MULTIPROCESSOR IMPLEMENTATION OF IMAGE PATTERN RECOGNITION: A GENERAL APPROACH

Philip H. Swain, Howard Jay Siegel, and Joseph El-Achkar
School of Electrical Engineering
Purdue University
West Lafayette, IN 47907

Abstract

Application of a class of parallel processing architectures for implementation of image processing and, more specifically, image pattern recognition is proposed. Noting the problem of optimally matching a wide variety of possible processor configurations to a broad spectrum of different algorithms, a general framework for characterizing image processing algorithms is proposed, and the task of mapping the algorithm characteristics into processing system characteristics is begun. A simple image pattern recognition method is considered from this point of view. An algorithm is developed and alternative SIMD (single instruction stream - multiple data stream) parallel implementations are compared. Complexity analyses are presented to show the computational speedup made possible by the parallelism.

1. Introduction

Raw computational power, measured in terms of the number of operations a computer system can perform in unit time, is cheap! Many applications of image pattern recognition require very large amounts of computational power, especially if they involve anything more than the most elementary pattern features and processing techniques. Today's challenge is to make lots of cheap computational power available for achieving practical image pattern recognition. An approach to meeting the challenge is the subject of this paper.

The use of parallel processing for image pattern recognition and other image processing tasks has been limited in the past due to cost constraints. Most systems have used small numbers of processors [1], processors of limited capability [2], or specialized logic modules [3,4]. With the development of the microprocessor and related technologies it has become reasonable to consider parallel systems using a very large number of processors, e.g., a thousand or even thousands.

Multiprocessor implementation of pattern recognition algorithms requiring large amounts of computation offers the possibility of making such algorithms more useful for practical, sometimes real-time, applications. However, this raises the interesting problem of matching, in an optimal way, very diverse algorithm characteristics with at least equally diverse multiprocessor architectures. A schema is proposed for analyzing image processing and pattern recognition algorithms in order to determine how best to exploit parallel implementations. To bound the situation somewhat, the candidate parallel implementations are limited to SIMD (single instruction stream - multiple data stream), MSIMD (multiple-SIMD) and pipeline architectures.

Section II contains a brief introduction to some principal parallel processing architectures. In Section III, a framework for algorithm analysis and parallel implementation of algorithms is suggested. An illustration of the approach, based on a simple method for image pattern recognition, is presented in Section IV. In Section V, directions for pursuing this research are outlined.

2. Some Multiprocessor Architectures

In order to somewhat bound initially the generality of the algorithm/processing system matching problem, only the more tightly coupled classes of multiprocessor systems will be considered.

2.1 SIMD Machines

Typically, an SIMD (single instruction stream - multiple data stream) machine [5] is a computer system consisting of a control unit, N processors,

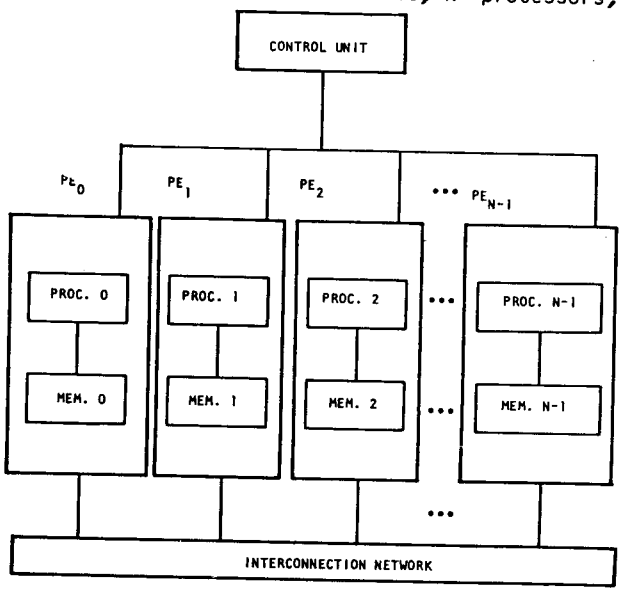

Fig. 2.1. A general model of an SIMD machine.

This research was supported by the Defense Mapping Agency, monitored by the United States Air Force Rome Air Development Center Information Sciences Division, under Contract No. F30602-78-C-0025 through the University of Michigan.

CH1499-3/80/0000-0309$00.75©1980 IEEE

N memory modules, and an interconnection network. The model used here is shown in Fig. 2.1, where each processor is paired with a memory module, forming a processing element (PE). The control unit broadcasts instructions to all of the processors, and all active processors execute the same instruction at the same time. Thus, there is a single instruction stream. Each active processor executes the instruction on data in its own associated memory module. Thus, there is a multiple data stream. The interconnection network, sometimes referred to as an alignment or permutation network, provides a communication facility for the processors and memory modules.

To demonstrate how SIMD machines operate, consider the following simple task. Assume that A, B, and C are each one-dimensional arrays (vectors) and that the task to be performed is the elementwise addition of A and B, storing the result in C. This computation will take N steps on a serial machine.

