Large-scale parallel processing systems

Parallel processing is widely perceived to be the way forward to greater computer power. Howard Jay Siegel, Thomas Schwederski, David G Meyer and William Tsun-yuk Hsu give examples of parallel algorithms, explain the different parallel interconnection strategies and highlight some microprocessor-based parallel computers.

Parallel processing is an area of growing interest to the computer science and engineering communities. This paper is an introduction to some of the concepts involved in the design and use of large-scale parallel systems. Parallel machines that are classified as SIMD (synchronous) and MIMD (asynchronous) systems, composed of a large number of microprocessors, are explored. Parallel algorithms are examined, using image smoothing, recursive doubling and contour tracing as examples. Single stage and multistage networks are discussed. The single stage Cube, PM21, Four Nearest Neighbor and Shuffle-Exchange networks are presented, and the multistage Cube network is described. Case studies of three microprocessor-based systems are given as examples of parallel machine designs, specifically the MPP SIMD machine, the Ultracomputer MIMD system, and the PASM SIMD/MIMD machine.

MODELS AND MODES OF PARALLEL COMPUTER SYSTEMS

Two basic models of parallel computer systems are shown in Figures 1 and 2. The processor-to-memory model (Figure 1) assumes N processors on one side of a bidirectional network and N memory modules on the other side. In the PE-to-PE model (Figure 2), PE i is connected to input i and output i of a unidirectional interconnection network, where each PE (processing element) is a processor paired with its own memory.

Two of the modes of parallelism described in a taxonomy originated by Flynn are the SIMD and MIMD modes. These modes will be described in terms of the PE-to-PE model. SIMD stands for single instruction, multiple...
data stream. An SIMD machine may consist of N PEs, an interconnection network which provides communications between PEs, and a single control unit (Figure 3). The control unit broadcasts instructions to all the PEs, and all enabled PEs execute the same instructions simultaneously. Thus, there is a single instruction stream. Each PE operates on its own data from its memory. Hence, there are multiple data streams. Examples of SIMD machines that have been built are the Illiac IV2, Staran3, MPPM, and the Connection Machine6. IBM’s GF117 is an SIMD machine currently under construction.

MIMD stands for ‘multiple instruction, multiple data’ stream. An MIMD machine may consist of N PEs linked by an interconnection network (Figure 2). Each PE stores and executes its own instructions and operates on its own data. Therefore, there are multiple instruction streams and multiple data streams. Examples of MIMD machines that have been built are the Illiac IV2, Staran3, MPPM, and the Connection Machine6. IBM’s GF117 is an SIMD machine currently under construction.

MSIMD machines. MSIMD machines are systems which can be reconfigured into a number of smaller independent SIMD machines of various sizes. Proposed MSIMD systems are MAP16, 17 and the original design for the Illiac IV2, 18. Partitionable SIMD/MIMD machines can be partitioned into smaller independent machines of different sizes working in SIMD or MIMD mode. Examples of partitionable SIMD/MIMD systems are PASM19, TRAC20 and DCA21.

ALGORITHMS FOR PARALLEL MACHINES

Designing algorithms for SIMD and MIMD machines requires the programmer to ‘think parallel’. Three examples of parallel algorithms for PE-to-PE systems are given in this section. Additional examples can be found in References 22–27.

Image smoothing

A simple window-based SIMD algorithm for image smoothing is considered. The algorithm has ‘I’ as an input image and ‘S’ as an output image. Assume both I and S contain 512 × 512 pixels (picture elements) for a total of 512² pixels each. Each pixel of I is an 8-bit unsigned integer representing one of 256 grey levels. The grey level of each pixel indicates how ‘dark’ that pixel is, where 0 means white and 255 means black. Each pixel in the smoothed image, S(i, j), is the average of the grey levels of I(i - 1, j), I(i - 1, j + 1), I(i, j - 1), I(i, j + 1), I(i + 1, j - 1), I(i + 1, j) and I(i + 1, j + 1). The top, bottom, left and right edge pixels of S are set to zero since their corresponding pixels in I do not have eight adjacent neighbours.

Assume an SIMD machine with N = 1024 PEs, logically arranged as an array of 32 × 32 PEs as shown in Figure 4a. Each PE stores a 16 × 16 sub-image block of the 512 × 512 image I; specifically, PE 0 stores the pixels in columns 0 to 15 of rows 0 to 15, PE 1 stores the pixels in columns 16 to 31 of rows 0 to 15, and so on. Each PE smoothes its own sub-image, with all PEs doing this simultaneously. At the edges of each 16 × 16 sub-image, data must be transmitted between PEs in order to calculate the smoothed value. The necessary data transfers are shown for PE J in Figure 4b. Transfers between different PEs can occur simultaneously, e.g. when PE J - 1 sends its upper right corner pixel to PE J, PE J can send its upper right corner pixel to PE J + 1, PE J + 1 can send its upper right corner pixel to PE J + 2, etc.
To perform a smoothing operation on a 512 X 512 image by the parallel smoothing of 1024 sub-image blocks of size 16 X 16, 256 parallel smoothing operations are executed. As described above, the neighbours of the sub-image edge pixels must be transferred from adjacent PEs. The total number of parallel data element transfers needed is (4 X 16) + 4 = 68: 16 for each of the top, bottom, left side, and right side edges, and four for the corners (see Figure 4b). The corresponding serial algorithm needs no data transfers between PEs, but 512^2 = 262 144 smoothing calculations must be performed. If no data transfers were needed, the parallel algorithm would be faster than the serial algorithm by a factor of 262 144/324 = 809. The inter-PE transfer time approximation is a conservative one. Thus, the overhead of the 68 inter-PE transfers that must be performed in the SIMD machine is negligible compared to the reduction in smoothing operations.

The last step is to set to zero the edge pixels of S. This creates an additional (although negligible) overhead which is to enable only the appropriate PEs when the zero values are stored for these edge pixels (only enabled PEs execute the instructions broadcasted). Serially, this would require (4 X 512) - 4 = 2044 stores, and in the SIMD machine only (4 X 16) = 64 parallel stores.

## Recursive doubling

Recursive doubling is a technique that is used to speed up parallel computations, such as summing all elements of a vector. In a serial computer, a variable would be initialized to the value of the first element of the vector, and all subsequent elements would be added to this variable. If the vector contained N elements, N - 1 additions are required. Assume that on an SIMD machine the vector is distributed among PEs, i.e. each of the N PEs holds one element. If the serial method were used on the parallel machine, one processor would accumulate the result and all other processors would send their element values to this processor. Clearly, this still requires N - 1 additions, all performed by the accumulating processor. Thus, no speed-up is achieved by using a parallel machine.

