Resource Sharing on Multiprocessors*

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

Abstract

In this paper, we have studied distributed scheduling of resources for multiprocessors. The resource mapping problem is a generalization of the conventional address mapping problem because a request is directed to any element (one or more) of a group of identical resources which can service the same type of tasks. The size of each group is one in the conventional address mapping problem. Interconnection networks for address mapping must be generalized to allow resource mapping. Four different networks have been investigated; namely, cross-bar switch, single shared bus, multiple buses with private resources and networks with logarithmic delays such as the cube and Omega networks. It is found that the last class of networks is the most versatile and cost-effective candidate for distributed resource scheduling.

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 [10], 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.

Another solution which avoids the sequential service 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.

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: $p/ix jxk\ N/r$ 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

* This research was supported by National Science Foundation Grants ECS 80-10580 and ECS 81-05968.

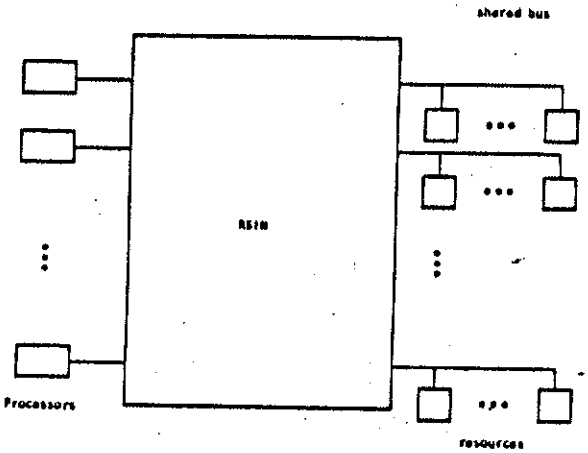

Figure 1. RSIN as used in a multiprocessor environment.

each RSIN. As an example, a 16 processor system with 2 private resources each and connected via private buses can be described as 16/16×1×1 UNIBUS/2. If a 16×16 cube network is used, we have 16/1×16×16 CUBE/2.

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). Assume processors 0, 1, 2 are making requests 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 2 resources can be allocated without blocking. This gives a resource utilization of 67%. 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 RSIN and the corresponding routing algorithm used is measured by d, the expected delay in the queue before free resources are allocated. In this paper, we compare four network configurations, namely, single bus connecting shared resources, multiple buses connecting private resources, Omega and cube networks, and cross-bar switch. Only distributed scheduling algorithms will be discussed.

3. RSINs Using Shared Bus(es)

A shared bus is used to communicate status information of resources to processors and to transmit tasks from processors to resources. Every time free resources are allocated or busy resources complete their tasks, the number of free resources available is broadcast to all the connected processors via the network. This new status infor-

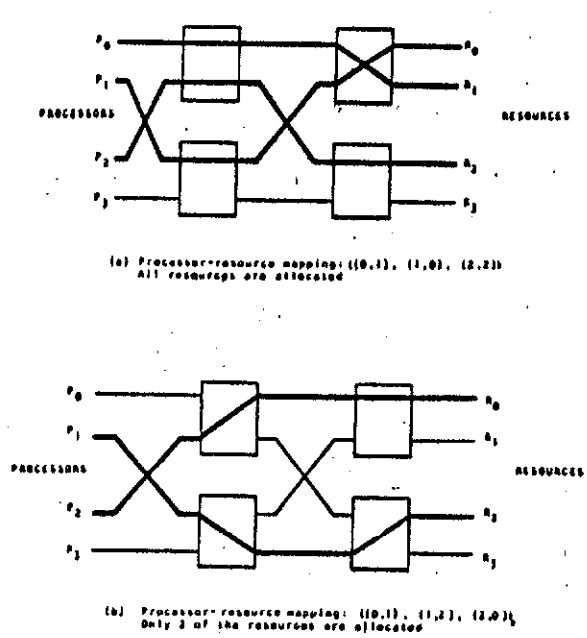

Figure 2. A RSIN using 4 by 4 Omega network.

mation 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.

