PROCEEDINGS 1982 NATIONAL COMPUTER CONFERENCE, AFIPS PRESS.

Distributed scheduling of resources on interconnection networks

by BENJAMIN W. WAH and ANTHONY HICKS
Purdue University
West Lafayette, Indiana

ABSTRACT

In this paper, we have studied the distributed scheduling of resources on interconnection networks. The resource scheduling problem is different from the conventional address mapping problem on interconnection networks because a request is not directed towards a particular destination address but to any one of a pool of destination addresses for free resources. To design an algorithm with the minimum transfer of control signals, priority is associated with the scheduling of multiple requests. This is illustrated by the distributed cross-bar switch which has one signal line in each direction of a switch node. For complete asynchronous operation, more signal lines are needed. This is illustrated by the distributed Omega and binary n-cube networks. Each exchange box in the network operates independently to resolve conflicts. The performance of the distributed scheduling algorithm for the Omega and cube networks is compared against the optimal centralized scheduling algorithm which has about 1% average blocking probability. The performance degradation is less than 20% in all cases. The theory of the design can be applied to other interconnection networks.

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

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 which interconnects processors and resources together. The function of the interconnection network is to route requests initiated from one point to another point connected on the network.[5,8,11,14,15,17,21] The notable characteristics in these networks is that routing is done by addresses. That is, a request is initiated with a specific destination or a set of destinations and the requests are supposed to be routed to the correct destinations. 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] Numerical Aerodynamic Simulation Facility (NASF)[1,4] and the Ballistic Missile Defense testbed.[12]

In general, an interconnection network routes requests from a set of source points to a set of destination points (they may coincide with each other). In a *resource sharing interconnection network (RSIN)*, the destination points are identical (or sets of identical) resources to which requests or tasks can be delegated to. Examples of these resources are special purpose VLSI chips. In this respect, jobs initiated at source processors can be sent to any one of the free resources of a given type at the destination. This is the important point that differentiates RSIN from interconnection networks using address mapping. Another application is to map the set of destination points directly into the source points. In this case, we have a load balancing network which can re-balance the load in the system dynamically. RSIN can also be applied in data-flow computers to distribute tasks to processing units.

Since the system operates continuously, requests from source processors can be initiated at random times. At any time, a set of processors may be making requests and a set of resources are free. It is the function of a scheduler to set the RSIN in order to connect the maximum number of resources to the processors, that is, to have the maximum resource utilization.

As an example, consider a 4 by 4 Omega network (see Figure 1). 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 66%. A similar example can be generated for the indirect binary n-cube network. This illustrates that the scheduler must be designed properly to give the maximum resource utilization.

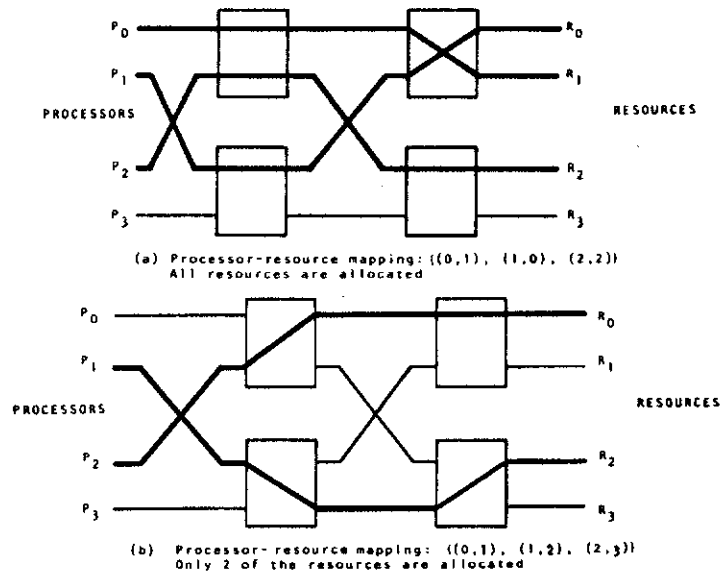


Figure 1—A RSIN using 4 by 4 Omega network

The earliest study of RSIN has been done with respect to centralized computer systems. A uni-bus is used in a time-shared fashion for connecting peripheral I/O devices to the CPU. Multiple time-shared busses have been used in the PLURIBUS minicomputer multiprocessor.[13] A cross-bar switch has been used in C.mmp[22] although the network is mostly used in address mapping mode. This network permits full interconnection capability between any source and destination ports. As long as each source port addresses a unique destination port, there is no blocking in the network and all messages can be routed through the network simultaneously. The single or multiple busses is a source of bottleneck, and is the least expensive design. The cross-bar switch is the most expensive network but has the least degree of blocking. A compromise is to use a less expensive network than the cross-bar switch that has a lower blocking probability than the single bus systems. 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 re-

*Processor 3 could have made a request earlier and have sent a job to resource 3 in a block transfer mode. The network will, therefore, be free at the present time.

sources 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. The tree network has a delay of $0(\log_2 n)$ in selecting a free resource (n is the total number of resources), and most notably, the scheduling of multiple requests is done sequentially.

A few comments can be made about the previous studies. First, the scheduling algorithms are centralized. For mapping n requesting processors to n resources, the scheduling algorithm has a worst case complexity of $0(n*\log_2 n)$. This complexity depends on the number of requesting processors, which is practical when n is small or when requests are not very frequent. Second, for scheduling requests on interconnection networks with logarithmic delays such as binary n-cube, Banyan and Omega, no optical scheduling algorithm has been established.

The objective of this paper is (a) to study the performance of distributed scheduling algorithms with a scheduling time that is independent of the number of requests made in the network but only dependent on the delay in the network and compare the performance against centralized scheduling algorithms; (b) to design interconnection networks to support distributed scheduling.

