A COMPARATIVE STUDY OF DISTRIBUTED RESOURCE SHARING ON MULTIPROCESSORS*

Benjamin W. Wah
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

Abstract

In this paper, we have studied the interconnection of resources to multiprocessors and the distributed scheduling of these resources. Three different classes of interconnection networks have been investigated; namely, single shared bus, multiple shared buses, and networks with logarithmic delays such as the cube and Omega networks. For a given network, the resource mapping problem entails the search of one (or more) of the free resources which can be connected to each requesting processor. To prevent the bottleneck of sequential scheduling, the type(s) and number(s) of resources desired by a processor are given to the network and it is the responsibility of the network to find the necessary resources and connect them to the processor. The addressing mechanism is, thus, distributed in the network. This is a generalization of conventional interconnection networks with routing tags in which all the resources are of different types.

Keywords and phrases: address mapping, cross-bar switch, Omega and cube networks, queueing delay, resource sharing, shared bus.

1. INTRODUCTION

The 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 interconnection network is an essential element of a parallel processing system as it interconnects processors and resources. Its function is to route requests initiated from one point to another point connected on the network [5,8,11,14,15,17,21]. 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. Examples of these networks are the Banyan [7], binary n-cube [15], cube [18], perfect shuffle [20], flip [3], Omega [11], data manipulator [5], augmented data manipulator [19], delta [14], and baseline [21]. Examples of systems designed with interconnection networks are TRAC [17], STARAN [2], C.mmp [22], ILLIAC IV [10], PLURIBUS [13], Numerical Aerodynamic Simulation Facility (NASF) [1,4] and the Ballistic Missile Defense testbed [12].

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 dataflow 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. This has been studied with respect to the Banyan network [9,16]. 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 [8]. A tree network is proposed to aid the scheduler in choosing a resource to allocate and has a delay of O(n) in selecting a free resource (n is the total number of resources) [16]. The major disadvantage of this approach is that the scheduler can become a bottleneck since it services requests sequentially. 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 elsewhere [25,28,29,30,31]. 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.

Another solution which avoids the sequential scheduling of requests is to allow requests to be sent 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 interconnection network. This approach permits multiple requests to be routed simultaneously. We termed this network a *resource sharing interconnection network (RSIN)* [23,24]. It is the goal of this paper to study the tradeoffs of different RSINs. Three classes of interconnection networks that include single shared bus, multiple shared buses and networks with logarithmic delays such as cube and Omega networks, have been investigated. In each case, the distri-

* This research was supported by National Science Foundation Grants ECS 80-16580 and ECS 81-05968.

buted control algorithm is described and illustrated. The performance of the single shared bus is analyzed using Markovian models while the performance of multiple shared buses is approximated as multiple single shared bus systems. The analytical performance of cube type networks is difficult and they have only been evaluated using simulations.

The RSIN discussed here is a generalization of address mapping interconnection networks with routing tags [11,18]. An address mapping network is a RSIN connecting processors and multiple types of resources with one resource in each type. In a resource sharing mode, multiple resources are allowed in each type.

In the next section, a classification of RSINs is described. Sections 3 to 5 discuss the different RSINs. In section 6, the performance of these networks are compared. Section 7 provides some concluding remarks.

2. RSINs in a Multiprocessor System

An organization showing the use of RSIN is depicted in Figure 1. Each processor has a connection to the network. Multiple resources may be connected on a single output port from the RSIN. The reasons for multiple resources to share a single output link are that each task may request multiple resources simultaneously, and an output link may not be fully utilized by a single resource.

A configuration of RSIN can be characterized by a triplet: $p/i{\times}j{\times}k$ N$/r$ where p is the number of processors, r is the number of resources per output port and N is the network configuration. For the network N, i the number of RSINs, and j/k is the number of input/output ports for each RSIN. As an example, a system has 16 processors and 32 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 cross-bar 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 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 sufficient number of resources. The task is sent to the resource(s). After data transmission is completed, the network connection is broken and the task is serviced at the resource(s) until finished. The results of processing are routed to the processor through a common memory or an address mapping network.

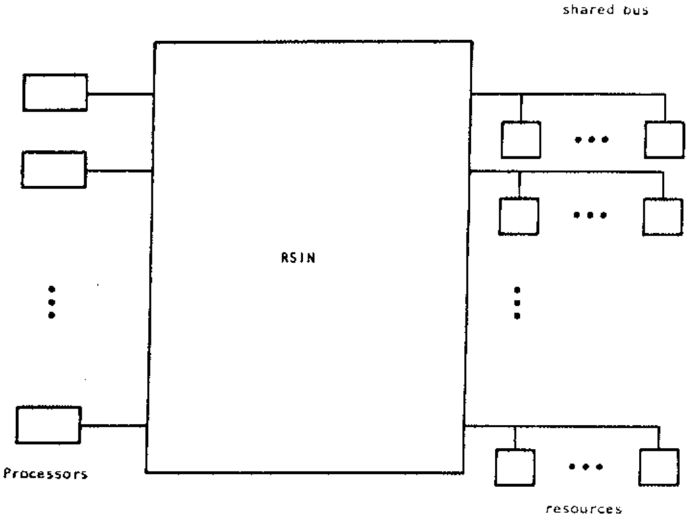


Figure 1. RSIN as used in a multiprocessor environment.

Tasks or requests are characterized by three values: the inter-arrival time of tasks in each processor, the time to transmit a task to the resource(s) and the time for a resource to service a task. We define,

$1/\lambda$ – average inter-arrival time of tasks in each processor;

$1/\mu_n$ –average time for a processor to transmit a task to the resource(s) 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 this study are:

(1) 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.

(2) 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 FIFO order. No queueing is allowed at the resources.

(3) The network delay is negligible. This assumption is made so that we can isolate the performance of the network due to blockages alone.

