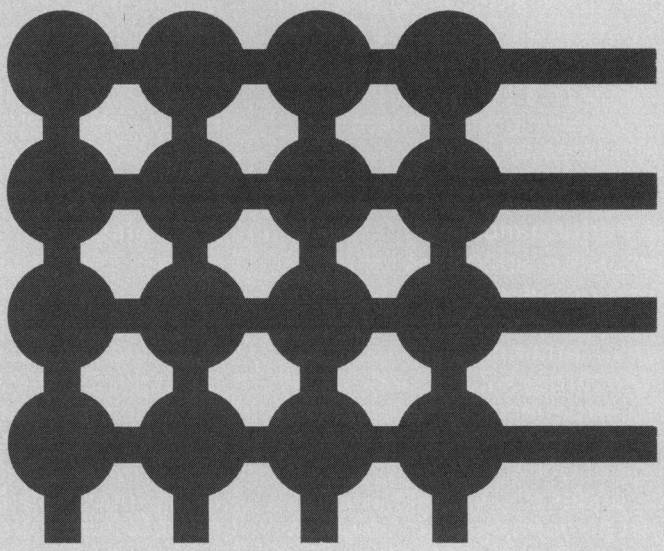
Concurrent processing depends on interconnection networks for communication among processors and memory modules. Various network topologies and switching strategies are covered here.

A Survey of Interconnection Networks



Tse-yun Feng
The Ohio State University

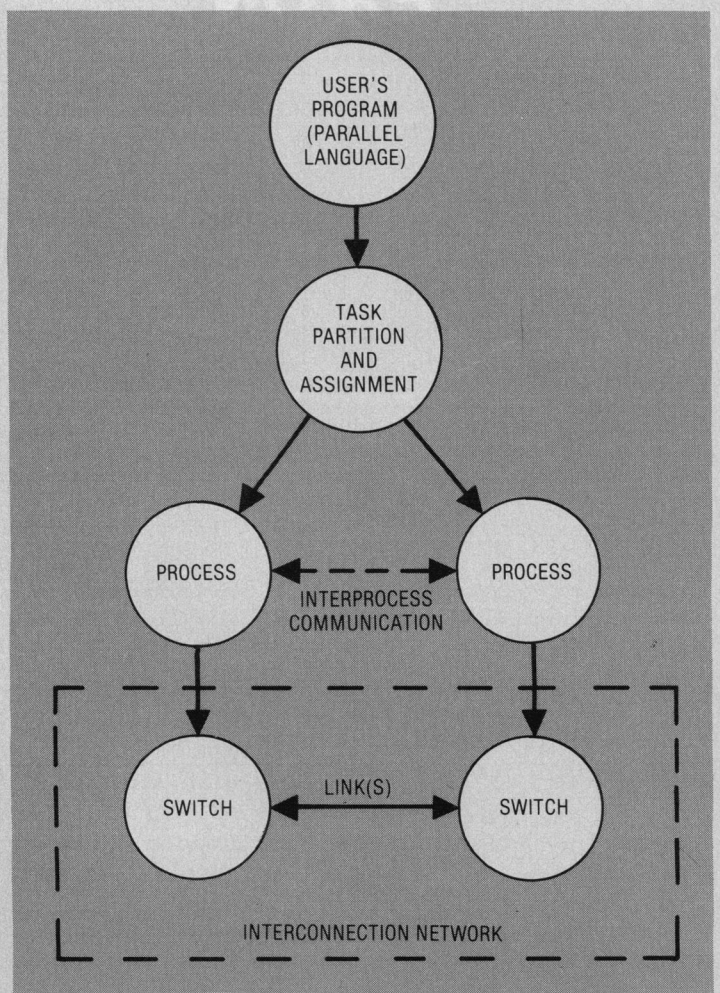


Figure 1. An overview of concurrent processing systems.

Concurrent processing of data items is considered a proper approach for significantly increasing processing speed.[1] In many real-time applications—such as image processing and weather computation, which need an instruction execution rate of more than one billion floating-point instructions per second—concurrent processing is unavoidable. And now, with the advent of LSI technology, it is economically feasible to construct a concurrent processing system by interconnecting hundreds—even thousands—of off-the-shelf processors and memory modules.

A basic concurrent processing system is shown in Figure 1. Processes, generated by compiling and partitioning a user's program, are assigned to individual processors, and an interconnection network implements interprocess communication. A general model of the hardware system is shown in Figure 2. The interconnection network facilitates communication not only among the n processors and the m memory modules but also between the processors and memory modules.

Many interconnection networks have been reviewed in other surveys.[2-9] In this article we consider interconnection networks from a practical design viewpoint. We examine design decisions that are essential in choosing a cost-effective communication network, survey the various topologies and communication protocols, and discuss connection issues related to concurrent processing.

Design decisions

In selecting the architecture of an interconnection network, four design decisions can be identified.[10] They concern operation mode, control strategy, switching method, and network topology.

0018-9162/81/1200-0012$00.75 © 1981 IEEE

Operation mode. Two types of communication can be identified: synchronous and asynchronous. Synchronous communication is needed for processing in which communication paths are established synchronously for either a data manipulating function[11] or a data/instruction broadcast. Asynchronous communication is needed for multiprocessing in which connection requests are issued dynamically. A system may also be designed to facilitate both synchronous and asynchronous processing. Therefore, typical operation modes of interconnection networks can be classified into three categories: synchronous, asnychronous, and combined.

Control strategy. A typical interconnection network consists of a number of switching elements and interconnecting links. Interconnection functions are realized by properly setting control of the switching elements. The control-setting function can be managed by a centralized controller or by the individual switching element. The latter strategy is called distributed control; the first strategy is called centralized control.

Switching methodology. The two major switching methodologies are circuit switching and packet switching. In circuit switching, a physical path is actually established between a source and a destination. In packet switching, data is put in a packet and routed through the interconnection network without establishing a physical connection path. In general, circuit switching is much more suitable for bulk data transmission, and packet switching is more efficient for short data messages. Another option, integrated switching, includes capabilities of both circuit switching and packet switching. Therefore, three switching methodologies can be identified: circuit switching, packet switching, and integrated switching.

Network topology. A network can be depicted by a graph in which nodes represent switching points and edges represent communication links. The topologies tend to be regular and can be grouped into two categories: static and dynamic. In a static topology, links between two processors are passive and dedicated buses cannot be reconfigured for direct connections to other processors. On the other hand, links in the dynamic category can be reconfigured by setting the network's active switching elements.

The cross product of the set of categories in each design decision—{operation mode} × {control strategy} × {switching methodology} × {network topology}—represents a space of interconnection networks. Obviously, the cross product contains some uninteresting cases, but a network designer can obtain a meaningful subspace by exercising a practical view of engineering technology.

Topologies

Network topology is a key factor in determining a suitable architectural structure, and many topologies have been considered for telephone switching connections.[12] Here, we review those proposed or used for connections in tightly coupled multiple-processor systems (see Figure 3).

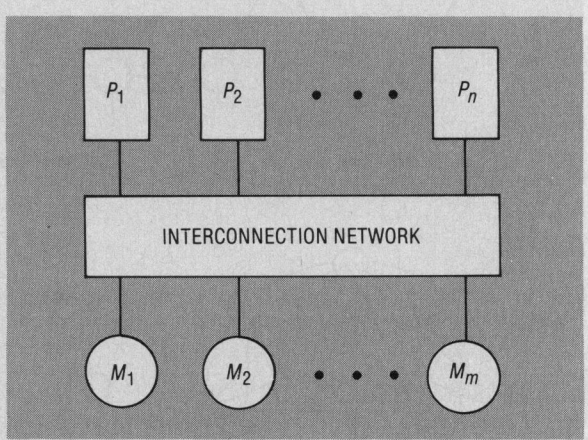


Figure 2. Hardware model of concurrent processing systems.

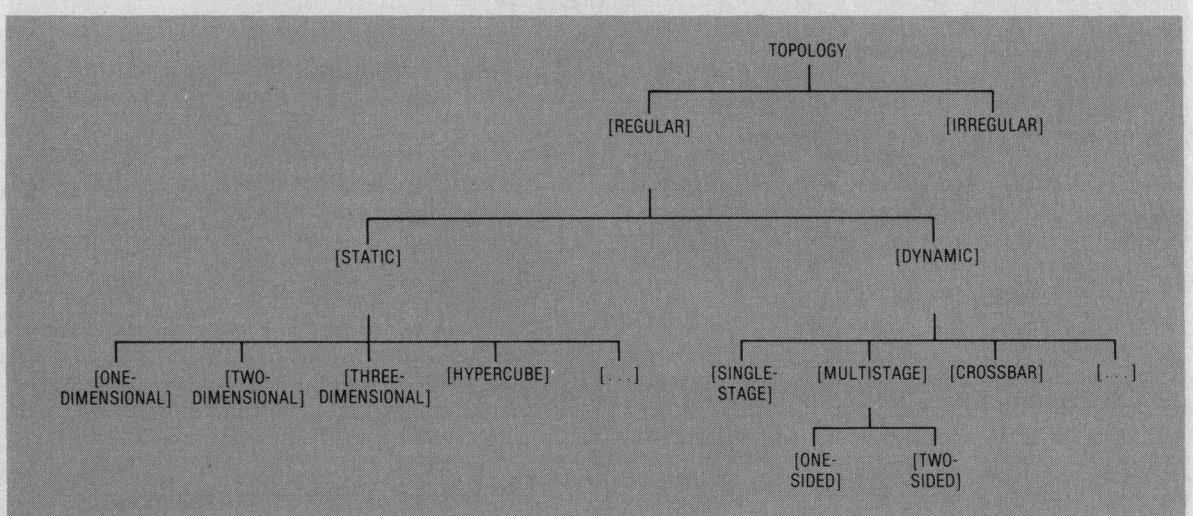


Figure 3. Topologies of interconnection networks.