The basic assumption made in this study is that each processor makes a request for one resource, although there may be multiple requests outstanding. The extension to the case when multiple resources are requested simultaneously is discussed in Section 5. Two types of request characteristics are identified. First, the resource requested by the processor must be continuously connected to the processor for the duration of the request. In this case, the RSIN may not be completely free when a set of new requests are initiated. Networks with logarithmic delays such as the Omega and binary n-cube may have a high blocking probability. A distributed cross-bar RSIN with no blocking is preferable in this case. This is discussed in Section 2. Second, requests are made in a block transfer mode. That is, a free resource is connected to a requesting processor for a short duration of time. After the request is sent to the resource, the connection between the processor and the resource is broken. The resource will continue to service the task, and the processor is free to generate new requests in the future. In this case, the network is almost or completely free when a set of new requests is initiated. We present in Sections 3 and 4 the centralized and distributed scheduling of request on Omega and binary n-cube networks with the assumption that the network is completely free when a set of requests is initiated. The performance of partially busy networks is presented elsewhere.[23]

2. DISTRIBUTED SCHEDULING ALGORITHM FOR THE CROSS-BAR SWITCH

We present in this section the design of a cross-bar switch to support distributed scheduling algorithms. The motivations behind studying cross-bar switches are that it is non-blocking and it is very suitable for VLSI implementation. It has been shown that cross-bar communication networks are favorable as compared to Banyan networks for VLSI implementation provided that the whole network is implemented on one chip.[6]

Figure 2 shows the overall structure of a cross-bar network. Processor $i, 0 \le i < n$, initiates a request by sending a request signal to the switch along the i-th row. Resource $j, 0 \le j < m$, 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 the state of the switch. 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 2, 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 a previous allocation. To solve this problem, we 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 binary n-cube networks.

Referring to Figure 2(b), the inputs and outputs of cell $C_{i,j}$ which connects processor i and resource j have the following meaning:

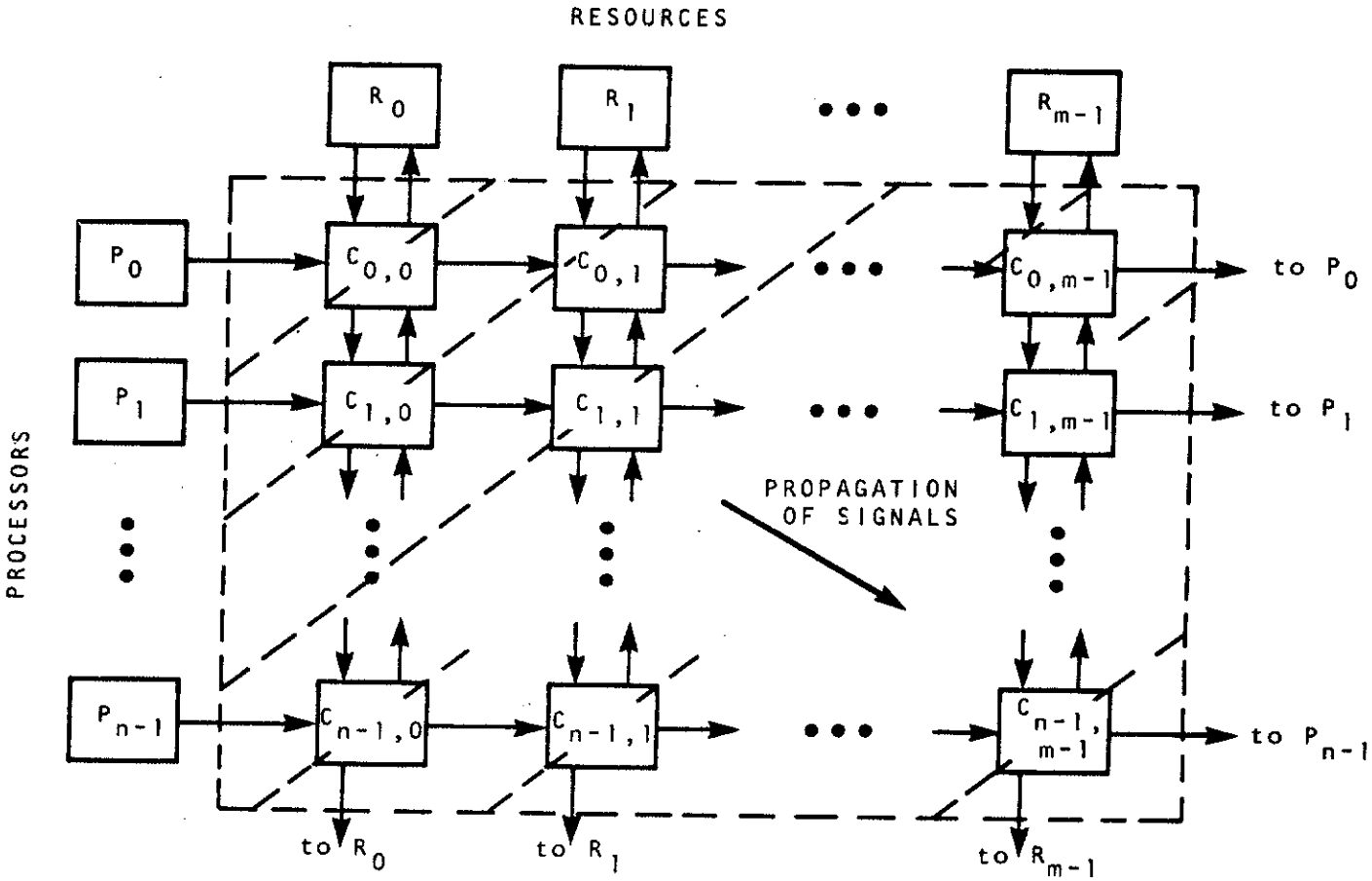
$X_{i,j} = \begin{cases} 0 & \text{processor } i \text{ is not searching for a free resource} \\ 1 & \text{processor } i \text{ is searching for a free resource} \end{cases}$
(request mode)

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

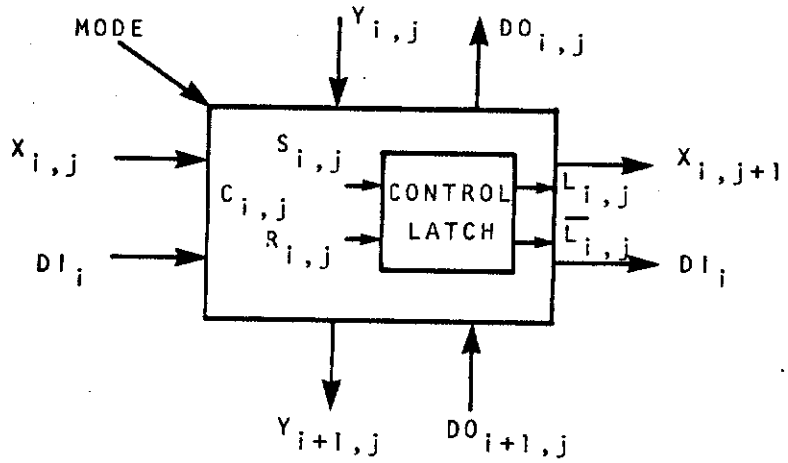
$X_{i,j}$ always returns to 0 at the end of each mode

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

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



(a) Structure of a cross-bar switch



(b) Structure of a cell

Figure 2—A cross-bar switch to support decentralized scheduling

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

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

$S_{i,j}/R_{i,j}$—the set/reset signal to 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 I.