The recursive doubling procedure, for N a power of 2, is illustrated in Figure 5 for N = 8. In the first step, each odd numbered PE (i.e. 1, 3, 5, 7) sends its vector element to an even numbered PE (i.e. 1 to 0, 3 to 2, etc.). All even numbered PEs then add the value they received to their own vector element, forming N/2 (= 4) partial results. The odd numbered PEs do not participate and are disabled. In the next step this procedure is repeated. PE 2 forwards its partial result to PE 0, while PE 6 sends its result to PE 4. PEs 0 and 4 add the new value to their respective partial results, and all other PEs are disabled. In the last step, PE 4 sends its partial result to PE 0. PE 0 adds this value to its own partial result, and the sum is found. In general, log N additions and data transfer steps are required. This is a significant improvement over the serial algorithm.

### Contour extraction

The contour extraction algorithm is designed for a system capable of operating in both the SIMD and MIMD modes of parallelism. A contour in an image is the boundary of some feature of the image. Two algorithms from a contour extraction task are edge-guided thresholding (EGT) and contour tracing. The SIMD EGT algorithm is used to determine a set of optimal thresholds for requantizing the image, i.e. translating each pixel from a grey level value to either black (1) or white (0) depending whether it is above or below the threshold. The MIMD contour tracing algorithm uses the set of thresholds to segment the image and trace the contours. It is assumed that the image to be processed is distributed among the PEs as in the smoothing example.

There are two major steps in the SIMD EGT algorithm. First, the Sobel edge operator used to generate a 'Sobel image' in which grey levels indicate the magnitude of the gradient. Information from the Sobel image is then used to determine which threshold level is most appropriate for performing the segmentation. This is done for each PE's sub-image independently. Thus, the threshold levels may differ from one sub-image to the next. The window-based Sobel operator calculations and inter-PE communications used in the SIMD EGT algorithm are very similar to those discussed earlier for the image smoothing algorithm.

The system is switched to MIMD mode to execute the two phases of the contour tracing algorithm. In phase I, PEs segment their sub-image based on the threshold value each calculated using the EGT algorithm, i.e. each PE converts all pixels of its sub-image from grey level values to either black or white. Within each sub-image, all contours (collections of connected black pixels) are traced. Some will be closed (complete), others will extend into neighbouring sub-images (partial). In phase II, partial contours traced during phase I are connected.

Consider phase I in more detail. Each PE creates a segmented sub-image for a particular threshold T, and the contour tracing starts with each PE scanning the rows of its segmented sub-image beginning with the first pixel of the top row. Scanning stops when a start point of a new contour is found. A start point is a pixel with a value 1 which has a zero-valued neighbour to either or both sides. Contours are traced in either a clockwise or counter-clockwise direction and the Freeman direction codes of the 'chain' of pixels are recorded. When a pixel from an adjacent sub-image would be required to determine the next direction of the contour, a point of indecision is reached. Such a point is recorded as an end point, and the algorithm returns to the start point to trace the contour in

Vol 11 No 1 January/February 1987
the opposite direction until another point of indecision is reached. Closed contours that are contained within a sub-image are traced completely during phase I. No inter-PE data transfers are used.

Each PE attempts to connect its partial contours to those located in neighbouring PEs in phase II. PEs consider each partial contour’s end point in turn and try to find a possible extending contour in a neighbouring PE. Once such an extended contour is found, the process is repeated, if necessary, by following the contour to the next PE until the contour is closed or cannot be extended. Phase II is complete when all contours have been connected. Inter-PE data transfers are used in this phase.

The advantage of an SIMD/MIMD machine over an SIMD machine or an MIMD machine is demonstrated by the contour extraction task. The EGT algorithm can be executed more efficiently in SIMD mode, while the tracing algorithm can be executed more efficiently in MIMD mode. Thus, the ability of a PE to operate in either SIMD or MIMD mode allows the most appropriate type of parallelism to be employed by each algorithm in the task.

INTERCONNECTION NETWORKS

The problem of how to interconnect N processors and N memory modules to form an efficient parallel processing system, where N may be in the range $2^0$ to $2^{16}$, is a difficult one. The interconnection scheme must provide fast and flexible communications without unreasonable cost. A single shared bus, as shown in Figure 6, is not sufficient, since it is often desirable to allow all PEs to send data to other PEs simultaneously (e.g. as in the smoothing algorithm). The ideal situation would be to link directly each PE to every other PE, so that the system is completely connected, as shown for $N = 8$ in Figure 7. Unfortunately this is highly impractical for large N since it requires $N - 1$ unidirectional lines for each processor. For example, if $N = 2^{10}$, then $2^{10} \times (2^{10} - 1) = 1047552$ links would be needed.

The crossbar switch, shown in Figure 8, is another way to completely connect a system. In this example, the processors communicate through the memories. The network can be viewed as a set of intersecting lines, where interconnections between processors and memories are specified by the crosspoint switches at each line intersection. The difficulty with crossbar networks is that the cost of the network (the number of crosspoint switches) grows with $N^2$, which, given current technology, makes it infeasible for large systems.

Many different networks between the extremes of the single bus and the completely connected scheme have been proposed in the literature as more cost-effective approaches. There is no one network that is generally considered ‘best’. The cost-effectiveness of a particular network design depends on such factors as the computational tasks for which it will be used, the desired speed of interprocessor data transfers, the actual hardware implementation of the network, the number of processors in the system, and any cost constraints on the construction. A variety of networks which have been proposed are overviewed in numerous survey articles and books, e.g. References 34–44.

In this section we overview four single stage networks — Cube, PM21, Four Nearest Neighbour, and Shuffle-Exchange — and one multistage network, Generalized Cube. These networks can provide the communications needed in a parallel processing system consisting of a large number of PEs (e.g. $2^6$ to $2^{16}$) which are working together to perform a single overall task. In a single stage network, data items may have to be passed through the switches several times before reaching their final destinations. In a multistage network, generally one pass through the multiple (usually log2 $N$) stages of switches is sufficient to transfer the data items to their final destinations. The PE-to-PE model will be used in the section, but the material presented is also applicable to processor-to-memory systems.

Figure 6. Single shared bus used to provide communications for N devices

Figure 7. Completely connected system for $N = 8$

Figure 8. Crossbar switch connecting N processors to N memories
The partitionability of an interconnection network is the ability to divide the network into independent subnetworks of different sizes. Each subnetwork of size \( N' < N \) must have all of the interconnection capabilities of a complete network of that same type built to be of size \( N' \). Multiple-SIMD systems use partitionable interconnection networks to dynamically reconfigure the system into independent SIMD machines of varying sizes. The multiple-SIMD model is used as a framework for the partitioning analyses in this section. However, the results can also be used to partition MIMD and partitionable SIMD/MIMD machines. Three of the networks presented are partitionable.

