Hierarchically interconnected multiprocessors

by P. A. Franaszek
C. J. Georgiou
A. N. Tantawi

The design of interconnection networks is a central problem in parallel computing, especially for shared-memory systems, where network latency, or delay, is one factor that limits system size. This paper discusses aspects of one particular approach to network structure, a design comprising a multiplicity of subnetworks that form a hierarchy of paths. The hierarchy includes fast paths that are used in the absence of contention, and alternate paths with contention resolution. That is, just as in the case of a memory hierarchy, the fastest component of the hierarchy that can provide the desired function is utilized at a given time. The viability and robustness of hierarchical networks is studied first by examining circuit and implementation issues, and then by considering performance modeling and analysis. The overall performance of the hierarchy is shown to be close to that of a contention-free network of fast paths.

1. Introduction

Shared-memory and message-passing architectures are widely studied designs for parallel machines. In this paper we deal mainly with shared-memory systems, because we believe that the interconnection network in shared-memory architectures affects the performance more significantly than in other parallel architectures (e.g., message-passing). A critical parameter in such a network is latency, or delay, which we investigate in this paper.

Storage in a shared-memory system may take the form of a set of distinct memory modules with uniform speeds of access by all processors. Alternatively, as in [1], memory may be shared and distributed, with part of the global memory associated with each processor. In this paper we concentrate on the latter type of organization. Consider a system with N processors (each with a cache and private memory) and N memory modules that hold shared data. Let each memory module be associated with one of the processors, so that an $N \times N$ network, or switch, is sufficient to connect a processor with any other processor-memory-module pair. The fact that data are shared may impose quite stringent performance requirements on the network, since every data reference may require a round trip. This point can be better illustrated as follows: Let us assume that in a shared-memory system, the delay for accessing data through the network must be no greater than a small number of instruction cycles (e.g., three [1]), in order to avoid significant performance degradation. This means that in a system with 10-MIPS processors (i.e., instruction cycles of 100 ns), the combined total of the round-trip network delay and memory access time must be under 300 ns; if a system were to be built with 100-MIPS processors, the maximum

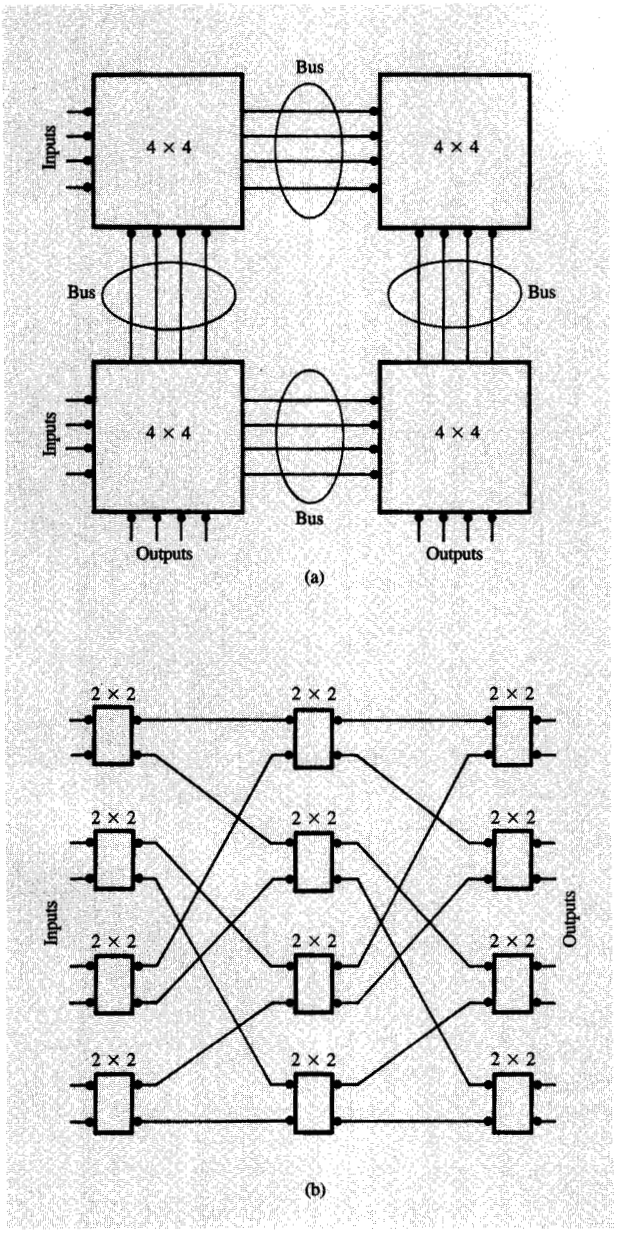

Figure 1

Typical 8 × 8 interconnection networks: (a) partitioned crossbar; (b) multistage.

latency would be reduced to only 30 ns. This requirement makes the design of the network very critical.

The types of networks we consider are crossbars and multistage networks such as banyans [2]. By a crossbar, we mean a single-stage, nonblocking circuit switch (implementable, for example, as an $N \times N$ array of crosspoints). **Figure 1(a)** is a diagram of an 8 × 8 crossbar. Multistage networks consist of sets of nodes (collections of switching elements, such as crossbars), each with input and output ports. **Figure 1(b)** is a diagram of an 8 × 8 multistage network with nodes consisting of 2 × 2 crossbars.

Crossbars have a circuit complexity of $O(N^2)$, and multistage networks typically have a complexity of $O(N \log N)$, so it is sometimes argued that the latter are preferable for large N. However, there is a large range of values within which crossbars perform well and are practical. With modern implementations, crossbars of size up to $N \sim 1000$ appear reasonable. In fact, the limit seems to be tied more closely to the problem of control (i.e., contention resolution) than to the quadratic growth in the number of crosspoints, usually considered to be the limiting factor. On the other hand, for large N, multistage networks may not have sufficiently high performance. As an example of the performance limitations of multistage networks, we note that in at least one instance [1] the system performance requirements have led to a design in which the multistage network had to be implemented with a circuit technology considerably faster than that of the processors.

The limitations of conventional multistage or crossbar networks for shared-memory architectures have led us to explore an alternative approach to network design: one that employs a hierarchy of networks that may include a combination of crossbars and multistage networks. We have found that such an approach can produce a network with sufficiently high performance at a reasonable cost. In this paper, we present the hierarchical network approach, first by reviewing reasons for a multiple-network structure (Section 2) and then by discussing network design and implementation issues (Sections 3 and 4). Finally, we present performance modeling and analysis results (Section 5).

2. Network performance limitations and design strategies

Network latency is the sum of the time required for the start of the message to travel from source to destination and the transmission time for the remainder of the message. The latter can be decreased almost arbitrarily, within the limits of a given technology, by using parallel paths. The former is a function of a) network design parameters, such as the amount of buffering encountered en route, the number of chip crossings (number of times signals go from chip to chip), and the number of levels of logic, and b) contention due to network traffic. Contention can be either for paths or for access to network control logic. In a typical network, new messages arrive, may be stored (buffered) internally, and ultimately are delivered to their destinations. A message may contend with new messages, stored messages at the input, or stored messages at the output (because of possible buffer overflow). The above suggest that network latency may be