Assume that A, B, and C are stored in an SIMD machine, with N PEs, such that A(i), B(i), and C(i) are all stored in the memory of PE i, $0 \leq i < N$. To perform an elementwise addition of the vectors A and B and store the result in C, all PEs would execute (simultaneously) "C ← A + B" with PE i doing the addition of A(i) and B(i), storing the result in C(i). Thus, in this case, the SIMD machine does in one step a task requiring N steps on a serial processor.

Consider a variation on this example. Assume the N-step serial task is C(i) ← A(i) + B(i-1), $1 \leq i < N$, and C(0) ← A(0). Given the i-th element of each vector is initially in PE i, an SIMD machine does this task in three different steps.

(1) The value of B(i-1) is moved, through the interconnection network, from PE i-1 to PE i, $1 \leq i < N$. Most proposed and existing SIMD interconnection networks can do this in one parallel data transfer [6].

(2) In PE i, add A(i) to B(i-1) and store the result in C(i), $1 \leq i < N$ (PE 0 is disabled, all other PEs do this simultaneously).

(3) In PE 0, store A(0) in C(0) (all other PEs are disabled).

This example demonstrates the need for the interconnection network and methods for disabling PEs.

SIMD machines can be used to achieve "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 and fast Fourier transforms, can be done efficiently.

2.2 Multiple - SIMD Systems

An MSIMD (multiple-SIMD) system is a parallel processing system which can be structured as one or more independent SIMD machines. A general model of an MSIMD system is shown in Fig. 2.2. The original design of the Illiac IV was as an MSIMD system [1]. As the microprocessor revolution makes processors less expensive, multimicroprocessor systems which can operate in MSIMD mode are being proposed [7-12]. These MSIMD designs consist of a large

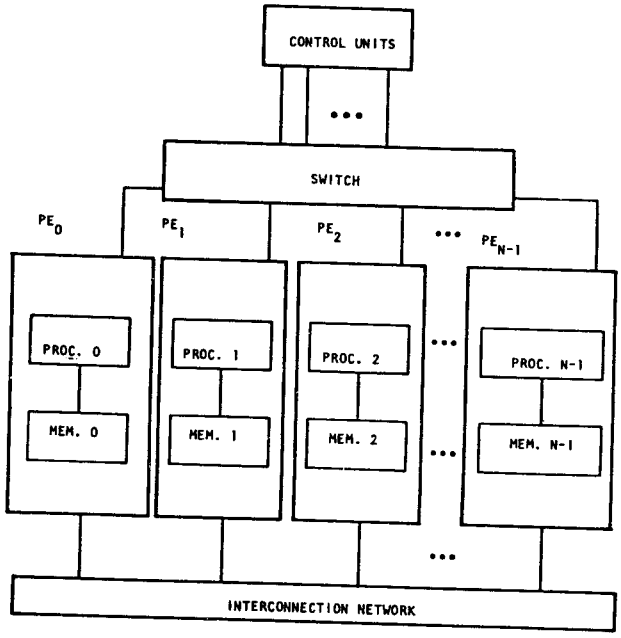

Fig. 2.2. A general model of a multiple-SIMD machine.

number of processors which can be partitioned into groups ("logical" SIMD machines) of varying sizes. Different configurations can be set up at different times.

In general, the possible advantages of an MSIMD system over an SIMD system with a similar number of PEs are as follows.

(1) fault detection - When high reliability is needed, have three partitions run the same job. If one partition produces results different from the others, assume it is faulty.

(2) fault tolerance - If a single PE fails, only those logical SIMD machines (partitions) which must include the failed PE need to be disabled.

(3) multiple simultaneous users - Since there can be multiple independent logical SIMD machines, there can be multiple simultaneous users, each executing a different SIMD program.

(4) program development - Rather than trying to debug an SIMD program on, for example, 1024 PEs, it can be debugged on a smaller size logical SIMD machine of 16 or 32 PEs.

(5) vary 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.

(6) subtask parallelism - Two independent SIMD subtasks that are part of the same job can be executed in parallel, sharing results if necessary.

The importance of each of the above advantages must be determined in the context of specific applications. In particular, advantages 5 and 6 involve a complicated analysis considering factors such as machine size, image sizes, input/output rate, number of images to be processed, system utilization, and measures of efficiency.

2.3 A Pipeline of Processors

The third class of multiprocessor architectures to be considered is a pipeline of processors. A general model of a pipeline is shown in Fig. 2.3.

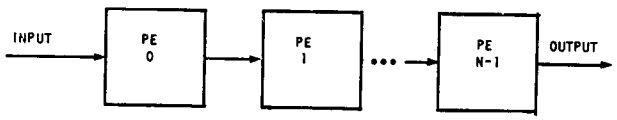

Fig. 2.3. A general model of a pipeline of processors.

PE 0 processes an input, I(k), then outputs a result, FO(I(k)), to PE 1. PE 1 then computes F1(FO(I(k))) while PE 0 processes a new input by computing FO(I(k+1)). In general, PE i inputs from PE i-1 the value Fi-1(Fi-2...(FO(I(k))) ...) and outputs to PE i+1 the value Fi(Fi-1...(FO(I(k)))...). While PE i is doing this computation based on input I(k), PE i-j is doing its computation based on input I(k+j).

