A Comparative Study of Distributed Resource Sharing on Multiprocessors

BENJAMIN W. WAH, MEMBER, IEEE

Abstract — **In this paper we have studied the interconnection of resources to multiprocessors and the distributed scheduling of these resources. For a given interconnection network, the resource-mapping problem entails the search of one of the free resources which can be connected to each requesting processor. To prevent the bottleneck of sequential scheduling, a request without any destination address is given to the network, and the network is responsible for finding the necessary resource and connecting it to the processor. The addressing mechanism is thus distributed in the network. Three different classes of networks have been investigated: namely, single shared bus, multiple shared buses, and multistage dynamic networks. In each case, the scheduling algorithm is described, and the tradeoffs of different network configurations are studied. The resource-sharing networks are a generalization of conventional interconnection networks with routing tags in which all the resources are of different types.**

Index Terms — **Address mapping, crossbar switch, multistage dynamic network, queueing delay, resource sharing, shared bus.**

I. INTRODUCTION

RECENT advances in large-scale integrated logic and communication technology, coupled with the explosion in size and complexity of new applications, have led to the development of parallel processing systems with a large number of general- and special-purpose processing units. An example is the PUMPS architecture that contains a large number of special VLSI functional units for pattern analysis and image database management [6]. An interconnection network is an essential element of these systems as it interconnects processors and resources [10]. Its function is to route requests initiated from one point to another point connected on the network. The network topology is *dynamic*, and the links can be reconfigured by setting the network's active switching elements. The notable characteristic of these networks is that they operate with address mapping. That is, a request is initiated with a specific destination or a set of destinations, and routing is done by addresses. Routing of requests is usually done in parallel. As classified by Feng [10], these networks include the single or multistage networks and the crossbar switch. Examples are the banyan [13], indirect binary n-cube [24], cube [27], perfect shuffle [29], flip [3], Omega [17], data manipulator [9], augmented data manipulator [28], delta [8], [23], baseline [32], Benes [4], and Clos [7]. Examples of systems designed with interconnection networks are Trac [26], Staran [2], C.mmp [33], Illiac IV [16], Pluribus [22], Numerical Aerodynamic Simulation Facility (NASF) [1], and the Ballistic Missile Defense testbed [21].

In a resource-sharing environment, a request is directed to any one or more of a pool of identical resources and not to any particular element in the pool. This exists in a multiprocessor system with a set of identical (or sets of identical) VLSI chips performing special functions like matrix inversion, fast Fourier transform, and sorting. Another application lies in a system with load balancing. Processors are considered as resources themselves. When a processor is overloaded, the excess load is sent to any available processor in the system. Resource sharing is also an important element in data flow machines. Tasks in node store are sent to a pool of identical processors for processing.

To use an address-mapping network in this environment, the address of a free resource must first be sought and given to the request before it enters the network. This implies a centralized scheduler which manages the free resources, and has been studied with respect to the banyan network. In these studies, it is shown that when a processor makes a request for multiple resources, by allocating resources with smaller distance functions, the amount of network blockage caused by the allocation of these resources is reduced [15]. A tree network is proposed to aid the scheduler in choosing a resource to allocate. It has a delay of $O(m)$ in selecting a free resource (m is the total number of resources) [25]. This sequential service of requests is a major overhead in a resource-sharing environment and may become a bottleneck. This approach is practical when the number of resources is not large or when requests are not very frequent. The performance of resource-sharing systems under address mapping has also been studied by Fung and Torng [12], Marsan *et al.* [19], [20], and Willis [31]. In these studies, resources are modules that requests can be directed to. Examples include memory modules and I/O devices. Under these applications, the destination address of a request is known *a priori*. Therefore, routing of requests can be done in parallel. Furthermore, these studies assume that resources are continuously connected during their usage. As a result, it is unnecessary to have more than one resources on each output port of the network.

Another solution that avoids the sequential scheduling of requests is to send requests without any destination tags, and it is the responsibility of the network to route the maximum number of requests to the free resources. In this way, the scheduling intelligence is distributed in the network. This approach permits multiple requests to be routed simulta-

Manuscript received November 1, 1982; revised August 8, 1983. This work was supported by the National Science Foundation under Grants ECS 80-16580 and ECS 81-05968.

The author is with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

0018-9340/84/0800-0700$01.00

neously. Further, the resource can be disconnected from the processor after the task has been transmitted. The resource will continue to process the task after the disconnection until completion. Since an output port may not be fully utilized when the scheduled resource is busy, more than one resource can be connected on the same output port. We termed this network a *resource-sharing interconnection network (RSIN)* [14], [30]. It is the goal of this paper to show the various approaches to distributed resource scheduling and to study the performance of different RSIN's.

The basic principle behind distributed resource scheduling is that the status information of resources is propagated through the network and is available to all the processors. The network is made up of multiple stages of interconnected cells that possess local status information about the resources. When a request is received by a cell, it is switched through the best output port of the cell, and the updated status information is relayed back to other related cells and processors. Occasionally, a wrong decision is made because the local status information is not updated in time, and the request has to be rerouted or rejected. Through the continuous propagation of status information, requests can be serviced in parallel.

When single-resource requests are considered in a system with one type of resource, distributed control can be implemented by logic gates in interconnected cells with a small number of control lines. This is illustrated in the three classes of networks studied in this paper. When multiple types of resources are allowed in the network or when multiple resources are requested by one request, the scheduling algorithm is dependent on the number of resources in each type, the way that resources are distributed to the output ports, and the network characteristics. Further, deadlocks may occur when multiple resources are requested by a request, and distributed resolution of deadlocks may have high overhead. A complete solution is beyond the scope of this paper. Briggs *et al.* have considered the problem of choosing the number of resources in each type in which one resource is connected to each output port and one resource is requested each time [5]. In this paper we restrict our considerations to single-resource requests in a system with one type of resource. The algorithms proposed can be extended easily to systems with multiple types of resources.

In analyzing the performance, we are interested in selecting the best network type and configuration that satisfy the requirements. A performance model using a Markov chain has been developed with respect to the single shared bus. The performance of multiple shared buses under limiting conditions is approximated as multiple single-bus systems. The analytical performance of multistage dynamic networks is difficult to evaluate. Simulations have been used to evaluate these networks.

The RSIN discussed here is a generalization of address-mapping interconnection networks with routing tags [17], [27]. In an RSIN, multiple resources are allowed in each type, and the type of resource requested is used as an address or tag of a set of possible destinations. When there is one resource in each type, the type number becomes the destination address, and the network operates in an address-mapping mode.

In the next section, a classification of RSIN's is described. Sections III–V discuss the different RSIN's. In Section VI, the performance of these networks are compared. Section VII provides some concluding remarks.

II. RSIN's in a Multiprocessor System

An organization showing the use of RSIN is depicted in Fig. 1. Each processor has a connection to the network. Multiple resources may be connected on a single output port from the RSIN. The configuration of a system using RSIN's can be denoted by a triplet: processor/network/resource. The number of processors is p, and this suffices to characterize the processors (assuming they are identical). The network is characterized as $i \times j \times k$ N where i is the number of RSIN's, j (respectively, k) is the number of input (respectively, output) ports for each RSIN, and N is the network configuration. Note that $p = i \cdot j$. The single type of resources are distributed uniformly to the output ports with r resources on each output port so that the output ports are utilized evenly. The configuration of the system can thus be represented as $p/i \times j \times k$ N$/r$. As an example, consider a system with 16 processors and 32 identical resources. If the RSIN is made up of 16 private buses connecting each processor to two private resources, the configuration is described as 16/16 × 1 × 1 SBUS/2. If the RSIN is a 16-by-32 crossbar switch, there is one resource on each output port, and the system is described as 16/1 × 16 × 32 XBAR/1. Lastly, if a 16-by-16 indirect binary n-cube network is used, we have 16/1 × 16 × 16 CUBE/2.

A task is serviced in the following fashion after it is generated in a processor. It is queued at the processor until the processor has established a connection with a resource. The task is sent to the resource. After data transmission is completed, the network connection is broken, and the task is serviced at the resource until finished. After the task is serviced, the result is routed to the originating processor. This can be done by a separate address-mapping network with parallel routing since the destination address is known. In this paper we concentrate on eliminating the overhead of sequential service of requests for resources.

