AN SIMD/MIMD MULTIMICROPROCESSOR SYSTEM FOR IMAGE PROCESSING AND PATTERN RECOGNITION

H. J. Siegel, L. J. Siegel,
R. J. McMillen, P. T. Mueller, Jr., S. D. Smith
Purdue University
School of Electrical Engineering
West Lafayette, IN 47907

ABSTRACT

PASM, a multimicroprocessor system being designed at Purdue University for image processing and pattern recognition, is described. This system can be dynamically reconfigured to operate as one or more independent SIMD and/or MIMD machines. The functions that the PASM operating system will perform are discussed, demonstrating how it will handle a variety of types of image processing tasks. Examples of how PASM will improve computational speeds in comparison to conventional computers are presented. In particular, smoothing, histogram, and two-dimensional FFT algorithms are analyzed.

Key words: parallel processing, SIMD, MIMD, multimicroprocessor systems, FFT, image processing.

I. INTRODUCTION

As a result of the microprocessor revolution, it is now feasible to build a dynamically reconfigurable large-scale multimicroprocessor system capable of performing image processing tasks more rapidly than previously possible. There are several types of parallel processing systems: SIMD, MSIMD, MIMD, and PSM.

An SIMD (single instruction stream - multiple data stream) machine [5] typically consists of a set of N processors, N memories, an interconnection network, and a control unit (e.g. Illiac IV [1]). The control unit broadcasts instructions to the processors and all active ("turned on") processors execute the same instruction at the same time. Each processor executes instructions using data taken from a memory to which only it is connected. The interconnection network allows interprocessor communication. An MSIMD (multiple-SIMD) system is a parallel processing system which can be structured as two or more independent SIMD machines (e.g. MAP [13]). An MIMD (multiple instruction stream - multiple data stream) machine [5] typically consists of N processors and N memories, where each processor may follow an independent instruction stream (e.g. C.mmp [41]). As with SIMD architectures, there is a multiple data stream and an interconnection network. A PSM (partitionable SIMD/MIMD) system [25] is a parallel processing system which can be structured as two or more independent SIMD and/or MIMD machines (e.g. PASM [28]).

PASM, a particular PSM-type system for image processing and pattern recognition, is currently being designed at Purdue University [25]. Due to the low cost of microprocessors, computer system designers have been considering various multimicrocomputer architectures [e.g.,2,9,11,12,18,36,39]. The system described here was the first in the literature to combine the following features:
(1) it may be partitioned to operate as many independent SIMD and/or MIMD machines of varying sizes (within certain constraints); and
(2) a variety of problems in image processing and pattern recognition are being used to guide the design choices.

Many designers have discussed the possibilities of building large-scale parallel processing systems, employing 2^{14} to 2^{16} microprocessors, in SIMD (e.g. binary n-cube array [18]) and MIMD (e.g. CHoPP [36]) configurations. Furthermore, developments in recent years have shown the importance of parallelism to image processing [7], using both cellular logic arrays (e.g. CLIP [33]) and SIMD systems (e.g. STARAN [21]). Thus, the time seems right to investigate how to construct a computer system such as the one proposed here: a machine which can be dynamically reconfigured as one or more SIMD and/or MIMD machines, optimized for a variety of important image processing and pattern recognition tasks.

II. PARALLELISM IN IMAGE PROCESSING

The use of parallel processing for image processing has been limited in the past due to cost constraints. Most systems used a small number of processors (e.g. [1]), processors of limited capabilities (e.g. [20]), or specialized logic modules (e.g. [10]). With the development of the microprocessor and related technologies, it is reasonable to consider parallel systems using a large number (e.g. 1024) of complete processors.

SIMD machines can be used for "local" processing of segments of images in parallel. For example, the image can be segmented, and each processor assigned a segment. Then, following the same set of instructions, such tasks as line thinning, threshold dependent operations, and gap filling can be done in parallel for all segments of the image simultaneously. Also in SIMD mode, matrix arithmetic used in image processing, for such tasks as statistical pattern recognition, can be done efficiently.

This research sponsored in part by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant No. AFOSR 78-3581, and by the National Science Foundation, under Grant No. ENG78-05789. The United States Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation hereon.

MIMD machines can be used to perform different "global" image processing tasks in parallel, using multiple copies of the image or one or more shared copies. For example, in cases where the goal is to locate two or more distinct objects in an image, each object can be assigned a processor or set of processors to search for it.

There are also tasks which require parallel processing in both SIMD and MIMD modes. As a simple example, consider the task of determining if a line drawing contains a square. In SIMD mode, a parallel processing system can segment the image and each processor can locally determine which points in its segment, if any, are possible corners of squares. The system can then switch to MIMD mode, where each corner will be assigned to a processor which examines the image globally to determine if the corner is actually part of a square. Another SIMD/MIMD application might involve using the same set of microprocessors for preprocessing an image in SIMD mode and then doing a pattern recognition task in MIMD mode.

More detailed examples illustrating the use of PASM will be presented in Section V.

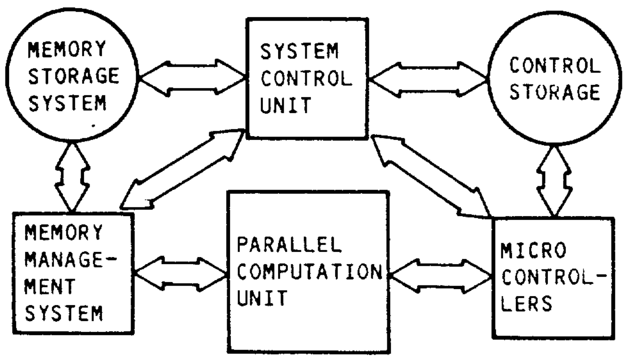

Figure 1: Block diagram overview of PASM.

III. OVERVIEW OF PASM

A block diagram of PASM (a partitionable SIMD/MIMD system) is shown in Figure 1. The heart of the system is the Parallel Computation Unit (PCU), which contains N processors, N memory modules, and the interconnection network. The PCU processors are microprocessors that perform the actual SIMD and MIMD computations. The PCU memory modules are used by the PCU processors for data storage in SIMD mode and both data and instruction storage in MIMD mode. The interconnection network provides a means of communication among the PCU processors and memory modules.

The Micro Controllers (MCs) are a set of microprocessors which act as the control units for the PCU processors in SIMD mode and orchestrate the activities of the PCU processors in MIMD mode. Control Storage (CS) contains the programs for the Micro Controllers. The Memory Management System (MMS) controls the loading and unloading of the PCU memory modules. The Memory Storage System (MSS) stores these files. The System Control Unit (SCU) is a conventional machine, such as a PDP-11, and is responsible for the overall coordination of the activities of the other components of PASM.