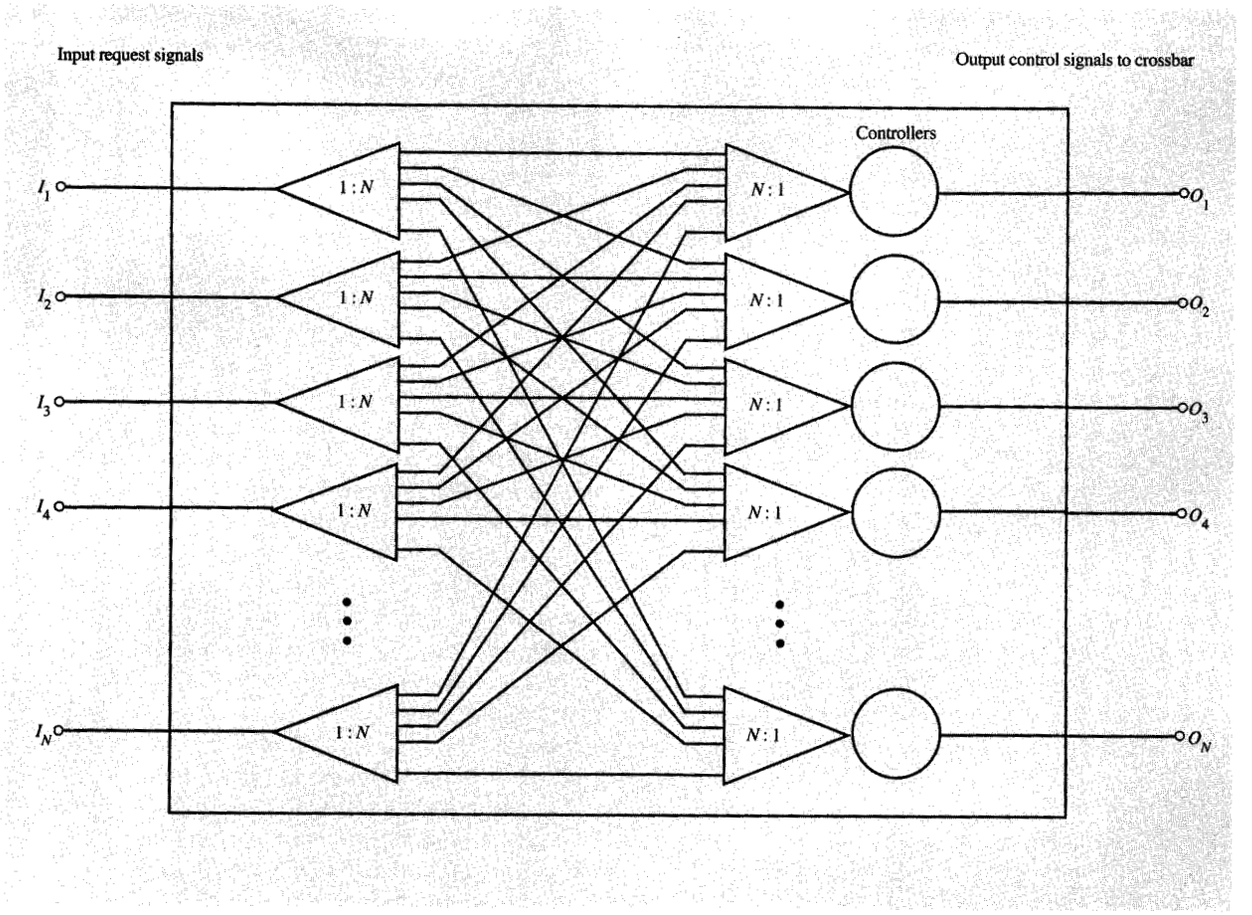

Figure 2

Crossbar control for fast contention resolution.

reduced by using structures that have little or no contention for paths, such as crossbars, and by directing data flow (at least most of the time) over paths that have no buffering, few chip crossings, and few levels of logic. These paths can be associated with either crossbars or multistage networks.

Crossbars typically have few chip crossings and the nonblocking property. In its simplest form, a crossbar can be viewed as a grid of crosspoints in which a single crosspoint is activated to connect an input to an output. Because of the pin limitations of chip packages, large crossbars must be partitioned into smaller subnetworks. For example, the crossbar of Figure 1(a) is partitioned into four 4 × 4 subnetworks. The partitions can be interconnected by means of buses [3], so that an input can be connected to an output by the activation of a single partition. This implies that the data-transfer delays of a large partitioned crossbar could be as small as those of a single-stage network. The number of crossbar partitions that can be interconnected by a single bus depends on the fan-out characteristics of the technology used for the partition implementation. With current VLSI technology, crossbar partitions can be interconnected by means of a single bus for fairly large shared-memory multiprocessor systems (e.g., $N = 512$).

As previously pointed out, crossbars are sometimes regarded as impractical because of the $O(N^2)$ growth of the number of crosspoints. However, the speed of control for large crossbars can present a more severe problem. An $N \times N$ crossbar can be controlled by a single controller that serially responds to connection requests received from the ports. (As many connections as desired may exist simultaneously.) But this approach can result in long queuing delays when N is large. Alternatively, the fastest control possible may be achieved by providing a controller at each output port and dedicated, direct signal paths from each of the input ports to each of the output port controllers (**Figure 2**). When an input port requires a

connection to a particular output port, it sends a request message to the controller of the output port. The controller resolves any possible contention for the port, due to requests received simultaneously from multiple input ports, and sets up a data path through the crosspoints. This scheme requires N outgoing control lines from every input port, and N incoming control lines to every output port. The resulting wiring complexity is of $O(2N^2/P)$, where P is the number of chip package pins allowed by the implementation technology.

By contrast, the data-transport part of the crossbar, which can be, as previously discussed, a matrix of simple crosspoint switching elements, requires only $O[(2N/P)^2]$ chips. This is a dramatic difference in complexity. For example, with $N = 512$ and $P = 256$ (a typical number of pins with modern packaging technologies), data transport would require approximately 16 chips, while fast contention resolution would require of the order of 2000 chips.

An alternative approach to the scheme outlined above shares one controller among multiple output ports. This reduces the wiring complexity but requires the introduction of additional complexity in the messages, e.g., addressing information, which causes transfer and processing delays. Furthermore, the sharing of a controller by multiple ports results in queuing delays as well.

Multistage networks, on the other hand, are structured so that data transmissions must pass through $\log_k N$ sequential network stages, each of which is a $k \times k$ crosspoint switch. The network of Figure 1(b) has three stages of 2×2 crosspoint switches. The network stages are connected by point-to-point interconnections, unlike the bus interconnections of partitioned crossbar networks; thus, they are not constrained by the fan-out characteristics of the technology. But the performance of the network is affected by its blocking nature and by the data-transfer delays between the stages. The latter can be a significant factor, as the transmission speed of the off-chip data paths is considerably less than that of the on-chip paths (typically, smaller by an order of magnitude). This is primarily due to transmission-line capacitances and the simultaneous off-chip driver (OCD) switching limitations of chip packages [4]. The blocking effects of the multistage network can be reduced by utilizing buffers at each stage. But the buffers, and their associated circuitry, increase the overall complexity of a stage, thus limiting the number of stages that can be integrated on a single chip. This results in a greater number of chip crossings and, consequently, in an increase in the delays associated with the data transfer.

Networks with no buffers in which messages are retransmitted upon contention can work well if the traffic is sufficiently low. However, in general, there will be traffic nonuniformities in space as well as time (e.g., "hot spots" [5]). It has been shown [6] that under the former condition bufferless networks can have problems—i.e., the number of required retries can be large. This forces the source to keep trying until a successful transmission takes place, and may lead to long delays. It also carries risks associated with instabilities in the case of a large number of contending sources [7].

The problems of slow performance in multistage networks and control complexity in crossbars can be alleviated by employing a structure that uses a combination of networks. The result can be viewed as a hierarchy of paths (studied more abstractly in [8]), for which we show that the latency can be, on average, close to that of the fastest path. The performance of the network hierarchy is analogous to that of a storage hierarchy, whose average performance in a good design is largely determined by the speed of the fastest component.

Aside from the above network traffic problems, there may also be congestion associated with saturation of a port that contains a hot spot [5]. Here, combining of memory operations (e.g., via fetch-and-adds) has been suggested as an approach. This, however, requires a network of substantial complexity and, therefore, delay. This complexity is not required under normal traffic conditions—that is, conditions under which requests would traverse the fast path in the hierarchy (discussed below).

The path hierarchies we consider here have only two levels, but this need not be true in general. Also, path hierarchies are not necessarily limited to cases that include a crossbar. Considerations similar to those we discuss below may lead to hierarchical organizations for other classes of networks, but we have concentrated on uniform-distance networks. Some preliminary results of the hierarchical-network approach are presented in [9].

3. Example of a hierarchical network

- *Network organization*

For the purpose of this discussion, we assume that the network outlined in the Introduction interconnects a distributed, shared-memory system. The interconnection network consists of a control-signal network and a data-transport network. The control-signal network is a hierarchical network of the sort discussed throughout this paper. The data-transport network is simply a crossbar, referred to below as the data-transport crossbar.

