# Massively Parallel Computers: The Connection Machine and NON-VON

RICHARD P. GABRIEL

**The demand for high-speed computers is increasing, and as the limits on single-processor computers are approached, researchers are turning their attention to parallel computers. Parallel computers have more than one processing element; massively parallel computers contain many processing elements. Constructing computers on this scale and learning how to program them effectively will be major challenges in the next decade. Several such computers, for example the Connection Machine and the NON-VON, are under development.**

COMPUTERS HAVE BECOME BOTH FASTER AND LARGER—larger in the sense of having more dynamic random access memory (RAM) and faster in the instruction-execution rate of the central processing unit (CPU). Because computer memories are growing faster than the speed of CPU's, the fraction of the computer performing useful computations has diminished sharply over the years.

The sorts of computations that are being done today or are being contemplated for the near future require considerable amounts of computer processing time, and many computations that one would like to run in real time or faster cannot now be done. For example, in weather prediction simulations of weather must be done faster than weather can happen.

Problems in large databases, artificial intelligence, signal processing, and weapons design are only some of the areas in which the computational limitations cannot be met with current computer architectures. The obvious solution to the computational-need problem is to apply parallelism. That is, if a computer with a single processor is not sufficient, then build a computer that has more than one processor—perhaps very many processors—to do the job.

## The Multiprocessor Spectrum

Multiprocessors come in various sizes, ranging from two processors connected by a simple serial link to computers with tens of thousands of processors connected by a complex communications network.

A massively parallel computer requires many processors and perhaps many memories. At present, most small parallel computers implement a coarse-grained parallelism, in which relatively large processors running relatively large, mostly independent computations communicate infrequently. Most massively parallel computers implement a fine-grained parallelism, in which small processors running small or identical operations communicate frequently. As the size of the processors decreases, the grain of parallelism decreases, the communication frequency tends to increase, and the ratio of communication to processing tends to increase. In this article I concentrate on computers at the larger end of the scale, but I will also mention some smaller parallel computers of historical importance.

## Communications Networks

In a certain class of multiprocessors a typical computer is a collection of $n$ processors connected to $m$ memories, and each processor can communicate with some subset of the memories. A special case is $n = m$, with each processor communicating with each memory. The physical means by which this communication between processors and memory occurs is called the communications network.

*Crossbar switches.* When $n$ and $m$ are small, it is reasonable for the network to have direct connections between each processor and each memory. For example, when $n = m$, it is possible to build a crossbar network in which there is a direct link from each processor to each memory. These links can be circuit-switched or package-switched. In a circuit-switched link a connection can be used continuously for communication, while in a packet-switched link each message is routed individually to its destination. A circuit-switched network is like a telephone link and a packet-switched network is like the U.S. Postal Service (*1*). The 16-processor Carnegie-Mellon C.mmp (*2*) is an example of a computer with a crossbar switch); performance experiments on it provided some of the first important data on shared-memory multiprocessors.

*Distributed memory.* In another variant of parallel machines the processors have memories attached to them and the global memory of the computer is nothing more than the sum of each of the individual memories. When a processor wishes to access some memory location, the address of the memory location determines the individual memory module in which the desired memory location is located. A memory reference, then, will take a shorter or a longer time depending on whether the reference is to a location in the local memory or to a location in the local memory of some other processor, respectively.

The communications topology and geometry become important for reasons of performance and programming. If it takes relatively long to refer to a memory location far away, it becomes more important to allocate tasks and data to processors and memories close together. If the solution to a problem can be reformulated in such a way that the structure of the communications in the solution corresponds to the structure of the multiprocessor on which the solution will run, then the reformulation may well be worth the reprogramming effort. The 50-processor Carnegie-Mellon Cm* is a distributed memory multiprocessor; several clusters of machines connected by a bus are themselves interconnected by intercluster buses (*3*).

Richard P. Gabriel is president and chief technical officer of Lucid, Inc., Menlo Park, CA 94025 and is consulting associate professor of computer science, Stanford University, Stanford, CA 94306.

*Grids.* Processors and their associated memories can be laid out as an $n \times m$ grid. In one form of grid, each processor is connected directly to its north, east, west, and south neighbors, with the exception of the processors on the edge of the grid. Algorithms such as relaxation are naturally performed on such processors because the structure of the processor matches the structure of the problem and the problem solution. The east and west edges and the north and south edges of a grid can be connected to produce a torus, which has a smaller diameter (*4*).

