
ELSEVIER

Parallel Computing 25 (1999) 499–523

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Practical aspects

Aspects of computational mode and data distribution for parallel range image segmentation [1]

Nicholas Giolmas [a], Daniel W. Watson [b], David M. Chelberg [c,*], Peter V. Henstock [d], June Ho Yi [e], Howard Jay Siegel [a]

[a] *Parallel Processing Laboratory, School of Electrical and Computer Engineering, Purdue University, West Lafayette IN 47907-1285, USA*

[b] *Department of Computer Science, Utah State University, Logan, UT 84322-4205, USA*

[c] *School of Electrical Engineering and Computer Science, Russ College of Engineering and Technology, Ohio University, Athens OH 45701-2979, USA*

[d] *Lincoln Laboratory, Massachusetts Institute of Technology, Lexington MA 02173, USA*

[e] *School of Electrical and Computer Engineering, Sungkyunkwan University, 300, Chunchun-dong, Jangan-gu, Suwon 440-746, South Korea*

Received 14 November 1996; received in revised form 6 August 1998

Abstract

Parallel processing methods are a means to achieve significant speedup of computationally expensive image understanding algorithms, such as those applied to range images. Practical implementations of these algorithms must deal with the problems of selecting an appropriate parallel architecture and mapping the algorithm onto that architecture. The parallel implementation approaches for range image segmentation that are presented here are applicable to many low-level image understanding algorithms in a variety of parallel architectures. An evaluation of initial data distribution is presented to determine whether a square subimage or a striped subimage distribution would result in the greatest overall reduction in execution time for the given range image segmentation problem. Novel implementations that consider each data distribution's treatment of edge pixels in window operations yield a trade-off between the number of data transfers versus the amount of computation. This trade-off is examined both analytically and experimentally. Additionally, using the same initial data distributions, a

* Corresponding author. Tel.: +740 593 1251; fax: +740 593 0406; e-mail: chelberg@ohiou.edu

[1] This research was supported by the Office of Naval Research under grant number N00014-90-J-1937, by NRaD under subcontract number 20-950001-70 and contract number N66001-96-M-2277, and by the Digital Equipment Corporation Incentives for Excellence Grant. The equipment used was supported by the National Science Foundation under grant number CDA-9015696.

PII: S0167-8191(99)00007-1

technique is introduced for changing the allocation of work to each of the processors to reduce the number of network settings by one half. This technique and the method for determining the better initial data distribution can be used with any machine and any window-based technique that requires a full window to perform image calculations. Comparisons of range image processing algorithms are performed using "pure" SIMD algorithms, "pure" MIMD algorithms, and mixed-mode implementations with both SIMD and MIMD elements. Each of these approaches are quantitatively analyzed and compared for implementing the different phases of a particular hybrid range segmentation algorithm. Results of this implementation study indicate that quantifiable reductions in execution time result from the proper choice of parallel mode for each portion of the segmentation process.

Keywords: Computer vision; Hybrid range image segmentation; Image processing; MIMD; Mixed-mode; Parallel processing; PASM; SIMD

1. Introduction

Parallel processing methods are a means to achieve significant speedup of computationally expensive image understanding algorithms, such as those applied to range images. Any practical implementation of these algorithms must deal with the problems of selecting an appropriate parallel architecture [1,2] and mapping the algorithm onto that architecture. The parallel implementation approaches for range image segmentation that are presented here are applicable to many low-level image understanding algorithms in a variety of parallel architectures. An analysis is presented for a square subimage initial data distribution versus a striped subimage distribution. Novel implementations that consider each data distribution's treatment of edge pixels in window operations yield a trade-off between the number of data transfers versus the amount of computation. This trade-off is examined both analytically and experimentally. Additionally, using the same initial data distribution, a technique is introduced for changing the allocation of work to each of the processors to reduce the number of network settings by one half. This technique and the method for determining the better initial data distribution can be used with any machine and any window-based technique that requires a full window to perform image calculations.

Also in this study, comparisons of range image processing algorithms are performed using "pure" SIMD algorithms, "pure" MIMD algorithms, and mixed-mode implementations with both SIMD and MIMD elements. Each of these approaches are quantitatively analyzed and compared for implementing the different phases of a particular hybrid range segmentation algorithm. Results of the implementation study indicate that quantifiable reductions in execution time result from the proper choice of parallel mode for each portion of the segmentation process.

