Resource Sharing Interconnection Networks in Multiprocessors

JIE-YONG JUANG, MEMBER, IEEE, AND BENJAMIN W. WAH, SENIOR MEMBER, IEEE

Abstract—**In this paper, circuit-switched interconnection networks for resource sharing in multiprocessors, named *resource sharing interconnection networks*, are studied. Resource scheduling in systems with such an interconnection network entails the efficient search of a mapping from requesting processors to free resources such that circuit blockages in the network are minimized and resources are maximally used. The optimal mapping is obtained by transforming the scheduling problems into various network flow problems for which existing algorithms can be applied. A distributed architecture to realize a maximum flow algorithm using token propagations is also described. The proposed method is applicable to any general loop-free network configuration in which the requesting processors and free resources can be partitioned into two disjoint subsets.**

Index Terms—**Circuit switching, distributed resource scheduling, interconnection network, linear programming, maximum flow, minimum cost flow, multicommodity network flow, resource sharing.**

I. Introduction

IN THIS paper, we investigate the problem on the sharing of computing resources in multiprocessors and the distributed scheduling of shared resources by a circuit-switched interconnection network.

A *resource* is a processing element to carry out a designated function. Examples include a general purpose processor, a special functional unit, a VLSI systolic array, an input/output device, and a communication channel. A resource is accessible by any processor via an interconnection network. A *request* generated by a processor can be directed to any one of a pool of free resources that are capable of executing the designated task. An interconnection network is an essential element of these systems as it interconnects processors and resources. Its function is to route requests initiated from one point to another point connected on the network. The network topology is dynamic, and the links can be reconfigured by setting the network's active switching elements. The notable characteristic of these networks is that they operate with address mapping. That is, a request is initiated with a specific destination or a set of destinations, and routing is done by examining the address bits. Routing of requests is usually done in parallel. As classified by Feng [16], these networks include the single or multistage networks and the crossbar switch. Examples are the Banyan [20], indirect binary n-cube [38], cube [41], perfect shuffle [43], Flip [3], Omega [27], data manipulator [15], augmented data manipulator [42], delta [37], [11], baseline [46], Benes [5], and Clos [9]. Examples of systems designed with interconnection networks are Trac [40], Staran [3], C.mmp [47], Illiac IV [26], Pluribus [34], PASM [48], Numerical Aerodynamic Simulation Facility (NASF) [2], the Ballistic Missile Defense testbed [32], MPP [4], and the Connection Machine [23]. The performance of resource sharing systems under address mapping has been studied by Rathi, Tripathi, and Lipovski [39], Fung and Torng [19], and Marsan, Gregoretti, and Gerla [29], [30].

Wah proposed a network with distributed scheduling intelligence, called *resource sharing interconnection network (RSIN)* [45], [44]. Instead of using an address mapping scheme, which requires a centralized scheduler to seek and give the address of a free resource to a request before it enters the network, the request is sent into the network without any destination tags. It is the responsibility of the network to route the maximum number of requests to the free resources. In this way, the scheduling intelligence is distributed in the network. Distributed resource scheduling avoids the bottleneck of a centralized scheduler [39]. The objective of a good scheduling scheme is to avoid network blockages and to maximize resource utilization, which requires an efficient algorithm at each switching node to collect the minimum amount of status information.

The PUMPS architecture [see Fig. 1(a)] for image analysis and pictorial database management [8] is a typical example of resource sharing multiprocessors, in which various VLSI systolic arrays, each realizing an image processing function, are organized into a pool of resources. Most data flow architectures can also be considered as resource sharing systems. For example, in Dennis' architecture [10] [see Fig. 1(b)], active instructions generated from cell blocks are routed to a processing unit for execution. Hence, the processing units constitute the pool of resources, and an RSIN connects them to cell blocks. In a resource sharing system with load balancing, processors are considered as resources; thus, requests generated are queued at the processors as well as the resources. There may be an imbalance of workload at the resources, and load balancing schemes are used to redistribute requests among resources.

The design of an RSIN with optimal resource scheduling is

Manuscript received May 26, 1986; revised July 6, 1988. This work was supported in part by National Science Foundation Grants MIPS 85-19649 and IRI-8709072 and National Aeronautics and Space Administration Contract NCC 2-481.

J.-Y. Juang is with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208.

B. W. Wah is with the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801.

IEEE Log Number 8823532.

0018-9340/89/0100-0115$01.00

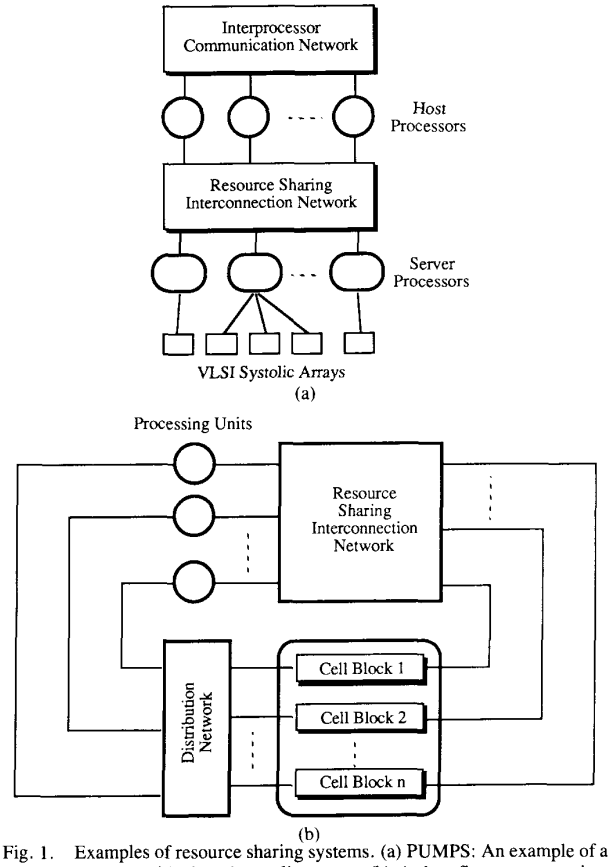


Fig. 1. Examples of resource sharing systems. (a) PUMPS: An example of a multiprocessor with shared systolic arrays. (b) A data flow computer is a resource sharing system (cell blocks are processors, while processing units are resources).

studied in this paper. The results are derived with respect to multistage interconnection networks, called *multistage resource sharing interconnection networks (MRSIN)*, and are applicable to any general loop-free network configuration in which the requesting processors and free resources are partitioned into two disjoint subsets. Central to the design of such an interconnection network is the development of an efficient distributed algorithm to disseminate status information through the complex interconnection structure. The algorithm to be presented is simple, efficient, and independent of the interconnection topology. For a typical interconnection structure, such as the Omega network [27], network blockages can be reduced to less than 5 percent. In the next section, the model of MRSIN is reviewed, and the issues on resource scheduling are discussed. In Section III, we present transformation methods for various scheduling disciplines. These transformations allow optimal request-resource mappings to be obtained through the evaluation of network flow algorithms. Architectures to carry out these algorithms are presented in Section IV. Conclusions are drawn in Section V.

II. Resource Sharing Interconnection Networks

The model of the RSIN used in this study is summarized as follows.

1) Circuit switching is assumed rather than packet switching for the following reasons. First, packet switching is used in conventional networks with address mapping because it allows a network path to be shared by more than one request concurrently. In an RSIN, reducing the packet delay by balancing traffic among alternate routing paths is less critical because a request can always search for another available resource if the network is free. Moreover, the overhead of rerouting a packet when a path or resource is blocked is higher than that of rerouting a resource request. Second, owing to the resource characteristics, a task cannot be processed until it is completely received. The extra delay in breaking a task into multiple packets may decrease the utilization of resources, and hence increase the response time of the system.

2) One or more types of resources may exist in the system. An RSIN connecting only one type of resource is called a *homogeneous* RSIN, while an RSIN connecting multiple types of resources is a *heterogeneous* one.

3) A priority level may be associated with a request to show the urgency of the request. A preference value may be associated with a resource to show the desirability of being used for service. The costs of allocation are inversely related to the priorities and preferences.

4) Each request needs one resource only. When multiple resources are needed, they can be requested from multiple ports concurrently, or can be requested sequentially from a single port.

5) A processor can transmit one task at a time to the resources. Other tasks arriving during the task transmission time are queued. The circuit between a processor and a resource can be released once the request has been transmitted. The processor can continue to make other requests, while the resource will be busy until the task is completed.

We have not investigated the problem on the selection of the number of resources in each type and their placements in the output ports. This problem has been studied by Briggs *et al.*, who have considered the problem of choosing the number of resources in each type in which one resource is connected to each output port and one resource is requested each time [7]. We have not considered the case in which more than one resource or multiple types of resources are requested by one request. Here, the scheduling algorithm is dependent on the number of resources in each type, the way that resources are distributed to the output ports, and the network characteristics. Furthermore, deadlocks may occur, and distributed resolution of deadlock may have a high overhead.

The goal of the scheduling algorithm is to find a request-resource mapping such that the total cost is minimized. In the special case in which all requests are of equal priorities and all resources have equal preferences, the scheduling problem becomes the mapping of the maximum number of requests to the free resources.