The single shared bus configuration exists in a single bus system to which all the resources are connected, and in a multiple bus system in which every processor is connected via a private bus to a pool of private resources. 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. A shared bus in which the bottleneck is in the bus can be modelled very simply as an M/M/1 queueing system. On the other hand, if the bottleneck is in the resources, then the shared bus can be approximated by an M/M/n queueing system. For cases in between, that is, when neither is the bottleneck, 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. The state transition is Markovian and the state diagram is two-dimensional. To develop a closed form equation for the queueing time is difficult if not impossible. We resort to simulations in this case. The performance of the shared and private resource approaches will be shown in Section 6.

4. RSIN Using Cross-bar Switch

In contrast to the shared bus, the cross-bar switch is non-blocking and will give the highest resource utilization and the least delay. Cross-bar communication networks have been shown to compare favorably to Banyan networks for VLSI implementation provided that the whole network is implemented on one chip [6]. Further, the cross-bar switch is useful in providing a lower bound on the queueing delay.

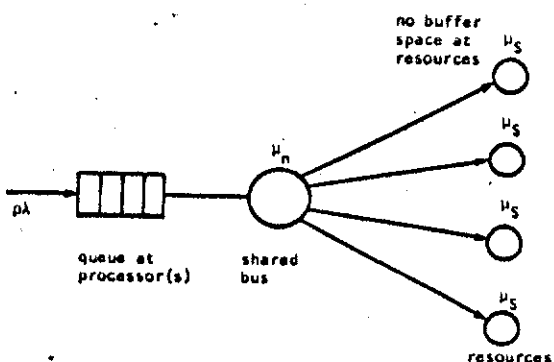

Figure 3. A queueing model of the shared-bus.

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 4 shows the overall structure of a cross-bar network. Processor i, $0 \leq i < n$, initiates a request by sending a request signal to the switch along the i-th row. Resource j, $0 \leq j < m$, indicates that it 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 4(a), if the request signal to cell $C_{i,j}$ is removed, then the latch in $C_{i,j}$ is reset and the resource becomes free. The resource signal will again propagate down the j-th column. Processor k may have made a request previously. Since resource j was busy, 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 the 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 con-

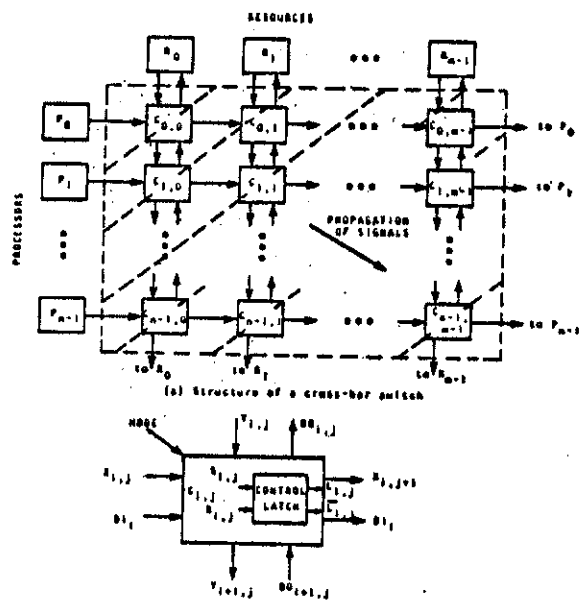

Figure 4. A cross-bar switch to support decentralized scheduling.

currency 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.

Referring to Figure 4(b), the inputs and outputs of cell $C_{i,j}$ which connects processor i and resource 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;

$Y_{i,j} = \begin{cases} 0 & \text{resource j is busy and cannot accept any request} \\ 1 & \text{resource j is free and can accept a new request} \end{cases}$

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

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

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

$S_{i,j}/R_{i,j}$ - 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	$L_{i,j}$	0	0
1	0	1	0	0	0
1	1	0	0	1	0

$X_{i,j+1} = X_{i,j} \overline{Y}_{i,j}$
$Y_{i+1,j} = X_{i,j} Y_{i,j} \overline{L}_{i,j}$
$S_{i,j} = X_{i,j} Y_{i,j}$
$R_{i,j} = 0$
$DO_{i,j} = L_{i,j} DI_i + DO_{i+1,j}$