Several earlier studies have broadly claimed that SIMD mode should be used with low-level image processing and MIMD for higher-level image understanding algorithms [3–6]. Hybrid mixed-mode SIMD/MIMD parallel systems are well-suited to the implementation of complete image analysis tasks because they provide the

flexibility of matching each portion of the algorithm with the parallel mode that best suits it [7,8]. This advantage overcomes the limitations of pure SIMD and pure MIMD machines that were encountered in Ref. [9]. This paper examines the trade-offs of the different parallel modes and explores different data distributions, processor loading, and data transfer schemes to improve computation speed in a hybrid range image segmentation algorithm that embodies a broad class of image understanding algorithms.

In summary, three of the major contributions of this paper are: (1) a thorough evaluation of the impact of data distributions on the trade-off between communication time versus computation time when edge pixels are considered for this class of algorithms; (2) a technique for reducing the number of network settings by one-half for this class of algorithms; and (3) a comparison of SIMD, MIMD, and mixed-mode implementations of the application.

In range image understanding, it is necessary to reduce the information contained in an image from a collection of range measurements, one for each image picture element, or *pixel*, to a symbolic description of surface types and edges found in the image. To obtain a symbolic description, pixels are grouped into *regions*, connected sets of pixels that have unifying characteristics determined by a property of the range image data (e.g., surface normal). Images that have been partitioned into non-overlapping regions are called *segmented* images, and the term *segmentation* refers to this process of dividing an image into non-overlapping regions.

Segmentation is a computationally expensive operation with a high degree of uniformity for the operations applied to all pixels in an image. Thus, it is a good candidate for parallelization [1]. Selection of the parallel architecture that is best suited to the algorithm is a critical step in the algorithm mapping process. Furthermore, an efficient implementation for different portions of an algorithm may require using different parallel architectures. The selection of the optimum mapping of program segments to computational modes is not necessarily straightforward. By encoding multiple versions of each segment, different computation mode assignments for each segment can be compared. This work adopts a phase optimized approach for the entire algorithm, although, in general, this method is not guaranteed to produce the optimal implementation for the overall task [10]. In addition to being a useful application study, this work helps to build the body of knowledge that is needed to develop software tools that will facilitate the mapping of tasks onto parallel machines. Using the knowledge gained from studies such as this one, attempts are being made to develop techniques for choosing modes in a mixed-mode compiler [11]. Many mixed-mode architectures have been prototyped [7], and mixed-mode studies in part are important to determine in what applications, if any, mixed-mode parallelism can make a significant difference.

The partitionable SIMD/MIMD (PASM) parallel processing system [12,13], designed at Purdue, is one of a number of multiprocessor systems that are capable of *mixed-mode* parallelism, i.e., they can operate in either the SIMD or MIMD mode of parallelism, and can dynamically switch between modes at instruction level granularity [7]. PASM is a distributed memory machine, where each processor is paired

with a memory module forming a processing element (PE). A 16-PE small-scale proof-of-concept prototype has been built [12] and is being used as a test bed for application studies, e.g., Ref. [13].

In addition to PASM, five other mixed-mode machines have been built: TRAC (University of Texas at Austin, USA) [14], OPSILA (University of Nice, France) [15], Triton (University of Karlsruhe, Germany) [16], EXECUTE (IBM, USA) [17], and MeshSP (ICE/MIT Lincoln Labs, USA) [18]. More information about mixed-mode computing is in Ref. [7].

The prior work most related to the research presented here is [9]. As in this study, they perceive that some portions of the range-image segmentation process are best suited to SIMD, while others are better suited to MIMD. Additionally, [9] considers the impact of the initial distribution of range image data on overall algorithm performance. Separate SIMD and MIMD implementations of range image segmentation are presented in [9] using a Connection Machine 200 for the SIMD application and a Transputer-based system (a Meiko) for the MIMD implementation. The research here extends and complements the earlier work in Ref. [9].