The maximal request-resource mapping may be hampered by blockages in the system. In a conventional address-mapped interconnection network, blockages may be caused by conflicts in cases when either the same resource is requested by more than one request or a network link is requested by two circuits. In an RSIN, a resource conflict can be resolved by rerouting all but one request to other free resources. However,

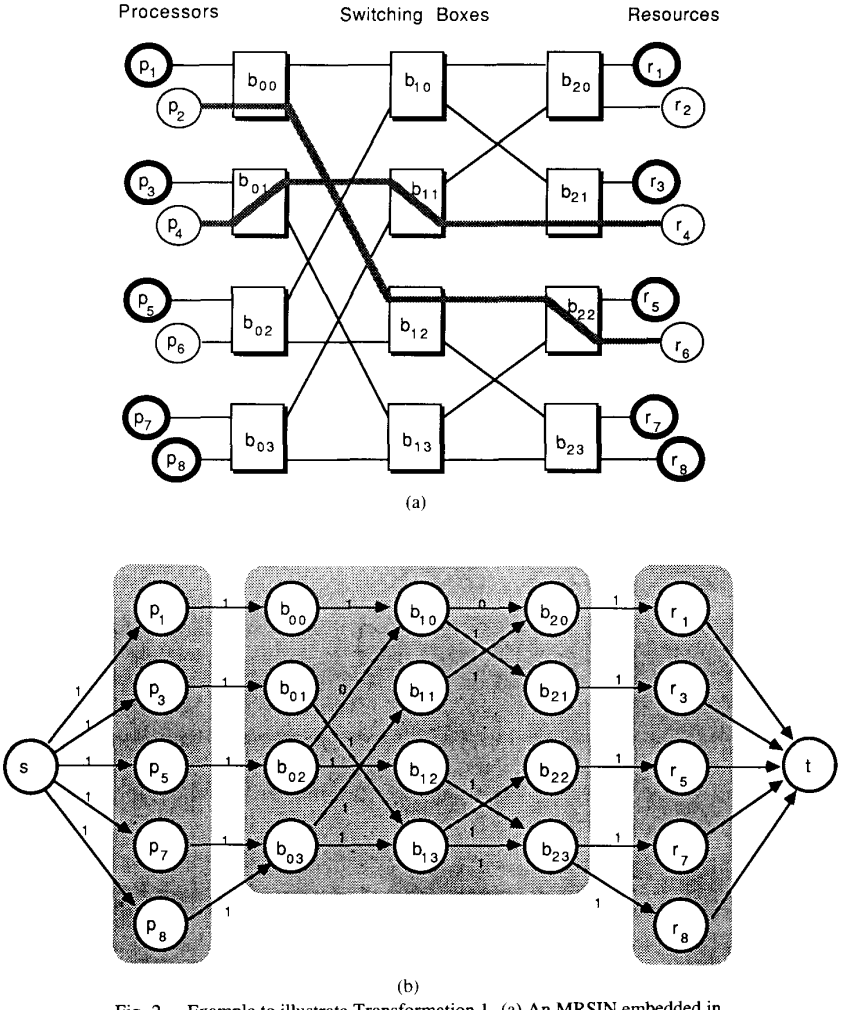


Fig. 2. Example to illustrate Transformation 1. (a) An MRSIN embedded in an 8 × 8 Omega network (thick shaded paths in the network show circuits that are already occupied; processors p_1, p_3, p_5, p_7, and p_8 are making requests; resources r_1, r_3, r_5, r_7, and r_8 are available). (b) The flow network obtained from the MRSIN in Fig. 2(a) using Tranformation 1 (the number associated with each arc is the amount of flow assigned to it by the maximum flow algorithm; all arcs have unit capacity).

this may not always lead to better resource utilization because the allocation of one request to a resource may block one or more other requests from accessing free resources. A scheduling algorithm that schedules requests according to the state of the network and resources is, therefore, essential. The necessity for a proper scheduler to give the maximum resource utilization is illustrated in the following example. Consider an 8 × 8 Omega network[1] [see Fig. 2(a)] with switchboxes that can be individually set to either a straight or an exchange connection. Processors p_1, p_3, p_5, p_7, and p_8 are requesting one resource each, and resources r_1, r_3, r_5, r_7, and r_8 are available. The circuits between p_2 and r_6 and p_4 and r_4 have been established previously. p_6 is not making a request, and r_2 is busy. All free resources will be allocated if one of the following request-resource mappings is used: $\{(p_1, r_3), (p_3, r_5), (p_5, r_7), (p_7, r_1), (p_8, r_8)\}$ or $\{(p_1, r_3), (p_3, r_8), (p_5, r_7), (p_7, r_1), (p_8, r_5)\}$. But if the request-resource mapping $\{(p_1, r_1), (p_3, r_5), (p_5, r_3), (p_7, r_7), (p_8, r_8)\}$ is used, then a maximum of four out of five resources can be allocated, since the path leading from p_8 to r_8 is blocked. Simulation results showed that the average blocking probability can be as low as 2 percent for an MRSIN embedded in an 8 × 8 cube network [44], [22]. If a heuristic routing algorithm is used, then the average blocking probability increases to around 20 percent. In the simulation, the network is assumed to be completely free, i.e., no link is occupied for other purposes. If the network is not completely free, then there will be fewer paths available for resource allocation. In this case, a heuristic routing algorithm may have poor performance. An optimal

[1] The input ports are numbered in a different way from Lawrie's Omega network [27] because all resources are homogeneous, and the permutation of requesting processors will not affect the resource utilization. Broadcast connection is not needed in the switchboxes since each request needs one resource.

scheduling algorithm will be able to better utilize these paths, and result in a low blocking probability (although it will be higher than that of the case when the network is completely free). If extra stages are provided, there will be more paths available. Resources may be fully allocated in most cases even when an arbitrary resource–request mapping is used. Finding an optimal mapping becomes less critical.

III. Optimal Resource Scheduling in MRSIN

To bind a request to a resource in the system, an RSIN determines a mapping from pending requests to free resources, and provides connections to as many request–resource pairs as possible. The objective of a scheduling algorithm is to obtain the optimal mapping that optimizes certain performance indexes, such as resource utilization. In this section, methods to optimize request–resource mappings are discussed. Exhaustive methods that examine all possible ordered mappings have exponential complexity. In a homogeneous MRSIN, suppose x processors are making requests, y resources are available, and the network is completely free. The scheduler has to try a maximum of $C_y^x \cdot y!$ (for $x \geq y$) or $C_x^y \cdot x!$ (for $y \geq x$) mappings to find the best one, where C_i^j is the number of combinations of choosing i objects out of j objects [44], [22]. Suboptimal heuristics can be used but it is only practical when x and y are small.

In this section, we transform the optimal request–resource mapping problem into various network flow problems for which many efficient algorithms exists [21], [24]. The basic concepts of flow networks are briefly reviewed first.

A. Flow Networks

A flow network is usually represented by a digraph in which each arc is associated with a capacity and possibly a cost. Let $\boldsymbol{D} = (\boldsymbol{V}, \boldsymbol{E})$ be a digraph with two distinct nodes: s (*source*) and t (*sink*). A capacity function $c(e)$ is defined on every arc of the graph, where $c(e)$ is a nonnegative real number for all $e \in \boldsymbol{E}$. A flow function f is an assignment of a real number $f(e)$ to arc e such that the following conditions hold.

1) *Capacity limitation:* For every arc $e \in \boldsymbol{E}$,

$$0 \leq f(e) \leq c(e).$$

2) *Flow conservation:* Let $\alpha(v)$ [resp., $\beta(v)$] be the set of incoming (resp., outgoing) arcs of vertex v. For every $v \in \boldsymbol{V} - \{s, t\}$,

$$\sum_{e \in \alpha(v)} f(e) = \sum_{e \in \beta(v)} f(e).$$

The capacity constraint restricts the amount of flow that can be assigned to a link; while flow conservation implies that an intermediate node in the flow network does not absorb or create flows.

A *legal flow* is a flow assignment that satisfies the capacity and flow conservation constraints. In a network flow problem, it is necessary to find the legal flow that optimizes a given objective function. For example, in the maximum flow problem, it is necessary to find F, the maximum amount of flow that can be advanced from source to sink under the capacity and flow conservation constraints. Given a flow network $\boldsymbol{G}(\boldsymbol{V}, \boldsymbol{E}, s, t, c)$, the maximum flow problem can be formulated as a linear programming problem as follows.

Maximum Flow Problem:

Maximize F

subject to

$$1) \sum_{e \in \alpha(v)} f(e) - \sum_{e \in \beta(v)} f(e) = \begin{cases} -F & v = s \\ F & v = t \\ 0 & \text{otherwise} \end{cases}$$

(flow conservation)

$$2) \ 0 \leq f(e) \leq c(e) \qquad \text{for all } e \in \boldsymbol{E}$$

(capacity limitation).

Many other examples of network flow problems, including the minimum cost flow and the transshipment problems, can be found in the literature [21].

B. Optimal Resource Mapping in Homogeneous MRSIN

A switchbox in an MRSIN is a crossbar switch without broadcast connections. We establish the following theorem to show that setting a nonbroadcast switch is equivalent to finding a legal integral flow assignment in a flow network of unit capacity. Note that an integral flow is a flow assignment in which the amount of flow assigned to each link is of integral value.

Theorem 1: For any MRSIN, there exists a flow network for which a legal integral flow is equivalent to a valid request–resource mapping.

Proof: Consider an $n \times m$ switchbox, where n is the number of input ports and m is the number of output ports. A nonbroadcast switch setting is one in which an input link is connected to at most one output link and vice versa. This switchbox can be transformed to a node v in a flow network with n incoming arcs and m outgoing arcs, i.e., $|\alpha(v)| = n$ and $|\beta(v)| = m$. The capacities of these arcs are set to unity. If one unit of flow is assigned to arcs for which their corresponding links of the switch are connected, then the flow conservation and link capacity constraints are satisfied at node v. Therefore, a switch setting is equivalent to an integral flow assignment for the corresponding node in the flow network. In other words, there is a direct correspondence between a switchbox and a node, and between a switch setting and a flow assignment. □

To use existing algorithms to solve a flow problem, an MRSIN has to be transformed into a flow network such that the optimization of request–resource mappings is equivalent to the optimization of the corresponding objective function in the flow network. To this end, additional nodes may be introduced, the capacity of a link may be greater than one, and a cost may be associated with a link.