(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

$X_{i,j+1} = X_{i,j}$
$Y_{i+1,j} = Y_{i,j}$
$S_{i,j} = 0$
$R_{i,j} = X_{i,j}$
$DO_{i,j} = L_{i,j} DI_i + DO_{i+1,j}$

(b) Reset mode

In the request mode, the latch is set ($S_{i,j} = 1$) if processor i is making a request and resource j is available. If resource j is not available ($Y_{i,j} = 0$), 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 ($L_{i,j}$) 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 $Y_{n,j}$ 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 it is free by setting $Y_{0,j} = 1$. If at the end of the request cycle, $Y_{n,j} = 1$, 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 it 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 4(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. Since requests arrive stochastically and if the number of requests arriving within a single request-reset interval is small, the undesirable effect of priority scheduling is negligible. There is not very much gain by using a system with tokens in which a request is a pulse of short duration.

An M/G/c queueing model of the cross-bar switch is shown in Figure 5. Solution techniques for this are scarce. The best known bounds for a GI/G/c queueing systems are due to Kingman [27] and Brummelle [26]. However, the lower and upper bounds differ so extensively that it is difficult to compare the performance with other systems. A reasonable approximation has also been derived by Allen and Cunneen [28]. It shows that for any GI/G/c queueing system, it is approximately true that the queueing delay is

$$d = \frac{C(c,c\rho)}{\mu c(1-\rho)} \left\{ \frac{C_i^2 + C_s^2}{2} \right\}$$

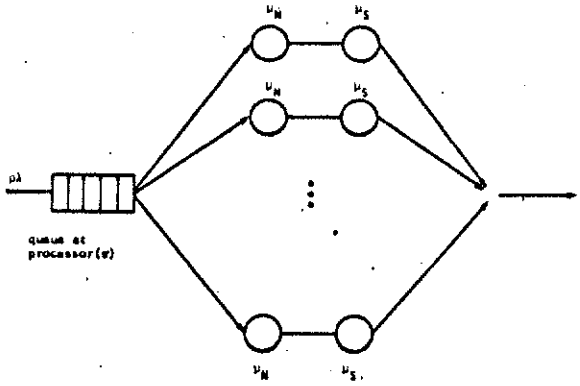

Figure 5. M/G/c-a queueing model of the cross-bar switch.

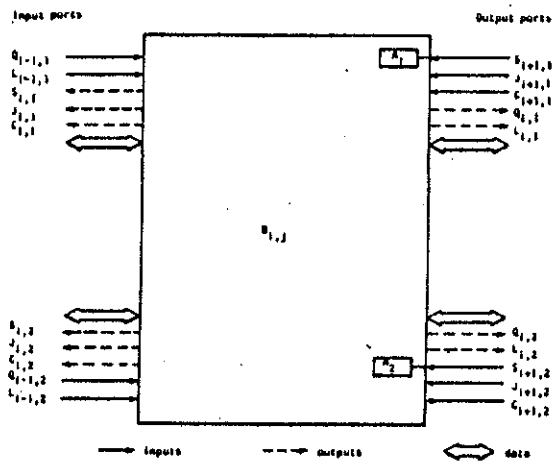

Figure 6. Control Signals for a 2 by 2 exchange box.

where
C_t^2, C_s^2 are the squared coefficient of variation for the interarrival time and service time;
$\frac{1}{\mu} = (\frac{1}{\mu_a} + \frac{1}{\mu_s})$ is the expected service time;
$\rho = \lambda/(\mu c)$ is the traffic intensity;

$$C(c,c\rho) = \frac{\frac{(c\rho)^c}{c!}}{\frac{(c\rho)^c}{c!} + (1-\rho)\sum_{k=0}^{c-1}\frac{(c\rho)^k}{k!}}$$

is the Erlang's C formula.

This approximation is found to perform very well when c is large and matches well with simulations. This is shown in Section 6.

