Parallel Processing Approaches to Image Correlation

LEAH J. SIEGEL, MEMBER, IEEE, HOWARD JAY SIEGEL, MEMBER, IEEE, AND ARTHUR E. FEATHER, STUDENT MEMBER, IEEE

***Abstract*—Image correlation is representative of a wide variety of window-based image processing tasks. The way in which multimicroprocessor systems (e.g., PASM) can use SIMD parallelism to perform image correlation is examined. Two fundamental algorithm strategies are explored. In one approach, all of the data that will be needed in a processor are transferred to the processor and operated on there. In the other, each processor performs all possible operations on its local data, generating partial results which are then transferred to the processor in which they are needed. The "time/space/interprocessor-transfer" complexities of the two algorithm approaches are analyzed in order to quantify the differences resulting from the two strategies. For both approaches, the asymptotic time complexity of the N-processor SIMD algorithms is $(1/N)$th that of the corresponding serial algorithms.**

***Index Terms*—Algorithms, convolution, image correlation, image processing, multimicroprocessor systems, parallel processing, parallel programming, PASM, pattern recognition, SIMD machines.**

I. INTRODUCTION

IMAGE correlation is a widely used procedure in many areas of image and picture processing. This process, also known as template matching, is used to locate an object in a picture [5], [14] or, in image registration, to match pieces of two pictures to one another [13]. It is used in some forms of edge detection to find the step edge between two areas, or to find lines, spots, or curves [14]. In digital photogrammetry image correlation is used to find the corresponding points of two images of a stereomodel [14]. In this application, image sizes are typically at least 4096×4096 with match areas on the order of 64×64.

Because image correlation requires comparing portions of two images in a large number of relative positions, it is an extremely time consuming process. The time required to complete these calculations can be reduced by exploiting the parallelism inherent in the task. The way in which multimicroprocessor systems (e.g., PASM [18]) can use "SIMD" parallelism to perform this task is examined here.

The SIMD (single instruction stream–multiple data stream) [9], [23] machine model used here consists of a control unit, interconnection network, and N PE's (processing elements), where each PE is a processor-memory pair [17]. This is shown in Fig. 1. In an SIMD machine of size $N = 2^n$, the PE's are addressed (numbered) from 0 to $N - 1$. In proposed systems, N is as large as 2^{10} [18] to 2^{14} [12]. The control unit broadcasts an instruction to all PE's, and all active (enabled) processors simultaneously execute the same instruction, each on data in its own memory. The interconnection network provides inter-PE communication. SIMD parallelism has been shown to yield significant reductions in computation time for image and speech processing tasks (e.g., [10], [15], [20]). Here, window-based image processing tasks are considered.

Manuscript received April 6, 1981; revised August 14, 1981. This work was supported by the U.S. Air Force Office of Scientific Research, Air Force Systems Command, U.S. Air Force under Grant AFOSR-78-3581 and the Defense Mapping Agency, monitored by the U.S. Air Force Rome Air Development Center Information Sciences Division under Contract F30602-78-C-0025 through the University of Michigan.

L. J. Siegel and H. J. Siegel are with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907.

A. E. Feather was with the School of Electrical Engineering, Purdue University, West Lafayette, IN 47907. He is now with the Bellaire Research Center, Shell Development, Houston, TX 77001.

In the complexity analyses that follow, it is assumed that each required parallel inter-PE data move can be done in one transfer step. This will be true if the interconnection network used is a multistage network such as: 1) one employing the generalized cube topology with individual box control [19] (e.g., omega [11], n-cube [12]), 2) the data manipulator network [7], or 3) the augmented data manipulator [19]. This is because each required transfer is either a type of exchange (cube connection [16]) or a "uniform shift" (i.e., from PE i to PE $i + k \bmod N$, $0 \leq i < N$, k fixed).

Only those SIMD machine features needed for the algorithms that follow have been described. The model is intended to provide a general framework in which SIMD algorithms can be developed. In Section VI the performance of the algorithms using an alternative model will be discussed.

The objectives of this study are as follows.

1) To demonstrate the applicability of the SIMD mode of parallelism to a class of image processing tasks. The operations performed in image correlation are representative of the types of data manipulations needed for a wide variety of window-based image processing tasks.

2) To explore two fundamental parallel algorithm strategies. In one approach all of the data that will be needed by a PE are transferred to the PE and processed there. In the other, each PE performs all possible operations on its local data, generating partial results which are then transferred to the PE in which they are needed.

3) To analyze and compare the computational requirements of the alternative algorithms. In serial algorithms, there is often a tradeoff between computation time and space. In parallel algorithms, the tradeoff may be a function of three parameters: computation time, space, and inter-PE communications.

In the next section image correlation is defined. In the subsequent sections parallel algorithms for image correlation are presented and analyzed.

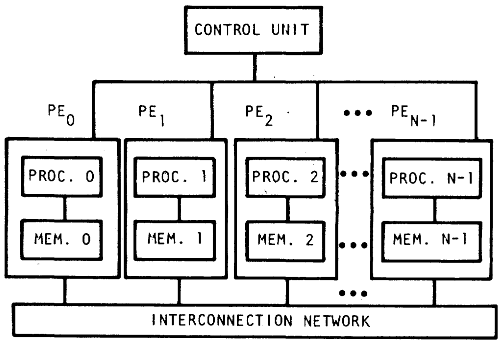


Fig. 1. Model of an SIMD machine.

II. Image Correlation

A. Definition and Serial Algorithms

An image is represented by a two-dimensional array where each element ("pixel") has an unsigned integer value representing the "gray level" of the pixel. Image correlation involves determining the position at which a relatively small match area best matches a portion of an input image. Correlation measures are used to measure the degree of similarity or disagreement between the match area and an equivalent size area on the input image. Let the symbols x and y denote single elements of arrays X and Y, where X is the match image and Y is an area of the input image which has the same dimensions as X. Let M be the number of elements in the match area X. Two representative correlation measures are

$$SXY = \sum xy - \sum x \sum y/M$$
$$RXY = SXY/(SXX * SYY)^{(1/2)}.$$

Correlation measure SXY is the covariance of the match area with a portion of the input area. Large positive values indicate similarity, while large negative values indicate similarity between a positive and a negative image. Values near zero indicate little or no similarity. Correlation measure RXY is the linear correlation coefficient of statistics. This measure is a normalized version of SXY, with values ranging between $+1$ and -1. A value of $+1$ indicates exact similarity, while values near zero indicate little similarity. In general, a correlation value will be computed for every possible position where the match area will fit on the input image. The match position where the correlation measure is maximized corresponds to the best placement of the match area on the image.

The computation time for image correlation is dominated by the time to compute the Σxy, Σy, and (for measure RXY) the Σy^2 values for all possible match positions. The Σx and Σx^2 values involve only the match area elements, and need to be computed (or precomputed) only once.

The way in which data elements are combined to obtain the Σxy values is similar to operations performed in a variety of important image processing tasks, including convolution and filtering. For an input image having R rows and C columns and a match area having r rows and c columns, there are $(R - r + 1)(C - c + 1)$ match positions. Serial computation of the Σxy terms over the entire image, performed by simply sliding the match area over the image and calculating the value of Σxy for each overlap position, requires $(R - r + 1)(R - r + 1)rc$ multiplications and $(C - c + 1)(C - c + 1)(rc - 1)$ additions.

In computing the Σxy values, each match position generates a new set of terms to be summed. No terms from one match position can be reused in a different match position. In computing the Σy and Σy^2 values, two (or more) input image elements summed for one match position may also be summed for another match position. The algorithms considered for calculating the Σy and Σy^2 values therefore attempt to avoid "redundant" operations, e.g., performing a sum for one match position which has already been performed for another. The operations performed in computing the Σy and Σy^2 values, i.e., the summing of elements under a window where the window moves over an image, are typical of operations required for a variety of image processing tasks. These include image smoothing, edge enhancement, and convolution using a rectangular window.