However, the results introduced in this paper differ from [9] in several important ways. The overall focus of [9] is the selection of specific range segmentation algorithms that are well-suited to parallel machines. Here, a different implementation is studied, one whose basic components are used in many image-processing applications. Furthermore (and unlike [9]), the analysis of initial data distribution in this study indicates the number of operations saved for different distributions and the number of network settings needed, thus the results are more generally applicable to other image processing studies. Most importantly, the use of PASM in this study as a single reconfigurable architecture capable of both SIMD and MIMD modes of operation makes possible the direct comparison of SIMD and MIMD approaches without regard to differences in system architecture. Additionally, the use of the PASM reconfigurable architecture facilitates the selection of the most appropriate mode of parallelism for each phase of the algorithm in a mixed-mode implementation; developers need not be constrained to a 'pure' MIMD or pure SIMD algorithm.

Background information about range image processing is included in Section 2. A description of the SIMD, MIMD, and mixed-mode models of parallel architecture are discussed in Section 3. In Section 4, the effect of the initial placement of data on algorithm execution is examined. Section 5 compares different parallel implementations and provides results obtained from the study.

2. Segmenting range images

The techniques for range image segmentation can be classified into two categories: region-based and edge-based. A *region-based* approach attempts to group pixels into surface regions based on the homogeneity or similarity of image properties [19]. Alternatively, an *edge-based* approach detects discontinuities in

depth values and in surface orientations. The algorithm study in Section 4 examines a parallel implementation of a *hybrid* approach [20] to the problem of range image segmentation, which is a combination of region-based and edge-based approaches.

One motivation for the choice of this algorithm is that it is characteristic of many image processing tasks, so that implementation results for this study are applicable to a broad class of image processing problems. Additionally, the hybrid approach employs both region-based and edge-based segmentation methodologies, and the combining portion is representative of other kinds of algorithms (e.g., connected-component labeling, contracting, and expanding). Finally, the many different components of the algorithm make it difficult to map it effectively to a single architecture, i.e., different portions of the algorithm are best suited for different parallel implementations.

In the hybrid algorithm, a local biquadratic surface fit is employed to approximate object surfaces. This method and related methods (e.g., linear and cubic fit methods) are common in range image segmentation. The computations performed for similar surface fitting algorithms, such as [21] (which uses low-order bivariate polynomials) and [22] (which employs B-spline surface fitting) closely resemble the computations performed for the hybrid algorithm. The overall algorithm can be considered as representative of a wide range of image understanding tasks, because the computations involved in each phase are common to many segmentation and image processing problems.

The algorithm consists of three major stages (Fig. 1). In the first stage, differential geometric properties of a surface (i.e., surface normal, Gaussian curvature, and mean curvature) are locally estimated. Object surfaces are locally approximated in a window using second-order bivariate polynomials. In the first computational step, the six coefficients of the polynomial are determined by a least square method. Application of differential geometric concepts in the vicinity of discontinuities yields inaccurate estimates of geometric properties because real objects are only piecewise smooth. Accurate surface fitting may be achieved in the neighborhood of a discontinuity by selecting the window and offset that provides a minimum fitting error, referred to as *squared error*. Thus, the second step in local surface characterization is to compute fitting errors for each pixel in the image. The next step is to calculate the best offset for the local neighborhood, which is the location within the window that has the minimum fitting error. This offset is used to compute more accurate geometric measurements. Once these offsets are calculated, the fitted image can be computed. From these fitted polynomials, first and second partial derivative estimates are obtained and the surface normal, Gaussian (K), and mean (H) curvatures are computed from these partial derivatives. These geometric measurements are used to segment an image into homogeneous regions likely to correspond to object surfaces.

In the second stage, three types of initial segmentations are computed. A region-based segmentation is obtained in the form of the surface type map. Surface points are classified, according to the sign of K and H, into one of eight possible surface primitives [21]. To eliminate small surface regions, typically due to noise, the surface

type map is contracted and expanded once. Two edge-based segmentations are performed to detect jump and roof edges. The *jump edge* magnitude is computed as the maximum difference in depth between the point and its eight neighbors, while the *roof edge* magnitude is computed as the maximum angular difference between adjacent unit surface normals. Both jump and roof edge magnitudes are thresholded to produce edge maps.

