization, and computer vision.

Authors' Present Address: Thinking Machines Corp., 245 First Street, Cambridge, MA 02142; email: {danny, tucker}@think.com