Consider the following serial (uniprocessor) algorithm for computing the Σy's, i.e., summing the pixel values in each match area. This algorithm will be used as a basis for parallel algorithms.

Assume that for input image I, the position of the match area is defined by the coordinates of the input image pixel covered by the upper left corner of the match area. Let "colsum" be a vector of length C, where

$$\text{colsum}\ (j) = \sum_{i=k}^{k+r-1} I(i, j)$$

where k is the row coordinate of the current position of the match area, and $0 \leq j < C$. Let SUM be an $R - r + 1$ by $C - c + 1$ array, where SUM (i, j) is the sum of pixels of I for the match area position (i, j), $0 \leq i < R - r + 1$, $0 \leq j < C - c + 1$.

The algorithm is shown in Fig. 2. First, colsum is initialized for row 0 of the image. The colsum values for columns 0 to $c - 1$ are summed to compute SUM $(0, 0)$. SUM $(0, j)$ for $1 \leq j < C - c + 1$ is computed from SUM $(0, j - 1)$ by subtracting colsum $(j - 1)$ and adding colsum $(j + c - 1)$. A similar strategy is used to compute SUM (i, j) for $1 \leq i < C - c + 1$ and $1 \leq j < R - r + 1$. To do this, each colsum (j) is first updated by subtracting $I(i - 1, j)$ and adding $I(i + r - 1, j)$, $0 \leq j < R - r + 1$.

The complexity of this serial algorithm in terms of additions is

$$4RC - Rc - 3Cr + rc + 5C + 3R - 2c - 3r + 4.$$

(For simplicity, the additions required for loop counting and indexing have not been included. In the SIMD algorithms these would be performed in the control unit, and could be overlapped with the PE operations.) This algorithm moves the match area along the rows of the input image. Depending on C, R, c, and r the algorithm complexity may be less by moving along columns.

Computation of the Σy^2's is similar. In this case the y^2 values subtracted from the colsum's in the update process (see Fig. 2) must be saved when they are first calculated. This increases the space required for the algorithm by rC. The arithmetic complexity is increased by RC multiplications.

If the Σxy, Σy, and Σy^2 values for a given match position are computed together, the correlation measure for that match

```
/* initialize values of colsum */
for j = 0 to C-1 do
   colsum(j) = I(0,j)
   for i = 1 to r-1 do
      colsum(j) = colsum(j) + I(i,j)
/* compute SUM(0,j) for 0 ≤ j < C-(c-1) */
SUM(0,0) = colsum(0)
for j = 1 to c-1 do
   SUM(0,0) = SUM(0,0) + colsum(j)
for j = 1 to C-(c-1) do
   SUM(0,j) = SUM(0,j-1) - colsum(j-1) + colsum(j+c-1)
/* compute SUM(i,j) for 1 ≤ i < R-(r-1) and 0 ≤ j < C-(c-1)) * /
for i = 1 to R-(r-1) do
   /* compute SUM(i,0) and update associated colsum values */
   for j = 0 to c-1 do
      colsum(j) = colsum(j) - I(i-1,j) + I(i+r-1,j)
   SUM(i,0) = colsum(0)
   for j = 1 to c-1 do
      SUM(i,0) = SUM(i,0) + colsum(j)
   /* compute SUM(i,j) and update associated colsum values for 1 ≤ j < C(c-1)) */
   for j = 1 to C-(c-1) do
      colsum(j+c-1) = colsum(j+c-1) - I(i-1,j+c-1) + I(i+r-1,j+c-1)
      SUM(i,j) = SUM(i,j-1) - colsum(j-1) + colsum(j+c-1)
```

Fig. 2. Serial algorithm to compute Σy terms.

position can be calculated, and is saved only if it is the current maximum over the correlation measure values computed so far. Thus, the Σxy, Σy, and Σy^2 values for each position do not have to be saved.

B. Parallel Image Correlation

In Section III a parallel algorithm for computing the Σx and Σx^2 values is given. In Sections IV and V, parallel algorithms for the Σxy, Σy, and Σy^2 computations are presented. For the Σxy, Σy, and Σy^2 operations, two algorithm strategies are explored. For both, the input image data will be divided among the PE's, and each PE will compute the values of the correlation measure for a portion of the input image. In the first, "complete sums" approach, all of the data which will be needed for the computations performed in a given PE are transferred into that PE. All subsequent operations can then be performed locally, so that each PE computes the "complete sums" for a set of match positions. In the second, "partial sums" approach, each PE performs as much of the computations as possible using its own data, then transfers partial results to the PE in which they are needed.

In order to distribute the input image, the N PE's of the system are logically configured as an $N_R \times N_C$ rectangular grid, on which the $R \times C$ image is superimposed. Thus, with the possible exception of the rightmost column and bottommost row of PE's, each PE holds an $R' \times C'$ subimage, where $R' = \lceil R/N_R \rceil$ and $C' = \lceil C/N_C \rceil$. This is shown in Fig. 3. The values for N_R and N_C will be chosen to minimize execution time of the algorithms, and will be discussed in Section IV-A.

(An alternative to these approaches is to assume that the SIMD machine has the capability to load the image data into several PE's simultaneously. With this capacity, an element of the input array which is needed in several PE's could be loaded into the appropriate PE's (with little or no cost) simultaneously. This would eliminate the need for inter-PE transfers. However, the memory management necessary to place each image point in the appropriate location in each PE may be significantly more complex than the memory management needed to load the PE memories with disjoint subimages. This approach will require an "intelligent" memory management system and more storage in each PE, and will not be considered here.)

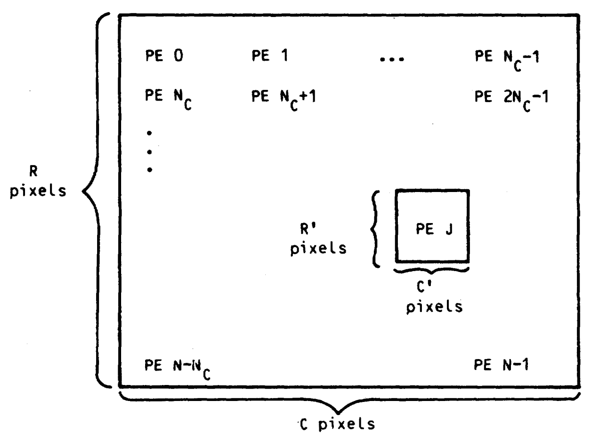


Fig. 3. Data assignment of $R \times C$ image to N PE's.

In the algorithms to compute the Σxy, Σy, and Σy^2 values, it will initially be assumed that the results calculated (i.e., the Σxy, Σy, and Σy^2 values) are saved. For the calculation of RXY and SXY, this will not be necessary, as will be described in Sections IV-C and V-C. However, so that each of the Σxy, Σy, and Σy^2 algorithms can be applied to other related computations, in the presentations it will be assumed that the results for the whole image are to be stored.

III. Σx AND Σx^2 COMPUTATION

The Σx and Σx^2 values may be precomputed and stored with the match area, or computed in a straightforward manner in parallel before calculating the Σxy, Σy, and Σy^2 values. Simply assign to each PE M/N of the match area pixels. Each PE first computes x^2 for all the elements it holds. It then sums its x values and sums its x^2 values. All of these local Σx and local Σx^2 sums are then combined using a recursive doubling approach [22] (see Fig. 4). Each even numbered PE J sends its local Σx result to PE $J + 1$. Simultaneously, each odd numbered PE $J + 1$ sends its local Σx^2 to PE J. The odd numbered PE's add the received data to their local Σx and then compute the whole Σx using recursive doubling, with the result saved in each odd numbered PE. Similarly, the even numbered PE's compute Σx^2. These two recursive doublings can occur simultaneously. The odd and even PE's then exchange results, so that each PE contains both Σx and Σx^2. This requires M/N multiplications, $n + (2M/N) - 2$ additions, and $n + 1$ inter-PE data transfers. (A serial algorithm will require M multiplications and $2M - 2$ additions.) Each PE will store Σx and Σx^2 for later use.

