JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING **34,** 14–28 (1996)
ARTICLE NO. 0042

A Fault-Tolerant Multistage Combining Network

NENG-PIN LU AND CHUNG-PING CHUNG[1]

Institute of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan 30050, Republic of China

In this paper, we propose a solution to both fault tolerance and hot-spot contention problems in multiprocessor systems with multistage interconnection networks. Combining networks are known to be effective in handling hot-spot traffic. However, the fault tolerance capability of unique-path combining network is insufficient and must be enhanced. Thus, we use the chaining scheme, which provides alternate routing paths by connecting intrastage switching elements with a chain, to enhance the fault tolerance capability of combining network. As a result, we propose a chained combining network. Because of the bidirectionality of combining networks, we also develop routing procedures for the chained combining network. With slight modifications, these routing procedures can also be used in other multipath fault-tolerant combining networks.

1. INTRODUCTION

Multistage interconnection networks (MINs) (as illustrated in Fig. 1) are cost-effective choices within the spectrum of interconnection networks. At a cost of $O(N \log N)$, where N denotes the size of the interconnected nodes, a MIN provides performance close to that of a crossbar network under moderate traffic. However, a basic MIN has two fundamental constraints: First, only a single path exists between each source–destination pair; and second, many source–destination pairs share common links. Two features must therefore be considered in designing a MIN—fault tolerance and traffic contention resolution. Since a failure in any single component in the network disconnects some number of source–destination pairs, fault tolerance capability must be incorporated to provide reliable communications. In a multiprocessor system, a MIN may be used to connect processor elements (PEs) and memory modules (MMs), and traffic load tends to be unevenly distributed within the MIN. For example, under hot-spot traffic conditions, which result from concurrent accesses to the same memory location, traffic contention is particularly serious. Schemes for providing smooth memory accesses under any traffic pattern have to be developed.

In this paper, we consider both fault tolerance and hot-spot contention problems in MINs of multiprocessor systems, and propose a highly reliable, high-performance multistage interconnection network. Various traffic control schemes have been proposed to handle hot-spot traffic [4–7, 17, 19, 24]. Among these schemes, the combining network [5, 6, 17], which can combine requests within a switching element accessing the same memory location, is an effective scheme for handling hot-spot traffic. It not only reduces network traffic, but also provides a fast synchronization rate if some synchronization primitives, such as *Fetch&Add* [6], are supported. However, a combining network is still a unique-path MIN; it lacks fault tolerance capability. To provide this capability, we use a chaining scheme [23] to enhance the combining network. Hence we propose a chained combining network.

This paper is organized as follows: Section 2 gives an overview of fault tolerance and hot-spot contention problems in MINs. To cope with possible faults in the MIN, in Section 3, we present our fault tolerance scheme—chaining, which provides alternate routing paths by connecting intrastage switching elements. In Section 4, through simulations, we examine the behavior of chained networks with various congestion control schemes under hot-spot contention. In Section 5, we propose a fault-tolerant combining network that is augmented via chaining, and derive routing procedures for this network. In Section 6, we examine the overhead of the chained combining network and consider the other fault tolerance schemes for combining networks. In Section 7, we summarize the results of this research.

2. RELATED RESEARCHES

2.1. Fault-Tolerant Multistage Interconnection Networks

In unique-path MINs, any single failure at a switching element or a link may render some outputs unreachable from certain inputs. To overcome this problem, many schemes have been introduced to improve fault tolerance capability. These schemes require some kind of redundancy to be built into a network, where the redundancy could be in the form of information [14], time [20], or hardware [1, 15, 18, 23]. Among these schemes, incorporating redundant hardware to provide multiple paths between

[1] E-mail: cpchung@csie.nctu.edu.tw.

0743-7315/96 $18.00

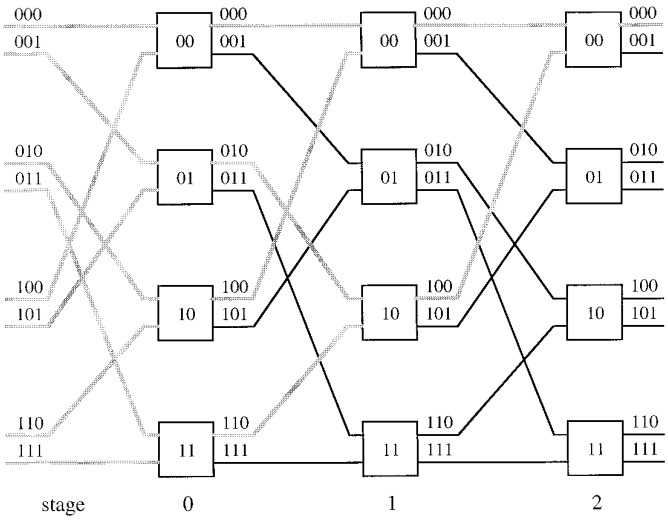


FIG. 1. An example of a multistage interconnection network. (Shaded lines show a fan-in tree.)

every source–destination pair is most widely used in designing fault-tolerant MINs. Of course, all of these schemes can be combined to provide several levels of fault tolerance capability.

To provide the fault tolerance capability in unique-path MINs, several multipath MINs have been proposed. In setting up a connection, multipath MINs allow an alternate path to be chosen, not only if faults have occurred in the network, but also if conflicts with other connections arise. Thus, multipath MINs provide both higher reliability and better performance than unique-path ones. On the other hand, multipath MINs have higher hardware and operational costs than unique-path MINs in terms of the number of stages, the number of switches per stage, or the size of the switching elements. Fault-tolerant MINs have been studied extensively for a long time. A comprehensive survey of fault-tolerant MINs can be found in [2].

2.2. Hot-Spot Contention and Tree Saturation

In a shared-memory multiprocessor, many processors may request the same memory location at the same time. This kind of shared-memory contention due to concurrent requests to a location is called *hot-spot* contention [17]. In practice, there are many potential sources of hot-spot contention, including synchronization instructions, scheduling or shared queue accesses in operating systems, and programs based on machine models that allow concurrent memory accesses. For instance, the *test-and-set* instruction is often used to preserve exclusive data accesses. If many processors need access to the resources controlled by a lock at about the same time, the highly repetitive accesses to the lock caused by the *busy-waiting* can result in contention for this memory location.

A hot-spot traffic model for studying the hot-spot contention problem has been proposed by Pfister and Norton [17]. Assume N is the number of processors in the system, and there are also N memory modules in a shared memory system. Each processor issues r requests to the shared memory per network cycle ($0 \le r \le 1$). Among these requests, a proportion of h are hot-spot requests that access the same memory location (hence the same memory module). Thus, in each network cycle, there are Nrh hot-spot requests and $r(1 - h)$ normal requests directed to the "hot" memory module, for a total of $Nrh + r(1 - h)$ requests. If each memory module can accept one request per network cycle (i.e., the maximum memory access rate), the maximum network throughput per processor is

$$H = \frac{1}{1 + h(N - 1)} \tag{1}$$

and the total effective memory bandwidth for the shared memory system is

$$B = \frac{N}{1 + h(N - 1).} \tag{2}$$

Fig. 2 shows the total effective memory bandwidth B as a function of the number of processors N for various hot-spot rates h. It can be seen that in a system with 1000 processors, hot-spot traffic of only 1% can limit the total memory bandwidth B to less than 10% of its peak value. Moreover, this fact is independent of network topology, the number of redundant paths, or the operation mode of the network.

If the hot-spot memory module is seen as a root and the processor elements are seen as leaves, then a fan-in tree is formed in the multistage interconnection network. Such a case is indicated in Fig. 1, where node 0 on the right represents the hot memory module and the nodes on the left represent processor elements. Due to the limited memory service rate, when the rate of total requests directed to the memory module in which the hot spot is allocated approaches or exceeds the maximum memory service rate, the queue at the memory module becomes full. The two queues feeding it can thus no longer send requests to it. They too will become full and stop the four queues feeding them from sending requests. Eventually, the entire fan-in tree will consist of full queues. *Tree saturation* has then occurred.

When tree saturation occurs, memory requests are severely blocked. Not only are hot-spot requests delayed, but normal requests are also affected. Moreover, if the system grows larger, its network performance degrades even more sharply.