5. RSIN 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.

```
Process net (i, j);
    /* distributed scheduling algorithm in exchange
       box j on stage i of Omega and cube networks
       */
    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 availa-
           bility 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 collected at the input
           ports. The largest reject is always ser-
           viced 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 selec-
           tion 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 previ-
           ously, 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 7. Control algorithm for each exchange box in the Omega and Cube networks.

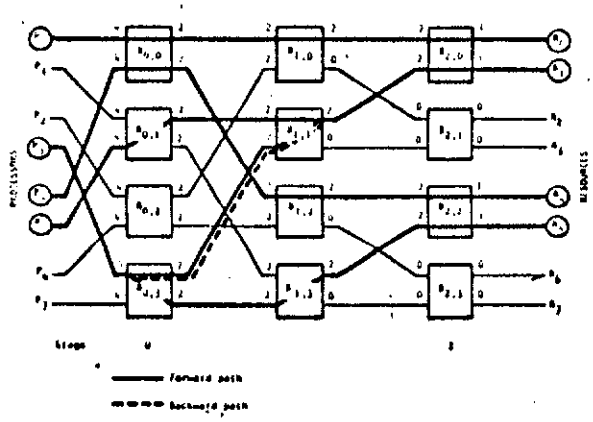

Figure 8. Example of Omega network with four requesting processors and four free resources, (25% of requests are blocked and backtracked; 100% resource allocation; average delay = 3.50 units.

They are chosen for their simplicity and versatility. The basic element in these networks is a 2-input 2-output 4-function interchange box which allows a straight, diagonal, 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 $\frac{N}{2} \cdot \log_2 N$ interchange boxes. The delay in the networks is, therefore, $0(\log_2 N)$. Figure 8 shows an example of an Omega network with $N = 8$.

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 8. 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 $\binom{x}{y} y!$ (for $x \geq y$) or $\binom{y}{x} x!$ (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 ($0(\log_2 N)$) 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 6. 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 7. The total 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 7 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 8 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 is to use a similar strategy as Ethernet [29] which initiates requests with a random delay after the arrival of new status information.

The Omega and cube networks have less blocking than the shared bus, but have higher blocking than the cross-bar switch. A queueing model is currently under investigation.

6. Performance Comparison of Different RSINs

In this section, we compare the performance of the different RSINs. Performance is measured in terms of queueing delay at the processors. We expect that the cross-bar switch to have the least delay and the shared bus to have the highest delay.

We assume that there are 16 processors and 32 identical resources. Four RSINs are compared, namely, single shared bus (16/1x1x1 UNIBUS/32), single bus for each processor with two private resources (16/1x1x1 UNIBUS/2), 16x16 Omega or cube network (16/1x16x16 CUBE/2) and 16x32 cross-bar switch (16/1x16x32 XBAR/1).

As mentioned before, the ratio of task transmission time to task service time is important in determining the performance of networks other than the cross-bar switch. We have investigated the cases of $\mu_s/\mu_n = 0.1$ and 1.0 respectively and they are plotted in Figures 9 and 10. The delay times are normalized with respect to the average task service times. The x axis shows $\rho_x = \frac{p\lambda}{r/(\frac{1}{\mu_n} + \frac{1}{\mu_s})}$, the traffic intensity of the 16x32 cross-bar switch. In Figure 9, it is