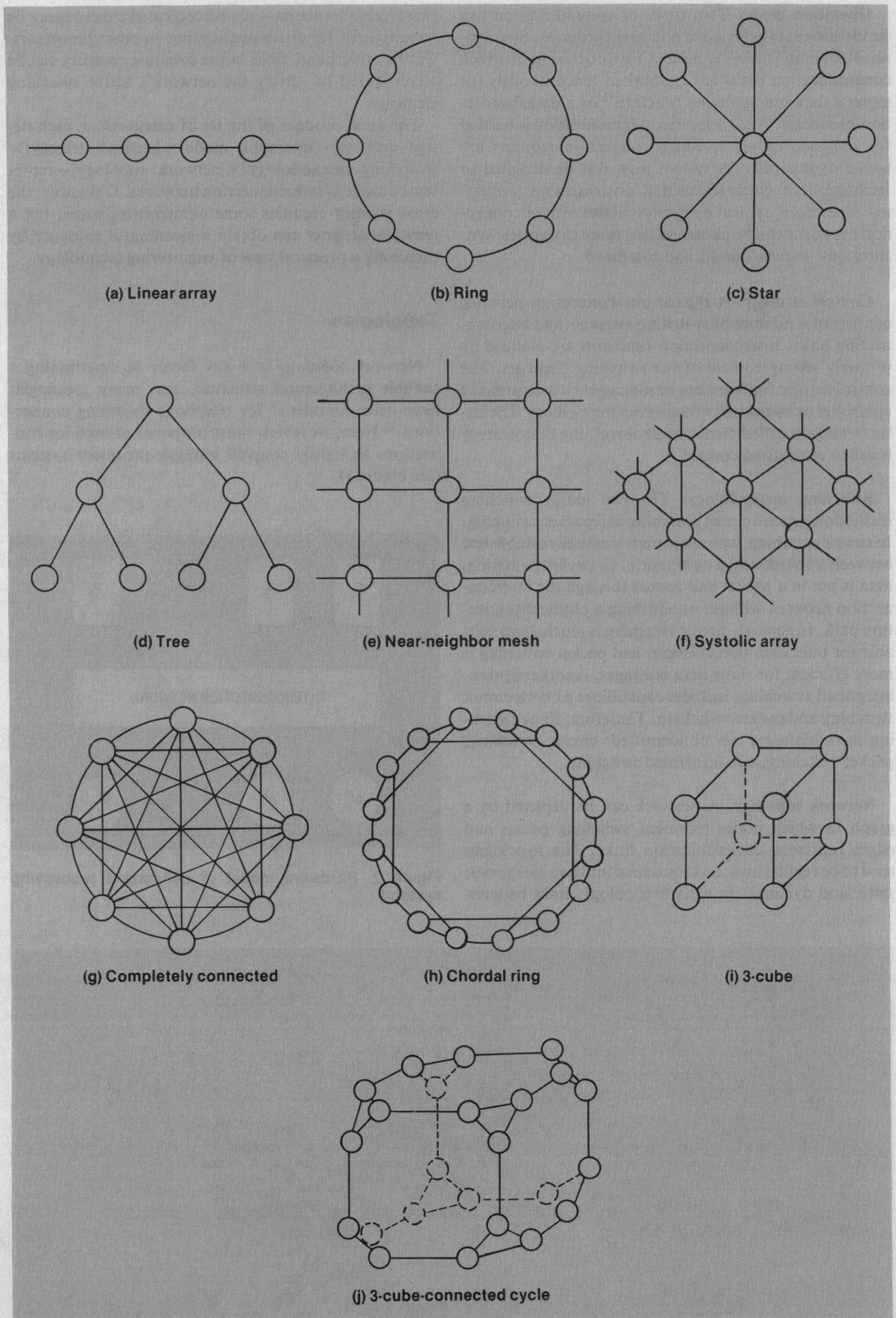


Figure 4. Examples of static network toplogies: (a) one dimensional; (b-f) two dimensional; and (g-j) three dimensional.

Static. Topologies in the static category can be classified according to dimensions required for layout —specifically, one-dimensional, two-dimensional, three-dimensional, and hypercube as shown in Figure 3. Examples of one-dimensional topologies include the linear array used for some pipeline architectures (Figure 4a).[13] Two-dimensional topologies include the ring,[14,15] star,[16] tree,[17] near-neighbor mesh,[18] and systolic array.[13] Examples are shown in Figure 4b-f. Three-dimensional topologies include the completely connected,[19] chordal ring,[20] 3-cube,[21] and 3-cube-connected-cycle[22] networks depicted in Figure 4g-j. A D-dimensional, W-wide hypercube contains W nodes in each dimension, and there is a connection to a node in each dimension. The near-neighbor mesh and the 3-cube are actually two- and three-dimensional hypercubes, respectively. The cube-connected-cycle is a deviation of the hypercube. For example, the 3-cube-connected-cycle shown in Figure 4j is obtained by replacing each node of the 3-cube by a 3-node cycle. Each node in the cycle is connected to the corresponding node in another cycle.

Dynamic. There are three topological classes in the dynamic category: single-stage, multistage, and crossbar (see Figure 5).

Single-stage. A single-stage network is composed of a stage of switching elements cascaded to a link connection pattern. The shuffle-exchange network[23] is a single-stage network based on a perfect-shuffle connection cascaded to a stage of switching elements as shown in Figure 5a. The single-stage network is also called a recirculating network because data items may have to recirculate through the single stage several times before reaching their final destination.

Multistage. A multistage network consists of more than one stage of switching elements and is usually capa-

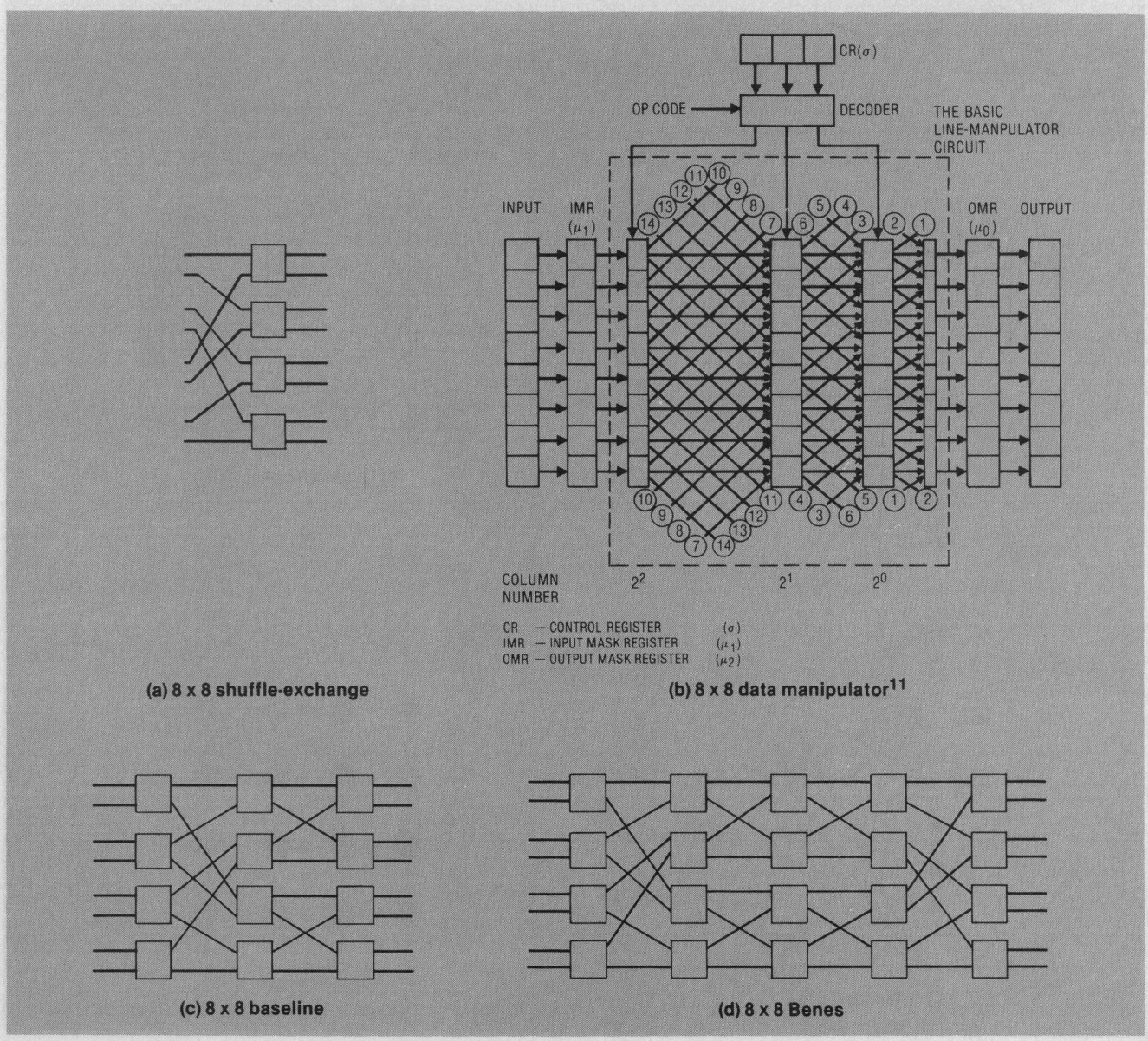
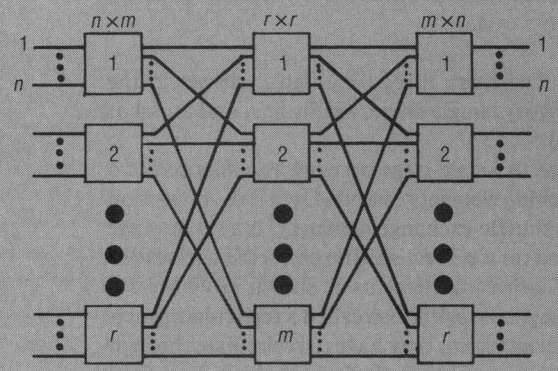
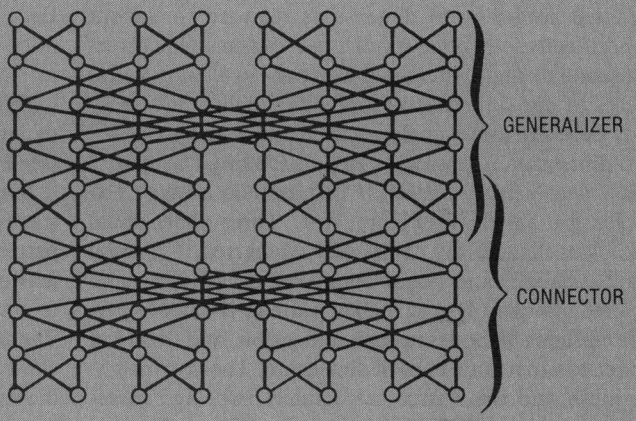


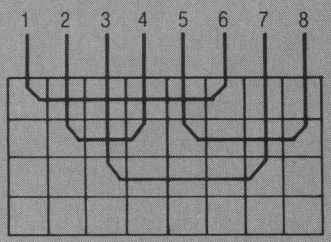
Figure 5. Examples of dynamic network topologies: (a) single stage; (b-i) multistage; and (j) crossbar. **(Cont'd on p. 16.)**



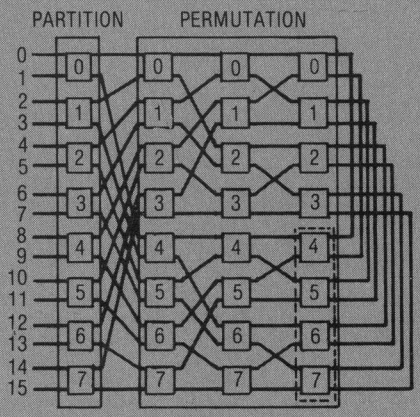
(e) $m \geq 2n - 1$ Clos[38]



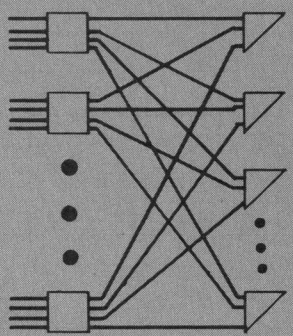
(f) One-to-many nonblocking[39]



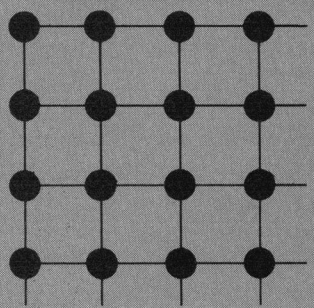
(g) One-sided cellular



(h) One-sided baseline



(i) One-sided Clos



(j) Crossbar

Figure 5 (cont'd from p.15). Examples of multistage and crossbar (j) dynamic network topologies.

ble of connecting an arbitrary input terminal to an arbitrary output terminal. Multistage networks can be one-sided or two-sided. The one-sided networks, sometimes called full switches, have input-output ports on the same side. The two-sided multistage networks, which usually have an input side and an output side, can be divided into three classes: blocking, rearrangeable, and nonblocking.

