An Overview of the PASM Parallel Processing System

Howard Jay Siegel, Thomas Schwederski, James T. Kuehn*, and Nathaniel J. Davis IV**

PASM Parallel Processing Laboratory
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907 USA
May 1986

Abstract

PASM is a multifunction partitionable SIMD/MIMD system being designed at Purdue for parallel image understanding. It is to be a large-scale, dynamically reconfigurable multimicroprocessor system, which will incorporate over 1,000 complex processing elements. Parallel algorithm studies and simulations have been used to analyze application tasks in order to guide design decisions. A prototype of PASM is under construction that includes 30 Motorola MC68010 processors, a multistage interconnection network, five disk drives, and connections to the Purdue Engineering Computer Network for access to peripherals, terminals, software development tools, and so on. PASM is to serve as a vehicle for studying the use of parallelism for performing the numeric and symbolic processing needed for tasks such as computer vision. The PASM design concepts and prototype are overviewed, and brief examples of parallel algorithms are given.

1. Introduction

This is an overview of the design for the thousand-processor PASM system and the 30-processor prototype being built to validate the system design concepts. Parallelism is one way to increase computational speed. Different modes of parallelism can be employed in a computer system. The *SIMD (single instruction stream—multiple data stream)* mode [Fly66] typically uses a set of N processors, N memories, an interconnection network, and a control unit (e.g., Illiac IV [BoD72], STARAN [Bat76, Bat77], CLIP4 [Fou81], MPP [Bat82]). The control unit broadcasts instructions to the processors, and all active (enabled) processors execute the same instruction at the same time. Each processor executes instructions on data taken from a memory with which only it is associated. The interconnection network allows interprocessor communication. An *MSIMD (multiple-SIMD) system* is a parallel processing system that can be structured as one or more independent SIMD machines of various sizes (e.g., MAP [Nut77]). The Illiac IV was originally designed as an MSIMD system [BaB68]. The *MIMD (multiple instruction stream—multiple data stream)* mode [Fly66] typically consists of N processors and N memories, where each processor can follow an independent instruction stream (e.g., Ultracomputer [GoG83], BBN Butterfly [CrG85], Cosmic Cube [Sei85], RP3 [PfB85]). As with SIMD architectures, there is a multiple data stream and an interconnection network. A *partitionable SIMD/MIMD system* is a parallel processing system that can be structured as one or more independent SIMD and/or MIMD machines of various sizes (e.g., TRAC [SeU80]).

PASM is a partitionable SIMD/MIMD machine being designed at Purdue University to be a large-scale dynamically reconfigurable multimicroprocessor system [SiS81]. It is a special-purpose system aimed at exploiting the parallelism of image understanding tasks. PASM is being developed by using a variety of problems in image processing and pattern recognition to guide the design choices. It can also be applied to related areas such as speech understanding and biomedical signal processing.

PASM is to serve as a research tool for experimenting with parallel processing. The design attempts to incorporate the needed flexibility for studying large-scale SIMD and MIMD parallelism, while keeping system costs "reasonable." Portions of PASM have been simulated and a prototype is under construction.

In section 2, the PASM architecture is overviewed. Section 3 gives some examples of how PASM can be used for image processing. The PASM prototype is described in section 4. A summary of the parallel programming language and distributed operating system development for PASM is

This work was supported by the Air Force Office of Scientific Research, under grant number F49620-86-K-0006.

*current address:
Supercomputing Research Center
4380 Forbes Boulevard
Lanham, Maryland 20706

**Current address:
Air Force Institute of Technology
Wright-Patterson Air Force Base, Ohio 45433

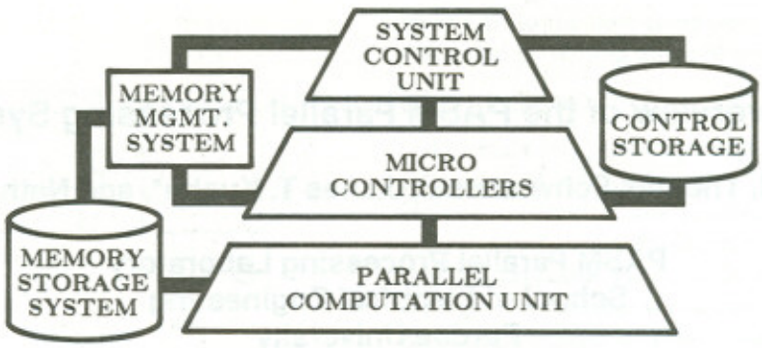

Figure 1. Block diagram overview of PASM.

given in section 5. The advantages of some of the features of PASM are discussed in section 6. Selected references for further reading about PASM appear at the end of this paper.

2. PASM Architecture

2.1. Parallel Computation Unit

A block diagram of the basic components of PASM is shown in Figure 1. The *Parallel Computation Unit* (Figure 2) contains $N = 2^n$ processors, N memory modules, and an interconnection network. The processors are microprocessors that perform the actual SIMD and MIMD computations. The memory modules are used by the processors for data storage in SIMD mode and both data and instruction storage in MIMD mode. A memory module is connected to each processor to form a processor-memory pair called a *Processing Element (PE)*. Each PE can operate in both the SIMD and MIMD modes of parallelism. The N PEs are numbered from 0 to N − 1, and each PE knows its number (address). A pair of memory units is used for each memory 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. The PASM N = 16 prototype under construction uses Motorola MC68010 processors; the final N = 1024 system, which the architecture is designed for, may employ custom VLSI processors specially designed for parallel image understanding. The *interconnection network* provides a means of communication among the PEs. Two types of multistage interconnection networks are being considered for PASM: the Generalized Cube and the Augmented Data Manipulator (ADM) [Sie85, SiM81b, SiM81a]. The ADM network is more flexible but is more complex. Features of the Generalized Cube network will be described to familiarize the readers with the properties of multistage networks.

The *Generalized Cube* network is representative of the multistage cube-type class of networks that include the baseline [WuF80], delta [Pat81], indirect binary n-cube [Pea77], omega [Law75], and SW-banyan (S = F = 2) [GoL73]. This class has been used or proposed for use in systems such as STARAN [Bat76, Bat77], Ultracomputer [GoG83], BBN Butterfly [CrG85], and RP3 [PfB85]. The Cube has N inputs and N outputs. It is shown in Figure 3a for N = 8. PE i, $0 \leq i < N$, would be connected to input port i and output port i of the unidirectional network. The Cube topology has $n = \log_2 N$ stages, where each stage consists of a set of N lines connected to N/2 interchange boxes. Each *interchange box* is a two-input, two-output device. The labels of the input/output *(I/O)* lines entering the upper and lower inputs of an interchange box are used as the labels for the upper and lower outputs, respectively. Each interchange box can be set individually to one of the four legitimate states shown in Figure 3a. Figures 3b, 3c, and 3d illustrate one-to-one, broadcast, and permutation connections, respectively. Note that many one-to-one and/or broadcasts can occur simultaneously.

The connections in this network are based on the cube interconnection functions [Sie77, Sie79]. Let $P = p_{n-1} \cdots p_1 p_0$ be the binary representation of an arbitrary I/O line label. Then the n cube interconnection functions can be defined as

$$\text{cube}_i(p_{n-1} \cdots p_1 p_0) = p_{n-1} \cdots p_{i+1} \bar{p}_i p_{i-1} \cdots p_1 p_0$$

where $0 \leq i < n$, $0 \leq P < N$, and $\bar{p}_i$ denotes the complement of p_i. This means that the cube_i interconnection function connects P to $\text{cube}_i(P)$, where $\text{cube}_i(P)$ is the I/O line whose

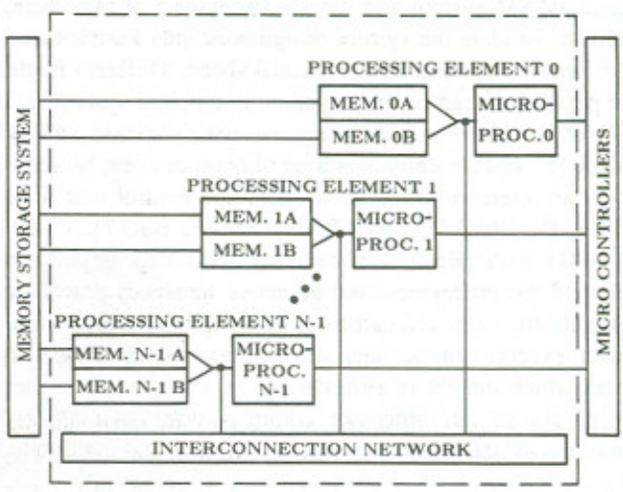

Figure 2. Parallel Computation Unit.

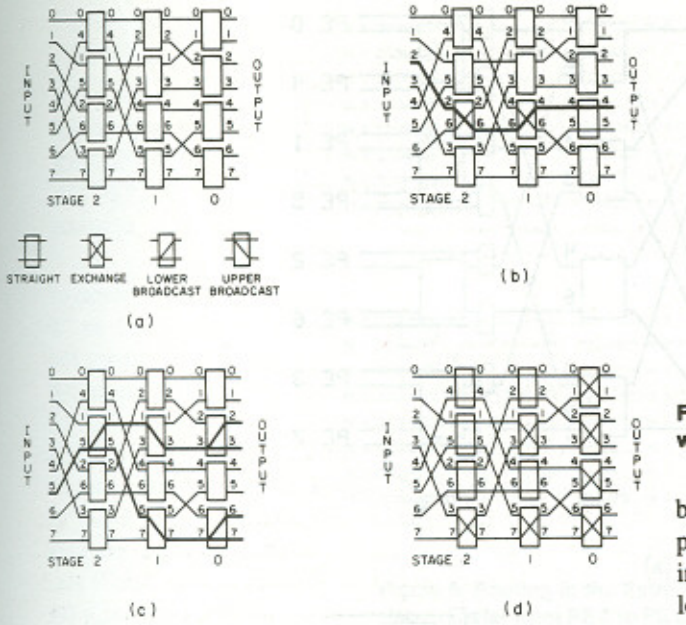

Figure 3. (a) Generalized Cube topology, shown for N = 8. (b) Example one-to-one connection (input 2 to output 4). (c) Example broadcast connection (input 5 to outputs 2, 3, 6, and 7). (d) Example permutation connection (input i to output i + 1 mod N). [Sie85]

label differs from P in just the i-th bit position. Stage i of the Cube topology contains the $cube_i$ interconnection function (i.e., it pairs I/O lines that differ only in the i-th bit position).

Routing tags are used as headers on messages; they (1) control each interchange box individually and (2) allow network control to be distributed among the PEs. The n-bit routing tag for one-to-one connections is computed from the input port number and desired output port number. Let S be the source address (input port number) and D be the destination address (output port number). Then the routing tag $T = S \oplus D$ (where "$\oplus$" means bitwise "exclusive-or"). Let $t_{n-1} \ldots t_1 t_0$ be the binary representation of T. An interchange box in the network at stage i need only examine t_i. If $t_i = 1$, an exchange is performed, and if $t_i = 0$, the straight connection is used. For example, if N = 8, S = 010, and D = 100, then T = 110. The corresponding stage settings are exchange, exchange, straight (see Figure 3b). Because the exclusive-or operation is commutative, the incoming routing tag is the same as the return tag. Since the destination PE has the routing tag to the source PE, it is easy to perform handshaking if desired. The address of the source PE can be computed by the destination PE by using $S = D \oplus T$. Routing tags that can be used for broadcasting data are an extension of this scheme [SiM81b].