In this section we illustrate the hierarchical-network principles by means of an example, shown in **Figure 3** (for $N = 512$). This network consists of three subnetworks, each of 512 inputs and outputs: a collision crossbar, a return crossbar, and a Delta network.

These subnetworks are used to form a two-level hierarchy of paths. A fast path is provided by the collision crossbar, which can reliably deliver requests from a

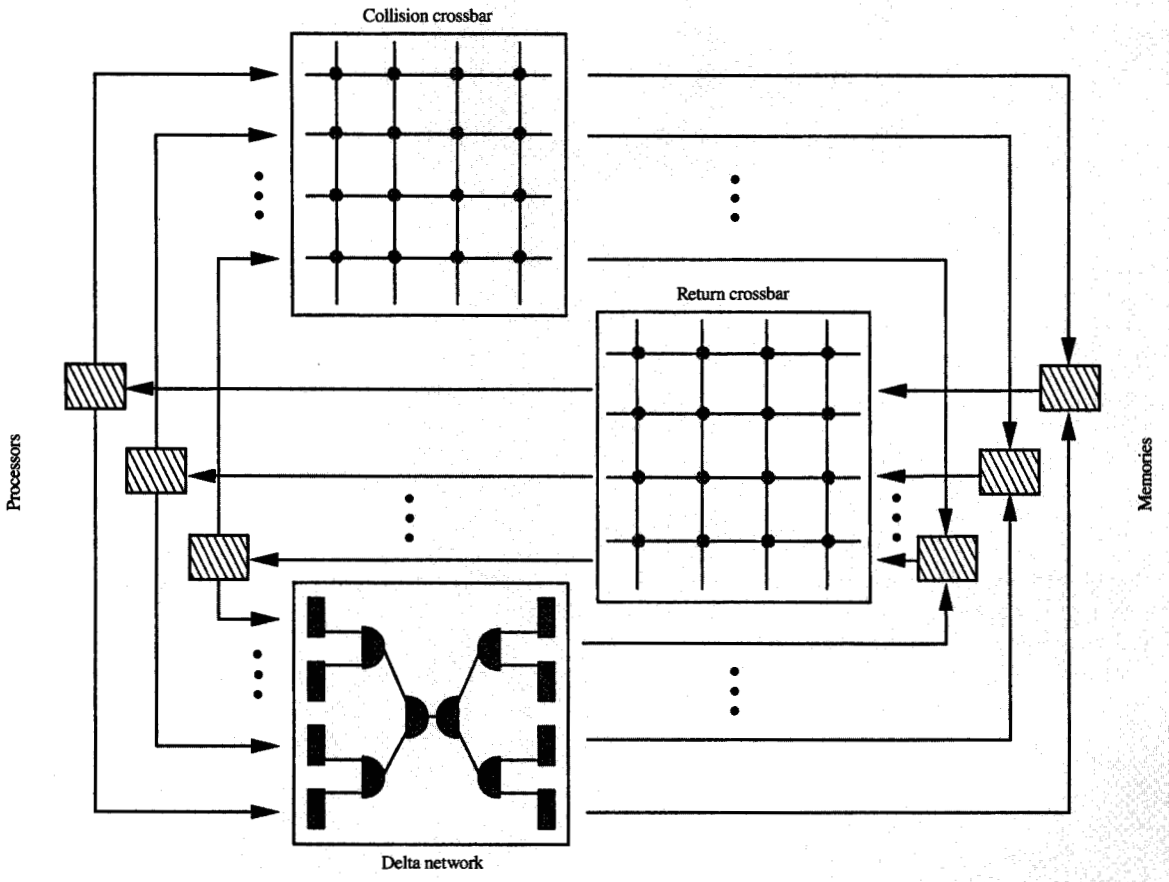

Figure 3

Example of a hierarchical network.

processor to a memory only in the absence of contention. A slow path is provided by the Delta network, which guarantees the delivery of messages to memories under all traffic conditions. A return path for sending acknowledgments from memories to requesting processors (indicating that the requests were properly received) is provided by the return crossbar. The transmission of acknowledgment messages is simultaneous with other memory operations, such as accessing, in order to minimize the protocol delays in the fast path. Data are sent over the data-transport crossbar.

The collision crossbar performs collision detection but not contention resolution. Contention in the collision crossbar by more than one input (processor) for a path to the same output (memory module) is detected at the output; thus, the complexities associated with contention resolution, discussed in Section 2, are avoided. Collision detection can be done by detecting errors in the transmitted bit stream [e.g., CRC (cyclic redundancy check) errors, or transmission-code violations] [10].

The data-transport and return crossbars have neither contention-resolution nor contention-detection mechanisms, because of the absence of collisions in the return path from the memories to processors. This is due to the nature of our protocol, in which only one outstanding request from a processor to a memory is allowed at a given time.

The Delta network is a multistage network of $\log_k N$ stages with store-and-forward capability at each stage, such as the Omega network found in [1].

• *Communication protocol*

All communication in the example network is done via messages, which have one of the two formats shown in **Figure 4**:

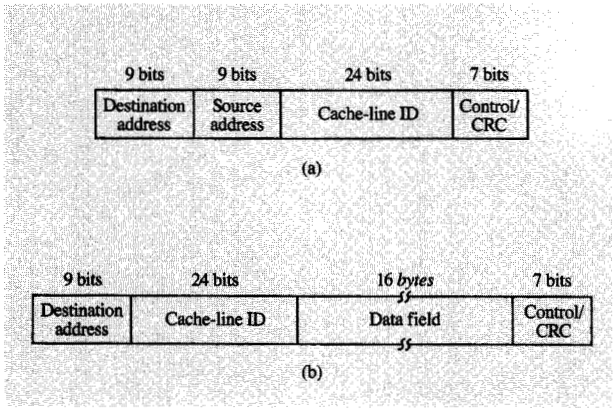

Figure 4

Formats of network messages: (a) request message; (b) data message.

1. *Request messages* consist of a destination-address field of $\log N$ bits, where N is the number of processors and of memories (in this example it would be 9 bits); a source-address field (9 bits); a cache-line ID field (24 bits); a field containing control information, for indicating the type of memory operation requested (read or write); and error-detection code (7 bits)—for a total of approximately 6 bytes.
2. *Data messages* (sent over the data-transport crossbar) consist of a destination address (9 bits); cache-line ID field (24 bits); data field (16 *bytes*); and CRC field (7 bits)—or approximately 21 bytes.

The sizes of the data and cache-line ID fields may vary, depending on system design considerations.

Figure 5 shows the network protocol for memory read operations. A request from a processor, generated at time t_0, enters the collision crossbar (CC) synchronously, under the control of a global clock (at time t_1).

In the case of *noncontention*, the request is received at the memory at time t_2, when a determination is made as to whether a collision has occurred. Two concurrent operations are initiated at time t_3: 1) the memory access cycle is started, and 2) an acknowledgment message (ACK) is sent back to the requesting processor and received at time t_4. When the data become available from the memory at time t_5, they are shipped to the processor on the data-transport crossbar. The data transmission is complete at time t_6.