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 ($X_{i+1,j}$) depends on the state of 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 ($\overline{L}_{i,j}$) 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 switch are reset. The logic equations for the controls and outputs are also shown in Table I. The design of cell $C_{i,j}$ is shown in Figure 3.

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, resource 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 the resource is not allocated and R_j should send out the $Y_{0,j} = 1$ signal continuously. Otherwise, it will set $Y_{p,j} = 0$ to indicate that it is allocated.

Requests and resets are accepted at the beginning of the corresponding cycle. 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 2) on 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 4 because of two gate delays in the control latch. The maximum length of the request cycle is $4(n + m)$ gate delays. In the reset cycle, the maximum delay in each cell is 1 due to the mode control gate. The maximum length of the reset cycle is $(n + m)$ gate delays.

A final remark about the scheduling algorithm is that it is asymmetric. That is, it favors processors with lower index numbers. In order to design an algorithm that is symmetric and to allow requests and resets to be initiated dynamically, more control lines are needed. Resources that are available

TABLE I—Truth table and control signals for cell $C_{i,j}$

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	$\overline{L}_{i,j}$	0	0
1	0	1	0	0	0
1	1	0	0	1	0

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

(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

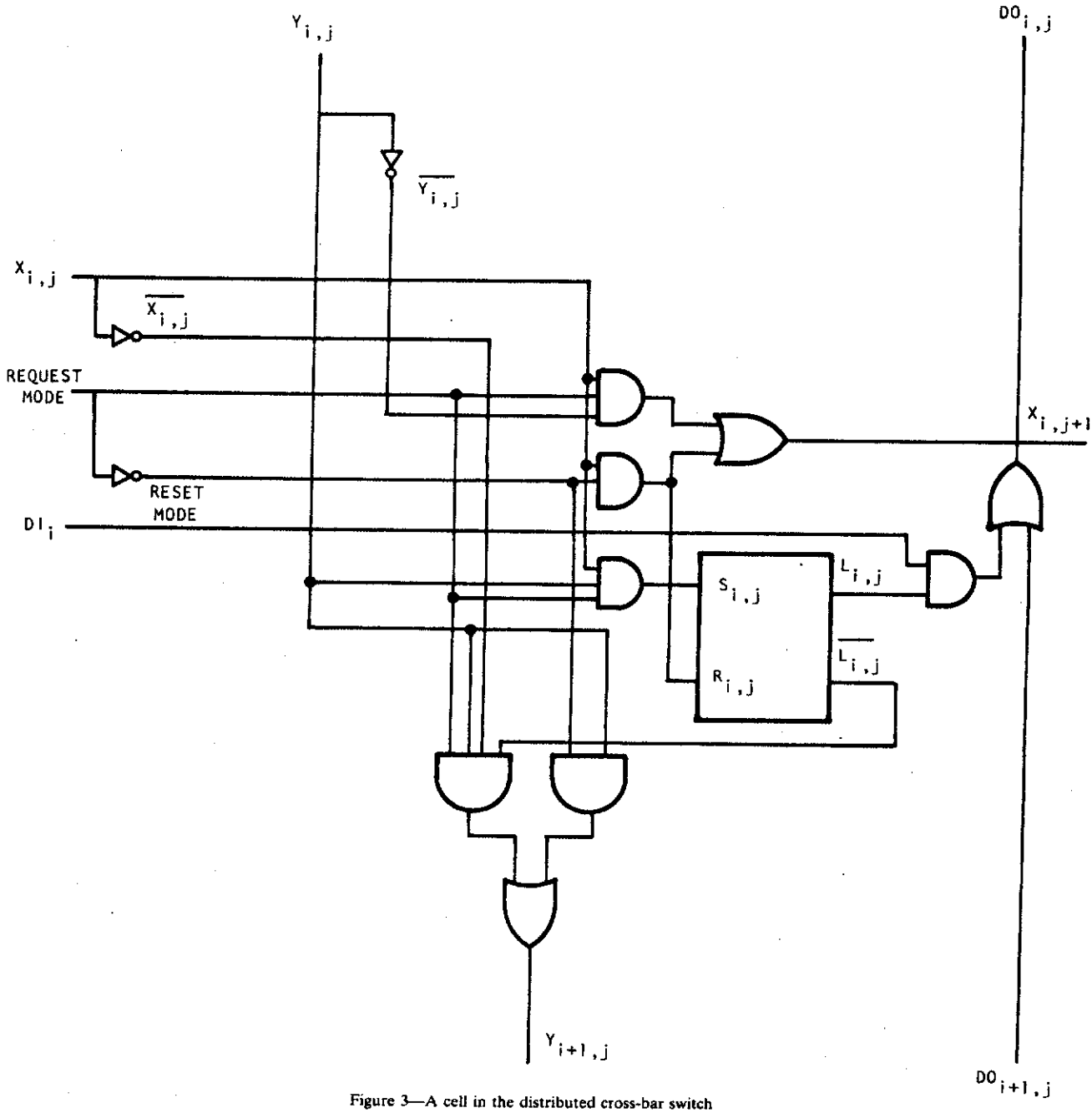


Figure 3—A cell in the distributed cross-bar switch

can send a pulse of a short duration along a column. Only processors that receive a pulse will be assigned the resource. In this sense, the pulse behaves like a token. When different resources issue tokens randomly, the algorithm is symmetric.