The processors, memory modules, and interconnection network of the PCU are organized as shown

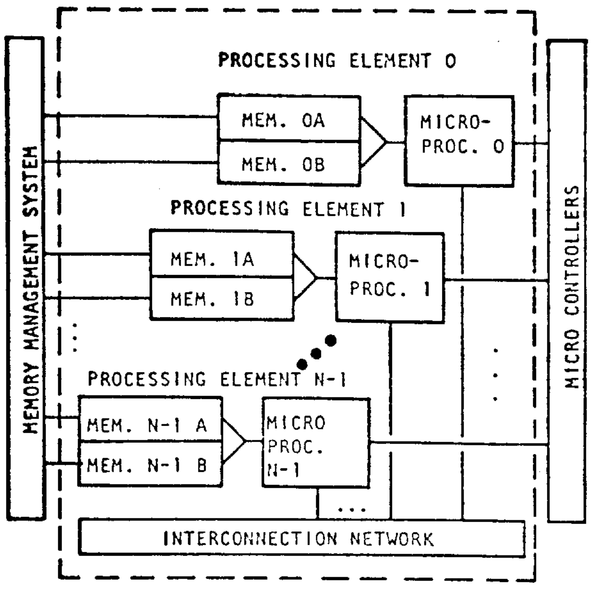

Figure 2: PASM parallel computation unit.

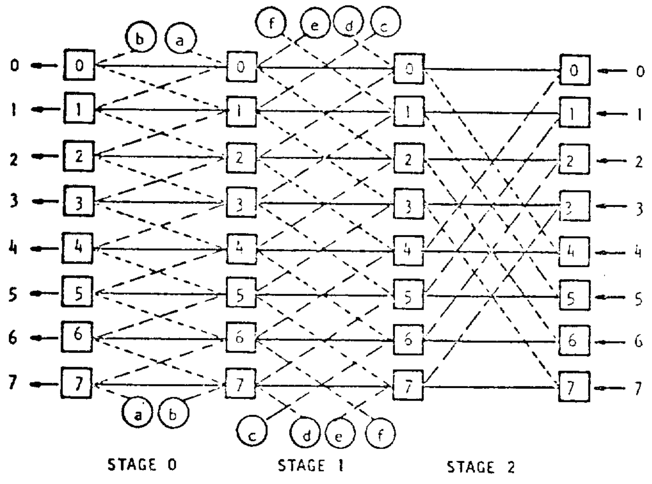

Figure 3: Augmented Data Manipulator network, N=8 (note that ⓐ connects to ⓐ, etc.).

in Figure 2. The processors, which are physically numbered (addressed) from 0 to N-1, where $N=2^n$, communicate through the interconnection network. The network being considered is a multistage implementation of the "PM2I" network [4,22,23,26] called the Augmented Data Manipulator (ADM) [27,32] (Figure 3). Each cell of the network is controlled independently. At stage x, cell j can be connected to any or all of the following cells at stage x-1: j, $j+2^x$ modulo N, and $j-2^x$ modulo N, where $0 \le x < n$. The interconnection network can be partitioned into independent sub-networks of varying sizes which are powers of two, if the physical addresses of the 2^p processors and memory modules in a partition have the same n-p low-order bits.

The PCU processors are microprogrammable microprocessors. This allows the instruction set to be tailored for parallel image processing [29].

In addition, there can be two different instruction sets, one for SIMD mode and one for MIMD mode. A pair of memory units is used for each PCU memory module so that data can be moved between one memory unit and the MSS while the PCU processor operates on data in the other memory unit.

Many computations can be more efficiently executed if the N PCU processors are partitioned into many smaller groups of processors, each group behaving like an SIMD or an MIMD machine. The method to provide multiple controllers is shown in

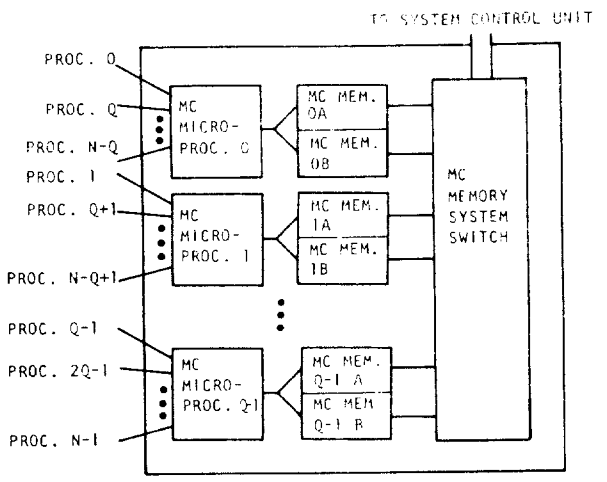

Figure 4: PASM micro controllers (MCs).

Figure 4. There are $\underline{Q=2^q}$ MCs, <u>physically addressed</u> (<u>numbered</u>) from 0 to Q-1. Each MC controls N/Q PCU processors. There is an MC memory module for each MC. Each MC memory module contains a pair of memories so that memory loading and computations can be overlapped. A virtual SIMD machine of size RN/Q, where $R=2^r$ and $1 \leq r \leq q$, is obtained by loading R MC memory modules with the same instructions simultaneously [28]. Similarly, a virtual MIMD machine of size RN/Q is obtained by combining the efforts of the PCU processors of R MCs. For either SIMD or MIMD mode, the physical addresses of these R MCs must have the same low-order q-r bits since the physical addresses of all PCU processors in a partition must agree in their low-order bits in order for the interconnection network to function properly. Possible values for N and Q are 1024 and 16, respectively.

More details about PASM and the ADM network can be found in [25,27-30,32].

IV. PASMOS - THE PASM OPERATING SYSTEM

In this section, some of the problems involved in the design of <u>PASMOS</u> - the <u>PASM</u> <u>o</u>perating <u>s</u>ystem are discussed. The operating system for PASM is different from that of "general purpose" multimicroprocessor systems, such as MAP [14], Cm* [40], or CHoPP [37], since PASM is being developed specifically for image processing and pattern recognition.

The SCU is responsible for orchestrating the MMS and the MCs. In addition, the SCU is capable of functioning as a serial processor, independent of the rest of PASM. It can handle such tasks as program development and file system supervision while the rest of PASM is executing a parallel computation. In order to perform all of these functions, the SCU will contain the PASM operating system and language compilers and assemblers. The operating system is being designed in such a way as to minimize the possibility of the SCU becoming a bottleneck. For example, the MMS will have its own set of dedicated processors and the MCs will perform many of the input-output tasks for the PCU processors.

In attempting to define the many functions that PASMOS must perform, it is instructional to look at a "typical" job and trace it through various stages of execution that occur while it is being processed by the system. The following scenario will be used to serve that purpose.

1. Input a program: Use conventional interactive facilities.
2. Compile (assemble) the user's program: The compiler (assembler) will need to include header information regarding PASM resource requirements, I/O requirements, and data formatting information, as well as standard loader information.
3. Input data from real-time device (e.g. camera) or mass storage: This requires a program to format the raw data into a structure that the MMS can handle, based on the header information mentioned above or user commands. The file management system will need to supervise distribution of data to the MSS and maintain file identification information. If the required data is already in the MSS, then PASMOS must create a table associating the data file ID with the job ID. Other desirable options may include system commands that specify I/O devices to be used and choice of format.
4. Transfer of the generated object code to CS: This requires a file management system for maintaining file identity and protection. The original object code may have come from the SCU disk, where it was temporarily stored after being compiled (assembled), or directly from the SCU during compilation (assembly), buffered in blocks. The files reside in CS on a long term basis.
5. Based on the header information and/or program declarations, set up the PCU environment: To establish a virtual SIMD or MIMD machine of size P, the SCU will need to allocate or schedule P/Q MCs and update assignment tables.
6. Transfer formated data from the MSS to the PCU memories: Instructions are issued to the MMS to transfer data from the MSS to the PCU memories associated with the MCs chosen in (5).
7. Transfer the instructions from CS to the appropriate MC memories: This will be performed (possibly simultaneously with (6)) by the loader or a system routine which uses special shared SCU/MC registers (see [28]) in conjunction with appropriate commands to the CS controller. When the MCs are ready, both the MCs and the PCU processors will be switched to read from the correct memory units.
8. Initiate task execution: The first program instructions will verify that all PCU memories have been properly loaded by checking all job and data IDs to make sure they match. Once this is accomplished the task is executed.
9. Respond to task interrupts: This can be handled by system software, special purpose hardware,

or a dedicated microprocessor (see [30]). Whatever the means, when an MC enters the wait state, the SCU must determine which task was assigned to that MC and whether the wait is due to an actual task termination, an I/O request, or an error.
10. Transfer data results from PCU memories to the MSS: This requires the SCU or MCs to issue commands to the MMS to effect the transfer.
11. Format the data for output: If the formatting required is an inherently parallel task, the PCU processors perform this function, otherwise it is taken care of by the SCU and MMS.