Circuit switching is used in RSIN's. Although the general question of whether circuit switching is better than packet switching is unsolved, the reasons for choosing circuit switching here are twofold. First, packet switching has been studied in conventional address-mapping networks in which a request is directed to a given destination. When a resource is connected continuously to a processor, the connected path may block other requests from using this resource or the network. By breaking a request into packets, the waiting time in accessing a resource can be shortened. However, in an RSIN, the issue is less critical because a request can always search for another available resource provided that the network path is free. Furthermore, the overhead of rerouting a packet when a path or resource is blocked is higher than that of rerouting a resource request. Second, and more im-

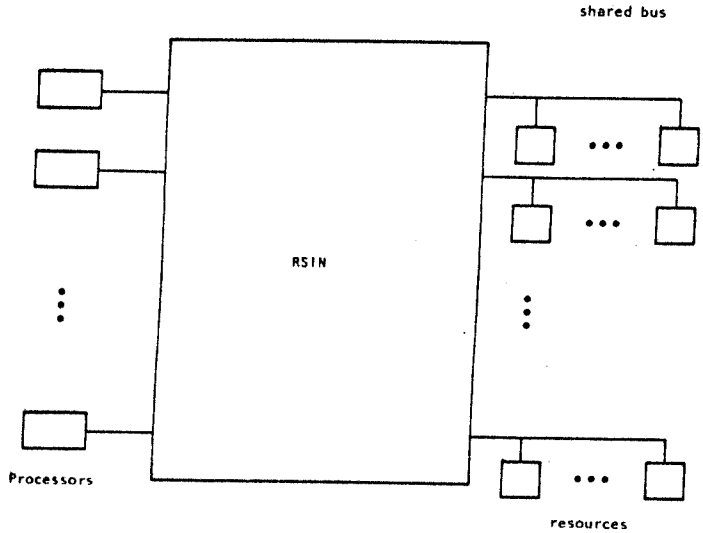

Fig. 1. RSIN as used in a multiprocessor environment.

portantly, due to the characteristics of resources, a task cannot be processed until it is completely received. The extra delay in breaking a task into multiple packets may decrease the utilization of resources, and hence, increase the response time of the system.

Tasks or requests are characterized by three values: the interarrival time of tasks in each processor, the time to transmit a task to the resource, and the time for a resource to service a task. We define the following:

$1/\lambda$ average interarrival time of tasks in each processor;

$1/\mu_n$ average time for a processor to transmit a task to the resource after the connection is established;

$1/\mu_s$ average time for a resource to service a task after data transmission is completed.

The basic assumptions made in the performance study are as follows.

(a) There is one class of tasks and their arrivals in each processor are governed by a Poisson distribution. Tasks' transmission and service times are exponentially distributed.

(b) Blocked or rejected tasks are queued at the processors and retried as soon as the network indicates that free resources are available. Task service is done in a FIFO order. No queueing is allowed at the resources.

(c) The network delay in propagating requests and status information is negligible.

(d) All the resources in the system are identical.

(e) Each request needs one resource only.

(f) A processor can transmit one task at a time to the resources. Other tasks arriving during the task transmission time are queued.

Blockages in the system are caused by two reasons regardless of whether centralized or distributed scheduling is used: namely, blockage due to shared links in the network, and blockage due to busy resources. To illustrate blockage due to the network, consider an 8-by-8 Omega network (Fig. 11) with interchange boxes that can be set to one of the two possible states: straight or exchange (broadcast connection is not needed since one resource is needed for each request). In this example assume processors 0, 1, 2 are requesting one resource each, and resources 0, 1, 2 are available. Other processors are not making requests and other resources are busy. Further, the network is completely free. All the resources will be allocated if the following processor–resource mappings are used: {(0, 0), (1, 1), (2, 2)}, {(0, 1), (1,0), (2, 2)}, {(0, 2), (1, 0), (2, 1)} or {(0, 2), (1, 1), (2, 0)}. But if the following processor–resource mappings are used: {(0, 0), (1, 2), (2, 1)} or {(0, 1), (1, 2), (2, 0)}, then a maximum of two out of three resources can be allocated without blocking. A similar example can be generated for the indirect binary n-cube network. This illustrates that the scheduler must be designed properly to give the maximum resource utilization.

The performance of the routing algorithm used in an RSIN is measured by d, the expected delay in the queue before free resources are allocated. The performance depends heavily on μ_s/μ_n, the ratio of task transmission to service times. When this ratio is large, more blocking is incurred in the network, and a more complex network has to be used. In this paper we compare three network configurations: namely, single shared bus, multiple shared buses, and Omega networks. Only distributed scheduling algorithms will be discussed.

III. RSIN's Using a Single Shared Bus

A shared bus is used to connect a subset of processors to a subset of resources. Other subsets of processors in the system cannot access resources connected for this subset. Since different subsets of processors do not interfere with each other in the accesses, the performance of each bus can be analyzed independently.

Status information of resources is communicated by the bus to processors, and tasks are transmitted over the bus from processors to resources. Every time a free resource is allocated or a busy resource completes its task, the number of free resources available on this bus is broadcast to all the connected processors via the network. This information will wake up blocked requests in the queues of processors. The first request in each queue will be sent to the network. If multiple requests are sent to the network simultaneously, an arbitrator will select one request, and the other requests are queued at the processors again. As a new request is generated in a processor, if a free resource is not available, the request is queued at the processor; otherwise, it is sent to the network. A possible implementation of the bus control is shown in the next section. In this section we analyze the performance of the single shared bus. The single-bus approach is interesting because it provides an upper bound on the queueing delay.

A queueing model of the shared bus is shown in Fig. 2. The degenerate cases of this model can be analyzed very easily using conventional methods. When μ_n is very small as compared to μ_s or when the number of resources is very large, free resources are always available and the system is modeled as an $M/M/1$ queueing system. On the other hand, when μ_s is small as compared to μ_n and the number of resources is small, the overhead in the bus is negligible and the system can be approximated by an $M/M/r$ queueing system. For cases in between, the analysis is elaborate. The reason is due to the fact that there is no buffer space at the resources, and the bus must be idle when all the resources are busy or

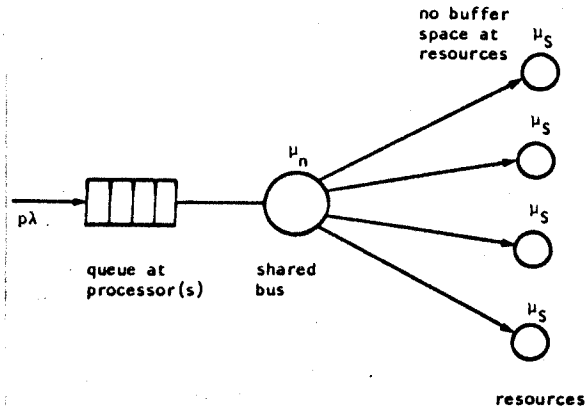

Fig. 2. A queueing model of the shared bus.

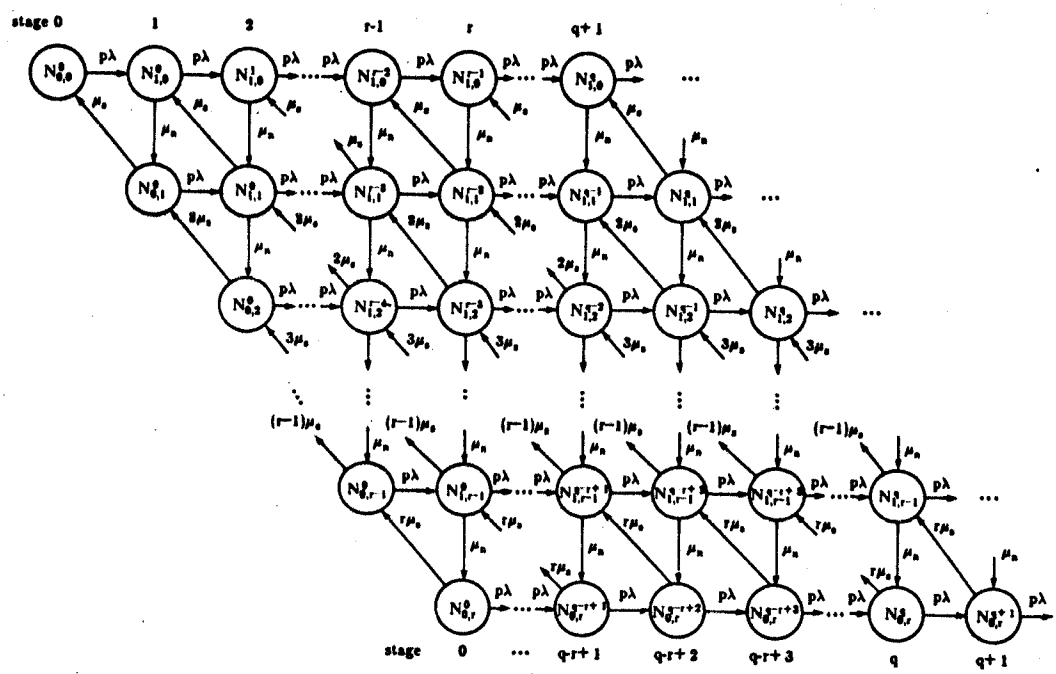

Fig. 3. State transition diagram for a $p/1 \times 1 \times 1$ SBUS/r system (with single-resource requests and one type of resources).