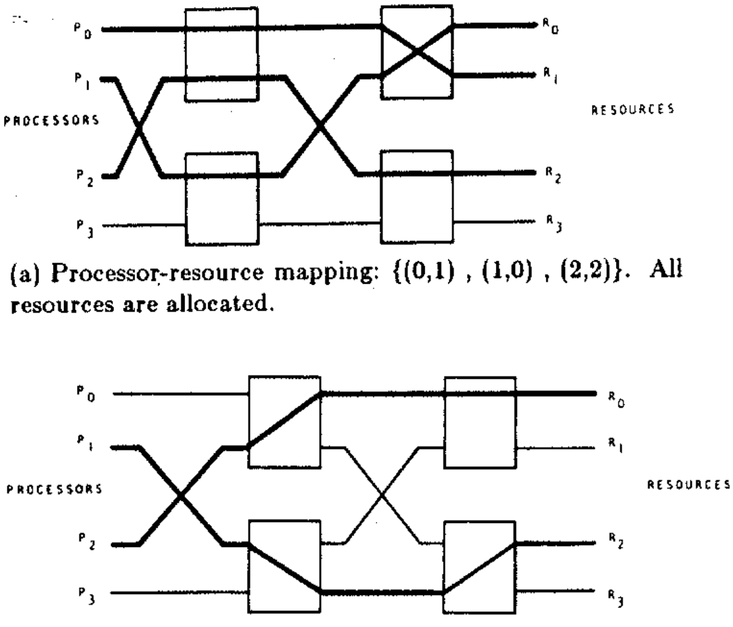
(4) All the resources in the system are identical. For multiple types of resources, the routing algorithm has to be modified by associating a routing tag corresponding to the resource type with each request.

(5) A task can request multiple resources simultaneously with a restriction that the maximum number of resources requested cannot exceed the number of resources accessible through the network. Because we want to compare the performance of processors with private versus shared resources, and the number of resources accessible in a system with private resources is very limited, we make the simplifying assumption that each task requests one resource in the performance analysis. However, the algorithm for requesting multiple resources will be discussed in systems with shared resources.

(6) 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 the shared links in the network and blockage due to busy resources. To illustrate blockage due to the network, consider a 4 by 4 Omega network (Figure 2) with interchange boxes that can be set to one of the four possible states: straight, exchange, upper broadcast, and lower broadcast. In this example, assume processors 0, 1, 2 are requesting one resource each and resources 0, 1, 2 are available. Processor 3 is not making a request and resource 3 is 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 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. In this paper, we compare three network configurations, namely, single shared bus, multiple shared buses, and Omega and cube networks. Only distributed scheduling algorithms will be discussed.



(a) Processor-resource mapping: {(0,1) , (1,0) , (2,2)}. All resources are allocated.

(b) Processor-resource mapping: {(0,1) , (1,2) , (2,0)}. Only 2 of the resources are allocated.

Figure 2. A RSIN using 4 by 4 Omega network.

3. RSINs Using 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 free resources are allocated or busy resources complete their tasks, 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, and the first request in each queue that requests less resources than what is available will be sent to the network. If multiple requests are sent to the network simultaneously, an arbitrator will select one request at random and the other requests are queued at the processors again. As a new request is generated in a processor, if the number of free resources available is less than what is requested, the request is queued at the processor until sufficient resources are available; otherwise, it is sent to the network.

When task transmission time is very small as compared to task service time, the single bus approach is the best. Otherwise, it is the major source of bottleneck in the system. The private resource approach is feasible when resources are plentiful. However, it is still expensive as the number of processors becomes large and the number of types of resources increases. It will be more efficient if processors can share the available resources in the system. 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 Figure 3. 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 modelled 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 when no task is queued for transmission. In the remainder of this section, a Markovian analysis of the single shared bus is shown.

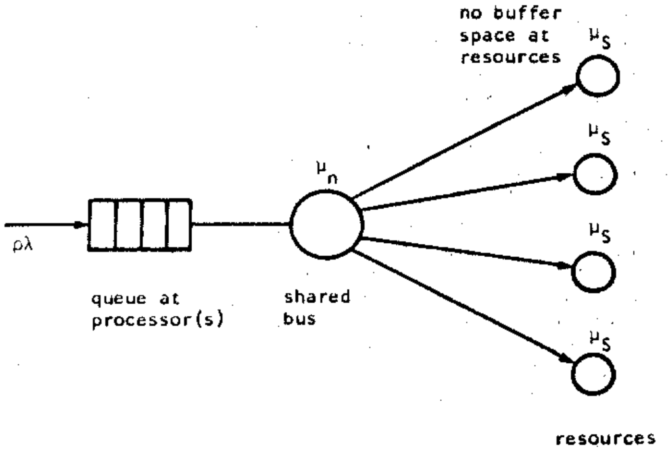


Figure 3. A queueing model of the shared-bus.

The state transition diagram for p/1×1×1 SBUS/r system is depicted in Figure 4 (assuming each task requests the use of one resource). Each state is represented as $N^{\ell}_{n,s}$ where $\ell \epsilon \{0,1,2,...\}$ is the number of queued tasks; $n \epsilon \{0,1\}$ is the number of task transmitting; and $s \epsilon \{0,1,...,r\}$ 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

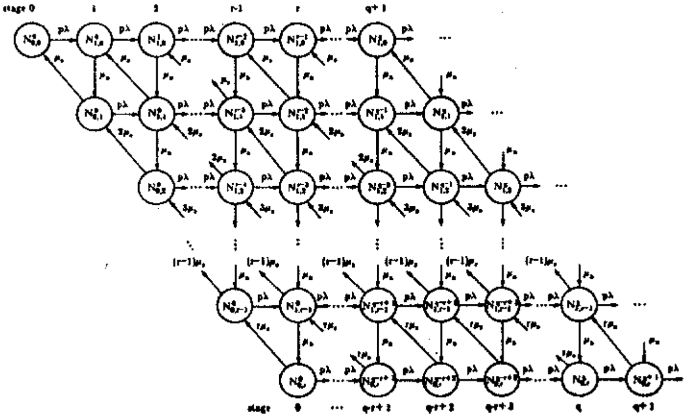


Figure 4. State transition diagram for a p/1x1x1 UNIBUS/r system. (Each task requests the use of one resource.)

on the bus cannot begin transmission until a free resource is available. Therefore, state $N^{i}_{1,r-1}$ is changed to state $N^{i}_{0,r}$ 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.