The *partitionability* of a network is its ability to divide the system into independent subsystems of different sizes. The Cube network can be partitioned into independent subnetworks of various sizes, where each subnetwork of size $N' \leq N$ has all of the connection properties of a Cube network

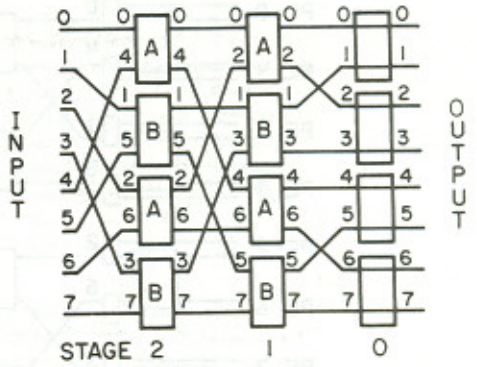

Figure 4. Cube network of size eight partitioned into two subnetworks of size four based on the low-order bit position. [Sie85]

built to be of size N′. In PASM, the partitioning is accomplished by requiring that the addresses of all of the I/O ports in a partition of size 2^i agree (have the same values) in their low-order n − i bit positions. For example, in Figure 4, subnetwork A consists of ports 0, 2, 4, and 6, and subnetwork B consists of ports 1, 3, 5, and 7. All ports in subnetwork A have a 0 in the low-order bit position; all ports in subnetwork B have a 1 in the low-order bit position. By setting all of the interchange boxes in stage 0 to straight, the two subnetworks are isolated. This is because stage 0 is the only stage that allows input ports that differ in their low-order bit to exchange data. Each subnetwork can be separately further subdivided, resulting in subnetworks of various sizes. The routing tag scheme can be used to enforce the partitioning by logically AND-ing the tags with masks to force to 0 tag positions which correspond to stages whose interchange boxes should be forced to the straight state. The network partitioning property allows the set of PASM PEs to be divided into independent partitions of various sizes.

The Cube network as presented here has the disadvantage that only a single path exists from any source to any destination. Thus, a single fault in the network makes many source-destination paths unavailable. The Extra Stage Cube network [AdS82], a single-fault tolerant variation of the Cube, overcomes this problem by introducing an additional "extra" stage. Figure 5a shows the expanded network. The extra (input) stage performs $cube_2$. This is followed by the "normal" Cube network stages, which perform $cube_0$, $cube_1$, and $cube_2$. The input (extra) and output stages can be bypassed as shown in Figures 5b through 5g. Under normal fault-free operation, the extra stage is disabled (i.e., the bypass circuitry is used). If a fault is present in the extra stage, the bypass circuitry is employed as in the no-fault case and network operation is not affected (see Figure 6a). If a fault occurs in an intermediate stage, both the extra stage and the output stage are enabled. This establishes two paths for every source-destination pair, and these paths are distinct except for the extra stage and output stage boxes. This is demonstrated in Figure 6b. Since there are two distinct paths, one must be fault free when a single fault occurs in an

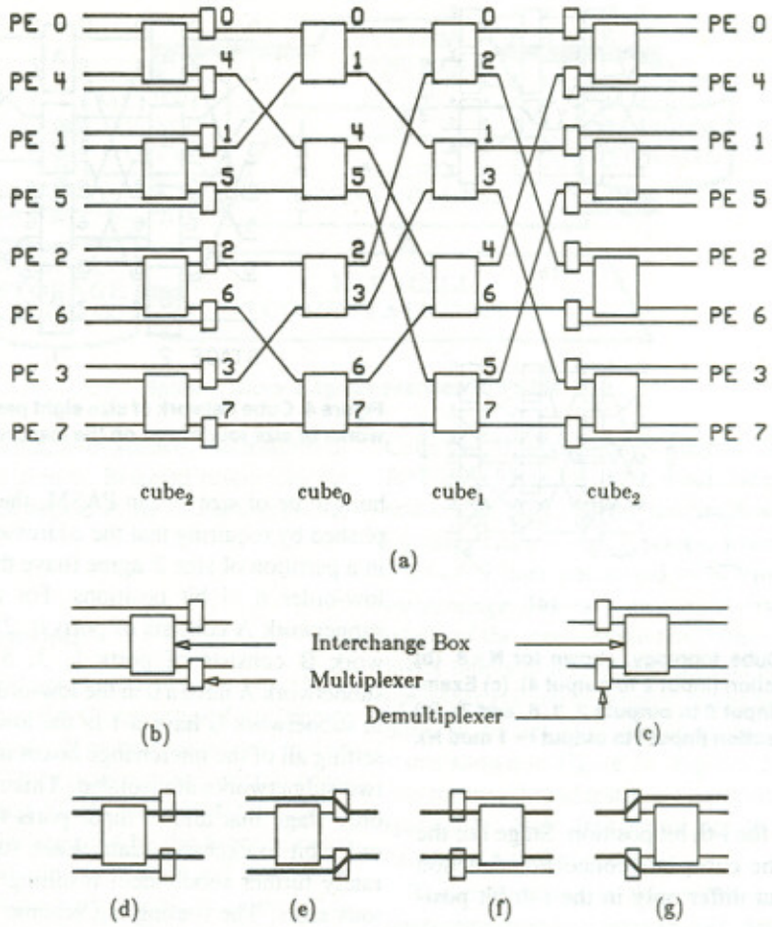

Figure 5. (a) The Extra Stage Cube Network topology, shown for N = 8. (b) Detail of input interchange box. (c) Detail of output interchange box. (d) Input stage enabled. (e) Input stage disabled. (f) Output stage enabled. (g) Output stage disabled.

intermediate stage. If a fault occurs in the output stage, its function (i.e., $cube_2$) is performed by the extra stage and the bypass circuitry in the output stage is used to bypass the fault there (see Figure 6c). The Extra Stage Cube network provides the same partitioning capabilities as does the regular Cube network, even in the presence of a fault. A partitioning example is shown in Figure 6d. In addition to being single-fault tolerant, it has been shown to be very robust under multiple faults [AdS84].

2.2. Micro Controllers

The *Micro Controllers (MCs)* (Figure 7) are a set of microprocessors that 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 $Q = 32$ (Q = 4 in the prototype). The MCs are the multiple control units needed to have a partitionable SIMD/MIMD system. Each MC memory module 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, executing the control flow instructions (e.g., branches) and broadcasting the data processing instructions to its PEs. In MIMD mode, each MC gets from its memory instructions and common data for coordinating its PEs.

The reconfiguration rule for PASM is that all PEs in a partition of size 2^p must agree in their low-order $n-p$ address bits. Thus, the physical addresses of the N/Q processors that are connected to an MC must all have the same low-order q bits so that they can form a partition. The value of these low-order q bits is the physical address of the MC. An SIMD machine partition of size RN/Q, where $R = 2^r$ and $0 \leq r \leq q$, is obtained by having R MCs use the same instructions and synchronizing the MCs. One way to provide the R MCs with the same instructions is by loading each of the memory modules associated with the R MCs with the same program (if this scheme is used, the reconfigurable bus in Figure 7 is not needed and MC microprocessor i is directly connected to MC memory module i). The physical addresses of these MCs must have the same low-

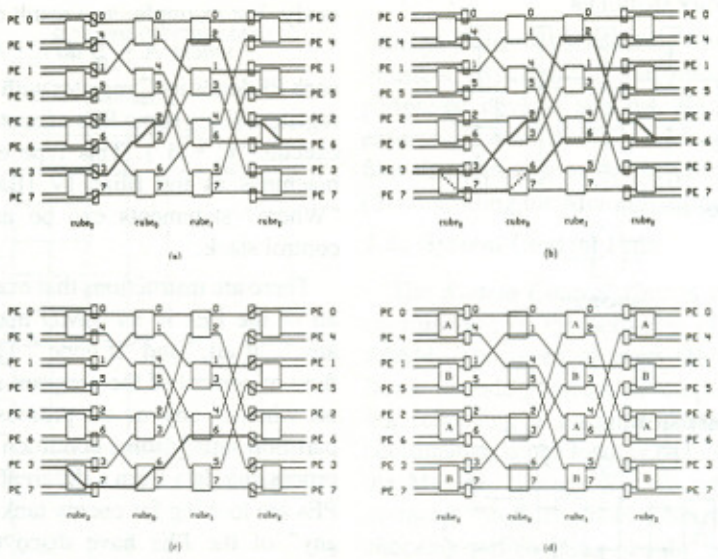

Figure 6. Routing in the Extra Stage Cube Network for N = 8. (a) Data transfer from PE 3 to PE 6, extra stage disabled (no fault or fault in the extra stage). (b) Data transfer from PE 3 to PE 6, extra stage and output stage enabled (fault in an intermediate stage). (c) Data transfer from PE 3 to PE 6, extra stage enabled and output stage disabled (fault in the output stage). (d) Partitioning the network into even ("A") and odd ("B") numbered PEs.

order q − r bits so that all of the PEs in the partition have the same low-order q − r physical address bits. Similarly, an MIMD machine partition of size RN/Q is obtained by combining the efforts of the PEs associated with R MCs that have the same low-order q − r physical address bits. In MIMD mode, the MCs are 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.

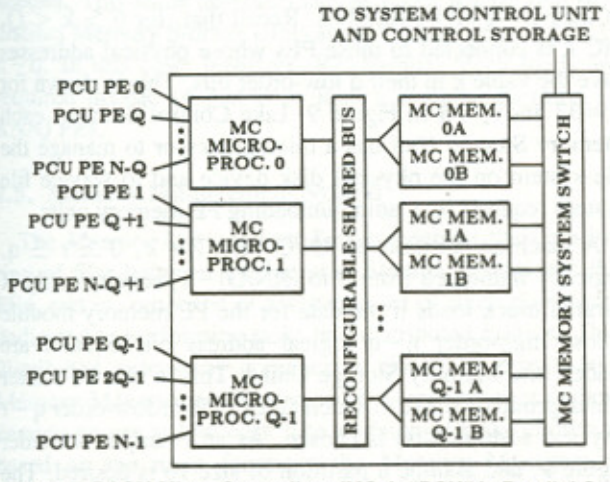

Figure 7. PASM Micro Controllers (MCs). "PCU" is Parallel Computation Unit.

As an alternative to loading the same program in R MC memories for an SIMD machine partition, the MC processors and MC memories may be connected by a shared reconfigurable ("shortable") bus [ArP76, KaK79], as shown in Figure 8. The MCs must be ordered on the bus in terms of the bit reverse of their addresses because of the partitioning rules. The advantages of the MC connection scheme are that it provides more program space for jobs using multiple MCs and provides a degree of fault tolerance, since known-faulty MC memory modules could be ignored. This scheme has the disadvantages of adding propagation delay through the bus switches and adding hardware complexity would could reduce reliability. These tradeoffs are currently being studied.

The PEs within each partition are assigned *logical addresses*. Given a machine partition of size RN/Q, the processors and memory modules for this partition have logical addresses (numbers) 0 to (RN/Q) − 1, $R = 2^r$, $0 \leq r \leq q$. The logical number of a PE is the high-order r + n − q bits of its physical number. Similarly, the MCs assigned to the partition are logically numbered (addressed) from 0 to R − 1. For R > 1, the logical number of an MC is the high-order r bits of its physical number. The PASM operating system chooses the partition to be used for executing a user's task. A system user only deals with logical addresses that are translated to physical addresses by the operating system.

A *masking scheme* is used in SIMD mode for determining which PEs will be active (i.e., execute instructions broadcast to them by their MC). PASM uses PE address masks and data conditional masks.

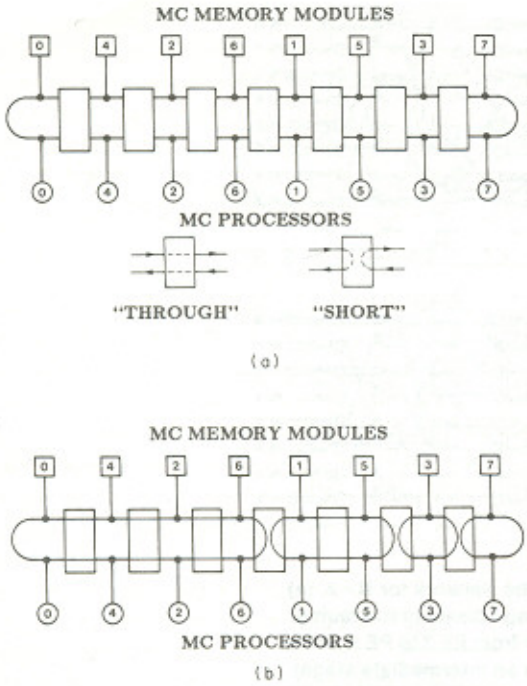

Figure 8. (a) Reconfigurable shared bus scheme for interconnecting MC processors and MC memory modules, shown for Q = 8, where each box can be set to "through" or "short." (b) Bus set for MCs 0, 2, 4, and 6 forming one machine partition, MCs 1 and 5 forming a second partition, MC 3 forming a third partition, and MC 7 forming a fourth partition.