A PDP-11 has been used as a controller in STARAN [3] and is a promising candidate for PASM's SCU. Software is readily available to perform conventional editing, filing, and interactive I/O functions. There are many additional capabilities required of an operating system designed for a parallel processor. Six major functions that PASMOS will perform are discussed here: data formatting, file management, SCU/MC job flow instructions, task scheduling, task interrupt processing, and memory management supervision.

Data Formatting. The data being processed may have to be formatted (stored in a particular manner) in the PCU memories and the MSS to expedite efficient parallel processing. In general, image or other data that has been read in and is stored as a sequence of binary words (raw data) is not necessarily in a form readily used by the PCU. It must be reformatted in such a manner as to aid the MMS in performing a fast and efficient transfer from the MSS to the PCU memories. Data formats will be specified explicitly and/or implicitly by the user's program. Several formats will be available along with system routines for the initial conversion of the raw data to a user (or compiler) specified format, conversion between formats, and conversion from the internal formats to external formats similar to that of the original data for use by display peripherals or storage on disk or tape. Unlike the serial pre- and post-processing (data formatting) discussed in [1], the format conversion routines will use the PCU to perform those tasks that are inherently parallel, leaving only serial tasks to be performed by the SCU or MMS. Many image processing tasks are currently being investigated to determine those formats that are most useful.

Included in the information maintained in tables by PASMOS are the names of data files stored in the MSS and their format. By maintaining this information, the same data file can be stored in two or more formats. By using the MSS for long term storage of files, it is possible to reuse processed data without having to format the original data with each new application.

File Management. The file management system will maintain the usual file identity information and the data format information discussed above. It will also keep track of which data files in the MSS are associated with each program file in CS when in SIMD mode. In MIMD mode, both program and data files are stored in the MSS. There is also the option of including a coordination program to be executed by the MCs when a group of MIMD programs are executed by the PCU. This necessitates the inclusion of a table in the file management system to keep track of which program and data files stored in the MSS are associated with each coordination program file in CS.

SCU/MC Job Flow Instructions. Before a job begins executing, the data must be properly formatted and loaded into the appropriate memory units. It must be possible to control interactively the processing of image data, which may involve either SCU or MC intervention during execution. In addition, there are applications that lend themselves to the ability to specify more than one parallel task such that several independent tasks can be executing at the same time, and one parallel task then uses the results of one or more previous parallel tasks as its input. Image registration [38] is one such application that can realize greater execution speed over implementation as a single parallel task if multiple parallel tasks are allowed. Some tasks may require the same set of PCU processors to preprocess an image in SIMD mode, and then continue processing in MIMD mode.

In order to satisfy all these requirements, job flow instructions executed by the SCU or MCs are provided. The job flow instructions make it possible to specify data usage, define more than one task, conditionally execute tasks, interactively control task execution, and manipulate data files between tasks, to name just a few possibilities.

Task Scheduling. The PASMOS scheduler allocates MCs to SIMD jobs and PCU processors to MIMD jobs. It follows that the smallest number of processors that can be allocated to an SIMD job is N/Q, however a smaller number can be allocated to an MIMD job if the processors operate as a "hostless" virtual machine. Because of the high execution speed and the overhead associated with moving jobs in and out of the system, no timesharing is performed within a virtual machine in the PCU. Timesharing of the SCU is performed when there are multiple users with simultaneous requests for SCU support.

To make scheduling more efficient and to simplify its implementation, the user must specify a maximum PCU time limit to complete a job. For interactive tasks, a user must specify the maximum PCU time limit to produce a desired visual output. Thus, a job is removed from the system when it has completed (or produced its output display) or when it has reached its maximum time limit. This means an interactive job may be removed from the PCU while the user is observing the output display. Other factors that influence scheduling are the number of processors the job requires and whether it is an SIMD or an MIMD job or one requiring both modes (only a subset of the processors is capable of execution in both SIMD and MIMD modes [29]).

In those jobs containing more than one task, maximum resource requirements are based on the sum of the requirements of those tasks that can be executed simultaneously or the requirements of the largest task, whichever is greater. Information on which to make an evaluation is supplied by the compiler and is contained in the header record accompanying the compiled code.

Because the MC and PCU memories are double buffered (i.e., use dual memory units to overlap the loading/unloading with job execution), the scheduler can begin assigning resources to the next batch of tasks and start loading the instructions and data immediately after initiation of the current tasks. Once a task is complete, if the preloading of the next task has been done, both the execution of this next task and the unloading of the output of the previous task can be initiated simultaneously by switching the memory unit connections. Where jobs containing more than one task are concerned, such that the output of one or more tasks is input to another, the intelligent scheduler will load the instructions for the latter task in the memory associated with the MCs whose PCU memories contain some or all of the results of the former tasks. This kind of careful scheduling will minimize the amount of data manipulation the MMS has to do.

There are image processing tasks where a large volume of data must be processed by the same program, e.g., analyzing many "memory frames" of data from satellite sensors. In such cases, the data input requirements are known and the scheduler can easily exploit the double-buffered PCU memories and load "frames" of data in advance.

Task Interrupt Processing. There are three conditions that can cause the MCs executing a task to enter the wait state and issue an interrupt: 1) normal termination; 2) the need for particular input or output servicing that only the SCU can perform; or 3) an error condition. When an interrupt is received, the SCU must determine which task was running on the MC it is responding to, check an error flag to verify normal interrupt, and, based on the job flow instructions associated with that task, proceed to issue commands to handle the I/O need or to transfer control to the scheduler to initiate a new task. The error is signalled by means of an error register Q bits wide. Once the interrupt has been serviced, all the interrupt signals of MCs assigned to the interrupting task are cleared. An investigation of several methods that can be used to determine which task interrupted the SCU, including time, cost, and complexity analyses, appears in [30].

Memory Management Supervision. Loading and unloading the PCU memories is relegated to the MMS. The SCU and MCs act as supervisors issuing commands to the MMS. The SCU uses job information supplied by the compiler to coordinate issuance of commands to the MMS with the activities of the scheduler. For most real time applications, it must be possible to perform a parallel DMA operation to achieve a high speed transfer of data from the PCU memories to a peripheral device. Foster estimates that a parallel I/O channel 256 bits wide capable of 0.3 microseconds per store can achieve a bandwidth of 0.85 gigabits per second, which far exceeds conventional I/O channels, and is more than adequate for real time applications [6].