In the case of *contention*, collision is detected and the request is ignored; consequently, no acknowledgment

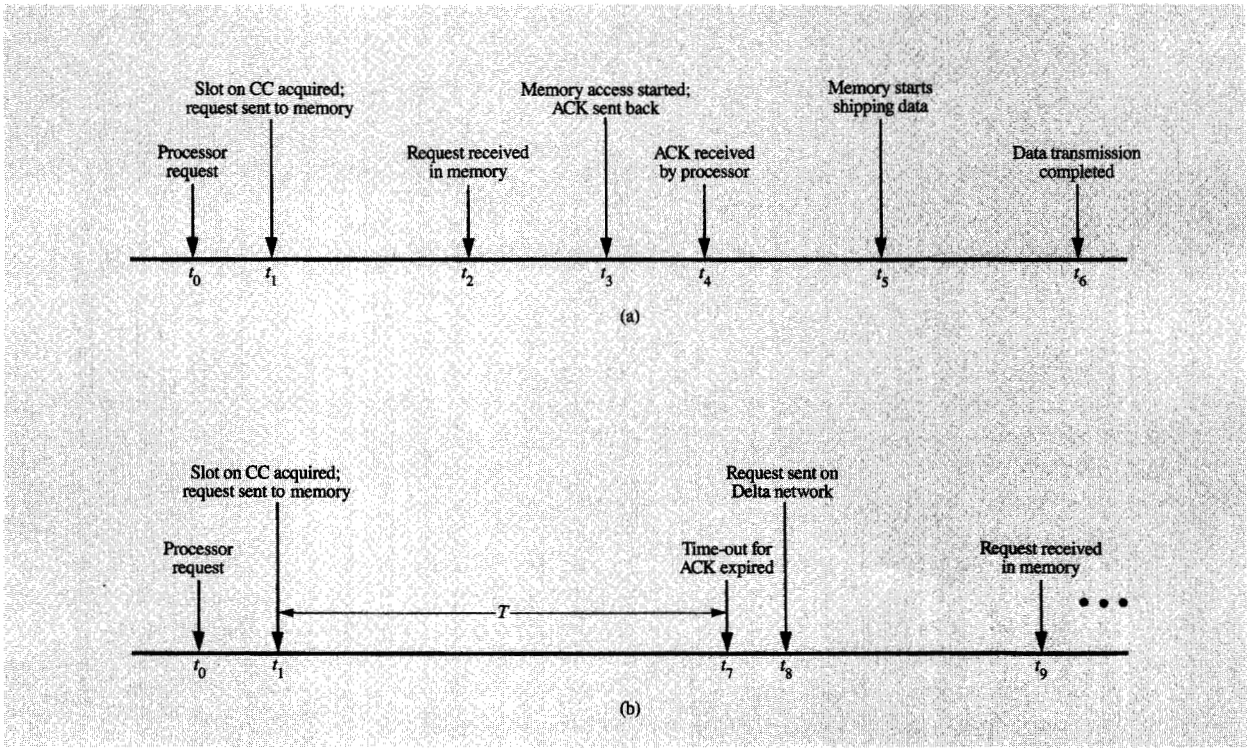

Figure 5

Hierarchical network protocol: (a) no contention; (b) contention. CC = collision crossbar.

message is returned to the requesting processor from the memory. The processor, upon expiration of a time interval T (at t_7) during which it expected to receive the acknowledgment, retransmits the request over the Delta network (time t_8). The request is received at the memory at time t_9, after which the same sequence of steps t_3–t_6, as described in the "noncontention" case, is repeated.

• *Self-routing crosspoint chip*

The basic building block of the collision and other crossbars is the self-routing crosspoint chip, shown in **Figure 6**. The crosspoint chip is organized as an array of crosspoints C_{ij}. A message enters the chip, as a series of bits, from an input pin I_i and is directed to an address decoder circuit A_i. The address bits of the message are then decoded and select an output O_j by activating a crosspoint C_{ij}. After a path is thus established between an input and an output, the message, stripped of its destination address bits, is sent to the destination.

The bandwidth of the serial path is dependent on the technology used to implement the crosspoint chip. If the bandwidth provided by a single chip is insufficient, the required bandwidth can be obtained by using multiple chips in parallel (i.e., several switching planes). For example, if k chips are used, a message of n address and m data bits can be transformed into k frames, each consisting of n address and m/k data bits. Each frame can then be sent via a separate crosspoint chip, with the data bits from all frames assembled at the destination to reconstruct the original message. Techniques exist for providing the clock distribution and synchronization required [11].

Since contention in the collision crossbar is detected at the destination, the output circuits of the crosspoint chips must be designed to protect against current overload conditions, thus preventing collisions from damaging the chips.

A variation of the above scheme to protect against current overload would be to provide collision-detection circuits at the outputs of each crosspoint chip. These circuits would monitor the status of their corresponding bus lines and, if traffic or collisions were detected, would signal the source to abort the transmission and retransmit on the slow path.

4. Performance and complexity estimates

The following analysis of the example network shows the feasibility of the hierarchical approach. The basic objective of this study is to show that with a currently available technology such as CMOS—a dense but low-power technology—we could build a network of size 512. The goal of this evaluation is to design a network with reasonable hardware complexity that can transfer 16-byte data words within five processor cycles, on the average.

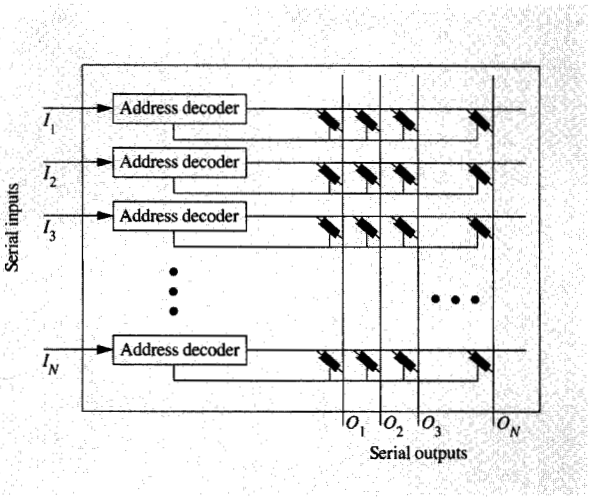

Figure 6

Self-routing crosspoint chip.

• *Network delay*

The message delay through the network depends on the delays incurred in the fast and slow paths. As previously discussed, if contention occurs at the fast path (collision crossbar), the message is retransmitted over the slow path (Delta network).

If we assume no contention for the networks, the network delay, as illustrated in **Figure 7**, can be derived as follows: We assume that the collision and return crossbars have eight and ten switching planes, respectively. We also assume that a memory request can be sent to the collision crossbar at the beginning of a transmission synchronization cycle. The length of the transmission synchronization cycle is four network global clock cycles. Since the memory requests can occur randomly with respect to the beginning of the synchronization cycle, we assume a 50% average delay for the beginning of a transmission, or two network cycles. The destination address header of a message (9 bits) requires nine cycles to enter the collision crossbar and be decoded. The remaining five message bytes for a request message require five cycles (over the eight switching planes) to be transferred to the destination. Thus, a message requires an average of 16 cycles to traverse the collision crossbar. (Circuit delay of the crossbar itself is negligible.)

We further assume that the memory access requires the equivalent of eight network cycles and that the assembly of the data message requires two cycles. This activity is overlapped with the transmission of the acknowledgment message to the processor via the return crossbar and with the set-up of the path for the transfer of the data message in the data-transport crossbar. Finally, the data-message