The following transformation produces a flow network such that the optimal request–resource mapping can be derived from its maximum flow.

Transformation 1: Generate a flow network $\boldsymbol{G}(\boldsymbol{V}, \boldsymbol{E}, s, t, c)$, from a homogeneous MRSIN.

(T1) Create three node sets $\boldsymbol{P}$, $\boldsymbol{X}$, and $\boldsymbol{R}$ for processors,

switchboxes, and resources, respectively. Introduce two additional nodes: source s and sink t. Let

$$V' = \{s, t\} \cup P \cup X \cup R.$$

(T2) Add an arc leading from the source s to every node associated with a processor. Denote this set of arcs by S, i.e.,

$$S = \{(s, v) | v \in P\}.$$

Add an arc between every node associated with a resource and the sink t. This set of arcs is called T.

$$T = \{(v, t) | v \in R\}.$$

For each link in the MRSIN that connects two switchboxes, or a processor to a switchbox, or a switchbox to a resource, add an arc between the corresponding nodes in the flow graph. Denote this set of arcs by B.

$$B = \{(v, w) | v \in P \cup X, w \in X \cup R\}.$$

Define

$$E' = S \cup T \cup B.$$

(T3) Assign link capacities according to the following function.

$$c(e)_{e \in B} = \begin{cases} 0 & \text{associated link is occupied or nonexistent in the MRSIN} \\ 1 & \text{associated link is free} \end{cases}$$

$$c(e)_{e \in S} = \begin{cases} 0 & \text{associated processor does not generate request} \\ 1 & \text{associated processor generates request} \end{cases}$$

$$c(e)_{e \in T} = \begin{cases} 0 & \text{associated resource is unavailable} \\ 1 & \text{associated resource is available.} \end{cases}$$

(T4) Obtain arc set E by removing those arcs with zero capacity.

$$E = E' - \{e | e \in E', c(e) = 0\}.$$

Obtain node set V by deleting those nodes that are not reachable from s.

Applying the above transformation to the MRSIN in Fig. 2(a) results in the flow network in Fig. 2(b). The following theorem shows that Transformation 1 can be used to find the optimal request–resource mapping.

Theorem 2: In a homogeneous MRSIN, the number of resources allocated by a mapping is equal to the amount of integral flow that can be advanced from the source to the sink in the flow network obtained by Transformation 1.

Proof: The nodes in the flow network transformed from a multistage interconnection network can be divided into stages, and an arc is assigned either zero or one unit of flow. In a flow network corresponding to a loop-free interconnection network with an arbitrary configuration, dummy nodes can be added to equalize all s–t paths and organize the network into stages. An integral flow assignment to nodes in stage i defines a one-to-one mapping g_i that maps a subset of incoming arcs to a subset of outgoing arcs. Hence, a legal integral flow assignment in such a network can be represented by a composite function $h = g_1 g_2 \cdots g_L$, where L is the number of stages. Since g_i is one-to-one, h is also one-to-one. For a one-to-one function, the norm of its domain is equal to the norm of its range. According to Transformation 1, the stage next to the source is comprised of nodes associated with the requesting processors, and the stage next to the sink is comprised of nodes associated with the free resources. Each node in these two stages has a single incoming arc and a single outgoing arc. Therefore, the one-to-one mapping of h implies the same number of nodes involved in the flow assignment in each of these two stages. Furthermore, the flow assignment defines a path for a requesting processor to a free resource, and the free resource can be allocated to the requesting processor through this path. The norm of h is equal to the total flow leaving the source and entering the sink. Thus, $|I| = |O| = F$, where I and O represent the domain and range of g_i, and F is the value of the flow. As a result, every legal integral flow defines a set of F nonoverlapping paths from s to t, and the number of resources allocated is equal to the value of the corresponding flow in the flow network. □

From Theorem 2 and a known result that the maximum flow of a network with integral capacity is integral [6], we conclude that the optimal request–resource mapping can be derived from the maximum flow in the transformed flow network.

Many algorithms have been developed to obtain the maximum flow in a flow network. The algorithm by Ford and Fulkerson [17] is a primal-dual algorithm in which the flow value is increased by iteratively searching for *flow augmenting paths* until the minimum cut-set of the network is saturated. At this point, no more flow can be advanced since the minimum cut-set is the bottleneck. A flow augmenting path is an s–t path through which additional flow can be advanced from the source to the sink. When an arc e on the s–t path points in the same direction as the s–t path, additional flow may be advanced through e if the current flow assigned to e is less than $c(e)$, the capacity of the arc. In contrast, if arc e points in the opposite direction as the s–t path, then additional flow may be pushed through the s–t path by cancelling its current flow. Advancing flow through an augmenting path in this way will always increase the total amount of flow, and the flow conservation and capacity limitations will not be violated.

As an example, in Fig. 3(a), an original flow f is assigned along path s–a–d–t. Path s–c–d–a–b–t is a possible flow augmenting path [see Fig. 3(b)]. Advancing one unit of flow through this augmenting path results in a new flow assignment f'. Two units of flow are pushed through two separate paths s–a–b–t and s–c–d–t according to this assignment [see Fig. 3(c)].

In an MRSIN, advancing a flow through an augmenting path is equivalent to a *resource reallocation*, i.e., a permutation of the possible request–resource mappings. Consider the MRSIN in Fig. 4(a), which is a counterpart of the flow network in Fig. 3.[2] The original flow f is equivalent to the request–resource mapping $\{(p_a, r_d), (p_c, r_b)\}$. The allocation

[2] Although the switchboxes are combined with the processors or resources in the flow network in Fig. 3, the discussion will not be affected.

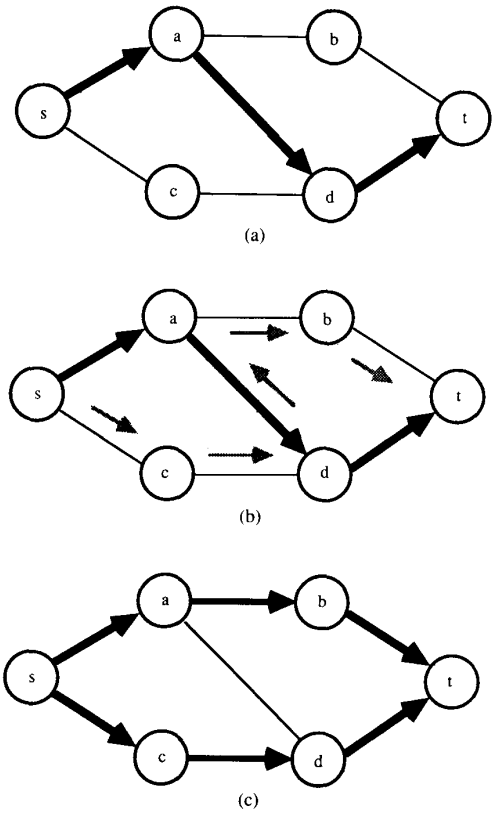


Fig. 3. An illustration of advancing flow through a flow augmenting path (all arcs have unit capacity). (a) A flow network with an initial flow assigned to path s–a–d–t. (b) A flow augmenting path s–c–d–a–b–t exists in the network. (c) Final flow assignment after advancing a unit of flow through the flow augmenting path.

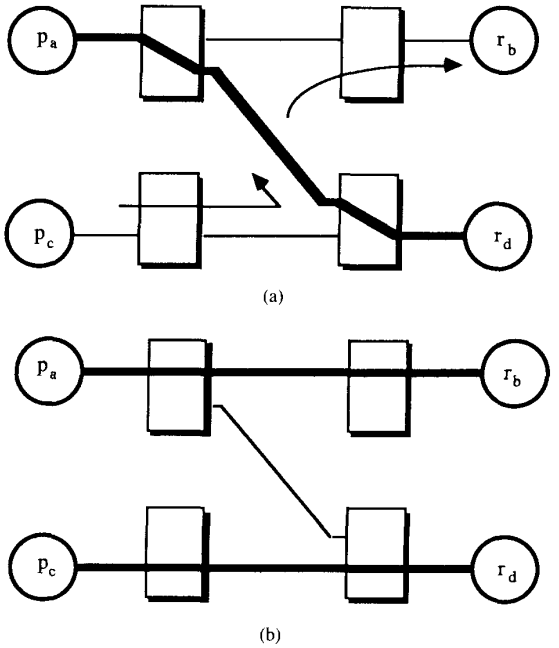


Fig. 4. A resource reallocation corresponding to flow augmentation in Fig. 3. (a) An initial resource allocation and a possible reallocation. (b) Two resources are allocated after reallocation.

of resource r_b to request p_c is blocked in this mapping. The existence of the flow augmenting path s–c–d–a–b–t shows that this blockage can be removed. Fig. 4(b) shows that advancing flow through this augmenting path results in a new mapping $\{(p_a, r_b), (p_c, r_d)\}$ and the allocation of both resources. As another example, applying the maximum flow algorithm to the flow network in Fig. 2(b), the flow assignment as shown in the figure is obtained, and the request–resource mapping $\{(p_1, r_3), (p_3, r_5), (p_5, r_8), (p_7, r_1), (p_8, r_7)\}$ is derived. Those mappings that cannot allocate all the free resources are eliminated by the maximum flow algorithm. Note that there may be more than one optimal mapping. For example, another possible optimal mapping in Fig. 2(b) is $\{(p_1, r_3), (p_3, r_5), (p_5, r_8), (p_7, r_7), (p_8, r_1)\}$.