The *PE address masking* scheme uses an n-position mask to specify which of the N PEs are to be activated. Each position of the mask contains either a 0, 1, or X ("don't care"), and the only PEs that are enabled are those whose address matches the mask: 0 matches 0, 1 matches 1, and either 0 or 1 matches X. Square brackets denote a mask. Superscripts are used as repetition factors. For example, "MASK $[X^{n-1}0]$" activates all even-numbered PEs, and "MASK $[0^{n-i}X^i]$" activates PEs 0 to $2^i - 1$. A *negative PE address mask* is similar to a regular PE address mask, except that it activates all those PEs that do not match the mask. Negative PE address masks are prefixed with a minus sign to distinguish them from regular PE address masks. For example, for N = 8, "MASK [−0X1]" activates all PEs except 1 and 3. PE address masks are specified in the SIMD program.

Data conditional masks are the implicit result of performing a conditional branch dependent on local data in an SIMD machine environment, where the result of different PEs' evaluations may differ. They are used when the decision to enable and disable PEs is made at execution time. As a result of a conditional *where statement* of the form

where <data-condition> *do* · · · *elsewhere* · · ·

each PE sets a flag to activate itself for either the "do" or the "elsewhere," but not both. The execution of the "elsewhere" statements must follow the "do" statements; that is, the "do" and "elsewhere" statements cannot be executed simultaneously. For example, as a result of executing the statement

where A < B *do* C ← A *elsewhere* C ← B

each PE loads its C register with the minimum of its A and B registers (i.e., some PEs execute "C ← A," and then the rest execute "C ← B"). This type of masking is used in such machines as the Illiac IV [BaB68] and PEPE [CrG72]. "Where" statements can be nested by using a run-time control stack.

There are instructions that examine the collective status of all of the PEs in an SIMD machine partition, such as "if any," "if all," and "if none." These instructions change the flow of control of the program at execution time depending on whether any or all processors in the SIMD machine partition satisfy some condition. For example, if each PE is processing data from a different section of an image, but all PEs are looking for enemy tanks, it is desirable to know "if any" of the PEs have discovered a tank. This requires communication among the MCs comprising the SIMD machine partition. There is a set of buses shared by MCs for this purpose.

2.3 Control Storage

Control Storage contains the programs for the MCs, and consists of a secondary storage device and a microprocessor for managing the file system on the device. The Control Storage processor acts as a file server by responding to requests to load programs into the MC memory units.

2.4. Memory Storage System

The *Memory Storage System* provides secondary storage space for the Parallel Computation Unit for the data files in SIMD mode and for the data and program files in 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. For $0 \leq i < N/Q$, Memory Storage Unit i is connected to those PE memory modules whose physical addresses have the value i in their n − q high-order bits. Recall that, for $0 \leq k < Q$, 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 9. Like Control Storage, each Memory Storage Unit has a microprocessor to manage the file system on the physical disk device and to service file system requests by loading/unloading PE memory units.

A machine partition of RN/Q PEs, $R = 2^r$, $0 \leq r \leq q$, logically numbered from 0 to (RN/Q) − 1, requires only R parallel block loads if the data for the PE memory module whose high-order n − q logical address bits equal i are 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. As an example, consider Figure 9, and assume a partition of size 16 is desired. The data for the PE memory modules whose logical addresses

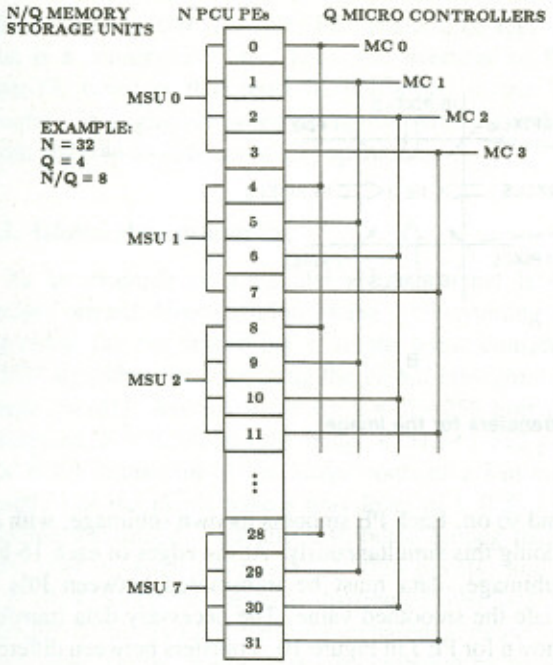

Figure 9. Organization of the Memory Storage System, shown for N = 32 and Q = 4. "MSU" is Memory Storage Unit. "PCU" is Parallel Computation Unit.

are 0 and 1 are loaded into Memory Storage Unit 0, the data 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. Given this assignment of MCs, the PE memory module whose physical address is $2i+1$ has logical address i, $0 \leq i < 16$. 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. This same approach can be taken if only $(N/Q)/2^d$ distinct Memory Storage Units are available, where $0 \leq d \leq n-q$. In this case, however, $R2^d$ parallel block loads are required instead of just R to load a machine partition of RN/Q PEs.

2.5. Memory Management System

The *Memory Management System* controls the transferring of files between the Memory Storage System and the PEs, and is composed of a separate set of microprocessors dedicated to performing tasks in a distributed fashion. This distributed processing approach is chosen to provide the Memory Management System with a large amount of processing power at low cost. The division of tasks chosen is based on the main functions the Memory Management System must perform, including (1) fielding file system requests from the System Control Unit, MCs, or PEs (via their MCs) and passing them on to the appropriate Memory Storage Units; (2) scheduling Memory Storage System data transfers; (3) controlling input/output operations involving peripheral devices and the Memory Storage System; (4) enforcing a consistent file naming and placement policy for files distributed over the Memory Storage Unit disks; and (5) controlling the Memory Storage System bus.

2.6. System Control Unit

The *System Control Unit* is responsible for the overall coordination of the activities of the other components of PASM. The types of tasks the System Control Unit performs include program development support, job scheduling (choosing a machine partition for a user job), and coordination of the loading of the PE memory modules from the Memory Storage System with the loading of the MC memory modules from Control Storage. By carefully choosing which tasks should be assigned to the System Control Unit and which should be assigned to other system components (e.g., the MCs and Memory Management System), the System Control Unit can work effectively and not become a bottleneck. For the N = 1024 PASM, the System Control Unit may consist of several processors in order to perform all of its functions efficiently. In the N = 16 prototype, the System Control Unit is a microprocessor, and the program development functions are performed by the host computer network.

3. Parallel Image Processing Algorithms

3.1. Introduction

A number of SIMD and MIMD algorithms to perform common image processing tasks have been developed by our group. Image processing algorithms that have been structured for parallel execution include image smoothing, histogramming, two-dimensional FFT calculation, local area histogram equalization, local area brightness and gain control, feature extraction, maximum likelihood classification, contextual statistical classification, image correlation (convolution, filtering), block truncation coding, resampling, rectification, rotation, translation, scaling, elevation/location determination, median filtering, Sobel edge sharpening, clustering feature enhancement, scene segmentation, Karhunen-Loeve transformation, three-dimensional shape analysis using Fourier descriptors, raster-to-vector and vector-to-raster conversion (image thinning, vectorization), and a computer vision task including edge labeling, perimeter, area, center of mass, and hole count computations. These have been analyzed in terms of machine-size/problem-size relationships, processor capability needed for efficient execution, memory requirements, and inter-PE communication requirements. Different algorithm strategies have been explored and compared. From these studies we are assessing the ways in which parallelism can be used for vision.

In this section, three brief examples of parallel image processing algorithms are given. The first is a simple SIMD image smoothing algorithm, the second a more complex

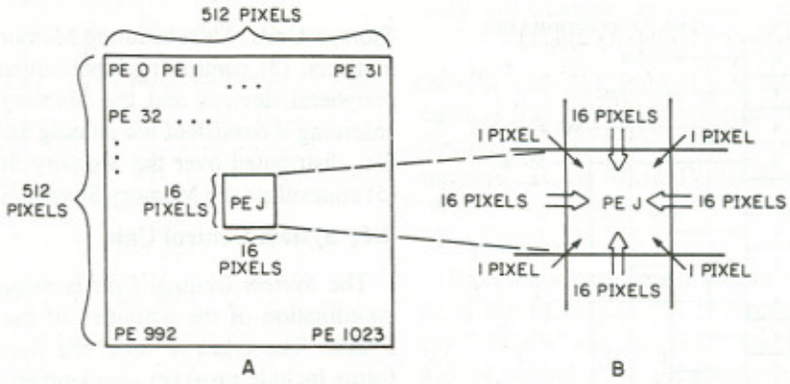

Figure 10. Data allocation and inter-PE transfers for the image smoothing algorithm.

SIMD histogramming algorithm, and the third an SIMD/MIMD contour extraction algorithm.

3.2. Image Smoothing

As an example of a simple parallel algorithm, consider the *smoothing of an image* [SiS81]. The algorithm described here smooths a gray level input image. The algorithm has "I" as an input image and "S" as an output image. Assume both I and S contain 512-by-512 *pixels* (picture elements), for a total of 512^2 pixels each. Each point of I is an eight-bit unsigned integer representing one of 256 possible gray levels. The gray level of each pixel indicates how "dark" that pixel is, where 0 means white and 255 means black. Each point in the smoothed image, S(i,j), is the average of the gray levels of I(i,j) and its eight nearest neighbors, I(i−1,j−1), 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 not calculated since their corresponding pixels in I do not have eight adjacent neighbors.

Consider how this could be implemented on an SIMD machine with N = 1024 PEs, logically arranged as an array of 32-by-32 PEs as shown in Figure 10. Each PE stores a 16-by-16 subimage block of the 512-by-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 smooths its own subimage, with all PEs doing this simultaneously. At the edges of each 16-by-16 subimage, data must be transmitted between PEs to calculate the smoothed value. The necessary data transfers are shown for PE J in Figure 10. Transfers between different PEs can occur simultaneously (e.g., when PE J − 1 sends its top right corner pixel to PE J, PE J can send its top right corner pixel to PE J+1, PE J+1 can send its top right corner pixel to PE J+2, etc.).

To perform a smoothing operation on a 512-by-512 image by the parallel smoothing of 1024 subimage blocks of size 16-by-16, $16^2 = 256$ parallel smoothing operations are performed. As described above, the neighbors of the subimage edge pixels must be transferred in from adjacent PEs. By using either the Cube or the ADM networks, the total number of parallel data element transfers needed is (4 * 16) + 4 = 68: 16 for each of the top, bottom, left side, and right side edges, and four for the corners (see Figure 10). 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 262,144/256 = 1024 = N. If the inter-PE data transfer time is included and it is assumed that each parallel data transfer requires at most as much time as one smoothing operation, then the time factor improvement is

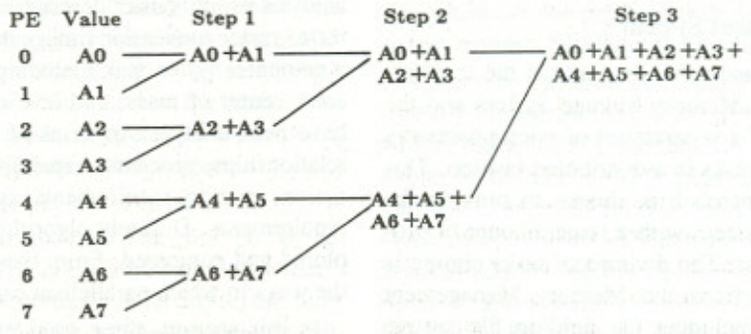

Figure 11. Recursive doubling with eight PEs.