This section has attempted to identify many of the major functions of PASMOS. The next section demonstrates how PASM may be used to solve pattern recognition and image processing problems.

V. IMAGE PROCESSING ON PASM

One typical image processing task consists of replacing each point of an image by the value of a function which depends on the point and its neighbors. Examples of such tasks are smoothing and line thinning. Other common tasks include building histograms and performing FFTs.

Ideally, a high level language for image processing will allow algorithms for such tasks to be expressed easily. As an example, a high level language algorithm which first smooths an image and then builds a histogram is given below. The language constructs used are described in [30]. The algorithm, "average," has "pixin" as an input image and "pixout" as an output image. Both "pixin" and "pixout" have 512 by 512 pixels. The "average" routine also has as an output a histogram called "hist." Each point of pixin is an eight bit unsigned integer representing one of 256 possible gray levels.

Each point in the smoothed image, "pixout," has the average gray level of the corresponding point in "pixin" and its eight nearest neighbors. Boundary points of "pixout" are not calculated since their corresponding points do not have eight adjacent neighbors. A 256 bin (one bin for each gray level) histogram of the smoothed image is constructed and stored in "hist." This algorithm is shown in Figure 5.

Consider how this could be implemented on a system such as PASM. Suppose that 1024 processor and memory pairs (PEs) are available, and that the compiler specifies that each stores a 16 by 16 block of the 512 by 512 image. Assume that the 1024 PEs are logically arranged as an array of 32 by 32 PEs, and that the PE addresses range from 0 to 1023:

```
PE 0    PE 1  . . .   PE 31
PE 32   PE 33 . . .   PE 63
                .
PE 992        . . .   PE 1023
```

Assume that the 16 by 16 blocks are stored in row major order. Thus, 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

```
PROCEDURE average
  /*define pixin and pixout to be 512x512
  arrays of unsigned eight bit integers */
  UNSIGNED BYTE pixin[512][512], pixout[512][512];
  /*define hist to be a 256 word integer array*/
  INTEGER hist[256];
  /*define x and y to be index sets */
  INDEX x, y;
  /*declare pixin to be loaded by input data and
  pixout and hist to be unloaded as output data*/
  DATA INPUT pixin OUTPUT pixout, hist;
  /*define the sets of indices which x and y
  represent, i.e., x and y represent the
  integers between 1 and 510 inclusive */
  x = y = {1 → 510};
  /*compute average of each point and its eight
  nearest neighbors (simultaneously if possible)*/
  pixout[x][y] = (pixin[x-1][y-1]+pixin[x-1][y]+
    pixin[x-1][y+1]+pixin[x][y-1]+pixin[x][y]+
    pixin[x][y+1]+pixin[x+1][y-1]+pixin[x+1][y]+
    pixin[x+1][y+1])/9;
  /*initialize each bin to zero*/
  hist[0 → 255] = 0;
  /*compute histogram*/
  hist[pixout[x][y]] = hist[pixout[x][y]]+1;
END average
```

Figure 5: High level language algorithm for smoothing and computing histogram.

on. In general, $N = 2^n$ PEs operate on a picture of 2^k by 2^k pixels. The PEs are arranged as a $2^{n/2}$ by $2^{n/2}$ array, where n/2 is an integer, such that each PE stores in its memory a block of pixels of size $2^{k-(n/2)}$ by $2^{k-(n/2)}$.

For notational purposes, let each PE consider its 16 by 16 matrix as

$$H = \begin{vmatrix} h(0,0) & \ldots & h(0,14) & h(0,15) \\ & \ldots & & \\ h(15,0) & \ldots & & h(15,15) \end{vmatrix}.$$

Also, let the subscripts of h(i,j) extend to -1 and 16, if necessary, in order to aid in calculations across boundaries of two adjacent blocks in different PEs. For example, the pixel to the left of h(0,0) is h(0,-1), and the pixel below h(15,15) is h(16,15). So, $-1 \leq i,j \leq 16$.

A general algorithm to perform the smoothing on pixel h(i,j) to yield smoothed pixel hs(i,j) is:

```
for i ← 0 to 15 do
  for j ← 0 to 15 do
    hs(i,j) ← 1/9*(h(i+1,j)+h(i-1,j)+h(i,j+1)
                  +h(i,j-1)+h(i+1,j-1)+h(i+1,j+1)
                  +h(i-1,j+1)+h(i-1,j-1)+h(i,j))
```

The approach of this algorithm is to perform 1024 16 by 16 pixel evaluations in parallel rather than one 512 by 512 pixel evaluation as in the sequential algorithm.

At the boundaries of each 16 by 16 array, data must be transmitted between PEs in order to calculate the smoothed value, hs. For example, h(-1,0) must be transferred from the PE "above" the local PE, except for PEs 0 through 31, those at the "top edge" of the logical array of PEs (pixels on the edges of the 512 by 512 array are not smoothed). To take these data transfers into consideration, the following steps must be executed before the algorithm above. "SET ICN to PE+j" sets the interconnection network so that PE P sends data to PE P + j modulo N, $0 \leq j < N$. PEs transfer data through Data Transfer Registers. The data are loaded into the DTRin of each PE, the TRANSFER command moves the data through the network, and the final data are retrieved from DTRout [32]. Each PE has a unique address expressed in binary as a number between 0 and N-1. The command "MASK [address set]" is a PE address mask that determines which PEs will execute the instructions that follow [23,24]. The address set is specified as an n-bit number composed of 0's, 1's, and X's, where X is "don't care". Superscripts are used as repetition factors. For example, MASK $[X^9 0]$ enables only PEs whose addresses match the mask, in this case, only even numbered PEs. A negative address set disables all PEs whose addresses match. MASK $[-X^9 0]$ enables all odd numbered PEs and disables all even numbered PEs. The absence of a mask implies all PEs are active.

In order to calculate the values of hs in PE i, data must be sent from PE i+1 (as well as others). The transfer of data from PE i+1 is illustrated as follows. h'(i,j) denotes an entry in the H matrix of PE i+1.

```
         PE i                          PE i+1
h(0,0)   . . .   h(0,15)          h(0,16) = h'(0,0)
h(1,0)   . . .   h(1,15)   ←──    h(1,16) = h'(1,0)
         . . .                        . . .
h(15,0)  . . .   h(15,15)         h(15,16) = h'(15,0)
```

The compiler generated code for this transfer can be expressed as follows, assuming 16 words can be transfered as a block. Note that the mask $[-X^5 1^5]$ deactivates the PEs on the right edge of the image, i.e., PEs 31,63,...,1023.

```
    SET ICN to PE-1;
    DTRin ← h(0 → 15, 0);
    TRANSFER;
    MASK [-X^5 1^5] h(0 → 15, 16) ← DTRout;
```

The transfers of data for the remaining three sides of the array H, i.e., PEs i-1, i+32, and i-32, are accomplished in a similar manner.

The four points h(0,0), h(0,15), h(15,0), and h(15,15) require data that reside in PEs i-33, i-31, i+31, and i+33, respectively. This necessitates four additional parallel pixel transfers.