Finding a flow augmenting path from the source to the sink in a flow network is central in most maximum flow algorithms. The improvement lies in the efficient search of flow augmenting paths [13], [12]. For example, in Dinic's algorithm, the shortest augmenting path is always advanced first with the aid of an auxiliary layered network, and hence the computational complexity is bounded by $O(|E|^3)$ for general networks. In our case, the links have unit capacity, and the time complexity is reduced to $O(|V|^{2/3} \cdot |E|)$ [35].

C. Homogeneous MRSIN with Request Priority and Resource Performance

In a homogeneous MRSIN with request priority and resource preference, each request is associated with a priority level, and each resource is assigned a preference value. Many application-dependent attributes, such as workload, execution speed, utilization, and capability, can be encoded into request priorities and resource preferences. The objective of resource scheduling here is to maximize the number of resources allocated, while requests of higher priority are to be allocated and resources of higher preference are to be chosen. However, it is not necessary for requests and resources to be allocated in order of their priorities and preferences. The allocation of a resource to a request may be blocked by requests of higher priority, and the resource may be allocated to a request of lower priority. A similar argument applies to resources.

With respect to a flow network, the priority of a request can be considered as the cost of carrying a flow through the path associated with this request. Likewise, the preference of a resource can be considered similarly. As a result, the request–resource mapping problem in the class of MRSIN's can be transformed into the minimum cost flow problem, which seeks a flow assignment to minimize the total cost of flows in the network.

Consider a flow network $G(V, E, s, t, c, w)$, in which $w(e)$, the cost per unit flow, is associated with arc $e \in E$. In the minimum cost flow problem, a legal s–t flow assignment is sought that allows a given amount of flow F_0 to be circulated from source to sink with the minimum cost. The objective is to determine the set of least expensive s–t paths through which the fixed amount of flow F_0 can be advanced. The constraints in this problem are the same as those in the maximum flow problem. The problem may be defined in a linear programming formulation.

Minimum Cost Flow Problem:
Minimize $\Sigma_{e\in E}\, w(e)f(e)$
subject to

$$1)\ \sum_{e\in\alpha(v)} f(e) - \sum_{e\in\beta(v)} f(e) = \begin{cases} -F_0 & \text{if } v = s \\ F_0 & \text{if } v = t \\ 0 & \text{otherwise} \end{cases}$$

(flow conservation)

$$2)\ 0 \le f(e) \le c(e) \qquad \text{for all } e \in E$$

(capacity limitation).

In allocating resources, the objective is to find a corresponding flow network whose optimal flow leads to an optimal request–resource mapping. The main idea behind the transformation is to embed priority and preference information into the objective function by proper cost assignments on links. The amount of flow to be circulated can be considered as the number of requests pending for allocation. However, this amount may exceed the capacity of the flow network or the number of available resources, and additional paths have to be introduced to prevent overflow. A possible transformation is given as follows.

Transformation 2: Generate a flow network $G(V, E, s, t, c, w)$, from a homogeneous MRSIN with request priorities and resource preferences.

(T1) Create node sets P, X, and R for processors, switchboxes, and resources, respectively, and introduce special nodes: source s, sink t, and a *bypass node* u. Let

$$V' = \{s, t, u\} \cup P \cup X \cup R.$$

(T2) Create arc sets S, T, and B as in Step (T2) of Transformation 1. Further, add an arc from the node associated with a processor to the bypass node, and connect the bypass node to the sink. This set of arcs is denoted as L.

$$L = \{(v, u) | v \in P\} \cup \{(u, t)\}.$$

Define

$$E' = S \cup T \cup B \cup L.$$

(T3) Define capacity function c as in Step (T3) of Transformation 1. In addition, define

$$c(e)_{e\in L} = \begin{cases} 1 & e \ne (u, t) \\ |\alpha(u)| & e = (u, t). \end{cases}$$

(T4) Define cost function w that represents the cost of advancing one unit of flow through a link as follows.

$$w(e) = \begin{cases} 0 & \text{for } e \in B \\ \max\,(y_{\max}+1,\ q_{\max}+1) & \text{for } e \in L \\ y_{\max} - y_p & \text{for } e \in S,\ p \in P \\ q_{\max} - q_w & \text{for } e \in T,\ w \in R \end{cases}$$

where $y_{\max}$ is the highest priority level, y_p is the priority of request from processor p, $q_{\max}$ is the highest preference level, and q_w is the preference of resource w. Note that any cost function that is inversely related to priorities and preferences can be used, for $e \in S$ and $e \in T$.

(T5) Create arc set E and node set V as in Step (T4) of Transformation 1.

(T6) Set the total flow F_0 to the number of requests.

As an example, in the MRSIN in Fig. 5(a), each request is attributed a priority level, and an available resource is given a preference value. The preference and priority levels range from 1 to 10. A minimum cost flow network obtained from Transformation 2 is shown in Fig. 5(b).

The following theorem proves the correctness of Transformation 2.

Theorem 3: The optimal request–resource mapping on a homogeneous MRSIN with request priority and resource preference can be derived from the minimum cost integral flow of the flow network obtained by Transformation 2.

Proof: It is easy to verify that a feasible flow always exists since one can always push the required amount of flow F_0 through the bypass node u. A flow passing through the bypass node means that the associated request it not allocated. Thus, minimizing the cost of a flow assignment is equivalent to assigning as much flow as possible to the part of the flow network other than the bypass node. This is achievable if the minimum cost flow assignment minimizes the amount of cost flow through the bypass node. The theorem can be proved by contradiction. Assume that the minimum cost flow assignment does not define the maximum resource allocation, then there exists an s–t path such that the bypass node u is not on this path, and the path is not saturated such that at least one unit of flow can be advanced through it. The additional flow that could have passed through this s–t path will pass through the bypass node u. According to the cost function w defined in Transformation 2, the cost of advancing flow through such an s–t path is less than that of advancing the same amount of flow through a path passing through node u. The total cost could be reduced if more flow is pushed through this s–t path instead of passing through the bypass node u. The existence of such a path implies that the original assignment is not minimum, which contradicts the assumption. □

Edmonds and Karp have developed a scaled out-of-kilter algorithm to obtain the minimum cost flow of a general flow network in polynomial time [18], [13]. For a flow network of 0–1 capacity, the time complexity is bounded by $O(|V| \cdot |E|^2)$. Furthermore, in the minimum cost flow assignment obtained, the flow assigned to a link is integral if the links have integral capacities. Thus, the optimal request–resource mapping of homogeneous MRSIN with request priorities and resource preferences can be obtained efficiently.

As an example, applying the minimum cost flow algorithm on the flow network in Fig. 5(b) results in the request–resource mapping $\{(p_3, r_5), (p_5, r_1), (p_8, r_7)\}$. The selected paths are shown as bold lines in Fig. 5(b). Note that the minimum cost flow obtained may not be unique, although alternative mappings will not improve the cost of allocation.

D. Optimal Resource Scheduling in Heterogeneous MRSIN

A heterogeneous MRSIN consists of multiple types of resources, and a processor may generate a request of a given

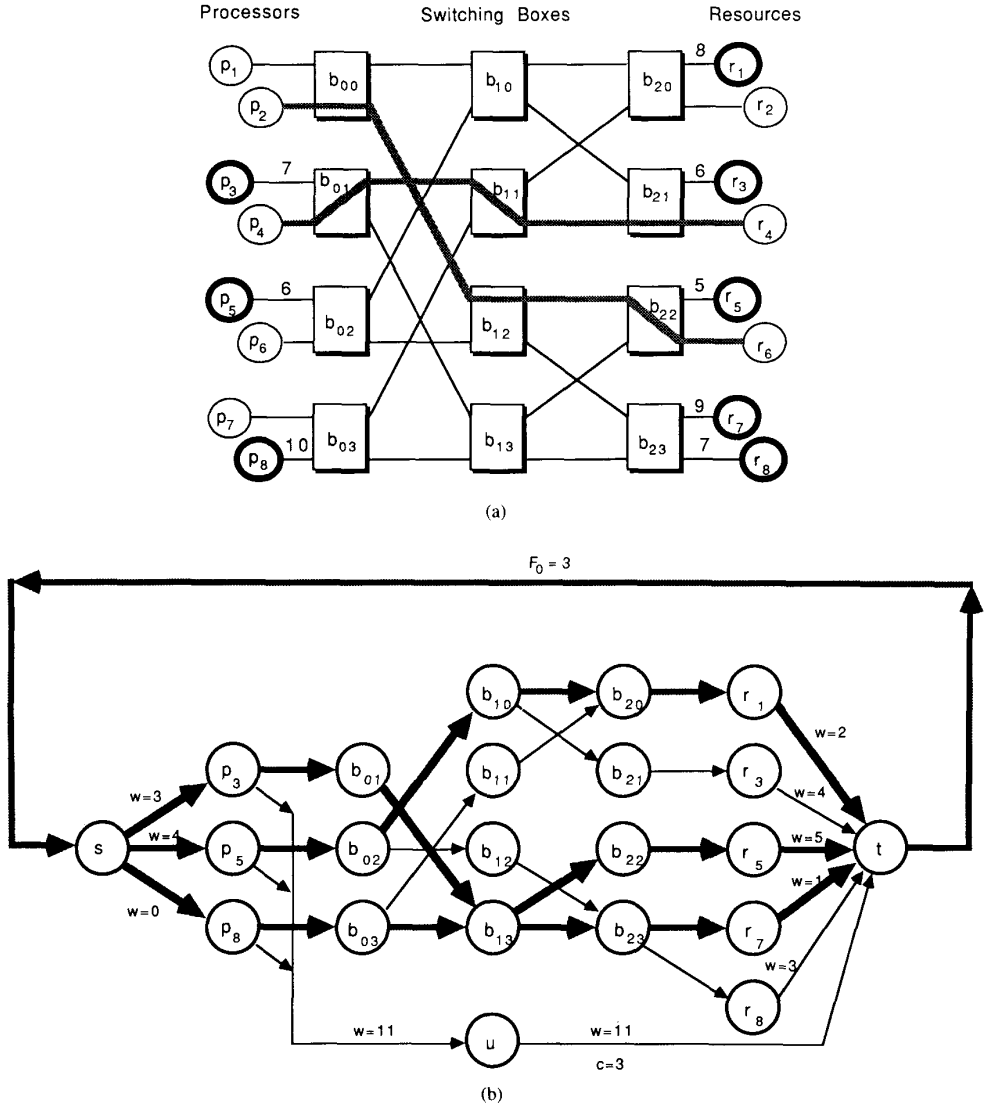


Fig. 5. Example to illustrate Transformation 2. (a) An MRSIN with request priority and resource preference (highest priority is 10; highest preference is 10; thick shaded paths in the network are already occupied; processors p_3, p_5, and p_8 are making requests; resources r_1, r_3, r_5, r_7, and r_8 are available). (b) The flow network transformed from the MRSIN in Fig. 5(a) using Transformation 2 (nonzero flows assigned by the out-of-kilter algorithm are shown as thick dark lines in the figure; all arcs have unit capacity; cost of arc is zero except where indicated).