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

where Pr(·) is the stationary probability for a state.

To solve for the stationary probability values, we can express all states in terms of an elementary state(s) and to solve for the elementary state(s) by using the relationship that all probability values sum to unity. Referring to figure 4, 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, we have the following matrix equation.

$$p\lambda \begin{bmatrix} N^{i-1}_{1,0} \\ N^{i-1}_{1,1} \\ N^{i-1}_{1,2} \\ \vdots \\ N^{i-1}_{1,r-1} \\ N^{i}_{0,r} \end{bmatrix} = \begin{bmatrix} p\lambda+\mu n & -\mu_s & 0 & 0 & & & \\ 0 & p\lambda+\mu_n+\mu_s & -2\mu_s & 0 & & & \\ 0 & 0 & p\lambda+\mu_n+2\mu_s & -3\mu_s & \cdots & & \\ \vdots & \vdots & \vdots & \vdots & \cdots & & \\ & & & & \cdots & & \\ & & & & & \vdots & \vdots \\ & & & & & p\lambda+\mu_n+(r-1)\mu_s & r\mu_s \\ & & & & & 0 & p\lambda+\mu_n+r\mu_s \end{bmatrix} \begin{bmatrix} N^{i}_{1,0} \\ N^{i}_{1,1} \\ N^{i}_{1,2} \\ \vdots \\ N^{i}_{1,r-1} \\ N^{i}_{0,r} \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & \cdots & & & \\ -\mu_n & 0 & 0 & \cdots & & & \\ 0 & -\mu_n & 0 & \cdots & & & \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ & & & \cdots & -\mu_n & 0 & 0 \\ & & & \cdots & 0 & -\mu_n & 0 \end{bmatrix} \begin{bmatrix} N^{i+1}_{1,0} \\ N^{i+1}_{1,1} \\ N^{i+1}_{1,2} \\ \vdots \\ N^{i+1}_{1,r-1} \\ N^{i+1}_{0,r} \end{bmatrix} \qquad (2)$$

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 RHS of eq. 2) is singular. Therefore, the states on stage i+1 cannot be expressed in terms of states on lower stages. However, from eq. 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 (eq. 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 against a procedure which solves for all the stationary probabilities simultaneously using (r + 1)(q + 1) balance equations and is found to be within four digits of accuracy in all cases.

Some performance results of the single shared bus are shown in Figures 5 and 6 for $\mu_s/\mu_n = 0.1$ and 1.0 respectively with 16 processors and 32 resources. 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 ($\rho_x = 16\lambda(\frac{1}{16\,\mu_n} + \frac{1}{32\,\mu_s})$). 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 Figure 5, we see that the delay is smaller as the number of partitions increases. A strange behavior is observed for the case of 16 partitions (16/16x1x1 SBUS/2). It has a worse delay than the case of 2 partitions (16/2x1x1 SBUS/16) for ρ_x below 0.64 and approaches the delay for the case of 8 partitions (16/8x1x1 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 Figure 5, 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 which saturates when $\rho_x = 6.0$.

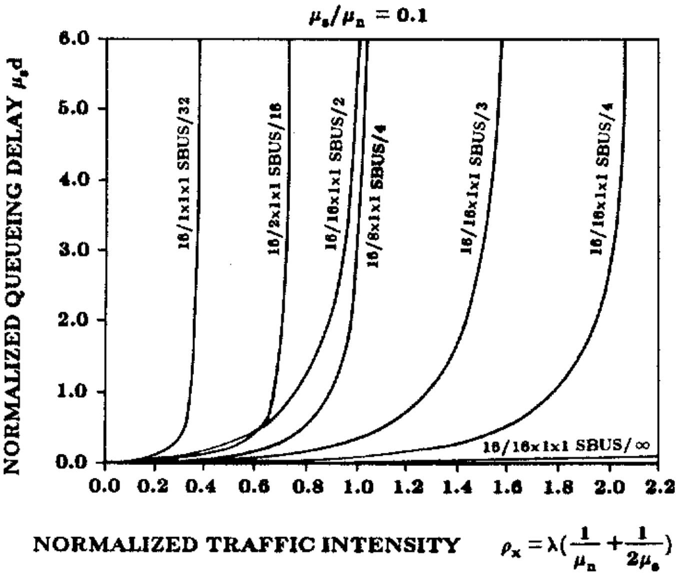


Figure 5. Normalized queueing delay of single shared bus for $\mu_s/\mu_n = 0.1$.

The strange behavior observed when $\mu_s/\mu_n = 0.1$ does not occur when $\mu_s/\mu_n = 1.0$ (Figure 6). In this case, the bus is always the bottleneck and as the number of partitions increases, the delay decreases. Further, the improvement of using infinite resources is very small due to the high data transmission time.

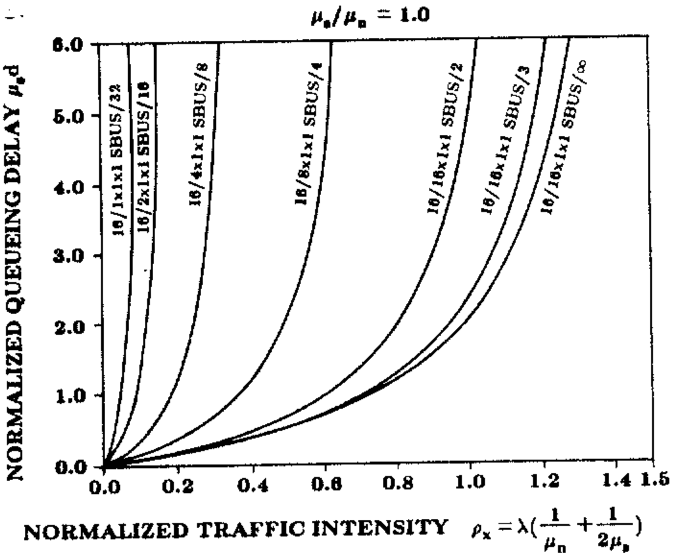


Figure 6. Normalized queueing delay of single shared bus for $\mu_s/\mu_n = 1.0$.