In order to perform a smoothing operation on a 512 by 512 image by the parallel smoothing of 256 point blocks of size 16 by 16, the total number of words transferred is 4(16) + 4 = 68 words. The corresponding sequential algorithm needs no data transfers between PEs, but calculates hs for 512 * 512 = 262,144 points. If no data transfers were needed, the parallel algorithm would be faster than the sequential algorithm by a factor of 262,144/256 = 1024. If it is assumed that the data transfer of each word requires about as much time as one smoothing operation, then the time factor is 262,144/324 = 809. That is, the parallel algorithm is about three orders of magnitude faster than the sequential algorithm. The approximation is a generous one, since calculating the addresses in memory of the nine pixels requires the compiler to perform nine multiplications using the subscripts [8]. Block transfers of the data on the four sides of the array could be done rapidly with a pipelined multistage interconnection network [32].

Another factor which must be considered is processor speed. An IBM 370 will process data faster than a typical microprocessor. Even with possible differences in processing speed, PASM will still perform this task two to three orders of magnitude faster.

Now consider implementing the histogram calculation. The image is distributed through 1024 PEs, as described above. The goal now is to create one 256 bin histogram, hist, of the image which will reside in PE 0. The histogram is to be used to determine a threshold value for the image.

Since the image hs is spread out over 1024 PEs, each PE will calculate a 256 bin histogram based on its segment of the image. Then these "local" histograms will be combined using the algorithm described below. This is demonstrated for N = 8, instead of 1024, in Figure 6. Until the last step, the histogram is passed as two independent halves. Two simultaneous recursive doubling algorithms sum the two halves, and the results reside in PEs 0 and 1. The last step merges the two halves into the final histogram of 256 bins. Each step of the algorithm is a data transfer followed by an addition. After step 1, even numbered PEs

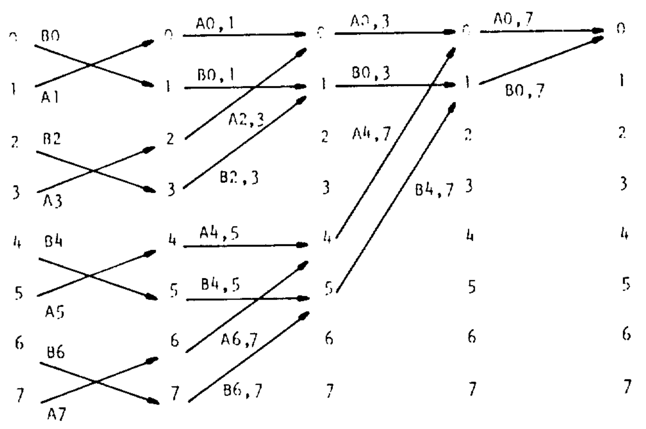

Figure 6: Histogram calculations for N=8.

Let Ai be the hist(0→127) data for PE i.
Let Bi be the hist(128→255) data for PE i.
Let Ai,j be the sum Ai + A(i+1) + ... + Aj.
Let Bi,j be the sum Bi + B(i+1) + ... + Bj.

hold all of the first half of hist, and odd numbered PEs hold all of the second half. After step 2, PEs 0 and 4 hold all of the first half of hist and PEs 1 and 5 hold the second half of hist. After step 3, PE 0 holds all of the first half of hist, and PE 1 holds all of the second half of hist. The final step merges the two halves in PE 0. Thus, N pieces of the histogram, hist, are summed to PE 0 in $((\log_2 N) + 1) * (\#\ \text{bins})/2$ parallel data transfers and additions, where, here, # bins = 256. For N = 1024, an algorithm is shown in Figure 7. (For increased efficiency, "unused" PEs are left enabled and their calculations are ignored.)

A sequential algorithm for calculating hist requires 512 * 512 = 262,144 steps. The parallel algorithm described above uses 16 * 16 = 256 steps for each PE to calculate its local histogram, and (n + 1) * 128 steps (transfer and add) to merge the histogram into PE 0, where n = 10. The difference in calculation time is thus about two orders of magnitude.

```
/*Step 1:Exchange between PE i and i+1, i even*/
MASK[X^9 0] index ←128; indx2 ←0; SET ICN to PE+1;
MASK[X^9 1] index ←0; indx2 ←128; SET ICN to PE-1;
DTRin ← hist(index → index + 127); TRANSFER;
hist(indx2 → indx2 + 127) ←
     DTRout + hist(indx2 → indx2 + 127);
/* 9 more steps yield hist(0 → 127) in PE 0, */
/* hist(128 → 255) in PE 1. */
for i = 2 to 10 do
  SET ICN to PE-2^(i-1);
  DTRin ← hist(indx2 → indx2 + 127); TRANSFER;
  hist(indx2 → indx2 + 127) ← DTRout +
    hist(indx2 → indx2 + 127);
/*merge the two halves in PE 0*/
SET ICN to PE-1;
DTRin ← hist(128 → 255); TRANSFER;
hist(128 → 255) ← DTRout;
```

Figure 7: Implementation of histogram calculation for N = 1024.

Recall that ideally the user will program in a high level language (as in Figure 5) and it will be the task of the compiler to produce a machine language implementation (as in Figure 7). In addition, there will be facilities for "parallel" programmers to program directly in PASM assembly language.

As another example of the applicability of parallel computations to image processing tasks, the two dimensional Fast Fourier Transform (FFT) is considered. The implementation of an M-point one-dimensional FFT, often used in speech processing [31], is presented first. The two-dimensional FFT on an M by M point array is then defined in terms of the one-dimensional transform.

The discrete Fourier transform (DFT) of a sequence $\{x(m)\}$, $0 \le m < M$ is defined as

$$X(k) = \sum_{m=0}^{M-1} x(m)e^{-j(2\pi/M)mk} \qquad 0 \le k < M$$

Straightforward computation requires $O(M^2)$ operations. The fast Fourier transform (decimation-in-frequency algorithm) divides $\{x(m)\}$ into sequences $\{x_1(m)\}$, equal to the first half of $\{x(m)\}$, and $\{x_2(m)\}$, equal to the second half of $\{x(m)\}$. The even and odd samples of the M-point DFT for $\{x(m)\}$ can be computed in terms of the two M/2 point DFTs:

$$X(2k) = \sum_{m=0}^{M/2-1} [x_1(m) + x_2(m)]\, W_{M/2}^{mk}$$

$$X(2k+1) = \sum_{m=0}^{M/2-1} [x_1(m)-x_2(m)]\, W_M^m\, W_{M/2}^{mk}$$

$$\text{where} \qquad W_M = e^{-j(2\pi/M)}$$

For M a power of 2, repeated application of this algorithm computes the DFT in $O(M\log_2 M)$ operations [15,19]. Figure 8 shows a flow graph of the computations in computing a 16-point FFT. Since the FFT requires $O(M\log_2 M)$ arithmetic operations, given M processors the asymptotic lower time bound is $O(\log_2 M)$. Parallel implementations of the FFT have been discussed by Pease and Stone [16-18,34].