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 when compared with the reduction from 262,144 to 256 smoothing operations.

3.3. Global Histogramming

As an example of a parallel algorithm that is not a straightforward decomposition of the corresponding serial algorithm (as the smoothing example was), consider an SIMD algorithm for computing the *global histogram of an image* [SiS81]. Assume there are $B = 2^b = 256$ bins in the histogram, $N = 1024$, and the image is 512-by-512 pixels. The B-bin histogram of the image contains a j in bin i if exactly j of the pixels have a gray level of i, $0 \leq i < B$. Assume the image is equally distributed among the 1024 PEs; that is, each PE has $512^2/1024$ pixels, and $B \leq 512^2/1024$. Since the image is distributed over 1024 PEs, each PE calculates a B-bin histogram based on its subimage. Then these "local" histograms are combined by using the techniques described below.

The method of *recursive doubling* [Sto80] is used in the combining procedure. Consider the problem of summing up 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 a parallel computer the vector is distributed among processors (i.e., each of N processors holds one element). The recursive doubling procedure is illustrated in Figure 11. Assume that N is a power of 2; in the Figure, $N = 8$. In the first step, all odd numbered PEs (i.e., 1, 3, 5, 7) send their 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. The overall procedure required three transfers and three additions. In general, for N a power of 2, $\log_2 N$ additions and data transfer steps are required.

After the local histograms are calculated, n steps are used to combine them. In the first b steps, each block of B PEs performs B simultaneous recursive doublings to compute the histogram for the portion of the image contained in the block (see Figure 12). In the first step, each block of PEs is divided in half such that the PEs with the lower addresses form one group and the PEs with the higher addresses form another. Each group accumulates the sums for half of the bins and sends the bins it is not accumulating to the other

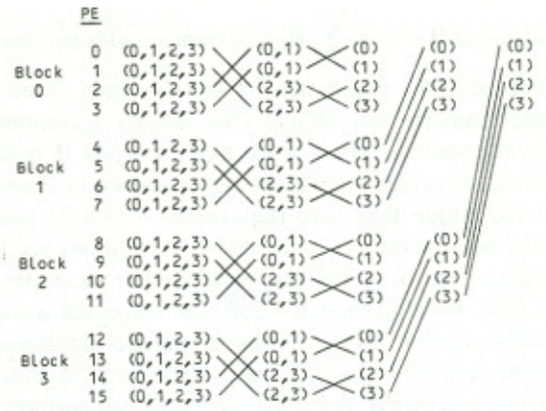

Figure 12. Histogram calculation for N = 16 PEs, B = 4 bins. (w, . . . , z) denotes that bins w through z of the partial histogram are in the PE.

group. The "lower-numbered group" accumulates the sums for the first half of the bins, while the "higher-numbered group" accumulates the second half of the bins. For each successive merging step, the groups of PEs are re-subdivided with the accumulated subtotals for each bin being combined into half as many PEs at each step. The subdividing process continues until there is one PE in each group and each PE has the total value for one bin from the portion of the image contained in the B PEs in its block. The next $n - b$ steps combine the results of these blocks to yield the histogram of the entire image distributed over B PEs. The sum for bin i ends up in PE i, for $0 \leq i < B$. This is done by performing $n - b$ steps of a recursive doubling algorithm to sum the partial histograms from the N/B blocks, shown by the last two steps of Figure 12. The recursive doubling steps are done for all B bins simultaneously.

A sequential algorithm to compute the histogram of an M-by-M image requires M^2 additions. The SIMD algorithm uses M^2/N additions for each PE to compute its local histogram. At step i in the merging of the partial histograms, $0 \leq i < b$, the number of parallel data transfer/adds required is $B/2^{i+1}$. This is because at step i each PE saves half of the bins it accumulated at the last step and sends the other half to another PE. For example, in Figure 12, when $B = 4$, at step $i = 0$, PEs 0, 1, 4, 5, 8, 9, 12, and 13 ("group L") send their bin 2 data, while PEs 2, 3, 6, 7, 10, 11, 14, and 15 ("group H") simultaneously send their bin 0 data. The bin data received by each PE is then added to the corresponding bin data the PE is holding. Then group L sends its bin 3 data while group H simultaneously sends its bin 1 data. Again in each PE, the received bin data is added to the corresponding bin data being held. Thus, $2 = B/2^{i+1}$ transfer/adds are used. At step $i = 1$, (1) PEs 0, 4, 8, and 12 send their bin 1 data, (2) PEs 1, 5, 9, and 13 send their bin 0 data, (3) PEs 2, 6, 10, and 14 send their bin 3 data, and (4) PEs 3, 7, 11, and 15 send their bin 2 data, all simultaneously. The matching bin data values are then combined in all PEs. Thus, $1 = B/2^{i+1}$ transfer/add is used.

A total of $B-1$ ($=\sum_{i=0}^{b-1} B/2^{i+1}$) transfer/adds are therefore performed in the first b steps of the algorithm. Then $n-b$ parallel transfers and additions are needed to combine the block histograms. These $n-b$ steps are just B recursive doublings executed simultaneously as shown in Figure 12. This technique therefore requires $B-1+n-b$ parallel transfer/add operations, plus the M^2/N additions needed to compute the local PE histograms. For example, for $N=1024$, $M=512$, and $B=256$, the sequential algorithm would require 262,144 additions; the parallel algorithm uses 256 addition steps plus 257 transfer/add steps. Again, both the Cube and ADM multistage networks can perform each of the required inter-PE data transfers in one pass through the network.

3.4. Contour Extraction

Now consider an SIMD/MIMD parallel implementation of *contour extraction* [TuA83]. Two algorithms from a contour extraction task are *edge-guided thresholding (EGT)* and *contour tracing*. The EGT algorithm is used to determine a set of optimal thresholds for quantizing the image [MiR81]. The 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.

The EGT algorithm consists of three major steps. First, the Sobel edge operator [DuH73] is used to generate an edge image in which gray levels indicate the magnitude of the gradient. A figure of merit that indicates how well a given threshold gray level matches edges in the edge image is then computed for every possible threshold. Finally, the maximum value of the figure of merit function is chosen to determine the threshold level. This is done for each PE's subimage independently; thus, the threshold levels may differ from one subimage 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 MIMD contour tracing algorithm has two phases. In phase I, PEs segment their subimages based on the threshold value each calculated by using the EGT algorithm. All local contours (both closed and partial) are traced and recorded. In phase II, partial contours traced during phase I are connected.

In phase I there is no PE-to-PE communication. Each PE creates a segmented subimage for a particular threshold T by assigning a value of one to subimage pixels having a grey level greater than or equal to T and a value of zero to the others. Contour tracing begins with each PE scanning the rows of its segmented subimage 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 value one which has a zero-valued neighbor to either or both sides. Contours are traced in either a clockwise or a counter-clockwise direction and the Freeman direction codes [Fre61] of the "chain" of pixels are recorded. When a pixel from an adjacent subimage 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 the opposite direction until another point of indecision is reached. Closed contours contained within a subimage are traced completely during phase I.

In phase II, each PE attempts to connect its partial contours to those located in neighboring PEs. PEs consider each partial contour's end point in turn and try to find a possible extending contour in a neighboring PE. Once such an extending 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. A protocol is necessary to prevent more than one PE from trying to use the same partial contour as an extending contour at the same time. Phase II is complete when all of the contours have been connected.

The contour extraction task demonstrates the advantages of several features of PASM. The local neighbor inter-PE transfers needed for the SIMD EGT algorithm can be performed by all PEs simultaneously, as was discussed for the SIMD smoothing algorithm. The global inter-PE communications for the MIMD contour tracing can be performed efficiently by either the multistage Cube or ADM networks. Last, the ability of the PEs to operate in either SIMD or MIMD mode allows the most appropriate type of parallelism to be employed by each algorithm in the task.

4. PASM Prototype Design

4.1. Introduction

A prototype of the PASM system, with $N=16$ and $Q=4$, is currently being constructed (see Figure 13). Only off-the-shelf components are used; this eliminates the need for VLSI chips and reduces development time and construction costs. By using a modular design concept, 12 different types of physical boards are being used to construct the prototype, two of which are commercial products. These are:

1. *CPU (commercial)*—Motorola MC68010 16-bit microprocessor, VMEbus interface, and local memory.
2. *Memory*—VMEbus-based dual-ported arbitrated access 1-Mbyte dynamic memory with byte parity using 256K-by-1 technology.
3. *MC-PE I/O*—channels for MC to PE instruction broadcast, PE enable signals, and MC-PE communication.
4. *PE-Network I/O*—interfaces for establishing paths and using the multistage interconnection network.
5. *Fetch Unit*—specialized MC unit for fetching and broadcasting SIMD instructions.
6. *Disk Controller (commercial)*—VMEbus-based, capa-

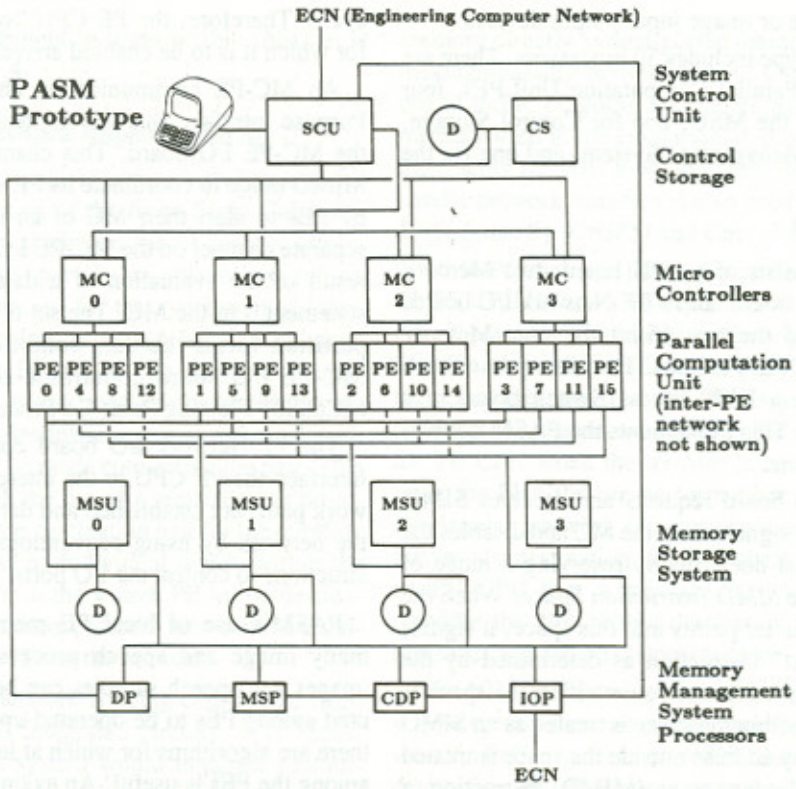

Figure 13. The PASM parallel processing system prototype.

ble of controlling up to four physical disk units, two of them Winchester-technology.

7. *Disk Access Switch*—interfaces Control Storage or a Memory Storage Unit to MC or PE memory units.
8. *Network Extra Stage Box*—two-by-two interchange box implementing the extra stage and extra stage bypasses of the Extra Stage Cube network.
9. *Network Intermediate Stage Box*—two-by-two interchange box implementing the intermediate stages of the Extra Stage Cube network.
10. *Network Output Stage Box*—two-by-two interchange box implementing the output stage and output stage bypasses of the Extra Stage Cube network.
11. *Parallel Port*—parallel I/O channels for the connections between the System Control Unit and MC, MC and Memory Management System, etc.
12. *Partition Controller*—inter-MC communication and synchronization for multiple-MC partitions.