when no task is queued for transmission. In the remainder of this section, a Markovian analysis of the single shared bus is shown.

The state transition diagram for a $p/1 \times 1 \times 1$ SBUS/r system is depicted in Fig. 3. Each state is represented as $N^{\ell}_{n,s}$ where $\ell \in \{0, 1, 2, \cdots,\}$ is the number of queued tasks, $n \in \{0, 1\}$ is the number of tasks transmitting, and $s \in \{0, 1, \cdots, r\}$ is the number of busy resources.

In state $N^{\ell}_{n,s}, \ell > 1, n = 1, 0 < s < r - 1$, and a new task arrives (with rate $p\lambda$); the new state becomes $N^{\ell+1}_{n,s}$. Similarly, when a task in transmission is completed (with rate μ_n), the resource receiving the task begins service, and a task in the queue is immediately sent to the bus. The new state becomes $N^{\ell-1}_{n,s+1}$. When a resource finishes serving a task (with rate $s\mu_s$), the new state is $N^{\ell}_{n,s-1}$. The boundary states are those with $\ell = 0$, or $n = 0$, or $n = 1$ and $s = 0$, or $n = 1$ and $s = r - 1$. The case $n = 0$ occurs when there is no queued request or when all the resources are utilized. In the latter case, a task queued on the bus cannot begin transmission until a free resource is available. Therefore, state $N^{\ell}_{1,r-1}$ is changed to state $N^{\ell}_{0,r}$ when data transmission in the bus is completed. For states with $n = 0$, there is no μ_n transition. Likewise, for states with $s = 0$, there is no μ_s transition.

The average queueing delay can be obtained by first solving the average queue length and applying Little's formula.

$$d = \frac{1}{p\lambda} \sum_{i=1}^{\infty} i \left[\sum_{j=0}^{r-1} \Pr(N^{i}_{1,j}) + \Pr(N^{i}_{0,r}) \right] \tag{1}$$

where $\Pr(\cdot)$ is the stationary probability for a state.

To solve for the stationary probability values, we can express the probability of all the states in terms of that of an elementary state(s), and to solve for the probability of the elementary state(s) by using the relationship that all probability values sum to unity. Referring to Fig. 3, we let the set of states on a 45° column to be a stage. We designate the states on stage 0 to be the elementary states. By expressing

the relationship among states on stages $i+1$, i and $i-1$, $i > 1$, we have the following matrix equation:

$$p\lambda \begin{bmatrix} \Pr(N^{i-1}_{1,0}) \\ \Pr(N^{i-1}_{1,1}) \\ \Pr(N^{i-1}_{1,2}) \\ \vdots \\ \Pr(N^{i-1}_{1,r-1}) \\ \Pr(N^{i-1}_{0,r}) \end{bmatrix} = \begin{bmatrix} p\lambda + \mu n & -\mu_s & 0 & \cdots & & \\ 0 & p\lambda + \mu_n + \mu_s & -2\mu_s & \cdots & & \\ 0 & 0 & p\lambda + \mu_n + 2\mu_s & \cdots & & \\ \vdots & \vdots & \vdots & & & \\ & & & & p\lambda + \mu_n + (r-1)\mu_s & r\mu_s \\ & 0 & & & 0 & p\lambda + \mu_n + r\mu_s \end{bmatrix}$$

$$\times \begin{bmatrix} \Pr(N^{i}_{1,0}) \\ \Pr(N^{i}_{1,1}) \\ \Pr(N^{i}_{1,2}) \\ \vdots \\ \Pr(N^{i}_{1,r-1}) \\ \Pr(N^{i}_{0,r}) \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & \cdots & & \\ -\mu_n & 0 & 0 & \cdots & & \\ 0 & -\mu_n & 0 & \cdots & & \\ \vdots & \vdots & \vdots & & & \\ & & & & -\mu_n & 0 \\ & 0 & & \cdots & 0 & -\mu_n \end{bmatrix} \begin{bmatrix} \Pr(N^{i+1}_{1,0}) \\ \Pr(N^{i+1}_{1,1}) \\ \Pr(N^{i+1}_{1,2}) \\ \vdots \\ \Pr(N^{i+1}_{1,r-1}) \\ \Pr(N^{i+1}_{0,r}) \end{bmatrix}. \quad (2)$$

A similar boundary equation can be written for states in stage 1. It is not difficult to see that the $r+1$-by-$r+1$ matrix multiplying the states on stage $i+1$ [second term on the right-hand side of (2)] is singular. Therefore, the states on stage $i+1$ cannot be expressed in terms of states on lower stages. However, from (2), we see that states on lower stages can be expressed in terms of states on higher stages. This does not imply that the elementary states can be chosen at infinity because the stationary probabilities there approach zero. A compromise is to choose the elementary states at a sufficiently large stage $q+1$, such that the stationary probabilities of states above stage $q+1$ are approximately zero, and the stationary probabilities of states below stage $q+1$ can be solved accurately to within the precision of the computer.

There is no good method for choosing q. A simple procedure is to start with $q = 2$ and to solve for the queueing delay d (1). This is repeated for increasing values of q until d starts to decrease. At this point, the maximum precision in solving for the elementary states is attained, and the procedure terminates. The iterative procedure is compared to a procedure that solves for all the stationary probabilities simultaneously using $(r+1)(q+1)$ balance equations. The results are found to be within four digits of accuracy in all cases.

Some performance results for a system with 16 processors and 32 resources are shown in Figs. 4 and 5 for $\mu_s/\mu_n = 0.1$ and 1.0, respectively. These results are plotted with respect to the traffic intensity of a hypothetical system with a single bus of service rate 16 μ_n and a single resource of service rate 32 μ_s ($\rho_x = 16\lambda(1/(16\,\mu_n) + 1/(32\,\mu_s))$). The delay times are normalized with respect to the average task service times. The processors can be connected to the resources via a single bus, or they can be partitioned and each partition is connected via a single bus to a subset of the resources. In Fig. 4, we see that the delay is smaller as the number of partitions increases. A strange behavior is observed for the case of 16 partitions ($16/16 \times 1 \times 1$ SBUS/2). It has worse delay than the case of 2 partitions ($16/2 \times 1 \times 1$ SBUS/16) for ρ_x below 0.64, and approaches the delay for the case of 8 partitions ($16/8 \times 1 \times 1$ SBUS/4) as ρ_x increases. The reason for this is that under light loads, the bottleneck is at the resources. Therefore, systems with a smaller number of accessible resources have higher delays. Under heavy loads, the bottleneck is at the bus. Thus, systems with a smaller number of partitions have higher delays. The above phenomenon is not observed for cases of 1, 2, and 8 partitions because they have a sufficient number of resources connected, and the resources do not pose a bottleneck under light load. In Fig. 4, we have also shown the performance when each processor is connected to 3, 4, and ∞ resources via a private bus. We see that the delay is almost halved as the number of private resources for each processor is increased from 2 to 4. For infinitely many resources, the bus is the bottleneck, and the system can be modeled as an M/M/1 queue that saturates when $\rho_x = 6.0$.