In blocking networks, simultaneous connections of more than one terminal pair may result in conflicts in the use of network communication links. Examples of this type of network, which has been extensively investigated, include data manipulator,[24] baseline,[25,26] SW banyan,[27] omega,[28] flip,[29] indirect binary n-cube,[30] and delta.[31] A topological equivalence relationship has been established for this class of networks in terms of the baseline network.[25,26] A data manipulator and a baseline network are shown in Figure 5b and 5c.

A network is called a rearrangeable nonblocking network if it can perform all possible connections between inputs and outputs by rearranging its existing connections so that a connection path for a new input-output pair can always be established. A well-defined network, the Benes network[12] shown in Figure 5d, belongs to this class. The Benes rearrangeable network topology has been extensively studied for use in synchronous data permutation[32-35] and asynchronous interprocessor communication.[36,37]

A network which can handle all possible connections without blocking is called a nonblocking network. Two cases have been considered in the literature. In the first case, the Clos network[38] shown in Figure 5e, a one-to-one connection is made between an input and an output. The other case considers one-to-many connections.[39] Here, a generalized-connection network topology is generated to pass any of the N^N mapping of inputs onto outputs where N is the number of inputs or outputs (see Figure 5f). In a one-sided network (or full switch), one-to-one connection is possible between all pairs of terminals.[40,41] A cellular implementation, a base-line topology construction, and a Clos construction are shown in Figure 5g-i.

Crossbar. In a crossbar switch every input port can be connected to a free output port without blocking. Figure 5j shows a schematic which is similar to one used in C.mmp.[42] A crossbar switch called a versatile line manipulator has also been designed and implemented.[43,44]

Communication protocols

The switching methodology and the control strategy are implemented in switching elements (or switching points) according to required communication protocols. The communication protocols can be viewed on two levels. The first level concerns switching control algorithms which generate necessary control settings on switching elements to ensure reliable data routings from source to destination. The first-level protocols are referred to as routing techniques here. The second level is concerned with the link control procedure that provides the handshaking process among switching points. The handshaking process is a basic function implemented by switching elements.

Routing techniques. The routing techniques depend on the network topology and the operation mode used. More or less, each multiple-processor system needs a routing algorithm. Here, we use several well-defined routing algorithms for examples.

Near-neighbor mesh. Bitonic sort has been adapted by several authors[45-47] for the routing of an $n \times n$ mesh-connected, single instruction-multiple data stream system. The procedure developed by Nassimi[47] is as follows:

Procedure SORT (n,n)
1) K ← S ← 1
2) **While** K < n **do**
   a) consider the n × n processor array as composed of many adjacent K × 2K subarrays
   b) **do** in parallel for each K × 2K array
      HORIZONTAL_MERGE(K, 2K)
   c) S ← S + 1
   d) Consider the n × n processor array as composed of many adjacent 2K × 2K subarrays
   e) **do** in parallel for each 2K × 2K subarray
      VERTICAL_MERGE(2K, 2K)
   f) S ← S + 1; K ← 2 * K

end

end SORT

The HORIZONTAL_MERGE sorts a bitonic sequence arranged in two arrays with the increasing sequence on the

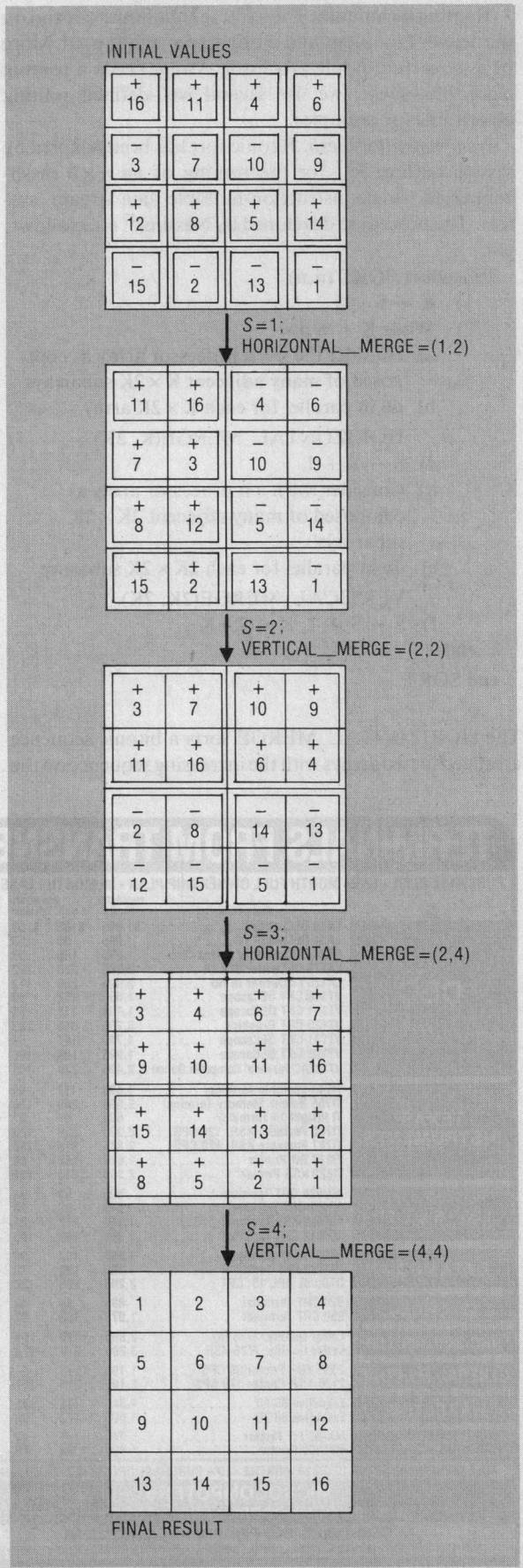


Figure 6. A complete example of sorting a 4 x 4 array.

left array and the decreasing sequence on the right array, or vice versa. Similarly, the VERTICAL_MERGE sorts a bitonic sequence arranged in two arrays with the increasing sequence on the upper array and the decreasing sequence on the lower array, or vice versa. A complete example of sorting a 4×4 array is shown in Figure 6. The order into which a subarray gets sorted is determined by the SIGN function, "+" and "−", used during a comparison-interchange where "+" is for nondecreasing order and "−" is for nonincreasing order. In Figure 6, the initial values given go through an HM sort on two 1×1 arrays, a VM sort on two 1×2 arrays, an HM sort on two 2×2 arrays, and finally a VM sort on two 2×4 arrays.

Shuffle-exchange network. Both centralized and distributed routings have been worked out for the shuffle-exchange network. It has been shown that the shuffle-exchange network can realize an arbitrary permutation in $3(\log_2 N) - 1$ passes where N is the network size.[48] An example is shown in Figure 7 for the following permutation:

$$p = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 14 & 12 & 5 & 7 & 15 & 8 & 9 & 13 & 4 & 3 & 10 & 6 & 1 & 0 & 2 & 11 \end{pmatrix}$$

The control setting developed consists of three matrices, $\overline{F}$, $\overline{S}$, and $\overline{T}$. Among these three control matrices, $\overline{S}$ is independent of the permutation and $\overline{F}$ and $\overline{T}$ are modified matrices obtained by performing some prescribed operations on the control matrix for the Benes binary network. The detailed transformation is shown in Wu and Feng.[48] The shuffle-exchange network can also be constructed to adapt to a distributed control scheme. The construction can be considered as a sorting network, and the binary codes of the destination names are used as the values to be sorted.[23,49] Figure 8 illustrates an example for 2^n elements where $n = 4$. Each of the n^2 steps in this scheme consists of a perfect-shuffle followed by simultaneous operations performed on 2^{n-1} pairs of adjacent elements. Each of the latter operations is either "0" (no operation, straight connection), "+" (comparator module which sends the larger value to the lower link), or "−" (a reverse comparator module). The sorting proceeds in n stages of n steps each: during stage s, for $s \leq n$, we do $n - s$ steps in which all operations are "0", followed by s steps in which the operations consist alternately of 2^t "+" followed by 2^t "−" for $t = 1, 2, \ldots, s$. During the last stage, all operations are "+".

Data manipulator. A centralized control scheme is designed for implementing data manipulating functions such as permuting, replicating, spacing, masking, and complementing.[11] To implement a data manipulating function, proper control lines of the six groups ($U_1^{2^i}$, $U_2^{2^i}$, $H_1^{2^i}$, $H_2^{2^i}$, $D_1^{2^i}$, $D_2^{2^i}$) in each column must be properly set through the use of the control register and the associated decoder. A "duplicate spaced substrings down" operation is illustrated in Figure 9. The two substrings to be duplicated are AB and EF. For this operation the control line groups $D_1^{2^i}$ and $H_1^{2^i}$ or $H_1^{2^i}$ and $H_2^{2^i}$ are activated, depending on whether the control bit is 1 or 0 as determined by substring length. In this example, the substring is 2; thus, only the control bit for column 2^1 has a value of 1, all others are 0's. Thus, in columns 2^2 and 2^0, $H_1^{2^i}$ and $H_2^{2^i}$ are activated,

and in column 2^1, $D_1^{2^i}$ and $H_1^{2^i}$ are activated. With this control pattern, the substrings can be generated at the output register.

A distributed control scheme has also been developed by McMillen and Siegel.[50] It uses a routing tag which contains $2n$ bits and is of the form $F = (f_{2n-1} \cdots f_{n+1} f_n f_{n-1} \cdots f_1 f_0)$. The n low-order bits represent the magnitudes of the route, and the n high-order bits represent the sign corresponding to the magnitudes. In stage i, a given switching element examines bits i and $n+i$ of the routing tag. If $f_i = 0$, the straight link is used, regardless of the value of f_{n+i}. If $f_i = 1$, bit $n+i$ is examined. If $f_{n+i} = 0$, the $+2^i$ link is used; if $f_{n+i} = 1$, the -2^i link is used. The source processor generates its own routing tag. For example, in a data manipulator of $N = 2^4$, if the source is 13 and the destination is 6, one possible value for F is 00000111. The path traversed is straight, $+2^2$, $+2^1$, $+2^0$. Multiple paths exist between a source-destination pair. For example, an alternative routing tag from source 13 to destination 6 is (0001 1001). The example is shown in Figure 10. A general rule to calculate the routing tag is shown as

$$D = S + (-1)^{f_{2n-1}}(f_n 2^{n-1}) + (-1)^{f_{2n-2}}(f_{n-1}2^{n-2}) + \cdots (-1)^{f_n}(f_0 2^0)$$

where S and D are the addresses of the source and the destination, respectively.