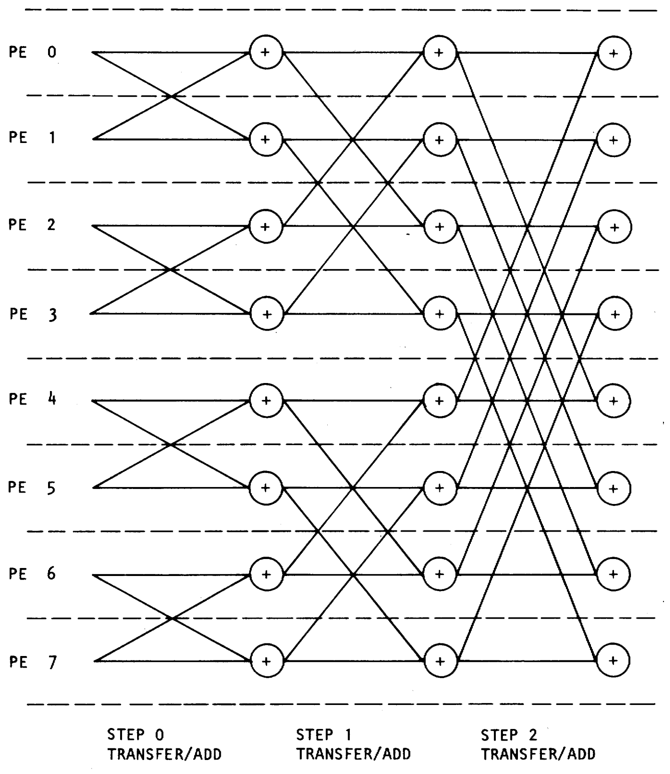


Fig. 4. Recursive doubling for an $N = 8$ PE system. In general, a recursive doubling algorithm will consist of $n = \log_2 N$ transfer/add steps. At step i, $0 \le i < n$, exchanges are performed between PE's j and $j + 2^i$, where the ith bit of j is 0.

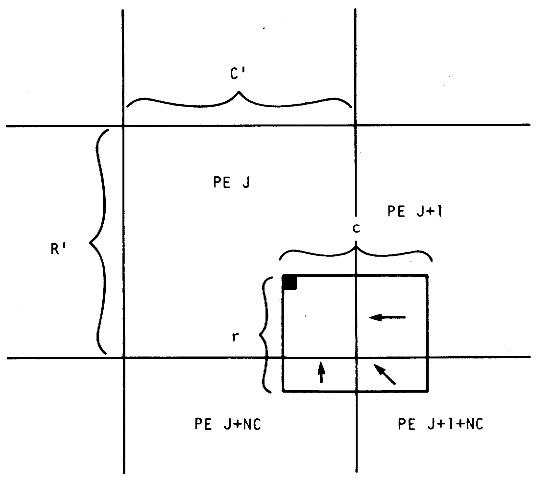


Fig. 5. Example of overlap position requiring data transfers. The shaded pixel represents the "beginning" of the overlap position. The arrows indicate the directions of the data transfers. (Proportions of match area to a PE's subimage are not necessarily to scale.)

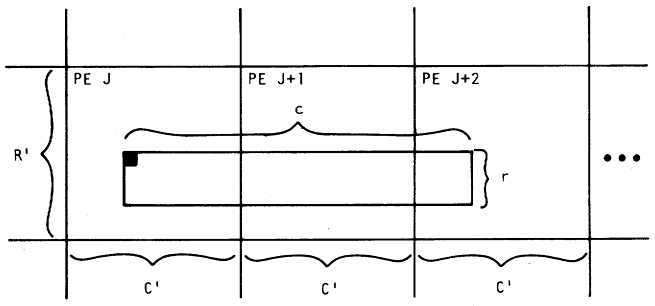


Fig. 6. Example of overlap position which requires PE J to receive data from PE $J + 1$ and PE $J + 2$.

IV. Complete Sums Approach

A. Σxy Computation

In the complete sums approach each PE will compute the correlation measure for overlap positions which "begin" in the PE (i.e., for which the upper left corner of the match area overlaps a point of the PE's subimage). Each PE will therefore compute the correlation measure for $R'C'$ match positions. The computations will be performed simultaneously in all PE's. For match positions where the portion of the input image is not fully contained in a single PE (Fig. 5), the needed points will be transferred before the computations are performed. Such transfers will occur simultaneously for all PE's, so that at the same time that a pixel is being transferred, for example, from PE $J + 1$ to PE J, the corresponding pixel is being transferred from PE $J + 2$ to PE $J + 1$, from PE $J + 3$ to PE $J + 2$, and so on.

Depending on the size relationships between r and R' or c and C', the transferred elements may come from PE's adjacent to PE J, or from several levels of adjacent PE's. If, for example, the match area dimension in one direction is large in comparison to the dimension of the portion of the input area stored in each PE, the matches will extend over several PE areas in that direction. This is shown for the case where $c > C' + 1$ and $r \le R' + 1$ in Fig. 6. PE J will transfer some y values a distance greater than one, and will receive some y values from a distance greater than one. Without loss of generality, in the subsequent discussions, it will be assumed that elements are needed only from adjacent PE's.

When computing the Σxy values, all PE's will use the same match area element simultaneously, so that element can be broadcast to all PE's from the control unit. Alternatively, if PE memory space is available, the match area, which is typically small, can be held in each PE's memory. The time to perform the broadcast from the control unit versus the memory fetch from the PE memory will be implementation dependent. In the space analyses that follow, it will be assumed that the match area values are broadcast from the control unit.

Computation of all of the Σxy terms will be accomplished in the time required to compute the Σxy terms for the $R' \times C'$ subimage held in a single PE. These times are summarized in the first column of Table I. Storage will be required for the PE's portion of the input image ($R'C'$ elements), for the computed Σxy values ($R'C'$ elements), and for the input image elements transferred to the PE in order to provide all of the data needed for the PE's match positions. The number of transferred elements is $(c - 1)R' + (r - 1)C' + (r - 1)(c - 1)$; however, it is not necessary to store all of these values at the same time. Consider the extra storage needed for nonlocal y values by a typical ("nonedge") PE J. The analysis is divided into two cases. It will show that at any point in time at most $(2c - 2)(r - 1) + 1$ locations are required.

First, consider when the match area (upper left corner) is positioned in row i, $0 \le i \le R' - r$ (see Fig. 7). $(c - 1)r$ locations are required for nonlocal y data, for the y data for col-

TABLE I
COMPLEXITY OF COMPLETE SUMS AND PARTIAL SUMS ALGORITHMS FOR COMPUTING Σxy TERMS

	Complete Sums	Partial Sums
# mult steps	R'C'rc	R'C'rc
# add steps	R'C'(rc-1)	R'C'(rc-1)
# transfer steps	R'(c-1)+C'(r-1) +(r-1)(c-1)	R'(c-1)+C'(r-1) +(r-1)(c-1)
space	2R'C'+(2c-2)(r-1)+1	2R'C'

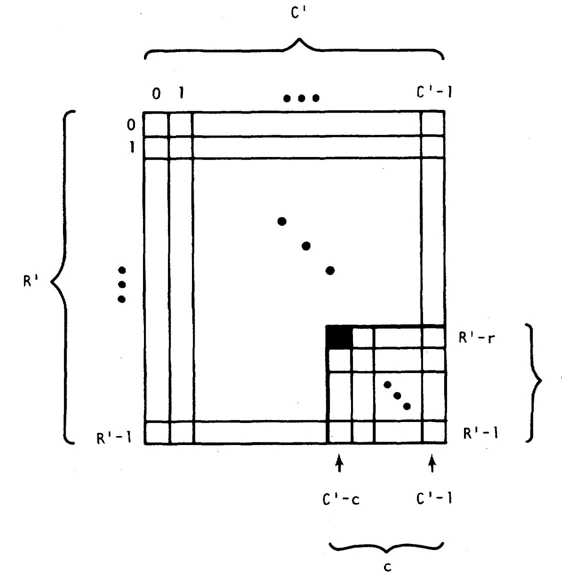


Fig. 7. Indexing in a PE's subimage.

umns 0 to $c-2$ of rows i to $r+i-1$ of PE $J+1$'s subimage. The Σxy values can be calculated by moving the match area from position (0, 0) to (0, 1) to $\cdots$ (0, $C'-1$), then from (1, 0) to (1, 1) to $\cdots$ (1, $C'-1$), $\cdots$ and finally from ($R'-r$, 0) to ($R'-r$, 1), to $\cdots$ ($R'-r$, $C'-1$).