ILLIAC IV (*5*) was a grid machine. It had an $8 \times 8$ array of processors, each having a cycle time of 240 nsec and 2048 words of memory. The east and west edges of the array were connected to produce a cylinder. The machine was used primarily for matrix manipulation and for solving partial differential equations. The array of processors ran as a single-instruction, multiple-data (SIMD) machine and appeared to the programmer as an attached, special-purpose processor on a Burroughs B6500 computer.

One of the first massively parallel computers was the Goodyear Massively Parallel Processor (MPP), built in 1979 (*6*). This machine is a $128 \times 128$ grid of 64-bit, 100-nsec processors. Each processor has 1024 bits of RAM. The processors are packaged eight per chip.

The connectivity of the edges of the MPP can be changed under software control. The edges can be left open, the east and west edges can be connected to create a vertical cylinder, the north and south edges can be connected to create a horizontal cylinder, both cylinders can be created to form a torus, and each processor number $n$ ($0 \leq n < 127$) on the east edge can be connected to processor $(n + 1)$ on the west edge, with processor 127 on the east being connected to processor 0 on the west, to create a spiral linear circuit.

*Omega networks.* The major problem to solve in building a massively parallel computer is how to interconnect a very large number of processors and memory modules. In a crossbar switch, if $n$ processors are being connected to $n$ memory modules, then the amount of hardware is proportional to $n^2$. For $n < 100$ this amount of hardware is feasible, but for $n > 10{,}000$, the cost of the crossbar switch is prohibitive and its size unmanageable.

There is a class of connection strategies whose hardware requirements grow as $\log n$; this family uses a network known as the omega or butterfly network. This type of network is a member of a class of networks called shuffle-exchange networks. Omega networks use a multistage network to implement the shuffle-exchange; in an alternative implementation, a recirculating network is used.

The omega network comprises a column of processors and a column of memory modules. Each memory module contains addresses in some range of addresses; to move a message from a given processor to a given memory module, a series of steps is taken, each bringing the message closer to the memory module to which the message is directed.

The address of a memory location can be used to determine the memory module in which that location is situated by looking at the address one bit at a time; as in binary search, the number of possible memory modules in which the desired memory location is contained is cut in half at each stage. Furthermore, if a different piece of hardware is used at each stage, several memory requests can be moving through this network concurrently, gaining the same sort of benefits as is obtained from pipelining.

Consider the case of $n = 8$; there are eight processors and eight memory modules. Three bits are used to move a memory request from a processor to a memory module. There are three columns of message-processing nodes on the network, where each column has eight message-processing nodes. Consider the address of a memory location. The first address bit selects the correct half of the network to which to direct the memory request. The second column of memory-processing nodes examines the second bit and directs the memory request to the correct half of the remainder of the network. That is, the second column of memory-processing nodes is partitioned into two parts, and each of the parts is able to communicate only with half of the remaining network. In each half of the column, the same algorithm that is used at the first level of the network is used to select the half of the network accessible to the second level. After the second column has processed the memory request, the memory request will have been routed to the correct quarter of the network.

Figure 1 shows the omega network that connects eight processors to eight memories. Memory requests flow from left to right. Figure 1 is drawn in the traditional manner, which deemphasizes the perfect shuffle between each stage.

A common simplification made in computers of this type is to package a memory with a processor so that some memory requests are not routed over the network. In Fig. 1, processor $P_1$ would be packaged with memory module $M_1$, the effect being to fold the diagram so that the left-hand edge coincides with the right-hand edge. This is the communications network architecture used by the Bolt, Beranek & Newman Butterfly computer (*7*).

*The hypercube.* Another simplification to the omega network is the hypercube, sometimes called a Boolean $n$-cube. A hypercube is an omega network with the processing elements in each row collapsed to one processor. For example, a three-dimensional hypercube has eight nodes, arranged as a cube. The nodes of the cube are located at the coordinates $\langle 0,0,0 \rangle$, $\langle 0,0,1 \rangle$, $\langle 0,1,0 \rangle$, $\langle 0,1,1 \rangle$, $\langle 1,0,0 \rangle$, $\langle 1,0,1 \rangle$, $\langle 1,1,0 \rangle$, and $\langle 1,1,1 \rangle$. Two nodes are connected if their coordinates differ in exactly one component.

Routing that involves relative addressing in an $n$-dimensional hypercube is particularly simple. If the relative node address of a message is all zeros, the message is at the correct node; otherwise, the router can pick a component of the node address that is not zero and send the message in a direction in which that component of the node address is one, zeroing that component before it is sent. Thus, every step of the transfer of a message zeroes one component of the node address; it takes at most $\log n$ time to transmit a message through an $n$-dimensional hypercube.