The System Control Unit, MCs, PEs, Memory Management System processors, Control Storage and Memory Storage Unit processors, and the PE interconnection network can all be implemented by using these boards in different configurations.

The Parallel Computation Unit consists of 16 PEs connected by an Extra Stage Cube interconnection network. Each group of four PEs is controlled by an MC. The prototype could be readily extended to 8 MCs and 8 PEs per MC.

The System Control Unit (SCU) for the prototype is a dedicated microprocessor responsible for the overall orchestration of the activities of the system. The Control Storage microprocessor (CS) responds to file system requests from both the System Control Unit and the MCs. Control Storage uses a Winchester-technology disk drive (D). To allow a larger group of users to access PASM, the System Control Unit serves as a link between PASM and the Engineering Computer Network (ECN) at Purdue University. ECN is a local network connecting several dozen micro, mini, and super-mini computers. A PASM user's terminal is physically connected to an ECN host computer. The host provides the environment for the development, compilation, and debugging of SIMD and MIMD programs to prevent the System Control Unit microprocessor from being burdened. Commands (jobs) initiated by users are sent by the ECN host to the System Control Unit which schedules the jobs to be run on the parallel machine. The prototype system console is used for system startup and monitoring of PASM activities.

Mass storage for the PEs is provided by four high capacity Winchester-technology disk drives (D), each controlled by a Memory Storage Unit (MSU) microprocessor. The Memory Storage Units are managed by the Memory Management System, consisting of a Directory Lookup Processor (DP), a Memory Scheduling Processor (MSP), a Command Distribution Processor (CDP), and an Input/Output and Reformatting Processor (IOP). User programs and data can be received from or sent to ECN peripherals such as

additional mass storage or image input/output devices.

The complete prototype includes 30 processors. There are 16 processors for the Parallel Computation Unit PEs, four for the MCs, four for the MSU, one for Control Storage, four for the Memory Management System, and one for the SCU.

4.2. PE Organization

A prototype PE consists of a CPU board, two Memory boards, an MC-PE I/O board, and a PE-Network I/O board. One "port" of each of the two 16-bit dynamic Memory boards resides on the CPU VMEbus. The other port of each board is connected to a Disk Access Switch board in a Memory Storage Unit. This implements the PASM double-buffered memory scheme.

A PE's MC-PE I/O board requests and receives SIMD instructions and enable signals from the MC and disables the PE when necessary and does this by reserving a range of addresses known as the *SIMD Instruction Space*. When the PE CPU's program counter points into this space, it signals a request for the "next" instruction as determined by the MC. The actual program counter value within the space is irrelevant; any access within the space is treated as an SIMD instruction request. Any address outside the space is treated normally (i.e., as a reference to an (MIMD) instruction, a datum, or an I/O device in the PE's memory space). Thus to cause the PEs to switch to MIMD mode, the MC broadcasts a "jump to subroutine at A" instruction, where A is an address not in the SIMD Instruction Space. The former program counter value (in the SIMD Instruction Space) is stacked and execution begins in MIMD mode. To revert to SIMD mode, PEs execute a "return" instruction, which re-loads the program counter residing in the SIMD Instruction Space. MIMD programs can use this simple mechanism to synchronize the PEs quickly.

When all active PEs in a partition request an instruction in SIMD mode, an instruction word is broadcast from the MC(s) and placed on all the PE data busses simultaneously. Each PE decodes the instruction and performs the operation or requests additional operand words. Note that conventional 68000-family CPU instructions are broadcast rather than control signals as in many other SIMD machines. In MIMD mode, instructions and data are contained wholly within a PE's memory, and no instructions are broadcast by the MC. Since MIMD program instructions for each PE are stored in the PE's local memory and SIMD program instructions are broadcast to the PE by its MC, the interconnection network is used solely for inter-PE data communication rather than for both inter-PE communication and instruction fetch operations.

In the prototype, a PE is disabled by intercepting instructions broadcast to it on the MC-PE I/O board. With an asynchronous bus such as the VMEbus, a PE CPU can be prevented from "seeing" an instruction by withholding the acknowledgment signal used to indicate the end of a bus cycle. Therefore, the PE CPU "waits" until an instruction for which it is to be enabled arrives.

An MC-PE communication channel based on General Purpose Interface Bus (GPIB) technology is also found on the MC-PE I/O board. This channel is used by an MC in MIMD mode to coordinate its PE's activities. It is also used by PEs to alert their MC of an internal fault or error. A separate channel on the MC-PE I/O board is used to send the result of the evaluation of a data-condition (in a "where statement") to the MC. The set of results from all PEs in a partition forms the data conditional mask. Finally, the MC-PE I/O board contains a socket for the Motorola MC68881 Floating Point Co-Processor.

The PE-Network I/O board contains parallel ports that interface the PE CPU to the interconnection network. Network paths are established and data are sent to or read from the network by using conventional 68000-family CPU instructions to control the I/O ports.

PASM's use of local PE memories is appropriate for many image and speech processing algorithms because images and speech samples can be subdivided and distributed among PEs to be operated upon in parallel. However, there are algorithms for which at least some memory shared among the PEs is useful. An example is the contour tracing algorithm where partial contour tables need to be accessed by several PEs. Shared memory plays a significant role in MIMD programs: shared variables are often used for process synchronization, interprocess communication, and so on.

One way to emulate shared memory in the prototype is to designate part of each PE's address space as shared. Assuming a machine-wide shared address space of S bytes, S/N bytes of physical shared memory would reside in each PE and each PE would hold memory responding to distinct addresses. An attempt to access a shared memory location for which there was physical memory in the PE would be handled normally. However, an attempt to access a shared memory location which is held by another PE results in a "bus error" trap (due to the nonexistent memory) and initiation of exception processing. During the exception processing, the nonlocal shared address that was generated is examined and the remote PE in which it resides can be determined. A message is generated and sent through the interconnection network to the remote PE requesting the value of the data item at the shared address. The remote PE responds to the request by fetching the data from its local memory and returning it through the interconnection network. As part of the exception processing, the value returned is patched into the run-time stack and the faulted bus cycle is re-run. These actions are equivalent to those that occur in a virtual memory system when an address is found to be nonresident. The performance penalty is rather severe since exception processing is done at both the local and the remote PEs and since the interconnection network is tra-

versed twice. An enhancement to improve this situation is discussed in subsection 4.4.

4.3. Interconnection Network Implementation

The PASM prototype's interconnection network is a circuit-switched implementation of the Extra Stage Cube network. Circuit switching was chosen for its ease of implementation as well as its particular suitability (when compared to packet switching) for the synchronized inter-PE data transfers of complete subimage rows or columns such as those shown in Figure 10. By using a circuit-switched network, prior to any message transmission between a network source-destination pair, a physical path through the network must be made connecting the pair. This path is established through the use of a request-grant protocol, and connects the source-destination pair for the duration of the message transmission. Individual words within the message are transferred from the source PE to the destination PE by using a handshaking protocol between the parallel ports that interface the PEs to the network.

The PASM prototype network is being constructed from readily available SSI/MSI integrated circuits. It is anticipated that a full 1024-PE system would integrate custom-designed LSI components into the network design. In its current configuration of 16 PEs, the network consists of five stages of interchange boxes with eight boxes per stage for a total of 40 Network Interchange Box boards. A path through the network can be established in approximately 1000 ns (assuming no delays due to network conflicts). The data itself can be transmitted from a network input port to a network output port at a rate of one 16-bit word every 400 ns. With the PEs themselves operating on a 10 Mhz system clock, these transfer times are fast enough so that the network does not act as a bottleneck to the computation process under execution.

The initial prototype implementation uses the PE CPUs to perform all actions required for interconnection network transfers. Thus a CPU has to set up a path through the network, perform error checking, start and monitor watchdog timers, etc. This causes high overhead whenever data have to be transferred through the network. The impact of the overhead is increased when emulating shared memory with the approach discussed earlier.

4.4. The Network Interface Unit

One way to enhance the performance of the interconnection network significantly is to use a special-purpose Network Interface Unit (NIU) as part of each PE. An NIU is currently being designed and will replace the current prototype PE-Network I/O board. An NIU for PASM has three primary functions: it handles PE-to-PE message passing, it emulates shared memory accesses, and it acts as a sophisticated DMA controller. In order to perform these functions efficiently, the NIU contains a fast bipolar microprocessor, a CPU-NIU interface that allows the NIU to access the CPU's memory directly, a dual-ported memory (mailbox RAM) for message-passing between the PE CPU and the NIU, memory to buffer outgoing and incoming messages, a block of memory used in the emulation of shared memory, and logic interfacing the NIU to the network itself. A co-processor to handle network transfers is also used in systems such as the BBN Butterfly [CrG85] and Cm* [SwF77].

PE-to-PE message passing is performed under control of the PE CPU. When a PE needs to send a message to another PE, it places a message transfer request into the mailbox RAM, and the NIU reads this request. The NIU then fetches the actual message from the CPU's main memory, sets up a path through the network, sends the message, and informs the PE CPU when the transfer is completed. In the meantime, the PE CPU can perform other processing tasks, thus overlapping network I/O and processing. When a message arrives at a destination, the NIU there accepts it and informs its PE CPU that a message is available. The PE CPU provides the NIU with a destination buffer address for the message, and the NIU stores it there.

The NIU facilitates complex message schemes. A simple way to specify a message is to give the NIU the location and size of a buffer in memory and to instruct it to transfer it to a destination PE. However, since the NIU processor is general-purpose, it can be programmed to move more complicated data structures. For example, a message can be specified by a starting address, size, stride, and count. Here, the NIU transfers "count" data items of the given "size" beginning at the starting address and incrementing the address by "stride" to obtain each new item.

If desired, the NIU can buffer a message and use high speed DMA transfers from a source buffer via the network into a destination buffer. This way the time required to perform the address calculations for fetching data from the CPU memory does not slow the actual network transfer, thus effectively reducing network traffic by decreasing the average time a network path stays established.

Through the use of an NIU, shared memory can be implemented with less performance penalty than the software emulation scheme described earlier. For the prototype, an NIU is being designed and constructed that contains 2K bytes of shared memory and is associated with each PE. This results in a total shared memory size of 16*2K bytes = 32 K bytes. Whenever a PE CPU accesses any location in the shared memory space, the access is decoded by the local NIU. If the access was directed to the shared memory slice located in the local NIU, the NIU performs the access on the memory without having to use the interconnection network. Otherwise, the NIU determines the PE in which the slice is located, sets up a path to the proper remote PE, and sends a message to the remote PE requesting shared memory access. This message is handled not by the remote PE CPU but exclusively by the remote NIU. If the message specifies a write to shared memory, the remote NIU writes the data into its shared space. If it was a read access, the remote NIU sets

up a path to the source PE, and passes the value obtained from the shared space to it. By using this scheme, processing in the remote PE CPU is not interrupted and the fast NIU processor results in high-speed accesses.

4.5. MC Organization

An MC consists of a CPU board, two Memory boards, an MC-PE I/O board, a Fetch Unit board, and a Parallel Port board. The MC CPU coordinates its PEs in MIMD mode; in SIMD mode, it performs the program flow operations (such as loop counting and branching) and the masking operations. One port of each of the Memory boards is on the MC CPU's VME bus while the other is attached to a Disk Access Switch board in Control Storage. The MC-PE I/O board is the same one used in the PEs, but populated with somewhat different chips. Its main functions in the MC are to control the GPIB channel, to accept N/Q bits of a data conditional mask from its PEs, and to interface the set of inter-MC result and synchronization buses used for evaluation of the "if any"-type conditions. The Parallel Port board is used to communicate with the System Control Unit and with the Memory Management System.