Next consider when the match area is positioned in row i, $R'-r<i<R'$ (see Fig. 7). In this case, at most $(2c-2)(r-1)+1$ locations are required for nonlocal y data. For these match positions the match area will move along columns instead of rows, from $(R'-r+1, C'-1)$ to $(R'-r+2, C'-1)$ to $\cdots$ $(R'-1, C'-1)$, then from $(R'-r+1, C'-2)$ to $(R'-r+2, C'-2)$ to $\cdots$ $(R'-1, C'-2)$, $\cdots$ and finally, from $(R'-r+1, 0)$ to $(R'-r+2, 0)$ to $\cdots$ $(R'-1, 0)$. For match positions (i, j) where $R'-r<i<R'$, and j is fixed at a value in the range $0 \leq j \leq C'-c$, the nonlocal y data needed are rows 0 to $r-(R'-i)-1$ of columns j to $j+c-1$ of PE $J+N_C$'s subimage. For match positions (i, j), where $R'-r<i<R'$, and j is fixed at a value in the range $C'-c<j<C'$, the nonlocal y data needed are rows i to $R'-1$ of columns 0 to $c-(C'-j)-1$ of PE $J+1$'s subimage, rows 0 to $r-(R'-i)-1$ of columns j to $C'-1$ of PE $J+N_C$'s subimage, and rows 0 to $r-(R'-i)-1$ of columns 0 to $c-(C'-j)-1$ of PE $J+N_C+1$'s subimage. The maximum nonlocal y storage needed for this range of i and j is $(2c-2)(r-1)+1$.

For given c, r, C, R, and N, the number of arithmetic operations required for the algorithm is minimized by minimizing the subimage area $\alpha = R'C'$. By choosing $\alpha = RC/N$, i.e., by dividing the input equally among the PE's, this minimum is attained. The number of transfer steps will be minimized by the values of C' and R' for which the expression

$$(r-1)C' + (c-1)R'$$

is minimized. Minimizing with respect to R' gives

$$R' = ((r-1) * \alpha/(c-1))^{(1/2)}$$

subject to the constraints that R' and α/R' be integers. It will follow that

$$C' = ((c-1) * \alpha/(r-1))^{(1/2)}.$$

In the special case where $c = r$, the image should be distributed such that

$$C' = R' = \alpha^{(1/2)}$$

that is, each PE should contain a square subimage.

B. Σy and Σy^2 Computation

The complete sums algorithms to compute Σy and Σy^2 values will be based on the serial Σy and Σy^2 algorithms, with each PE operating on an $(R'+r-1)(C'+c-1)$ subimage.

Consider computing Σy and Σy^2 in a typical ("nonedge") PE J. A total of $(r-1)C' + (c-1)R' + rc - 1$ y values must be transferred into the PE from adjacent PE's, as discussed in the previous subsection. The transfers are as shown in Fig. 5. However, it is not necessary to store all of these if a data item is transferred only when it is first needed. This is explained below in two cases. It will be shown that at most $(c-1)r$ locations will be required at any point in time.

When the match area is positioned in row i of the PE's subimage, $0 \leq i \leq R'-r$, $(c-1)r$ storage locations are required for nonlocal y data, for the y data for columns 0 to $c-2$ of rows i to $r+i-1$ of PE $J+1$'s subimage.

When the match area is positioned in row i of the PE's subimage, $R'-r<i<R'$, at most $(c-1)(r-2)$ locations are required for nonlocal y data. Most y data can be incorporated into the current Σy being computed and the appropriate "colsum" vector location when it is transferred into a PE. The only y data that needs to be saved is that which will be needed for later "colsum" updates. Specifically, this is rows $R'-r+1$ to $R'-2$ of columns 0 to $c-2$ of PE $J+1$.

Using these data storage strategies, the Σy and Σy^2 values for each match position can be calculated as described in the serial algorithms (for a $(C'+c-1)(R'+r-1)$ image). The complexities for the Σy and Σy^2 computations are given in column one of Tables II and III, respectively.

C. RXY and SXY Computation

To compute RXY (or SXY) the previously described operations are interleaved so that the Σxy, Σy, Σy^2, and RXY (SXY) values for one match position are computed before the match area is moved to a new position. The maximum RXY (SXY) value and its match position coordinates are saved. The computation of RXY is described; the SXY computation is a subset of those operations.

Consider the computation performed in a typical ("nonedge") PE J. In order to combine the algorithms of Sections IV-A and IV-B, the Σxy algorithm must be slightly modified. The match area will move over the image in the way that was described in the Σy algorithm, that is, from position (0, 0) to (0, 1) to $\cdots$ (0, $C'-1$), then from (1, 0) to (1, 1) to $\cdots$ (1, C'

TABLE II
COMPLEXITY OF COMPLETE SUMS AND PARTIAL SUMS ALGORITHMS FOR COMPUTING Σy TERMS

	Complete Sums	Partial Sums
# add steps	4R'C'+3R'c+C'r -R'+C'+rc -r	4R'C'+R'c+3C'r -3R'-5C'+rc -3r-c+3
# transfer steps	R'(c-1)+C'(r-1) +(r-1)(c-1)	R'(c-1)+C'(r-1) +(r-1)(c-1)
space	2R'C'+(C'+c-1) +(c-1)r	2R'C'+C'

TABLE III
COMPLEXITY OF COMPLETE SUMS AND PARTIAL SUMS ALGORITHMS FOR COMPUTING Σy^2 TERMS

	Complete Sums	Partial Sums
# mult steps	(R'+r-1)(C'+c-1)	R'C'
# add steps	4R'C'+3R'c+C'r -R'+C'+rc -r	4R'C'+R'c+3C'r -3R'-5C'+rc -3r-c+3
# transfer steps	R'(c-1)+C'(r-1) +(r-1)(c-1)	R'(c-1)+C'(r-1) +(r-1)(c-1)
space	2R'C'+(C'+c-1)(r+1)	2R'C'+C'(r+1)

$-1), \cdots$ and finally, from $(R'-1, 0)$ to $(R'-1, 1)$ to $\cdots (R'-1, C'-1)$. The worst case for space is for $0 \le i \le R'-r$, when $r(C'+c-1)$ space is needed for y^2 values and $r(c-1)$ for y values (plus "colsums" and the original image). Less space is needed when $R'-r < i < R$ because space is not needed for nonlocal y^2 values.

Column one of Table IV summarizes the total time, transfers, and space used. The time is a summation of that for computing Σxy, Σy, and Σy^2 for every match position. The transfers are for the nonlocal y data needed. The space is for the PE's own subimage, the nonlocal y storage described above, and the extra storage used for intermediate results in calculating Σy and Σy^2.

Once each PE has found its own maximum *RXY* value, recursive doubling [22] can be used to find the overall maximum and its location. This will require $0(n)$ additional inter-PE transfers.