Baseline network. Routing techniques for baseline networks described here are also useful for other topological-

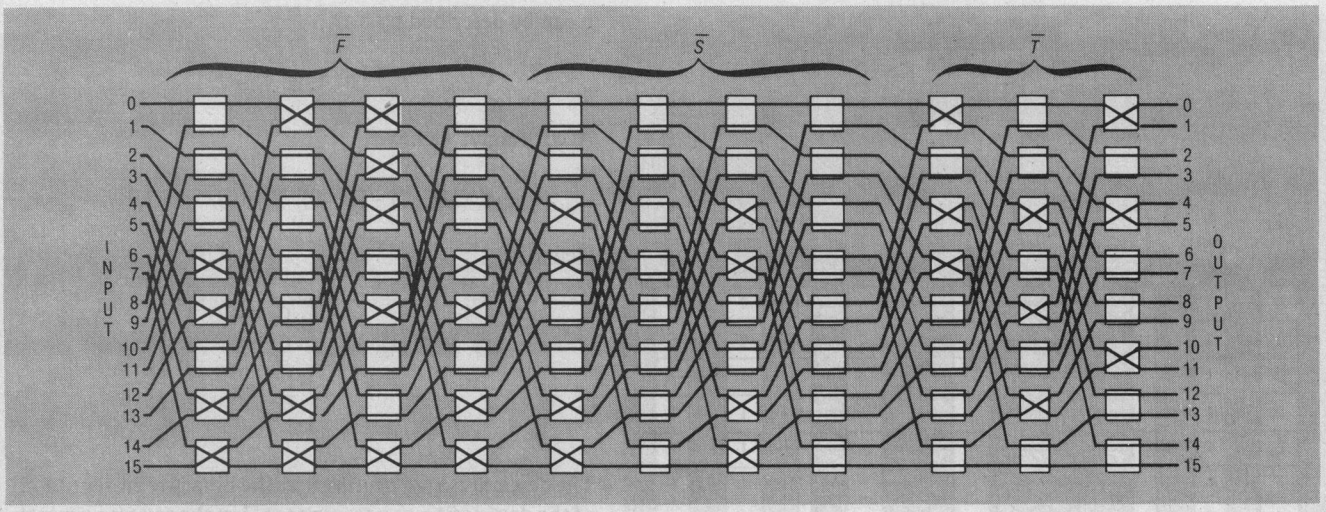


Figure 7. An example for universal realization of permutations.[48]

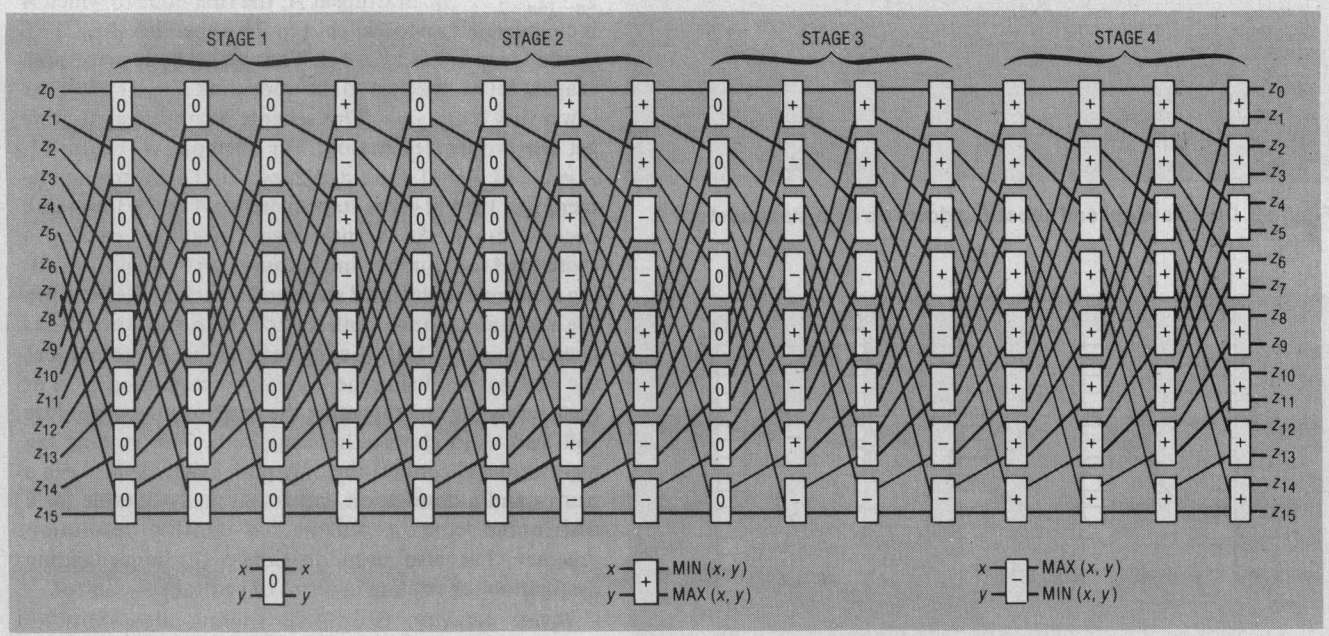


Figure 8. Sorting with shuffle-exchange. (Adapted from *The Art of Computer Programming, Vol. 3: Sorting and Searching* by D. E. Knuth; Addison-Wesley, Reading, Mass., © 1973.)

ly equivalent blocking multistage networks.[25] Basically, two types of routing are available: recursive routing and destination tag routing.[25,28,51] The recursive routing algorithm determines the control pattern according to permutation names. For some permutation, useful in parallel processing, the control pattern can be calculated recursively on the fly as the data pass through the network. Six categories of such permutations have been identified. For our purpose, we describe one here and show the recursive routing algorithm. The flip permutation function[29] is described as follows:

$$F_k^{(n)}\ (0 \le k < 2^n)\colon p(X^r \oplus k) = X \text{ and } p(X \oplus k) = X^r$$

where X^r is the number whose binary representation is the reverse of X. Let $k = 2k^1 + k_0$ and $[L;R]$ denote the cascaded matrix whose left part and right part are L and R, respectively. Also let $V^{(n-1)}(b)$ be the 2^{n-1} bit vector whose components are all equal to b. The control pattern $K^{(n)}$ of the flip function can then be expressed in terms of the following recursive formula:

$$K^{(n)}(F_k^{(n)}) = [V^{(n-1)}(k_0); K^{(n-1)}(F_k^{(n)})],$$

where

$$K^{(1)}(F_k^{(n)}) = [V^{(n-1)}(k)].$$

For example, assuming

$$p = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 5 & 3 & 7 & 0 & 4 & 2 & 6 \end{pmatrix}$$

p can be described by

$$F_4^{(3)}\colon p(X^r \oplus 4) = X.$$

Accordingly, we have

$$K^{(3)}(F_4^{(3)}) = [V^2(0); V^{(2)}(0); V^{(2)}(1)].$$

Hence

$$K^{(3)}(p) = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

The destination tag routing uses the binary representation of the destination as a routing tag. Let the source terminal link and destination terminal link be A and Z, respectively. Also, let the binary representation of Z be $z_{n-1}z_{n-2} \cdots z_0$. Starting at A, the first node to which A is connected is set to switch A to the upper link if $z_{n-1} = 0$ or the lower link if $z_{n-1} = 1$. The second node in the path is again set to switch A to the upper link if $z_{n-2} = 0$ or the lower link if $z_{n-2} = 1$. This scheme is continued until we get the proper destination. For example, in Figure 11, $A = 2$ and $Z = 11$ (i.e., $z_3z_2z_1z_0 = 1011$). Switching element 1 of the left-most stage switches A to the lower link because $z_3 = 1$. At the next stage, switching element 4 switches A to the upper link because $z_2 = 0$. Again, switching element 4 in the third stage and switching element 5 in the right-most stage both switch A to the lower links because $z_1 = z_0 = 1$. If we consider Z as the source and A as the destination, using the binary representation of A as the routing tag and repeating the same routing procedure will lead us to choose the same path. This routing tag algorithm will connect the only path available between a source and a destination and is extremely suitable for a distributed control scheme. A conflict resolution scheme[25] has also been developed for implementing destination tag routing in terms of centralized control.

Benes network. Sequential routing algorithms[34,52] need $O(N \log N)$ steps where N is the network size. Many researchers have worked toward improving this time com-

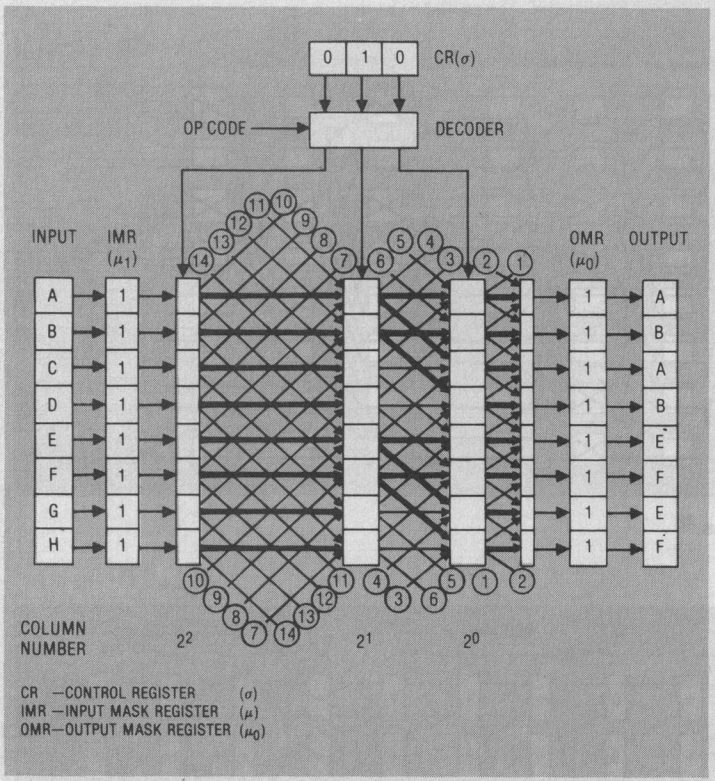


Figure 9. Duplicate spaced substring down on data manipulator.[11]