A 68000-family MC CPU alone cannot fetch SIMD instructions, decode them, execute the control instructions, and broadcast the arithmetic instructions to the PEs at a rate fast enough to avoid "starving" the PE CPUs of instructions. One of the specialized components of the MC that performs some of these functions is the Fetch Unit, which is controlled by the MC CPU. The Fetch Unit consists of a high-speed dual-ported Fetch Unit Memory, a finite state machine controller, and a FIFO buffer. SIMD PE instructions are placed in Fetch Unit Memory, while MC control instructions are placed in the dynamic MC CPU memories. When a block of one or more PE instructions is to be broadcast to the PEs, the MC CPU writes a special command word to the Fetch Unit controller indicating the base address and the length of the block. The Fetch Unit Controller fetches the words of the block one-by-one from Fetch Unit Memory and places them in the FIFO buffer in preparation for broadcast. As each instruction is placed in the FIFO, the enable signals controlling which PEs are to be enabled for that instruction are also enqueued in the FIFO. The MC CPU can also create instructions at execution time and place them in the FIFO. This would be done if the MC needed to broadcast some run-time-dependent value to each of the PEs at a certain point in a program.

The Fetch Unit Controller fills the FIFO buffer while the PEs drain it. When all of the PEs associated with an MC request an instruction, those requests are combined by the Partition Controller board. Only when all PEs in a partition request an instruction, one is dequeued from the FIFO and broadcast to them. The FIFO buffer allows overlap between the operations of an MC and its PEs in SIMD mode, thereby improving performance.

4.6. Secondary Memory Systems

Control Storage and the Memory Storage Units each consist of a CPU board, two Memory boards, a Disk Controller board (and physical disks), one or more Parallel Port boards providing channels for incoming file system requests, and a Disk Access Switch board for each MC or PE memory it serves. Memory Management System processors each consist of a CPU board and a number of Parallel Port boards. The CPU boards lie on a common VMEbus with a shared dynamic memory allowing easy exchange of shared data structures. They communicate control information over the parallel port channels which do not require VMEbus access.

5. PASM Languages and Operating System

5.1. Languages

MIMD programs for PASM can be written by coding each instruction stream in a conventional serial programming language and using calls to operating system functions to allow the streams to communicate. Even if higher-level languages with embedded multi-tasking primitives are used (e.g, Ada™, CSP), the compiler produces a number of sequential instruction streams that can be run in parallel and that communicate and synchronize via operating system calls. Either approach can be used for PASM, although only the first approach is supported by the C compiler and 68000-family assembler developed for use by the PASM prototype. The MIMD-specific operating system functions called in MIMD programs are being developed as part of the PASM operating system.

SIMD mode programming requires a different type of language since the single instruction stream controls a machine consisting of a scalar unit (the control unit) and a variable-length vector unit (the PEs). A prototype-specific SIMD assembly language has been implemented. In this language, instructions to be executed by the 68000-family MC CPU have their assembly language mnemonics prefixed by a "c_" (for control unit), while instructions to be broadcast to the PEs are prefixed with a "p_." The SIMD program is coded in a single stream with the control and PE instructions intermixed. The assembler separates the control and PE instructions so that they can be placed in separate memories at execution time. Where blocks of PE instructions are removed to separate them from the control stream, Fetch Unit control words are inserted to effect the broadcast of the PE instructions at execution time. Programs written in this assembly language have been run on a PASM simulator. In addition, SIMD extensions to the C and Ada programming languages have been proposed by members of the PASM development group. A parallel implementation of a LISP interpreter for use in image understanding applications has also been developed and simulated.

Ada is a registered trademark of the US Government—Ada Joint Program Office.

5.2. Operating System

The PASM operating system is distributed among all of the PASM CPUs and is logically divided into two layers: the local kernel layer and the PASMOS (PASM Operating System) layer. A local kernel is resident on all CPU boards and is responsible for local memory management, local process scheduling and synchronization, local I/O device control, local file operations, and reliable communication with other CPUs. All local kernels are identical except for their hardware dependencies (e.g., the physical locations of memory and I/O devices). PASMOS is a collection of operating system routines distributed among the PASM components. For example, the PASMOS routines for choosing the partition on which a job is to be run exist only in the System Control Unit. On the other hand, PASMOS routines for handling file load/unload requests from the PEs exist on the MCs, the processors of the Memory Management System, and on the Memory Storage Units.

Shared resources and activities that involve the control of multiple CPUs are the domain of PASMOS. The PASMOS routines use the facilities of the local kernels to perform their functions. User programs make calls to operating system functions in both layers. For example, a request for a PE data file loading operation in SIMD mode results in simultaneous calls by all PEs to their own local kernel memory allocation routines to determine where the data should be placed. The synchronization of the request (to determine when all of the files have been loaded) is a PASMOS process; however, local kernel I/O drivers are used to perform the low-level communication of the request itself. Development of the PASM prototype's operating system is underway.

PASMOS makes scheduling decisions based on processor availability and task priorities. However, it is incapable of automatically making intelligent decisions about the "best" parallel algorithm to use for a particular data set or for highest efficiency given the number of processors available. For example, several algorithms may be available to do a single task: one SIMD, one MIMD, one very fast but suitable only if a large number of PEs can be assigned to it, one that is slow but does the task particularly "well," and so on. The powerful reconfiguration capabilities of PASM can be exploited by an *Intelligent Image Understanding System* that is programmed to take best advantage of the machine [DeS85]. Such a system, which is currently under study, would be instructed to perform a task consisting of many subtasks. It would automatically select the best algorithm to execute each of the subtasks from an algorithm database and schedule these subtasks so that an optimum execution time is achieved. A precedence graph of subtasks would be employed to show the interdependence of the operations to be performed. Decisions would be based upon the current system state (i.e., available PEs, free memory, state of the subtasks) and are continually updated as the system state changes. Thus the image understanding system is capable of adapting to changing requirements. For example, it can concentrate computing power on a certain part of an image if it is determined that this part is of particular interest and it can concentrate computing power on a particular subtask to speed its execution.

Table 1 The PASM design parameters, based on current plans.

	general	full PASM	PASM prototype
Number of PEs	N	1024	16
Number of network stages (Extra Stage Cube, if 2×2 boxes)	$\log_2 N + 1$	11	5
Number of MCs	Q	32	4
Number of PEs per MC	N/Q	32	4
Number of Memory Storage Units	N/Q	32	4
Number of Memory Management System processors	fixed	≥4	4
Smallest size partition	N/Q	32	4
Maximum number of partitions	Q	32	4

6. Summary

This paper provided an overview of the PASM design concepts, the PASM prototype, and examples of parallel image processing algorithms. The PASM design is for a large-scale, dynamically reconfigurable, SIMD/MIMD parallel processing system. The system design was developed as a result of simulation studies and parallel algorithm analyses. Thus, both the system hardware and the software were application driven. A 30-processor prototype is currently under construction. Table 1 summarizes the PASM design parameters. A reading list for further information about PASM is provided at the end of this paper. The rest of this section summarizes some of the PASM design features.

The possible advantages of a reconfigurable system such as PASM include:

1. *Fault Tolerance*—If a single PE fails, only those machine partitions that must include the failed PE need to be disabled. The rest of the system can continue to function.
2. *Multiple Simultaneous Users*—Since there can be multiple independent machine partitions, there can be multiple simultaneous users of the system, each executing a different program.
3. *Program Development*—Rather than trying to debug a program on, for example, 1024 PEs, it can be debugged on a smaller size partition of 32 PEs.

4. *Variable Machine Size for Efficiency*—If a task requires only N/2 of N available PEs, the other N/2 can be used for another task.
5. *Subtask Parallelism*—Two independent subtasks that are part of the same job can be executed in parallel, sharing results if necessary.
6. *Multiple Processing Modes*—An algorithm can be executed by using either the SIMD or MIMD mode of parallelism, whichever is more efficient. A task requiring algorithms that use both modes of parallelism can be executed by using the same set of PEs.

The advantageous features of both the multistage Cube and the ADM networks include:

1. Up to N simultaneous transfers are possible.
2. They are partitionable into independent subnetworks.
3. They can be controlled in a distributed fashion using routing tags.
4. One PE can broadcast to all or a subset of the others.
5. There are a variety of implementation options for these networks.
6. They can be used in SIMD and/or MIMD operations.
7. They can support efficient global as well as local (nearest neighbor) inter-PE communications.

An additional advantage of the Extra Stage Cube is that it is single-fault tolerant.

Permanently assigning a fixed number of PEs to each MC has several advantages over allowing a varying assignment such as used in MAP [Nut77]:

1. *Scheduling*—The operating system need only schedule (and monitor the "busy" status of) Q MCs, rather than N PEs (when $Q = 32$ and $N = 1024$, this is a substantial savings).
2. *Hardware Simplicity*—No crossbar switch is needed for connecting PEs and control units (such as proposed for MAP [Nut77]).
3. *Software Simplicity*—There is no need to do the bookkeeping of recording PE to MC assignments.
4. *Network Partitioning*—The fixed assignment supports network partitioning.
5. *Secondary Storage*—The fixed assignment allows the efficient use of multiple secondary storage devices.

The main disadvantage of this approach is that the size of each partition must be a power of 2, with a minimum value of N/Q. However, for PASM's intended experimental environment, flexibility at "reasonable" cost is the goal, not maximum PE utilization.

The Memory Storage System design allows the loading/unloading of a machine partition of RN/Q PEs in R parallel block moves. The double-buffered PE memory modules permit this loading/unloading to be overlapped with PE execution. Similarly, the double-buffered MC memory modules allow the loading/unloading to be overlapped with execution. The Memory Management System, which coordinates these memory transfers, is implemented with multiple processors, providing parallelism at another level.

The PASM operating system functions are distributed over various system components to prevent the System Control Unit from being a bottleneck. A parallel assembly language for 68000-family processors exists, and a parallel C language for PASM is under development.

In conclusion, the objective of the PASM design is to achieve a system that attains a compromise between flexibility and cost effectiveness for a specific problem domain. A dynamically reconfigurable system such as PASM is a valuable tool for both image understanding and parallel processing research.

Acknowledgments—Portions of this overview of the PASM system and its use in parallel image processing are based on [SiM81b], [SiS81], [TuA83], and [MeS85]. The contributions of the coauthors on these papers are gratefully acknowledged.

References

[AdS82] G. B. Adams III and H. J. Siegel, "The extra stage cube: A fault-tolerant interconnection network for supersystems," *IEEE Transactions on Computers*, Vol. C-31, May 1982, pp. 443-454.

[AdS84] G. B. Adams III and H. J. Siegel, "Modifications to improve the fault tolerance of the extra stage cube interconnection network," *1984 International Conference on Parallel Processing*, August 1984, pp. 169-173.

[ArP76] R. G. Arnold and E. W. Page, "A hierarchical restructurable multimicroprocessor architecture," *Third Annual Symposium on Computer Architecture*, January 1976, pp. 40-45.

[BaB68] G. H. Barnes, R. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, and R. A. Stokes, "The Illiac IV computer," *IEEE Transactions on Computers*, Vol. C-17, August 1968, pp. 746-757.

[Bat76] K. E. Batcher, "The flip network in STARAN," *1976 International Conference on Parallel Processing*, August 1976, pp. 65-71.

[Bat77] K. E. Batcher, "STARAN series E," *1977 International Conference on Parallel Processing*, August 1977, pp. 140-143.

[Bat82] K. E. Batcher, "Bit serial parallel processing systems," *IEEE Transactions on Computers*, Vol. C-31, May 1982, pp. 377-384.

[BoD72] W. J. Bouknight, S. A. Denenberg, D. E. McIntryre, J. M. Randall, A. H. Sameh, and D. L. Slotnick, "The Illiac IV system," *Proceedings of the IEEE*, Vol. 60, April 1972, pp. 369-388.

[CrG72] B. A. Crane, M. J. Gilmartin, J. H. Huttenhoff, P. T. Rux, and R. R. Shively, "PEPE computer

architecture," *IEEE Computer Society Compcon 72*, September 1972, pp. 57-60.