As an example of a task which could be organized for a pipeline of processors, consider the "maximum likelihood classification" of pixels [13]. Assume that each pixel of an image is to be classified as being in one of C classes using the decision rule that pixel P is in class J if the maximum value of F(i,P), $0 \leq i < C$, occurs for i=J. Each pixel P may be represented as a vector of values; for example, Landsat data is typically composed of four spectral bands.

This task may be organized for a pipeline of C processors, numbered from 0 to C-1, as follows. PE 0 receives as its input the pixel P(k). Its output to PE 1 is a three-tuple, P(k), temp = F(0,P(k)), and 0 (the class number associated with temp). PE 1 outputs to PE 2 the three-tuple P(k), temp = max(F(1,P(k)), temp), and temp-class (the class number associated with the new temp). In general PE i inputs from PE i-1 the three-tuple P(k), temp = max(F(i-1,P(k)), temp), and temp-class, and outputs to PE i+1 the three-tuple P(k), temp = max(F(i,P(k)), temp), and temp-class (the class number associated with the new temp). That is, PE i computes F(i,P(k)) and compares it to the maximum of F(j,P(k)), $0 \leq j < i$. The parallelism is achieved by having PE 0 process pixel P(k+C-1), while PE 1 processes pixel P(k+C-2), ..., while PE C-1 processes pixel P(k).

An important feature of any computation pipeline is that all of the stages of the pipe require approximately the same amount time to execute [14]. This will be true for the above example.

The F(i,P) calculations in a maximum likelihood classification generally involve vector and matrix arithmetic operations. Thus, for this example, the pipeline of processors could be replaced by a pipeline of SIMD machines, a special case of MSIMD parallelism.

3. Analysis of Image Processing Algorithms For Multimicroprocessor Implementations

The problem domain of image processing includes a large class of algorithms exhibiting widely varying characteristics and degrees of potential for benefitting from multiprocessor implementation. There exist numerous alternatives for implementing these algorithms using multimicroprocessor systems, and it is desirable to match implementation with algorithm in an optimal way. Developing systematic methods for achieving such a match is the goal of this investigation. For this study, the problem domain is narrowed to implementations based on SIMD, MSIMD, and pipeline architectures. Our approach is to attempt to develop a way to parameterize the algorithms or classes of algorithms, parameterize multiprocessor architectures or classes of architectures, and then find a mapping from algorithms to architectures based on the parameterizations. It is required that it be possible to associate a measure of "goodness" or figure of merit with the mapping, possibly a combination of factors such as processing speed, implementation cost, system reliability, and system utilization.

3.1 Classification of Image Processing Algorithms

There are many different ways to classify image processing algorithms. Six classification criteria are discussed briefly here and it will subsequently be shown why these are particularly relevant to multiprocessor implementation.

(1) type of operation - Image processing operations may be loosely classified as "enhancement" or "information extraction" operations. Enhancement operations typically involve image-to-image transformations of various levels of complexity (e.g., edge enhancement). Information extraction operations usually result in a significant reduction in the size of the data stream from input to output (e.g., pixel or region classification).

(2) context-free versus context-dependent - Some algorithms operate on each pixel without regard to neighboring pixels (e.g., gray-level histogramming). Others are "neighborhood-dependent," where the size and shape of the neighborhood are at least algorithm-dependent and may even be data-dependent (e.g., high-pass filtering).

(3) single-pass versus multipass - For many operations, only a single pass through the image data is required, i.e., operations are performed serially on the incoming data after which these data are no longer needed (e.g., run-length encoding for data compression). But for some operations, multiple passes through the image data are required, so that further access to the original data must be possible (e.g., gray scale display based on histogram equalization). For some iterative processes (many clustering algorithms, for example), the number of passes required may be relatively large and unknown beforehand.

(4) univariate versus multivariate - Both input and output considerations are relevant, as either or both may be univariate or multivariate in any combination. Some examples are shown in Table 3.1.

Table 3.1 Examples of Univariate and Multivariate Image Processing Operations.

		INPUT	
		Univariate	Multivariate
OUTPUT	Univariate	Interpolation for display enlargement	Multispectral data classification
OUTPUT	Multivariate	Spatial feature extraction	Multispectral feature extraction

Multispectral data is a familiar type of multivariate imagery which might be the input to the algorithm, but the multiple variables could as well be digitally registered maps containing diverse types of information (e.g., imagery, topographic data, geomagnetic data, land use information).

(5) real-time versus non-real-time - Although this is usually a characteristic of the application environment, it is sometimes a useful characterization of processing operations as well. Analog-to-digital (A/D) conversion is an operation most frequently found in real-time data acquisition applications. On the other hand, multivariate classification is most often restricted to non-real-time processing, when conventional computer architectures are employed, because of the amount of computation involved.

(6) computational complexity - If P is the number of pixels to be processed (sometimes expressed as a total, sometimes as pixels/image), it is important to know the time required for processing as a function of P. (Space complexity is also a relevant issue but will not receive attention here.) For sufficiently large P, it is generally possible to express the time complexity in the form "$T(P) = a \cdot f(P) + b$" where f(P) may be a linear, quadratic, logarithmic, etc., function of P and a and b are constants. For conventional serial computers, f(P) depends primarily on inherent characteristics of the algorithm. For multiprocessing systems, however, f(P) depends on the system configuration as well (e.g., number of PEs, type of network) [15]. The parameters a and b may depend on such factors as the data type (floating point versus integer) and the sophistication of the programming used for the implementation.