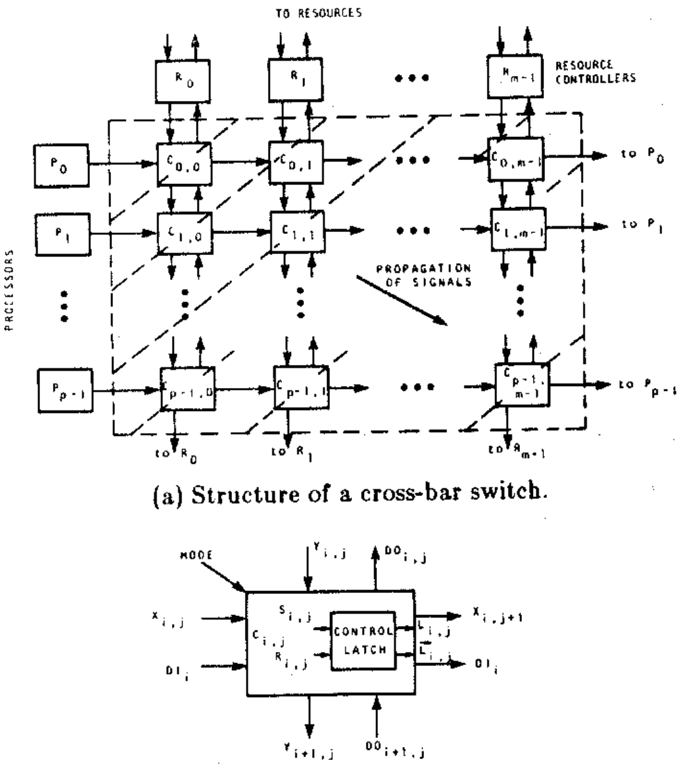
4. RSINs Using Multiple Shared Buses

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

In this section, the design of a cross-bar switch to support distributed resource scheduling is shown. The cell design for single resource requests is presented, and can be generalized to multi-resource requests. Figure 7 shows the overall structure of a cross-bar network. Processor i, $0 \leq i < p$, initiates a request by sending a request signal to the switch along the i-th row. Resource controller j, $0 \leq j < m$, indicates that bus j is free and at least one resource is free by sending a resource signal along the j-th 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 i-th row. Similarly, the resource signal is removed from any further cells along the j-th column. Each cell in the switch has enough intelligence to resolve the conflicts and to route the 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 Figure 7(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 j-th column. Processor k may have made a request previously. Since no resource signal was passed along the j-th column, it tried to search for another resource and found one. The new resource signal passed along the j-th column should be ignored in cell $C_{k,j}$ in order not to upset the state of a previous allocation.

We also assume that the system operates in two modes: request mode and reset mode. 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 which 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 investigated in the distributed Omega and cube networks.



(a) Structure of a cross-bar switch.

(b) Structure of a cell.

Figure 7. A cross-bar switch to support decentralized scheduling.

Referring to Figure 7(b), the inputs and outputs of cell $C_{i,j}$ which connects processor i and bus j have the following meaning:

$$X_{i,j} = \begin{cases} 0 & \text{processor i is not searching for a free resource} \\ 1 & \text{processor i is searching for a free resource} \end{cases}$$

(request mode)

$$X_{i,j} = \begin{cases} 0 & \text{processor i does not want to change the state of allocation} \\ 1 & \text{processor i wishes to relinquish the allocated resource} \end{cases}$$

(reset mode)

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

$$Y_{i,j} = \begin{cases} 0 & \text{bus j is busy or all the resources connected through bus j busy; new request cannot be accepted.} \\ 1 & \text{bus j is free and a free resource on bus j is available; a new request can be accepted.} \end{cases}$$

DI_i - data line to send data from the i-th processor;

$DO_{i,}$ - data line for resources on the j-th bus to receive data from the i-th processor;

$$L_{i,j} = \begin{cases} 0 & \text{Latch is off; any request made by processor i} \\ & \text{is passed to the next cell, C sub i,j+1} \\ 1 & \text{Latch is on; processor i is connected to bus j} \end{cases}$$

$S_{i,j}/R_{i,j}$ - the set/reset signal for the control latch in cell $C_{i,j}$;

MODE - controls the cell to be in request or reset mode.

The input/output relationship of the control signals is shown in the truth table in Table 1.

Table 1 Truth table and control signals for cell $C_{i,j}$ in a cross-bar switch.

Inputs		Outputs			
$X_{i,j}$	$Y_{i,j}$	$X_{i,j+1}$	$Y_{i+1,j}$	$S_{i,j}$	$R_{i,j}$
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

$$\begin{aligned} X_{i,j+1} &= X_{i,j}\,\bar{Y}_{i,j} \\ Y_{i+1,j} &= \bar{X}_{i,j}\,Y_{i,j}\,\bar{L}_{i,j} \\ S_{i,j} &= X_{i,j}\,Y_{i,j} \\ R_{i,j} &= 0 \\ DO_{i,j} &= L_{i,j}\,DI_i + DO_{i+1,j} \end{aligned}$$

(a) Request mode

Inputs		Outputs			
$X_{i,j}$	$Y_{i,j}$	$X_{i,j+1}$	$Y_{i+1,j}$	$S_{i,j}$	$R_{i,j}$
0	0	0	0	0	0
0	1	0	1	0	0
1	0	1	0	0	1
1	1	1	1	0	1

$$\begin{aligned} X_{i,j+1} &= X_{i,j} \\ Y_{i+1,j} &= Y_{i,j} \\ S_{i,j} &= 0 \\ R_{i,j} &= X_{i,j} \\ DO_{i,j} &= L_{i,j}\,DI_i + DO_{i+1,j} \end{aligned}$$

(b) Reset mode

In the request mode, the latch is set ($S_{i,j} = 1$) if processor i is making a request, bus j is free, and a resource connected on bus j is available. 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 ($X_{i,j+1} = X_{i,j}$). The resource signal to the next cell ($Y_{i+1,j}$) depends on the state of the control latch in the cell. If $Y_{i,j} = 0$, then $Y_{i+1,j} = 0$. If $Y_{i,j} = 1$ and $X_{i,j} = 1$, then the control latch is set 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 which 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. The logic equations for the controls and outputs are also shown in Table 1.