3. CENTRALIZED RESOURCE SCHEDULING ON NETWORKS WITH LOGARITHMIC DELAYS

In the remainder of this paper, we present the resource scheduling algorithms for interconnection networks with logarithmic delays. Specifically, we study the Omega and binary n-cube networks and first establish the optimal scheduling algorithm of these networks. Based on the optimal behavior, we compare the performance degradation of other heuristics and the distributed algorithm.

The Omega[11] and binary n-cube[15] networks are chosen for their simplicity and versatility. The basic element in these networks is a 2 input, 2 output interchange box which allows a straight or a diagonal connection. For a network connecting N inputs to N outputs (N is a power of 2), there are $\log_2 N$ stages and $\frac{N}{2} * \log_2 N$ interchange boxes. The delay in the net-

work is therefore O ($\log_2 N$). Figure 8 shows an example of an Omega network with $N = 8$.

The Omega network is equivalent to the binary n-cube network with the difference that it operates in the reverse direction. Using these networks as RSINs, they are statistically identical. The performance of these networks is evaluated by selecting a random subset of processors and resources and finding the maximum number of resource allocations. If the Omega network can be rearranged into a binary n-cube network, then their performance as RSINs are identical. 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. In the following discussion, we will only present the result on the Omega network. The performance of the binary n-cube network is equivalent.

(a) Optimal Scheduling Algorithm

Simulation results presented in Franklin[6] show that with $N = 8$, there is a message blocking probability of about 30% using address mapping. We show in this section that there is virtually no blocking when the Omega and binary n-cube networks operate as RSINs.

The results are obtained by exhaustive enumeration over all the possible combinations of connections for a subset of requesting processors and free resources. Because of the large number of combinations, only networks with $N = 8$ can be studied. Even in this case, the total number of possible combinations is slightly under 600 million. The large number of combinations is attributed to the fact that the order of connections is important. For a set of i requesting processors and j free resources, there are $i! * j!$ possible ordered connections.

A faster method is developed by observing that each box can be set in 2 states. With 12 interchange boxes, there are $2^{12} = 4096$ states or possible connections. These 4096 possible connections are arranged into multiple trees so that the maximum number of connections can be found efficiently. Using this method, the enumerations were completed using 10 hours of CPU time and 64 K bytes of memory on a VAX 11/780.

A selected set of the simulation results are plotted in Figures 4 and 5 for the blocking probability and standard deviation of processor allocations when the number of requesting processors equals the number of free resources. These results are based on the assumption that the network is completely free before the allocations. The average processor blocking probability is defined as:

$$\text{processor blocking probability} = \frac{\text{Number of allocated processors}}{\text{Number of requesting processors}}$$

The blocking probability must be interpreted correctly. For example, if there are 4 processors and 2 resources, then at most 2 processors can be allocated resources and the minimum blocking probability is 50%. If there are 2 processors and 4 resources, the minimum blocking probability is 0%.

It is seen that the blocking probability and the standard deviation of processor allocations are very small. We can conclude that with a good scheduling algorithm, the Omega and binary n-cube network serve almost equally as well as the distributed cross-bar switch for resource sharing.

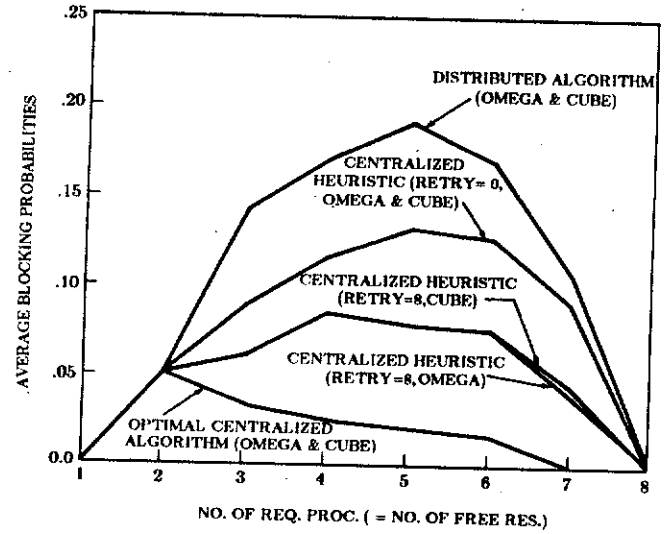


Figure 4—Blocking probability for resource allocations on Omega and Cube networks ($N = 8$)

(b) Centralized Scheduling Heuristic

As a comparison, we present a centralized heuristic and compare its performance against the optimal algorithm. Let

P_R = Set of requesting processors = $\{P_i, P_{ii}, P_{iii}, \ldots P_x\}$
R_A = Set of free resources = $\{R_i, R_{ii}, R_{iii}, \ldots R_y\}$

We assume that the processors and resources in P_R and R_A are ordered by their index numbers. A parameter of the heuristic

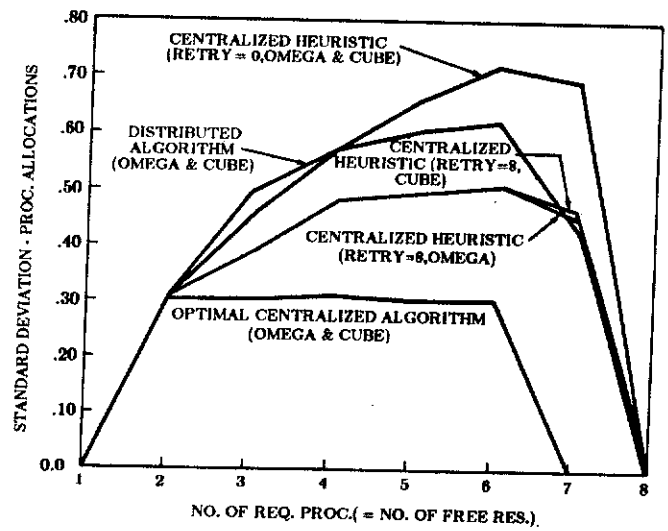


Figure 5—Standard deviation of number of requesting processors allocated for Omega and Cube networks ($N = 8$)