The strange behavior observed when $\mu_s/\mu_n = 0.1$ does not occur when $\mu_s/\mu_n = 1.0$ (Fig. 5). In this case the bus is always the bottleneck. As the number of partitions increases, the delay decreases. Further, the improvement of using infinitely many resources is very small due to the high data-transmission time.

IV. RSIN's Using Multiple Shared Buses

The approach using multiple shared buses is a hybrid of crossbar switch and single shared bus. The RSIN has a crossbar configuration while each output port of the crossbar is connected to a single shared bus with one or more resources. In contrast to the shared bus, the crossbar switch is nonblocking and will give the highest resource utilization and the least delay. The crossbar switch is useful in providing a lower bound on the queueing delay.

In this section the cell design of a crossbar switch to

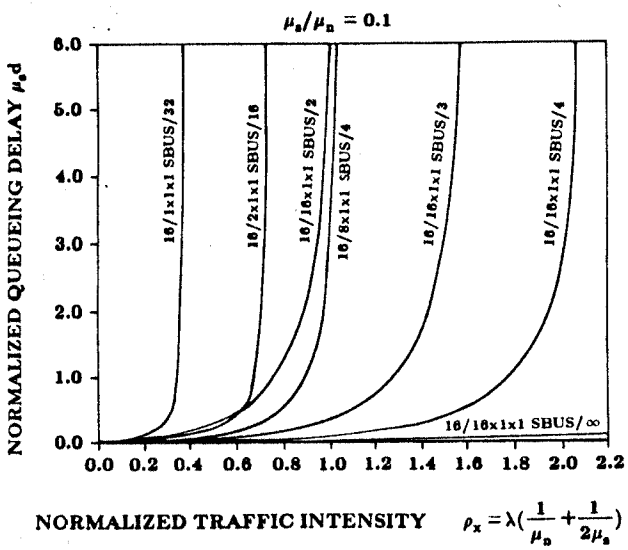

Fig. 4. Normalized queueing delay of single shared bus for $\mu_s/\mu_n = 0.1$.

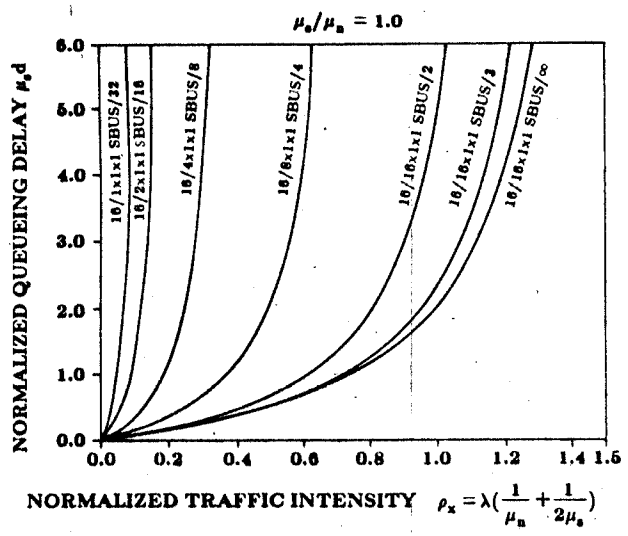

Fig. 5. Normalized queueing delay of single shared bus for $\mu_s/\mu_n = 1.0$.

support distributed resource scheduling is shown. Recall that one type of resource exists in the system, and each request needs one resource. The extension of the analysis techniques presented in the last section will also be shown on the multiple shared buses.

In Fig. 6, the overall structure of a crossbar network for distributed resource sharing is shown. Processor $i, 0 \leq i < p$, initiates a request by sending a request signal to the switch along the ith row. Resource controller $j, 0 \leq j < m$, indicates that bus j is free and at least one resource is available by sending a resource signal along the jth column. At cell $C_{i,j}$ where there are request and resource signals, the switch is set on and data transfer can begin. The request signal is removed from any further cells along the ith row. Similarly, the resource signal is removed from any further cells along the jth column. Each cell in the switch has intelligence (to be discussed) to resolve conflicts and to route requests. There is a control latch in each cell to indicate its state. It is obvious that there is no centralized control for the routing of requests.

Because requests can appear and disappear at any time, it is important that a change in request state for one processor does not affect the state of allocation of other processors. To illustrate this, referring to Fig. 6(a), if the request signal to cell $C_{i,j}$ is removed, then the latch in $C_{i,j}$ is reset, and a free resource is available. The resource signal will again propagate down the jth column. Processor k may have made a request previously and have found another resource since no resource signal was passed along the jth column. The new resource signal passed along the jth column should be ignored in $C_{k,j}$ in order not to upset the state of a previous allocation.

We also assume that the system operates in two modes: request and reset modes. In the request mode, processors can make requests for free resources. In the reset mode, processors can relinquish previously acquired resources. This method degrades performance because requests and resets cannot operate concurrently. However, a single signal line suffices to indicate which mode is active. Other alternatives that allow concurrency in requests and resets include: (a) the use of state-saving latches in each cell, and (b) the use of separate request and reset control lines. These alternatives require more hardware and will be discussed in the next section.

Referring to Fig. 6(b), the inputs and outputs of cell $C_{i,j}$ that connects processor i and bus j have the following meaning:

$X_{i,j} =$

- 0 processor i is not searching for a free resource
- 1 processor i is searching for a free resource

(request mode)

$X_{i,j} =$

- 0 processor i does not want to change the state of allocation
- 1 processor i wishes to relinquish the allocated resource

(reset mode).

$X_{i,j}$ always returns to 0 at the end of each mode;

$Y_{i,j} =$

- 0 bus j is busy or all the resources connected through bus j are busy; new request cannot be accepted.
- 1 bus j is free and a free resource on bus j is available; a new request can be accepted.

DI_i data line to send data from the ith processor;
$DO_{i,j}$ data line for resources on the jth bus to receive data from the ith processor;

$L_{i,j} =$

- 0 Latch is off; any request made by processor i is passed to the next cell, $C_{i,j+1}$.
- 1 Latch is on; processor i is connected to bus j.

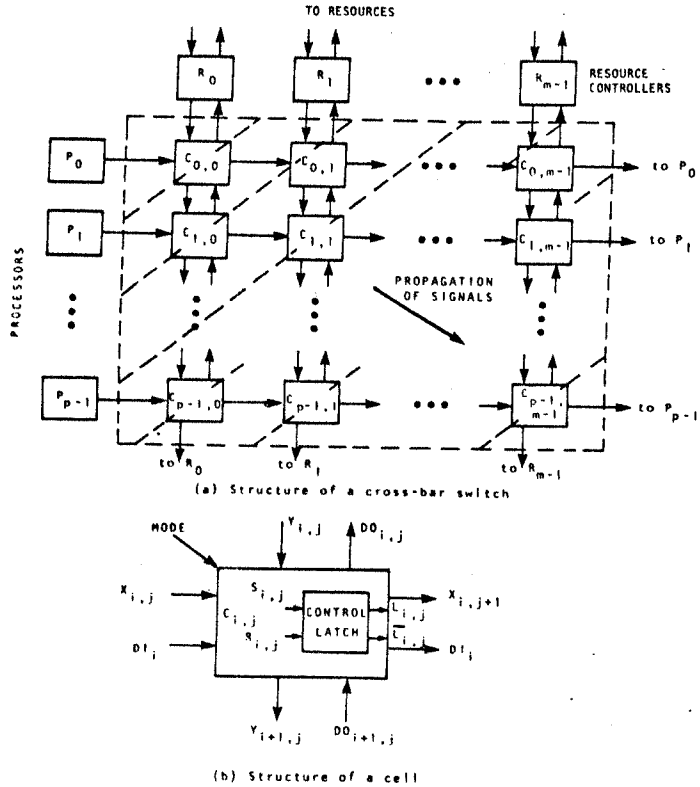

Fig. 6. A crossbar switch to support distributed resource scheduling.

$S_{i,j}/R_{i,j}$ the set/reset signal for the control latch in cell $C_{i,j}$;
MODE the signal to control the cell to be in request or reset mode.

The input/output relationship of the control signals is shown in Table I.