V. PARTIAL SUMS APPROACH

A. Σxy Computation

The partial sums procedure for computing the Σxy values consists of three steps. The first step is the generation of partial sums by performing all parts of the calculation that can be done using the data within each PE. In the second step the results of the partial sums generation are transferred so that each PE contains all of the partial sums needed to form the Σxy terms. In the last step the final sums are developed within each PE by combining the appropriate partial sums. The details for this procedure follow. It is assumed that the match area elements are either broadcast from the control unit or stored in each PE's memory, as was discussed in Section IV-A.

In the first step of the algorithm, each PE, independently of the others, computes the "partial sums" of match point-image point products that can be computed with its own data. This can be visualized by sliding the match area M over the image area in each PE, as shown in Fig. 8. At each match

TABLE IV
COMPLEXITY OF COMPLETE SUMS AND PARTIAL SUMS ALGORITHMS FOR COMPUTING *RXY*

	Complete Sums	Partial Sums
# mult steps	those for Σxy and Σy^2	those for Σxy and Σy^2
# add steps	those for Σxy, Σy, and Σy^2	those for Σxy, Σy, and Σy^2
# transfer steps	R'(c-1)+C'(r-1) +(r-1)(c-1)	3[R'(c-1)+C'(r-1) +(r-1)(c-1)]
space	R'C'+(C'+c-1)(r+2)+r(c-1)	R'C'+r(3C'+2c-2)

In Addition to the Above, Each Approach Uses 2 Subtractions, 3 Multiplications, 2 Divisions, and 1 Square Root Operation for Each of the $R'C'$ Match Positions in Order to Combine Terms. Both Methods also Require $0(n)$ Additional Transfers for Determining the Maximum *RXY* Value (and Its Coordinates in the Input Image) Over All PE's.

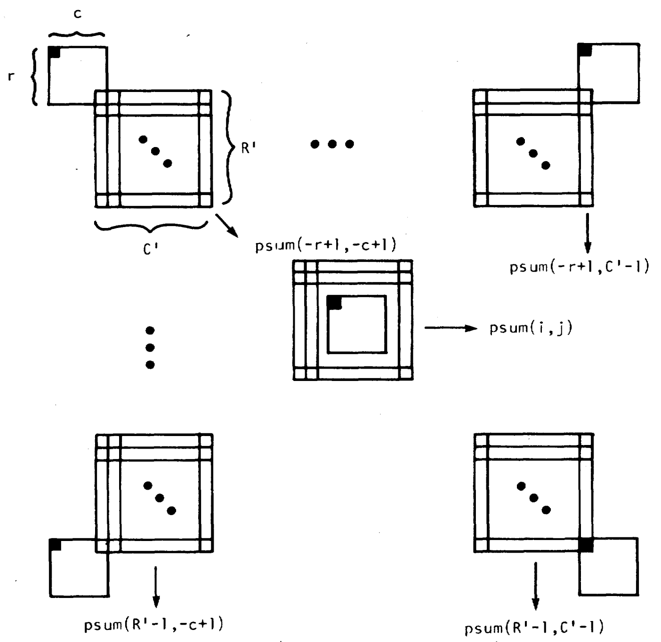


Fig. 8. Overlap positions in the partial sums approach and the terms of the partial sums (psum) array calculated.

area-image area position from Fig. 8, a "partial sum" is generated. For each location where an image point and match point overlap in a given position, the product of the image and the match points is calculated; all the products for that match area-image area position are then summed. The partial sum terms generated by this procedure can be viewed as forming a $(R'+r-1) \times (C'+c-1)$ array called "psum." In Fig. 8 the element of "psum" into which the partial sum is stored is given for each of the example overlap positions. A match position will again be numbered by the input subimage coordinates (i, j) of the upper left corner of the match area. Since the match area may not be contained in the input image area, however, the ranges of i and j differ from those in the serial and complete sums algorithms. If the upper left corner of the input subimage is considered to be position $(0, 0)$, $-r < i < R'$ and $-c < j < C'$. The number of partial sums that must be computed in each PE is $(r+R'-1)(c+C'-1)$. To develop these terms, every element of the match area M will be multiplied by every element of the input area in the PE. Therefore, the number of multiplications required is $rcR'C'$. The number of additions required is equal to the number of multiplications minus the number of terms generated, or $rcR'C' - (r+R'-1)(c+C'-1)$.

Once the partial sums have been computed independently in the PE's, it is necessary to combine the results from other PE's to build the complete sums. Using the criterion that the upper left corner element of the match area must be present in a partial sum for it to remain in a PE, the terms in the rightmost C' columns and bottommost R' rows of the "psum" array are kept in the PE, and are labeled "KEEP" in Fig. 9. Those elements "above" the kept area are transferred to PE's "above" this PE. Similarly, those elements "to the left" of the kept area are transferred to PE's "to the left," and the terms on the upper left are transferred to PE's diagonally above and to the left. As in the complete sums approach, the distance which elements will be transferred depends on the size relationships between r and R' and c and C'. The number of interprocessor transfers which will be required is equal to the total number of "psum" terms generated minus the number of terms kept (which is the number of image area points originally in each PE). Thus, the number of transfers required is $(r + R' - 1)(c + C' - 1) - R'C'$.

In the final step of this method, the partial sums transferred are combined with the partial sums that were kept to yield the final sums Σxy. The number of additions required to complete these calculations is equal to the number of partial sum terms that were transferred.

Rather than implementing the partial sums method as three separate steps, less space is required if the three steps for a given match position are executed in sequence. As soon as a non-"kept" partial sum is computed, it can be transferred to its destination PE and saved in the memory location which will eventually hold the Σxy term of which it is a part. The execution time remains the same, and the only storage that is needed in each PE is two $R'C'$ element arrays, one for the input image and one for the Σxy values. The complexity of the partial sums Σxy algorithm is summarized in column two of Table I.

B. Σy and Σy^2 Computation

The partial sums algorithms for computing the Σy and Σy^2 values are similar in strategy to the partial sums method for computing the Σxy values. Each PE computes Σy or Σy^2 terms for all match positions or portions of match positions for the $R' \times C'$ subimage which it contains. The partial sums Σy and Σy^2 algorithms are based on the serial Σy and Σy^2 algorithms. As in the serial algorithms, a C'-element vector "colsum" is used to save the column sums computed so far. After processing of row k, $-r < k < R'$

$$\text{colsum (j)} = \begin{cases} \sum_{i=0}^{k+r-1} I(i,j) & -r < k \le 0 \\ \sum_{i=k}^{k+r-1} I(i,j) & 0 < k \le R' - r \\ \sum_{i=k}^{R'-1} I(i,j) & R' - r < k < R' \end{cases}$$

where $0 \le j < C'$. Unlike the serial and complete sums algorithms, for each row the leftmost sum consists of a single column sum (colsum (0)), and for $-c + 1 < j \le 0$, the sum for position (i, j) is computed by adding colsum $(j + c - 1)$ to the sum for position $(i, j - 1)$. Similarly, for $C' - c < j < C'$, the sum for position (i, j) is obtained by subracting colsum $(j - 1)$ from the sum for position $(i, j - 1)$. The sums (and colsums) for the topmost and bottommost r rows are computed in an analogous manner. In the "center" of each PE's subimage, the operations performed are identical to those in the serial and complete sums algorithms. The number of additions performed to generate the partial sums in each PE will be $4R'C' + 2C'r - 2R' - 4C' - 2r + 2$. As for the partial sums Σxy algorithm, the results which must be transferred are those in the non-"KEEP" area in Fig. 9. Each of these elements is added to a "kept" partial sum in the appropriate PE. The complexities of the partial sums Σy and Σy^2 algorithms are summarized in column two of Tables II and III.