An algorithm to perform the FFT computations on an SIMD machine which contains a virtual machine (partition) having M complete processors is presented here. The PEs are numbered from 0 to M-1. Assume the processor in each PE contains at least four fast access general purpose registers (A,B,X, and Y), and an address register (ADDRESS). For $0 \le i < M$, the register ADDRESS in PE i contains the integer i. ADDRESS (j) denotes the j-th bit of ADDRESS. Interprocessor communications are specified in terms of the Cube interconnection network [23]. Let $\underline{z} = \log_2 M - 1$. The Cube network consists of z+1 interconnections, Cube_i, $0 \le i \le z$, defined as:

$$\text{Cube}_i\ (p_z \cdots p_1 p_0) = p_z \cdots p_{i+1}\bar{p}_i p_{i-1} \cdots p_0,$$

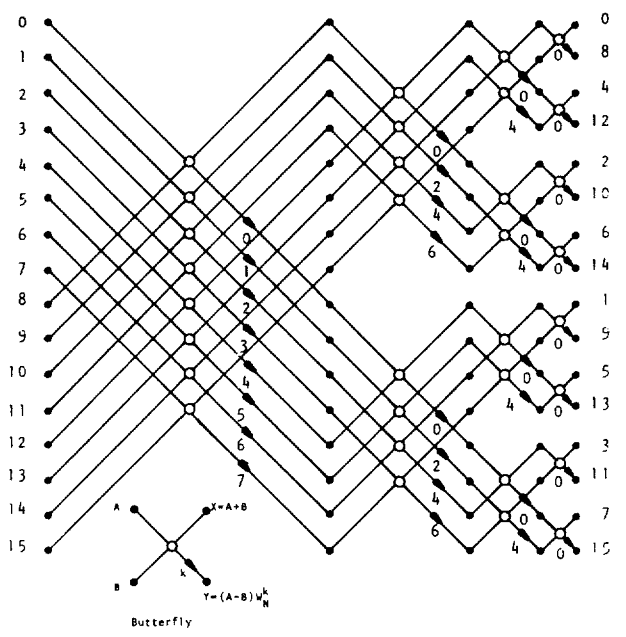

Figure 8: Computation of a 16-point FFT (decimation-in-frequency algorithm).

where $p_z \cdots p_0$ is the binary representation of an arbitrary PE address, and $\overline{p}_i$ is the complement of p_i. When the $Cube_i$ interconnection is executed by the PEs, the contents of PE j's DTRin are copied into the DTRout of PE $Cube_i(j)$. This occurs for all j simultaneously, for $0 \leq j < M$ and PE j active. Thus, $Cube_i$ connects each PE to the PE whose address differs from its address only in the i-th bit position. Each Cube interconnection is realizable in one pass by a number of networks discussed in the literature, including PASM's ADM network [27].

For the SIMD algorithm, it is assumed that the i-th sample point s(i) of the sequence for which the FFT is being computed is in the memory of PE i. The algorithm in Figure 9 uses the Cube network to perform the FFT, using PEs 0 to (M/2)-1 to perform the computations. The $Cube_z$ interconnection is used to perform the initial data alignment. Figure 10 illustrates the pattern of data transfers and computations performed. As shown in Figure 10, upon exit from the algorithm, PEs 0 to M/2-1 contain the FFT of the M-point signal {s(m)} in bit reversed order, i.e., $f(i_3 i_2 i_1 i_0)$ is in position $i_0 i_1 i_2 i_3$. PE i contains the FFT samples whose "bit reversed" indices are 2i and 2i+1 in the X and Y registers respectively (e.g., for M=16, PE 6 contains f(3=0011=reverse of 1100) and f(11=1011=reverse of 1101). To complete the algorithm, f(i) must be moved to PE i, $0 \leq i < M$. It can be shown that using the $Cube_z$ function to transfer the set of Y samples from PEs 0 to M/2-1 to PEs M/2 to M-1, respectively, produces two bit reversed sequences of length M/2, spread out over the M PEs. The samples that belong in PEs 0 to M/2-1 are in PEs 0 to M/2-1, but each sample is in the PE the low order z bits of whose address is the bit reverse of where the sample should be. Similarly, PEs M/2 to M-1 contain the samples that belong in PEs M/2 to M-1, but in bit reversed order (based on the low-order z bits). It is there-

```
/*Compute the M-point FFT using PEs 0 to M/2-1,
where z = $\log_2 M$ - 1 and initially the A register
of PE i contains s(i) and the B register of PE i
contains s(i+M/2). The "twiddle factors" $W_M^k$ have
been precomputed in each PE.*/
X ← A+B;              /*Butterfly    */
k ← ADDRESS;          /*for M-point  */
Y ← (A-B) * $W_M^k$;       /*FFT.         */
/* An $M/2^i$-point FFT is performed by a butterfly
(see Figure 8) followed by two $M/2^{i+1}$-point FFTs
executed in parallel. For an $M/2^i$-point FFT, the
twiddle factor for the r-th butterfly, $0 \leq r <$
$(M/2^i)/2$, is $W^r_{M/2^i} = W_M^{r2^i}$ [19]. */
for i ← 1 to z do
  MASK [$X^i 0 X^{z-i}$] A ← X; DTRin ← Y;      /*Align*/
  MASK [$X^i 1 X^{z-i}$] DTRin ← X; B ← Y;     /*data */
  SET ICN to $Cube_{z-i}$; TRANSFER;         /*for  */
  MASK [$X^i 0 X^{z-i}$] B ← DTRout;  /*$M/2^i$-point    */
  MASK [$X^i 1 X^{z-i}$] A ← DTRout;  /*FFT butterfly.*/
  X ← A+B;           /*Perform $2^i$ $M/2^i$-point FFT */
  k ← (ADDRESS * $2^i$) mod M/2;  /*butterflies    */
  Y ← (A-B) * $W_M^k$;                  /*in parallel. */
```

Figure 9: Algorithm to compute the M-point FFT.

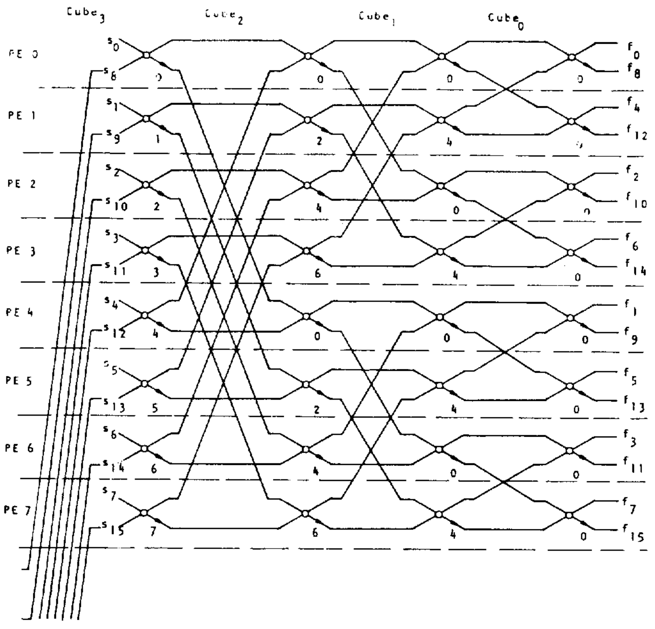

Figure 10: Computation of a 16-point FFT using the Cube interconnection network (see Figure 8 and 9).

fore necessary to apply to the two M/2 length sequences an algorithm which transfers each data point to the location equal to the bit reversed value of its current location. Pease [18] has shown that the indirect binary n-cube network can do a bit reversal in two passes. Using this information in conjunction with the equivalence and partitioning results presented in [27], it can be shown that PASM's ADM network can do the two M/2 point bit reversals in a total of two passes (plus one pass to set up the two M/2 point bit-reversed sequences).