**Single stage interconnection networks**

Consider a parallel system with \( N = 2^m \) PEs, numbered (addressed) from 0 to \( N - 1 \). An interconnection network can be described by a set of interconnection functions, where each interconnection function is a bijection (permutation) on the set of PE addresses. Interconnection functions represent inter-PE data transfers using mathematical mappings. When an interconnection function \( f \) is executed, PE \( i \) sends data to PE \( f(i) \). If a system is operating in SIMD mode, this means that every PE sends data to exactly one PE, and every PE receives data from exactly one PE (assuming all PEs are active). Otherwise, the data transfer from PE \( i \) to PE \( f(i) \) may occur only for a subset of the PEs in the system.

Let the binary representation of an arbitrary PE address \( P \) be \( P_m - 1 \ P_m - 2 \ldots P_1 P_0 \), let \( \overline{P} \) be the complement of \( P \), and let the integer \( n \) be the square root of \( N \). It is assumed that \( -j \mod N = N - j \mod N \), for \( j > 0 \); e.g., \( -5 \mod 16 = 11 \mod 16 \).

**Single stage Cube network**

The Cube network, shown in Figure 9, consists of \( m \) interconnection functions

\[
cube_i(p_{m-1} \ldots p_i + 1 p_i \ldots p_0) = \frac{p_{m-1} \ldots p_i + 1 p_i \ldots p_0}{p_{m-1} \ldots p_i + 1 p_i \ldots p_0}
\]

for \( 0 < i < m \). For example, \( \text{cube}_2(3) = 7 \). This network is called the Cube because when the PE addresses are considered as the corners of an \( m \)-dimensional cube,

Figure 9. Cube network for \( N = 8 \): a, \( \text{cube}_0 \) connections; b, \( \text{cube}_1 \) connections; c, \( \text{cube}_2 \) connections

using an appropriate labelling, this network connects each PE to its \( m \) neighbours, as shown in Figure 10 for \( N = 8 \). In terms of mapping source addresses to destinations, the Cube interconnection function \( \text{cube}_i \) maps the address \( P \) to \( f(P) \), i.e., the \( \text{cube}_i \) function sends data from PE \( P \) to PE \( \text{cube}_i(P) \).

Systems using the single stage Cube include the Cosmic Cube MIMD machine, the proposed CHoPP MIMD machine, the Intel iPSC MIMD machine, and the Connection Machine SIMD system. The Cube network provides the underlying structure for many multistage networks, such as the SW-banyan \((S = F = 2)\), Staran flips, indirect binary \( n \)-cube, Generalized Cube, Extra Stage Cube, and the BBN Butterfly. Various properties of the single stage Cube network are discussed in References 41, 46–48, 57–61.

When operating in SIMD mode the network settings for data transfers can be determined by means of a system control unit. The system control unit simply specifies the Cube function to be performed and the set of PEs which would be involved in the transfer, all as part of the SIMD program. A given data movement among the PEs may require a sequence of Cube functions to be executed.

In MIMD mode, inter-PE data transfers are less structured. Individual PEs generate their own data and propagate it through the network. There may be several ways to route a message through intermediate PEs. For example, for \( N = 8 \), to transfer data from PE 4 to PE 2, one possible sequence of \( \text{cube}_i \) transfers would be \( \text{cube}_2 \) and then \( \text{cube}_1 \), moving the data from PE 4 to 6, and then from PE 6 to 2. The order of performing the cube functions is not important. For example, \( \text{cube}_2 \) and then \( \text{cube}_1 \) would move data from PE 4 to 0, and then from PE 0 to 2.

The partitioning of the Cube network can be done based on any bit position of the PE addresses. If the use of the \( \text{cube}_i \) function is disallowed, all PEs with a ‘0’ in the \( i \)th bit of their addresses cannot communicate with PEs with a ‘1’ in the \( i \)th bit of their addresses. Two subnetworks of size \( N/2 \) are formed, each with \( m - 1 = \log_2(N/2) \) Cube functions. Each of these Cube subnetworks can then be further subdivided into smaller partitions. In general, the physical addresses of all the PEs in a partition of size \( 2^s \) must agree in the \( m - s \) bit positions not corresponding to the \( s \) Cube functions the partition will use for communications.

Figure 11 shows an example, for \( N = 8 \), of partitioning the PEs into two groups: GE, those with even physical numbers \((0, 2, 4, \text{and 6})\), and GO, those with odd physical numbers \((1, 3, 5, \text{and 7})\). Within each group, the PEs will be numbered logically from 0 to 3. Let logically numbered PEs 0, 1, 2, and 3 in GE be the physical PEs 0, 2, 4, and 6, respectively. Similarly, let logical PEs 0, 1, 2, and 3 of GO be the physical PEs 1, 3, 5, and 7, respectively. If the \( \text{cube}_0 \)
Figure 11. Partitioning a Cube network for N = 8: a, physical cube₁ (logical cube₀); b, physical cube₂ (logical cube₃)

function is not used, then the two groups are independent and cannot communicate. This is because all PEs in GE have a '0' in the lower order physical address bit position and all PEs in GO have a '1' in that position. The only way for a PE in GE to communicate with a PE in GO is by using the cube₀ connection. Using the logical numbering of the PEs, the physical cube₁ connections act as logical cube₀ connections and the physical cube₂ connections act as logical cube₁ connections. In a multiple-SIMD environment, if the physically even numbered PEs can be connected to one control unit and the physically odd numbered PEs can be connected to a second control unit, the two partitions can operate independently with complete Cube networks of size four. In an MIMD environment, each partition can operate as an independent virtual MIMD machine.

Single stage Plus-Minus 2¹ (PM2₁) network
The Plus-Minus 2¹ (PM2₁) network, shown in Figure 12, consists of 2m interconnection functions

\[
PM2_{+i}(P) = P + 2^i \text{ modulo } N \\
PM2_{-i}(P) = P - 2^i \text{ modulo } N
\]

for \(0 < i < m\). For example, \(PM2_{+2}(4) = 8\) if \(N > 8\). Since \(P + 2^{m-1} = P - 2^{m-1} \text{ modulo } N\), \(PM2_{+2} - PM2_{-2}\) functions are equivalent. This network is called the Plus-Minus 2¹ since, in terms of mapping source addresses to destinations, it can add or subtract \(2^i\) from the PE addresses, i.e. it allows PE \(P\) to send data to any one of PE \(P + 2^i\) or PE \(P - 2^i\), arithmetic modulo \(N\), \(0 < i < m\).