The Connection Machine (*1*) uses the hypercube connection scheme.

## Massively Parallel Machines: The Connection Machine and NON-VON

In this section two of the massively parallel machines in existence or under development are discussed. Special attention is paid to the hardware characteristics of the machines, but programming issues are also raised. The Connection Machine, built by Thinking Machines Corporation, is highlighted because it offers the possibility of a high degree of extendable parallelism and because the programming issues have been well stated and well thought out by the machine's designers. Columbia University's NON-VON is discussed as a testing ground for a range of possible parallel architectures and programming methodologies.

*The Connection Machine.* This machine is an SIMD computer comprising 65,536 ($2^{16}$) processors connected as a Boolean $n$-cube. In addition, there is an x-y grid superimposed on the machine. Each processor in the machine is a 1-bit processor with 8 bits of internal-state information and 4096 bits of local memory. The internal state is in the form of eight general-purpose registers; in addition, there are eight special-purpose registers. These 16 registers are referred to as "flags registers."

Except for processors disabled for the current instruction, all processors execute the same instruction at the same time. The

instructions are broadcast to each processor by another computer, the microcontroller, which takes as input a series of "macroinstructions" from a controlling computer, converts each macroinstruction to a series of three "nanoinstructions," and distributes the nanoinstructions to each of the processors in the Connection Machine. The nanoinstructions arrive at each Connection Machine processor at the rate of 3 million per second (this assumes that the processors are controlled by a 3-MHz clock).

Each processor takes three inputs: 2 bits from memory and a flag. From these inputs 2 bits are computed: one to be placed in the local memory and another in the flags registers. Three nanoinstructions are required to perform each such operation. The operation performed on the three inputs is determined by a truth table of all Boolean functions on 3 bits; in fact, one truth table is used to determine the memory bit written and another for the flags register bit written.

Every nanoinstruction performs one memory transfer. The first nanoinstruction gets the first memory bit, specifies the truth table for computing the memory result, and specifies the internal flag bit for conditionalizing this cycle. The second nanoinstruction gets the second memory bit, specifies the truth table for computing the flags register result, and specifies the input flag bit. The third nanoinstruction conditionally stores the result in memory and specifies which flags register gets the flag bit result.

The microcontroller provides 55 bits of information during this three-instruction cycle: two 12-bit memory addresses, 4 bits to specify the flags register from which to obtain the flag input, 4 bits to specify the flags register to which to write the flag output, 8 bits for the truth table for the memory bit computation, 8 bits for the flags register computation, 2 bits for the x-y grid motion, and 5 bits for conditional execution.

Each processor receives the same 55 bits of control information. The 5 bits of conditional execution information determine whether a particular processor executes the nanoinstruction: 4 bits determine the flags register to use to determine whether to execute and the remaining bit determines how the flag is to be interpreted—if "on," the instruction is executed only if the flag is on; if "off," the instruction is executed only if the flag is off.

A microcontroller distributes nanoinstructions to 16,384 processors; there are four such microcontrollers in a full Connection Machine. The microcontroller takes a sequence of macroinstructions from a host computer—either a Symbolics 3600 or a Vax—and translates the macroinstructions into a sequence of nanoinstructions that are broadcast to the Connection Machine. The microcontroller is isolated from the host by a pair of first in–first out buffers (FIFO buffers) that buffer macroinstructions from the host to the microcontroller and that buffer data returning from the connection machine.

The router moves messages among the processors. Each router serves 16 processors; the router and its processors are built on a single chip. The router can perform five different operations: injection, delivery, forwarding, buffering, and referral. Injection is putting a message into the network, delivery is putting a message in the memory of the processor to which it is addressed, forwarding is sending a message from one router to another (when a message must leave the chip), buffering is used to store multiple messages to a single router, and referral is used when the buffer of a destination router is full (the message is sent to an adjacent router whose buffer is not full). When a message is referred it is not sent any closer to its destination.

When a message is sent to a target processor it moves closer to the target by means of the relative routing algorithm mentioned earlier. Because the address of any processor in the Connection Machine is 12 bits and because there can be only 12 bits in an address, the maximum time required to move a message from one processor to another is $12k$ for some constant $k$, assuming that no referral occurs. However, because a message can move away from its destination when it must be referred, there are worse cases than $12k$.