The algorithm presented can perform M-point FFT calculations using $\log_2 M$ transfers and the bit-reversal using three data transfers. The algorithm performs $O(\log_2 M)$ parallel arithmetic operations. In a system such as PASM, the FFT algorithm would be a system function, callable as a library routine from a user's program.

The two-dimensional discrete Fourier transform of an L by M array of elements $S(l,m)$ is defined as

$$F(j,k) = \sum_{l=0}^{L-1} \sum_{m=0}^{M-1} S(l,m)\, W_L^{jl}\, W_M^{km}$$

for $0 \leq j < L$, $0 \leq k < M$. The two-dimensional transform can be decomposed in such a way as to reduce its computation to the execution of a number of one-dimensional DFTs. Performing the DFT on each row of the array yields

$$G(l,k) = \sum_{m=0}^{M-1} S(l,m)\, W_M^{km}$$

for $0 \leq l < L$, $0 \leq k < M$. The DFT of the array can then be obtained by taking the DFT of each column of G:

$$F(j,k) = \sum_{l=0}^{L-1} G(l,k)\, W_L^{jl}$$

for $0 \leq j < L$, $0 \leq k < M$. Thus, the two-dimensional transform can be obtained by computing L one-dimensional M-point transforms on the L rows of the S array, then computing M one-dimensional L-point transforms on the M columns of the G array resulting from the row transforms [15]. If L and M are powers of 2, the one-dimensional FFT algorithm can be used, and the special case where L = M is considered below. Serial implementation would require $M^2 \log_2 M$ multiplications.

The DFT is to be performed on an M by M array S. For example, assume that M = 512, and the array represents a 512 by 512 point picture. The actual one-dimensional FFT computation, without the bit reversal, needs only M/2 PEs for an M-point FFT, so the computation of the M-point FFTs on the M rows (columns) will be performed in M/2 steps, where at each step two one-dimensional FFTs are computed in parallel. S will denote the address in each PE of the first element of the portion of the original array that is stored in that PE; G will denote the address of the first element of the portion of the array representing the DFT on the rows of S. b will denote the number of bytes of storage needed for each element of the array. It is assumed that the original picture array is stored in row major order, i.e., element S(i,j) is stored in location S+i*b of PE j. G and F will be stored in skewed format [35]. This is shown for M=4 in Figure 11. G and F are stored in skewed format since this allows parallel access to both rows and columns. The algorithm to perform the two dimensional DFT is presented in Figure 12, where a virtual machine with M PEs is used. Like the one-dimensional FFT algorithm, it would be a system function callable from a user's program.

PE:	0	1	2	3
	A(0,0)	A(0,1)	A(0,2)	A(0,3)
	A(1,3)	A(1,0)	A(1,1)	A(1,2)
	A(2,2)	A(2,3)	A(2,0)	A(2,1)
	A(3,1)	A(3,2)	A(3,3)	A(3,0)

Figure 11: M by M array A stored in M PEs in skewed format, for M = 4.

The algorithm requires $M \log_2 M + 10M$ data transfers and $M \log_2 M$ parallel complex multiplications to perform the two-dimensional FFT on an M by M array. For M = 512, a total of 9728 transfers and 4608 parallel complex multiplications would be executed. Serial implementation of the FFT computation would require $M^2 \log_2 M$ complex multiplications, which for M = 512 would be 2,359,296. Thus, there is an improvement of approximately two orders of magnitude.

VI. CONCLUSIONS

PASM, a large scale partitionable SIMD/MIMD multimicroprocessor system for image processing and pattern recognition, has been presented. The functions which PASM's operating system will have to perform to support the execution of a variety of image processing tasks have been outlined. The way in which PASM can realize significant computational improvements over conventional systems has been demonstrated by the parallel formulations of smoothing, histogram, and two-dimensional FFT algorithms.

In the design of PASM, various image processing tasks have been and will be considered. The philosophy of examining the problem and then designing the machine which can best solve the problem, under certain economic and technological constraints, will be used. It is felt that this will lead to a multimicroprocessor system that will be a valuable tool for image processing and pattern recognition.

```
/* perform FFT on each row of S */
for i ← 0 to M/2-1 do
   /* FFT on rows i and i + M/2 */
   /* Transfer row i data to PEs 0 to M/2-1,
   row i + M/2 data to PEs M/2 to M-1*/
   MASK[0X^z] DTRin ← S + (i+M/2) * b;
   MASK[1X^z] DTRin ← S + i * b;
   SET ICN to Cube_z;  TRANSFER;
   /*Within each partition of M/2 PEs, load A re-
   gisters with S(j,0) to S(j,M/2-1) and B regis-
   ters with S(j,M/2) to S(j,M-1), where j=i in one
   partition and i+M/2 in the other*/
   MASK[0X^z] A ← S + i * b; B ← DTRout;
   MASK[1X^z] A ← DTRout; B ← S + (i+M/2) * b;
   call FFT in PEs 0 to M/2-1 and
        call FFT in PEs M/2 to M-1 in parallel;
   call BIT REVERSE on X & Y data in PEs 0 to
      M/2-1, placing resulting sequence in A
      registers of PEs 0 to M-1
   call BIT REVERSE on X & Y data in PEs M/2 to
      M-1, placing resulting sequence in B
      registers of PEs 0 to M-1;
   DTRin ← A; SET ICN to PE+i;  /*Store rows i    */
   TRANSFER;   G+i*b ← DTRout;  /*and i+M/2 of G  */
   DTRin ← B; SET ICN to PE+i+M/2;    /*in skewed */
   TRANSFER; G+(i+M/2)*b ← DTRout;    /*format.*/
/* perform FFT on each column of G */
for i ← 0 to M/2-1 do
   /* FFT on columns i and i+M/2 */
   /* unskew column i */
   DTRin ← G + ((ADDRESS - i) mod M) * b;
   SET ICN to PE-i; TRANSFER; X ← DTRout;
   /* unskew column i+M/2 */
   DTRin ← G + ((ADDRESS-i+M/2) mod M) * b;
   SET ICN to PE-(i+M/2); TRANSFER; Y ← DTRout;
   /*Transfer column i data to PEs 0 to M/2-1,
   column i+M/2 data to PEs M/2 to M-1*/
   MASK[0X^z] DTRin ← Y;
   MASK[1X^z] DTRin ← X;
   SET ICN to Cube_z; TRANSFER;
   /*Within each partition of M/2 PEs, load the A
   registers with G(0,j) to G(M/2-1,j) and the B
   registers with G(M/2,j) to G(M-1,j) where j=i in
   one partition and i+M/2 in the other*/
   MASK[0X^z] A ← X; B ← DTRout;
   MASK[1X^z] A ← DTRout; B ← Y;
   call FFT in PEs 0 to M/2-1 and
        call FFT in PEs M/2 to M-1 in parallel;
   call BIT REVERSE on X & Y data in PEs 0 to
      M/2-1, placing resulting sequence in A
      registers of PEs 0 to M-1;
   call BIT REVERSE on X & Y data in PEs M/2 to
      M-1, placing resulting sequence in B
      registers of PEs 0 to M-1;
   /*Store columns i and i+M/2 of F in place of
   corresponding G array elements (skewed
   storage)*/
   DTRin←A; SET ICN to PE+i; TRANSFER;
   G+((ADDRESS-i) mod M) * b←DTRout;
   DTRin←B; SET ICN to PE+i+M/2; TRANSFER;
   G + ((ADDRESS-i+M/2) mod M) * b←DTRout
```

Figure 12: Two-dimensional FFT algorithm. FFT on M by M array, using M PEs.