2.3. Solutions to Hot-Spot Contention

There have been many approaches proposed to solving hot-spot contentions. Examples include the various switch buffer mechanisms [24], network congestion control schemes [4, 7, 19], software combining [21, 25], combining network [5, 6, 17], etc. Better switch buffer mechanisms

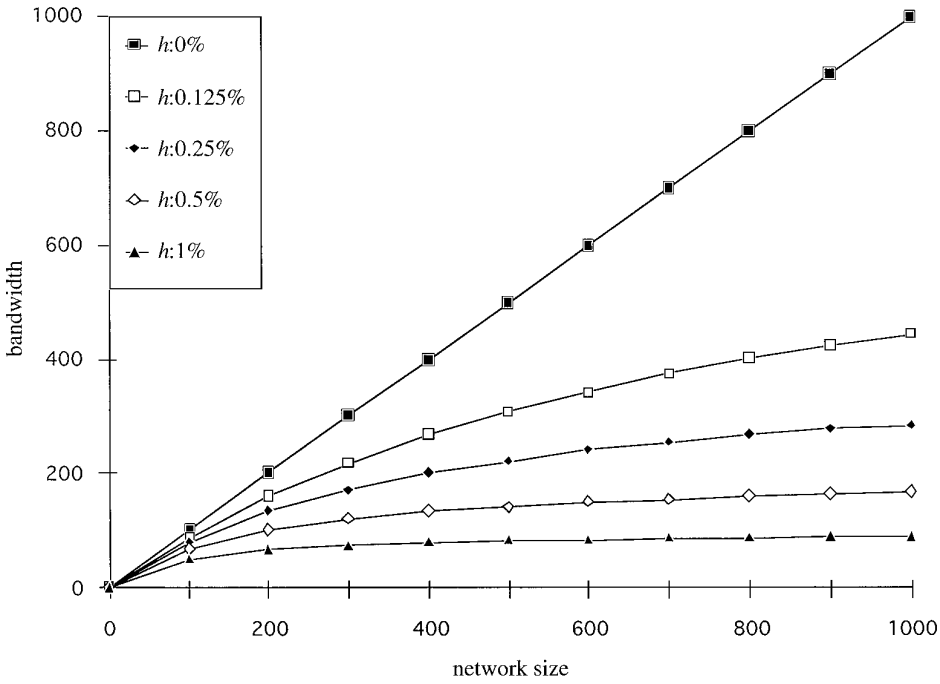


FIG. 2. Bandwidth degradation under hot-spot contention.

and network congestion control schemes can alleviate unnecessary memory access delays caused by the hot-spot contention. However, because of the limited memory service rate, combining several hot-spot requests into one whenever appropriate to reduce traffic is the most effective solution to handling hot-spot traffic. Combining can be done by software or hardware. To alleviate memory contention due to the hot-spot accesses, a software tree may be used to do the combining. The software scheme is used to distribute the hot-spot accesses to several memory modules so that hot-spot contention is reduced. Detailed schemes for software combining can be seen in [21, 25].

In this paper, we consider the design of the combining network in solving hot-spot contention. Combining networks, which have been studied in the *NYU Ultracomputer* [6] and *IBM RP3* [16], are multistage interconnection networks composed of combining switches. The combining network is bidirectional; the forward paths transmit information from processors to memory modules, and the return paths from memory modules to processors [17]. A combining switch has two queues for each direction (see Fig. 3). Queues on the forward paths are called forward queues (or combining queues). Combining works as follows: When several combinable requests that access the same memory location meet at a forward queue in a switch, they are combined into a single request and forwarded towards the shared memory. A record of this process is kept at the wait buffer of the switch where combining takes place. When the response returns, the switch satisfies all the combined requests, one at a time, and the record is eventually removed from the wait buffer. On the return path, there are return queues (or normal queues) in each switch to pass responses from memory modules to processor elements.

The implementation of a combining queue in the NYU Ultracomputer is as follows [5, 6]. Three sets of shift registers, called the IN set, the OUT set, and the CHUTE set, are associated with each combining queue of a switch. These sets of shift registers are connected as shown in Fig. 4. Packets arrive in the IN set and shift up one position in each cycle. Similarly, packets shift down one position in the OUT set in each cycle. If a packet in the IN set is adjacent to a slot in the OUT set that is empty, then it shifts to that slot. In addition, this scheme detects whether any packet in the IN queue is going to the same address as another packet already in the OUT queue. If such a packet exists, the packet in the IN queue is then placed in

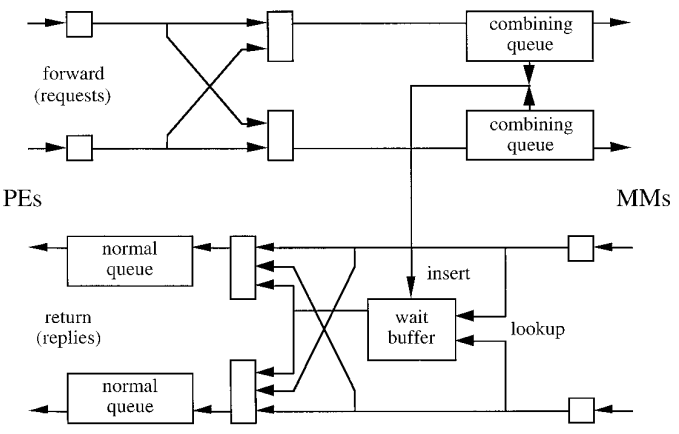


FIG. 3. A 2 × 2 combining switching element [17].

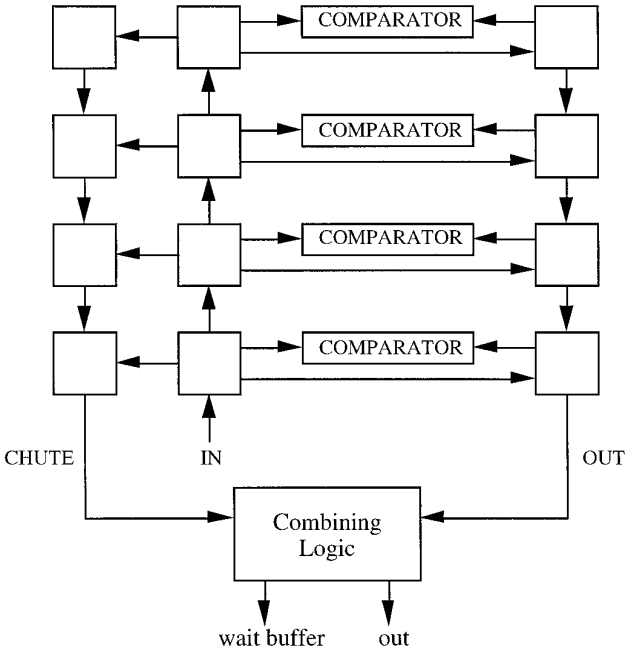


FIG. 4. An implementation of a combining queue [6].

the CHUTE set. The two matched packets then move synchronously and arrive at the combining logic simultaneously. The combining logic detects the possibility of combining the two packets and combines them so that only one packet is sent to the next stage of the network.

Combining networks are effective in handling hot-spot traffic, and they also can achieve a high synchronization rate, if some synchronization primitives, such as *Fetch&Add*, are supported [6]. Unfortunately, since the combining network is still a single-path MIN, any failure will disconnect it. Thus, a fault-tolerant combining network still needs to be developed.

2.4. Fault-Tolerant Combining Networks

In the literature, fault-tolerant combining networks have seldom been considered. Banerjee and Dugar have proposed a fault-tolerant combining network that supports the *Fetch&Add* primitive [3]. In their design, an omega network composed of 4×4 switches and augmented by an extra stage is used (as illustrated in Fig. 5). There are four disjoint paths between every source–destination pair. For each memory request, four copies of a packet are transmitted through the multipath combining network. Each of the four packets is combined individually with other packets if possible. By a special scheduling discipline in the switch, four copies of each packet are sent through the network simultaneously no matter whether they are combined or not, and the messages are voted upon at the memory-network interface or processor-network interface, thereby providing single-error correction and double-error detection. If a single fault occurs, at least three copies of each packet still arrive simultaneously at the network interface, and the voting is then carried out. Details of the network design and scheduling discipline are described in [3].

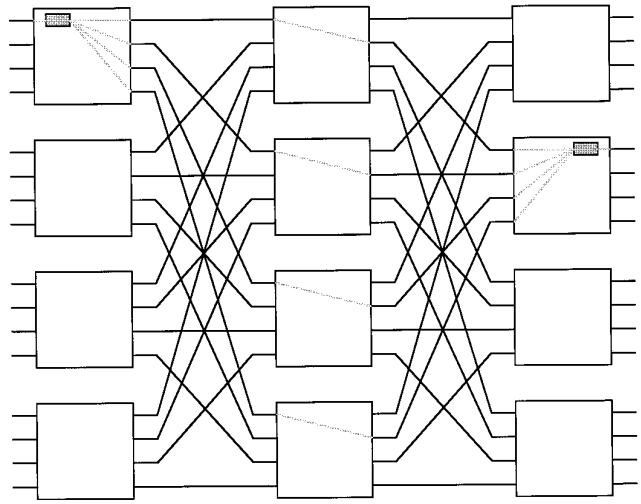


FIG. 5. The multipath omega network augmented by an extra stage [3].

The fault tolerance scheme of this network comprises information redundancy and hardware redundancy. However, this network is only single-fault tolerant, and there is an extremely high redundancy in the network design. The multipath omega network is costly, and transmitting four copies of packets through the network limits the effective network bandwidth to 25%. A cost-effective fault-tolerant combining network has to be developed. In following sections, we present a design for this purpose.