[CrG85] W. Crowther, J. Goodhue, R. Thomas, W. Milliken, and T. Blackadar, "Performance measurements on a 128-node butterfly parallel processor," *1985 International Conference on Parallel Processing*, August 1985, pp. 531-540.

[DeS85] E. J. Delp, H. J. Siegel, A. Whinston, and L. H. Jamieson, "An intelligent operating system for executing image understanding tasks on a reconfigurable parallel architecture," *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Mangement*, November 1985, pp. 217-224.

[DuH73] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, John Wiley, New York, 1973.

[Fly66] M. J. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, Vol. 54, December 1966, pp. 1901-1909.

[Fou81] T. J. Fountain, "CLIP4: progress report," in *Languages and Architectures for Image Processing*, M. J. B. Duff and S. Levialdi, eds., Academic, New York, 1981, pp. 281-291.

[Fre61] H. Freeman, "Techniques for the digital computer analysis of chain-encoded arbitrary plane curves," *Proceedings National Electronics Conference*, Vol. 17, October 1961, pp. 421-432.

[GoL73] G. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multiprocessor systems," *First Annual Symposium on Computer Architecture*, December 1973, pp. 21-28.

[GoG83] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer—Designing an MIMD shared-memory parallel computer," *IEEE Transactions on Computers*, Vol. C-32, February 1983, pp. 175-189.

[KaK79] S. I. Kartashev and S. P. Kartashev, "A multicomputer system with dynamic architecture," *IEEE Transactions on Computers*, Vol. C-28, October 1979, pp. 704-720.

[Law75] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Transactions on Computers*, Vol. C-24, December 1975, pp. 1145-1155.

[MeS85] D. G. Meyer, H. J. Siegel, T. Schwederski, N. J. Davis IV, and J. T. Kuehn, "The PASM parallel system prototype," *IEEE Computer Society Spring Compcon 85*, February 1985, pp. 429-434.

[MiR81] O. R. Mitchell, A. P. Reeves, and K-S. Fu, "Shape and texture measurements for automated cartography," *1981 IEEE Computer Society Conference on Pattern Recognition and Image Processing*, August 1981, pp. 367.

[Nut77] G. J. Nutt, "Microprocessor implementation of a parallel processor," *Fourth Annual Symposium on Computer Architecture*, March 1977, pp. 147-152.

[Pat81] J. H. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Transactions on Computers*, Vol. C-30, October 1981, pp. 771-780.

[Pea77] M. C. Pease III, "The indirect binary n-cube microprocessor array," *IEEE Transactions on Computers*, Vol. C-26, May 1977, pp. 458-473.

[PfB85] G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, K. P. McAuliffe, E. A. Melton, V. A. Norton, and J. Weiss, "The IBM Research Parallel Processor Prototype (RP3): Introduction and architecture," *1985 International Conference on Parallel Processing*, August 1985, pp. 764-771.

[Sei85] C. L. Seitz, "The Cosmic Cube," *Communications of the ACM*, Vol. C-34, January 1985, pp. 22-33.

[SeU80] M. C. Sejnowski, E. T. Upchurch, R. N. Kapur, D. P. S. Charlu, and G. J. Lipovski, "An overview of the Texas Reconfigurable Array Computer," *AFIPS 1980 National Computer Conference*, June 1980, pp. 631-641.

[Sie77] H. J. Siegel, "Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks," *IEEE Transactions on Computers*, Vol. C-26, February 1977, pp. 153-161.

[Sie79] H. J. Siegel, "A model of SIMD machines and a comparison of various interconnection networks," *IEEE Transactions on Computers*, Vol. C-28, December 1979, pp. 907-917.

[Sie85] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*, Lexington Books, D. C. Heath, Lexington, Mass., 1985.

[SiM81a] H. J. Siegel and R. J. McMillen, "Using the augmented data manipulator network in PASM," *Computer*, Vol. 14, February 1981, pp. 25-33.

[SiM81b] H. J. Siegel and R. J. McMillen, "The multistage cube: A versatile interconnection network," *Computer*, Vol. 14, December 1981, pp. 65-76.

[SiS81] H. J. Siegel, L. J. Siegel, F. C. Kemmerer, P. T. Mueller, Jr., H. E. Smalley, Jr., and S. D. Smith, "PASM: a partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Transactions on Computers*, Vol. C-30, December 1981, pp. 934-947.

[Sto80] H. S. Stone, "Parallel computers," *in Introduction to Computer Architecture (2nd ed.)*, H. S. Stone, ed., Science Research Associates, Chicago, Ill., 1980, pp. 363-425.

[SwF77] R. J. Swan, S. Fuller, and D. P. Siewiorek, "Cm*: A modular multimicroprocessor," *AFIPS 1977 National Computer Conference*, June 1977, pp. 637-644.

[TuA83] D. L. Tuomenoksa, G. B. Adams III, H. J. Siegel, and O. R. Mitchell, "A parallel algorithm for contour extraction: Advantages and architectural implications," *1983 IEEE Computer Society Symposium on Computer Vision and Pattern Recognition*, June 1983, pp. 336-344.

[WuF80] C-L. Wu and T. Y. Feng, "On a class of multistage interconnection networks," *IEEE Transactions on Computers*, Vol. C-29, August 1980, pp. 694-702.

Reading List of Selected PASM Related Publications

General Architecture and Design:

H. J. Siegel, "Controlling the active/inactive status of SIMD machine processors," *1977 International Conference on Parallel Processing*, August 1977, pp. 183.

H. J. Siegel, P. T. Mueller, Jr., and H. E. Smalley, Jr., "Control of a partitionable multimicroprocessor system," *1978 International Conference on Parallel Processing*, August 1978, pp. 9-17.

H. J. Siegel, F. C. Kemmerer, and M. Washburn, "Parallel memory system for a partitionable SIMD/MIMD machine," *1979 International Conference on Parallel Processing*, August 1979, pp. 212-221.

H. J. Siegel, L. J. Siegel, F. C. Kemmerer, P. T. Mueller, Jr., H. E. Smalley, Jr., and S. D. Smith, "PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Transactions on Computers*, Vol. C-30, December 1981, pp. 934-947.

J. T. Kuehn, H. J. Siegel, and M. J. Grosz, "A distributed memory management system for PASM," *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, October 1983, pp. 101-108.

D. G. Meyer, H. J. Siegel, T. Schwederski, N. J. Davis IV, and J. T. Kuehn, "The PASM parallel system prototype," *IEEE Computer Society Spring Compcon 85*, February 1985, pp. 429-434.

J. T. Kuehn, T. Schwederski, and H. J. Siegel, "Design of a 1024-processor PASM system," *First International Conference on Supercomputing Systems*, December 1985, pp. 603-612.

N. J. Davis IV and H. J. Siegel, "The PASM prototype interconnection network," *1985 National Computer Conference*, July 1985, pp. 183-190.

J. T. Kuehn and H. J. Siegel, "Multifunction processing with PASM," in *Intermediate-Level Image Processing*, M. J. B. Duff, ed., Academic, New York, to appear, 1986.

Interconnection Network — Multistage Cube:

R. J. McMillen and H. J. Siegel, "The hybrid cube network," *Distributed Data Acquisition, Computing, and Control Symposium*, December 1980, pp. 11-22.

R. J. McMillen, G. B. Adams III, and H. J. Siegel, "Performance and implementation of 4x4 switching nodes in an interconnection network for PASM," *1981 International Conference on Parallel Processing*, August 1981, pp. 229-233.

H. J. Siegel and R. J. McMillen, "The multistage cube: A versatile interconnection network," *Computer*, Vol. 14, December 1981, pp. 65-76.

G. B. Adams III and H. J. Siegel, "The extra stage cube: A fault-tolerant interconnection network for supersystems," *IEEE Transactions on Computers*, Vol. C-31, May 1982, pp. 443-454.

G. B. Adams III and H. J. Siegel, "The use of 4×4 switching elements in the multistage cube network," *First International Conference on Computers and Applications*, June 1984, pp. 585-592.

G. B. Adams III and H. J. Siegel, "Modifications to improve the fault tolerance of the extra stage cube interconnection network," *1984 International Conference on Parallel Processing*, August 1984, pp. 169-173.

N. J. Davis IV and H. J. Siegel, "The performance analysis of partitioned circuit switched multistage interconnection networks," *Twelfth Annual Symposium on Computer Architecture*, June 1985, pp. 387-394.

N. J. Davis IV, W. T.-Y. Hsu, and H. J. Siegel, "Fault location techniques for distributed control interconnection networks," *IEEE Transactions on Computers*, Vol. C-34, October 1985, pp. 902-910.

N. J. Davis and H. J. Siegel, "Performance analysis of multiple-packet multistage cube networks and comparison to circuit switching," *1986 International Conference on Parallel Processing*, August 1986, pp. 108-114.

Interconnection Network — ADM:

S. D. Smith, H. J. Siegel, R. J. McMillen, and G. B. Adams III, "Use of the Augmented Data Manipulator multistage network for SIMD machines," *1980 International Conference on Parallel Processing*, August 1980, pp. 75-78.

R. J. McMillen, G. B. Adams III, and H. J. Siegel, "Permuting with the augmented data manipulator network," *Eighteenth Allerton Conference on Communica-*

tion, Control, and Computing, October 1980, pp. 544-553.

H. J. Siegel and R. J. McMillen, "Using the augmented data manipulator network in PASM," *Computer*, Vol. 14, February 1981, pp. 25-33.

R. J. McMillen and H. J. Siegel, "Performance and fault tolerance improvements in the Inverse Augmented Data Manipulator network," *Ninth Annual Symposium on Computer Architecture*, April 1982, pp. 63-72.

G. B. Adams III and H. J. Siegel, "On the number of permutations performable by the augmented data manipulator network," *IEEE Transactions on Computers*, Vol. C-31, April 1982, pp. 270-277.

R. J. McMillen and H. J. Siegel, "Routing schemes for the augmented data manipulator network in an MIMD system," *IEEE Transactions on Computers*, Vol. C-31, December 1982, pp. 1202-1214.

Interconnection Network — Multistage Cube/ADM Comparisons:

H. J. Siegel and S. D. Smith, "Study of multistage SIMD interconnection networks," *Fifth Annual Symposium on Computer Architecture*, April 1978, pp. 223-229.

H. J. Siegel, "Interconnection networks for SIMD machines," *Computer*, Vol. 12, June 1979, pp. 57-65.

H. J. Siegel, R. J. McMillen, and P. T. Mueller, Jr., "A survey of interconnection methods for reconfigurable parallel processing systems," *AFIPS 1979 National Computer Conference*, June 1979, pp. 529-542.

H. J. Siegel, "The theory underlying the partitioning of permutation networks," *IEEE Transactions on Computers*, Vol. C-29, September 1980, pp. 791-801.

G. B. Adams III and H. J. Siegel, "A survey of fault-tolerant multistage networks and comparison to the extra stage cube," *Seventeenth Annual Hawaii International Conference on System Sciences*, January 1984, pp. 268-277.

R. J. McMillen and H. J. Siegel, "Evaluation of cube and data manipulator networks," *Journal of Parallel and Distributed Computing*, Vol. 2, February 1985, pp. 79-107.

H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies*, Lexington Books, D. C. Heath, Lexington, Mass., 1985.

Distributed Operating System:

H. J. Siegel, L. J. Siegel, R. J. McMillen, P. T. Mueller, Jr., and S. D. Smith, "An SIMD/MIMD multimicroprocessor system for image processing and pattern recognition," *1979 IEEE Computer Society Conference on Pattern Recognition and Image Processing*, August 1979, pp. 214-224.

D. L. Tuomenoksa and H. J. Siegel, "Application of two-dimensional bin packing algorithms for task scheduling in the PASM multimicrocomputer system," *Nineteenth Allerton Conference on Communication, Control, and Computing*, October 1981, p. 542.