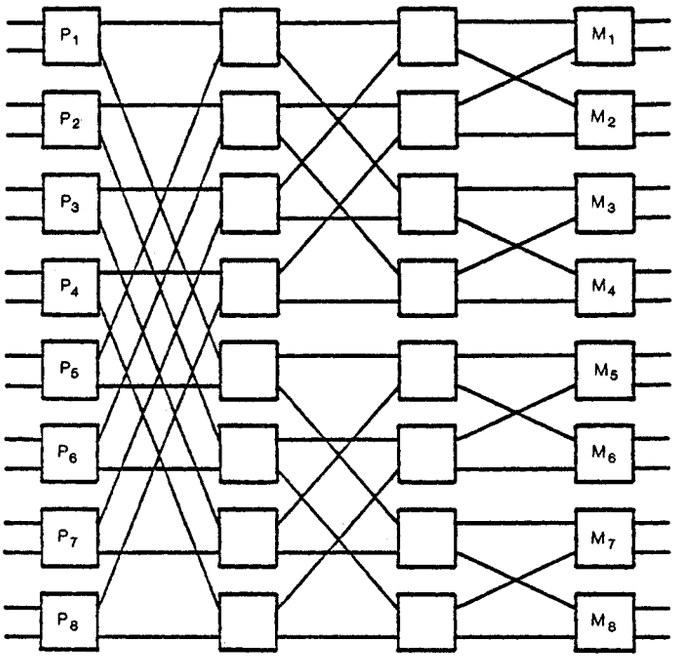


Fig. 1. Omega network.

Part of the message is the address to which the message is addressed, and messages move in bit-serial form. Assuming a 32-bit message and assuming that each processor is sending a message to some other processor, one can expect all messages to reach their destinations within 8 to 20 petit cycles. A petit cycle is the cycle during which the router directs all its messages to some other router. The router considers each dimension in the 12-bit address at a time; it scans all messages for 1 bit in that dimension and transmits each of them somewhere during a dimension cycle. Twelve dimension cycles move all messages to some other router. One measure of the performance of the router is the time it takes to transmit a bit from one router to an adjacent one: 1 μsec.

The Connection Machine components are simple, and one might expect that programming it would be similar in complexity to microcoding a uniprocessor—single bits are operated one at a time, with concurrent activities constantly happening. The host computer and the microcontrollers help the programmer by translating higher level instructions into the low-level nanoinstructions.

Programming the connection at the higher level can be accomplished in Connection Machine LISP (CmLISP). This LISP can be characterized as a radical extension of an existing programming language, in this case Common LISP (*8*). The extension is the addition of a new data structure, called a xector, which is a sequence-like structure. A xector comprises a domain, a range, and a mapping from the domain to the range. One can think of this data structure as being a set of processors in the Connection Machine (the range) and a set of names for the processors (the domain); the mapping of domain to range is handled by the association of a name with a processor. The mapping of an element in the domain to an element in the range is called an element of the xector, and the two components of the element are called the index and the value.

The following is a simple xector that maps the domain $\{a\ b\ c\}$ onto the range $\{1\ 2\ 3\}$: $\{a \rightarrow 1\ \ b \rightarrow 2\ \ c \rightarrow 3\}$. In this xector there are

three elements: one with index *a* and value 1, one with index *b* and value 2, and one with index *c* and value 3.

Operations are defined on xectors that enable the programmer to express operations which are performed concurrently on the elements of xectors. LISP programs manipulate xectors, and operations on xector-based objects run faster on the Connection Machine than they would on a uniprocessor. In this respect, the Connection Machine is like a special-purpose processor attached to the host machine.

There are two basic operations, $\alpha$ and $\beta$. Operation $\alpha$ takes an expression and produces a xector that has the value of that expression as its constant value, and its domain is every possible object. Operation $\alpha$ is most useful when used in conjunction with an extension to Common LISP function calling: function calling using a xector of functions.

Operation $\beta$ takes three arguments, a function and two xectors. The two xectors are assumed to be over the same domain. The result of $\beta$ is a third xector; the values of the resultant xector are taken from the values of the first argument xector and the indices of the resultant xector are taken from the values of the second argument xector.

By definition, a mapping cannot contain two elements each with the same index; the resultant xector produced by $\beta$ can contain such a collision. The first argument in a $\beta$—the function—is used to resolve a collision by supplying a function to apply to the colliding values to compute a single value.

The Connection Machine must implement these abstract programming constructs by being programmed to manipulate xectors, which are used as active memory. The allocation of objects in the Connection Machine is one of a class of low-level programming tasks whose running efficiency determines the overall efficiency of the machine and whose programmability determines the machine's short-term usefulness.