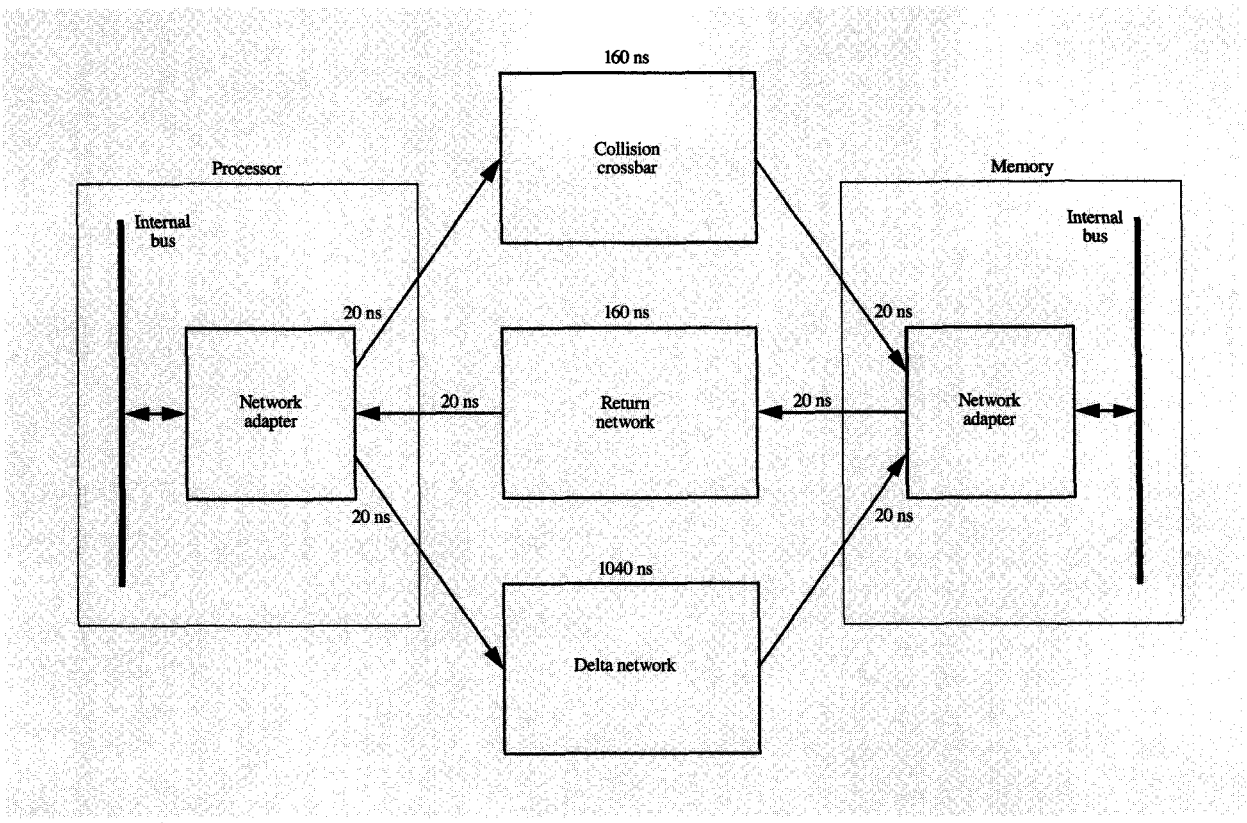

Figure 7

Network delay.

transfer on the data-transport crossbar over ten switching planes requires 16 cycles (20 bytes). The total number of network cycles is, therefore, 16 + 10 + 16 = 42 plus the cable delay. We assume 20-ns cable delays between each network adapter and the crossbar. If we assume a cycle time of 10 ns, the total transfer delay is 42 × 10 ns + 4 × 20 ns = 500 ns.

- *Complexity estimates*

We assume that the basic building block of the network is a 128 × 128-crosspoint chip. Such a chip would require a 300-pin package, which is well within the capabilities of current technology [12]. A 512-port collision-crossbar plane, therefore, could be constructed with a 4 × 4 array of chips. Thus, for a design with eight planes, the collision crossbar would require 128 chips. Likewise, the return crossbar could be constructed with 16 chips (one plane). An additional 160 chips are required for the data-transport crossbar (ten planes). We assume that the Delta network could be constructed with 100 chips for a design such as that found in [1], but with a denser and much slower technology. Thus, the network structure, consisting of the above four subnetworks, could be built with a total of 372 chips packaged on 24 printed-circuit boards. It would occupy a volume of three cubic feet.

5. Performance modeling and analysis

The performance of the hierarchical network is a function of the collision probability. The purpose of this section is to model the hierarchical interconnection network, calculate the collision probability for various system parameters, and evaluate the overall system performance.

- *Model description*

We model the multiprocessor system by the closed queuing network shown in **Figure 8**. The population in the queuing network corresponds to the number of processors, which is denoted by N. There are four queuing models: one for the processors, one for the memory modules, one for the forward-network path to memory, and one for the reverse-network path back to the processor. The queuing

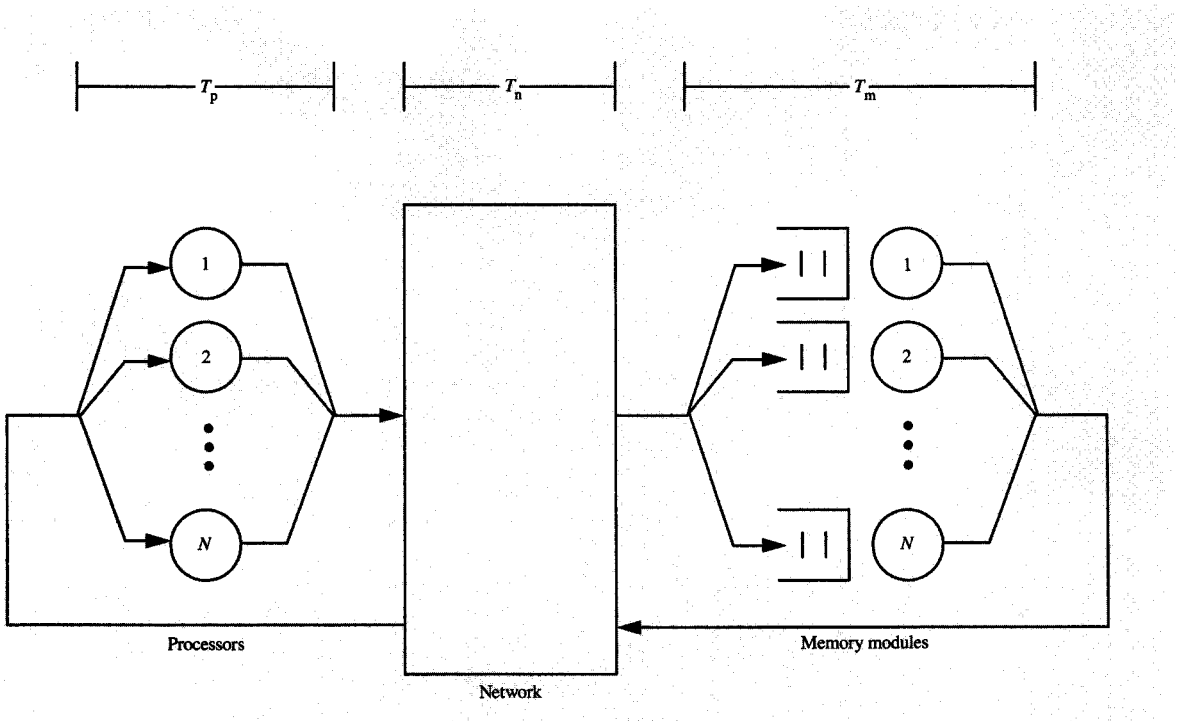

Figure 8

Queuing network model.

models for the processors and the network are delay models. Memory modules are modeled by independent single-server queuing systems.

The flow of a request in such a closed queuing network model is described as follows. After spending time at a processor corresponding to the processing time between two consecutive cache misses (with mean T_p), a request leaves the processor and travels over the forward-network path to one of the memory modules. The request is queued at the memory module in question until it receives its service and then returns to the originating processor through the reverse-network path. We denote the mean total network delay, including forward and reverse paths, by T_n and the mean memory delay, consisting of both queuing and service at a memory module, by T_m. The mean cycle time of a request in this closed queuing network (denoted by $1/\lambda$) is therefore given by the sum

$$1/\lambda = T_p + T_n + T_m, \tag{1}$$

and the system throughput (memory access rate) equals $\Lambda = N\lambda$.