In the final stage, the three initial segmentation maps are combined to produce a final region map where each region is homogeneous in curvature sign and contains no discontinuities. This final map can then be used in higher-level image understanding algorithms.

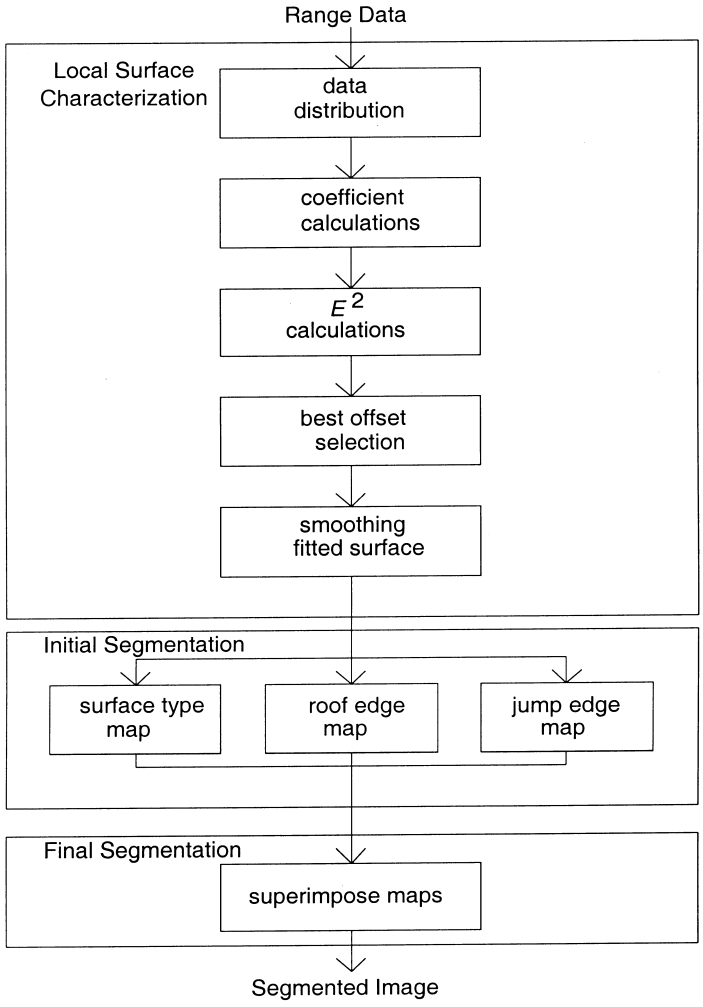

Fig. 1. Range image segmentation algorithm flow.

3. Modes of parallelism

There are trade-offs that exist between the SIMD and MIMD modes of parallelism that explain why some sequences of instructions perform better in one mode than in the other [7,23,24]. Some of the advantages and disadvantages of each mode, as they apply to image processing algorithms, are summarized here.

It is possible that the execution time of an instruction is data dependent, taking a variable length of time to perform on each PE. For example, on some processors the time it takes a floating point operation to execute is a function of the value of the operands. Variable-time instructions execute more efficiently in MIMD mode than in SIMD mode. In SIMD mode, the control unit (CU) broadcasts the next instruction to the PEs only after they have all completed the current instruction. Therefore, each instruction takes as long as it takes the slowest PE. In MIMD mode, the PEs are not synchronized and each PE executes the next instruction independently. More formally, let T_i^p represent the time it takes instruction i to execute in PE p. Assume that T_i^p in SIMD mode is equal to T_i^p in MIMD mode. The execution time in SIMD mode of a sequence of data-dependent instructions can be expressed as $\sum_i \max_p(T_i^p)$, for all i in the sequence. The time to perform the same sequence of instructions in MIMD mode can be expressed in terms of T_i^p as $\max_p(\sum_i T_i^p)$ (Fig. 2). Because $\max_p(\sum_i T_i^p) \leqslant \sum_i(\max_p T_i^p)$, the time to execute the sequence of data-dependent instruction in MIMD mode is less than or equal to the time to execute the same sequence of instruction in SIMD mode. Thus, MIMD mode is more appropriate for sequences of data-dependent instructions because of this "max of sums" versus "sum of maxs" effect. Many current processors use all fixed execution time instructions; however, the same effect occurs for constructs such as a sequence of "while" statements contained in a loop.