Table 3.2 shows how these characteristics of image processing algorithms impact a number of implementation requirements. In several instances, noted in the table by AD, the impact of the algorithm characteristic on the implementation requirements is considered more highly dependent on the specific algorithm in question than on the general class of algorithms to which it belongs.

3.2 Implementation Considerations

Numerous factors must be considered in arriving at the most suitable system configuration. The factors include: algorithm characterization (as discussed in Section 3.1), nature of the application environment, allocation of processors, ratio of the number of processors available to the number of pixels to be processed, I/O speed compared to processor speed, interconnection network speed compared to processor speed, interconnection network flexibility, processor speed compared to secondary memory speed, PE program memory size, and fault tolerance requirements.

The nature of the application environment and the algorithm characterization are the chief factors in determining minimum acceptable and achievable performance characteristics of the processing system. The application determines the rate at which data will become available to the system and the required system throughput. These considerations together with the complexity of the image processing algorithms determine whether the system throughput will effectively be I/O limited or computation limited. In this paper, the computation limited situation is addressed.

Assume that a fixed, relatively large number of

Table 3.2 The impact of algorithm characteristics on six implementation requirements.

	IMPLEMENTATION REQUIREMENTS					
	I/O capacity	I/O speed	Processor interconnection	SIMD/MSIMD/ pipeline compatibility	Floating-point (f.p.) arithmetic	Primary memory
Type of operation	Enhancement: uniform info extraction: reducing	Online: very high speed Offline: variable	AD	AD	Often req'd for info extraction	AD
Context-free/ context-dependent	AD	AD	High complexity for context-dependent	AD	AD	High for context-dependent
Single-pass/ multipass	Depends on primary memory size	Depends on primary memory size	AD	AD	May need f.p. to hold precision	Usually high for multipass
Univariate/ multivariate	Higher for multivariate	AD	May be very complex if context-dependent	AD	May need f.p. to hold precision	High for multivariate
Real-time/non-real-time	Depends on data sources(s)	High speed usually required	AD	Pipeline may best accommodate data flow	Hardware f.p. needed for real-time	AD
Computational complexity	AD	Often inversely related	AD	AD	Most complex algorithms require fast f.p.	AD

AD = algorithm-dependent

microprocessors is available for the system implementation. If the system throughput is computation limited, then the primary consideration in allocating or configuring the microprocessors will be maximizing the effective computational speed of the system. Given N microprocessors, the idealized speedup, compared to a single microprocessor, is a factor of N, and this may be taken as a design goal.

Consider the following four strategies for allocating N microprocessors to the task of processing an N-by-N image: (1) pixel-per-processor; (2) row ("line")-per-processor; (3) subimage-per-processor; and (4) subimage-per-group-of-processors. To clarify, note that a "pixel-per-processor" allocation means that each processor receives the data for one pixel at a time and is immediately responsible for carrying out the computations needed to get the results for that pixel. (It does not necessarily imply that a distinct processor is available and assigned to process each pixel in the image in parallel with all of the other pixels.) In the "row-per-processor" allocation, an entire row or line of the image is loaded into the processor memory, and a "stage" of processing is complete when all processors finish with their respective rows. Similarly, the subimage-per-processor allocation assigns a rectangular subimage to each processor; e.g., each of N processors might be assigned to a $\sqrt{N}$-by-$\sqrt{N}$ subimage of the N-by-N image. The subimage-per-group-of-processors allocations is similar to the previous scheme, except that two or more processors may be assigned to a given subimage. Thus the allocation strategy may treat picture elements and processing elements as well either individually or in groups.

The various implementation considerations discussed above interact in determining the appropriate allocation strategy. If the algorithm in question is the context-free variety, then any of the first three allocation strategies will provide roughly the factor N speedup. Therefore, the choice may be based on the size and ease of loading of the primary memory associated with each processor. On the other hand, if the algorithm is context-dependent, i.e., if the computations involve pixel neighborhoods, then the characteristics of the processor interconnections must be considered. In this case, the pixel-per-processor implementation requires the largest number of interprocessor data transfers although it has the simplest (most regular) interconnect logic. The subimage-per-processor implementation requires less actual transfer of data but has somewhat more complex transfer logic. (Implicit in this discussion is the assumption that the image data to be used by each processor is stored in the primary memory associated with the processor.) In either case the speedup factor will be somewhat less than N due to the time required for the transfer operations.

MSIMD and pipelined schemes are of greatest advantage when the processing algorithm produces multiple outputs which can be computed in parallel and/or the computations are complex but can be decomposed to parallel computation of intermediate results. Again, processing resources can be allocated on a pixel-per-processor or subimage-per-processor basis, and the remarks above concerning context-free and context-dependent algorithms apply equally for these cases.

4. Parallel Implementation of a Simple Pattern Recognition Algorithm

In this section, an algorithm is developed and analyzed, and alternative parallel implementations are compared. Complexity analyses are performed to show the computational speedup achieved by each of the alternative implementations.