Expressions for the different delays are derived in [13]. The mean processing delay T_p is given by

$$T_p = \frac{C_p}{\theta\beta}, \tag{2}$$

where C_p is the average processor instruction execution time, θ is the average number of memory references per instruction, and β is the probability that a memory reference to the local cache is a miss. Let ρ_p denote the processor utilization, which is defined as the fraction of time that the processor is busy and not waiting for a cache miss to be satisfied. The processor utilization is given by

$$\rho_p = \lambda T_p. \tag{3}$$

In order to derive the network delay T_n, we consider three types of network models, as shown in **Figure 9**: an ideal network, a fixed-delay network, and a hierarchical network. A comparison of the performance of systems utilizing these three types of networks will help us evaluate the relative merits of hierarchically interconnected multiprocessor systems. An ideal network is a network without any delays ($T_n = 0$), and serves as a point of reference. A fixed-delay network consists of constant delays for the forward and reverse paths, which are denoted by D_f and D_r, respectively. (Obviously, the

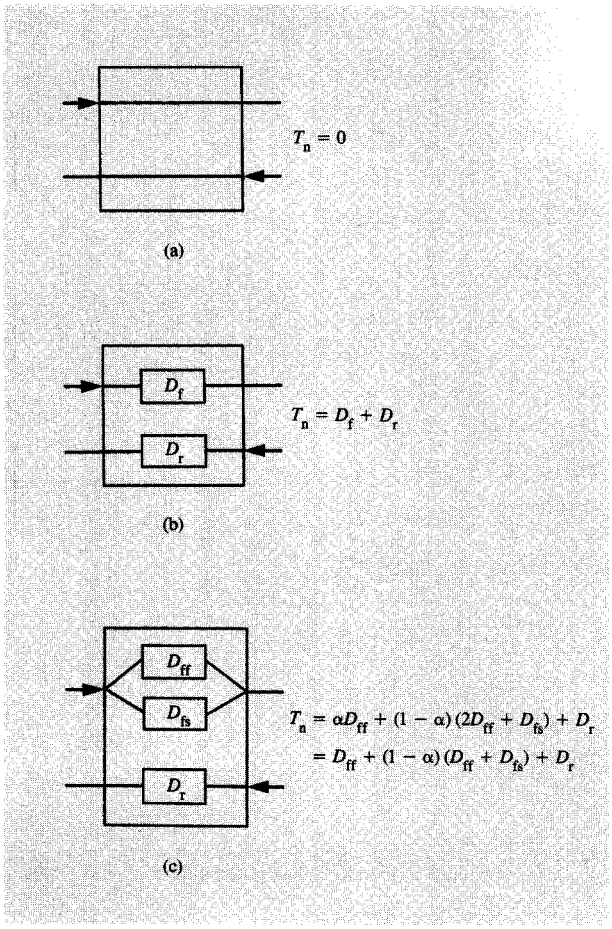

Figure 9

Network models: (a) ideal network; (b) fixed-delay network; (c) hierarchical network.

network delay is a nondecreasing function of the traffic. But the assumption that the network delay is simply the network service time permits us to derive an upper bound on the network performance.) The third type of network model that we consider is a hierarchical network. The forward path of this network, as previously discussed, consists of a hierarchy of two paths: a fast path with average delay D_{ff} and a slow path with an average, traffic-independent delay D_{fs}. We may let D_{fs} be the minimum network delay in order to determine an upper bound on network performance. We denote by α the probability that a request succeeds in taking the fast path. This event corresponds to the case in which only one request destined for a particular memory module is generated in a given network synchronization cycle. If requests collide, they all receive negative acknowledgments, after a round-trip delay of $2D_{ff}$, and then use the slow path. We show that since the probability α is high, the traffic flowing over the slow path is rather light. Consequently, it is quite reasonable to neglect queuing effects in the slow network.

Thus, the network delay may be written as

$$T_n = \alpha D_{ff} + (1 - \alpha)(2D_{ff} + D_{fs}) + D_r$$
$$= D_{ff} + (1 - \alpha)(D_{ff} + D_{fs}) + D_r, \quad (4)$$

where D_r is the average delay of the reverse network path, which is assumed to be a crossbar switch. Since we assume that we have at most one request outstanding per processor, collision cannot occur on the reverse path. The value of α may be approximated [12] by

$$\alpha \simeq 1 - p(\rho_p + \delta), \quad (5)$$

where p is the probability that a request is generated from an active processor during a given network synchronization cycle and

$$\delta = \lambda C_n/2. \quad (6)$$

(C_n is the network synchronization cycle discussed in Section 4.) Approximating the processor delay by an exponential distribution, we get

$$p = 1 - e^{-C_n/T_p}.$$

A memory module is modeled by a single-server queuing system. We assume that memory modules are pipelined, so that a constant memory service time, which is denoted by S, consists of an initial service time S_0 to access the first portion of the data being fetched, and a number of service epochs, each of length S_1, to access consecutive portions of the data. For example, if a cache line is 16 bytes long and the unit of memory access is four bytes, after a delay of S_0 the first four bytes are available from memory, and subsequent four-byte portions become available every S_1 time units afterward. Thus, a total of $S_0 + 3S_1$ time units constitute the memory service times. In this case, we assume that the processor becomes active as soon as the first four bytes are transferred over the return-network path. We approximate the behavior of a memory module by that of an M/D/1 queuing system with mean response time [13]

$$T_m = \frac{\rho_m S}{2(1 - \rho_m)} + S_0, \quad (7)$$

where $\rho_m = \lambda S$ is the memory utilization.

By using Equations (3), (5), and (6) and substituting Equations (4) and (7) into Equation (1), we obtain a cubic equation in ρ_m,

$$u\rho_m^3 + \left(v - u - \frac{1}{2}\right)\rho_m^2 - (v + 1)\rho_m + 1 = 0, \quad (8)$$

where

$$u = \frac{p\left(T_p + \frac{C_n}{2}\right)(D_{ff} + D_{fs})}{S^2}$$

and

$$v = \frac{1}{S}(D_{ff} + D_r + S_0 + T_p).$$

Equation (8) has a unique root in the interval [0, 1].

- *Model validation*

To assess the error due to our model assumptions and analysis approximations, we simulated a system with 64 processors [13]. We considered two types of networks: a buffered multistage interconnection network (MIN) and a hierarchical network. In the first case, the network consisted of 4 × 4 switches with a maximum buffer size of four requests per port. In the second case, the hierarchy consisted of a 64 × 64 collision crossbar and a slow buffered MIN. In both cases, the return network was a crossbar. We obtained the relative disparity between our analytic model and simulation results for various values of the cache-miss probability. Our model underestimates the network delay, since it does not take the network congestion into consideration. From simulation, we found the amount of underestimation to be at most 7%. Our model overestimates the memory delay by at most 6%. This discrepancy is due primarily to the assumption of Poisson arrivals at the memory modules and the infinite-queue model. The relative disparity in system throughput, hence in the processor and memory utilizations, is at most 5%. For a typical value of miss probability of 0.06, the relative disparity between model and simulation for both types of networks (MIN and hierarchical) is 2–3%.

- *Performance analysis*

Using the model and analysis presented in the previous section, we evaluate the performance of multiprocessor systems with hierarchical networks. It is interesting to compare the performance of such systems to systems in which the interconnection network is either ideal (zero delay) or nonhierarchical. For fixed-delay networks, we consider two extremes: a fast network and a slow network, with forward-path delays D_f = 200 ns and D_f = 360 ns, respectively. The fast network corresponds to a crossbar switch that uses CMOS technology, whereas the slow network corresponds to a buffered Delta network that uses a faster technology, such as bipolar. The hierarchical network has a forward path that consists of two levels: a fast path similar to the fast network with delay D_{ff} = 200 ns, and a slow path that is slower than the slow network (D_{fs} = 1080 ns) and uses the same technology as the fast path. In all networks, we assume the return-path and forward-path delays to be the same, i.e., $D_r = D_f$, or $D_r = D_{ff}$.

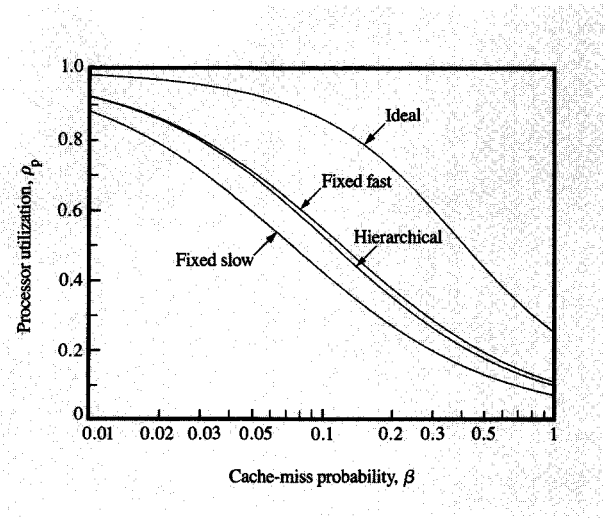

Figure 10

Effect of cache-miss probability on processor utilization.

We assume that the cycle times of the processors and the network synchronization cycle are C_p = 100 ns (i.e., 10-MIPS processors) and C_n = 40 ns (see Section 4), respectively, and that the memory service times are S_0 = 80 ns and S_1 = 20 ns, with a 4-byte access unit and 16-byte cache lines.