Figure 12. PM2₁ network for N = 8: a, PM₂₊₀ connections; b, PM₂₊₁ connections; c, PM₂₊₂ connections

Networks similar to the PM2₁ are used in the Novel Multiprocessor Array, the Omen computer, and the SIMDA machine. The data manipulator, ADM, IADM and gamma multistage networks are based on the PM2₁ connection pattern. Various properties of the single stage PM2₁ network are discussed in References 41, 46–48, 57, 61, 68–70.

In SIMD mode, network control can be achieved by means of a system control unit, as in the Cube network. Data transfers in MIMD mode can be implemented based on the difference between the source and destination addresses. For example, for \(N = 16\), to route data from PE 1 to PE 15, 14 has to be added to 1. One sequence of PM2₁ functions that would perform this transfer is \(PM2_{+1}, PM2_{+2}\) and \(PM2_{+3}\), and the message would go through PEs 1, 3, 7 and 15. As in the Cube network, the order of performing the PM2₁ functions is not important. Also, a different set of PM2₁ functions may result in the same routing if the sum of the functions is equal to the difference between the source and destination addresses. For example, it is also possible to use \(PM2_{-1}\) to route data from PE 1 to PE 15.

To partition the PM2₁ network into two PM2₁ subnetworks of size \(N/2\), use of the \(PM2_{+0}\) interconnection function must be disallowed. This subdivides the PEs into an even-numbered group and an odd-numbered group, and PEs in one group cannot communicate with PEs in the other group. This is because the only way for PEs in the odd-numbered group to communicate with a PE in the even-numbered group is to use \(PM2_{+0}\). Each of the two subnetworks formed has \(2(m-1) = 2\log_2(N/2)\) PM2₁ functions. Each of these subnetworks can be further divided into smaller subnetworks by disallowing \(PM2_{+1}, PM2_{+2}\), and so on, in order. In general, the physical addresses of all of the PEs in a partition of size \(2^s\) must agree in their low order \(m-s\) positions. The physical \(PM2_{+r}\) connections are used by the partition as logical connections \(PM2_{+r} - (m-s)\) mod \(m\), \(s < r < m\). An example of partitioning a size eight PM2₁ network into two subnetworks of size four is shown in Figure 13. The two groups and the logical numbering for each are the same as those for the Cube partitioning example. The physical \(PM2_{+1}\) interconnection functions act as the logical \(PM2_{+0}\) functions for each partition, and the physical \(PM2_{+2}\) function acts as the logical \(PM2_{+1}\). In a multiple-SIMD environment, if the physically even numbered PEs can be connected to one control unit and the physically odd numbered PEs can be connected to a second control unit, the two partitions can operate independently with complete Cube networks of size four. In an MIMD environment, each partition can operate as an independent virtual MIMD machine.

Figure 13. Partitioning a PM2₁ network for N = 8: a, physical \(PM2_{+1}\) (logical \(PM2_{+0}\)); b, physical \(PM2_{+2}\) = physical \(PM2_{-2}\) (logical \(PM2_{+1}\))
odd numbered PEs can be connected to a second control unit, the two partitions can operate independently with complete PM21 networks of size four. The same partitioning can be used in an MIMD environment to create two independent virtual MIMD machines.

**Single stage Four Nearest Neighbor network**

The Four Nearest Neighbor (FNN) network, shown in Figure 14, consists of four interconnection functions

\[
\begin{align*}
\text{FNN}_{+1}(P) &= P + 1 \mod N \\
\text{FNN}_{-1}(P) &= P - 1 \mod N \\
\text{FNN}_{+n}(P) &= P + n \mod N \\
\text{FNN}_{-n}(P) &= P - n \mod N
\end{align*}
\]

where \(n\) (the square root of \(N\)) is assumed to be an integer.

For example, if \(N = 16\), \(\text{FNN}_{+2}(2) = 6\). This network allows PE \(P\) to send data to any one of PEs \(P + 1, P - 1, P + n\) or \(P - n\), arithmetic modulo \(N\). The FNN network is a subset of the PM21 network, i.e. \(\text{FNN}_{+1} = \text{PM21}_{+0}\) and \(\text{FNN}_{+n} = \text{PM21}_{+\frac{n}{2}}\).

The FNN type of network was used in the Illiac IV SIMD machine, and is included in the MPP and DAP SIMD systems. It is similar to the eight nearest neighbour network used in the CLIP72 machine. The 'mesh' interconnection network is like the FNN network except there are no 'wrap-around' connections (see Figure 14). Various properties and capabilities of the FNN network are discussed in References 2, 76, 77.

The operation of the FNN network in SIMD mode is similar to that for the Cube. To do data transfers in MIMD mode, a sequence of FNN functions which add up to the difference between the source and destination addresses is used to transfer the data. For example, for \(N = 64\), for PE 2 to transmit data to PE 28, one possible sequence would be to execute \(\text{FNN}_{+3}\) three times and \(\text{FNN}_{+2}\) twice. The route taken would be through PEs 2, 10, 18, 26, 27, and 28.

As in the PM21 network, the order of performing the transfers is not important, and it is also possible to find different sets of FNN functions which would perform the same routing, e.g. going from PE 2 to PE 28 could also be accomplished by executing \(\text{FNN}_{-n}\) five times and \(\text{FNN}_{+1}\) twice, and the data would go through PEs 2, 58, 50, 42, 34, 26, 27, and 28.

The FNN network cannot be partitioned into independent subnetworks, each of which has the properties of a complete FNN network. To have a subnetwork that has the same properties as the FNN, each PE must have four interconnection functions. Allowing each PE to use all four functions, however, results in the full network.

**Single stage Shuffle-Exchange network**

The Shuffle-Exchange network, shown in Figure 15, consists of a shuffle function and an exchange function. The shuffle is defined by

\[
\text{shuffle}(P_m-1, P_m-2, \ldots, P_1, P_0) = P_{m-2}, P_{m-3}, \ldots, P_1, P_0
\]

and the exchange is defined by

\[
\text{exchange}(P_m-1, P_m-2, \ldots, P_1, P_0) = P_{m-1}, P_m-2, \ldots, P_1, P_0
\]

For example, shuffle \((6) = 5\) for \(N = 8\). Shuffling a PE's address is equivalent to taking the left cyclic end-around shift of its binary representation. The name 'shuffle' has its origin in shuffling cards, by perfectly intermixing two halves of a deck, as shown in Figure 16. The exchange function is equivalent to cube0.

The shuffle is included in the networks of the Omen and RAP systems. The multistage omega network is a series of \(m\) Shuffle-Exchanges. Mathematical properties of the shuffle are discussed in References 80 and 81. Features of the single stage Shuffle-Exchange network are discussed in References 41, 46–48, 57, 58, 60, 61, 68, 76, 82–87.