The pattern recognition method considered here involves "feature identification using signatures of the regional meso-structure" [16] to recognize features of interest in 1:110,000-scale aerial photography. The "regional meso-structure" is defined as a 3-by-3 or 5-by-5 matrix of pixels about each pixel. For each pixel, a set of parameters is computed consisting of: PMAX = the maximum gray level in the matrix, PMIN = the minimum gray level in the matrix, β = PMAX - PMIN, and ν = number of distinct gray levels in the matrix. Comparing the values of these parameters to predetermined thresholds, each pixel is classified as "woods," "large field," "water," "urban area," or "small field."

Only the computation of PMAX, PMIN and β will be considered here. Assume the image to be an M-by-M array, and the meso-structure a 3-by-3 matrix of pixels about each pixel.

4.1. Algorithm Description

The main idea underlying the algorithm to be used for computing PMAX and PMIN is the following (refer to Fig. 4.1). First compare pixels 1, 2 and 3 and get TMAX and TMIN for these three pixels. Then do the same successively for pixels 4, 5 and 6 and 7, 8 and 9. Finally, compare the TMAX and TMIN values of the three rows, obtaining the maximum and minimum for the whole 3-by-3 matrix, PMAX and PMIN. This is accomplished by the following general algorithm, in which PMAX, PMIN, and β are computed for pixel (i,j) (refer to Fig. 4.1).

```
Subalg. A: Find TMAX and TMIN for row i.
 if P(i,j) ≤ P(i,j+1) /*look right*/
      then  TMAX(i,j) ← P(i,j+1)
            TMIN(i,j) ← P(i,j)
      else  TMAX(i,j) ← P(i,j)
            TMIN(i,j) ← P(i,j+1)
 if P(i,j-1) ≥ TMAX(i,j) /*then look left*/
      then  TMAX(i,j) ← P(i,j-1)
      else if P(i,j-1) < TMIN(i,j)
            then  TMIN(i,j) ← P(i,j-1)
```

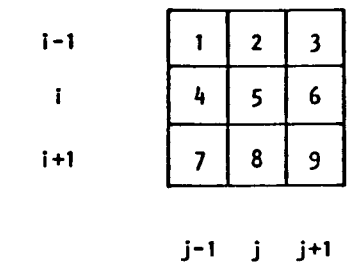

Fig. 4.1 3-by-3 subimage.

Subalg. B: Find the PMAX, PMIN and β for pixel (i,j), given the needed TMAX and TMIN values.

```
if TMAX(i,j) < TMAX(i+1,j) /*first look down*/
   then   TMAX(i,j) ← TMAX(i+1,j)
if TMAX(i,j) < TMAX(i-1,j) /*then look up*/
   then   PMAX(i,j) ← TMAX(i-1,j)
   else   PMAX(i,j) ← TMAX(i,j)
if TMIN(i,j) > TMIN(i+1,j) /*first look down*/
   then   TMIN(i,j) ← TMIN(i+1,j)
if TMIN(i,j) > TMIN(i-1,j) /*then look up*/
   then   PMIN(i,j) ← TMIN(i-1,j)
   else   PMIN(i,j) ← TMIN(i,j)
β(i,j) ← PMAX(i,j) - PMIN(i,j) /*compute β*/
```

This algorithm can be characterized in terms of the factors discussed previously. It is an information extraction algorithm. It is context-dependent since it uses the gray-level values of pixel neighborhoods. It is a single-pass algorithm. It has a univariate input, since the input consists of gray levels only, and a multivariate output, since the output consists of PMAX, PMIN and β (although the ultimate classification will be univariate). The processing is assumed to be performed in non-real time. It has a low computational complexity.

4.2. Potential for Exploiting Parallelism

Identical operations are performed on a large number of pixels, suggesting that an SIMD architecture could be exploited. Since different parameters are being computed for each pixel, one might think that MSIMD would be helpful. However, the results of Subalg. A are used for the computation of both PMAX and PMIN. Thus, there is an overlap which can be taken advantage of with SIMD. Also, β depends on PMAX and PMIN; thus, it cannot be evaluated in parallel with them. Furthermore, there is no benefit from pipelining because there is no logical depth to the computations.

4.3. Alternative Implementations Compared

Based on the previous discussion, the "implementation considerations" can be summarized as follows. The pixel-per-processor and the subimage-per-processor allocations will be considered. The ratio of the number of processors to the number of pixels will vary from 1/M to 1 to < 1/M (recall that the image size is M-by-M pixels). It is assumed that the whole image is available as a block. The interconnection speed is assumed to be less than the processor speed. The interprocessor communication complexity will be unconstrained (but will be related to proposed networks). The PE memory capacity is assumed much larger than M*M/N and therefore not a limiting factor. Three cases will be considered: N = M PEs, N = M*M PEs, and N < M PEs.

In the following algorithms, PE address masks [17] are used. Each of the n = log N positions of the mask corresponds to a bit position in the addresses of the processors (all logarithms are base 2). Each position of the mask is either a 0, 1, or X ("don't care") and the only PEs that will be active are those whose address matches the mask: 0 matches 0, 1 matches 1, and either 0 or 1 matches X. Parentheses are repetition factors; square brackets denote a mask. For example, [X(n-1)0] activates all even number PEs.