type of resource. Such an MRSIN is equivalent to a flow network carrying different types of commodities. A multicommodity flow network has multiple source-sink pairs, each of which is associated with one type of commodity. A flow coming out of a source of a given commodity can only be absorbed by the sink of the same type of commodity. Flows of different commodities may share a link as long as the total flow does not exceed the capacity of the link.

For a flow network with k types of commodities, there are k source-sink pairs, (s^i, t^i), for $i = 1$ to k. Let F^i be the flow of the ith commodity. The search for the maximum flow can be formulated as a linear programming problem [1].

Multicommodity Maximum Flow Problem:

Maximize $\sum_{i=1}^{k} F^i$

subject to

$$1)\ \sum_{e\in\alpha(v)} f^i(e) - \sum_{e\in\beta(v)} f^i(e) = \begin{cases} -F^i & v = s^i \\ F^i & v = t^i \\ 0 & \text{otherwise} \end{cases}$$

(flow conservation) for $i = 1, \cdots, k$

$$2)\ 0 \le \sum_{i=1}^{k} f^i(e) \le c(e) \quad \text{for all } e \in E$$

(capacity limitation).

A multicommodity flow network may be visualized as the superposition of k single-commodity flow networks. Each layer in the superposition represents a single-commodity flow.

To obtain the optimal request–resource mapping in a heterogeneous MRSIN without priority and preference, a transformation similar to Transformation 1 can be applied to obtain a single-commodity flow network for each type of resource, and the single-commodity flow networks are superposed to form a multicommodity flow network.

The optimal mapping for a heterogeneous MRSIN with request priorities and resource preferences can be obtained by transforming the problem into the multicommodity minimum cost flow problem. Let $w^i(e)$ be the cost per unit flow for the ith commodity on edge e, and $f^i(e)$ be the corresponding flow. The problem can be formulated as follows.

Multicommodity Minimum Cost Flow Problem:

Minimize $\Sigma_{i=1}^{k} \Sigma_{e \in E} w^i(e) f^i(e)$

subject to

$$1) \sum_{e \in \alpha(v)} f^i(e) - \sum_{e \in \beta(v)} f^i(e) = \begin{cases} -F_0^i & v = s^i \\ F_0^i & v = t^i \\ 0 & \text{otherwise} \end{cases}$$

(flow conservation) for $i = 1, \cdots, k$

$$2) \ 0 \le \sum_{i=1}^{k} f^i(e) \le c(e) \qquad \text{for all } e \in E$$

(capacity limitation).

The equivalent flow network consists of k source–sink pairs and k bypass nodes, where k is the number of types of requested resources. Similar to the case before, this flow network may be regarded as the superposition of k single-commodity flow networks, and Transformation 2 can be applied to each of them.

The problem of finding the maximum integral flow in a general multicommodity flow network has been shown to be NP-hard. Fortunately, interconnection networks of restricted topology have transformations that belong to a class of multicommodity flow networks in which the optimal flow values are always integral [14]. For this class of flow networks, the integral multicommodity optimal flows can be obtained efficiently by the *Simplex Method,* which has been shown empirically to be a linear time algorithm [31].

IV. Architecture of MRSIN to Support Optimal Scheduling

The optimal scheduling algorithms described in Section III can be supported by various architectures. An efficient design is needed to avoid an intolerable overhead.

In a conventional interconnection network using address mapping, resource binding is done at the requesting processor. To obtain an optimal mapping, a requesting processor has to know the status of the network, availability of resources, and all requests generated by other processors. This information is very costly to obtain because it changes dynamically. In practice, resource binding is done without global information. A resource mapping so obtained may be suboptimal and may lead to heavy resource contention and severe circuit blockage. In contrast, it is less costly to maintain global status information in the network. Thus, by carrying out resource binding in the RSIN, the overhead of obtaining an optimal resource mapping can be reduced.

Two architectures of an MRSIN for carrying out the optimal resource scheduling algorithms have been studied. In the first approach, a dedicated monitor is responsible for resource scheduling (see Fig. 6). It maintains the status of the interconnection network and resources. The monitor enters a scheduling cycle when there are pending requests. Requests received or resources released during a scheduling cycle will not be processed until the next cycle. In a scheduling cycle, a flow network is generated according to the status of the network. The optimal request–resource mapping is derived by the monitor using a flow algorithm implemented in software. The monitor then sends an acknowledgment to each requesting processor that has been allocated a resource, notifies resources that are allocated, and establishes paths in the network. The implementation is sequential, and the overhead is measured by the number of instructions executed in the algorithm.

A distributed architecture, on the other hand, distributes the scheduling intelligence in the switchboxes of the interconnection network. Optimal scheduling is achieved through cooperation among processes in the switchboxes. No transformation to a network flow problem is necessary because the network flow algorithm is carried out in a distributed fashion in the switchboxes. The complexity of the process in each switchbox is central to the design of the distributed architecture. Our previous study shows that the maximum flow algorithm for homogeneous MRSIN without priority and preference can be efficiently implemented in a distributed fashion [25]. For systems with heterogeneous resources or with priorities and preferences, there is no significant advantage of a distributed implementation over a monitor architecture except for reasons such as fault tolerance and modularity.

In the following sections, we describe a distributed realization of Dinic's maximum flow algorithm to obtain the optimal request–resource mapping.

A. Dinic's Maximum Flow Algorithm

Dinic's algorithm is based on the flow-augmentation method described in Section III-B. It improves over Ford and Fulkerson's algorithm by advancing flow through the shortest augmenting path, which can be found from a *layered network* derived from the original flow graph. A flow chart summarizing Dinic's algorithm is shown in Fig. 7. it comprises two alternating phases. In the first phase, a layered network is constructed, while in the second phase, an increment to the flow assignment is determined by finding the maximal flow in the layered network. The algorithm alternates between these two phases until no more flow can be augmented.

In the layered network, nodes of the original flow network are organized into layers. The first layer consists of the source node(s) of the network, and the remaining layers are constructed iteratively. A layer consists of nodes that are not included in the previous layers and have either an unsaturated arc or an arc with nonzero flow originating from any node in

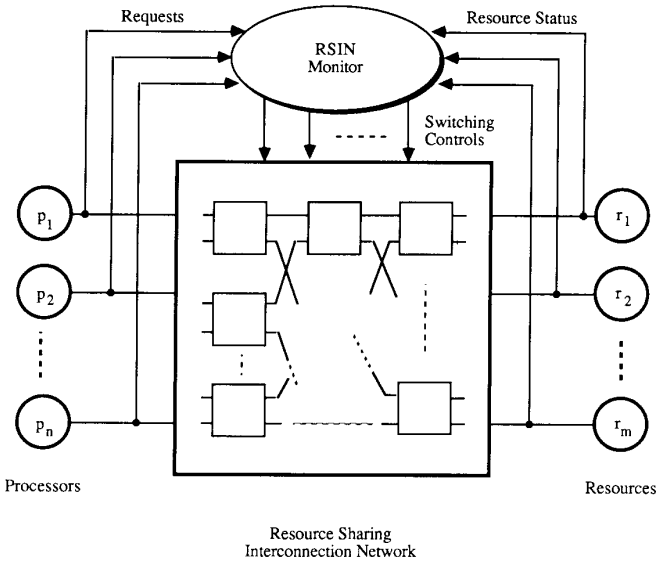


Fig. 6. A monitor architecture to carry out optimal resource scheduling in an RSIN.