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 at the end of the request cycle, $X_{i,m} = 1$; 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 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 (Figure 7(a)) in a wave-like motion. The maximum time for signal propagation is, therefore, proportional to n+m. In the request cycle, the maximum gate delays in each cell is four. The maximum length of the request cycle is 4(n+m) gate delays. In the reset cycle, the maximum delay in each cell is one. The maximum length of the reset cycle is (n+m) gate delays.

A final remark about the design is that it is asymmetric. That is, it favors processors with lower index numbers. This means that processors which 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. 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 and separate request and reset signal lines are built into each cell so that requests and resets can be carried out simultaneously. This is the approach taken in the Heidelberg POLYP Polyprocessor [27]. The major disadvantage is that the extra signal lines pose a problem in VLSI implementation.

A Markovian analysis similar to that of the single bus is difficult due to the extensiveness of the 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 not present. As far as a processor is concerned, the cross-bar 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 3 can be applied directly when the load is light. The approximate delays are very close to the simulation results for $\mu_s d \leq 1$.

Under 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 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 3 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·r/p resources. As far as a processor is concerned, the multi-

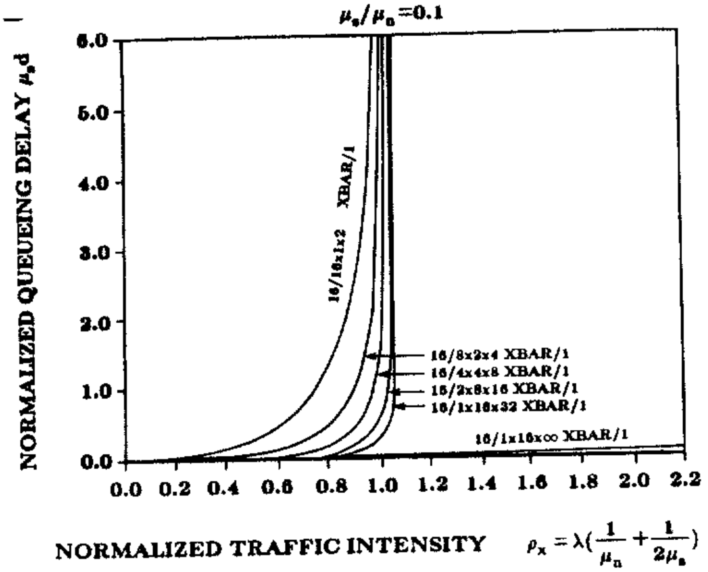


Figure 8. Normalized queueing delay of multiple shared buses for $\mu_s/\mu_n = 0.1$.

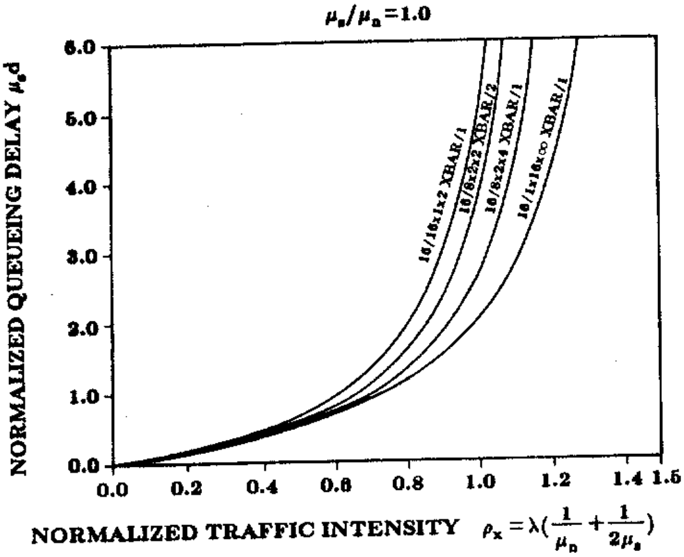


Figure 9. Normalized queueing delay of multiple shared buses for $\mu_s/\mu_n = 1.0$.

ple buses do not improve delay over a single bus. The analysis for delay is similar to that of Section 3 with a single bus connecting 1 processor to m·r/p resources. The heavy load approximation is found to be satisfactory when μ_sd is large. Simulations are used for cases in between.

Some performance results of the cross-bar switch are depicted in Figures 8 and 9. An observation from these figures is that a system with eight 2 × 4 cross-bar switches is nearly as good as a system with one 16 × ∞ cross-bar switch (with one resource attached to each output port). As can be expected, by partitioning the network into smaller cross-bar switches and by attaching more resources to the output ports we can achieve a smaller delay.

5. RSINs Using Omega and Cube Networks

The Omega [11] and generalized cube [18] networks belong to a class of 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 4-function interchange box which allows a straight, exchange, upper broadcast, or lower broadcast connection. For a network connecting N inputs to N outputs (N is a power of 2), there are $\log_2$N stages and $\frac{N}{2}*\log_2N$ interchange boxes. The delay in the networks is, therefore, O($\log_2$N). Figure 12 shows an example of an Omega network with N = 8. The O(N$\log_2$N) hardware complexity is much better than that of the cross-bar switch (O(N^2)).

The Omega network is equivalent to the cube network with the difference that it operates in the reverse direction. Furthermore, the Omega network can be rearranged into a cube network by renaming the inputs and outputs. This rearrangement is exemplified in the Omega network in Figure 12. If $B_{0,1}$ and $B_{1,1}$ are moved so that they are adjacent to $B_{0,3}$ and $B_{1,3}$, and with proper relabeling of processors and resources, the Omega network is transformed into a cube network. Using these networks as RSINs, they are, therefore, statistically equivalent. In the following discussion, we will only present results on the Omega network. The performance of the cube network is identical.