is the number of retries ($0 \le \text{RETRY} \le y - 1$). Supposed P_i fails to be connected to R_j due to a blocked connection, then the heuristic successively retries the next set of RETRY free resources to see if a connection can be made. Whether a connection can be made within the fixed number of retries or not, the next processor in P_R is always matched with the first free resources in R_A immediately following the resource matched for the current processor. The heuristic, written in pidgin Algol, is shown in Figure 6.

```
PROCEDURE cent_heuristic

/*  Assume that match (P_i, R_j) is a boolean procedure which
    returns TRUE if P_i can be connected to R_j and FALSE
    otherwise.  If TRUE is returned, the connection is actually made.

    i - index of a requesting processor (i = φ means there is no
        requesting processor)

    j - index of a free resource (j = φ means there is no free resource)

    r - variable indicating the number of retries
*/

i=1; j=1;  /* initialization */

WHILE (i ≠ φ AND j ≠ φ) DO
    BEGIN
    done = FALSE; r=0;

    WHILE (NOT match (P_i, R_j) AND done .EQ. FALSE) DO
        BEGIN
        r = r+1;
        IF (r > RETRY)  /* Test for # of retries */
           THEN  done = TRUE
           ELSE  j ← next free resource in R_A;
        END;

    i ← next requesting processor in P_R;
    j ← next free resource in R_A;
    END
END
```

Figure 6—Centralized heuristic for resource allocation

To illustrate the heuristic, consider an 8 by 8 Omega network with $P_R = \{0, 3, 4, 5\}$, $R_A = \{0, 1, 3, 4\}$ (see Figure 8),

1. The algorithm connects P_0 to R_0 and is successful.
2. The algorithm connects P_3 and R_1 and is successful.
3. The algorithm tries to connect P_4 to R_3, but is blocked.

If RETRY = 0, then the algorithm connects P_5 to R_4 and is successful.

If RETRY = 1, then the algorithm tries to connect P_4 to R_4 and is successful. It continues to connect P_5 to R_3 and is successful. For this example, the resource utilization is 100% if RETRY ≥ 1, otherwise, it is 75% for RETRY = 0.

The procedure *match* in Figure 6 has a complexity of $0(\log_2 N)$. (It is proportional to the number of stages in the network). The worst case complexity of the heuristic for x requesting processors and y free resources ($0 < x, y \le N$) is, therefore, $0(N(\text{RETRY} + 1)\log_2 N)$. If RETRY = 0, the heuristic has complexity $0(N^* \log_2 N)$. If RETRY = $N - 1$, the heuristic has complexity $0(N^2 \log_2 N)$.

Since the heuristic assumes a predetermined sequence of allocations and no backtracking is provided if a wrong decision is made, the heuristic is sub-optimal. The performance of the heuristic with RETRY = 0 and RETRY = 8 are shown in Figures 4 and 5. It is seen that the blocking probability is higher than the optimal case (around 7%). As the number of retries is increased, the blocking probability reduces. Further, the Omega and the binary n-cube networks have different performance on the centralized heuristic. This is due to the fact that the order in which resources are tried are different in the two networks. Although the Omega and binary n-cube networks seem to have identical performance for the centralized heuristic with RETRY = 0, the Omega network has worse performance when the number of processors and resources are different.

4. DISTRIBUTED RESOURCE SCHEDULING ON NETWORKS WITH LOGARITHMIC DELAYS

The centralized scheduling algorithm has a high overhead when the number of processors and resources to be scheduled is large since every requesting processor has to be scheduled sequentially. In a distributed algorithm, all the requesting processors are 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 scheduling intelligence into the interconnection network so that there is no centralized control. Each exchange box can resolve conflicts and route requests to the appropriate destination. 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. The information paths for exchange box j in stage i, $B_{i,j}$, is shown in Figure 7. There are four types of signals, S, Q, J and D:

S—carries information about the number of resources reachable from this link

$$Q = \begin{cases} 1 & \text{a request of a free resource is made on this link} \\ 0 & \text{otherwise} \end{cases}$$

$$J = \begin{cases} 1 & \text{a block has been detected in stages after the current stage and the request along this link is rejected} \\ 0 & \text{otherwise} \end{cases}$$

D—data transmission links

RA—resource availability register which stores the number of resources reachable from an output terminal of $B_{i,j}$.

For simplicity of representation, subscripts of symbols for signals incident upon and originating from $B_{i,j}$ are set to be i,j. The index of the box that they are connected to is not included in the representation as a mapping function. There are four types of superscripts, *UL* (upper left), *UR* (upper right), *LL* (lower left) and *LR* (lower right) and they indicate the corner of the exchange box that the signal links are connected to. The distributed scheduling algorithm utilizes these signals to connect the data paths from the $i-1$-*st* stage ($D^{UL}_{i-1,j}$, $D^{LL}_{i-1,j}$) to the $i+1$-*st* stage ($D^{UR}_{i,j}$, $D^{LR}_{i,j}$).

Consider a situation when the network is completely free,

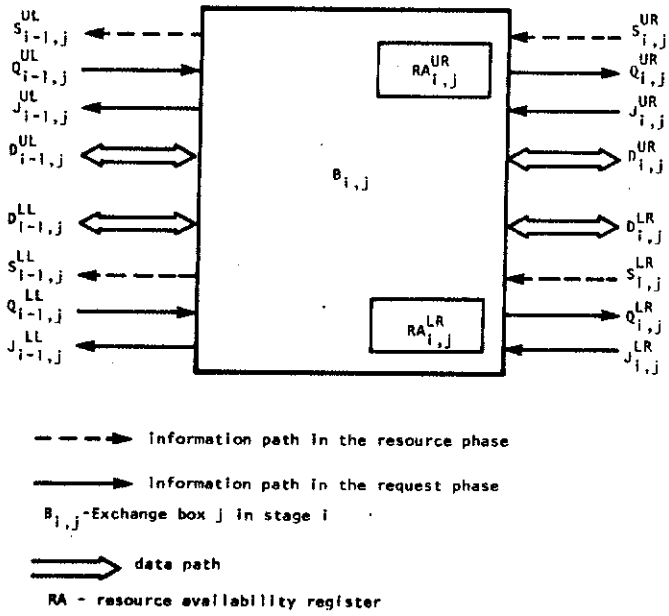


Figure 7—The information paths of an exchange box in the distributed scheduling algorithm

and there is a set of requesting processors and free resources. All the resource availability registers are set to be zero initially. We will generalize later to the situation in which requests can be initiated dynamically.