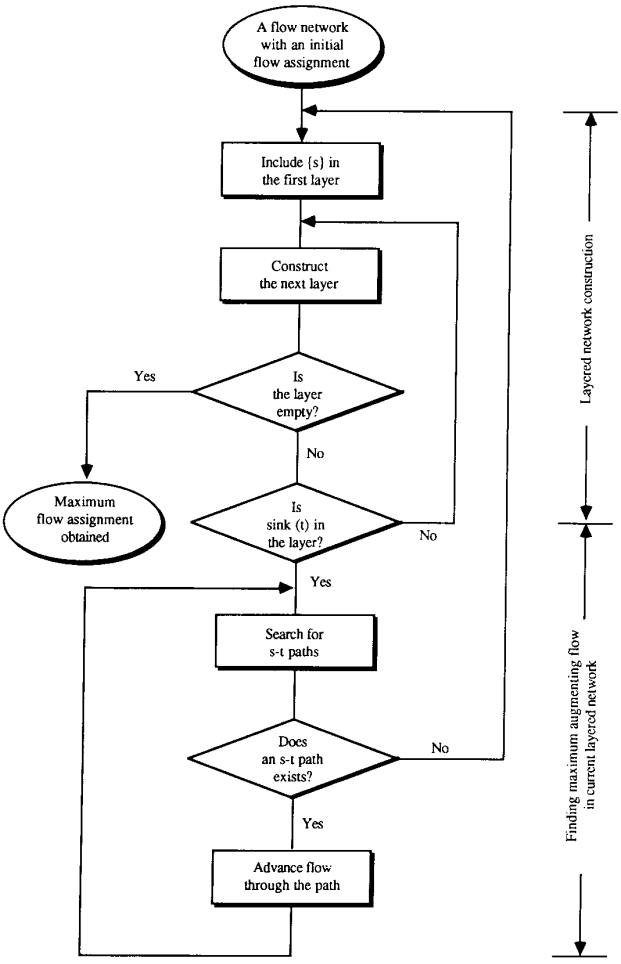


Fig. 7. Control flow of Dinic's algorithm.

the layer before it. These two types of arcs, called *useful links*, are transformed to arcs in the layered network. Depending on the direction of the associated useful link, its capacity in the layered network can be either the remaining capacity or its current flow. As a result, nodes in a layered network are arranged into disjoint subsets, $V_0, V_1, \cdots, V_0$, such that no arc points from V_j to V_i for $i \leq j$.

A legal flow in a layered network is said to be *maximal* if every (s, t)-directed path in the layered network is saturated. Note that it is not necessary to find the maximum flow of the layered network. Finding a maximal flow is sufficient since the objective of the layered network is to obtain a net increase to the total flow assignment in each iteration. Moreover, computing the maximal flow is easier than computing the maximum flow. In Dinic's algorithm, the maximal flow is obtained by a depth-first search.

Since the amount of flow that can be advanced through an arc in the layered network is the net increase of flow to the associated arc in the original network, the maximal flow obtained in the layered network is a net increment to the existing flow. Moreover, since the maximum flow of a flow network is finite, it can be obtained in a finite number of iterations in constructing the layered network.

An example illustrating the construction of a layered network is shown in Fig. 8. Fig. 8(a) is a flow network associated with an MRSIN in which three processors, p_1, p_2, and p_4, are making requests and three resources, r_1, r_3, and r_4, are available. The flow assignment shown by darkened arcs in Fig. 8(a) results in a mapping such that p_1 is mapped to r_4 and p_4 is mapped to r_1. The request generated by p_2 is blocked. Fig. 8(b) is a layered network constructed from the flow network in Fig. 8(a). The layered network shows that there is a flow augmenting path from p_2 to r_3. This path includes the arc leading from node 6 to node 5, which is associated with the arc

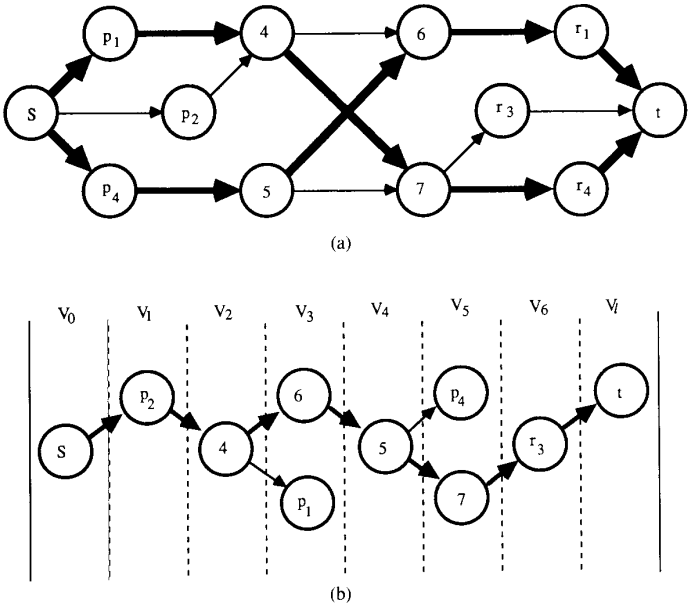


Fig. 8. An illustration of a layered-network construction (all arcs have unit capacity). (a) A flow network (transformed from a 4 × 4 MRSIN) in which flow is advanced through the two dashed paths. (b) The layered network derived from the flow network in (a). The darkened (s, t)-path is a flow augmenting path.

leading from node 5 to node 6 in the original network [see Fig. 8(a)]. It indicates that the flow leading from node 5 to node 6 can be cancelled. New flow should be routed through two other arcs: one from node 4 to node 6, the other from node 5 to node 7. This flow augmenting path shows that all three resources can be allocated if p_4 is reallocated to r_3 and p_2 is reallocated to r_1.

B. A Distributed Architecture for Homogeneous MRSIN without Priority

A distributed MRSIN embedded in an 8 × 8 Omega network is shown in Fig. 9. In this architecture, a processor is connected to the network through a request server (RQ), a resource is monitored by a resource server (RS), and each switchbox is controlled by an independent process (NS). A common status bus connects these components together. The scheduling intelligence is distributed in the switchboxes of the MRSIN. In each switchbox, there is an autonomous process implemented as a finite-state machine. The process communicates with other processes via direct links. Processes are synchronized by exchanging status information via the status bus to cooperatively realize a distributed Dinic's algorithm. In general, the design of a distributed Dinic's algorithm is not trivial. However, it can be greatly simplified in the MRSIN due to the property of unit flow capacity.

A scheduling cycle begins when there are pending requests and ready resources. A request generated in the middle of a scheduling cycle has to wait until the next cycle. A scheduling cycle consists of many iterations. In each iteration, protocols governing process interactions are carried out to perform layered-network construction and maximal flow assignment. Distributed data structures are also needed for representing the

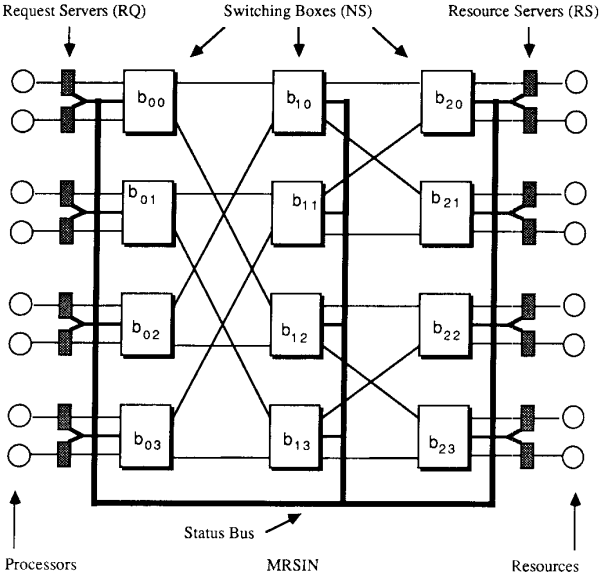


Fig. 9. A distributed MRSIN embedded in an 8 × 8 Omega network.

layered network, flow assignment, and other intermediate results.

In the proposed architecture, flow augmentation is done by token propagations. Tokens are propagated in the network to iteratively search for flow augmenting paths and rearrange resource mappings until the optimum is obtained. Because each link is of unit capacity and is not to be shared by multiple allocated paths, a token can simply be represented by a signal traversing from one element to another. It carries neither identification nor other information. Its type is determined by the function being performed. With such kinds of tokens,

scheduling speed is limited only by the switching delay of logic gates. The layered network obtained in each iteration can be represented implicitly by recording the token propagation status in a bit array associated with each port. The bit pattern for the token propagation status is referenced as a *port marking* in the sequel. A flow augmenting path in the layered network can be identified by token propagation markings along the path.

During a flow augmentation process, a free link may become "*registered*" if the two ports associated with it are marked. On the contrary, a registered link may become free if the flow assigned to it is cancelled and port markings are erased. A scheduling cycle consists of a request-token-propagation phase for constructing a layered network, a resource-token-propagation phase for finding the maximal flow of the layered network, and a path registration phase for registering paths associated with the maximal flow. At the end of a scheduling cycle, any surviving registered link becomes "*occupied*;" that is, it will be used in an allocated path. A request will be *bonded* to a free resource when a request token has successfully propagated to the resource in the request-token-propagation phase, and a resource token has successfully propagated to the requesting processor in the resource-token-propagation phase. This corresponds to finding a maximal flow of the layered network in Dinic's algorithm.

Since a token is nothing but a propagating signal, Dinic's algorithm is in fact realized by distributing tokens in the network. Token propagation rules for carrying out each function and the mechanism for synchronizing token propagations are described next.

1) Token Propagation for Layered Network Construction: At the beginning of the request-token-propagation phase, each RQ with unbonded pending request sends a token to its output port, which is connected to an NS in the first stage of the MRSIN.

For every NS receiving a request token through its input port, it duplicates the token and sends one to each of its free output ports and registered input ports. Note that an input port may be registered in previous iterations of the request-token-propagation and resource-token-propagation phases. All receiving and sending ports in an NS receiving request tokens are *marked.*

Token propagation is clocked, i.e., each token traverses across one link in a clock period. Propagation direction depends on the status of the link over which the token is traversing. The token traverses forward if the link is free, and backward if the link is registered. Accordingly, an NS may receive a request token either from its free input ports or from its registered output ports. Tokens may arrive at an NS during any clock period. Only the first batch is considered. All of the rest are discarded; that is, subsequent token arrivals will not cause a port to be marked if the NS had received tokens previously. If a token goes backward to a bounded RQ, it is absorbed by the RQ. This phase comes to an end when one or more RS's has received a token. In the following theorem, we show that the layered network will be obtained correctly by the propagation of request tokens.