As seen in Figure 2, some of the feasible mappings from sources to destinations do not lead to maximal resource allocation. A centralized scheduler has 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 $\begin{bmatrix} x \\ y \end{bmatrix}$y! (for x ≥ y) or $\begin{bmatrix} y \\ x \end{bmatrix}$x! (for y > x) mappings in order to find the best one. Sub-optimal heuristics can be used [24], but will only be practical when x and y are small.

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.

The distributed algorithm is implemented by distributing the routing intelligence into the interconnection network so that there is no centralized control. Each exchange box can resolve conflicts and route requests to the appropriate destinations. If a request is blocked, it will be sent back to the originating exchange box in the previous stage. Request routing is, thus, dynamic and all the exchange boxes operate independently.

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

Q = number of resources requested;
L = number of allocated resources to be released;
S = number of resources reachable from this link;
J = number of resources rejected from the search;
C = number of free resources successfully found.

There are associated registers in each exchange box which store this information. These control signals are indicated in Figure 10. 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.

The control algorithm for each exchange box is written in pidgin Algol and is shown in Figure 11. The total

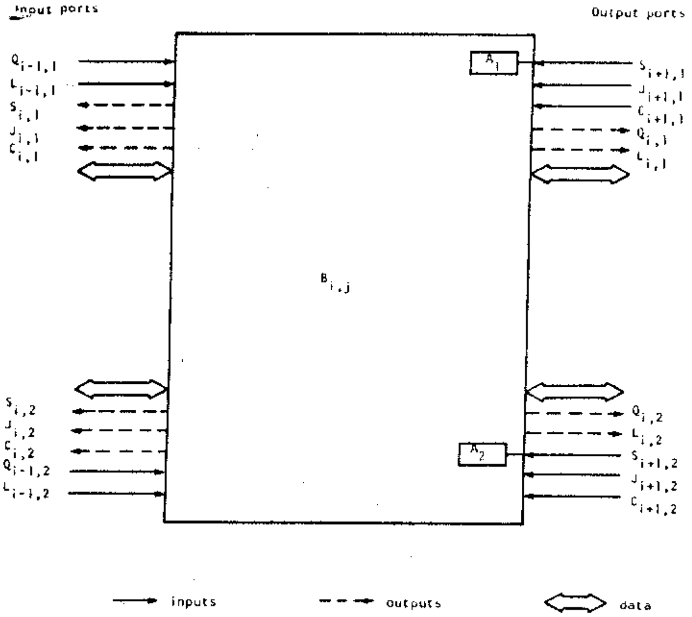


Figure 10. Control signals for a 2 by 2 exchange box.

number of reachable resources from the two input ports are calculated at the beginning and at the end of the loop. If any change is detected, this information is passed back to the previous stage. This allows status change to be propagated as early as possible. 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 have higher priority. Reject/query with the largest number of resources is always serviced first. Output ports ordered by the number of accessible resources are chosen successively. In case of ties, a random selection is made. After a query is sent to an output port, the corresponding availability register is zeroed because resources are no longer accessible from this port. In servicing completion signals, since a query may request multiple resources and they may be sent through multiple output ports, all the completion signals for a query must be assembled before they are sent back to the previous stage. The algorithm shown in Figure 11 is applicable to exchange boxes with a larger number of input and output ports (such as the Banyan and delta networks).

As an example, Figure 12 shows an 8x8 Omega network. Suppose resources R_0, R_1, R_4 and R_5 are available and status information are passed to the processors. The numbers on the output/input ports represent the status information received/sent. Assuming that P_0, P_3, P_4, and P_5 are requesting one resource each, the requests are sent simultaneously to the network after new status information arrives. In stage 0, no conflict is encountered. $B_{1,1}$ in stage 1 receives two requests. Since only one output terminal leads to free resources, the request originating from $B_{0,3}$ 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 exchange boxes on the average before it finds a free resource. For clarity, status changes due to new requests are not indicated in the figure.

One peculiar characteristic of the network is that status information changes always arrive at the processors simultaneously since the delay through all the boxes are identical. Requests queued at processors, therefore, enter the network simultaneously. This may cause undue conflict, especially to multi-resource requests. A solution

```
Process net (i, j);
    /* distributed scheduling algorithm in exchange
       box j on stage i of Omega and cube networks
       (refer to Figure 10) */

    while (true) do
        Begin
        wait (arrival of any control signal);

        /* calculate total number of resources
           reachable from the output ports */

        /* service status signal (S) change,
           Store $S_{i+1,1}$ and $S_{i+1,2}$ into the availability registers $A_1$ and $A_2$ */

        /* 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 collected at the input
           ports. The largest reject is always serviced first. Available output port(s) are
           scanned successively until one with the
           largest number of available resources is
           found. In case of ties, a random selection is made. Set the corresponding
           availability register to zero and send
           query. Continue searching until all the
           resources needed for this reject are
           found, otherwise send the unsatisfied
           rejects along the original input ports
           over which the queries are sent and
           decrease the resources queried. If all
           the resources requested by a query are
           rejected, the query is eliminated from
           the exchange box */

        /* service query (Q),
           Queries are serviced in a similar fashion
           as rejects. The largest query is always
           serviced first. */

        /* service completion (C),
           A completion signal received is held in
           an exchange box until all the necessary
           completion signals are collected. When
           all the resources queried are found, a
           completion signal is sent to stage i-1
           along the original input port over
           which the query is sent */

        /* send status signals back to the previous
           stage if any change is made. Calculate
           the total number of resources reachable
           from the output ports. If this is
           different from the total calculated previously,                      send
           $S_{i,1} = S_{i,2} = S_{i+1,1} + S_{i+1,2}$ along the
           status links to stage i-1. */

        end;

end process
```

Figure 11. Control algorithm for each exchange box in the Omega and Cube networks.