In the request mode, if processor i is making a request ($X_{i,j} = 1$), bus j is free, and a resource connected on bus j is available ($Y_{i,j} = 1$), then the latch is set ($S_{i,j} = 1$), and $Y_{i+1,j} = 0$. If bus j is not available or all the resources on bus j are busy ($Y_{i,j} = 0$), then the request signal is passed to the next cell ($X_{i,j+1} = X_{i,j}$), and $Y_{i+1,j} = 0$. Since the $X_{i,j}$ signal returns to 0 at the end of the request mode, but the $Y_{i,j}$ signal may still be kept at 1, so $Y_{i+1,j}$ equals the output of the control latch ($\bar{L}_{i,j}$) when $X_{i,j} = 0$ and $Y_{i,j} = 1$. For those processors that have made requests previously, the state of allocation is not disturbed in the current request mode, and data transmission can continue. In the reset mode, if processor i issues a reset signal, all the control latches in row i of the network are reset. A circuit implementation of the cell is shown elsewhere [30]. Each cell can be realized with eleven gates and one latch. The maximum gate delay in the request mode is four while the maximum gate delay in the reset mode is one.

The boundary connections for the switch are as follows. Each $X_{i,m}$ signal is connected directly back to P_i. Similarly, each $Y_{p,j}$ signal is connected back to R_j. Suppose P_i makes a request by setting $X_{i,0} = 1$, and it receives $X_{i,m} = 1$ at the end of the request cycle. This means that the request is not satisfied, and P_i should resubmit its request in the next request cycle. Likewise, R_j indicates that bus j is free and resources are available by setting $Y_{0,j} = 1$. If at the end of the request cycle, $Y_{p,j} = 1$, this means that no resource is allocated, and

TABLE I
TRUTH TABLE OF CELL IN SHAVED BUSES

MODE	Inputs		Outputs			
	$X_{i,j}$	$Y_{i,j}$	$X_{i,j+1}$	$Y_{i+1,j}$	$S_{i,j}$	$R_{i,j}$
Request	0	0	0	0	0	0
	0	1	0	$\bar{L}_{i,j}$	0	0
	1	0	1	0	0	0
	1	1	0	0	1	0
Reset	0	0	0	0	0	0
	0	1	0	1	0	0
	1	0	1	0	0	1
	1	1	1	1	0	1

R_j should send out the $Y_{0,j} = 1$ signal continuously. Otherwise, it will set $Y_{0,j} = 0$ to indicate that the bus is allocated.

Requests and resets are accepted at the beginning of the corresponding cycles. They are not accepted in the middle of a cycle because the next cycle cannot start until all the signals in the current cycle have settled. In each cycle, the signals propagate from the top left corner at 45° to the bottom right corner [Fig. 6(a)] in a wave-like motion. The maximum number of cells that the signal has to propagate is therefore $p + m$. Using the gate delay values of our design, the maximum length of the request cycle is $4(p + m)$ gate delays; the maximum length of the reset cycle is $(p + m)$ gate delays. Comparing these to a crossbar switch with a centralized scheduler, the delay to find a free resource is $O(\log_2 m)$ using a priority circuit [34], and the time to decode the location of the switch and to set the corresponding cross point is $O(\log_2(p \cdot m))$. The delay to service p requests is thus $O(p \cdot \log_2 m)$ (assuming $p < m$).

A final remark about the design is that it is asymmetric. That is, it favors processors with small index numbers. This means that processors that are located closer to the resources always have higher priority. However, it is inevitable in this approach due to the fact that request signals are initiated simultaneously at the beginning of a request cycle, and control signals are propagated sequentially. There are two solutions to this problem. First, the request cycle can be lengthened, and requests are initiated randomly within the request cycle. This degrades the performance of the system. Second, more control with separate request and reset signal lines are built into each cell so that requests and resets can be carried out concurrently. This is the approach taken in the Heidelberg POLYP Polyprocessor [18]. With this approach, a random scheduling algorithm can be implemented. A short pulse (token) indicating that resources are available on an output port is circulated continuously in the corresponding resource signal line. A requesting processor sets its request signal line. Due to the random positioning of the tokens in the network, the processor that is allocated a resource is also random. The major disadvantage of this approach is that the extra signal lines may pose a problem in VLSI implementation.

A Markovian analysis similar to that of the single bus is difficult due to the extensive number of states. For a system with m buses and r resources on each bus, the number of states in each stage is $(r + 1)^m$. The analysis method shown in the last section can only be applied when m is very small. However, we observe that under light load, each processor

generates requests and sends data to resources as if other processors are absent. As far as a processor is concerned, the crossbar switch just looks like a single shared bus with multiple resources connected because a processor can only transmit one task at a time to the resources. This implies that the analysis techniques of Section III can be applied directly under a light load. The approximate delays are very close to the simulation results for $\mu_s d \leq 1$.

Under a heavy load, the multiple buses are "partitioned" among the processors in a sense that each processor can only access a subset of the buses because all the other buses are busy. If p is the number of processors and m is the number of buses, this partitioning effect can be analyzed easily if m/p or p/m is an integer. Two cases are considered. If p is greater than m and p/m is an integer, then p/m is the number of processors assigned on a single bus. The analysis for delay is similar to that of Section III with a single bus connecting p/m processors to r resources. If p is smaller than m and m/p is an integer, then each processor is connected by m/p buses to $m \cdot r/p$ resources. As far as a processor is concerned, the multiple buses do not improve delay over a single bus. The analysis for delay is similar to that of Section III with a single bus connecting one processor to $m \cdot r/p$ resources. The heavy load approximation is found to be satisfactory when $\mu_s d$ is large. Simulations are used for cases in between.

Some performance results of the crossbar switch are depicted in Figs. 7 and 8. When μ_s/μ_n is small, the resources are the bottleneck. Partitioning the network into multiple smaller crossbar switches reduces the cost and has relatively small effects on delay except when the load is heavy. On the other hand, when μ_s/μ_n is large, the network is the bottleneck. Therefore, using a private output port for each resource results in smaller delay than the case of using shared output ports for resources. Partitioning the network and increasing the number of resources have relatively small impact on delay except when the load is heavy.

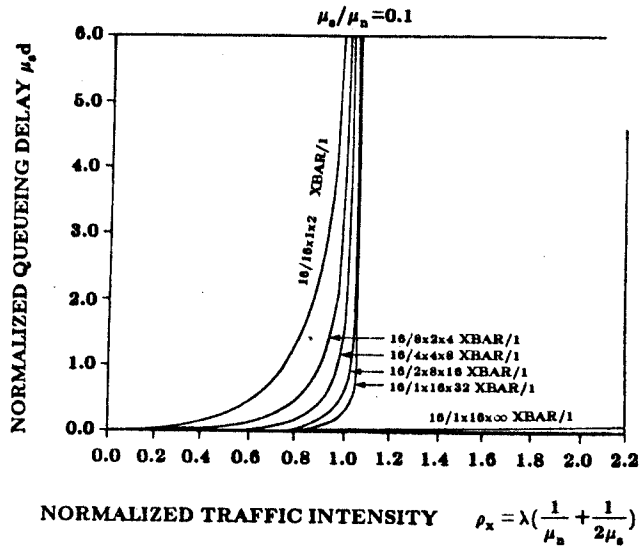

Fig. 7. Normalized queueing delay of multiple shared buses for $\mu_s/\mu_n = 0.1$.

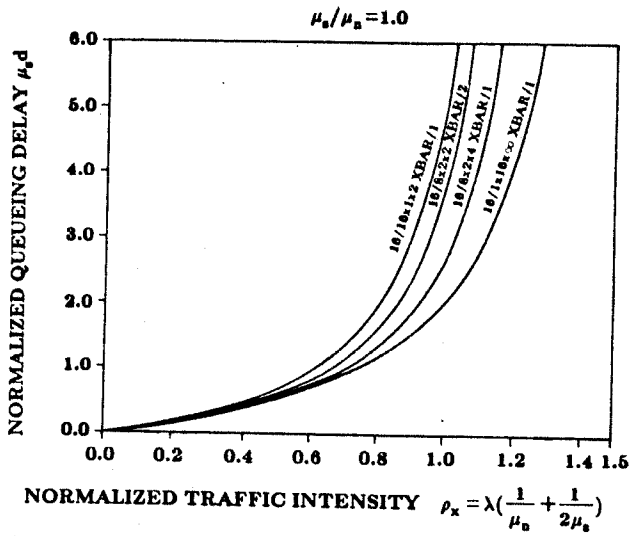

Fig. 8. Normalized queueing delay of multiple shared buses for $\mu_s/\mu_n = 1.0$.