The algorithm consists of two phases. In the first phase *(Resource Phase)*, information concerning the number of free resources is passed from the resource side to the processor side. Specifically, each resource that is free sends a "+1" along the S link to the exchange box connected to it. Referring to Figure 7, which shows exchange box $B_{i,j}$, the dashed lines represent the information flow in the resource phase. The exchange box receiving this information increments the corresponding resource availability registers. It then adds the two numbers stored in the two resource availability registers and sends the result to the two exchange boxes connected in stage $i-1$. Conceptually, the numbers $S_{i,j}^{UR}$ and $S_{i,j}^{LR}$ represent the number of resources reachable from the upper and lower output terminals of $B_{i,j}$. Therefore, the total number of resources reachable from this box is $S_{i,j}^{UR}+S_{i,j}^{LR}$ and this information is passed to the two exchange boxes connected in the previous stage along links $S_{i-1,j}^{UL}$ and $S_{i-1,j}^{LL}$. The delay for this phase is proportional to the number of stages of the network.

As an example, refer to Figure 8. Suppose processors P_0, P_3, P_4 and P_5 are making requests and resources R_0, R_1, R_4 and R_5 are free. Resource availability information is passed from the resource side to the processor side starting with stage 2. Box $B_{2,0}$ receives "+1" from R_0 and R_1. Therefore, it passes a "+2" to boxes $B_{1,0}$ and $B_{1,1}$. Likewise, box $B_{2,2}$ receives "+1" from R_4 and R_5 and passes this information to boxes $B_{1,2}$ and $B_{1,3}$. The propagation of this information is similar in stages 1 and 0. At the end of the resource phase, P_0, P_3, P_4 and P_5 know that there are 4 resources available.

In the second phase *(Request Phase)*, the network propagates the requests from the processors to the resources. This uses the information that is obtained in the resource phase. The maximum total number of request and rejection signals pending in each exchange box is 2 since the exchange box can only make two connections at any time. For example, it is impossible to have two rejection signals received together with a request signal, because in order for the rejection signals to be received, two request signals must have been received earlier. A new request cannot be received until the two previous requests have been rejected. Therefore, we can have any one of the following six combinations of signals pending in an exchange box: 2 Qs, 2 Js, 1 Q and 1 J, 1 J, 1 Q, or no signal pending.

When multiple signals are pending in a box, priority must be set to determine the order of servicing these requests. Two priority rules are used:

(P1) For two request signals received ($Q_{i-1,j}^{UL}=1$, $Q_{i-1,j}^{LL}=1$), the request originating from the top input terminal ($Q_{i-1,j}^{UL}$) has priority over the other ($Q_{i-1,j}^{LL}$).
(P2) For one request and one reject signal received, the reject signal has priority over the request signal in service.

In servicing a request or a reject, two service rules are applied.

(S1) To service a request ($Q_{i-1,j}^{UL}$ or $Q_{i-1,j}^{LL}$), find a free output link where free resources can be accessed (contents of resource availability register is greater than zero). If both output links are free, then $S_{i,j}^{UR}$ is checked before $S_{i,j}^{LR}$. If such an output link is found, the output link is marked busy so that no further request can be made along this link and a request is sent to stage $i+1$. If the free output links do not lead to any free resources, a reject signal is sent from the original input terminal to stage $i-1$.
(S2) To service a reject ($J_{i,j}^{UR}$ or $J_{i,j}^{LR}$), the corresponding resource availability register ($RA_{i,j}^{UR}$ or $RA_{i,j}^{LR}$) is set to zero to indicate that no free resource is reachable from this output terminal. The output terminal is marked free and service rule (S1) is applied to search for another available output terminal where free resources can be reached.

For the six possible input combinations of signals pending in $B_{i,j}$, the sequence of priority and service rules applied is shown in Table II.

If a request successfully reaches a free resource, the resource sends a "−1" along the S link to the exchange box connected to it. For exchange box $B_{i,j}$ receiving a "$-k$" ($k=1, 2, \ldots$) along the S link ($S_{i,j}^{UR}=-k$ or $S_{i,j}^{LR}=-k$), if the content of the corresponding resource availability register is zero, then nothing is done. If not, the corresponding resource availability register is decremented and the "$-k$" information is passed to stage $i-1$ along $S_{i-1,j}^{UL}$ and $S_{i-1,j}^{LL}$. If both $S_{i,j}^{UR}$ and $S_{i,j}^{LR}$ are negative ($S_{i,j}^{UR}=-k_1$ and $S_{i,j}^{LR}=-k_2$), then both $RA_{i,j}^{UR}$ and $RA_{i,j}^{LU}$ are decremented and "$-(k_1+k_2)$" is sent along $S_{i-1,j}^{UL}$ and $S_{i-1,j}^{LL}$ to stage $i-1$.

Referring to the example in Figure 8, $B_{1,1}$ 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