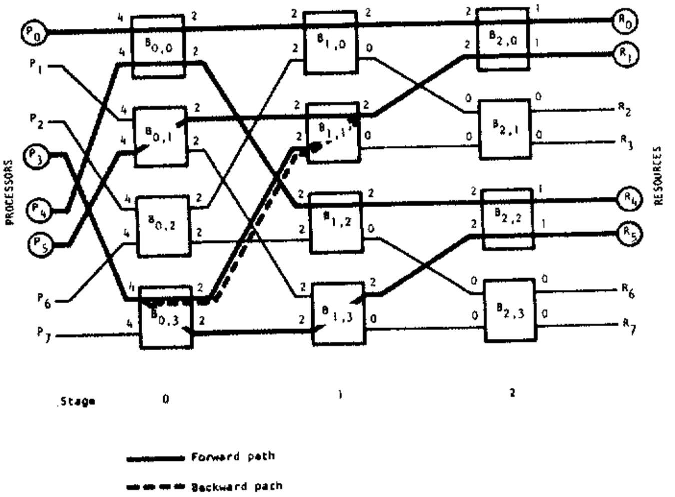


Figure 12. Example of Omega network with four requesting processors and four free resources, (25% of requests are blocked and backtracked; 100% resource allocation; average delay = 3.50 units).

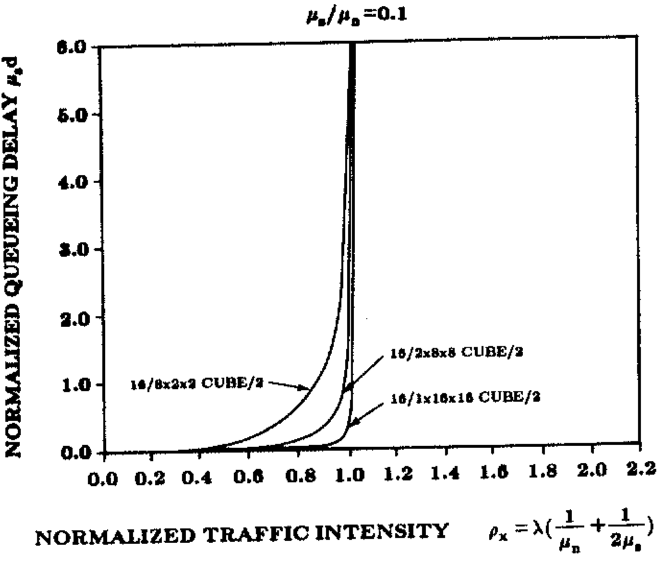


Figure 13. Normalized queueing delay of Omega or cube networks for μ_s/μ_n = 0.1.

is to use a similar strategy as Ethernet [26] which initiates requests with a random delay after the arrival of new status information.

The delay characteristics of the Omega and cube networks were evaluated by simulations and are plotted in Figures 13 and 14. There is basically very little difference in delays between eight 2 x 2 networks or one 16 x 16 network, although the cost of one 16 x 16 network is much higher. A conclusion similar to the crossbar switch can also be reached. That is, it is more cost-effective to partition the system into multiple smaller networks instead of using a single network.

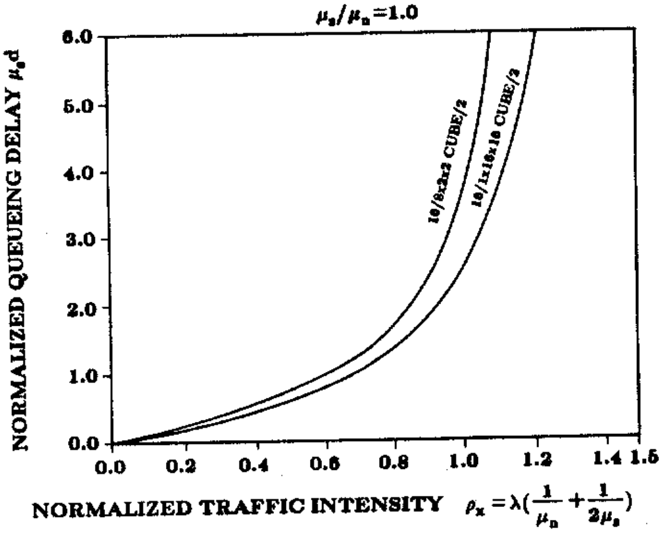


Figure 14. Normalized queueing delay of Omega or cube networks for μ_s/μ_n = 1.0.

6. Comparison of Different RSINs

In this section, we discuss the tradeoffs of different RSINs. The tradeoffs have to be made with respect to the relative cost of resources and networks.

If the cost of resources is small as compared the cost of a 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 3, 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 in Figures 5, 6, 8, 9, 13 and 14, the multiple private bus approach has the worst delays. The cube and Omega networks have slightly larger delays than the cross-bar switch when the load is high, but the difference is usually insignificant. The choice, therefore, depends on the cost of implementation. Cross-bar 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 [6]. When the network is built on multiple chips, Banyan type networks are still less expensive to implement.

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 CUBE/2 or a 16/4×4×4 XBAR/2 system.

In summary, the multiple private bus approach is attractive when the cost of resources is not high. When resources are expensive, the cube, Omega, or cross-bar networks are good candidates of RSINs. This conclusion is true for all values of μ_s/μ_n.

7. Conclusion

In this paper distributed scheduling algorithms for resource sharing are studied. Resource sharing differs from conventional accesses through addresses in that a request is directed towards any one 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 all 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.

Three resource sharing interconnection networks utilizing distributed scheduling are compared in this paper. The cross-bar switch results in the least delay time, but is the most expensive. The single bus has the highest blocking and is the least expensive. The private resource approach suffers from the unnecessary replication of resources and is not practical when the number of types of resources is large or when resources are expensive. Networks which have queueing delays between the private resource approach and the cross-bar switch are networks with logarithmic delays such as the Omega and cube networks. They represent versatile and cost-effective interconnection networks for resource sharing. The networks can be designed so that they operate in both resource sharing and address mapping modes.

Although we have studied cases with one class of identical resources, the approach can be extended easily to a general system which has multiple types of resources. The algorithms discussed have to be modified by identifying the type of resource requested by a processor and the type of resources reachable from an exchange box. This can be done by sending a binary request code (instead of 1 bit) indicating the type of resources requested in the distributed algorithms. In the distributed Omega and cube networks, multiple resource availability registers, one for each type of resources, have to be included in an exchange box. In the degenerate case in which there is one resource in each type, the network operates in the address mapping mode and the resource code in each request becomes its address. Resource accesses, therefore, are 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," *IEEE Computer*, Vol. 14, No. 12, pp. 31-41, Dec. 1981.