We consider two cases of traffic distribution: uniform traffic, in which memory modules are selected with equal probabilities, and skewed traffic, in which one of the modules is selected with a higher probability than the other modules.

Uniform traffic distribution

System performance is evaluated as a function of the cache-miss probability β. In **Figure 10**, we plot the processor utilization ρ_p as a function of β for the various types of interconnection networks. As β increases, the request rate (the required network throughput) increases; thus, memory contention increases, resulting in poor performance. Since the processing time decreases, the effective use of the processors decreases. This is evident in the case of the ideal network, in which requests are either at the processors or memory modules. For example, when β = 1, about 25% of the requests are at the processors and the remaining 75% are either waiting at the memory module or being serviced there. By comparing the hierarchical network and the fixed fast network, we find that, for low β, both networks exhibit almost identical

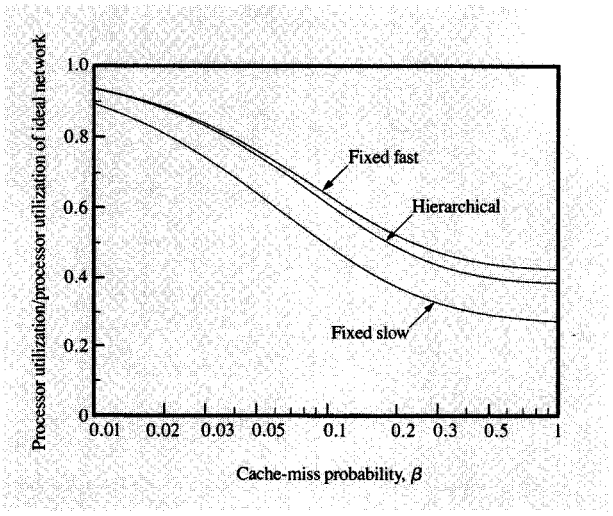

Figure 11

Processor utilization relative to ideal networks.

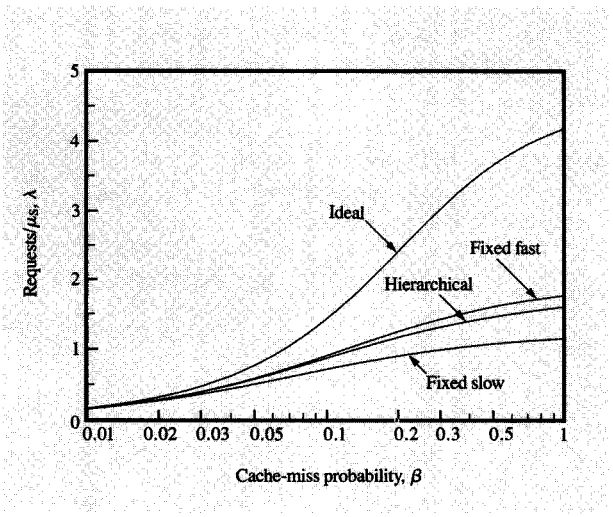

Figure 12

System request rate.

performance. This is due to the very small collision probability, which causes very little traffic to flow over the slow path of the hierarchy. It is also interesting to note that even for high values of β, the processor utilization for the hierarchical network is higher than that of the fixed slow network, even though the network in the latter is three times faster than the slow path in the former. This suggests that the collision probability remains small even for high traffic rates, so that the existence of the fast path in the hierarchical network overcomes the fact that its slow path is quite slow.

In **Figure 11**, we plot the ratio of processor utilization to that of an ideal network. Typically, the miss probability is around 0.06 [13]. From the figure we see that for $\beta = 0.06$ the performance of a hierarchical network is 72% of that of an ideal network. For $\beta = 1.0$, this decreases to about 40%. For small values of β, the performance of a hierarchical network approaches that of a fixed fast network.

The request rate λ is plotted as a function of the miss probability β in **Figure 12**. For small β, memory references are mostly to cache; therefore, the request rate is small. For medium values of β, the request rate increases almost logarithmically with β for all network types. It is interesting to note that both the hierarchical and fixed fast networks have almost identical request rates for $\beta < 0.1$ and that the throughput of the hierarchical network is uniformly higher than that of the fixed slow network (the Delta network).

Another quantity of interest is α, the probability that a request does not collide with other requests going to the same memory destination during the same network synchronization cycle. This is a crucial performance measure for hierarchical networks. In **Figure 13**, we plot α as a function of the cache-miss probability β. We note that α is always greater than 0.93; i.e., at least 93% of memory references succeed in using the fast path, and at most 7% use the slow path. Again, this shows the validity of the assumption of our model that queuing delays due to using the slow path are negligible.

The effect on network performance of other factors, such as line length, processor speed, and memory speed, is described in [13].

Skewed traffic distribution

In the previous section, we assumed that memory references are uniformly distributed among the different memory modules. In fact, the memory access pattern may be skewed, due to the possibility of a "hot spot," where a large number of processors refer to the same memory location [5]. To model such a skewed access pattern, we assume that a fraction γ of all memory references is destined to a particular memory module, which we refer to as a "hot module," or *HM*. The remaining fraction $(1 - \gamma)$ is uniformly distributed among *all* memory modules. Thus, the access rate of the HM is given by

$$\lambda^{HM} = \Lambda\left(\gamma + \frac{1-\gamma}{N}\right),$$

where N is the number of processors, as well as the number of memory modules, and Λ is the total system throughput. The HM utilization, which is denoted by ρ_m^{HM}, is obtained by multiplying λ^{HM} by the memory service time S, yielding

$$\rho_m^{HM} = [1 + (N - 1)\gamma]\rho_m , \tag{9}$$

where $\rho_m = \Lambda S/N$ denotes the average memory utilization.

By multiplying T_p [given by Equation (2)] by λ, we obtain the processor utilization

$$\rho_p = \frac{\lambda C_p}{\theta\beta} = \frac{C_p}{S\theta\beta}\rho_m . \tag{10}$$

Substituting ρ_m from Equation (9) into Equation (10) yields

$$\rho_p = \frac{C_p}{S\theta\beta}\frac{\rho_m^{HM}}{[1 + (N - 1)\gamma]},$$

which gives the processor utilization as a function of the utilization of the "hot module." Since $\rho_m^{HM} < 1$, an upper bound on the processor utilization is given by

$$\rho_p < \frac{a}{\beta}, \tag{11}$$

where $a = C_p/\{S\theta[1 + (N - 1)\gamma]\}$ is a constant. Equation (11) is a very interesting relationship between the processor utilization bound ρ_p and the miss probability β when the "hot module" is busy with probability close to 1. In such a case, ρ_p is inversely proportional to β. This behavior is depicted in **Figure 14**, for a system with $N = 512$ processors, where we evaluate ρ_p for γ varying from 0 to 0.1. The case $\gamma = 0$ corresponds to a uniform memory access pattern (as displayed in Figure 11). As γ increases, we find that ρ_p, a function of β, approximates a/β, given by Equation (11). This suggests that the utilization of the "hot module" approaches its upper limit of 1. It is interesting to note that even for a small value of γ, system performance is degraded quite significantly for a wide range of values of the miss probability.

The analysis presented above shows that the processor utilization is limited by memory rather than communication bandwidth. It also supports the suggestion [2] to incorporate a fetch-and-add network into a connection hierarchy with an average performance close to that of a network with no combining. This may be an effective way to handle "hot spots" without much penalty to overall system performance.

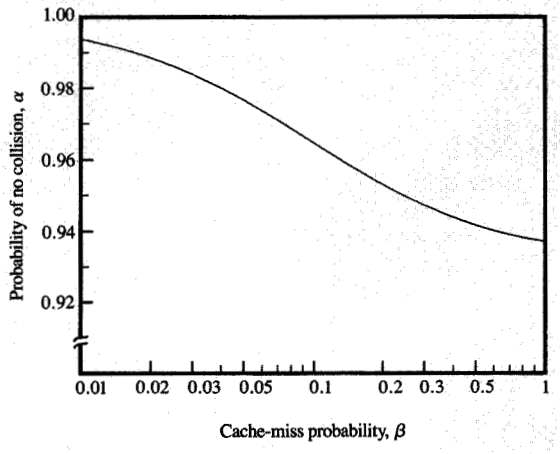

Figure 13

Probability of no collision for hierarchical networks.