3. CHAINING—A FAULT-TOLERANT NETWORK SCHEME

As shown in Fig. 1, a *fan-in tree* can be formed in the MIN from each output (as the root of the tree) to all the inputs (as the leaves of the tree). By revealing the tree structure, Tzeng *et al.* have proposed a chaining scheme to improve the fault tolerance capability of the MIN [23]. A chained network is a network that chains together all the switching elements in the same level of the fan-in tree by using extra links between the switching elements, and hence allows each input node to have more than one path to the root. Figure 6 shows one way of connecting the switching elements in each stage to create redundant paths. To allow switching elements to be chained together, each switching element is augmented by a chain-in link and a chain-out link. In a chained network, 3 * 3 switching elements are needed. Such a network can tolerate a single link fault. To tolerate switch faults, specific chaining schemes can be used. Kumar and Reedy have proposed augmented shuffle-exchange networks (ASENs) for this purpose [9]. In these networks, switching elements within a stage are chained based on certain conjugate properties. Dual I/O links and reconfiguration of the last stage make ASENs single switch-fault tolerant. Figure 7 illustrates an

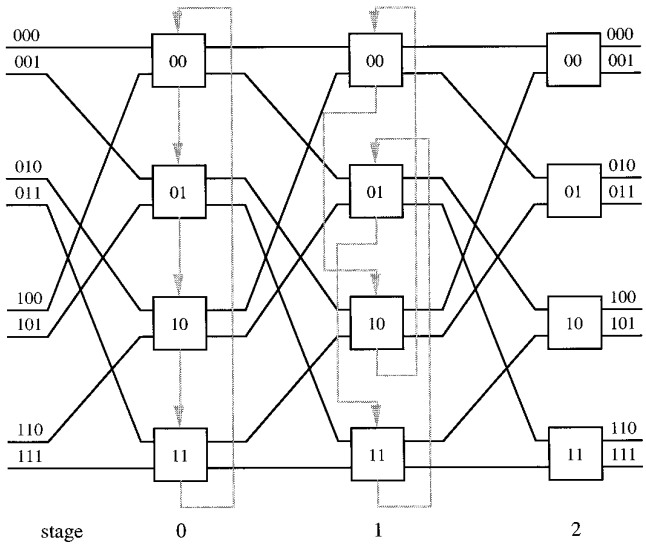


FIG. 6. A chained omega network for $N = 8$.

ASEN with a maximum loop size. (A chain in [23] is called a loop in [9].)

Chained networks exploit all inherent paths embedded in the tree structure of MINs so that they provide high terminal reliability. Moreover, chained networks have distributed control, dynamic rerouting, and on-line diagnosis capabilities [9, 22, 23]. The chained network is chosen as the base network for our design of a fault-tolerant combining network because of these properties. Chained networks have multiple source-to-destination paths to provide fault tolerance capability. Multipath networks can also provide better performance under nonuniform traffic [10]. In the next section, we examine the behavior of chained networks under hot-spot traffic conditions.

4. CHAINED NETWORKS UNDER HOT-SPOT TRAFFIC

4.1. Chaining Scheme

In this section, we examine the behavior of original MINs and chained networks under hot-spot traffic when different network congestion control schemes are used. The naming scheme for these networks is as follows. The stages are labeled in a sequence from 0 to $(\log_2 N - 1)$, with 0 being the leftmost source-side stage. In each stage, a switching element is identified with $t = (\log_2 N - 1)$ binary bits $p_0 p_1 \ldots p_{t-1}$, which constitute the binary representation of its location in the stage. Each input/output link is labeled with $t + 1$ bits $p_0 p_1 \ldots p_t$, of which the left-most t bits are the binary representation of the switching element. The last bit p_t is 0 if the link is the upper link; and p_t is 1 if the link is the lower link. Figure 1 shows the naming scheme used for switching elements and links of an 8×8 omega net-

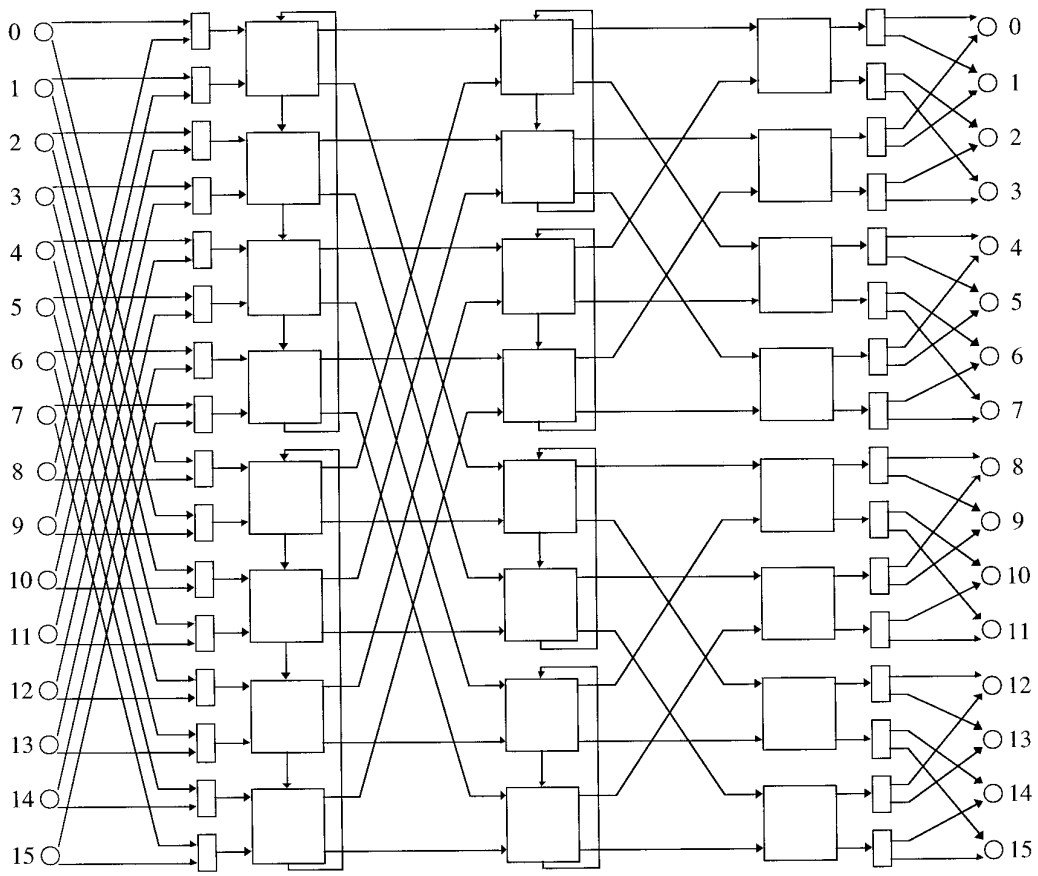


FIG. 7. A 16-input augmented shuffle-exchange network with maximum-size loops [9].

work. To devise a chaining scheme, we state the following definitions in an omega network:

DEFINITION 1. All the switching elements in stage i, $0 < i < \log N$, belong to the same *partition* if $p_{t-i} \ldots p_{t-1}$ in the binary representation of their name, $p_0 p_1 \ldots p_{t-1}$, are the same. All switching elements in stage 0 constitute a *partition.*

DEFINITION 2. Any number of switching elements in the same partition can form a *chain* by connecting the chain-in link of a switching element to the chain-out link of itself or another switching element within the same partition. A *complete chain* is a chain formed by connecting together all the switching elements within the same partition.

In our simulations, chained omega networks are used, in which interstage switches are connected in the conventional way. According to Definitions 1 and 2, switches within a stage are chained based on the formula

$$chain\_to(SE\_ID, stage) = (SE\_ID + 2^{stage}) \text{ modulo } 2^{n-1},$$

where SE_ID is the name of a switching element within a stage and $n = \log N$. For example, in stage 0 (1), the chain-out link of switch 0 is connected to the chain-in link of switch 1 (2). An example of a chained omega network is illustrated in Fig. 6.

4.2. Simulation Models

In our simulations, original omega networks and chained omega networks are used. In a chained network, 3×3 switching elements are needed. A 3×3 switching element is augmented from a 2×2 switching element with chain in/out links. The buffer mechanisms for these two types of switching elements are shown in Fig. 8. Output queues and input latches are used in the switching elements. The size of output queues is 4. If the destination output queues are not full, all packets in the input latches will be routed to the destination output queues, in a network cycle. If the destination output queues are full, the packets have to wait in the input latches. Various congestion control schemes are used in the 2×2 and 3×3 switching elements:

(1) *Blocking switch.* If the associated output queue is full, then in a 2×2 switching element, a newly arrived input packet remains at the input latch, and further transmission of packets through the switch is blocked. In a 3×3 switching element, the blocked input packet may be thrown into other switching elements within the same stage by the chain in/out links.