V. RSIN's Using Multistage Dynamic Networks

The multistage dynamic networks are made up of multiple stages of active switching elements and passive links. They are classified into blocking, rearrangeable nonblocking, and nonblocking networks [10]. In blocking networks, simultaneous connections of more than one terminal pair may result in conflicts in the use of links. In rearrangeable nonblocking networks, a new input–output connection pair can always be established provided that some rearrangements on existing connections are made. Lastly, a network that can handle all possible connections is called nonblocking.

To utilize a multistage dynamic network for distributed resource sharing, the scheduling intelligence is implemented in the switching elements. Each element is responsible for passing the status information of resources to processors and to schedule a request to an output port with free resources. Regardless of whether the network is blocking or nonblocking, a wrong decision may be made because the scheduling algorithm in a switching element may use outdated status information in routing a request. Therefore, a request may have to be rerouted or rejected when a reject signal on this request is received. We present in this section the functional design of a switching element for scheduling single-resource requests in a system with a single type of resources. Although the algorithm is discussed with respect to an Omega network, the design is applicable to other types of multistage networks as well.

The Omega networks [17] belong to a class of blocking networks with the property that the delay from a source to any reachable destination is proportional to the logarithm of the number of source points. The basic element in these networks is a 2-input 2-output 2-function interchange box that allows a straight or exchange connection (broadcast connection is not needed here since one resource is needed for each request). Each interchange box can be regarded as a 2×2 crossbar switch. For a network connecting N inputs to N outputs (N is a power of 2), there are $\log_2 N$ stages and $N/2 \times \log_2 N$ interchange boxes. The delay in these networks is thus $O(\log_2 N)$. The $O(N \times \log_2 N)$ hardware complexity is much better than that of the crossbar switch ($O(N^2)$).

As discussed in Section II, some of the mappings from sources to destinations do not lead to maximal resource allocation. A centralized scheduler using exhaustive enumeration would have to examine all the different possible ordered mappings in order to allocate the maximum number of resources. Suppose x processors are making requests and y resources are free. The scheduler has to try a maximum of $\binom{x}{y} y!$ (for $x \geq y$) or $\binom{y}{x} x!$ (for $y > x$) mappings in order to find the best one. Suboptimal heuristics can be used [30], but this is only practical when x and y are small. Polynomial-time optimal scheduling algorithms will be discussed in a future paper [35].

On the other hand, a distributed scheduling algorithm allows all the requests to be scheduled in parallel. The resource scheduling overhead is therefore proportional to the delay time in the network ($O(\log_2 N)$) and independent of the number of requesting processors.

Before the algorithm is described, some symbols must be defined. Functionally, there are five control signals for each interchange box:

Q = a resource-request signal;
L = a resource-release signal;
S = number of resources reachable from this link;
J = a resource-reject signal;
C = a resource-found signal.

These control signals are indicated in Fig. 9. The first subscript in the notations indicates the stage at which the signal originates. The second subscript indicates that the signal is originated from or directed to the upper/lower half of the box. The index of the box j is implicit and not included in the notations.

Since the state of these control signals does not change continuously, the number of interstage control links can be reduced by storing these control signals in registers and multiplexing the signals on a single bidirectional control link at each output port. In particular, there are resource-availability registers A_j in each box that stores the number of resources reachable from the top ($j = 1$) or bottom ($j = 2$) output port. Since it is assumed that each request needs one resource, the status information can be simplified to one bit indicating whether at least one free resource is reachable. This is valid because a request is propagated as long as one free resource is available.

The control algorithm for each interchange box is written in pidgin Algol and is shown in Fig. 10. The status of resources reachable from the two output ports (0 or 1) is ORed at the beginning and at the end of the loop. If any change is detected, this status information is passed back to the previous stage. This allows status change to be propagated as early as possible. Status signals on requests include release, reject, query, and resource-found and are serviced in this order. When a connection is released, the status information does not change because resources may still be processing the tasks. Rejects are serviced before queries because they belong to requests that have waited longer and therefore should have higher priority. After a query is sent to an output port, the corresponding resource-availability register is zeroed because resources are no longer accessible from this port. The

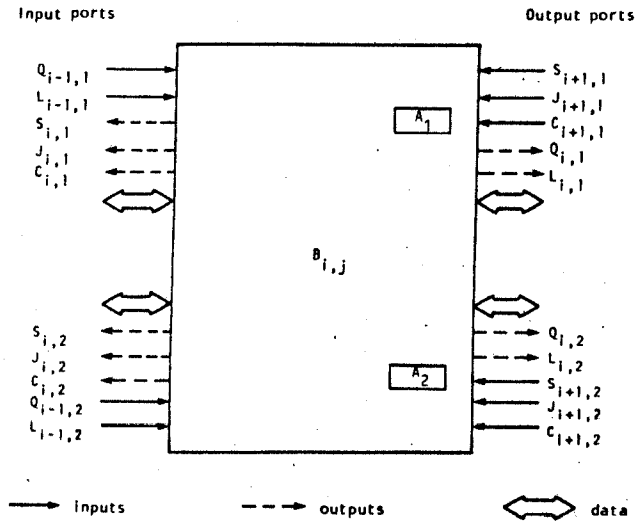

Fig. 9. Control signals for a 2 × 2 interchange box.

```
PROCESS net (i, j);
    /* Distributed scheduling algorithm in a 2x2 interchange box B(i,j) (refer to Figure 9). Each request needs one resource in a system with a single type of
    resources. */
    WHILE (true) DO
        BEGIN
        wait (arrival of any control signal);
        /* Calculate status of resources reachable from the output ports by ORing the contents of A1 and A2. */
        /* Service status signal (S) change:
        Store S(i+1,1) and S(i+1,2) into the availability registers A1 and A2. */
        /* Service release (L):
        If release(s) is received, send release(s) to appropriate output port(s) in stage i+1 */
        /* Service reject (J):
        All rejects are serviced at the input ports. For each reject, select an output port with non-zero availability register randomly. Zero the corresponding availability register and send query. Continue searching until all the rejects are serviced. For those rejects that cannot be satisfied, send rejects along the original input paths over which the queries were sent, and eliminate the corresponding queries from this interchange box. */
        /* Service query (Q):
        Queries are serviced in a similar fashion as rejects.
        /* Service resource-found (C):
        Send a resource-found signal to stage i-1 along the original input path over which the query was sent */
        /* Send status signals to the previous stage if any change is detected:
        Calculate status of resources reachable from the output ports by ORing the contents of A1 and A2. If this is different from the total calculated previously, send the new status along the status links to stage i-1. */
        END;
END PROCESS
```

Fig. 10. Control algorithm in a 2 × 2 interchange box for multistage dynamic networks.

algorithm discussed is general enough to be applicable to interchange boxes with any number of input and output ports.

We illustrate the algorithm with the 8 × 8 Omega network in Fig. 11. Suppose resources R_0, R_1, R_4, and R_5 are available and processors P_0, P_3, P_4, and P_5 are requesting. The algorithm can be considered to operate in two phases. In the first phase, status information is passed from the resources to processors. For example, $B_{2,2}$ receives 1 on both output ports from R_4 and R_5. This means that one resource is reachable from each output port of $B_{2,2}$. From the input ports of $B_{2,2}$, at least one resource is reachable. Therefore, this status information is passed to interchange boxes in stage 1. Similarly, status information is propagated in all the other boxes. At the end of phase 1, every processor knows that at least one re-

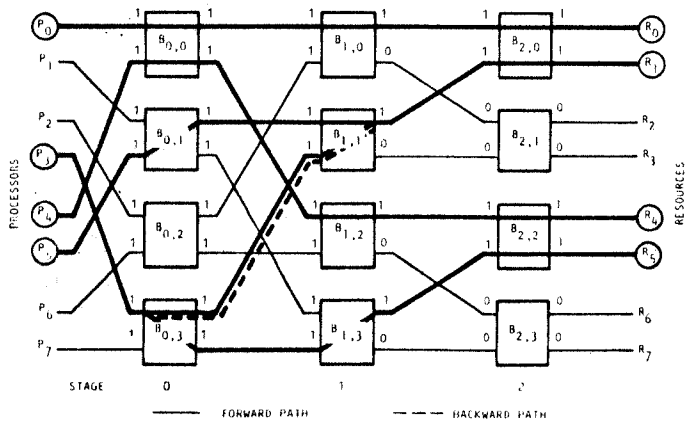

Fig. 11. Example of an Omega network supporting single-resource requests in a system with a single type of resources (circled processors are making requests; circled resources are available; status information are shown at the input/output ports of interchange boxes).

source is available. In phase 2, the processors propagate the requests to the resources simultaneously. In stage 0, no conflict is encountered. $B_{1,1}$ in stage 1 receives two requests. Since only one output port is available, one of the requests is propagated, and the other request is rejected. This request, subsequently, finds another route via $B_{1,3}$ and $B_{2,2}$ to R_5. In this example, each request has to pass through 3.5 interchange boxes on the average before it finds a free resource. In practice, the two phases operate concurrently. When the status information is changed in an interchange box, it is propagated immediately back to the previous stage. Requests continue to propagate in the presence of possibly outdated status information. This tends to lengthen the time to find a free resource. For clarity, status changes are not indicated in the figure.