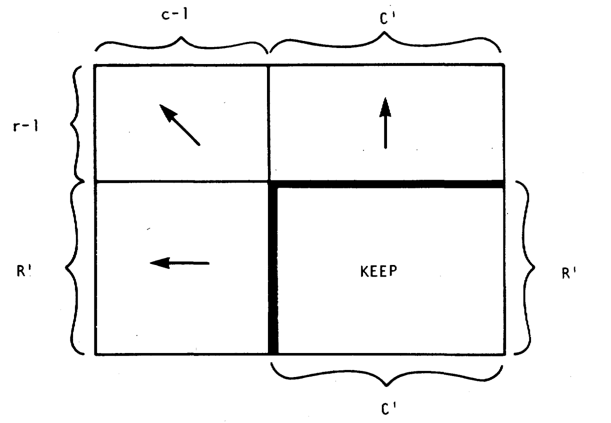


Fig. 9. Partition and direction of transfer of elements of partial sums array.

C. RXY and SXY Computation

As described for the complete sums method in Section IV-C, for image correlation measures RXY or SXY, the Σxy, Σy, and Σy^2 computations will be interleaved so that all three are computed for a given match position before the match area is moved to a new position. The computation of RXY is described.

The modifications required for the RXY computation involve the storage for partial (incomplete) Σxy, Σy, and Σy^2 sums. In the algorithms described, a non-"kept" partial sum was transferred from the PE in which it was computed to its destination PE, and stored in the memory location for the Σxy (or Σy or Σy^2) of which it was a part. For the complete RXY computation, space is not needed for all of a PE's local Σxy, Σy, and Σy^2 values. Provisions must therefore be made for the incomplete sums. The procedure will be based on the Σy algorithm. It will be explained in terms of three cases (ranges of match positions). It will be shown that at most $2[(C' + c - 1)(r - 1) + c - 1]$ locations are required to hold incomplete sums at any point in time.

Consider first a match position which is fully contained in the PE's subimage, i.e., position (i, j), where $0 \le i \le R' - r$ and $0 \le j \le C' - c$. The Σxy, Σy, and Σy^2 computations can be interleaved, and RXY for the position can be computed. For the same i range, when j exceeds $C' - c$ (i.e., $C' - c < j < C'$), two partial sums must be combined to produce the complete sum for each of Σxy, Σy, and Σy^2. After computing the Σxy partial sum for position $(i, C' - c + k)$, $1 \le k < c$, each PE can compute the partial sum for position $(i, -c + k)$ and transfer its value to the left, where the total Σxy sum for position $(i,$

$C' - c + k$) is then completed. By postponing computation of Σxy for position $(i, -c + k)$ until it is needed, no extra space is required for row i's incomplete Σxy values. A comparable savings in space cannot be realized in the Σy and Σy^2 computations, since the computation of these partial results cannot be postponed (without doing additional summations). Except for the topmost and leftmost edges, the Σy (and Σy^2) sum for each match position is computed in terms of the sum for a previous match position, so: 1) some previous results must be saved, and 2) the order in which the sums are computed cannot be altered or interrupted. The partial sum for position $(i, -c + k)$, $1 \leq k < c$, must be computed and saved until it can be combined with the partial sum for position $(i, C' - c + k)$. The same $c - 1$ locations can be used for each row i in the range. Thus, $c - 1$ locations are needed to save partial Σy results, and $c - 1$ locations for partial Σy^2 results for match positions (i, j) where $0 \leq i \leq R' - r$ and $-c + 1 \leq j < 0$.

Similarly, for partial match positions along the top of the PE's subimage, where the row index is $-r + k$, $1 \leq k < r$, computation of partial Σxy sums can be postponed until they are needed, but the partial Σy and Σy^2 sums must be computed and saved until the corresponding Σy and Σy^2 sums for row $R' - r + k$ have been calculated. Here, separate storage locations are needed for each row i, $-r < i < 0$, and column j, $-c < j < C'$. Therefore, for each of Σy and Σy^2, $(C' + c - 1)(r - 1)$ additional locations will be needed.

The partial sums complexity for computing RXY is summarized in Table IV. The time is a summation of that for computing Σxy, Σy, and Σy^2 for every match position. The transfers are for the non-"kept" partial sums. The space is for the PE's own subimage, the incomplete Σy and Σy^2 values which must be saved until they can be completed, and the intermediate results in calculating Σy and Σy^2.

As in the complete sums method, recursive doubling can be used to obtain the position of maximum correlation over all of the PE's.

VI. Comparison of Approaches

Tables I-IV contrast quantitatively the complete and partial sums approaches to the operations involved in image correlation. In order to more readily compare the two approaches, let $R' = C' = I'$ and $r = c = M'$. The results are shown in Table V. Figs. 10, 11, and 12 show an example of the differences in the number of addition, multiplication, and transfer steps, respectively, for the two strategies for a range of input image sizes when the match area size is 64×64 ($M' = 64$) and there are 256 PE's ($N = 256$).

As can be seen from the table, for each of the individual Σxy, Σy, and Σy^2 algorithms, the complete sums approach requires more space and/or arithmetic operations than the partial sums approach. However, when these algorithms are interleaved to compute and locate the maximum RXY value, the complete sums method requires more arithmetic operations, but fewer inter-PE transfers and less space. Which method is faster will therefore depend on the relative time to perform arithmetic operations versus transfers. For example, if the time to perform a transfer equals the time to perform a multiplication, then the complete sums method will be faster.

TABLE V
COMPARISON OF THE COMPLETE SUMS AND PARTIAL SUMS APPROACHES

	Ac-Ap	Mc-Mp	Tp-Tc	Space Difference
Σxy	0	0	0	Sc-Sp= 2(M')^2-4M'+3
Σy	8I'+3M'-3		0	Sc-Sp= (M')^2-1
Σy^2	8I'+3M'-3	2I'M'-2I' +(M')^2-2M'+1	0	Sc-Sp= (M')^2-1
RXY	16I'+6M'-6	2I'M'-2I' +(M')^2-2M'+1	2(2I'M'-2I' +(M')^2-2M'+1)	Sp-Sc=2I'M' -2I'-2M'+2

Notation:
$R' = C' = I'$
$r = c = M'$
Ap = Adds for Partial Sums Approach
Ac = Adds for Complete Sums Approach
Tp = Inter-PE Transfers for Partial Sums Approach
Tc = Inter-PE Transfers for Complete Sums Approach
Mp = Multiplies for Partial Sums Approach
Mc = Multiplies for Complete Sums Approach
Sp = Space for Partial Sums Approach
Sc = Space for Complete Sums Approach

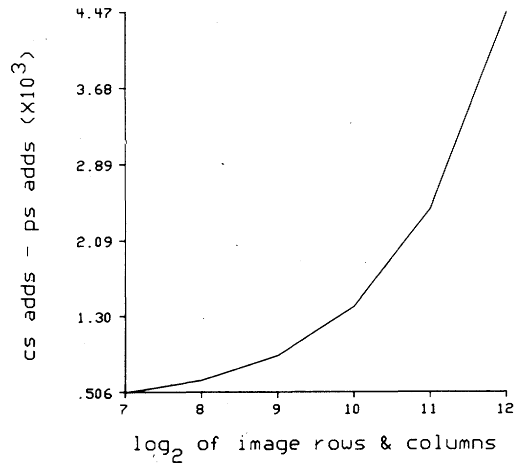


Fig. 10. Difference in the number of addition steps for the RXY computation between the complete sums (cs) and partial sums (ps) algorithms, under the assumptions that $R = C$, $r = c = 64$, and $N = 256$. Shown as a function of $\log_2 R$.

If inter-PE transfers can be overlapped with arithmetic operations, then the partial sums method will be faster. Thus, in order to determine which approach will be faster on a particular system, the exact timings for these operations must be considered.

The difference in the space required for the two approaches is not large. However, if the PE memories are small, or for the RXY computation if C' is large, the space difference may be a factor in selecting an algorithm.