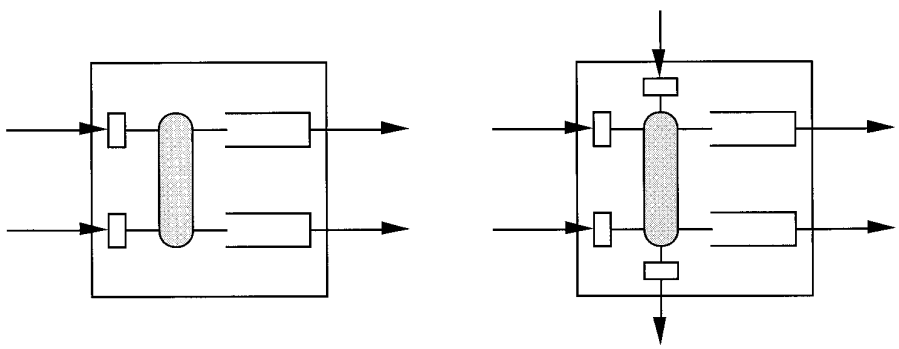
(2) *Discarding switch.* In a 2×2 switching element, if the packets in the latches cannot be queued, they are discarded immediately. When a packet is discarded, the issuing processor will be notified to retransmit the packet again. In a 3×3 switching element, an attempt is first made to send overflow packets to other switching elements within a stage by the chain in/out links. If this attempt fails, then the packets are discarded.

(3) *Diverting switch.* In a 2×2 switching element, when the associated queue is full, packets may be diverted to another queue. Because they are diverted, the packets will go to incorrect destinations, and retransmissions from the wrong intermediate destinations are needed. Similarly, 3×3 switching elements also provide chain in/out routing before diverting.

In all of these congestion control schemes, the packets in chain-in latches have the highest priority to enter their destination queues. To prevent packets from circulating forever within a chain, packets in chain-in latches are forbidden to be routed to chain-out latches.

In the simulations, we made the following assumptions:

- Each request is a single packet.
- The networks are synchronous. Only at the network cycle times, t_c, $2t_c$, ..., are the packets transmitted. For simplicity, assume t_c equals 1.
- The networks are pipelined. That is, processors can issue other requests even before previous requests are returned.
- The service time of a request in a switching element is the same as the network cycle time, so the waiting time



(a) 2 * 2 switch (b) 3 * 3 switch

FIG. 8. Buffer mechanisms of switches.

of a request at a switching element equals the number of requests ahead of it in the queue.

• Each processor has an infinite queue for requests. If a request is blocked from entering the network, it is placed on the processor queue, and the processor may continue to issue requests.

In the simulations, we also adopt the hot-spot traffic model of Pfister and Norton described in Section 2. In this model, there are two types of requests: the *noncombinables*, which access each memory module with equal probability as in the uniform model, and the *combinables*, which access the same shared variable (and hence the same memory module). In our simulations, only *combinables* are candidates for combining.

4.3. Simulation Results

Figure 9 illustrates the behaviors of various networks with different congestion control schemes under uniform traffic conditions. Because of sufficient queue size in each output link, all of these networks have similar performance behavior under light traffic conditions. As the request rate increases, the limited buffer sizes force the networks to make congestion control decisions—that is discarding or diverting packets. Because it incurs no unnecessary routing overhead, the blocking switch performs more favorably than discarding and diverting switches under uniform traffic conditions. Yet under heavy traffic conditions, better congestion control schemes, such as discarding and diverting, have lower memory access delays.

If hot-spot traffic occurs, due to the limited memory service rate, tree saturation occurs in all networks (as illustrated in Fig. 10). A request rate of $r = 60\%$ and a hot spot rate of $h = 2\%$ exceeds the amount of traffic the network can handle. By Eq. (1), when hot-spot traffic occurs, the network throughput per processor is limited to $1/(1 + h(N - 1))$. So a network size of $N = 64$ and a hot spot rate $h = 2\%$ limits the network throughput per processor to 0.44. However, if a combining network is used, the congestion due to hot-spot traffic can be relieved. Figure 11 shows the simulation results.

On hot-spot contention, blocking switches that block the packets from using the buffer resource of the network have the worst performance. Tree saturation occurs quickly. Blocking affects not only the *combinable* requests but also the *noncombinable* requests. Discarding switches that discard the overflow packets to resolve conflicts have better performance than blocking switches. They reserve free buffers for further routing at any time by discarding blocked packets. Diverting is another strategy for keeping the network operational under any traffic condition. To summarize, discarding, diverting, or other congestion control schemes shortens memory access delays due to tree saturation. However, due to the limited memory service rate, the network is still unstable and tree saturation still may occur. The simulation results also show that the multipath chained network performs better than the unique-path MIN, regardless of the congestion control scheme used. Needless to say, if hot-spot traffic continues unabated, the chain-in and chain-out links can do little to improve performance.

4.4. Discussion

Intuitively, the hot-spot traffic model we adopted does not seem realistic. In reality, hot-spot memory accesses are not likely to persist continuously; rather, they occur in short bursts. An example is when synchronization is needed among processors. Kumar and Pfister have observed that a relatively short period of hot-spot contention

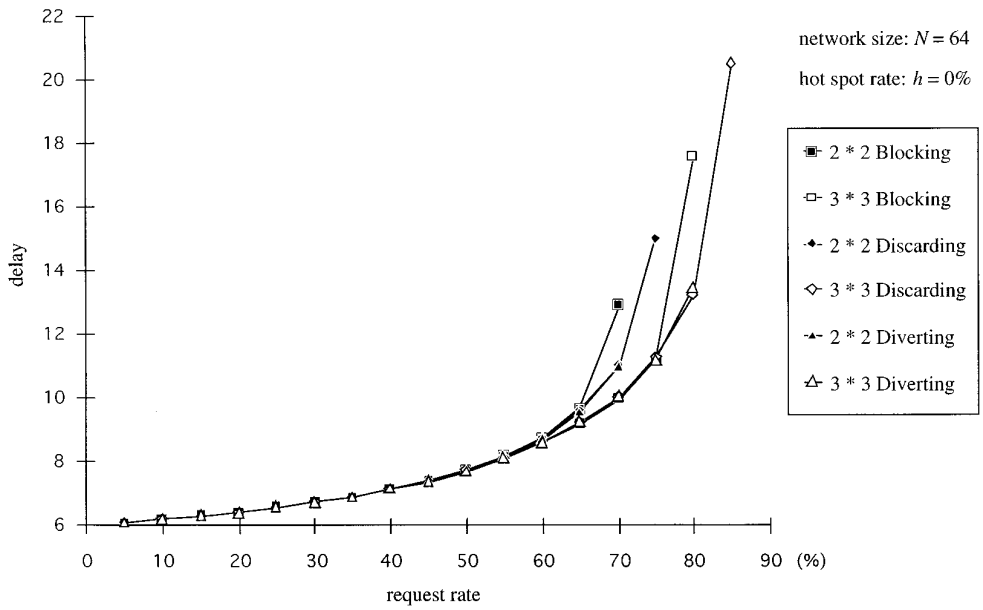


FIG. 9. Performance of chained networks under uniform traffic.

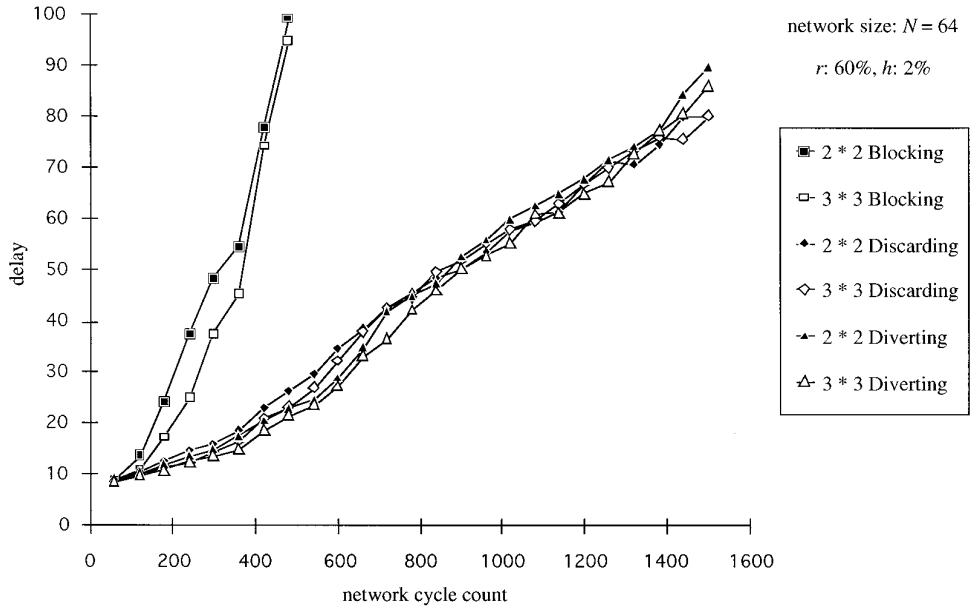


FIG. 10. Tree saturation in the chained networks.

will still cause tree saturation [8]. Furthermore, it takes a long time for the network to return to its normal state even after processors stop issuing hot-spot requests. Better congestion control schemes have better performance: the delay increases more slowly, the onset time is longer, and the recovery time is shorter. Multipath MINs also have better performance under such conditions.

Lang and Kurisaki noticed that the chained omega network can handle nonuniform traffic spots [10]. However, in our simulations, this type of network does not handle hot-spot traffic well. This discrepancy occurs because with nonuniform traffic spots, contention occurs in the switching elements, whereas under hot-spot traffic, contention occurs in the memory modules. When contention occurs in the switching elements, the traffic can be redirected to other switches through the chain in/out links. On the other hand, when contention occurs in the memory modules, alternate paths are of no use in traffic control, because memory modules are the sole contended resources.