A negative PE address mask [18] is similar to a regular PE address mask, except that it activates all those PEs which do not match the mask. Negative PE address masks are prefixed with a minus sign, e.g., [-0(n)] activates all PEs except for 0.

A logical OR of two (or more) PE address masks is equivalent to taking the union of the sets of PEs activated by the masks. A logical AND is equivalent to the intersection. One way to implement PE address masks is discussed in [10].

Case 1: Assume N = M PEs are available.

(a) If the PEs are arranged in a row of M elements (Fig. 4.2), the image will be processed one row (line) at a time. One column of data from the image will be stored in each PE (values from column j will be stored in PE j-1, $1 \leq j \leq M$). Each PE is able to communicate directly with its two nearest neighbors.

Fig. 4.2. Row of M PEs and interconnections (number indicates column of image stored in PE).

The complete algorithm to find the β values for the whole image is shown in Fig. 4.3. The PEs start at the top row of the image and move down to the Mth row. The pixel values and TMAX, TMIN values of only three rows need to be stored in a PE memory at the same time. When a PE is working on β of row i, it only needs the values of rows i-1, i and i+1. The time it takes to perform the algorithm for the whole image is

[2M transfer + (7M-8) comparison + (13M-14)storing + (M-2) subtraction] time units (rows 1 and M are not classified).

The above equation does not explicitly account for the fact that N = M PEs are being used. What is needed, for comparison purposes, is a global expression for the "Total Price" which accounts for both the time spent and the number of PEs used.

Assume all operations except for transfer require the same number of time units. Let A1 be the "price" of a transfer time unit, A2 the "price" of an operation time unit, and B the "price" of a PE.

```
MASK [-0(n)] AND [-1(n)]
/*Cols. 1 and M not classified */
/*For rows 1, 2 get the row max. and min.*/
for i ← 1 to 2 do
        Subalg. A /*all j in parallel*/
for k ← 2 to M-1 do /*Rows 1 & M not classified.*/
        i ← k+1
        Subalg. A /*all j in parallel*/
        i ← k
        Subalg. B /*all j in parallel*/
```

Fig. 4.3. Computation of PMAX, PMIN and β for the image (row of M PEs, N=M). Subalg. A uses inter-PE transfers.

The expression for the "Total Price" can be written as:

$$TP1a = 2M * A1 + (21M-24) * A2 + M * B.$$

As indicated above, due to the way in which data is allocated in this case, Subalg. A requires inter-PE data transfers. The transfers are of the form PE x to PE x+1, $0 \leq x \leq M-2$, or PE x to PE x-1, $0 < x < M$. Each of these transfers can be realized by a single pass through a number of the SIMD networks discussed in the literature including the generalized cube, indirect binary n-cube, omega, STARAN flip, data manipulator, ADM, and Illiac networks [19,6].

(b) If the M PEs are arranged as a $\sqrt{M}$ -by- $\sqrt{M}$ array, each PE will be assigned a $\sqrt{M}$ -by- $\sqrt{M}$ subpicture that will be stored in its memory. This is shown in Fig. 4.4, where $b = \sqrt{M}$. For the first row of the array of PEs, PE i contains the pixels (j,k) for $1 \leq j \leq \sqrt{M}$ and $i\sqrt{M}+1 \leq k \leq (i+1)\sqrt{M}$. The allocation of pixels is done in a similar manner for the remaining rows. Four interconnections, with the four nearest neighbors, are needed for each PE except for the ones on the edges, which need only three, and those in the four corners which need two.

The complete algorithm for finding the TMAX and TMIN values for the whole image is shown in Fig. 4.5. The row maximum and minimum are computed serially for all the pixels of a PE's subimage, but all subimages are processed in parallel. PMAX, PMIN and β are computed in a similar manner. It will be necessary to disable row 1 of the PEs when working on row 1 of the subimage in each PE, row $\sqrt{M}$ of the PEs when working on row $\sqrt{M}$ of the subimage, column 1 of the PEs when working on column 1 of the subimage, and column $\sqrt{M}$ of the PEs when working on column $\sqrt{M}$ of the subimage.