To calculate the overhead of the algorithm, assume that there are r input–output ports in an interchange box and that control signals are passed bit serially between stages. The size of the status signal passed from one box to another is one bit. $2\lceil \log_2 r \rceil$ stages of OR operations are necessary in each iteration in order to calculate the changes in status information. It takes a maximum of $O(r \cdot \log_2 r)$ time to service the queries/rejects. Further, $O(1)$ time is needed to send control signals such as rejects and releases. The worst case time in each stage is $O(r \times \log_2 r)$. Since r is usually very small ($r = 2$ for Omega networks), the worst case delay of propagating N requests through the network of $\lceil \log_2 N \rceil$ stages is $O(\log_2 N)$. In contrast, in a centralized algorithm, it takes $O(\log_2 N)$ time to find a free resource and to set the switches in the network. Due to blockage in the network, $O(N)$ trials have to be made before a successful connection can be established. The delay for servicing N requests is $O(N^2 \log_2 N)$.

Basically, an interchange box is a crossbar switch, and the complexity of implementation is high when r is large. This box can be implemented by adding some peripheral controls to the distributed logic that is shown in the last section.

One peculiar characteristic of the network is that status changes always arrive at the processors simultaneously since all the boxes are clocked. Requests queued at processors therefore enter into the network simultaneously. This may cause undue conflict. A solution is to initiate requests with a random delay after the arrival of new status information.

The proposed algorithm can be adapted to systems with single-resource requests and multiple types of resources. In this case, control signal Q has to be augmented by the type of resource requested, and status signal S has to be sent for each type of resource. The number of resource-availability registers at each output port of an interchange box is increased so that there is one register for each type of resources reachable from this output port. The status change in each interchange box is also computed with respect to each type of resource. The overhead of the scheduling algorithm for a network of $\log_2 N$ stages and t types of resources is $O(t \times \log_2 N)$.

The analytical delay characteristics of multistage networks are difficult to evaluate. Delta networks with address mapping and packet switching have been analyzed under heavy load [8] or lost packets [23]. Behavior under circuit switching has been studied by simulations [11]. In an RSIN, the complexity of analysis is worse than that under address mapping. As a result, we study the performance using simulations alone.

The delay characteristics of the Omega networks are plotted in Figs. 12 and 13. In connecting 16 processors to 32 resources, there is very little difference in delay between using eight 2×2 networks or one 16×16 network except when the load is heavy. Therefore, it is cost effective to use multiple small networks in connecting the processors and resources together. We also observe that the blocking probability is reduced as compared to a conventional Omega network with address mapping. The average blocking probability obtained is 0.15 for an 8×8 network [14] as compared to approximately 0.3 for a similar network under address mapping [11]. This is due to the fact that a request can always search for another available resource when a particular path is blocked. The blocking probability obtained above is based on random sets of requesting processors and available resources and the fact that the network is free. This is true when μ_s/μ_n is small. When μ_s/μ_n is large, the utilization of the network is high, and the blocking probability increases.

VI. Comparison of Different RSIN's

In this section, we discuss the tradeoffs of different RSIN's. The tradeoffs have to be made with respect to the relative cost of resources and networks and the ratio of task transmission to service times μ_s/μ_n.

If the cost of resources is small as compared to the cost of an RSIN, the obvious solution is to connect a large number of resources to each processor by a private bus. As we have seen in Section III, this results in the least cost and delay.

If the cost of resources is large as compared to the cost of an RSIN, then for a given number of resources, the problem is to find the most efficient RSIN. As seen from the performance results, the multiple-private-bus approach has the worst delays. The Omega networks with distributed resource sharing have reduced blocking probability and are very favorable as compared to crossbar switches when μ_s/μ_n is relatively small ($\simeq 1$). As seen from Figs. 7, 8, 12, and 13, the

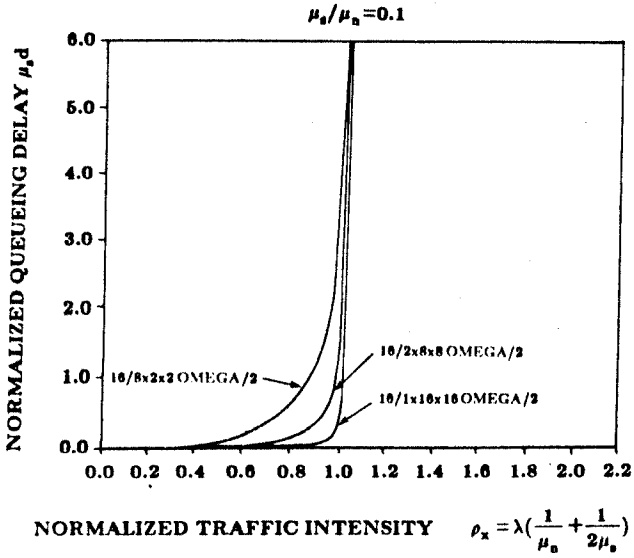

Fig. 12. Normalized queueing delay of Omega networks for $\mu_s/\mu_n = 0.1$.

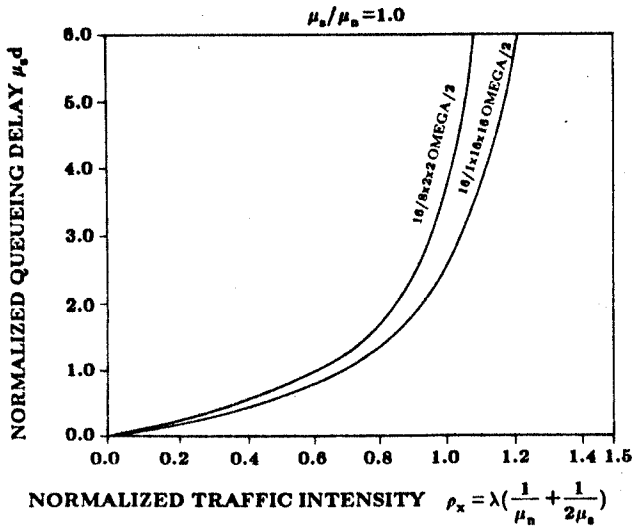

Fig. 13. Normalized queueing delay of Omega networks for $\mu_s/\mu_n = 1.0$.

delay only increases slightly when the load is light. When the load is heavy, the resources are the bottleneck, and the Omega and crossbar networks have almost identical delay characteristics. When μ_s/μ_n is large, the network is the bottleneck, and crossbar switches have smaller blockage than Omega networks. The choice, therefore, depends on the cost of implementation and the value of μ_s/μ_n in the application. Crossbar communication networks have been shown to compare favorably to banyan-type networks for VLSI implementation provided that the whole network is implemented on one chip [11]. When the network is built on multiple chips, banyan-type networks are still less expensive.

If the cost of resources is about the same as the cost of an RSIN, the choice is more difficult. In this case, a large number of small interconnection networks coupled with a larger number of resources will give good performance. This is illustrated in our evaluations which show that a 16/16 × 1 × 1 SBUS/3 system has a much better delay behavior than a 16/4 × 4 × 4 OMEGA/2 or a 16/4 × 4 × 4 XBAR/2 system. The network to be used depends on μ_s/μ_n. The performance results we have obtained can guide the designers in selecting the appropriate configuration.