Hot-spot contentions can be roughly divided into two categories: network contention and memory contention. Yet they are related, not independent. In the literature, various congestion control schemes have been proposed to handle hot-spot traffic [4, 7, 19, 24]. Typically, these

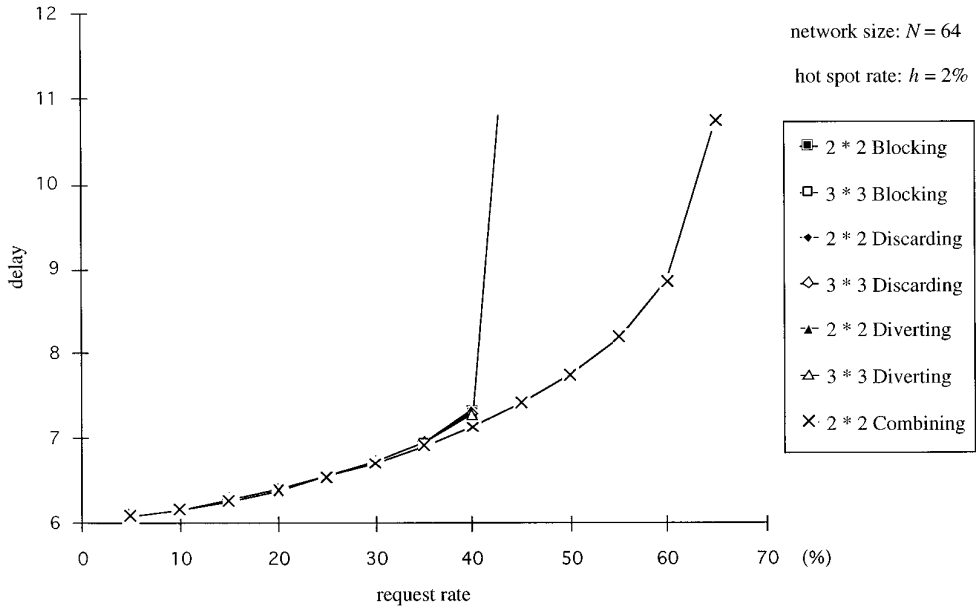


FIG. 11. Performance of chained networks under hot-spot traffic.

schemes are based on contention prevention. For instance, a discarding switch makes a network more operative by discarding packets. Prevention-based schemes reduce the unnecessary memory access delays caused by hot-spot accesses and improve network performance to a certain extent. However, because of limited memory service rates, prevention-based schemes still fail to handle hot-spot traffic well. On the other hand, combining networks that can reduce the number of hot-spot requests are the solution.

5. CHAINED COMBINING NETWORK

5.1. Architecture of the Chained Combining Network

A chained network is fault tolerable. A combining network reduces hot-spot traffic. Hence, we use the chaining scheme to build a fault-tolerant combining network. Our base network is the omega network. The naming scheme of the network is the same as that described in Section 4. According to Definition 2, there may be various chaining schemes in the network. The chaining scheme we used is based on the following formulas:

Forward Chaining. $f\_chain\_to(SE\_ID, stage) = (SE\_ID + 2^{stage})$ modulo 2^{n-1}.

Return Chaining. $r\_chain\_to(SE\_ID, stage) = (SE\_ID - 2^{stage})$ modulo 2^{n-1}.

where SE_ID is the name of the switching element within a stage and $n = \log N$. The formula *f_chain_to* specifies the chaining function in the forward path (PEs to MMs), and the formula *r_chain_to* specifies the chaining function in the return path (MMs back to PEs).

According to the chaining formulas, an 8×8 completely chained omega network is as in Fig. 6. (Arrows indicate the forward chaining, and the return chaining is just the reverse of the forward chaining.) In the chained combining network, 3×3 combining switching elements are needed. A 3×3 combining switching element is illustrated in Fig. 12. For a clear illustration, the switching element is depicted in two separate parts: a forward part (PEs to MMs) and a return part (MMs back to PEs). In this fault-tolerant combining network, we consider only the link-faults. Chain in/out links are used when the associated output link to which the packets are routed fails. The link-faults are assumed symmetric: when the forward link fails, the corresponding return link also fails. And if a link is good, then the forward part and the return part are both in working status.

5.2. Routing in the Chained Combining Network

Routing in a unique-path combining network is simple. The routing of combined packets are the same as those that are not combined, and the original routing algorithms are valid. For example, destination-tag routing is used in the omega network [11]. Assume that a packet has a source tag $s_0 s_1 \ldots s_{n-1}$ and a destination tag $d_0 d_1 \ldots d_{n-1}$. Then, in

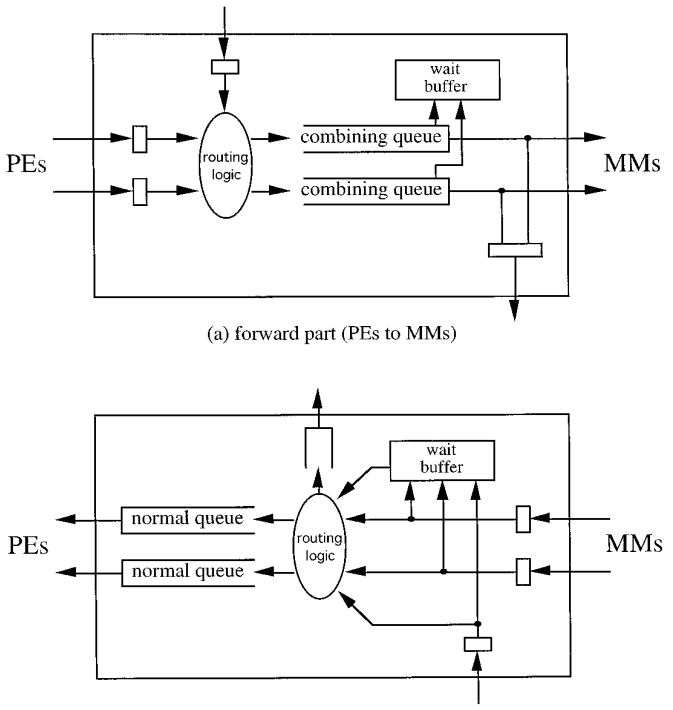


FIG. 12. The 3×3 combining switching element.

stage i, if $d_i = 0$ (1), this packet is routed to an upper link (lower link). When requests are issued from processors to memory modules, the IDs of the memory modules are examined. When requests are returned from memory modules to processors, the IDs of the processor elements are examined. On the other hand, in the chained combining network, alternate paths are provided. Once a packet has been routed through the network from PE to MM, because of the possibility of combining, the packet must traverse the original PE-to-MM path on its return routing. Otherwise, some combined packets may be pending in the wait buffers of switches forever, and wait buffers may thus be blocked, causing the combining network to be out of operation. Hence, a new routing scheme which can keep track of the forward routing path among alternate paths in the chained combining network must be developed. To devise such a scheme, we add a *chain-out record* to each packet and modify the PE ID of each packet to route packets back and forth correctly.

In this paper, we consider only the routing of a combining network under link-faults, because of our chaining scheme. Nevertheless, our discussion is easily extensible to a routing scheme for networks that can tolerate switch-faults.

5.2.1. Routing in the Absence of Faults

In the absence of faults, the routing of the chained combining network is the same as in the original omega network [11]. Destination-tag routing is used. Assume that a packet has source tag: $s_0 s_1 \ldots s_{n-1}$; destination tag: $d_0 d_1 \ldots$

d_{n-1}. In stage i, if $d_i = 0$, then the packet is routed to the upper link (link 0); if $d_i = 1$, then it is routed to the lower link (link 1). When requests are issued from processor elements to memory modules (destinations are memory modules), the IDs of the memory modules are examined. When requests are returned from memory modules to processor elements (destinations are processor elements), the IDs of the processor elements are examined.

5.2.2. Routing under Link Faults

In this case, when requests are issued from processor elements to memory modules, the modified destination-tag routing algorithm in [23] can send the requests to the destination memory modules even after the packets have been chained out. In their routing algorithm, if a packet has been chained out to another switch within stage i, the ith destination tag bit d_i is also examined in routing (if $d_i = 0/1$, then the packet is routed to the upper/lower link). If a fault-free link is found in a switch, then the packet is routed to the next stage. In this way, their routing algorithm routes the packet to the destination. When packets are not combined on the forward routing, different paths or networks can be used on the return routing. However, the combining network has to be bidirectional. When the memory requests are returned, the packets must traverse its previous path to guarantee that all the original combined packets will be decombined properly.

To solve this problem, we add a *chain-out record* to each packet, and modify the processor element ID of each packet to maintain the correctness of routing when packets are chained out. The format of the *chain-out record* is as follows:

(1) Chain-out indicator: $C = c_0 c_1 \ldots c_{n-2}$

$c_i = 1$, if the packet has been chained out in stage i,

$c_i = 0$, otherwise.