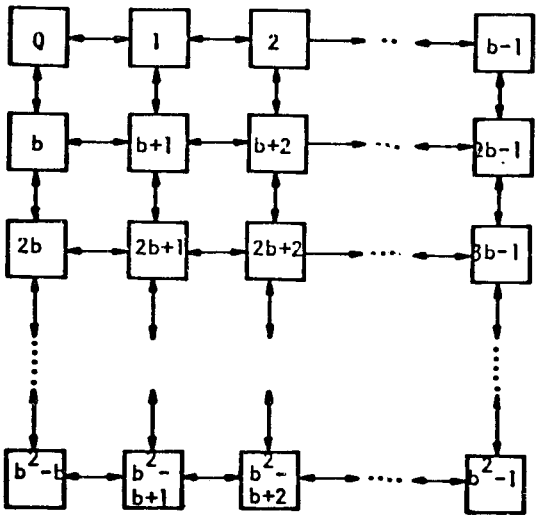

Fig. 4.4. Array of b-by-b PEs with interconnection (numbers indicate PE addresses). Case 1(b), $b = \sqrt{M}$; case 2, b = M; case 3, $b = \sqrt{N}$.

```
/*i and j are the indices for the pixels in each
PE's subimage. For columns j=1 and j=√M within all
PE's subimages the execution of Subalg. A requires
inter-PE data transfers*/
r ← n/2

for i ← 1 to √M do
        j ← 1
        MASK[-X(r)0(r)] /*disable first col. PEs*/
        Subalg. A

        j ← √M
        MASK[-X(r)1(r)] /*disable last col. PEs*/
        /*disable last col. PEs*/
        Subalg. A

        MASK [X(n)] /*enable all PEs*/

        for j ← 2 to √M-1 do
                Subalg. A
```

Fig. 4.5. Computation of TMAX and TMIN for the image ($\sqrt{M}$-by-$\sqrt{M}$ PE array, N=M).

So, in the above overall algorithm, the sequence of operations is as follows. First, each PE scans its assigned subimage evaluating TMAX and TMIN for all of the pixels and storing these values. Then, it scans the TMAX and TMIN values for the subimage, computing PMAX, PMIN and β. Execution of this sequence of operations requires ($6\sqrt{M}$ transfer + 7M comparison + 13M storing + M subtraction) time units. Using the earlier notation, an expression for the "Total Price" of this implementation is:

$$TP1b = 6\sqrt{M} * A1 + 21M * A2 + M * B$$

All of the networks listed for case 1(a) can also provide (in a single pass through the network) the connections needed for case 1(b) as well as cases 2, and 3 below. In general, the time for case 1(b) is less than that for case 1(a) because fewer transfer operations are required.

Case 2. Assume that there are N = M*M PEs arranged in an M-by-M array as shown in Fig. 4.4 (b = M). One pixel will be stored in each PE and the interconnections will be the same as in case 1 (b). Each PE will work on one pixel and get all the needed values for it (PMAX, PMIN, β), as shown in the algorithm of Fig. 4.6. The overall time to execute this algorithm is (6 transfer + 7 comparison + 13 storing + 1 subtraction) time units. The "Total Price" of this implementation is:

$$TP2 = 6 * A1 + 21 * A2 + M^2 * B$$

Comparing TP1a, TP1b and TP2, the following can be concluded. If the "price" of a PE, B, is negligible as compared to A1 and A2 (B << A1 and A2), which means that almost as many PEs as needed can be used but the execution time is very costly, then case 2 is better than case 1. If A1 and A2 are negligible as compared to B (A1 and A2 << B), mean-

```
r ← n/2
MASK[-X(r)0(r)] AND [-X(r)1(r)]
Subalg. A /*all pixels in parallel except cols. 1 &
     M*/

MASK[-0(r)X(r)] AND [-1(r)X(r)]
AND [-X(r)0(r)] AND [-X(r)1(r)]
Subalg. B /*all pixels in parallel except rows 1
     and M, cols. 1 & M*/
```

Fig. 4.6. Computation of PMAX, PMIN and β for the image (M-by-M PE array, N = M*M). Subalg. A uses inter-PE transfers for left and right neighbors. Subalg. B uses inter-PE transfer for above and below neighbors.

ing that the PEs are very costly compared to the execution time which is relatively less important, then case 1 is better. If none of A1, A2 and B is negligible, then the result of the comparison depends on the relative values of these "prices."

Case 3: In this final case, it is assumed that there are $N < M$ PEs available. The image will be divided into N subimages, each of size $M/\sqrt{N}$-by-$M/\sqrt{N}$ (assuming $N = a*a$ and M to be a multiple of a). The PEs are arranged as shown in Fig. 4.4 (b = $M/\sqrt{N}$).

An $M/\sqrt{N}$-by-$M/\sqrt{N}$ subimage will be stored in each PE. The interconnections are as shown in Fig. 4.4 (same as in case 1(b)). The algorithm and sequence of operations for this implementation are the same as in case 1(b), except for replacing $\sqrt{M}$ by $M/\sqrt{N}$.

Thus, the time it takes to do all the needed computations is (6 $M/\sqrt{N}$ transfer + 7 M^2/N comparison + 13 M^2/N storing + M^2/N subtraction) time units. The "Total Price" of this implementation is:

$$(6M/\sqrt{N}) * A1 + (21M^2/N) * A2 + N * B$$

Comparing total prices, the following can be concluded. If B is negligible compared to A1 and A2, then cases 1 and 2 are better than case 3. If A1 and A2 are negligible as compared to B, then case 3 is better than the previous ones. If none of A1, A2 and B is negligible, then the result of the comparison depends on the relative values of these "prices."

Deciding which is the best implementation is dependent on many factors.

Case 1(a) has the simplest interconnection requirements, with, in general, two interconnections per PE whereas the other cases all have slightly more complex interconnection schemes requiring four interconnections per element. Case 1(a) also has the simplest transfer logic, but it requires more transfers than all the other implementations. Case 2 is the fastest, but also requires the largest number of PEs. Case 3 uses the smallest number of PEs, but also is the slowest. Cases 1(a) and (b) have the same number of PEs, but 1(b) is slightly faster since it requires fewer transfers. Also, if the image is available in a raster format and the processing is input/output limited, the "row at a time" implementation would be better (case 1(a)). If the image is available in a frame format, then the "subimage per PE" implementation (cases 1(b), 2, 3) would be faster since it requires fewer transfers.