seen that the performance of the cross-bar switch is the best and is very close to that of the cube network. Allen-Cunneen's GI/G/c approximation of the cross-bar switch is also very close to the simulated performance. The single bus system is acceptable for ρ_x below 0.34. The private resource approach is feasible for small ρ_x. As the workload increases, the delay becomes intolerable. In Figure 10, which shows the case when the average task transmission time is equal to the average task service time, the cross-bar switch is found to be unaffected because the network is non-blocking. The cube network and private bus

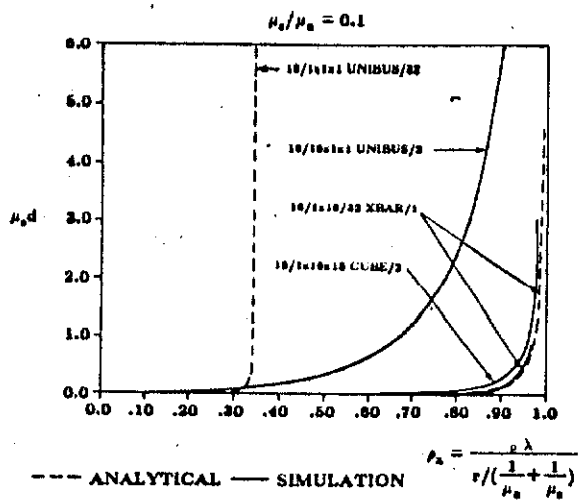

Figure 9. Normalized queueing delay of RSINs with respect to ρ_x, traffic intensity of cross-bar switch for $\mu_s/\mu_n = 0.1$.

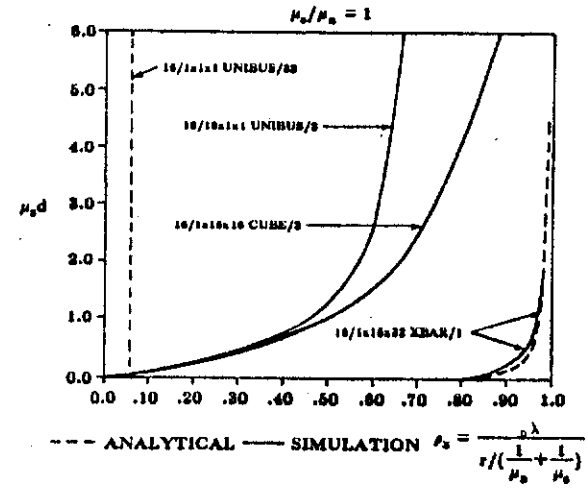

Figure 10. Normalized queueing delay of RSINs with respect to ρ_x, traffic intensity of cross-bar switch for $\mu_s/\mu_n = 1.0$.

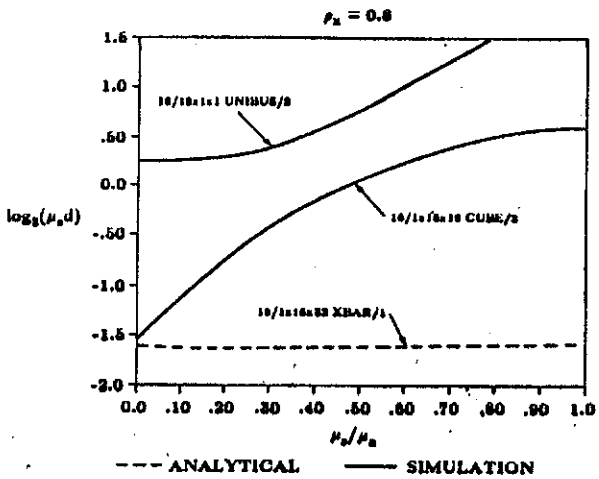

Figure 11. Normalized queueing delay for different ratios of μ_s/μ_n under heavy load ($\rho_x = 0.8$).

behave identically at low load. As the load increases, the private bus approach is worse. The single bus approach has high blocking at very low load.

In Figure 11, we have compared the different RSINs with varying ratios of task transmission time and task service time under heavy load. It is seen that the queueing delay in the cube network levels off as the ratio increases while the delay for the private bus tends to diverge to infinity (not indicated in the figure).

In summary, the cross-bar switch results in the minimum queueing delay, but is the most expensive approach. The single bus approach is only acceptable when the task transmission time is very small as compared to the task service time. The private bus approach compares favorably with the cube network under light load, but becomes unacceptable under heavy load. We

conclude that networks with logarithmic delays is the most versatile and cost-effective candidate as a RSIN.

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.

Four 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.

Although we have studied cases with one class of identical resources, the approach can be extended easily to a general system where there are 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) in the distributed algorithms. In the distributed Omega and cube networks, multiple resource availability registers have to be used in each exchange box. In the degenerate case where there is one resource of each type, the network operates in the address mapping mode and the resource type 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 Multiprocessor 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] Brummelle, S. L., "Some Inequalities for Parallel Server Queues," *Operations Research*, Vol. 19, pp. 402-413, 1971.

[27] Kingman, J. F. C., "Inequalities in the Theory of Queues," *Journal of the Royal Statistical Society, Series B*, Vol. 32, 102-110, 1970.

[28] Allen, A. O., "Queueing Models for Computer Systems," *IEEE Computer*, pp. 13-24, April 1980.

[29] Metcalfe, R. M. and D. R. Boggs, "Ethernet: Distributed Packet Switching for Local Computer Networks," *Comm. ACM*, Vol. 19, pp. 395-404, July 1976.