(2) Partial switching element ID of the first chain-out location in stage i;

$$L_i = l_0 \ldots l_{n-2-i} \quad (i = 0, \ldots, n-2).$$

According to Definition 1, a chain can be formed only if the switching elements are in the same partition. In stage i, the switching elements are in the same partition if the binary representations of their names, $p_{t-i} \ldots p_{t-1}$ ($t = \log_2 N - 1$), have the same value. So, in stage i, to specify a switching element within a chain, only $t - i$ bits $p_0 \ldots p_{t-i-1}$ have to be recorded.

In an $N \times N$ completely chained network, the size of chain-out record is calculated as follows. First, the chain-out indicator requires ($\log N - 1$) bits. Second, in stage i there are $N/2^{i+1}$ associated possible locations of the first chain-out switching element within a chain, thus L_i of ($\log N - i - 1$) bits must be appended. Therefore, other than the data bits, source tag, destination tag, and some special control or check bits, there are

$$\log N - 1 + \frac{\log N(\log N - 1)}{2}$$

additional detour bits needed in each packet. For $N = 64$, 20 detour bits are needed. The number of detour bits is of order $O((\log N)^2)$. Furthermore, if all the chains of the network are of size 2, then the switching element IDs of the first chain-out location within a chain (L_i, $i = 0, \ldots, n-2$) can be eliminated. Only the chain-out indicator, which is ($\log N - 1$) bits long, is required, and the number of detour bits is of order $O(\log N)$.

5.2.3. The Routing Procedures

Figure 13 illustrates the routing procedures for the chained combining network. In the forward routing (PE to MM), when a packet is routed to a switching element SE of stage i, if the links of SE are fault-free, the destination-tag routing is used, i.e., d_i of the memory module ID is examined. When a packet finds the first link fault, before the packet is chained out, the chain-out record must be updated: bit c_i of chain-out indicator is set and the partial SE ID $s_{i+1} \ldots s_{n-1}$ is recorded in L_i to specify SE within a chain. Then the packet is chained into switching element SE'. In addition, based on the ID of SE', the PE ID must be modified properly for correct routing:

In stage i, according to the routing of omega network, a packet will be routed to the switching element

$$SE: s_{i+1} \ldots s_{n-1} d_0 \ldots d_{i-1}.$$

If a packet finds the first link fault at switching element SE, then through the chain in/out links, the packet will be routed to

$$SE': s'_{i+1} \ldots s'_{n-1} d_0 \ldots d_{i-1}$$

to find a fault-free link to proceed to the next stage. To let the packet return to the switching element SE', the processor element ID of the packet must be modified to

$$s_0 \ldots s_i s'_{i+1} \ldots s'_{n-1}.$$

If the packet finds further link faults within stage i, then the packet is thrown to the next switching element SE'' via chain in/out links, and the processor element ID of the packet must be modified as described above. When a packet is chained into switching element SE, if $c_i = 1$ and $L_i =$ (ID of SE), this indicated that the network is disconnected. Routing is hence impossible.

In the return routing (MM back to PE), when a packet is routed to a switching element SE of stage i, if bit i of the chain-out indicator is zero ($c_i = 0$), the destination-tag

```
type  packet = record                    /* the data structure of packets */
            S: s0s1...sn-1               /* source tag--PE ID tag */
            D: d0d1...dn-1               /* destination tag--MM ID tag */
            C: c0c1...cn-2               /* chain-out indicator */
            L[0..n - 2]                  /* first chain-out locations within stage i */
      end

procedure forward_routing(P : packet);   /* from PE to MM */
begin
      for i = 0 to n - 1      /* n = log N */
            /* routing decision: examine the MM ID tag */
            switch (di)
                  case di = 0: go to upper link;
                  case di = 1: go to lower link;
            endswitch;
            /* current switching element ID: si+1...sn-1d0...di-1 */
            while (the associated output link is at fault)
                  if ci = 0 then          /* the fault is the first one found in stage i */
                        /* in order to specify the switching element within a chain,
                           record the partial switching element ID: si+1...sn-1 in Li */
                        Li := si+1...sn-1;
                        ci := 1;
                  endif;
                  route the packet P to SE' s'i+1...s'n-1d0...di-1,
                        to find a fault-free link di,
                        by the forward chain in/out links;
                  if Li = s'i+1...s'n-1 and ci = 1 then
                        /* the packet has been routed through all switch elements
                           within a chain and no fault-free output link found */
                        the network is disconnected;
                        exit;
                  endif;
                  S := s0...si s'i+1...s'n-1          /* modify the PE ID tag */
                  switch (di)                         /* reexamine the MM ID tag */
                        case di = 0: go to upper link;
                        case di = 1: go to lower link;
                  endswitch;
            endwhile;
      endfor;
endproc;


procedure return_routing(P : packet);  /* MM return to PE */
begin
      for i = n - 1 downto 0;
            /* In stage i, examine chain-out indicator ci */
            if ci = 0 then          /* fault-free in this stage */
                  switch (si)     /* routing decision: examine the PE ID tag */
                        case si = 0: go to upper link;
                        case si = 1: go to lower link;
                  endswitch;
            else                    /* fault occurred in this stage */
                  repeat
                        route the packet P by the return chain in/out links within the chain;
                  until the packet P is sent to switching element Li within the chain;
                  /* modify the PE ID tag S back based on the Li */
                  S := s0...si l0...ln-i-2;
                  /* original path is found and route packet P to next stage */
                  switch (si)     /* routing decision: examine the PE ID tag */
                        case si = 0: go to upper link;
                        case si = 1: go to lower link;
                  endswitch;
            endif;
      endfor;
endproc;
```

FIG. 13. The routing procedures for the chained combining network.

routing is used (bit i of the processor element ID s_i is examined). If $c_i = 1$, then through the return chain in/out links, the packet is routed to switching element L_i, and the processor element ID is modified back accordingly. Then the packet is routed to the next stage according to bit i of processor element ID s_i, and the routing process proceeds until the packet is sent back to the issuing processor element.

THEOREM 1. *The routing procedure stated in Fig.* 13 *can route a packet from a processor to its destination memory module if a fault-free path is found, and the packet can be routed back to the issuing processor element through the original forward routing path.*

Proof. Assume that the processor element whose ID is $s_0 s_1 \ldots s_{n-1}$ issues a packet to the memory module whose tag is $d_0 d_1 \ldots d_{n-1}$. If the chained network is fault-free, according to the routing of an omega network, in stage i, the packet will be routed to switch $s_{i+1} \ldots s_{n-1} d_0 \ldots d_{i-1}$, and the packet will be put in the output buffer of link $s_{i+1} \ldots s_{n-1} d_0 \ldots d_{i-1} d_i$. If a link fault is found, the chain-out record will be updated, the packet will be sent by the chain in/out link to a fault-free switching element $SE' s'_{i+1} \ldots s'_{n-1} d_0 \ldots d_{i-1}$, and the processor element ID of the packet will be modified according to the ID of SE'. Then routing will proceed in subsequent stages until the packet is routed to the destination memory module.

In general, in an omega network, modification of a processor element ID will cause a packet to return to an erroneous source—the source designated by the modified processor element ID. By the return routing procedure, however, because of the setting of c_i (bit i of the chain-out indicator), the packet will be chained out by the return chaining links. While the return chain-out routing within a stage is being performed, if the first chain-out location of the packet L_i matches the ID of the switching element, the packet has found the original forward path through which it was previously routed, and the packet is then routed back to the issuing processor element accordingly (see Fig. 14). Q.E.D.

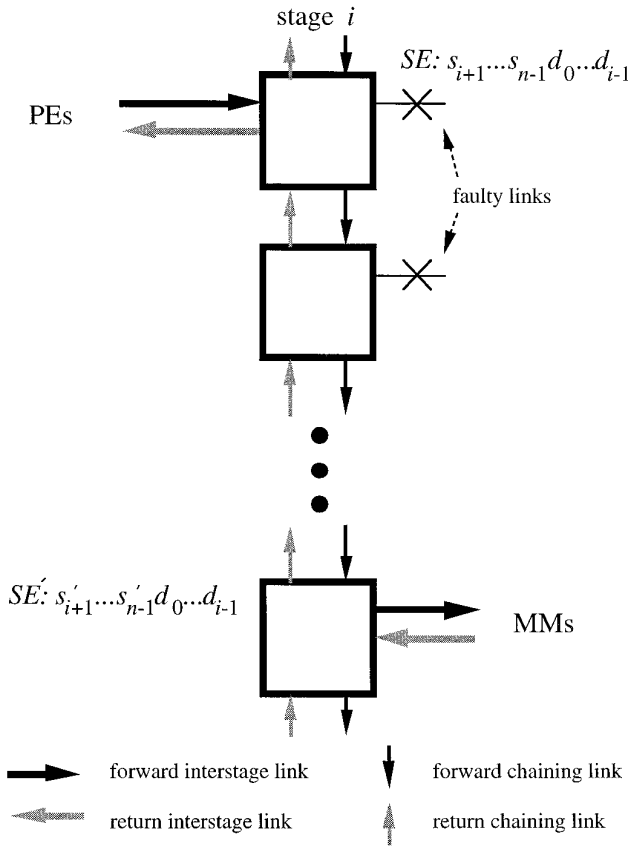


FIG. 14. Routing of the chained combining network under link faults.