D. L. Tuomenoksa and H. J. Siegel, "Analysis of the PASM control system memory hierarchy," *1982 International Conference on Parallel Processing*, August 1982, pp. 363-370.

D. L. Tuomenoksa and H. J. Siegel, "Analysis of multiple-queue task scheduling algorithms for multiple-SIMD machines," *Third International Conference on Distributed Computing Systems*, October 1982, pp. 114-121.

D. L. Tuomenoksa and H. J. Siegel, "A distributed operating system for PASM," *Seventeenth Hawaii International Conference on System Sciences*, January 1984, pp. 69-77.

D. L. Tuomenoksa and H. J. Siegel, "Task preloading schemes for reconfigurable parallel processing systems," *IEEE Transactions on Computers*, Vol. C-33, October 1984, pp. 895-905.

D. L. Tuomenoksa and H. J. Siegel, "Task scheduling on the PASM parallel processing system," *IEEE Transactions on Software Engineering*, Vol. SE-11, February 1985, pp. 145-157.

E. J. Delp, H. J. Siegel, A. Whinston, and L. H. Jamieson, "An intelligent operating system for executing image understanding tasks on a reconfigurable parallel architecture," *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Mangement*, November 1985, pp. 217-224.

D. L. Tuomenoksa and H. J. Siegel, "Determining an optimal secondary storage service rate for the PASM control system," *IEEE Transactions on Computers*, Vol. C- 35, January 1986, pp. 43-53.

T. Schwederski and H. J. Siegel, "Adaptable software for supercomputers," *Computer*, Vol. 19, February 1986, pp. 40-48.

Parallel Programming Languages:

P. T. Mueller, Jr., L. J. Siegel, and H. J. Siegel, "A parallel language for image and speech processing," *IEEE Computer Society Fourth International Computer Software and Applications Conference*, October 1980, pp. 476-483.

C. Cline and H. J. Siegel, "A comparison of parallel language approaches to data representation and data transferral," *Computer Data Engineering Conference (COMPDEC)*, April 1984, pp. 60-66.

J. T. Kuehn and H. J. Siegel, "Extensions to the C programming language for SIMD/MIMD parallelism," *1985 International Conference on Parallel Processing*, August 1985, pp. 232-235.

C. L. Cline and H. J. Siegel, "Augmenting Ada for SIMD parallel processing," *IEEE Transactions on Software Engineering*, Vol. SE-11, September 1985, pp. 970-977.

Algorithm/Architecture Modeling and Simulation:

P. H. Swain, H. J. Siegel, and J. El-Achkar, "Multiprocessor implementation of image pattern recognition: A

general approach," *Fifth International Conference on Pattern Recognition*, December 1980, pp. 309-317.

J. T. Kuehn and H. J. Siegel, "Simulation studies of PASM in SIMD mode," *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, November 1981, pp. 43-50.

L. J. Siegel, H. J. Siegel, and P. H. Swain, "Performance measures for evaluating algorithms for SIMD machines," *IEEE Transactions on Software Engineering*, Vol. SE-8, July 1982, pp. 319-331.

J. T. Kuehn, H. J. Siegel, and P. D. Hallenbeck, "Design and simulation of an MC68000-based multimicroprocessor system," *1982 International Conference on Parallel Processing*, August 1982, pp. 353-362.

L. H. Jamieson, H. J. Siegel, E. J. Delp, and A. Whinston, "The mapping of parallel algorithms to reconfigurable parallel architectures," *ARO Workshop on Future Directions in Computer Architecture and Software*, to appear, May 1986.

T. Schwederski, H. J. Siegel, E. J. Delp, A. Whinston, and L. H. Jamieson, "Modeling the PASM parallel processing system," *Proceedings of the SIAM 1986 National Meeting*, abstract, July 1986, p. A86.

J. T. Kuehn and H. J. Siegel, "Simulation based performance measures for SIMD/MIMD processing," in *Computing Structures and Image Processing*, K. Preston, Jr., L. Uhr, M. J. B. Duff, and S. Levialdi, eds., Academic, New York, 1986.

Parallel Image Processing:

L. J. Siegel, P. T. Mueller, Jr., and H. J. Siegel, "FFT algorithms for SIMD machines," *Seventeenth Allerton Conference on Communication, Control, and Computing*, October 1979, pp. 1006-1015.

P. T. Mueller, Jr., L. J. Siegel, and H. J. Siegel, "Parallel algorithms for the two-dimensional FFT," *Fifth International Conference on Pattern Recognition*, December 1980, pp. 497-502.

H. J. Siegel and P. H. Swain, "Contextual classification on PASM," *IEEE Computer Society Conference on Pattern Recognition and Image Processing*, August 1981, pp. 320-325.

L. J. Siegel, E. J. Delp, T. N. Mudge, and H. J. Siegel, "Block truncation coding on PASM," *Nineteenth Allerton Conference on Communication, Control, and Computing*, October 1981, pp. 891-900.

H. J. Siegel, "Image processing on a partitionable SIMD machine," in *Languages and Architectures for Image Processing*, M. J. B. Duff and S. Levialdi, eds., Academic, New York, 1981, pp. 293-300.

E. J. Delp, T. N. Mudge, L. J. Siegel, and H. J. Siegel, "Parallel processing for computer vision," *Society of Photo-Optical Instrumentation Engineers Proceedings Vol. 336: Robot Vision*, May 1982, pp. 161-167.

T. N. Mudge, E. J. Delp, L. J. Siegel, and H. J. Siegel, "Image coding using the multimicroprocessor system PASM," *IEEE Computer Society Conference on Pattern Recognition and Image Processing*, June 1982, pp. 200-205.

L. J. Siegel, H. J. Siegel, and A. E. Feather, "Parallel processing approaches to image correlation," *IEEE Transactions on Computers*, Vol. C-31, March 1982, pp. 208-218.

M. R. Warpenburg and L. J. Siegel, "SIMD image resampling," *IEEE Transactions on Computers*, Vol. C-31, October 1982, pp. 934-942.

H. J. Siegel, P. H. Swain, and B. W. Smith, "Remote sensing on PASM and CDC Flexible Processors," in *Multicomputers and Image Processing: Algorithms and Programs*, K. Preston and L. Uhr, eds., Academic, New York, 1982, pp. 331-342.

D. L. Tuomenoksa, G. B. Adams III, H. J. Siegel, and O. R. Mitchell, "A parallel algorithm for contour extraction: advantages and architectural implications," *1983 IEEE Computer Society Symposium on Computer Vision and Pattern Recognition*, June 1983, pp. 336-344.

J. T. Kuehn and H. J. Siegel, "Simulation studies of a parallel histogramming algorithm for PASM," *Seventh International Conference on Pattern Recognition*, July 1984, pp. 646-649.

K. D. Smith and L. H. Jamieson, "A parallel algorithm for normalized fourier descriptors," *Twenty-Second Annual Allerton Conference on Communication, Control, and Computing*, October 1984, pp. 765-774.

J. T. Kuehn, J. A. Fessler, and H. J. Siegel, "Parallel image thinning and vectorization on PASM," *1985 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, June 1985, pp. 368-374.

T. A. Rice and L. H. Jamieson, "Parallel processing for computer vision," in *Integrated Technology for Parallel Image Processing*, S. Levialdi, ed., Academic, New York, 1985, pp. 57-78.

J. T. Kuehn, H. J. Siegel, D. L. Tuomenoksa, and G. B. Adams III, "The use and design of PASM," in *Integrated Technology for Parallel Image Processing*, S. Levialdi, ed., Academic, New York, 1985, pp. 133-152.

L. H. Jamieson, P. T. Mueller, and H. J. Siegel, "FFT algorithms for SIMD parallel processing systems," *Journal of Parallel and Distributed Computing*, March 1986, pp. 48-71.

T. A. Rice and L. H. Jamieson, "Scaling and rotational registration," in *Computing Structures and Image Processing*, M. J. B. Duff, S. Levialdi, K. Preston, and L. Uhr, eds., Academic, New York, 1986.

Parallel Speech Processing:

L. J. Siegel, "Parallel processing algorithms for linear predictive coding," *1980 IEEE International Conference*

on Acoustics, Speech, and Signal Processing, April 1980, pp. 960-963.

L. J. Siegel, H. J. Siegel, R. J. Safranek, and M. A. Yoder, "SIMD algorithms to perform linear predictive coding for speech processing applications," *1980 International Conference on Parallel Processing*, August 1980, pp. 193-196.

L. J. Siegel, "Using SIMD machines for speech analysis," *Fourteenth Annual Hawaii International Conference on System Sciences*, Vol. 1, January 1981, pp. 309-318.

E. C. Bronson and L. J. Siegel, "A parallel architecture for speech understanding," *1981 International Conference on Acoustics, Speech, and Signal Processing*, March 1981, pp. 1176-1179.

M. A. Yoder and L. J. Siegel, "Systolic array and SIMD algorithms for digital filtering," *Nineteenth Allerton Conference on Communication, Control, and Computing*, October 1981, pp. 880-889.

E. C. Bronson and L. J. Siegel, "Overview of a distributed parallel architecture for speech understanding," *Fifteenth Hawaii International Conference on System Sciences*, Vol. 1, January 1982, pp. 350-359.

M. A. Yoder and L. J. Siegel, "Dynamic time warping algorithms for SIMD machines and VLSI processor arrays," *1982 International Conference on Acoustics, Speech, and Signal Processing*, May 1982, pp. 1274-1277.

E. C. Bronson and L. J. Siegel, "A parallel architecture for acoustic processing in speech understanding," *1982 International Conference on Parallel Processing*, August 1982, pp. 307-312.

T. A. Rice and L. J. Siegel, "A parallel algorithm for finding the roots of a polynomial," *1982 International Conference on Parallel Processing*, August 1982, pp. 57-61.

L. J. Siegel and E. C. Bronson, "Parallel and distributed processing for speech analysis and recognition," *1982 Government Microcircuits Applications Conference*, November 1982, pp. 178-181.

T. A. Rice and L. J. Siegel, "Parallel processing for computationally intensive speech analysis operations," *1983 International Conference on Acoustics, Speech, and Signal Processing*, April 1983, pp. 471-474.

E. C. Bronson and L. J. Siegel, "Distributed processing for speech understanding," *1983 IEEE Trends and Applications Conference*, May 1983, pp. 126-132.

E. C. Bronson and L. J. Siegel, "A parallel architecture for labeling, segmentation, and lexical processing in speech understanding," *1983 International Conference on Parallel Processing*, August 1983, pp. 275-280.

L. J. Siegel and E. C. Bronson, "A survey of parallel and distributed speech processing," *Seventeenth Hawaii International Conference on System Sciences*, January 1984, pp. 214-223.

M. A. Yoder and L. H. Jamieson, "Simulation of a highly parallel system for word recognition," *1985 International Conference on Acoustics, Speech, and Signal Processing*, March 1985, pp. 1449-1453.

M. A. Yoder and L. H. Jamieson, "Simulation of a word recognition system on two parallel architectures," *1985 International Conference on Parallel Processing*, August 1985, pp. 171-179.

E. C. Bronson and L. H. Jamieson, "A distributed parallel architecture for speech understanding," in *Algorithmically Specialized Parallel Computers*, L. Synder, L. H. Jamieson, D. B. Gannon, and H. J. Siegel, eds., Academic, New York, 1985, pp. 139-148.

T. A. Rice and L. H. Jamieson, "Parallel processing applied to the Karhunen-Loeve Transform," *1986 IEEE-Academia Sinica Workshop on Acoustics, Speech, and Signal Processing*, April 1986, pp. 462-465.

E. C. Bronson, J. T. Kuehn, and L. H. Jamieson, "Simulation of SIMD Signal Processing Algorithms on the PASM Parallel Processing System," *1986 International Conference on Parallel Processing*, August 1986.

T. A. Rice and L. H. Jamieson, "A parallel algorithm for finding the roots of a polynomial," *IEEE Transactions on Computers*, to appear, 1986.