REFERENCES

1 Bouknight, W. J., et al., "The Illiac IV system," Proc. IEEE, Vol. 60, Apr. 1972, pp. 369-388.

2 Briggs, F., Fu, K. S., Hwang, K., and Patel, J., PM4: A Reconfigurable Multiprocessor System for Pattern Recognition and Image Processing, School of Electrical Engineering, Purdue University, Technical Report TR-EE 79-11, Mar. 1979.

3 Davis, E. W., "STARAN parallel processor system software," Nat'l. Computer Conf., May 1974, pp. 17-22.

4 Feng, T., "Data manipulating functions in parallel processors and their implementations," IEEE Trans. Comp., Vol. C-23, Mar. 1974, pp. 309-318.

5 Flynn, M. J., "Very high-speed computing systems," Proc. IEEE, Vol. 54, Dec. 1966, pp. 1901-1909.

6 Foster, C. A., Content Addressable Parallel Processors, Van Nostrand, Reinhold, New York, N.Y., 1976.

7 Fu, K. S., "Special computer architectures for pattern recognition and image processing - an overview," Nat'l. Computer Conf., June 1978, pp. 1003-1013.

8 Gries, D., Compiler Construction for Digital Computers, John Wiley & Sons, Inc., New York, N.Y., 1971.

9 Keng, J., and Fu, K. S., "A special computer architecture for image processing," 1978 IEEE Comput. Soc. Conf. Pattern Recognition and Image Processing, June 1978, pp. 287-290.

10 Kruse, B., "A parallel picture processing machine," IEEE Trans. Comp., Vol. C-22, Dec. 1973, pp. 1075-1087.

11 Lipovski, G. J., "On a varistructured array of microprocessors," IEEE Trans. Comp., Vol. C-26, Feb. 1977, pp. 125-138.

12 Lipovski, G. J., and Tripathi, A., "A reconfigurable varistructure array processor," 1977 Int'l. Conf. Parallel Processing, Aug. 1977, pp. 165-174.

13 Nutt, G. J., "Microprocessor implementation of a parallel processor," 4th Symp. Comp. Arch., Mar. 1977, pp. 147-152.

14 Nutt, G. J., "A parallel processor operating system comparison," IEEE Trans. Software Engineering, Vol. SE-3, Nov. 1977, pp. 467-475.

15 Oppenheim, A. V. and Schafer, R. W., Digital Signal Processing, Prentice-Hall, Englewood Cliffs, N.J., 1975.

16 Pease, M. C., "An adaptation of the fast Fourier transform for parallel processing," JACM, Vol. 15, Apr. 1968, pp. 252-264.

17 Pease, M. C., "Organization of large scale Fourier processors," JACM, Vol. 16, July 1969, pp. 474-482.

18 Pease, M. C., "The indirect binary n-Cube microprocessor array," IEEE Trans. Comp., Vol. C-26, May 1977, pp. 458-473.

19 Rabiner, L. R. and Gold, B., Theory and Application of Digital Signal Processing, Prentice-Hall, Englewood Cliffs, N.J., 1975.

20 Rohrbacker, D., and Potter, J. L., "Image processing with the Staran parallel computer," Computer, Vol. 10, Aug. 1977, pp. 54-59.

21 Ruben, S., Faiss, R., Lyon, J., and Quinn, M., "Application of a parallel processing computer in LACIE," 1976 Int'l. Conf. Parallel Processing, Aug. 1976, pp. 24-32.

22 Siegel, H. J., "Single instruction stream - multiple data stream machine interconnection network design," 1976 Int'l. Conf. Parallel Processing, Aug. 1976, pp. 273-282.

23 Siegel, H. J., "Analysis techniques for SIMD machine interconnection networks and the effect of processor address masks," IEEE Trans. Comp., Vol. C-26, Feb. 1977, pp. 153-161.

24 Siegel, H. J., "Controlling the active/inactive status of SIMD machine processors," 1977 Int'l Conf. Parallel Processing, Aug. 1977, p. 183.

25 Siegel, H. J., "Preliminary design of a versatile parallel image processing system," Third Biennial Conf. on Computing in Indiana, Apr. 1978, pp. 11-25.

26 Siegel, H. J., "Partitionable SIMD computer system interconnection network universality," 16th Annual Allerton Conf. on Communication, Control, and Computing, Oct. 1978, pp. 586-595.

27 Siegel, H. J., and Smith, S. D., "Study of multistage SIMD interconnection networks," 5th Symp. Comp. Arch., Apr. 1978, pp. 223-229.

28 Siegel, H. J., Mueller, P. T., Jr., and Smalley, H. E., Jr., "Control of a partitionable multimicroprocessor system," 1978 Int'l. Conf. Parallel Processing, Aug. 1978, pp. 9-17.

29 Siegel, H. J., and Mueller, Jr., P. T., "The organization and language design of microprocessors for an SIMD/MIMD system," 2nd Rocky Mt. Symp. on Microcomputers, Aug. 1978, pp. 311-340.

30 Siegel, H. J., McMillen, R. J., Mueller, P. T., Jr., and Smith, S. D., A Versatile Parallel Image Processor: Some Hardware and Software Problems, School of Electrical Engineering, Purdue University, Technical Report TR-EE 78-43, Oct. 1978.

31 Siegel, L. J., "Features for the identification of mixed excitation in speech analysis," 1979 IEEE Int. Conf. Acoust., Speech, Signal Processing, Apr. 1979, pp. 752-755.

32 Smith, S. D., and Siegel, H. J., "Recirculating, pipelined, and multistage SIMD interconnection networks," 1978 Int'l. Conf. Parallel Processing, Aug. 1978, pp. 206-214.

33 Stamopoulous, C. D., "Parallel algorithms for joining two points by a straight line segment," IEEE Trans. Comp., Vol. C-23, June 1974, pp. 642-646.

34 Stone, H. S., "Parallel processing and the perfect shuffle," IEEE Trans. Comp., Vol. C-20, Feb. 1971, pp. 153-161.

35 Stone, H. S., "Parallel Computers," in Introduction to Computer Architecture, H. S. Stone, editor, Science Research Associates, Chicago, IL, 1975, pp. 327-355.

36 Sullivan, H., Bashkow, T. R. and Klappholz, K., "A large scale homogeneous, fully distributed parallel machine," 4th Symp. Comp. Arch., Mar. 1977, pp. 105-124.

37 Sullivan, H., et al., "The node kernel: resource management in a self-organizing parallel processor," 1977 Int'l. Conf. Parallel Processing, Aug. 1977, pp. 157-162.

38 Svedlow, M., Analytical and Experimental Design and Analysis of an Optimal Processor for Image Registration, Ph.D. Thesis, Purdue University, Aug. 1976.

39 Swan, R. J., Fuller, S. H., and Siewiorek, D. P., "Cm*: a modular, multi-microprocessor," Nat'l. Computer Conf., June 1977, pp. 645-655.

40 Swan, R. J., et al., "The implementation of the Cm* multi-microprocessor," Nat'l. Computer Conf., June 1977, pp. 645-655.

41 Wulf, W. A., Bell, C. G., "C.mmp - a multi-miniprocessor," Proc. FJCC, Dec. 1972, pp. 765-777.