[2] K. E. Batcher, "STARAN Parallel Processing System Hardware," *Proc. of AFIPS 1974 National Computer Conf.*, Vol. 43, pp. 405-410, May 1974.

[3] K. E. Batcher, "The Flip Network in STARAN," *Proc. of 1976 Int'l Conf. on Parallel Processing,* Michigan, pp. 65-71, 1976.

[4] Burroughs Corp., *Final Report, Numerical Aerodynamic Simulation Facility Feasibility Study,* NASA Contractor Reports CR152284 and CR152285, Burroughs Corp., Paoli, PA, March 1979.

[5] T. Feng, "Data Manipulating Functions in Parallel Processors and Their Implications," *IEEE Trans. Computers,* Vol. C-23, No. 3, pp. 309-318, Mar. 1974.

[6] M. A. Franklin, "VLSI Performance Comparison of Banyan and Cross-bar Communication Networks," *Proc. of Workshop on Interconnection Networks,* pp. 20-28, Apr. 1980.

[7] L. R. Goke and G. J. Lipovski, "Banyan Networks for Partitioning Multi-processor Systems," *Proc. 1st Annual Comp. Architecture Conf.*, pp. 21-28, Dec. 1973.

[8] L. R. Goke, *Banyan Networks for Partitioning Multiprocessor Systems,* Ph.D. Thesis, Univ. of Florida, 1976.

[9] R. Jenevein, D. Degroot and G. J. Lipovski, "A Hardware Support Mechanism for Scheduling Resources in a Parallel Machine Environment," *Proc. of 8'th Annual Symposium on Computer Architecture,* pp. 57-66, May 1981.

[10] D. J. Kuck, "ILLIAC IV Software and Application Programming," *IEEE Trans. on Comp.*, Vol. C-17, pp. 746-757, Aug. 1968.

[11] D. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Trans. Computers,* Vol. C-24, No. 12, pp. 215-255, Dec. 1975.

[12] W. C. McDonald and J. M. Williams, "The Advanced Data Processing Test Bed," *Proc. of COMPSAC 78,* pp. 346-351, March 1978.

[13] S. M. Ornstein et al., "Pluribus-A Reliable Multiprocessor," *Proc. AFIPS 1975 National Computer Conference,* AFIPS Press, Montvale, N.J., pp. 551-559, 1975.

[14] J. H. Patel, "Performance of Processor-Memory Interconnections for Multiprocessors," *IEEE Trans. on Computers,* Vol. C-20, No. 10, pp. 771-780, Oct. 1981.

[15] M. C. Pease, "The Indirect Binary n-binary n-cube Microprocessor Array," *IEEE Trans. on Computers,* Vol. C-26, No. 5, pp. 458-473, May 1977.

[16] B. D. Rathi, A. R. Tripathi and G. J. Lipovski, "Hardwired Resource Allocators for Reconfigurable Architectures," *Proc. of 1980 International Conference on Parallel Processing,* pp. 109-117, Aug. 1980.

[17] M. C. Sejnowski et al., "Overview of the Texas Reconfigurable Array Computer," *AFIPS Conference Proceedings,* Vol. 49, pp. 631-642, 1980.

[18] H. J. Siegel and R. J. McMillen, "The Multistage Cube: A Versatile Interconnection Network," *IEEE Computer,* Vol. 14, No. 12, pp. 65-76, Dec. 1981.

[19] H. J. Siegel and R. J. McMillen, "Using the Augmented Data Manipulator Network in PASM," *IEEE Computer,* Vol. 14, No. 2, pp. 25-33, Feb. 1981.

[20] H. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Trans. on Computers,* Vol. C-20, No. 2, pp. 153-161, Feb. 1971.

[21] C. Wu and T. Y. Feng, "On a Class of Multistage Interconnection Networks," *IEEE Trans. on Computers,* Vol. C-29, No. 8, pp. 694-702, Aug. 1980.

[22] W. A. Wulf and C. G. Bell, "C.mmp--A Multi-mini Processor," *Proc. AFIPS 1972 Fall Joint Comp. Conf.*, Vol. 41, AFIPS Press, Montvale, NJ, pp. 765-777, 1972.

[23] Hicks, A., *Resource Scheduling on Interconnection Networks.* M.S. Thesis, Purdue University, Aug. 1982.

[24] Wah, B. W. and A. Hicks, "Distributed Scheduling of Resources on Interconnection Networks," *Proc. National Computer Conference,* AFIPS Press, pp. 697-709, 1982.

[25] Marsan, M. A., and M. Gerla, "Markov Models for Multiple Bus Multiprocessor Systems," *IEEE Trans. on Computers,* Vol. C-31, No. 3, pp. 239-248, March 1982.

[26] Metcalfe, R. M. and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Net-

works," *Comm. ACM,* Vol. 19, pp. 395-404, July 1976.

[27] Männer, R. and B. Deluigi, "The Heidelberg POLYP - A Flexible and Fault-Tolerant Polyprocessor," *Computer Physics Communications,* North Holland Publishing Co., Vol. 22, pp. 279-284, 1981.

[28] Marsan, M. A., "Bounds on Bus and Memory Interference in a Class of Multiple Bus Multiprocessor Systems," *Proc. of 3rd Int'l Conf. on Distributed Computing Systems,* Oct. 1982.

[29] Willis, P. J., "Derivation and Comparison of Multiprocessor Contention Measures," *IEE J. of Computers and Digital Techniques,* Aug. 1978.

[30] Marsan, M. A. and F. Gregoretti, "Memory Interference Models for a Multimicroprocessor System with a Shared Bus and a Single External Common Memory," *Microprocessing and Microprogramming - EUROMICO Journal,* Vol. 7, no. 2, Feb. 1981.

[31] Fung, F., and H. Torng, "On the Analysis of Memory Conflicts and Bus Contentions in a Multiple-Microprocessor System," *IEEE Trans. on Computers,* Jan. 1979.