![Figure 15](image)

---

Figure 14. **FNN network for \(N = 16\)**. Vertical lines are \(+ N^{1/2}\) and \(- N^{1/2}\). Horizontal lines are \(+1\) and \(-1\). Lower-case letters indicate wraparound connections.

![Figure 16](image)

---

Figure 16. Perfectly shuffling a deck of eight cards. \(S\) stands for 'shuffle'.
The network’s operation in SIMD mode is similar to that of the Cube. In MIMD operations, data routing is accomplished by a sequence of shuffles and exchanges. For example, for \( N = 16 \), one way to transmit data from PE 11 (1011) to PE 0 (0000) is to perform one shuffle (1011 to 0111), an exchange (0111 to 0110), two shuffles (0110 to 1100 to 1001), an exchange (1001 to 1000), one shuffle (1000 to 0001), and an exchange (0001 to 0000). Note that even though the Shuffle-Exchange contains only two interconnection functions, it takes at most \( 2m \) moves (at most \( m \) shuffles and \( m \) exchanges) to go from any source to any destination.

The Shuffle-Exchange network cannot be partitioned into independent subnetworks. The reasoning is similar to that for the FNN network, since the Shuffle-Exchange network also has a constant number of interconnection functions.

### Multistage Cube network

The multistage cube family of networks is being studied and used by many research teams designing large-scale parallel processing systems. As a representative of this family, the Generalized Cube (GC) network is described in this section. This topology is equivalent to that used by the omega, indirect binary n-cube, Staran, and SW-banyan (\( F = S = 2 \)) networks. Other networks in this family include the delta and baseline. This type of network is used or proposed for use in Staran, DISP, PASM, Ultracomputer, BBN Butterfly, the Burroughs Flow Model Processor for the Numerical Aerodynamic Simulator, the IBM Research Parallel Processor Prototype (RP3), and data flow machines. This network can operate in the SIMD, multiple-SIMD, MIMD, and partitionable SIMD/MIMD modes of parallelism.

An \( N \) I/O GC network is shown in Figure 17. It has \( m = \log_2 N \) stages, where each stage consists of a set of \( N \) lines (links) connected to \( N/2 \) interchange boxes. Each interchange box is a two-input, two-output switch, and can be set to one of four legitimate states, as shown in Figure 18. The links are labelled from 0 to \( N - 1 \), and the labels of the links entering the upper and lower box inputs have the same labels as the upper and lower outputs, respectively. Each interchange box will be controlled independently through the use of routing tags.

To set the GC to perform one-to-one connections, the straight and swap box settings are used. When an interchange box in stage \( i \) is set to swap it is implementing the cube\( i \) interconnection function. For example, in Figure 19 the path from source \( S = 2 \) to destination \( D = 4 \) in the GC network for \( N = 8 \) is shown. The interchange boxes through which the data from source 2 travels are set to swap in stages 2 and 1 (cube\( 2 \) and cube\( 1 \)), and straight in stage 0. In general, to go from a source \( S = s_m \ldots s_0 \) to a destination \( D = d_m \ldots d_0 \), the stage \( i \) box in the path from \( S \) to \( D \) must be set to swap if \( s_i \neq d_i \) and to straight if \( s_i = d_i \). There is only one path from a given source to a given destination, since only stage \( i \) can
determine the \( i \)th bit of the destination address. To perform a broadcast (one-to-many) connection, the lower and/or upper broadcast states of interchange boxes are also used. An example is shown in Figure 20, where input 5 broadcasts to outputs 2, 3, 6, and 7.

In SIMD mode, permutation connections, where each input is connected to a single distinct output, are used. Since each connection in a permutation is one-to-one, only the straight and swap box settings are used (see Figure 21 for example). Since each of the \( Nm/2 \) boxes can be set to straight or swap, the GC can do \( 2^{Nm/2} \) of the \( N! \) different permutations, where for \( N > 8 \), \( 2^{Nm/2} \ll N! \). However, the GC can do most permutations that are important in SIMD processing.

The GC network can be controlled by using a routing tag as a header on each message. This allows network control to be distributed among the PEs. An \( m \)-bit tag for one-to-one (non-broadcast) connections or permutations can be computed from the source and destination address. Let \( S = s_m \ldots s_1 s_0 \) be the source address and \( D = d_m \ldots d_1 d_0 \) be the destination address. Then the tag is \( T = t_m \ldots t_1 t_0 = S \oplus D \) (\( \oplus \) is 'exclusive-OR'\(^{45} \)), \( t_i = 1 \) means \( s_i \neq d_i \), \( t_i = 0 \) means \( s_i = d_i \). An interchange box in the network at stage \( i \) need only examine \( t_i \). If \( t_i = 1 \), a swap is performed (i.e. cube is performed) and if \( t_i = 0 \), the straight connection is used. For example, if \( N = 8 \), \( S = 2 = 010 \), and \( D = 4 = 100 \), then \( T = 110 \), and the corresponding stage settings are swap, swap, and straight, as shown in Figure 19. Since each source generates its own tag, it is possible that a conflict will occur in the network, e.g. the tag on the upper input link of a box specifies a swap while the tag on the lower input specifies a straight. In a situation like this, one message must wait until the other has completed its transmission. Both requests cannot be serviced simultaneously. If broadcasts are allowed, then \( 2m \) bits are used in the tag\(^{45} \).

The partitioning of the GC is related to the partitioning of the single stage Cube network which was described previously. First consider partitioning a GC of size \( N \) into two independent subnetworks, each of size \( N/2 \). There are \( m \) ways to do this, each being based on a different bit position of the I/O port addresses (i.e. a different Cube interconnection function). One way is to force all boxes in stage \( m - 1 \) to the straight state (i.e. disallow the use of cube\(_{m-1} \)). This would form two subnetworks, one consisting of those I/O ports with a 0 in the high order bit position of their addresses and the other consisting of those ports with a 1 in the high order bit position. These two groups could communicate with each other only by using the swap setting in stage \( m - 1 \) (i.e. the cube\(_{m-1} \) function). By forcing this stage to straight, the subnetworks are independent and have full use of the rest of the network (stages \( m - 2 \) to 0, corresponding to cube\(_{m-2} \) to cube\(_0 \)). This is shown for \( N = 8 \) in Figure 22.