Often, the low-level programming involves finding out which processors contain certain information and placing specific processors in touch with other specific processors. The following simple algorithm is representative of the style of programming used at the lower levels of the Connection Machine.

Suppose that it is important for a processor to find another processor that is free and that is close to the first one. The first processor, called the requesting processor, sends a message to all neighboring processors that contains a request for a free processor as well as the address of the requesting processor. Each processor that receives such a message sends it along to all its neighbors (except the one from which it received the message) unless it is free; in the latter case, a response is sent to the requesting processor. When the requesting processor receives the response from the found processor, it sends out a cancellation notice along the same paths followed by the original request, but at a higher rate. This cancellation message will eventually reach the wave front of the original request and cancel it.

For the cancellation message to move faster than the original request, the original request must move slower than it naturally could through the machine, and this is accomplished with artificial delays. Further complications arise when it is possible for many such requests to be active in the machine at once. For example, when two waves meet, both must halt at the meeting place.

*The NON-VON computer.* NON-VON is used to describe a family of parallel machines having a similar architecture. Designed by David Elliot Shaw and his colleagues at Columbia University, the first 64-element prototype was completed in late 1984 (*9*). The NON-VON family of architectures is represented by the general NON-VON machine, which contains all the elements present in the family.

There are two categories of processors in the NON-VON, the small processing element (SPE) and the large processing element (LPE). SPE's are configured as a complete binary tree—each processor, except for the root and the leaves, is connected to three others: its parent, a left child, and a right child. The leaf processors are connected by a two-dimensional grid in which each leaf is connected to four neighbors. This tree is referred to as the active memory tree. SPE's operate in SIMD mode under the control of the LPE's.

Each SPE is connected to an LPE, which has an associated active memory controller that broadcasts an instruction stream to SPE's at or below the level of connection. Therefore, subsets of the active memory tree operate as a set of SIMD processors.

An LPE is a processor having approximately the power of the Motorola MC68020. It is a full 32-bit microprocessor which, at a clock speed of 16.67 MHz, is a 2 to 4 MIPS machine. An SPE is a much smaller processor; each is connected to a small amount of RAM—32 to 256 bytes. The processors are 8-bit computers with some number of 1-bit registers and some number of 8-bit registers.

The LPE's are connected via a communications network called the root-point network. In summary, NON-VON can support SIMD, MIMD (multiple-instruction multiple-data), and MSIMD (multiple SIMD) operations.

NON-VON machines share the SIMD programming style of the Connection Machine: The same instructions are issued to all SPE's under the control of a given LPE, but each SPE can be directed to set a flag that depends on its own state that will cause that SPE to ignore the broadcast instruction stream.

## Conclusions

Many programmers wish to solve problems that require more computer power than is easily available. As the range and the size of problems that can be solved by computers grow, the range and size of the problems that programmers would like to solve also grow.

The only known way to achieve consistent increases in computer power is through the use of parallelism. As might be expected, those who would design the parallel machines to solve the computer power problem are among the people who most need faster computers. The key to designing a successful computer is to simulate—on another computer—its characteristics carefully and thoroughly before fabrication and construction.

Programming a parallel computer of a design greatly dissimilar to that of computers currently in use will require computer scientists to stretch and extend programming methodologies. To solve the computer power problem, it will be necessary to design machines that have a high instruction-execution rate and that are programmable. In other words, there must be power to tap and there must be a way to tap that power.


REFERENCES AND NOTES

1. W. D. Hillis, *The Connection Machine* (MIT Press, Cambridge, 1985).
2. S. H. Fuller and S. P. Harbison, *Technical Report Carnegie-Mellon University* (No. CMU-CS-78-146, Carnegie-Mellon University, Pittsburgh, 1978).
3. R. J. Swan *et al.*, *Proc. AFIPS 1977 Natl. Comput. Conf. 46* (1977), p. 645.
4. Diameter is a term used to compare the characteristics of a communications network. It is the maximum of the minimum distances between pairs of nodes in a network, where distance is measured either as the number of routing stops through which the message passes or the time it takes the message to travel from its source to its destination.
5. G. H. Barnes *et al.*, *IEEE Trans. Comput.* C-17, 746 (1968).
6. Goodyear Aerospace Co., *Tech. Rep. GER-16684* (1979).
7. R. Rettberg *et al.*, *Bolt, Beranek & Newman Rep. 4098* (1979).
8. G. L. Steele, *Common Lisp Reference Manual* (Digital Press, Burlington, MA, 1984).
9. D. E. Shaw, *Technical Report* (No. CUCS-29-82, Columbia University, New York, 1982).