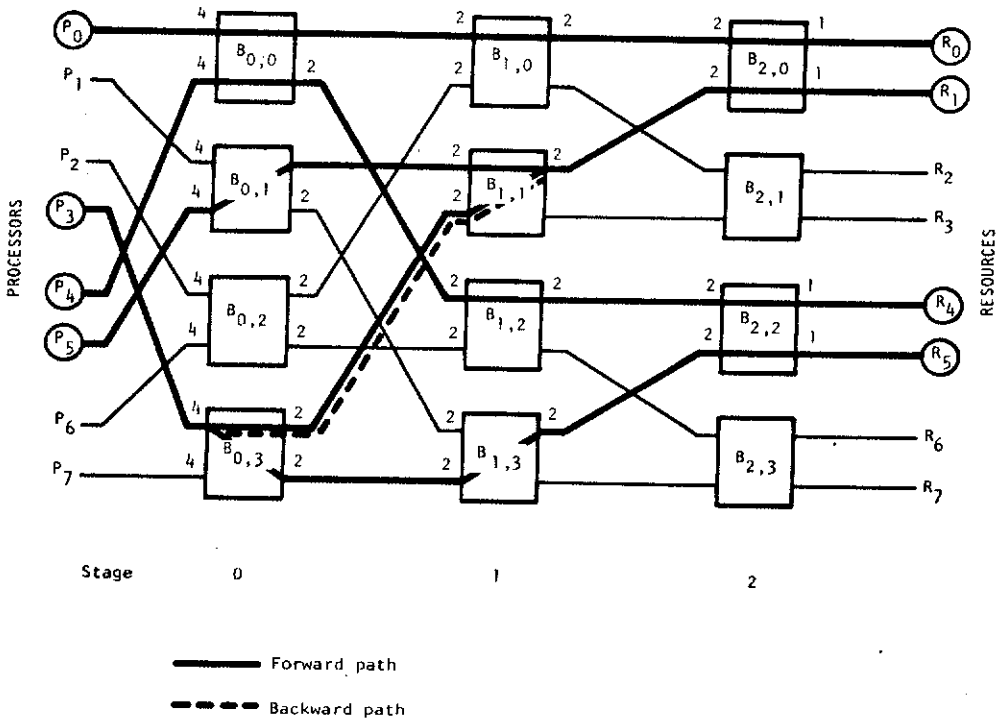


Figure 8—Example of Omega network with four requesting processors and four free resources (25% of requests are blocked and backtracked; 100% resource allocation; average delay = 4.67 units)

request, subsequently, finds another route via $B_{1,3}$ and $B_{2,2}$ to R_5. The average delay time is 4.67 units in this example (a unit is the time to pass through an exchange box).

The algorithm described above does not preclude dynamic operation. In fact, requests can be initiated at random times and they will be routed to a free resource or be rejected. The operation of the exchange box can be completely asynchronous. An accepted request is known to a processor when an acknowledgement is received along the data link. A request is rejected when a rejection signal is received by the processor along the J signal link. A rejected request can be retried later.

The performance of the distributed algorithm is again plotted in Figures 4 and 5 and it is identical for the Omega and binary n-cube networks. It is seen that the blocking probability is less than 20% in all cases and compares favorably with the optimal algorithm and centralized heuristic. The standard deviation is approximately doubled as compared with the optimal case. The average delay time for a request to access a free resource or be rejected is shown in Figure 9. The delay is never greater than 4.2 units of time in which the delay through an exchange box is 1 unit. The delay time of the algorithm is dependent on the delay in the network and not on the number of requesting processors.

TABLE II—Sequence of priority and service rules applied for the six possible combinations of signals pending in $B_{i,j}$

Combinations of signals pending in $B_{i,j}$	Sequence of priority and service rules applied
2 Q	P1, S1, S1
2 J	S2, S2
1 Q, 1 J	P2, S2, S1
1 Q	S1
1 J	S2
0 Q, 0 J	no action

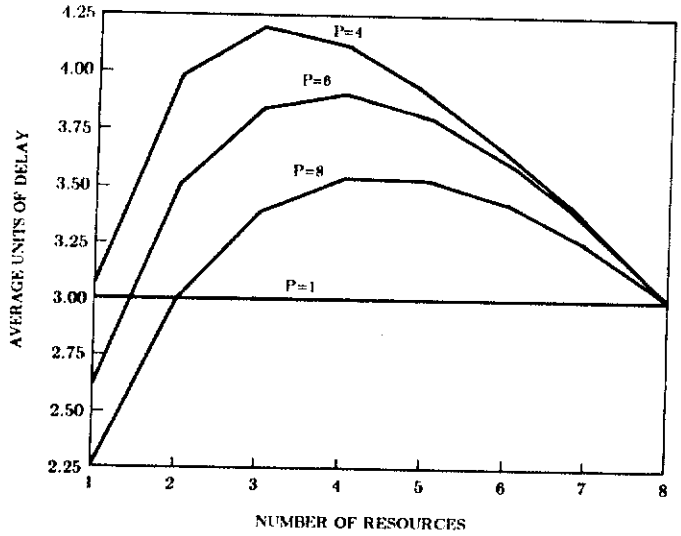


Figure 9—Average delay time for distributed algorithm in 8 by 8 Omega and binary n-cube networks (a unit is the time to pass through an exchange box)

5. CONCLUSION AND EXTENSIONS

In this paper, we have presented the scheduling of resources on interconnection networks. The resource scheduling problem is different from the conventional address mapping problem on interconnection networks because a request is not directed towards a particular destination address but to any one of a pool of destination addresses for free resources. A broadcast technique does not work effectively because it precludes other processors in making requests when one of the processors is making a request. Centralized scheduling algorithms are inefficient because all the requesting processors must be serviced sequentially. A distributed scheduling algorithm allows the scheduling of all the processors to be performed in time proportional to the delay in the network.

An interconnection network for resource sharing may operate in two ways. First, the network is "circuit switched" and the processor and resource are continuously connected for the duration of use. In this case, the network may be partially busy when a new request is initiated. To avoid excessive blocking, the network should provide conflict-free access even when other connections are present. A distributed scheduling algorithm is designed for the cross-bar switch and the implementation of a cross-bar switch cell is presented. Each cell can be implemented with 12 gates and a flip-flop when the data path is one bit wide. The cell is designed with the minimum number of signal lines. As a result, the scheduling algorithm is divided into two phases and the algorithm is asymmetric, that is, priority is induced into the scheduling of multiple requests. In order to allow the algorithm to operate in one phase, more signal lines are needed between adjacent cells as shown in the distributed Omega and binary n-cube networks. To remove the assymmetry in the scheduling, each resource can send a short pulse along the resource availability line. This pulse acts like a token and one of the requesting processors receiving this pulse will be allocated. When different resources issue tokens randomly, the algorithm is symmetric.

Second, the network is "packet switched" or operates in block-transfer mode. In this case, the resources are connected to the processors for a short duration of time and can relinquish the network after tasks are assigned. When a new set of requests are initiated, the network is almost or completely free. Networks with logarithmic delays are suitable for this application. An optimal centralized scheduling algorithm has been studied for the Omega and binary n-cube networks. It is shown that there is an average blocking probability of 1%. This means that these networks have behavior close to the cross-bar switch for resource sharing in a block transfer mode.

The centralized optimal algorithm has exponential time complexity. We studied, respectively, two centralized heuristics (with time complexities $O(N^2 \log_2 N)$ and $O(N \cdot \log_2 N)$) and a distributed algorithm (with time complexity $0(\log_2 N)$). In the distributed algorithm, each exchange box in the network operates asynchronously and is responsible for resolving multiple requests directed to it. Resource availability information is also passed along the network to the processors. The control of the network can be hardwired or microprogrammed. The blocking probability increases as the time complexity decreases. In the worst case (distributed algorithm), the blocking probability is around 19%.