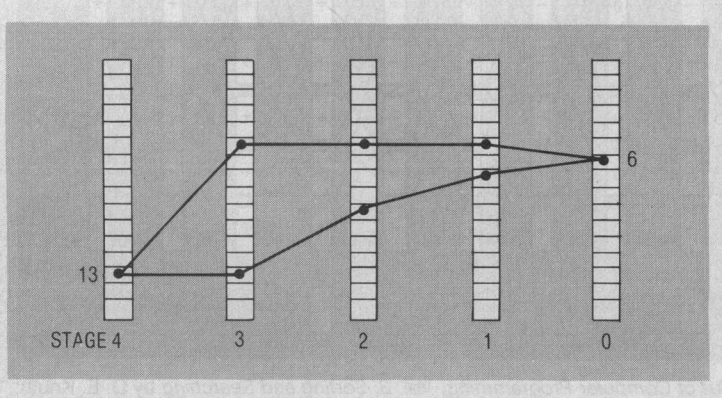


Figure 10. Distributed routing on the data manipulator.

plexity in terms of parallel processing technique,[53] heuristic method,[37] or recursive formula.[35] Here, we demonstrate the very basic routing algorithm, called the looping algorithm. The basic principle, in terms of the permutation to be realized by the Benes binary network shown in Figure 5d, is

$$p = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 3 & 7 & 4 & 0 & 2 & 6 & 1 & 5 \end{pmatrix}$$

The loop algorithm starts recording the permutation, p, as shown in Figure 12. The two output numbers of a switching element in the output stage are shown in the same column, and the two input numbers of a switching element in the input stage are shown in the same row. We then choose an arbitrary entry in the chart as a starting point. For example, electing to start at row 23 and column 01, we then look for a same-row or column entry to form a loop and, in Figure 12, choose row 23 and column 45. The process continues until we obtain a loop by re-entering row 23 and column 01. The loop's member entries are then assigned "*a*" and "*b*" alternately. The second loop can be formed in the same way. Then, we assign input and output lines named "*a*" to subnetwork *a* and those named "*b*" to subnetwork *b*. The control of the input and output switching elements must be set as depicted in Figure 13. This looping algorithm can be applied recursively to the two subnetworks.

Construction of interconnection networks. Interconnection networks are usually designed so they can be constructed of a single type of modular building block called a switching element. The switching element realizes communication protocols which specify the control strategy and the switching methodology.

The logic design of switching elements has been explored in many projects,[54-56] including recent LSI implementations.[10,57] Here, we describe in more detail three designs that have been implemented and are operational.

Flip network 2 × 2 switching element. The flip network uses centralized control and circuit switching.[29] The 2 × 2 switching element can be set by a control line into a direct-connection or crossed-connection state. Assume that I_0, I_1, O_0, O_1, and C represent the two inputs, the two outputs, and the switching element control. The switching element's output function can be expressed as follows:

$$O_0 = \overline{C} I_0 + C \cdot I_1, \text{and}$$

$$O_1 = \overline{C} I_1 + C \cdot I_0$$

where $C = 0$ means straight connection and $C = 1$ crossed connection (see Figure 14).

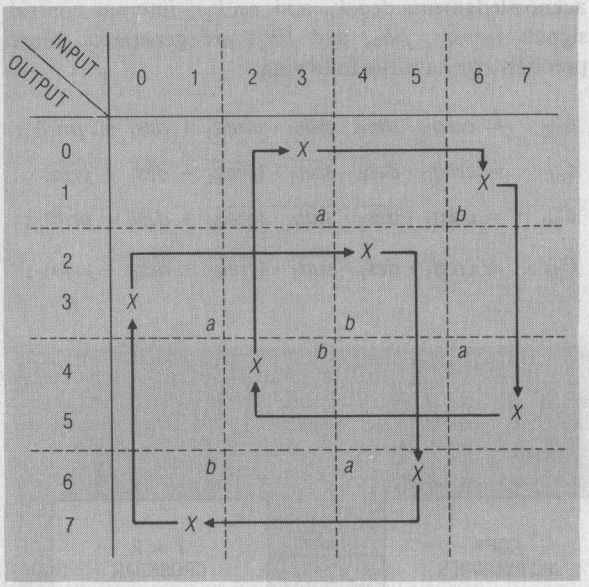


Figure 12. An example of the looping algorithm.

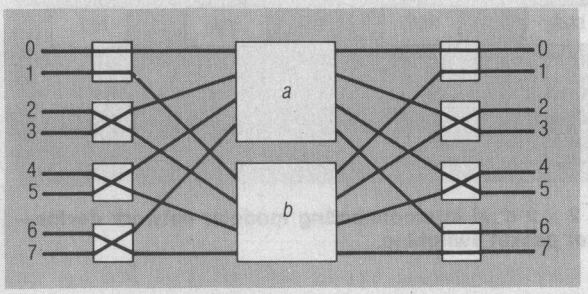


Figure 13. Control setting result from the first iteration of the looping algorithm.

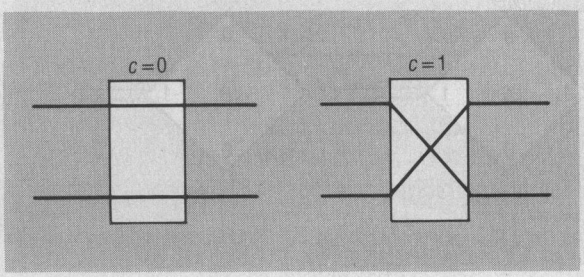


Figure 14. A 2 x 2 switching element.

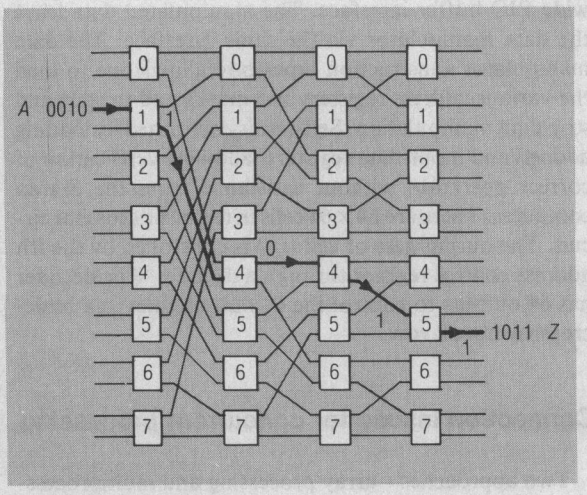


Figure 11. Distributed routing on a baseline network.

Dimond 2 × 2 switching element. A switching element with two input and output ports, called Dimond for dual interconnection modular network device,[58] allows modular construction of interconnection networks. A packet of messages (containing routing information) arriving at a Dimond is switched to a designated output port, where it is stored in a register. Figure 15 shows an implementation of Dimond which requires one control clock for all interconnected switching elements. The central clock has two phases. In the first clock phase, it is determined which inputs have to be copied into which registers. The copy allowances so determined are stored in four flip-flops: C_{00}, C_{01}, C_{10}, and C_{11} (C_{01} is the allowance for copying in_0 into reg_1). In addition, output signals of copy acknowledgments ($cack_0$ and $cack_1$), internal control signals ($cross_0$, $fill_0$, and $fill_1$) are generated. More precisely, we have the following:

$$C_{00} = creq_0 \cdot \overline{des_0} \cdot \overline{stat_0} \cdot (\overline{creq_1} + des_1 + , \overline{prio}) \ ;$$

$$C_{01} = creq_0 \cdot des_0 \cdot \overline{stat_1} \cdot (\overline{creq_1} + \overline{des_1} + \overline{prio}) \ ;$$

$$C_{10} = creq_1 \cdot \overline{des_1} \cdot \overline{stat_0} \cdot (\overline{creq_0} + des_0 + prio) \ ;$$

$$C_{11} = creq_1 \cdot des_1 \cdot \overline{stat_1} \cdot (\overline{creq_0} + \overline{des_0} + prio) \ ;$$

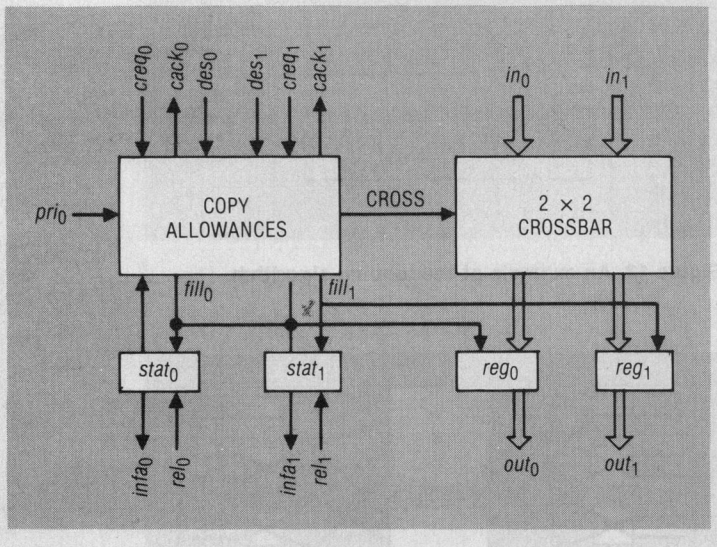


Figure 15. A 2 x 2 dual interconnecting modular network device—Dimond[58]—for packet switching.

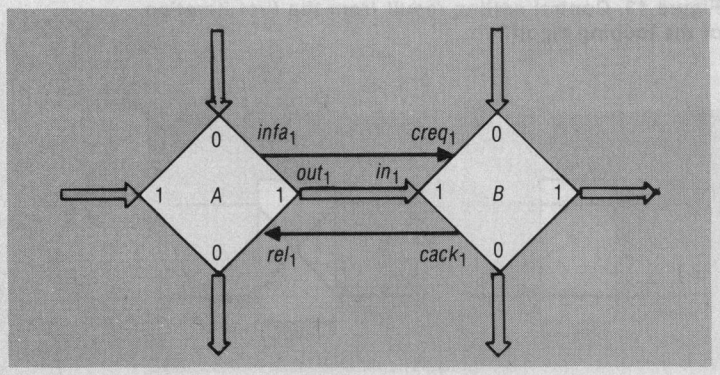


Figure 16. Connecting two Dimonds.[58]

$$Cack_0 = C_{00} + C_{01} \ ;$$

$$Cack_1 = C_{10} + C_{11} \ ;$$

$$Cross = C_{00} + C_{11} \ ;$$

$$Fill_0 = C_{00} + C_{10} \ ;$$

$$Fill_1 = C_{01} + C_{11} \ ;$$

where *prio* is the priority line indicating the index (0,1) of the input served first in the event of conflict, and des_0 and des_1 are destination lines for in_0 and in_1. In the second clock phase, two actions are performed concurrently. Inputs are copied into the output register, if required, and the status flip-flops, $stat_0$ and $stat_1$ (status of reg_0 and reg_1, respectively) are adapted. Precisely, we have the following:

$$Fill_0 \rightarrow reg_0 = cross \cdot in_0 + \overline{cross} \cdot in_1 \ ;$$

$$Fill_1 \rightarrow reg_1 = \overline{cross} \cdot in_0 + cross \cdot in_1 \ ;$$

$$Fill_0 \cdot rel_0 \rightarrow stat_0 = 1 \ ;$$
$$rel_0 \rightarrow stat_0 = 0 \ ;$$

$$Fill_1 \cdot \overline{rel_1} \rightarrow stat_1 = 1 \ ;$$
$$rel_1 \rightarrow stat_1 = 0 \ .$$

The information-available lines are connected to the status flip-flops:

$$infa_0 = stat_0 \ ;$$

$$infa_1 = stat_1 \ .$$

The interconnection of two Dimonds is shown in Figure 16, which depicts the relation of handshaking lines.