Since each subnetwork is a GC, each can be further subdivided. In Figure 22, subnetwork B can be further divided using the middle bit position, forming two smaller subgroups of two PEs each, as shown in Figure 23. Thus, a size \( N/2 \) subnetwork can be divided into two size \( N/4 \) subnetworks by setting all the stage \( i \) boxes in the size \( N/2 \) subnetwork to straight, for any \( i, 0 < i < m \), as long as...
stage $i$ was not used to create the size $N/2$ subnetworks (as stage $m - 1$ was above). This process of dividing subnetworks into independent halves can be repeated to create any size subnetwork from one to $N/2$. The only constraints are that the size of each subnetwork must be a power of two, the physical addresses of the I/O ports of a subnetwork of size $2^m$ must all agree in any fixed set of $m - s$ bit positions, and each I/O port can belong to at most one subnetwork.

For a partition of size $2^s$, to map logical stage numbers to physical stage numbers, once the network has been partitioned, follow any path within the partition from input to output. The first stage containing a box not forced to straight acts as logical stage $s - 1$, the next stage containing a box not forced to straight acts as $s - 2$, etc. For example, consider the $A$ subnetwork in Figure 22. By following link number 1 it can be seen that physical stage 1 becomes logical stage 1 and physical stage 0 becomes logical stage 0.

Partitioning can be enforced using the routing tags and an $m$-bit partitioning mask. The mask is set to 0 in all the bits that correspond to the stages which are forced to straight in that partition, and all other bits are set to 1. The routing tag is logically ANDed with the mask to force corresponding stages to straight.

One implementation aspect of all multistage networks is the way in which paths through the network are established and released. Circuit switching implies the establishing of a complete path between the source and the destination. Once such a path is established, subsequent transfers along the path can be accomplished at high speed and efficiently. A packet-switched network, often referred to as message-switched, does not establish a complete path from source to destination. Instead, data and routing information are collected into a packet, and this packet makes its way from stage to stage, releasing links and interchange boxes immediately after using them. Thus, a packet uses only one interchange box at a time.

The next section overviews three parallel processing system designs. One of them uses an FNN-type network and the other two use multistage cube-type networks.
the bit currently on the data bus. The result is left in the P register. The adder sums the content of the A and P registers and the carry in the C register; it leaves the least significant bit of the result in the B register, and the carry in the C register. The length of the shift register can be programmed to be 2, 6, 10, 14, 18, 22, 26 and 30 bits. The A and B registers can be thought of as additional elements of the shift register; the total shift register length is thus a multiple of four. These simple components are sufficient to perform all basic arithmetic operations: addition, subtraction, multiplication, division, and floating point operations. In many applications (such as in the smoothing example), PEs have to be disabled. To disable PEs, the G register is provided. If a masked instruction is executed, only those PEs with a 1 in their G register participate in the execution. A comparator speeds up certain algorithms, e.g., normalization of floating point numbers. Global minimum value and maximum value searches and other global operations can be performed by the sum-OR tree, which is a tree of inclusive-OR gates. It has inputs from all PEs, and its output is connected to the array control unit.

Data enters and leaves the ARU through the S-registers. Each PE has one 1-bit S-register, and together, the $128^2$
S-registers form a plane that can shift data. Each S-register operates independently from the rest of the PE. An S-register sends its content to its right neighbour and receives the content from the S-register that is its left neighbour, i.e., 128-bit columns of data are shifted across the S-register plane. The 128 bits of data that are simultaneously shifted into the S-registers of the leftmost column of PEs originate from the input switches (Figure 24). This shifting is done without interrupting normal processing. After 128 shift cycles, a complete 128 × 128 bit plane is loaded into the S-registers. Processing is interrupted for a single 100 ns cycle and this complete data plane of 128 × 128 bit is transferred from the S-registers into the PE memories, each S register sending one bit to its PE. If necessary, all PEs can simultaneously move a memory bit from an old plane into their associated S-register in a second cycle. Then a new plane of 128 × 128 bit can be shifted in and the old plane can be shifted out at the same time. The data that is shifted out of the rightmost PEs is accepted by the output switches, one column (128 bit) per cycle (Figure 24). This method of overlapping shift in and shift out provides a very efficient way to get data into and out of the MPP PEs.

The staging memory can interface the ARU with magnetic tapes, discs, terminals, line printers, and the external host computer. Data arriving from the outside world, e.g., from an image sensor, will be organized as a stream of pixel values. The same is true for output, e.g., the output of a smoothed image to an image display device usually needs to be formatted as a stream of pixel values. The PEs on the other hand have to be loaded with a bit slice from all 128 × 128 pixel values at once. The staging memory (up to 40 Mbyte capacity) facilitates this reformatting process for the input and output data.

NYU Ultracomputer

As an example of a large scale MIMD system, the proposed design of the NYU Ultracomputer machine is described. Ultracomputer could have as many as 4096 processors connected to 4096 memories through a packet-switched multistage cube interconnection network; thus, the basic Ultracomputer configuration is of the processor-to-memory type. Figure 26 shows the basic organization of the Ultracomputer. The N processors are not necessarily identical as in many other designs; at least some of them could be special-purpose machines for FFT computations or matrix multiplications, or they could be I/O processors interfacing the system with the outside world. Due to the large number of processors (up to 4096), most of them would very likely be identical one-chip microprocessors with some local memory (cache). The processors interface to the interconnection network through a processor network interface (PNI), and the network is connected to each of the N memories through a memory network interface (MNI).

The most innovative feature of the Ultracomputer is the fetch-and-add instruction, denoted F&A(V, e), which provides a simple and efficient way for processor synchronization and is supported by hardware. V is an integer variable in shared memory and e is an integer expression. The execution of an F&A instruction returns the old value of V and adds to V the value of e. The instruction is indivisible, i.e., while the F&A operation is being performed on V by one processor, no other processor can access V. If two processors A and B try to execute a F&A operation on the same variable at the same time, the effect will be as if one processor preceded the other in time. For example processor A can receive the old value of V and V will be incremented by the value of e of processor A. Processor B will then receive the new value of V, and V is incremented by the value of e of processor B.

The following example illustrates how the F&A operation can be used to solve a variety of synchronization problems. Suppose N processors are to be used to perform a complex operation on an N element array, e.g., calculate the tangent. The first task is to associate one processor with each of the N array elements. This is easily accomplished by initializing a shared variable V1 to zero, and then having each of the N processors execute an F&A(V1, 1). Each processor will receive a distinct value for V1 in the range from 0 to N - 1; thus each processor knows the array element it is supposed to handle. Only after all processors have completed their task, processing that utilizes the result of the computations can resume. To ensure this, a second shared variable V2 is set to N - 1 before the parallel execution starts. After a processor has completed its task, it executes an F&A(V2, -1). If the returned value is not 0, other processors are still working on their assigned array element. Therefore the processor simply terminates working on this problem and can be reassigned to other processes. The processor that receives a value of 0 is the last one to complete the task, and this processor continues execution of the program.