Theorem 4: A layered network can be constructed correctly by propagating tokens according to the rules described above.

Proof: Since request tokens are only generated by RQ's in the first clock period of a request-token-propagation phase, the RQ's making requests in this phase can only be included in the first layer. A virtual source node can be considered to be connected to every RQ in the first layer. Next, we would like to show that, given a layer, the next layer can be determined uniquely by token propagations. In each clock period, tokens are distributed from the current layer to unmarked free output ports and unmarked registered input ports, and they traverse exactly one link. Thus, only those elements that are directly connected to the current layer may receive a token. By eliminating those that had received tokens before, we obtain a set of elements corresponding to those nodes in the next layer of the layered network. If no RS is included in the next layer, then these nodes are responsible for token propagation in the next clock period. If an RS does appear in the layer, all tokens stop propagating, and a virtual sink node is implicitly generated in the last layer, although no actual token propagation is necessary to construct this virtual layer. In summary, given a layer, the next layer can be constructed correctly by request-token propagations. By induction, the theorem is proved. □

2) Token Propagation to Find Maximal Flow: The RS's receiving a token in the request-token-propagation phase represent resources that have not been allocated to any request so far and can possibly be allocated with some rearrangement of resource mapping. Since the rearrangement is done by flow augmentation in Dinic's algorithm, a new phase of token propagation is started to find the maximal flow in the layered network after the layered network has been constructed.

In this phase, each RS appearing in the last layer sends a token (called resource token for convenience) back to the layered network hoping to find a matching RQ. In effect, the token serves as a positive acknowledgment to the request tokens from the RQ's. The token traverses across one link per clock period, and an NS expects to receive tokens only from those ports to which a request token was sent, that is, those ports that were marked. A resource token is not duplicated by an NS since an RS can be assigned to only one RQ. When multiple resource tokens arrive at a point where a request token was duplicated, only one of them is allowed to go through the link from which the request token was received. The rest have to backtrack to find alternative paths. If backtracking causes a token to return to its originating RS, then the token is discarded. This means that the RS cannot find a matching RQ in this iteration. The marking of a port is cleared whenever a resource token backtracks through the port. This prevents subsequent attempts of fruitless backtracking. The number of propagating resource tokens is reduced as tokens are received by RQ's or backtracked to RS's. Resource token propagation stops when the number of propagating resource tokens is reduced to zero. It is easy to show that the set of paths explored by successful resource token propagations represents the maximal flow of the layered network.

The maximal flow of the layered network is the flow to be augmented to the original flow assignment in the current iteration. Flow augmentation has to be done before the next iteration starts. To achieve this, the MRSIN enters a third phase. All it needs in this phase is to change the state of those

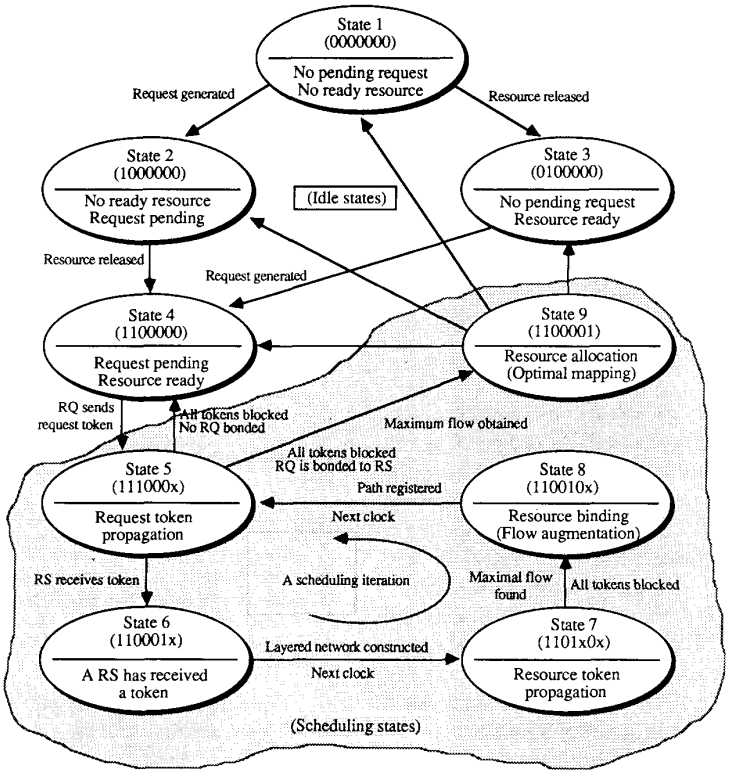


Fig. 10. A state transition diagram of distributed MRSIN with a status bus.

paths associated with the maximal flow to being "*registered.*" These paths are readily known at the end of the resource-token-propagation phase since they are the paths through which resource tokens successfully propagated to RQ's. The RQ's that received a resource token are bonded to the corresponding RS's. Their binding status bits are set to 1.

3) Synchronization via Status Broadcasting: To ensure that each element of the MRSIN applies the right rule to propagate tokens, phase transitions must be strictly synchronized. A synchronization scheme based on message passing is too slow to match the speed of the token-propagation scheme. Although a broadcast bus can greatly simplify synchronization, especially when processes are located in close proximity [25], the cost of maintaining the status of processes received from the bus is too high. In this section, we propose the design of an efficient status bus to address this problem.

Instead of being used as a transmission media for sending messages, the status bus is in fact a specialized global "memory" device that can be accessed concurrently. Each bit of the bus is associated with an event that reflects the collective status of a subset of processes. To realize such a bus, each process maintains its own status in a single-bit register, and the output of the register is connected to a wire-OR logic gate. The output of the gate is then connected to the corresponding bit of the status bus. Accordingly, the status observable from the bus is the logical OR of the status of associated processes.

To determine what are the necessary events that require synchronization, possible phase transitions in an MRSIN are examined, and the results are summarized in a state transition diagram in Fig. 10. In an idle period, the MRSIN is in one of the states in which either no pending request or no ready resource exists, or no ready resource can be allocated to pending requests. The MRSIN may enter a scheduling period when there are requests pending and resources ready. However, to avoid repeated attempts of allocating blocked resources (i.e., the case of cycling between states 4 and 5 in Fig. 10) and to improve the scheduling efficiency, the MRSIN may choose to wait for more requests to arrive and more resources to become available before entering a scheduling cycle. Each scheduling iteration consists of five states. To conclude a scheduling cycle, the MRSIN enters the allocation state in which registered paths are changed to being "bonded."

Based on the state transition diagram, seven events that require synchronization are identified. We have chosen to implement the status bus with seven bits. The definition of these events and their associated processes are shown in Table I. Since these events are observable on the status bus, an occurrence of a state transition can be disseminated instantly, and processes can react to the new state immediately. For example, when an event vector (111000x) is observed on the bus, an NS knows that the MRSIN is in a request-token-propagation phase. (Note that the "DON'T CARE" symbol "x" in the state vector means that the designated bit can be 0 or 1.) It can determine immediately which rule to apply whenever it receives a token. When propagating a token, elements on the propagation path turn on their E_3 status bits for one clock period in turn. This operation keeps bit 3 of the status bus on whenever there are tokens propagating. The MRSIN moves to

TABLE I
EVENT DEFINITIONS AND ASSOCIATED PROCESSES

Event	Definition	Associated Processes	Bit Number of Status Bus
E_1	Request pending	RQs	6 (MSB)
E_2	Resource Ready	RSs	5
E_3	Request token propagation	RQs, NSs	4
E_4	Resource token propagation	RSs, NSs	3
E_5	A RQ has received a token	RQs	2
E_6	A RS has received a token	RSs	1
E_7	A RQ is bonded to a RS	RQs	0 (LSB)

a new state ($111001x$) when an RS sets E_6 to 1 upon receiving a token. The MRSIN will stay in this state for one clock period to allow all tokens to come to a stop. At the beginning of the next clock cycle, E_6 will be turned off, and E_4 will be turned on. The MRSIN moves into state ($110100x$) representing a resource-token-propagation phase. The next transition will bring the MRSIN into a path registration state ($110110x$). E_4 and E_5 will be turned off after one clock period. Finally the MRSIN returns to state ($111000x$) for a new iteration.

Since a token is simply a signal, token propagation rules can be expressed in terms of Boolean functions. A distributed process at an NS, RQ, or RS does nothing but distribute the token according to the global status and local conditions. It can be realized easily by a finite-state machine, the design of which can be found elsewhere [25]. The design has a very low gate count and a very short token propagation delay.

Overall, the token-propagation architecture has two factors that contribute to a significant speedup as compared to a monitor architecture: 1) the augmenting paths are searched in parallel, and 2) the time complexity is measured in gate delays instead of instruction cycles. As a result, the scheduling algorithm will run at a much higher speed than a software implementation of the network flow algorithm.

V. CONCLUSIONS

An RSIN is suitable to support resource sharing in multiprocessors. Optimal request-resource mapping in an RSIN with homogeneous resources and requests of equal priority is obtained by maximizing the number of communication paths that interconnect pairs of processors and resources. In this paper, we have transformed various request-resource mapping problems into network flow problems for which efficient algorithms exist. Table II summarizes the results we have obtained. The proposed method is independent of the interconnection structure and is applicable to any network configuration in which the requesting processors and free resources can be partitioned into two disjoint subsets. In particular, the method is applicable to networks with multiple paths between source-destination pairs, such as the data manipulator [15], augmented data manipulator [33], and gamma network [36]. The resource utilization, however, will depend on the network configuration, the resources available, the arrangement of the various types of resources, and the arrangement of the requesting processors.