64 × 64 switching element. A centralized-control and circuit-switching 64 × 64 versatile data manipulator[11] (see Figure 17) is operating in conjunction with the Staran computer at the Rome Air Development Center.[44] The data manipulator operates under the control of the Staran computer's parallel input-output unit. The contents of the input and output masks, of the address control register, and of the input and output control registers, as well as the data to be manipulated, are entered via the 256-bit wide PIO buffer interface. The manipulated data leave the data manipulator via the same interface. The data manipulator's instruction repertoire allows one to load the various address registers and masks and to start and stop data manipulation. Self-test is performed by loading address and input-data registers, allowing verification of correct operation without assistance from the Staran computer. There are 64 × 64 cells in the basic crossbar circuit. The output gate of cell (i,j) is controlled by the ith address control register through a decoder. The decoder has 64 outputs to control the 64 output gates in a basic-crossbar-circuit row.

Connection issues for concurrent processing

Two approaches—array processing and multiprocessing—have been tried to provide processing concurrency.

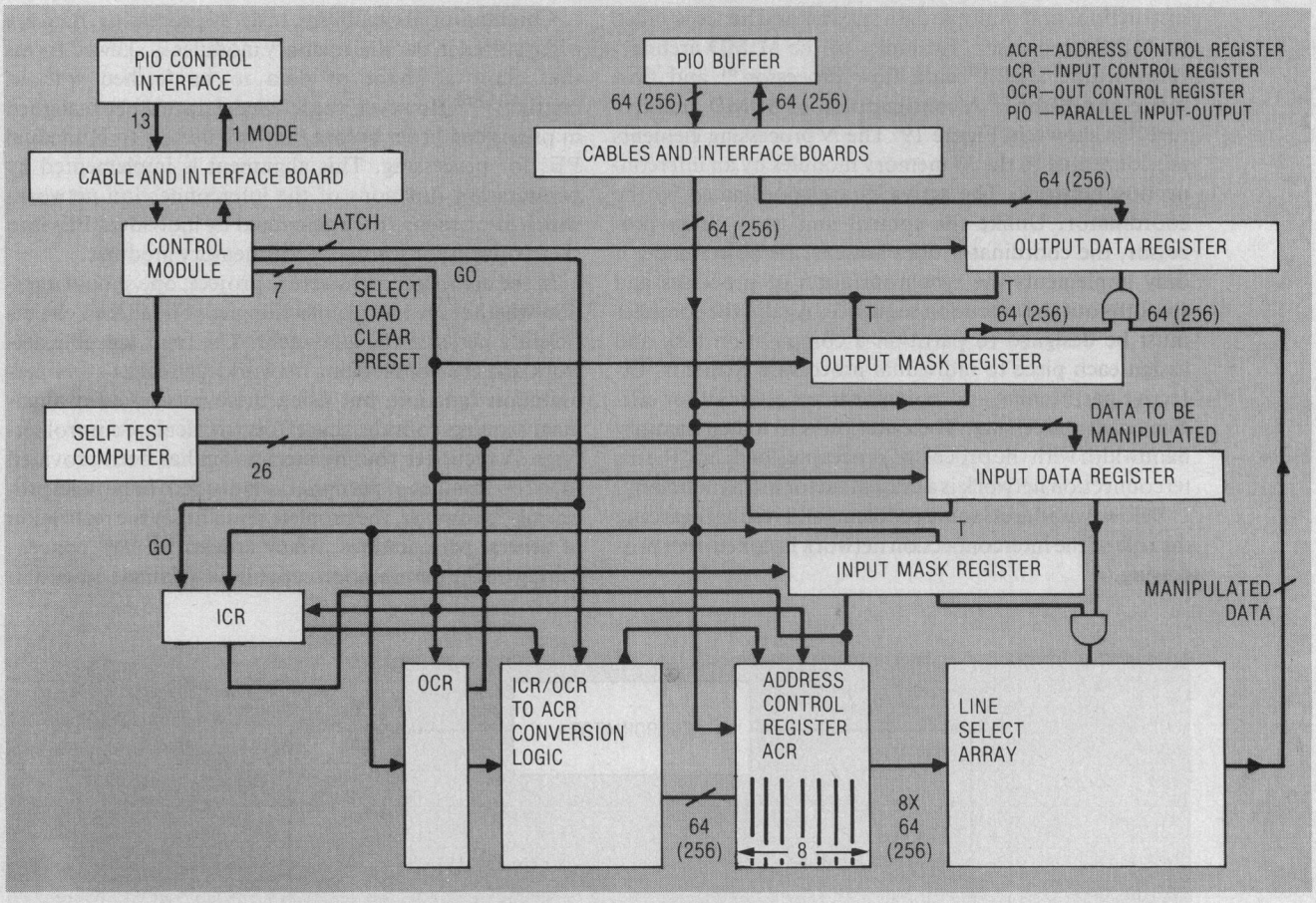


Figure 17. Block diagram of a versatile data manipulator.

Since array processors, which consist of multiple processing elements and parallel memory modules under one control unit, can handle single instructions and multiple data streams, they are also known as SIMD computers. Existing examples include Illiac IV and Staran. An overall SIMD machine organization[59] is shown in Figure 18. The N processing elements, or PEs, are connected by two interconnection networks to the M parallel memory modules. The control unit in the center provides control over PEs and memory modules.

Array processors allow explicit expression of parallelism in user programs. The compiler detects the parallelism and generates object code suitable for execution in the multiple processing elements and the control unit. Program segments which cannot be converted into parallel executable forms are executed in the control unit; program segments which can be converted into parallel executable forms are sent to the PEs and executed synchronously on data fetched from parallel memory modules under the control of the control unit. To enable synchronous manipulation in the PEs, the data are permuted and arranged in vector form. Thus, to run a program more efficiently on an array processor, one must develop a technique for vectorizing the program (or algorithm). The interconnection network plays a major role in vectorization.

The second approach for concurrent processing uses multiprocessing. The multiprocessor can handle multiple

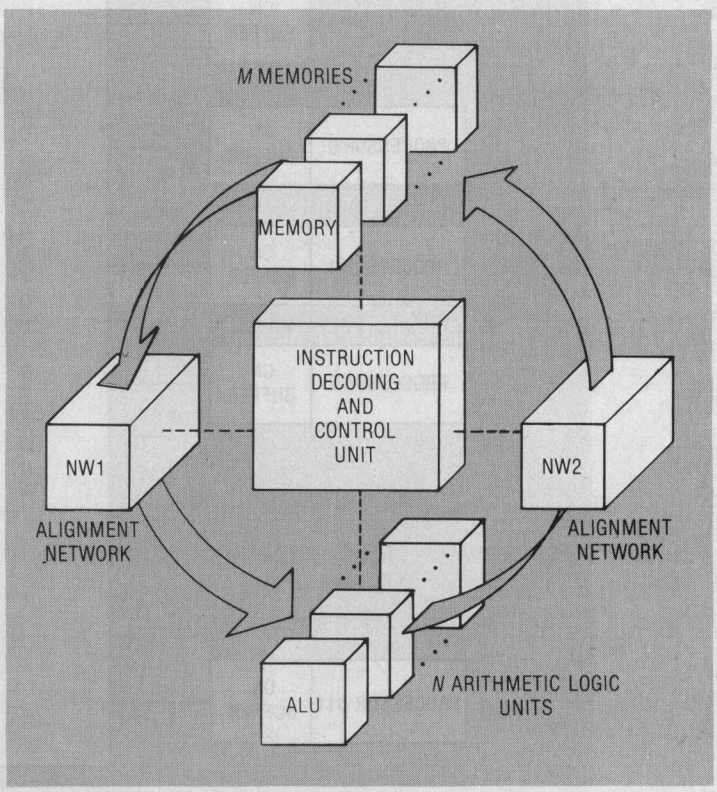


Figure 18. SIMD model.[59]

instructions and multiple data streams and hence is called an MIMD processor. Examples of the MIMD architecture include HEP,[60] data flow processor,[61] and flow model processor.[62] A configuration of MIMD architecture[62] is shown in Figure 19. The N processing elements are connected to the M memory modules by an interconnection network. The activities are coordinated by the coordinator. Unlike the control unit in an array processor, the coordinator does not execute object code; it only implements the synchronization of processes and smooths out the execution sequence. Again, the compiler must be designed to partition a computation task and assign each piece to individual processing elements. Effective partitioning and assignment are essential for efficient multiprocessing. The criterion is to match memory bandwidth with the processor processing load, and the interconnection network is a critical factor in this matching.

Below, we address some problems and results regarding the role of the interconnection network in concurrent processing.

Combinatorial capability. In array processing, data are often stored in parallel memory modules in skewed forms that allow a vector of data to be fetched without conflict.[63-65] However, the fetched data must be realigned in prescribed order before they can be sent to individual PEs for processing. This alignment is implemented by permutation functions of the interconnection network, which also realigns data generated by individual PEs into skewed form for storage in the memory modules.