The F&A operation is implemented by hardware provided at two places: adders in the MNI and intelligent interchange boxes in the network. When an F&A(Y, e) has passed through the network and reaches the MNI, the value of Y is fetched from memory. This value is passed back through the network to the source of the request, and the value of Y is added to e. The sum is then written back to memory, resulting in the desired F&A operation. An adder and a local memory are associated with each interchange box in the network. Whenever two requests (from different processors) for the same memory location, e.g., F&A(X, e1) and F&A(X, e2), arrive at a box, the box stores e1 in its memory, adds e1 and e2, and passes a request for F&A(X, e1 + e2) on to the memory. The

Figure 26. Block diagram of the NYU Ultracomputer

Microprocessors and Microsystems
memory updates the variable X by adding e1 + e2 to it, and passes the original value of X back to the network. When this return message reaches the above box, the box passes the original value of X back to the location that requested F&A(X, e1), adds the stored value of e1 to the value of X, and passes the result back to the second source. If requests to the same memory location that were combined as described above meet at another box, they can be combined further, thus combining previously combined requests. In this manner, multiple requests to the same memory location can be served in the time of a single request. Interprocessor coordination is therefore not serialized.

It was estimated that a 4096 PE Ultracomputer could be packaged into a 1.5 x 1.5 x 3.0 m enclosure using 1990 technology. This small physical size is possible only through custom-designed processors, and the use of large-scale integration in the interconnection network. Due to such integration, the complex operations required for a network capable of F&A do not impede the network speed significantly. Any on-chip delay will be dominated by chip-to-chip delay. A 64-processor prototype is currently under development.

**PASM parallel processing system**

PASM is a partitionable SIMD/MIMD machine, using the PE-to-PE configuration, being designed at Purdue University to include over a thousand PEs. A 30-processor prototype is under construction. A block diagram of the basic components of PASM is shown in Figure 27. The parallel computation unit (Figure 28) contains N = 2^m PEs and an interconnection network. The PE processors are sophisticated microprocessors that perform the actual SIMD and MIMD operations. In the PASM prototype, Motorola MC68010 CPUs are employed. The PE memory modules are used by the processors for data storage in SIMD mode and both data and instruction storage in MIMD mode. The N PEs are numbered from 0 to N - 1. A pair of memory units is used for each module to allow data to be moved between one memory unit and secondary storage (the memory storage system), while the processor operates on data in the other memory unit. Two types of multistage interconnection networks are being considered for PASM: the Generalized Cube and the Augmented Data Manipulator. A fault tolerant variation of the Generalized Cube network, the Extra Stage Cube (ESC) network, is included in the prototype.

A set of processors called the micro controllers (MCs) act as the control units for the PEs in SIMD mode and orchestrate the activities of the PEs in MIMD mode. There are Q = 2^q MCs, physically addressed (numbered) from 0 to Q - 1. Each MC controls N/Q PEs. PASM is being designed for N = 1024 and Q = 32 (N = 16 and Q = 4 in the prototype). The MCs are multiple control units needed in order to have a partitionable SIMD/MIMD system. Each MC includes a memory module, which consists of a pair of memory units so that memory loading and computations can be overlapped. In SIMD mode, each MC fetches instructions and common data from its memory module, executes the control flow instructions (e.g. branches) and broadcasts the data processing
instructions to its PEs. In MIMD mode, each MC gets instructions and common data for coordinating its PEs from its memory. The programs for the MCs are contained in control storage.

When constructing partitions (i.e. virtual SIMD/MIMD machines) in PASM, the combining rule is that all PEs in a partition of size $2^r$ must agree in their $m - s$ low-order bit positions. The physical addresses of the N/Q processors which are connected to an MC must all have the same low-order $q$ bits so that they are in the same partition. The value of these low-order $q$ bits is the physical address of the MC. A virtual SIMD machine of size RN/Q is obtained by having $R = 2^r$, $0 < r < q$, MCs use the same instructions and synchronizing the MCs. The physical addresses of the MCs must have the same low-order $q - r$ bits so that all PEs in the partition have the same low-order $q - r$ physical address bits. A virtual MIMD machine of size RN/Q is obtained similarly. In MIMD mode, the MCs may be used to help coordinate the activities of their PEs. Q is the maximum number of partitions allowable, and N/Q is the size of the smallest partition.

The memory storage system provides secondary storage space for the PEs in the parallel computation unit. It contains data files for SIMD mode, and data and control instructions for MIMD mode. It consists of N/Q independent memory storage units, numbered from 0 to N/Q - 1. Each memory storage unit is connected to Q PE memory modules and includes a processor for file management. For $0 < i < N/Q$, memory storage unit $i$ is connected to those PE memory modules whose physical addresses have the value $i$ in their $m - q$ high-order bits. Recall that, for $0 < k < Q$, MC $k$ is connected to those PEs whose physical addresses have the value $k$ in their $q$ low-order bits. This is shown for $N = 32$ and $Q = 4$ in Figure 29. Consider a virtual machine of RN/Q PEs, $R = 2^r$, $0 < r < q$, where the PEs are logically numbered from 0 to RN/Q - 1, using the $r + m - q$ high-order bits of their physical addresses as their logical addresses. To load such a virtual machine requires only $R$ parallel block moves if the data for the PE memory module whose high-order $m - q$ logical address bits equal $i$ is loaded into memory storage unit $i$. This is true no matter which group of R MCs (which agree in their low order $q - r$ physical address bits) is chosen. This is done by making use of the full memory bandwidth available. The transferring of files between the memory storage system and the PEs is controlled by the memory management system, which is composed of a separate set of microprocessors (four in the prototype) dedicated to performing tasks in a distributed fashion.

For example, consider Figure 29, and assume a virtual machine of size 16 is desired. The data for the PE memory modules whose logical addresses are 0 and 1 are loaded into memory storage unit 0, for memory modules 2 and 3 into unit 1, etc. Assume the partition of size 16 is chosen to consist of the PEs connected to MCs 1 and 3. The memory storage units first simultaneously load PE memory modules physically addressed 1, 5, 9, 13, 17, 21, 25, and 29 (logically addressed 0, 2, 4, 6, 8, 10, 12, and 14), and then simultaneously load PE memory modules physically addressed 3, 7, 11, 15, 19, 23, 27, and 31 (logically addressed 1, 3, 5, 7, 9, 11, 13, and 15). No matter which pair of MCs is chosen (i.e. MCs 1 and 3, or MCs 0 and 2), only two parallel block loads are needed.