Some basic differences resulting from the two algorithm strategies are evidenced in the RXY complexities. In the complete sums approach, two PE's hold and operate on some of the same image elements. As a result, redundant arithmetic

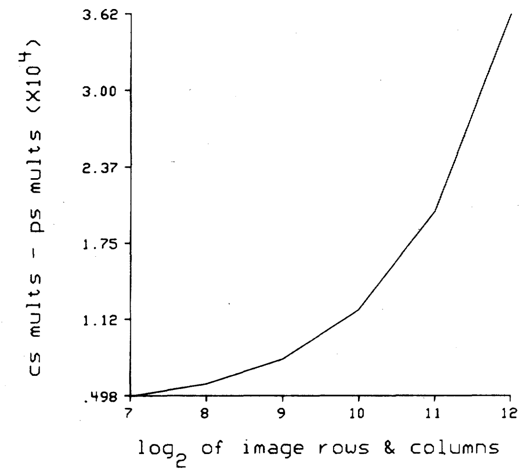


Fig. 11. Difference in the number of multiplication steps for the *RXY* computation between the complete sums (*cs*) and partial sums (*ps*) algorithms, under the assumptions that $R = C, r = c = 64$, and $N = 256$. Shown as a function of $\log_2 R$.

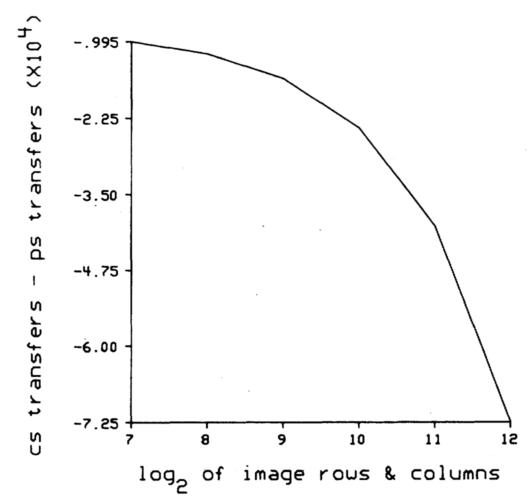


Fig. 12. Difference in the number of transfer steps for the *RXY* computation between the complete sums (*cs*) and partial sums (*ps*) algorithms, under the assumptions that $R = C, r = c = 64$, and $N = 256$. Shown as a function of $\log_2 R$.

operations are performed in the Σy and Σy^2 computations, i.e., the sum (or product) of the same two elements is sometimes performed in two PE's. These redundant operations are not performed in the partial sums algorithms. On the other hand, the partial sums method requires more transfers. Each nonlocal y value needed by a PE is transferred in only once in the complete sums approach and three times (as part of partial Σy, Σy^2, and Σxy terms) in the partial sums approach.

One aspect of parallel algorithms that may be of interest is the application of measures evaluating what portion of time is being spent "managing" the parallelism rather than performing useful computations [21]. One such measure is the ratio of communication ("overhead") time to computation time. For both approaches, for subimage sizes much greater than match area sizes, the order of magnitude of this ratio for the Σxy computation (and therefore for *RXY*, since the Σxy computation dominates the *RXY* computation) is proportional to the square root of the number of PE's and inversely proportional to the geometric mean of the sizes of the image and match areas:

$$\text{(communication time)/(computation time)}$$
$$\sim [R'(c-1) + C'(r-1) + (r-1)(c-1)]/(R'C'rc).$$

Under the assumptions that $R = C = I$, $R' = C' = I/\sqrt{N}$, and $r = c = M'$, it can be shown that this ratio is proportional to $\sqrt{N}/IM'$.

The SIMD machine model used assumed a multistage network which can perform each required data transfer in a single step. Consider instead an SIMD machine where the PE's are connected in a nearest neighbor pattern, i.e., PE i is connected to PE $i+1, i-1, i+N^{1/2}$, and $i - N^{1/2}$ (arithmetic mod N). Examples of such machines are the Illiac IV [4], DAP [8], CLIP4 [6], and MPP [3]. In analyzing the two algorithm approaches, the number of transfer steps must be increased. Assuming $N_C = N_R = N^{1/2}$, the nearest neighbor connection scheme requires 1 transfer step to do each of the PE i to PE $i+1, i-1, i+N_C$, and $i - N_C$ transfers, and 2 transfer steps to do each of the PE i to PE $i + N_C + 1, i + N_C - 1, i - N_C + 1$, and $i - N_C - 1$ transfers. Furthermore, if the match area extends over more than two PE's (see Fig. 6), additional multiple data transfer steps will be needed. (Even though two transfers are required for some steps, typically each transfer in a nearest neighbor network will be faster than a transfer through a multistage network.) The results in Tables I–V can therefore be applied to nearest neighbor connected systems by modifying the transfer step counts as described. (The number of transfers for recursive doubling will also be increased.)

In the SIMD machine model in Section I it was assumed that each processor was associated with a local memory to form a PE. Consider a different organization where the processors are separate from the memories, and the interconnection network is used to connect the processors to the memories. Interprocessor communications can be accomplished by writing into and reading from the shared memory. STARAN is an SIMD machine organized in this way [1], [2]. Since all memory accesses go through the interconnection network, there are no explicit interprocessor data transfers (assuming a network such as one of those mentioned in Section I were used). Thus, with such an organization the partial sums approach is faster than the complete sums approach. (In the STARAN machine, the interconnection network is not flexible enough to allow the processors to access the appropriate memories in all cases (e.g., processor i to memory $i + N_C + 1$). In these cases, an additional pass through the network will be required to align the data.)

It is also possible to make some general observations about the adaptability of the SIMD algorithms and strategies presented here to the MIMD (multiple instruction stream-multiple data stream) [9] mode of operation. In an MIMD system communicating processors have their own instruction streams as well as their own data streams. Although MIMD processing allows each processor to operate completely independently of the others, for this application it is most reasonable to consider a limited MIMD operation in which the processors are performing their computations asynchronously, but with each processor executing the same program. The performance of the image correlation algorithms in an MIMD environment will depend on the actual model assumed for the MIMD system.

One possible MIMD model is analogous to the initial SIMD model described, in which each processor has an associated local memory. For both the complete sums and partial sums approaches, efficient operation in the MIMD mode will require giving up the data processing sequencing and interleaving of operations introduced in the SIMD algorithms to save space. The complete sums algorithm can be implemented efficiently (with respect to execution time) by having all PE's initially send the data points to be shared to the appropriate neighboring PE's. Then the PE's can proceed asynchronously. Similarly, the partial sums algorithm can be implemented efficiently if each PE first computes all of its partial sums then transfers them to the PE's in which they will be needed. Each PE can then proceed independently. In both cases, since there will not be the implicit synchronization of SIMD processing, inter-PE transfer protocols will have to be established.

The alternative model discussed for an SIMD system (i.e., all memory accesses processed through the interconnection network) can also be a model for an MIMD system. With this model any processor can access any memory module. Processor i will still treat the subimage in memory i as the "local" data it is responsible for processing. To implement the complete sums approach, when processor j needs data from memory i, $i \neq j$, it just accesses memory j through the network. However, due to the asynchronous nature of MIMD machines, the algorithm timings may be increased as a result of network and/or memory contention. For the partial sums method to be feasible in this environment, it will again be necessary to compute all partial sums at the outset in order to avoid possibly long delays caused by a processor j requesting a partial sum that has not yet been computed from memory i. Processing can then proceed directly, with a provision that there must exist a mechanism for verifying that a requested partial sum has in fact been computed, and again with the possibility of degraded performance due to network and/or memory contention.