Figure 15 depicts a routing example of the chained combining network. Assume that processor element 0 wants to access memory module 0, and the output link 0 of switching element 0 of stage 0 is faulty. By the forward routing procedure, a packet will be sent to switching element 0 of stage 1 through output link 0. However, the output link is at fault, so the packet has to be rerouted to switching element 1 of stage 0 and then forwarded to memory module 0. To record the rerouting path, bit 0 of the chain-out indicator c_i has to be set, the partial switching element ID of the first chain-out location 00 has to be recorded in L_0, and the processor element ID S has to be modified to $S' = 001$. In the return routing, according to S', the packet will be sent back to switching element 1 of stage 0. Because bit 0 of c_i is set and L_0 is recorded, in stage 0, the packet is rerouted back to switching element 0 via return chain in/out link. Then the packer returns to processor element 0.

5.3. Performance Evaluation

In a chained combining network, the combining capability can relieve hot-spot traffic, and the chaining scheme offers alternate paths to provide fault tolerance capability. The multipath feature also improves network performance under heavy traffic conditions. To evaluate the performance of chained combining network, we conduct a set of simulations.

In this set of simulations, we also adopt the hot-spot traffic model of Pfister and Norton [17] as described in

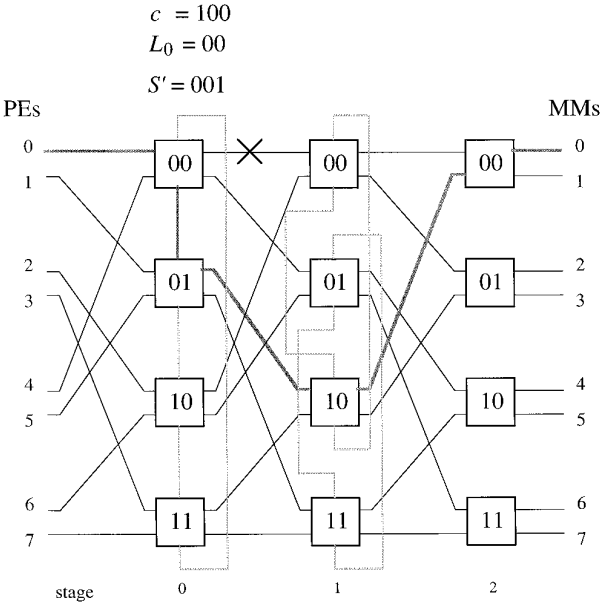


FIG. 15. A routing example for the chained combining network.

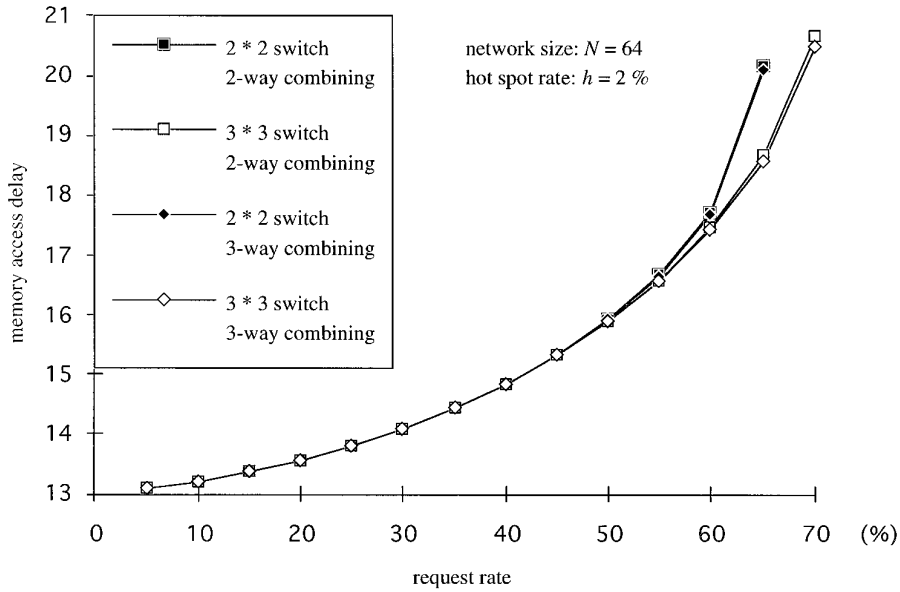


FIG. 16. Performance of fault-free chained combining network.

Section 2. The original omega network and the completely chained omega network are used, and their network sizes are $N = 64$. We assume the use of 2×2 switching elements and 3×3 switching elements in the omega network and the completely chained omega network, respectively. The configuration of a switching element is as follows. The degrees of combining, meaning the largest numbers of requests that can be combined simultaneously, are 2 or 3; the forward queue sizes are 4; and the wait buffer and return queue sizes are assumed to be infinite. In a chained combining network, if the forward queue is full, a packet may be rerouted to the next SE within a chain. To prevent packets from running forever within a chain, we constrain the chain-out routing in our simulations: each packet can be chained out at most once in each stage, because unlimited chain-out routing may incur live lock in the system.

Figures 16 and 17 illustrate the performance of chained combining network under fault-free and faulty conditions, respectively. When the network is fault-free, alternate paths provide routing choices to improve performance (see Fig. 16). Moreover, if the degree of combining is higher, there is still some performance improvement when traffic is heavy [12, 13]. When a link fault occurs in the chained combining network, the traffic associated with the faulty output link must be directed to other switches within the stage by chain in/out links. To observe behavior of the

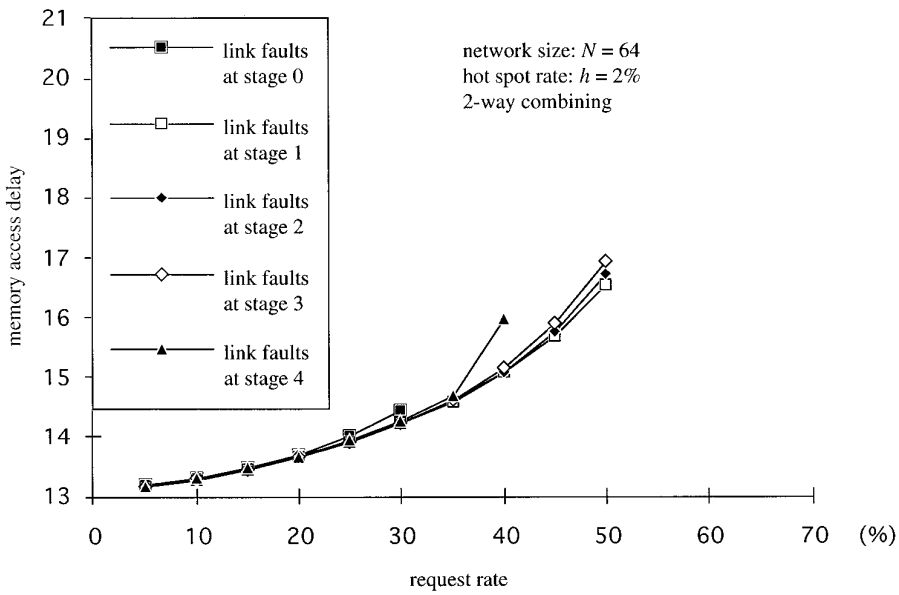


FIG. 17. Performance of chained combining network under link faults.

network under link faults, we define a set of fault patterns: Both output links of the switching element 0 at stage i ($0 \le i \le n-2$) are faulty. (These fault patterns do not disconnect the network.) Assume that the maximum request rate a fault-free network can handle is 100%. As a result, when a fault occurs, the traffic at the switch to which the extra traffic is directed is then doubled, and the maximum request rate that the network can sustain is at most 50%. Figure 17 shows the simulation result of chained combining network under link faults. In general, when faults occur near the processors (except at stage 0), the memory access delay is increased because of the severe contention caused by link faults and hot-spot accesses. When faults occur at stage 0, the request rate that can be sustained is only about 30% because all the requests of PE 0 and PE 32 are severely congested at the chain-out link between SE 0 and SE 1 of stage 0.

6. DISCUSSION

In a combining network, to correctly decombine the packets waiting in the wait buffers of switches, the combined packets, on their return routing, must traverse their forward route. In unique-path MINs, this is a trivial problem, because only a unique path exists between any source–destination pair in the network. However, in multipath MINs, which provide fault tolerance, forward routing path needs to be recorded to help in the return routing. Thus, in the chained combining network, chain-out records are used to identify the forward paths taken. For example, a completely chained network provides $2^{(n^2-n)/2}$ paths (not all disjoint) between any source–destination pairs, where $n = \log N$. So an $O((\log N)^2)$-bit chain-out record is required. (If the chain size is limited to 2, the chain-out record size can be reduced to $O(\log N)$ because only $2^{(n-1)}$ alternate paths exist.)