The system control unit is responsible for the overall coordination of the other components of PASM. The types of tasks the system control unit will perform include

CONCLUSIONS

Parallel processing is one of the ways being investigated to achieve higher performance computers than are presently available, and is currently one of the most actively researched areas in computer science. In this paper, we

CONCLUSIONS

Parallel processing is one of the ways being investigated to achieve higher performance computers than are presently available, and is currently one of the most actively researched areas in computer science. In this paper, we
have overviewed some of the concepts involved in the design and use of parallel processing systems. We placed emphasis on SIMD and MIMD architectures and related topics like algorithms and interconnection networks for large-scale systems. This paper is a brief introduction and thus does not cover all topics of interest; there are other concepts of parallelism, such as data flow machines, and other important issues, such as parallel programming languages. Sources which provide more information on parallel processing concepts and systems include References 34, 37, 38, 41, 44, 98–104.

REFERENCES


3 Batcher, K E 'STARAN parallel processor system hardware' AFIPS 1974 NCC (May 1974) pp 405–410


8 Swan, R J, Fuller, S and Siewiorek, D P 'Cm*': a modular microprocessor 'AFIPS 1977 NCC (June 1977) pp 637–644


10 Jones, A K, Chancellor, R J Jr., Durham, I, Feller, P and Schwans, K 'Software management of Cm* — a distributed multiprocessor' AFIPS 1977 NCC (June 1977) pp 657-663

11 Crowther, W, Goodhue, J, Thomas, R, Milliken, W and Blackadar, T 'Performance measurements on a 128-node butterfly parallel processor' 1985 Int. Conf. on Parallel Processing (August 1985) pp 531–540


13 Hagan, M T, Demuth, H B and Singgih, P H 'Parallel signal processing on the HEP' 1985 Int. Conf. on Parallel Processing (August 1985) pp 599–606


26 Yoder, M A and Jamieson, L H 'Simulation of a highly parallel system for word recognition' 1985 Int. Conf. on Acoustics, Speech and Signal Processing (March 1985) pp 1449–1453

27 Yoder, M A and Siegel, L J 'Dynamic time warping algorithms for SIMD machines and VLSI processor arrays' 1982 Int. Conf. on Acoustics, Speech and Signal Processing (May 1982) pp 1274–1277


31 Duda, R O and Hart, P E Pattern classification and scene analysis John Wiley, New York, NY, USA (1973)

32 Freeman, H 'Techniques for the digital computer analysis of chain-encoded arbitrary plane curves' Proc. NEC Vol 17 (October 1961) pp 421–432

Vol 11 No 1 January/February 1987 17
34 Baer, J-L Computer systems architecture Computer Science Press, Potomac, MD, USA (1980)
35 Broomell, G and Heath, J R 'Classification categories and historical development of circuit switching topologies' ACM Comput. Surveys Vol 15 (June 1983) pp 95–133
37 Hockney, R W and Jesshope, C R Parallel computers Adam Hilger, Bristol, UK (1981)
46 Smith, S D and Siegel, H J 'Recirculating, pipelined, and multistage SIMD interconnection networks' 1978 Int. Conf. on Parallel Processing (August 1978) pp 206–214
50 'A new direction in scientific computing' Order No. 28009-001 Intel Corp. (1985)
52 Batcher, K E 'The flip network in STARAN' 1976 Int. Conf. on Parallel Processing (August 1976) pp 65–71
60 Nassimi, D and Sahni, S 'Parallel permutation and sorting algorithms and a new generalized connection network' J. ACM Vol 29 (July 1982) pp 642–667
64 Wester, A H 'Special features in SIMD' 1972 Sagamore Computer Conf. on Parallel Processing (August 1972) pp 29–40
69 Siegel, H J and Smith, S D 'An interconnection network for multimicroprocessor emulator systems' 1st Int. Conf. on Distributed Computing Systems (October 1979) pp 772–782
Vol 11 No 1 January/February 1987


81 Johnson, P B ‘Congruences and card shuffling’ American Mathematical Monthly Vol 63 (December 1956) pp 718–719


90 Batcher, K E ‘STARAN series E’ 1977 Int. Conf. on Parallel Processing (August 1977) pp 140–143

91 Filip, A E ‘A distributed signal processing architecture’ 3rd Int. Conf. on Distributed Computing Systems (October 1982) pp 49–55


Howard Jay Siegel is a professor and director of the PASM Parallel Processing Laboratory in the School of Electrical Engineering at Purdue University. Dr Siegel received BS degrees in electrical engineering and in management from the Massachusetts Institute of Technology (MIT), MA, USA, in 1972, and the MA and MSE degrees in 1974 and the PhD degree in 1977, all in electrical engineering and computer science, from Princeton University, NJ, USA. He has published over 100 technical papers on parallel/distributed computer systems, has authored the book Interconnection networks for large-scale parallel processing, and does consulting and gives tutorials in these areas. He has been an IEEE Computer Society Distinguished Visitor and a NATO Advanced Study Institute Lecturer. He is a member of theEta Kappa Nu and Sigma Xi honorary societies.

Thomas Schwederski is currently working toward a PhD in computer engineering in the School of Electrical Engineering at Purdue University, where he is the member of the PASM Parallel Processing Laboratory in charge of the system prototype construction. Mr Schwederski received the Diplom-Ingenieur degree in electrical engineering from Ruhr-Universität Bochum, FRG, in 1983, and the MS degree in electrical engineering from Purdue University in 1985. His research interests include computer architecture, parallel processing, multimicroprocessing systems and VLSI design. He is a member of theEta Kappa Nu honorary society.

William Tsun-Yuk Hsu is working towards his PhD degree in the Computer Science Department at the University of Illinois at Champaign-Urbana, USA, where he is a research assistant in the Center for Supercomputing Research and Development. He received his BS and MS degrees from Purdue University, both from the School of Electrical Engineering. While at Purdue, he was a member of the PASM Parallel Processing Laboratory specializing in interconnection network design. His research interests are computer architecture, parallel processing, interconnection networks and graph theory. He is a member of theEta Kappa Nu and Phi Kappa Phi honorary societies.
David Meyer is an assistant professor and member of the PASM Parallel Processing Laboratory in the School of Electrical Engineering at Purdue University. He received the BS degree in electrical engineering in 1973, the MSE degree in electrical engineering in 1975, the MS degree in computer science in 1979, and the PhD degree in electrical engineering in 1981, all from Purdue University. His research interests include advanced microprocessors, computer architecture, acoustics, digital signal processing, parallel processing systems, VLSI design and educational delivery systems. Dr. Meyer has consulted in acoustics for both AT&T Bell Laboratories and Electro-Voice Inc. He is a member of the Eta Kappa Nu and the Tau Beta Pi honorary societies.

(1987)


(1987)