TABLE II
SUMMARY OF OPTIMAL RESOURCE SCHEDULING SCHEMES FOR RESOURCE SHARING INTERCONNECTION NETWORKS

Scheduling Discipline		*Homogeneous Resources*		*Heterogeneous Resources*	
		No Priority & Preference	*Priority & Preference*	*Restricted Topology*	*General Topology*
Optimal Scheduling	*Equivalent Flow Problem*	Max.-Flow	Min.-Cost Circulation	Real Multi-Commodity	Integer Multi-Commodity
	Algorithms	Ford-Fulkerson, Dinic	Out-of-Kilter	Linear Programming	NP-hard
Architecture		Distributed	Monitor (Centralized)		
Implementation		Synchronized by Status Bus; Communicate via Token Propagation	Software		

REFERENCES

[1] A. A. Assad, "Multicommodity network flows—A survey," *Networks,* vol. 8, pp. 37-91, 1978.

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

[3] K. E. Batcher, "The Flip network in STARAN," in *Proc. Int. Conf. Parallel Processing,* 1976, pp. 65-71.

[4] ——, "Design of a massively parallel processor," *IEEE Trans. Comput.,* vol. C-29, pp. 836-840, Sept. 1980.

[5] V. Benes, *Mathematical Theory of Connecting Networks.* New York: Academic, 1965.

[6] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications.* New York: North-Holland, 1981.

[7] F. A. Briggs, M. Dubois, and K. Hwang, "Throughput analysis and configuration design of a shared-resource multiprocessor system: PUMPS," in *Proc. 8th Annu. Symp. Comput. Architecture,* 1981, pp. 67-79.

[8] F. A. Briggs, K. S. Fu, K. Hwang, and B. W. Wah, "PUMPS architecture for pattern analysis and image database management," *IEEE Trans. Comput.,* vol. C-31, pp. 969-983, Oct. 1982.

[9] C. Clos, "A study of nonblocking switching networks," *Bell Syst. Tech. J.,* vol. 32, pp. 406-424, 1953.

[10] J. B. Dennis, "Data flow supercomputers," *IEEE Computer,* vol. 13, pp. 48-56, Nov. 1980.

[11] D. M. Dias and J. R. Jump, "Analysis and simulation of buffered delta networks," *IEEE Trans. Comput.,* vol. C-30, pp. 273-282, 1981.

[12] E. A. Dinic, "Algorithm for solution of a problem of maximal flow in a network with power estimation," *Soviet Math. Dokl.,* vol. 11, pp. 1277-1280, 1970.

[13] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM,* vol. 19, pp. 248-264, Apr. 1972.

[14] J. R. Evans and J. J. Jarvis, "Network topology and integral multicommodity flow problems," *Networks,* vol. 18, pp. 107-119, 1978.

[15] T. Y. Feng, "Data manipulating functions in parallel processors and their implications," *IEEE Trans. Comput.,* vol. C-23, pp. 309-318, 1974.

[16] ——, "A survey of interconnection networks," *Computer,* pp. 12-27, Dec. 1981.

[17] L. R. Ford and D. R. Fulkerson, *Flow in Networks.* Princeton, NJ: Princeton University Press, 1962.

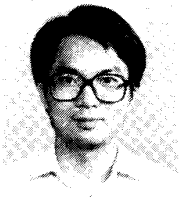
[18] D. R. Fulkerson, "An out-of-kilter method for minimum cost flow problems," *SIAM J. Comput.,* vol. 9, pp. 18-27, 1961.

[19] F. Fung and H. Torng, "On the analysis of memory conflicts and bus contentions in a multiple-microprocessor system," *IEEE Trans. Comput.,* vol. C-28, pp. 28-37, Jan. 1979.

[20] L. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," in *Proc. 1st Annu. Comput. Architecture Conf.,* Dec. 1973, pp. 21-28.

[21] B. Golden, M. Ball, and L. Bodin, "Current and future research directions in network optimization," *Comput. Oper. Res.,* vol. 8, pp. 71-81, 1981.

[22] A. Hicks, "Resource scheduling on interconnection networks," M.S. thesis, Purdue Univ., West Lafayette, IN, Aug. 1982.
[23] W. D. Hillis, "The connection machine: A computer architecture based on cellular automata," *Physica,* pp. 213-228, 1984.
[24] J. Y. Juang and B. W. Wah, "Optimal scheduling algorithms for multistage resource sharing interconnection networks," in *Proc. 8th Int. Comput. Software Appl. Conf.,* Nov. 1984, pp. 217-225.
[25] J. Y. Juang, "Resource allocation in computer networks," Ph.D. dissertation, Purdue Univ., West Lafayette, IN, Aug. 1985.
[26] D. J. Kuck, "ILLIAC IV software and application programming," *IEEE Trans. Comput.,* vol. C-17, pp. 746-757, Aug. 1968.
[27] D. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.,* vol. C-24, pp. 215-255, Dec. 1975.
[28] M. Lee and C.-L. Wu, "Performance analysis of circuit switching baseline interconnection networks," in *Proc. 11th Annu. Int. Symp. Comput. Architecture,* 1984, pp. 82-90.
[29] M. A. Marsan and F. Gregoretti, "Memory interference models for a multimicroprocessor system with a shared bus and a single external common memory," *Microprocessing Microprogramming—EUROMICO J.,* vol. 7, Feb. 1982.
[30] M. A. Marsan and M. Gerla, "Markov models for multiple bus multiprocessor systems," *IEEE Trans. Comput.,* vol. C-31, pp. 239-248, Mar. 1982.
[31] E. H. McCall, "Performance results of the simplex algorithm for a set of real-word linear programming models," *Commun. ACM,* vol. 25, no. 3, pp. 207-213, Mar. 1982.
[32] W. C. McDonald and J. M. Williams, "The advanced data processing test bed," in *Proc. COMPSAC,* Mar. 1978, pp. 346-351.
[33] R. J. McMillen and H. J. Siegel, "Routing schemes for the augmented data manipulator network in an MIMD system," *IEEE Trans. Comput.,* vol. C-31, pp. 1202-1214, Dec. 1982.
[34] S. M. Ornstein *et al.*, "Pluribus—A reliable multiprocessor," in *Proc. Nat. Comput. Conf.,* 1975, pp. 551-559, AFIPS Press.
[35] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity.* Englewood Cliffs, NJ: Prentice-Hall, 1982.
[36] D. S. Parker and C. S. Raghavendra, "The gamma network: A multiprocessor interconnection network with redundant paths," in *Proc. 9th Annu. Symp. Comput. Architecture,* 1982, pp. 73-80.
[37] J. H. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Trans. Comput.,* vol. C-20, pp. 771-780, Oct. 1981.
[38] M. C. Pease, "The indirect binary *n*-binary *n*-cube microprocessor array," *IEEE Trans. Comput.,* vol. C-26, pp. 458-473, May 1977.
[39] B. D. Rathi, A. R. Tripathi, and G. J. Lipovski, "Hardwired resource allocators for reconfigurable architectures," in *Proc. Int. Conf. Parallel Processing,* Aug. 1980, pp. 109-117.
[40] M. C. Sejnowski *et al.*, "Overview of the Texas Reconfigurable Array Computer," in *Proc. Nat. Comput. Conf.,* vol. 49, 1980. pp. 631-642.
[41] H. J. Siegel and R. J. McMillen, "The multistage cube: A versatile interconnection network," *IEEE Computer,* vol. 14, pp. 65-76, Dec. 1981.
[42] H. J. Siegel and R. J. McMillen, "Using the augmented data manipulator network in PASM," *IEEE Computer,* vol. 14, pp. 25-33, Feb. 1981.
[43] H. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.,* vol. C-20, pp. 153-161, Feb. 1971.
[44] B. W. Wah and A. Hicks, "Distributed scheduling of resources on interconnection networks," in *Proc. Nat. Comput. Conf.,* 1982, pp. 697-709.
[45] ——, "A comparative study of distributed resource sharing on multiprocessors," *IEEE Trans. Comput.,* vol. C-33, pp. 700-711, Aug. 1984.
[46] C. Wu and T. Y. Feng, "On a class of multistage interconnection networks," *IEEE Trans. Comput.,* vol. C-29, pp. 694-702, Aug. 1980.
[47] W. A. Wulf and C. G. Bell, "C.mmp—A multi-mini processor," in *Proc. Fall Joint Comput. Conf.,* 1972, pp. 765-777.
[48] H. J. Siegel *et al.*, "PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Trans. Comput.,* vol. C-30, pp. 934-947, 1981.



Jie-Yong Juang (S'82-M'85) received the B.S. degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1976, the M.S. degree in computer science from University of Nebraska, Lincoln, in 1981, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, in 1985.

He is an Assistant Professor in the Department of Electrical Engineering and Computer Science at Northwestern University, Evanston, Illinois. His areas of research include computer architecture, parallel processing, distributed processing, fault-tolerant computing, and logic programming.

Benjamin W. Wah (S'74-M'79-SM'85) is an Associate Professor in the Department of Electrical and Computer Engineering and a Research Associate Professor in the Coordinated Science Laboratory of the University of Illinois at Urbana-Champaign, Urbana, IL.

He is currently on leave at the National Science Foundation as a Program Director in the Microelectronic Information Systems Division. His areas of research include computer architecture parallel processing, artificial intelligence, distributed databases, and operating systems.

Dr. Wah is the Associate Editor-in-Chief of the forthcoming IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, and an editor of IEEE TRANSACTIONS ON SOFTWARE ENGINEERING and *Journal of Parallel and Distributed Computing* and *Information Sciences*. He serves as a member of the Governing Board of the IEEE Computer Society. He also serves as a program evaluator for ABET (computer engineering) and CSAC (computer science).