In the computer architecture project, one should question whether the interconnection network chosen can efficiently perform the alignment. The rearrangeable network and the nonblocking network can realize every permutation function, but using these networks for alignment requires considerable effort to calculate control settings. A recursive routing mechanism has been provided for a few families of permutations needed for parallel processing[35]; however, the problem remains for the realization of general permutations. Many articles[45,51,66,67] concentrating on the permutation capabilities of single-stage net-

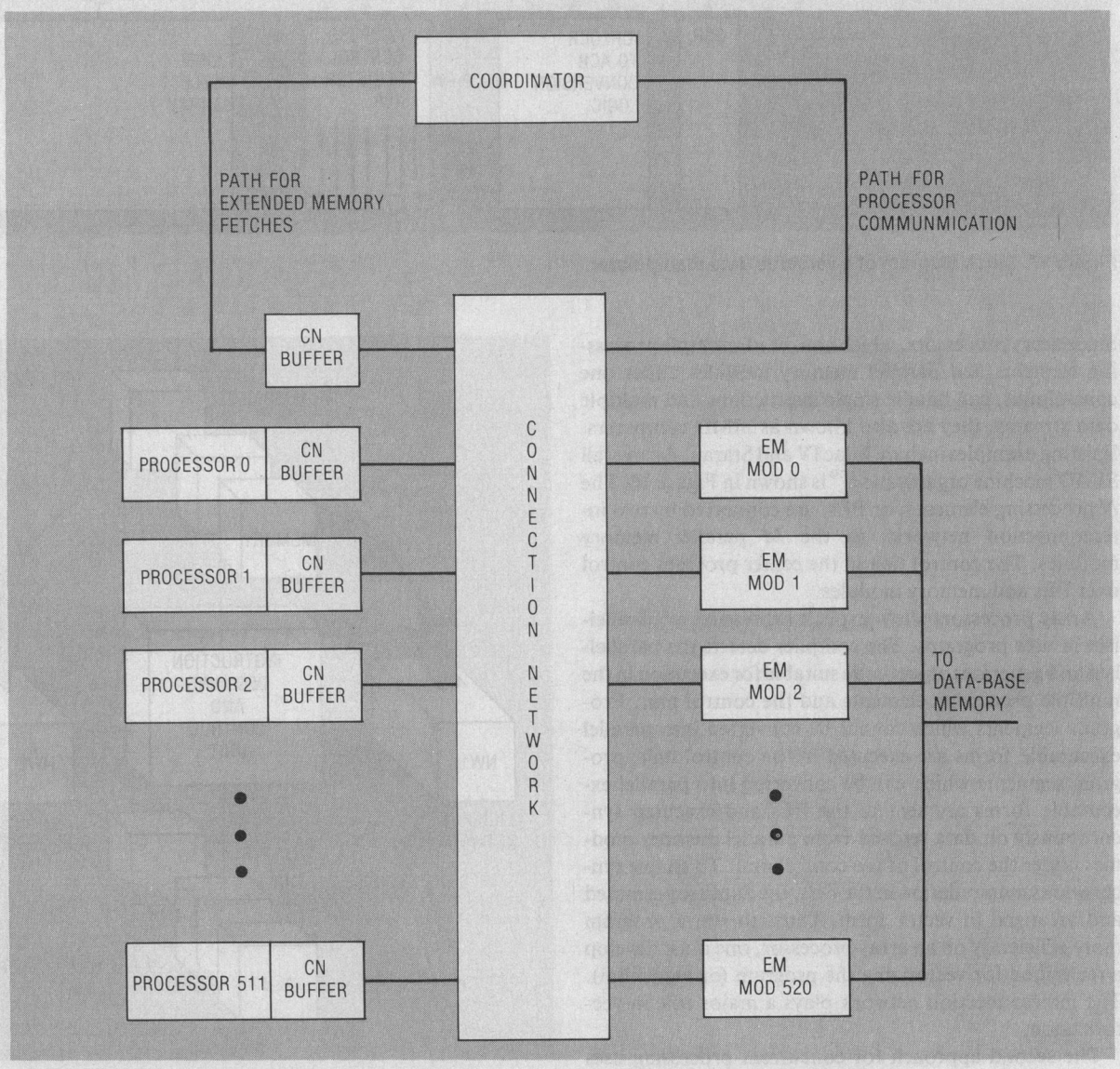


Figure 19. MIMD model.

works and blocking multistage networks have shown that these networks cannot realize arbitrary permutations in a single pass. Recent results show that the baseline network can realize arbitrary permutations in just two passes[51] while other blocking multistage networks, such as the omega network, need at least three passes.[66] As mentioned previously, the shuffle-exchange network can realize arbitrary permutations in $3(\log_2 N) - 1$ passes where N is the network size.[48]

Task assignments and reconfiguration. Consider a parallel program segment using M memory modules and N processing elements. During execution, data is usually transferred from memory modules to processing elements or vice versa. It is also necessary to transfer data among processing elements for data sharing and synchronization. Simultaneous data transfers through the interconnection network, which implements the transfers, may result in contention for communication links and switching elements. In case of conflict, some of the data transfers must be deferred; consequently, throughput decreases because the processing elements which need the deferred data cannot proceed as originally expected. To minimize delays caused by communication conflicts, program codes must be assigned to proper processing elements and data assigned to proper memory modules. The assignment of data to memory modules, called mapping,[68] has recently been extended to include assignment of program modules to processing elements.[69]

A configuration concept has been proposed to better use the interconnection network.[51] Under this concept, a network is just a configuration of another one in the same, topologically equivalent class.[25] To configure a permutation function as an interconnection network, we can assign input/output link names in a way that realizes the permutation function in one conflict-free pass. The problem of assigning logical names that realize various permutation functions without conflicts is called a reconfiguration problem. It has been shown that, through the reconfiguration process, the baseline network can realize every permutation in one pass without conflicts.[69] This implies that concurrent processing throughput could be enhanced by proper assignment of tasks to processing elements and data to memory modules.

Partitioning. In partitioning—that is, dividing the network into independent subnetworks of different sizes—each subnetwork must have all the interconnection capabilities of a complete network of the same type and size. Hence, with a partitionable network, a system can support multiple SIMD machines. By dynamically reconfiguring the system into independent SIMD machines and properly assigning tasks to each partition, we can use resources more efficiently.

Several authors have noted the importance of partitioning.[30] One recent study[70] shows that single-stage networks, such as the shuffle-exchange and Illiac networks, cannot be partitioned into independent subnetworks, but blocking multistage networks, such as the baseline and data manipulator, can be partitioned.

Bandwidth of interconnection networks. The bandwidth can be defined as the expected number of requests accepted per unit time. Since the bus system cannot provide sufficient bandwidth for a large-scale multiprocessor system and the crossbar switch is too expensive, it is particularly interesting to know what kind of bandwidth various interconnection networks can provide.

The analytic method has been used to estimate bandwidth.[31,71,72] However, one cannot obtain a closed-form solution, and the analytic model is sometimes too simplified. Just for example, one result showed that for a blocking multistage interconnection network of size 256×256, the bandwidth is 77 requests (per memory cycle) and for a crossbar switch of the same size, the bandwidth is 162. However, the crossbar costs about 20 times as much as the multistage network, and with buffering (packet switching), the performance of the multistage network is quite comparable to the crossbar switch.[71]

Numerical simulation, also used to estimate the bandwidth,[71] can simulate actual PE connection requests by analyzing the program to be executed. The access conflicts in the network and memory modules can be detected as shown by Wu and Feng.[25] Using the simulation method, Barnes[73] concluded that the baseline network is more than adequate to support connection needs of a proposed MIMD system which can execute one billion floating-point instructions per second.

Reliability. Reliable operation of interconnection networks is important to overall system performance. The reliability issue can be thought of as two problems: fault diagnosis and fault tolerance. The fault-diagnosis problem has been studied for a class of multistage interconnection networks constructed of switching elements with two valid states.[74] The problem is approached by generating suitable fault-detection and fault-location test sets for every fault in the assumed fault model. The test sets are then trimmed to a mimimal or nearly minimal set. Detecting a single fault (link fault or switching-element fault) requires only four tests, which are independent of network size. The number of tests for locating single faults and detecting multiple faults are also workable.

The second reliability problem mainly concerns the degree of fault tolerance.[75] It is important to design a network that combines full connection capability with graceful degradation—in spite of the existence of faults. ■

Acknowledgment

The author wishes to acknowledge the original contribution of Dr. C. Wu in preparing this article.

References

1. T. Feng, editor's introduction, special issue on parallel processors and processing, *Computing Surveys*, Vol. 9, No. 1, Mar. 1977, pp. 1-2.

2. C. V. Ramamoorthy, T. Krishnarao, and P. Jahanian, "Hardware Software Issues in Multi-Microprocessor Computer Architecture," *Proc. First Annual Rocky Mountain Symp. Microcomputers,* 1977, pp. 235-261.
3. K. J. Thurber, "Interconnection Networks—A Survey and Assessment," *AFIPS Conf. Proc.,* Vol. 43, 1974 NCC, pp. 909-919.
4. K. J. Thurber, "Circuit Switching Technology: A State-of-the-Art Survey," *Proc. Compcon Fall 1978,* Sept. 1978, pp. 116-124.
5. K. J. Thurber and G. M. Masson, *Distributed-Processor Communication Architecture,* Lexington Books, Lexington, Mass., 1979, 252 pp.
6. H. J. Siegel, "Interconnection Networks for SIMD Machines," *Computer,* Vol. 12, No. 6, June 1979, pp. 57-66.
7. H. J. Siegel, R. J. McMillen, and P. T. Mueller, Jr., "A Survey of Interconnection Methods for Reconfigurable Parallel Processing Systems," *AFIPS Conf. Proc.,* Vol. 48, 1979 NCC, pp. 387-400.
8. G. M. Masson, G. C. Gingher, and Shinji Nakamura, "A Sampler of Circuit Switching Networks," *Computer,* Vol. 12, No. 6, June 1979, pp. 32-48.
9. T. Feng and C. Wu, *Interconnection Networks in Multiple-Processor Systems,* Rome Air Development Center report, RADC-TR-79-304, Dec. 1979, 244 pp.
10. C. Wu and T. Feng, "A VLSI Interconnection Network for Multiprocessor Systems," *Digest Compcon Spring 1981,* pp. 294-298.
11. T. Feng, "Data Manipulating Functions in Parallel Processors and Their Implementations," *IEEE Trans. Computers,* Vol. C-23, No. 3, Mar. 1974, pp. 309-318.
12. V. Benes, *Mathematical Theory of Connecting Networks,* Academic Press, N.Y., 1965.
13. H. T. Kung, "The Structure of Parallel Algorithms," in *Advances in Computers,* Vol. 19, M. C. Yovits, ed., Academic Press, N.Y., 1980.
14. D. J. Farber and K. C. Larson, "The System Architecture of the Distributed Computer System—the Communications System," *Proc. Symp. Computer Comm. Networks and Teletraffic,* Brooklyn Polytechnic Press, Apr. 1972, pp. 21-27.
15. C. C. Reames and M. T. Liu, "A Loop Network for Simultaneous Transmission of Variable Length Messages," *Proc. Second Symp. Computer Architecture,* Jan. 1975, pp. 7-12.
16. S. I. Saffer et al., "NODAS—The Net Oriented Data Acquisition System for the Medical Environment," *AFIPS Conf. Proc.,* Vol. 46, 1977 NCC, pp. 295-300.
17. J. A. Harris and D. R. Smith, "Hierarchical Multiprocessor Organization," *Proc. Fourth Symp. Computer Architecture,* Mar. 1977, pp. 41-48.
18. G. H. Barnes et al., "The Illiac IV Computer," *IEEE Trans. Computers,* Vol. C-17, No. 8, Aug. 1968, pp. 746-757.
19. E. M. Aupperle, "MERIT Computer Network: Hardware Considerations," in *Computer Networks,* R. Rustin, ed., Prentice-Hall, Englewood Cliffs, N.J., 1972, pp. 49-63.
20. B. W. Arden and H. Lee, "Analysis of Chordal Ring Network," *IEEE Trans. Computers,* Vol. C-30, No. 4, April 1981, pp. 291-295.
21. H. Sullivan, T. R. Bashkow, and K. Klappholz, "A Large Scale Homogeneous, Fully Distributed Parallel Machine," *Proc. Fourth Symp. Computer Architecture,* Nov. 1977, pp. 105-125.
22. F. P. Preparata and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," *Comm. ACM,* Vol. 24, No. 5, May 1981, pp. 300-309.
23. H. S. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Trans. Computers,* Vol. C-20, No. 2, Feb. 1971, pp. 153-161.
24. T. Feng, *Parallel Processing Characteristics and Implementation of Data Manipulating Functions,* Rome Air Development Center report, RADC-TR-73-189, July 1973.
25. C. Wu and T. Feng, "On a Class of Multistage Interconnection Networks," *IEEE Trans. Computers,* Vol. C-29, No. 8, Aug. 1980, pp. 694-702.
26. C. Wu and T. Feng, "On a Distributed-Processor Communication Architecture," *Proc. Compcon Fall 1980,* pp. 599-605.
27. L. R. Goke and G. J. Lipovski, "Banyan Networks for Partitioning Multiprocessing Systems," *Proc. First Annual Computer Architecture Conf.,* Dec. 1973, pp. 21-28.
28. D. H. Lawrie, "Access and Alignment of Data in an Array Processor," *IEEE Trans. Computers,* Vol. C-24, No. 12, Dec. 1975, pp. 1145-1155.
29. K. E. Batcher, "The Flip Network in STARAN," *Proc. 1976 Int'l Conf. Parallel Processing,* Aug. 1976, pp. 65-71.
30. M. C. Pease, "The Indirect Binary n-Cube Microprocessor Array," *IEEE Trans. Computers,* Vol. C-26, No. 5, May 1977, pp. 548-573.
31. J. H. Patel, "Processor-Memory Interconnections for Multiprocessors," *Proc. Sixth Annual Symp. Computer Architecture,* Apr. 1979, pp. 168-177.
32. A. Waksman, "A Permutation Network," *J. ACM,* Vol. 9, No. 1, Jan. 1968, pp. 159-163.
33. A. E. Joel, Jr., "On Permutation Switching Networks," *B.S.T.J.,* Vol. 67, 1968, pp. 813-822.
34. D. C. Opferman and N. T. Tsao-Wu, "On a Class of Rearrangeable Switching Networks—Part I: Control Algorithm; Part II: Enumeration Studies of Fault Diagnosis," *B.S.T.J.,* 1971, pp. 1579-1618.
35. J. Lenfant, "Parallel Permutations of Data: A Benes Network Control Algorithm for Frequently Used Permutations," *IEEE Trans. Computers,* Vol. C-27, No. 7, July 1978, pp. 637-647.
36. T. Feng, C. Wu, and D. P. Agrawal, "A Microprocessor-Controlled Asynchronous Circuit Switching Network," *Proc. Sixth Annual Symp. Computer Architecture,* 1979, pp. 202-215.
37. Y-C. Chow, R. D. Dixon, T. Feng, and C. Wu, "Routing Techniques for Rearrangeable Interconnection Networks," *Proc. Workshop on Interconnection Networks,* Apr. 1980, pp. 64-69.
38. C. Clos, "A Study of Nonblocking Switching Networks," *Bell System Tech. J.,* Vol. 32, 1953, pp. 406-424.
39. C. D. Thompson, "Generalized Connection Networks for Parallel Processor Intercommunication," *IEEE Trans. Computers,* C-27, No. 12, Dec. 1978, pp. 1119-1125.
40. J. Gecsei, "Interconnection Networks from Three-State Cells," *IEEE Trans. Computers,* Vol. C-26, No. 8, Aug. 1977, pp. 705-711.
41. Y-C. Chow, R. D. Dixon, and T. Feng, "An Interconnection Network for Processor Communication with Optimized Local Connections," *Proc. 1980 Int'l Conf. Parallel Processing,* Aug. 1980, pp. 65-74.
42. W. A. Wulf and C. G. Bell, "C.mmp—A Multimicroprocessor," *AFIPS Conf. Proc.,* Vol. 41, 1972 FJCC, pp. 765-777.
43. T. Feng, *The Design of a Versatile Line Manipulator,* Rome Air Development Center report, RADC-TR-73-292, Sept. 1973.