VII. Conclusions

The SIMD algorithms presented demonstrate how SIMD parallelism can be used to reduce the execution time of computationally intensive image processing tasks. For the image correlation algorithms, the asymptotic complexity for arithmetic operations is reduced from $0(RCrc)$ for the serial algorithm to $0(RCrc/N)$ for the N-PE parallel algorithms. The overhead of inter-PE communications incurred has asymptotic complexity $0(C'r + R'c + rc)$.

In summary, SIMD algorithms to perform the window-based operations needed for image correlation have been explored. Two fundamental algorithm strategies were presented, and their time-space-transfer complexities were compared. Through studies and analyses such as this, more can be learned about both the art of parallel programming and the ways in which parallelism can be exploited in image processing.

Acknowledgment

The authors thank G. B. Adams, III for his comments, and the referees for their helpful suggestions.

References

[1] K. Batcher, "The flip network in STARAN," in *Proc. 1976 Int. Conf. Parallel Processing*, Aug. 1976, pp. 65-71.
[2] ——, "The multidimensional access memory in STARAN," *IEEE Trans. Comput.*, vol. C-26, pp. 174-177, Feb. 1977.
[3] ——, "MPP—A massively parallel processor," in *Proc. 1979 Int. Conf. Parallel Processing*, Aug. 1979, p. 249.
[4] W. Bouknight *et al.*, "The Illiac IV system," *Proc. IEEE*, vol. 60, pp. 369-388, Apr. 1972.
[5] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis.* New York: Wiley, 1973.
[6] M. J. B. Duff, "CLIP4: A large scale integrated circuit array parallel processor," in *Proc. 3rd Int. Joint Conf. Pattern Recog.*, 1976, pp. 729-732.
[7] T. Feng, "Data manipulating functions in parallel processors and their implementations," *IEEE Trans. Comput.*, vol. C-23, pp. 309-318, Mar. 1974.
[8] P. M. Flanders, "Efficient high speed computing with the distributed array processor," in *Proc. Symp. High Speed Comput. Algorithm Organization*, Apr. 1977, pp. 113-128.
[9] M. Flynn, "Very high-speed computing systems," *Proc. IEEE*, vol. 54, pp. 1901-1909, Dec. 1966.
[10] A. J. Krygiel, "An implementation of the Hadamard transform on the STARAN associative array processor," in *Proc. 1976 Int. Conf. Parallel Processing*, Aug. 1976, p. 34.
[11] D. Lawrie, "Access and alignment of data in an array processor," *IEEE Trans. Comput.*, vol. C-24, pp. 1145-1155, Dec. 1975.
[12] M. Pease, "The indirect binary *n*-cube microprocessor array," *IEEE Trans. Comput.*, vol. C-26, pp. 458-473, May 1977.
[13] W. K. Pratt, "Correlation techniques of image registration," *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-10, pp. 353-358, May 1974; also in *Tutorial and Selected Papers in Digital Image Processing*, H. C. Andrews, Ed. New York: IEEE, 1978.
[14] A. Rosenfeld and A. C. Kak, *Digital Picture Processing.* New York: Academic, 1976.
[15] S. Ruben *et al.*, "Application of a parallel processing computer in LACIE," in *Proc. 1976 Int. Conf. Parallel Processing*, Aug. 1976, pp. 24-32.
[16] H. J. Siegel, "Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks," *IEEE Trans. Comput.*, vol. C-26, pp. 153-161, Feb. 1977.
[17] ——, "A model of SIMD machines and a comparison of various interconnection networks," *IEEE Trans. Comput*, vol. C-28, pp. 907-917, Dec. 1979.
[18] H. J. Siegel, L. J. Siegel, F. Kemmerer, P. T. Mueller, Jr., H. E. Smalley, Jr., and S. D. Smith, "PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Trans. Comput.*, vol. C-30, pp. 934-947, Dec. 1981.
[19] H. J. Siegel and S. D. Smith, "Study of multistage SIMD interconnection networks," in *Proc. 5th Symp. Comput. Arch.*, Apr. 1978, pp. 223-229.
[20] L. J. Siegel, H. J. Siegel, R. Safranek, and M. Yoder, "SIMD algorithms to perform linear predictive coding for speech processing applications," in *Proc. 1980 Int. Conf. Parallel Processing*, Aug. 1980, pp. 193-196.
[21] L. J. Siegel, H. J. Siegel, and P. H. Swain, "Performance measures for evaluating algorithms for SIMD machines," *IEEE Trans. Software Eng.*, to be published.
[22] H. Stone, "Parallel computers," in *Introduction to Computer Architecture*, H. Stone, Ed. Chicago, IL: SRA, 1975.
[23] K. J. Thurber, *Large Scale Computer Architecture: Parallel and Associative Processors.* Rochelle Park, NJ: Hayden, 1976.

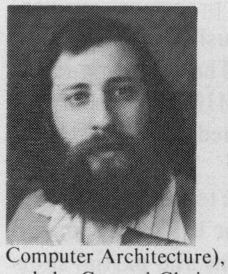
Leah J. Siegel (S'75–M'77) was born in Trenton, NJ, on August 27, 1949. She received the S.B. degree in mathematics in 1972 from the Massachusetts Institute of Technology, Cambridge, the M.A. and M.S.E. degrees in 1974, and the Ph.D. degree in 1977, all in electrical engineering and computer science, from Princeton University, Princeton, NJ.

Since 1976 she has been an Assistant Professor in the School of Electrical Engineering, Purdue University, West Lafayette, IN. Her research interests include speech analysis and recognition, and the design of parallel processing algorithms for digital speech, signal, and image processing. At Purdue she has been involved in the development of the Laboratory for One-Dimensional Signal Processing.

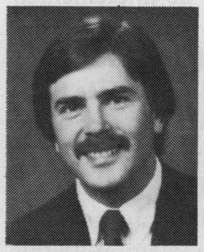
Dr. Siegel is a member of the Administrative Committee of the IEEE Acoustics, Speech, and Signal Processing Society. She is also a member of the Association for Computing Machinery, Eta Kappa Nu, and Sigma Xi.

Howard Jay Siegel (M'77) was born in New Jersey on January 16, 1950. He received the S.B. degree in electrical engineering and the S.B. degree in management from the Massachusetts Institute of Technology, Cambridge, in 1972, the M.A. and M.S.E. degrees in 1974, and the Ph.D. degree in 1977, all in electrical engineering and computer science, from Princeton University, Princeton, NJ.

In 1976 he joined the School of Electrical Engineering, Purdue University, West Lafayette, IN, where he is currently an Associate Professor. Since January 1979 he has also been affiliated with Purdue's Laboratory for Applications of Remote Sensing. His research interests include parallel/distributed processing, multimicroprocessor systems, image processing, and speech processing.



Dr. Siegel was Chairman of the Workshop on Interconnection Networks for Parallel and Distributed Processing held in April 1980, which was cosponsored by the IEEE Computer Society and the Association for Computing Machinery. He is currently a Vice Chairman of both the IEEE Computer Society TCCA (Technical Committee on Computer Architecture) and TCDP (Technical Committee on Distributed Processing), the Vice Chairman of ACM SIGARCH (Special Interest Group on Computer Architecture), an IEEE Computer Society Distinguished Visitor, and the General Chairman of the Third International Conference on Distributed Computing Systems, to be held in October 1982. He is a member of Eta Kappa Nu and Sigma Xi.



Arthur E. Feather (S'73) was born in Troy, NY, on October 2, 1953. He received the B.S.E.E. and M.S.E.E. degrees from the University of Kentucky, Lexington, in 1974 and 1976, respectively.

Currently, he is a Research Engineer at the Bellaire Research Center, Shell Development, Houston, TX. From 1974 to 1976 he served as a Research Assistant at the University of Kentucky. From 1976 to 1981 he was a Research and Teaching Assistant at Purdue University. His research interests include computer applications, systems design, parallel and distributed processing, and artificial intelligence.

Mr. Feather is a member of Tau Beta Pi and Eta Kappa Nu.