TABLE II
SELECTION OF SUITABLE RSIN

RELATIVE COSTS	VALUE OF μ_s/μ_n	NETWORKS TO BE USED
$COST_{net} << COST_{res}$	small	single multistage network
	large	single crossbar network
$COST_{net} \simeq COST_{res}$	small	large number of small multistage networks and a larger number of resources
	large	large number of small crossbar networks and a larger number of resources
$COST_{net} >> COST_{res}$	all	private bus with a large number of resources

In summary, the multiple-private-bus approach is attractive when the cost of resources is not high. When resources are expensive, the multistage or crossbar networks are good candidates of RSIN's. The choice between multistage networks and crossbar switches depends on the value of μ_s/μ_n. Table II summarizes the networks to be used in different situations.

VII. Conclusion

In this paper distributed scheduling algorithms for resources are studied. Resource sharing differs from conventional accesses through addresses in that a request is directed towards any one (or more) of a pool of free resources. A centralized scheduling algorithm can be used to search for the addresses of free resources and supply them to the requests. A conventional address-mapping network can be used. The scheduler is a potential source of bottleneck because requests are serviced sequentially. On the other hand, a distributed scheduling algorithm allows requests to be scheduled in parallel with a delay time that is proportional to the network delay and independent of the number of requests.

We have studied systems with single-resource requests and a single type of resources. We have developed queueing analysis for the single shared bus that is extendible to limiting cases in multiple shared buses. The Omega networks are evaluated by simulations due to the extensive number of states. These results are useful in determining the network configuration suitable for resource sharing.

Scheduling of multiresource requests is not studied here due to the overhead and complexity in passing status information and resolving deadlocks. The algorithms presented in this paper can be extended easily to systems with multiple types of resources. The request and status signals have to be augmented by a type number. However, the problem on the number and placement of each type of resources in the network is still open. In the degenerate case of resource sharing, each output port is connected to a different type of resource. In this case, the resource type number in a request defines uniquely the destination address of the request. Resource accesses are therefore a generalization of the conventional address-mapping accesses.

References

[1] G. H. Barnes and S. F. Lundstrom, "Design and validation of a connection network for many-processor multiprocessor systems," *Computer*, vol. 14, pp. 31–41, Dec. 1981.

[2] K. E. Batcher, "STARAN parallel processing system hardware," in *Proc. AFIPS 1974 Nat. Comput. Conf.*, May 1974, vol. 43, pp. 405–410.
[3] ——, "The flip network in STARAN," in *Proc. 1976 Int. Conf. on Parallel Processing*, MI, 1976, pp. 65–71.
[4] V. Benes, *Mathematical Theory of Connecting Networks*. New York: Academic, 1965.
[5] F. A. Briggs, M. Dubios, and K. Hwang, "Throughput analysis and configuration design of a shared-resource multiprocessor system: PUMPS," in *Proc. 8th Annu. Symp. on Comput. Arch.*, 1981, pp. 67–79.
[6] F. A. Briggs, K. S. Fu, K. Hwang, and B. W. Wah, "PUMPS architecture for pattern analysis and image database management," *IEEE Trans. Comput.*, vol. C-31, pp. 969–983, Oct. 1982.
[7] C. Clos, "A study of nonblocking switching networks," *Bell Syst. Tech. J.*, vol. 32, pp. 406–424, 1953.
[8] D. M. Dias and J. R. Jump, "Analysis and simulation of buffered delta networks," *IEEE Trans. Comput.*, vol. C-30, pp. 273–282, Apr. 1981.
[9] T. Y. Feng, "Data manipulating functions in parallel processors and their implications," *IEEE Trans. Comput.*, vol. C-23, pp. 309–318, Mar. 1974.
[10] ——, "A survey of interconnection networks," *Computer*, pp. 12–27, Dec. 1981.
[11] M. A. Franklin, "VLSI performance comparison of banyan and cross-bar communication networks," *IEEE Trans. Comput.*, vol. C-30, pp. 283–291, Apr. 1981.
[12] F. Fung and H. Torng, "On the analysis of memory conflicts and bus contentions in a multiple-microprocessor system," *IEEE Trans. Comput.*, vol. C-28, pp. 28–37, Jan. 1979.
[13] L. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," in *Proc. 1st Annu. Comput. Arch. Conf.*, Dec. 1973, pp. 21–28.
[14] A. Hicks, "Resource scheduling on interconnection networks," M.S. thesis, Purdue Univ., W. Lafayette, IN, Aug. 1982.
[15] R. Jenevein, D. Degroot, and G. J. Lipovski, "A hardware support mechanism for scheduling resources in a parallel machine environment," in *Proc. 8th Annu. Symp. on Comput. Arch.*, May 1981, pp. 57–66.
[16] D. J. Kuck, "ILLIAC IV software and application programming," *IEEE Trans. Comput.*, vol. C-17, pp. 746–757, Aug. 1968.
[17] D. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.*, vol. C-24, pp. 215–255, Dec. 1975.
[18] R. Männer and B. Deluigi, "The Heidelberg POLYP—A flexible and fault-tolerant poly-processor," *Comput. Phys. Commun.*, vol. 22, pp. 279–284, 1981.
[19] M. A. Marsan and F. Gregoretti, "Memory interference models for a multimicroprocessor system with a shared bus and a single external common memory," *Microprocessing and Microprogramming—EUROMICO J.*, vol. 7, Feb. 1982.
[20] M. A. Marsan and M. Gerla, "Markov models for multiple bus multiprocessor systems," *IEEE Trans. Comput.*, vol. C-31, pp. 239–248, Mar. 1982.
[21] W. C. McDonald and J. M. Williams, "The advanced data processing test bed," in *Proc. COMPSAC 78*, Nov. 1978, pp. 346–351.
[22] S. M. Ornstein *et al.*, "Pluribus—A reliable multiprocessor," in *Proc. AFIPS 1975 Nat. Comput. Conf.*, pp. 551–559.
[23] J. H. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Trans. Comput.*, vol. C-20, pp. 771–780, Oct. 1981.
[24] M. C. Pease, "The indirect binary *n*-binary *n*-cube microprocessor array," *IEEE Trans. Comput.*, vol. C-26, pp. 458–473, May 1977.
[25] B. D. Rathi, A. R. Tripathi, and G. J. Lipovski, "Hardwired resource allocators for reconfigurable architectures," in *Proc. 1980 Int. Conf. on Parallel Processing*, Aug. 1980, pp. 109–117.
[26] M. C. Sejnowski *et al.*, "Overview of the Texas reconfigurable array computer," in *Proc. AFIPS Conf.*, 1980, vol. 49, pp. 631–642.
[27] H. J. Siegel and R. J. McMillen, "The multistage cube: A versatile interconnection network," *Computer*, vol. 14, pp. 65–76, Dec. 1981.
[28] ——, "Using the augmented data manipulator network in PASM," *Computer*, vol. 14, pp. 25–33, Feb. 1981.
[29] H. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, pp. 153–161, Feb. 1971.
[30] B. W. Wah and A. Hicks, "Distributed scheduling of resources on interconnection networks," in *Proc. Nat. Comput. Conf.*, 1982, pp. 697–709.
[31] P. J. Willis, "Derivation and comparison of multiprocessor contention measures," *IEE J. Comput. Digital Techniques*, Aug. 1978.
[32] C. Wu and T. Y. Feng, "On a class of multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-29, pp. 694–702, Aug. 1980.
[33] W. A. Wulf and C. G. Bell, "C.mmp—A multi-mini processor," in *Proc. AFIPS 1972 Fall Joint Comput. Conf.*, vol. 41, pp. 765–777.
[34] C. C. Foster, "Determination of priority in associative memories," *IEEE Trans. Comput.*, vol. EC-17, pp. 788–789, Aug. 1968.
[35] J.-Y. Juang and B. W. Wah, "Optimal scheduling algorithms for resource-sharing interconnection networks," in *Proc. COMPSAC*, Nov. 1984.

Benjamin W. Wah (S'74–M'79), for a photograph and biography, see p. 268 of the March 1984 issue of this TRANSACTIONS.