Several extensions can be included in the design. We discuss them briefly here.

1. The resources connected on the network do not have to be identical. In a general system, there are multiple types of resources, each type is made up of a set of identical resources. The algorithms discussed have to be modified by identifying the type of resource requested by a processor and the type of resource available on a resource availability line. This can be done by sending a binary request code (instead of 1 bit) in the distributed algorithms. In the distributed Omega and binary n-cube networks, multiple resource availability registers have to be used in each exchange box.
2. There is a tradeoff between the time complexity of the algorithm and the number of signal lines between two adjacent cells in the distributed RSINs. A one-bit data paths is assumed in the distributed cross-bar switch. In the distributed Omega and binary n-cube networks, parallel data paths are assumed. This can be reduced by appropriate multiplexing at the external chip interface level.
3. The scheduling algorithms can be extended to the case when multiple resources are requested by a processor to operate in a broadcast mode. In the distributed cross-bar switch, a count can be sent with a request signal. Each time a free resource is allocated to this request, the count is decremented by one before it is sent to the next cell. In the centralized heuristic for the Omega and binary n-cube networks, request for multiple resources is serviced by searching for a free resource and allocating it until the required number of resources are allocated. In the distributed Omega and binary n-cube networks, the exchange boxes are extended so that they can operate in broadcast mode. That is, an input terminal to an exchange box can be connected to the two output terminals simultaneously. The algorithm is modified to proceed as

follows. A count K is sent with each request or reject signal. This count indicates the number of additional resources needed by the request or reject. Referring to Figure 7, a request is sent to the upper request line ($Q_{i,j}^{UR}$) if the content of the resource availability register, ($RA_{i,j}^{UR}$), is greater than K. Otherwise, the request is sent along both the upper and lower request lines. Of course, if the content of any resource availability register is zero, a request is not sent along the corresponding request line. If $(RA_{i,j}^{UR}) + (RA_{i,j}^{LR}) > K$, that is, the request cannot be satisfied completely, then a reject signal with count $= K - (RA_{i,j}^{UR}) - (RA_{i,j}^{LR})$ is sent from the original input terminal to stage $i-1$ to search for additional resources. It is also assumed in the generalized distributed algorithm, that the two priority rules (P1 and P2) are followed. To avoid deadlock in the generalized distributed algorithm, a requesting processor should relinquish its allocated resources if it cannot find the required number of resources and resubmit its request again later.

The distributed algorithm implemented in each exchange box does not preclude operation under the address mapping mode. Further, the theory underlying the design of the distributed Omega and binary n-cube networks can be applied to other interconnection networks such as the Banyan,[7] and Delta.[14] In these networks, the number of processors and the number of resources can be different. The performance will be evaluated in the future. Future studies also include the performance evaluation of the algorithm under dynamic operations.

REFERENCES

1. Barnes, G. H., and S. F. Lundstrom. "Design and Validation of a Connection Network for Many-Processor Multiprocessor Systems." *IEEE Computer*, 14 (1981) pp. 31–41.
2. Batcher, K. E., "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. Feng, T. "Data Manipulating Functions in Parallel Processors and Their Implications." *IEEE Trans. Computer*, Vol. C-23, No. 3, pp. 309–318, Mar. 1974.
6. Franklin, M. A. "VLSI Performance Comparison of Banyan and Cross-bar Communication Networks." *Proc. of Workshop on Interconnection Networks*, pp. 20–28, Apr. 1980.
7. Goke, L. R., and G. J. Lipovski. "Banyan Networks for Partitioning Multiprocessor Systems," *Proc. 1st Annual Comp. Architecture Conf.*, pp. 21–28, Dec. 1973.
8. Goke, L. R., *Banyan Networks for Partitioning Multiprocessor Systems* Ph.D. Thesis, Univ. of Florida, 1976.
9. Jenevein, R., D. Degroot and G. J. Lipovski. "A Hardware Support Mechanism for Scheduling Resources in a Parallel Machine Environment." *Proc. of 8th Annual Symposium on Computer Architecture*, pp. 57–66, May 1981.
10. Kuck, D. J. "ILLIAC IV Software and Application Programming." *IEEE Trans. on Comp.*, Vol. C-17, pp. 746–757, Aug. 1968.
11. Lawrie, D. "Access and Alignment of Data in an Array Processor." *IEEE Trans. Computers*, Vol. C-24, No. 12, pp. 215–255, Dec. 1975.
12. McDonald, W. C. and J. M. Williams, "The Advanced Data Processing Test Bed." *Proc. of COMPSAC 78*, pp. 346–351, March 1978.
13. Ornstein, S. M., et al., "Pluribus—A Reliable Multiprocessor." *Proc. AFIPS 1975 National Computer Conference*, AFIPS Press, Montvale, N.J., pp. 551–559, 1975.
14. Patel, J. H. "Performance of Processor-Memory Interconnections for Multiprocessors." *IEEE Trans. on Computers*, Vol. C-20, No. 10, pp. 771–780, Oct. 1981.
15. Pease, M. C. "The Indirect Binary n-cube Microprocessor Array," *IEEE Trans. on Computers*, Vol. C-26, No. 5, pp. 458–473, May 1977.
16. Rathi, B. D., 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. Sejnowski, M. C., et al. "Overview of the Texas Reconfigurable Array Computer." *AFIPS Conference Proceedings*, Vol. 49, pp. 631–642, 1980.
18. Siegel, H. J., and R. J. McMillen. "The Cube Network as a Distributed Processing Test Bed Switch." *2nd Int'l. Conf. on Dist. Comp. Sys.*, pp. 377–387, April 1981.
19. Siegel, H. J., and R. J. McMillen. "Using the Augmented Data Manipulator Network in PASM." *IEEE Computer*, Vol. 14, No. 2, pp. 25–33, Feb. 1981.
20. Stone, H. "Parallel Processing with the Perfect Shuffle." *IEEE Trans. on Computers*, Vol. C-20, No. 2, pp. 153–161, Feb. 1971.
21. Wu, C., 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. Wulf, W. A., 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, 1982.