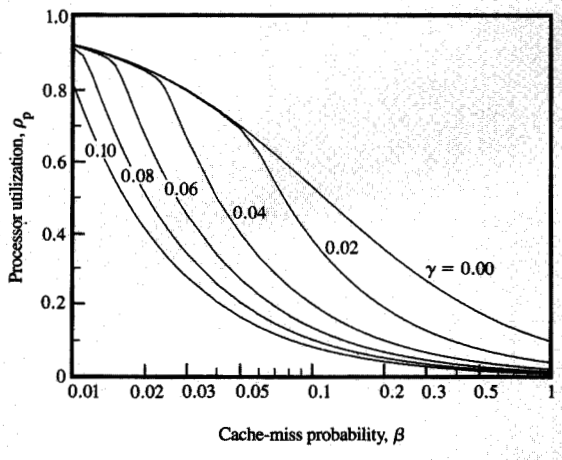

Figure 14

Effect of "hot spots."

6. Conclusions and summary

In this paper, we have presented a new approach to interconnection network design for high-performance multiprocessors. This approach assumes a hierarchy of subnetworks, each with different performance and complexity characteristics, that is analogous to a memory hierarchy. We showed that the complexity of the control mechanism is the primary limiting factor in large crossbar networks, whereas performance is the limiting factor in large multistage networks, and that a combination of networks in a hierarchy of paths can alleviate such problems. We studied an example hierarchy, consisting of crossbar and multistage subnetworks, that is realizable with current VLSI technology. Through modeling and analysis, we investigated the performance of the example hierarchical network as a function of several parameters. We showed that its performance is close to that of a fast crossbar network.

We believe that network control (i.e., buffering, contention resolution, and pacing) may be a limiting factor in multistage networks, as well. This suggests the possibility of a hierarchy of multistage networks, with the fastest having no buffers.

References

1. G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss, "The IBM Research Parallel Processor Prototype (RP3): Introduction and Architecture," *Proceedings of the 1985 International Conference on Parallel Processing*, St. Charles, IL, 1985, pp. 764–771.
2. L. R. Goke and G. J. Lipovski, "Banyan Networks for Partitioning Multiprocessor Systems," *Proceedings of the First Annual Symposium on Computer Architecture*, University of Florida, Gainesville, FL, 1973, pp. 21–28.
3. C. J. Georgiou, "Fault-Tolerant Crosspoint Switching Networks," *Proceedings of the 14th International Conference on Fault-Tolerant Computing (FTCS 14)*, Kissimmee, FL, June 1984, pp. 240–245.
4. E. E. Davidson, "Electrical Design of a High-Speed Computer Package," *IBM J. Res. Develop.* **26,** 349–361 (1982).
5. G. F. Pfister and V. A. Norton, " 'Hot Spot' Contention and Combining in Multistage Interconnection Networks," *IEEE Trans. Computers* **C-34,** 943–948 (1985).
6. P. Heidelberger and P. A. Franaszek, "Traffic Studies of Unbuffered Delta Networks," *IBM J. Res. Develop.* **35,** 288–299 (1991).
7. W. A. Rosenkranz, "Some Theorems on the Instability of the Exponential Back-off Protocol," *Performance '84*, Elsevier Science Publishers B.V., 1984, pp. 199–205.
8. P. A. Franaszek, "Path Hierarchies in Interconnection Networks," *IBM J. Res. Develop.* **31,** 120–131 (1987).
9. P. A. Franaszek and C. J. Georgiou, "Multipath Hierarchies in Interconnection Networks," *Proceedings of the First International Conference on Supercomputing, ICS '87*, Athens, Greece, June 8–12, 1987; *Lecture Notes in Computer Science*, No. 297, Springer-Verlag, New York, 1987.
10. P. A. Franaszek and C. J. Georgiou, "Collision Crossbar Switch," U.S. Patent 4,929,940, 1990.
11. D. G. Messerschmitt, "Synchronization in Digital System Design," *IEEE J. Selected Areas in Commun.* **8,** 1404–1419 (1990).
12. W. Larkins, S. Canaga, G. Lee, W. Terrell, and I. Deyhim, "13,000 Gate ECL Compatible GaAs Gate Array," *Proceedings of the IEEE 1989 Custom Integrated Circuits Conference*, San Diego, CA, 1989, pp. 15.7.1–15.7.4.
13. A. N. Tantawi, "Performance of a Hierarchically Interconnected Multiprocessor," *Proceedings of the Tenth International Conference on Distributed Computing, ICDCS-10*, Paris, May–June 1990, pp. 352–359.

Received October 2, 1990; accepted for publication April 25, 1991

Peter A. Franaszek *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.*
 Dr. Franaszek is manager of Systems Theory and Analysis in the Computer Sciences Department at the Thomas J. Watson Research Center. He received an Sc.B. degree from Brown University in 1962, and M.A. and Ph.D. degrees from Princeton University in 1964 and 1965, respectively. Dr. Franaszek's interests include analytical and design issues in computer system organization, algorithms, and communication networks and coding. He has received IBM Outstanding Innovation Awards for his work in the areas of algorithms, interconnection networks, concurrency control theory, and constrained coding. Dr. Franaszek was also the recipient of two IBM Corporate Awards for his work in the latter area. In 1991, he was elected to the IBM Academy of Technology. He was named the recipient of the 1989 Emanuel R. Piori Award of the Institute of Electrical and Electronics Engineers for his contribution to the theory and practice of digital recording codes. During the academic year 1973–1974, he was on sabbatical leave from the Thomas J. Watson Research Center to Stanford University as a Consulting Associate Professor of Computer Science and Electrical Engineering. Prior to joining IBM in 1968, he was a member of the technical staff at Bell Telephone Laboratories. Dr. Franaszek is a member of Tau Beta Pi and Sigma Xi, and a Fellow of the IEEE.

Christos J. Georgiou *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.*
 Dr. Georgiou received the B.S. degree in electrical engineering and the Ph.D. degree in electrical and computer engineering from the University of California at Santa Barbara in 1974 and 1981, respectively, and the M.S. degree in electrical engineering and computer science from the University of California at Berkeley in 1975. From 1976 to 1978, he worked on the design of floppy-disk-based communication subsystems at Scientific Micro Systems, Mountain View, California. From 1978 to 1981, he was the co-founder and engineering manager at Voicetek, Goleta, California, responsible for the development of a line of speech-recognition and voice-response peripherals for personal computers. In 1982, he joined the IBM Thomas J. Watson Research Center as a research staff member; in 1984, he became manager of the System Interconnection Structures group, conducting research on high-performance switching systems, interconnection networks for parallel processing, and multiprocessor system architectures. During the academic year 1988–1989, Dr. Georgiou was on sabbatical leave from the Research Division as a visiting professor at the National Technical University, Athens, Greece, and as a visiting research scientist with the Institute of Microelectronics at the Demokritos Research Center, also in Athens. Dr. Georgiou has received three IBM Outstanding Innovation Awards for his work in the area of switching networks and the ESCON™ Director architecture. He is also the recipient of six IBM Invention Achievement Awards, and has more than 20 U.S. patents issued or pending. Dr. Georgiou is a senior member of the IEEE.

Asser N. Tantawi *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598.*
 Dr. Tantawi received the B.S. and M.S. degrees in computer science from Alexandria University, Egypt, and the Ph.D. degree in computer science from Rutgers University in 1975, 1978, and 1982, respectively. He joined the IBM Thomas J. Watson Research Center in 1982 as a research staff member, and is currently manager of Systems Connectivity Performance in the High Bandwidth Systems Laboratory. His fields of interest include performance modeling, queuing theory, load balancing, parallel processing, reliability modeling, and high-speed networking. Dr. Tantawi has received an IBM Outstanding Innovation Award for his work on the ORCHID Communication Subsystem. He is a senior member of the IEEE and a member of the ACM and ORSA/TIMS, and served as an ACM National Lecturer from 1984 to 1988.