In the chained combining network, the packet size overhead—the chain-out record—is high: it is $O((\log N)^2)$ for a completely chained network. This increases both the packet size and the transmission time. To reduce the chain-out record size, a rerouting buffer may be introduced into the switching elements. A rerouting buffer can be similar to the wait buffer of a combining switch. While the wait buffer is used to combine packets, the rerouting buffer is used to reroute packets. Obviously, when a rerouting buffer is used, it should have enough space to accommodate rerouted packets, or else the performance of the network may suffer. In addition, because of high network delay, switching elements closer to processors need more rerouting buffer space.

Because of the chaining scheme, our fault-tolerant combining network can tolerate only link faults. However, if the network is carefully augmented by a special chaining formula, such as that in the augmented shuffle-exchange networks (ASENs) [9], switch faults can also be tolerated. With a slight modification, our routing procedures can also be used to handle switch faults: the chain-out record in each packet is modified according to the associated chaining schemes.

Chaining is not the only scheme that can be used to enhance the fault tolerance capability of a combining network; other fault tolerance schemes, such as the extra-stage cube network [1], the gamma network [15], and the INDRA network [18], may also be applied to combining networks. Furthermore, because these schemes offer only limited number of alternate paths, the routing procedures in these schemes are easier. For example, in the extra-stage cube network [1], there are two paths between each source-destination pair. To record the specific path in the forward routing, one bit is sufficient.

7. CONCLUSIONS

It is known that tree structures are embedded in the MIN. When the tree structures are enhanced, the fault tolerance capability of the MIN can be increased, and the traffic contention may be relieved. For providing fault tolerance capability in a MIN, we use a chaining scheme to enhance the connectivity of the tree structures. The chained networks allowing alternate paths not only provide fault tolerance capability, but also improve performance when the network traffic is congested. However, because of limited memory service rate, the alternate paths cannot handle the hot-spot traffic well. The combining capability can be implemented in the MIN to relieve hot-spot traffic. Thus, we propose the chained combining network with its routing procedures, as the solution to both fault tolerance and hot-spot contention problems in MINs.

The network performance often sharply degrades when faults occur. In the chained combining network, when faults occur, alternate paths are taken to redirect traffic to other switching elements. Because of traffic redirection and the limited service rate of the switching elements, contention is more likely to arise in the directed switching elements. In this situation, alternate paths can again be used to share the traffic load. Moreover, certain complex congestion control schemes, such as, diverting [10], may also improve network performance and alleviate performance degradation further when faults occur. Due to the bidirectionality of combining networks, congestion control schemes must be carefully developed. In a word, if a combining network supports any kind of congestion control schemes, the routing procedures must keep track of the forward routing path for the sake of decombining packets properly on their return routing.

ACKNOWLEDGMENTS

The authors thank the referees for their knowledgeable comments that are helpful to improve the quality of this paper greatly.

REFERENCES

1. Adams, G. B., III, and Siegel, H. J. The extra stage cube: A fault-tolerant interconnection network for supersystems. *IEEE Trans. Comput.* **C-31,** 5 (May 1982), 443–454.

2. Adams, G. B., III, Agrawal, D. P., and Siegel, H. J. A survey and comparison of fault-tolerant multistage interconnection networks. *IEEE Comput. Mag.* **20,** 6 (June 1987), 14–27.
3. Banerjee, P., and Dugar, A. The design, analysis and simulation of a fault-tolerant interconnection network supporting the fetch-and-add primitive. *IEEE Trans. Comput.* **C-38,** 1 (Jan. 1989), 30–45.
4. Dias, D. M., and Kumar, M. Preventing congestion in multistage networks in the presence of hotspots. *Proc. 1989 International Conference on Parallel Processing.* IEEE Computer Society, Silver Spring, MD, 1989, Vol. I, pp. 9–13.
5. Dick, S. R., and Kenner, R. Hardware combining and scalability. *Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures.* 1992, pp. 296–305.
6. Gottlieb, A., *et al.* The NYU Ultracomputer-Designing a MIMD shared memory parallel machine. *IEEE Trans. Comput.* **C-32,** 2 (Feb. 1983), 175–189.
7. Ho, W. S., and Eager, D. L. A novel strategy for controlling hot-spot congestion. *Proc. 1989 International Conference on Parallel Processing.* IEEE Computer Society, Silver Spring, MD, 1989, Vol. I, pp. 14–18.
8. Kumar, M., and Pfister, G. F. The onset of hot spot contention. *Proc. 1986 International Conference on Parallel Processing.* IEEE Computer Society, Silver Spring, MD, 1986, pp. 28–34.
9. Kumar, V. P., and Reedy, S. M. Augmented shuffle–exchange multistage interconnection networks. *IEEE Comput. Mag.* **20,** 6 (June 1987), 30–40.
10. Lang, T., and Kurisaki, L. Nonuniform traffic spots (NUTS) in multistage interconnection networks. *J. Parallel Distrib. Comput.* **10,** 1 (Sep. 1990), 55–67.
11. Lawrie, D. H. Access and alignment of data in an array processor. *IEEE Trans. Comput.* **C-24,** 12 (Dec. 1975), 1145–1155.
12. Lee, G. H., Kruskal, C. P., and Kuck, D. J. The effectiveness of combining in shared memory parallel computers in the presence of 'hot spots.' *Proc. 1986 International Conference on Parallel Processing.* IEEE Computer Society, Silver Spring, MD, 1986, pp. 35–41.
13. Lee, G. A performance bound of multistage combining networks. *IEEE Trans. Comput.* **C-38,** 10 (Oct. 1989), 1387–1395.
14. Lilienkamp, J. E., Lawire, D. H., and Yew, P. C. A fault tolerant interconnection network using error correcting codes. *Proc. 1982 International Conference on Parallel Processing.* IEEE Computer Society, Silver Spring, MD, 1982, pp. 123–125.
15. Parker, D. S., and Raghavendra, C. S. The gamma network. *IEEE Trans. Comput.* **C-33,** 4 (Apr. 1984), pp. 367–373.
16. Pfister, G. F., *et al.* The IBM research parallel processor prototype (RP3): Introduction and architecture. *Proc. 1985 International Conference on Parallel Processing.* IEEE Comput. Soc., Silver Spring. 1985, pp. 764–771.
17. Pfister, G. F., and Norton, V. A. Hot-spot contention and combining in multistage interconnection networks. *Proc. 1985 International Conference on Parallel Processing.* IEEE Comput. Soc., Silver Spring. 1985, pp. 790–797.
18. Raghavendra, C. S., and Varma, A. INDRA : A class of interconnection networks with redundant paths. *1984 Real-Time System Symposium,* 1984, pp. 153–164.
19. Scoot, S. L. and Sohi, G. S. Using feedback to control tree saturation in multistage interconnection networks. *Proc. 16th Annual International Symposium on Computer Architecture* (1989), pp. 167–176.
20. Shen, J. P., and Hayes, J. P. Fault tolerance of dynamic full-access interconnection networks. *IEEE Trans. Comput.* **C-33,** 3 (Mar. 1984), pp. 241–248.
21. Tang, P., and Yew, P. C. Software combining algorithms for distributing hot-spot addressing. *J. Parallel Distrib. Comput.* **10,** 2 (Oct. 1990), 130–139.
22. Tzeng, N. F., Yew, P. C., and Zhu, C. Q. Fault-diagnosis in a multiple-path interconnection network. *Proc. 16th Annual International Symposium on Fault-Tolerant Computing System,* 1986, pp. 98–103.
23. Tzeng, N. F., Yew, P. C., and Zhu, C. Q. Realizing fault-tolerant interconnection networks via chaining. *IEEE Trans. Comput.* **C-37,** 4, (Apr. 1988), 458–462.
24. Tzeng, N. F. Alleviating the impact of tree saturation on multistage interconnection network performance. *J. Parallel Distrib. Comput.* **12,** 2 (June 1991), 107–117.
25. Yew, P. C., Tzeng, N. F., and Lawrie, D. H. Distributing hot-spot addressing in large-scale multiprocessors. *IEEE Trans. Comput.* **C-36,** 4 (Apr. 1987), 388–395.

Received March 12, 1993; accepted March 21, 1995

NENG-PIN LU received the B.S. and M.S. degrees in computer science and information engineering from the National Chiao Tung University, Taiwan, Republic of China in 1989 and 1991, respectively. Currently he is pursuing the Ph.D. degree in computer science and information engineering at the National Chiao Tung University, Hsinchu, Taiwan, Republic of China. His research interests include computer architecture, interconnection network, and parallel processing.

CHUNG-PING CHUNG received the B.E. degree from the National Cheng-Kung University, Taiwan, Republic of China in 1976, and the M.E. and Ph.D. degrees from the Texas A&M University in 1981 and 1986, respectively, all in electrical engineering. He was a lecturer in electrical engineering at the Texas A&M University while working towards the Ph.D. degree. Since 1986 he has been with the Department of Computer Science and Information Engineering at the National Chiao Tung University, Hsinchu, Taiwan, Republic of China, where he is a professor. From 1991 to 1992, he was a visiting associate professor of computer science at the Michigan State University. His research interests include computer architecture, parallel processing, and parallel compiler design.