5. Conclusions

This discussion has illustrated the fact that effective utilization of multimicroprocessor systems for implementing image pattern recognition algorithms can involve a significant number of tradeoffs. Factors to be considered include the characteristics of the application, the specific image processing algorithms to be implemented, and the nature of the hardware to be used. Whereas past efforts often began with specific multiprocessor systems and sought optimal implementations of given image processing tasks, the present design problem has many more "degrees of freedom," involving determination of the optimal processing system architecture as well as the details of the implementation. The system architecture is restricted only to be from a relatively broad class of computer architectures, SIMD, MSIMD and pipeline.

Some candidate lists of algorithm characteristics and implementation considerations have been suggested. The goal ultimately is to develop useful correspondences between elements of the different lists, a mapping, as it were, from one to the other which will facilitate the design of optimal multimicroprocessor systems. To provide some insight into this process, alternative multiprocessor implementations of a simple pattern recognition algorithm have been formulated and compared. The results presented are of a rather specific and limited nature. The ongoing research has begun to yield more general relationships but has also pointed to a pressing need for new criteria for evaluating and comparing the cost-effectiveness of multiprocessor systems.

Acknowledgements: The authors wish to thank L. J. Siegel, P. T. Mueller, Jr., and A. E. Feather for many useful discussions.

References

1. G. Barnes et al., "The Illiac IV Computer," IEEE Trans. Comp., Vol. C-17, Aug. 1968, pp. 746-757.
2. J. M. Vocar and R. O. Faiss, "Warp Processing Using STARAN," 1977 Conf. Picture Data Description and Management, Apr. 1977, pp. 68-75.
3. B. Kruse, "A Parallel Picture Processing Machine," IEEE Trans. Comp., Vol. C-22, Dec. 1973, pp. 1075-1087.
4. A. P. Reeves and R. Rindfuss, "The BASE 8 Binary Array Processor," 1979 IEEE Comp. Soc. Conf. Pattern Recog. and Image Processing, Aug. 1979, pp. 250-255.

5. M. J. Flynn, "Very High-Speed Computing Systems," Proc. IEEE, Vol. 54, Dec. 1966, pp. 1901-1909.
6. H. J. Siegel, "Interconnection Networks for SIMD Machines," Computer, Vol. 12, June 1979, pp. 57-65.
7. J. Bogdanowicz and H. J. Siegel, "A Partitionable Multimicroprogrammable-Microprocessor System for Image Processing," 1978 IEEE Computer Society Workshop on Pattern Recognition and Artificial Intelligence, Apr. 1978, pp. 141-144.
8. G. J. Lipovski and A. Tripathi, "A Reconfigurable Varistructure Array Processor," 1977 Intl. Conf. Parallel Processing, Aug. 1977, pp. 165-174.
9. G. J. Nutt, "Microprocessor Implementation of a Parallel Processor," 4th Symp. Comp. Architecture, Mar. 1977, pp. 147-152.
10. H. J. Siegel, P. T. Mueller, Jr., and H. E. Smalley, Jr., "Control of the Partitionable Multimicroprocessor System," 1978 Int'l. Conf. Parallel Processing, Aug. 1978, pp. 9-17.
11. 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," IEEE Conf. Pattern Recognition and Image processing, Aug. 1979, pp. 214-224.
12. F. Briggs, K. S. Fu, K. Hwang, and J. Patel, "PM4-A Reconfigurable Multimicroprocessor System for Pattern Recognition and Image Processing," Natl. Comp. Conf., June 1979, pp. 225-265.
13. P. H. Swain and S. M. Davis, eds., Remote Sensing: The Quantitative Approach, McGraw-Hill, New York, 1978.
14. T. C. Chen, "Overlap and Pipeline Processing," in Introduction to Computer Architecture, H. S. Stone, ed., Science Research Associates, Inc., Chicago, 1975, pp. 375-431.
15. L. J. Siegel, P. T. Mueller, Jr., and H. J. Siegel, "FFT Algorithms for SIMD Machines," 17th Allerton Conf. on Communication, Control and Computing, Oct. 1979, pp. 1006-1015.
16. M. Faintich, "Feature Identification Using Statistical Signatures of the Regional Mesostructure," Defense Mapping Agency Aerospace Center/IGN Technical Project Report, 1977.
17. H. J. Siegel, "Analysis Techniques for SIMD Machine Interconnection Networks and the Effects of Processor Address Masks," IEEE Trans. Comp., Vol. C-26, Feb. 1977, pp. 153-161.
18. H. J. Siegel, "Controlling the Active/Inactive Status of SIMD Machine Processors," 1977 Intl. Conf. Parallel Processing, Aug. 1977, p. 183.
19. H. J. Siegel and S. D. Smith, "Study of Multistage SIMD Interconnection Networks," 5th Symp. Comp. Architecture, Apr. 1978, pp. 223-229.