44. W. W. Gaertner, *Design, Construction, and Installation of Data Manipulator,* Rome Air Development Center report, RADC-TR-77-166, May 1977, 80 pp.

45. S. E. Orcutt, "Implementation of Permutations Functions in an Illiac IV-Type Computer," *IEEE Trans. Computers,* Vol. C-25, No. 9, Sept. 1976, pp. 929-936.

46. C. D. Thompson and H. T. Kung, "Sorting on a Mesh-Connected Parallel Computer," *Comm. ACM,* Vol.20, No. 4, Apr. 1977, pp. 263-271.

47. D. Nassimi and S. Sahni, "Bitonic Sort on a Mesh-Connected Parallel Computer," *IEEE Trans. Computers,* Vol. C-28, No. 1, Jan. 1979, pp. 2-7.

48. C. Wu and T. Feng, "Universality of the Shuffle-Exchange Network," *IEEE Trans. Computers,* Vol. C-30, No. 5, May 1981.

49. D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching,* Addison-Wesley, Reading, Mass., 1973.

50. R. J. McMillen and H. J. Siegel, "MIMD Machine Communication Using the Augmented Data Manipulator Network," *Proc. Seventh Symp. Computer Architecture,* June 1980, pp. 51-58.

51. C. Wu and T. Feng, "The Reverse-Exchange Interconnection Network," *IEEE Trans. Computers,* Vol. C-29, No. 9, Sept. 1980, pp. 801-811; also *Proc. 1979 Int'l Conf. Parallel Processing,* pp. 160-174.

52. S. Anderson, "The Looping Algorithm Extended to Base 2^t Rearrangeable Switching Networks," *IEEE Trans. Comm.,* Vol. COM-25, No. 10, Oct. 1977, pp. 1057-1063.

53. G. Lev, N. Pippenger, and L. G. Valiant, "A Fast Parallel Algorithm for Routing in Permutation Networks," *IEEE Trans. Computers,* Vol. C-30, No. 2, Feb. 1981, pp. 93-100.

54. D. H. Lawrie, *Memory-Processor Conneciton Networks,* UIUCDCS-R-73-557, University of Ilinois, Urbana, Feb. 1973.

55. *Numerical Aerodynamic Simulation Facility Feasibility Study,* Burroughs Corporation, Mar. 1979.

56. U. V. Premkuma, R. Kapur, M. Malek, G. J. Lipovski, and P. Horne, "Design and Implementation of the Banyan Interconnection Network in TRAC," *AFIPS Conf. Proc.,* Vol. 49, 1980 NCC, pp. 643-653.

57. M. A. Franklin, "VLSI Performance Comparison of Banyan and Crossbar Communication Networks," *IEEE Trans. Computers,* Vol. C-30, No. 4, Apr. 1981, pp. 283-290.

58. P. G. Jansen and J. L. W. Kessels, "The DIMOND: A Component for the Modular Construction of Switching Networks," *IEEE Trans. Computers,* Vol. C-29, No. 10, Oct. 1980, pp. 884-889.

59. D. J. Kuck, "A Survey of Parallel Machine Organization and Programming," *Computing Surveys,* Vol. 9, No. 1, Mar. 1977, pp. 29-59. Also in *Proc. 1975 Sagamore Computer Conf. Parallel Processing,* pp. 15-39.

60. B. J. Smith, "A Pipelined, Shared Resource MIMD Computer," *Proc. 1978 Int'l Conf. Parallel Processing,* pp. 6-8.

61. J. B. Dennis, "Data Flow Supercomputers," *Computer,* Vol. 13, No. 11, Nov. 1980, pp. 48-56.

62. S. F. Lundstrom and G. Barnes, "A Controllable MIMD Architecture," *Proc. 1980 Int'l Conf. Parallel Processing,* pp. 19-27.

63. D. J. Kuck, "ILLIAC IV Software and Application Programming," *IEEE Trans. on Computers,* Vol. C-17, No. 8, Aug. 1968, pp. 758-770.

64. K. E. Batcher, "The Multi-Dimensional Access Memory in STARAN," *IEEE Trans. Computers,* Vol. C-26, No. 2, Feb. 1977, pp. 174-177.

65. D. H. Lawrie and C. Vora, "The Prime Memory System for Array Access," *Proc. 1980 Int'l Conf. Parallel Processing,* pp. 81-87.

66. A. Shimer and S. Ruhman, "Toward a Generalization of Two- and Three-Pass Multistage, Blocking Interconnection Networks," *Proc. 1980 Int'l Conf. Parallel Processing,* pp. 337-346.

67. T. Lang and H. S. Stone, "A Shuffle-Exchange Network with Simplified Control," *IEEE Trans. Computers,* Vol. C-25, No. 6, Jan. 1976, pp. 55-65.

68. H. T. Kung and D. Stevenson, "A Software Technique for Reducing the Routing Time on a Parallel Computer with a Fixed Interconnection Network," in *High Speed Computer and Algorithm Organization,* Academic Press, N.Y., 1977, pp. 423-433.

69. C. Wu and T. Feng, "A Software Technique for Enhancing Performance of a Distributed Computer System," *Proc. Compsac 80,* Oct. 1980, pp. 274-280.

70. H. J. Siegel, "The Theory Underlying the Partitioning of Permutation Networks," *IEEE Trans. Computers,* Vol. C-29, No. 9, Sept. 1980, pp. 791-801.

71. D. M. Dias and J. R. Jump, "Analysis and Simulation of Buffered Delta Networks," *IEEE Trans. Computers,* Vol. C-30, No. 4, Apr. 1981, pp. 273-282.

72. D. A. Padua, D. J. Kuck, and D. H. Lawrie, "High-Speed Multiprocessors and Compilation Techniques," *IEEE Trans. Computers,* Vol. C-29, No. 9, Sept. 1980, pp. 763-776.

73. G. H. Barnes, "Design and Validation of a Connection Network for Many-Processor Multiprocessing Systems," *Proc. 1980 Int'l Conf. Parallel Processing,* pp. 79-80.

74. C. Wu and T. Feng, "Fault Diagnosis for a Class of Multistage Interconnection Networks," *Proc. 1979 Int'l Conf. Parallel Processing,* pp. 269-278.

75. J. P. Shen and J. P. Hayes, "Fault Tolerance of a Class of Connecting Networks," *Proc. Seventh Symp. Computer Architecture,* 1980, pp. 61-71.

Tse-yun Feng is a professor in the Department of Computer and Information Science, Ohio State University, Columbus. Previously, he was on the faculty at Wayne State University, Detroit, and Syracuse University, New York. He has extensive technical publications in the areas of associative processing, parallel and concurrent processors, computer architecture, switching theory, and logic design, and has received a number of awards for his technical contributions and scholarship.

A past president of the IEEE Computer Society (1979-80), Feng was a distinguished visitor (1973-78), and has served as a reviewer, panelist, or session chairman for various technical magazines and conferences. He also initiated the Sagamore Computer Conference on Parallel Processing and the International Conference on Parallel Processing.

He received the BS degree from the National Taiwan University, Taipei, the MS degree from Oklahoma State University, Stillwater, and the PhD degree from the University of Michigan, Ann Arbor, all in electrical engineering.