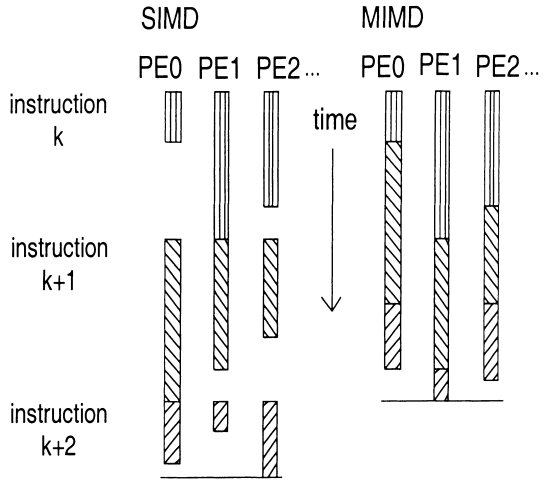

Fig. 2. Execution of variable-time instructions in SIMD and MIMD mode.

Conditional statements in the synchronous execution of an SIMD program can introduce serialization. Consider an if-A-then-B-else-C statement. Let the conditional test A depend on PE data. In some PEs, A is true and in others false. Those PEs where A is false are disabled (masked off) for the execution of clause B. Once B has executed, the PEs where A is true are disabled and the PEs where A is false are enabled. C is then executed. This serializes the execution of B and C. Conversely, in MIMD mode those PEs where A is true execute B, and PEs where A is false execute C. In MIMD mode, the maximum time for a PE to execute the if-A-then-B- else-C statement in a PE is approximately $T_A + \max(T_B, T_C)$, while in SIMD mode the time would be approximately $T_A + T_B + T_C$ (where the PE is idle for T_B or T_C). Thus, in general, MIMD mode is more effective for executing conditional statements.

Another distinction between SIMD mode and MIMD mode pertains to synchronization overhead. In SIMD mode, the synchronization of program execution is implicit, because there is a single thread of control. However, when synchronization of program execution is required among PEs in MIMD mode, explicit synchronization mechanisms, such as semaphores and barriers, must be employed in the parallel program. Thus, synchronization costs are greater for MIMD mode.

One benefit of implicit PE synchronization in SIMD mode becomes apparent when inter-PE data transfer are needed. In SIMD mode, when one PE sends data to another PE, all enabled PEs send data. Therefore, the ''send'' and ''receive'' commands are implicitly synchronized. Because all enabled PEs are following the same single instruction stream, each PE knows from which PE the message has been received and for what use the message is intended. Conversely, MIMD mode programs are executed asynchronously among all PEs. As a result, the PEs may need to execute explicit synchronization and identification protocols for each inter-PE transfer. While the details of the inter-PE transfer protocols in both SIMD and MIMD mode are implementation dependent, there is substantially more overhead associated with MIMD mode inter-PE transfers. Like the synchronization overhead above, this protocol overhead is a cost of the flexibility of programming in MIMD mode.

In SIMD mode, CU can be used to overlap operations with PEs [25]. For example, the CU can perform the increment and compare operations on loop control variables, while the PEs compute the contents of the loop. Furthermore, any operations common to all PEs, such as local subimage array address calculations, can be performed in the CU while the PEs are performing other computations. The CU can then broadcast this information to the PEs. In MIMD parallelism, each PE has its own set of instructions and its own instruction pointer. Because each PE performs all of the instructions, even those common to all PEs, the advantage of CU/PE overlap found in SIMD machines is not present. Furthermore, while a commercial MIMD machine can attempt to use a master/slave configuration to emulate SIMD, this would be quite inefficient. This is due to the excessive instruction level (or at least block level) synchronization that would be necessary, as well as due to the lack of a fast mechanism for broadcasting information to all of the PEs that a SIMD CU has for broadcasting common information it calculates concurrently with PE execution.

From the discussion above it is evident that there are many trade-offs between operating in SIMD mode and operating in MIMD mode. This motivated the design

of mixed-mode machines. Although it is often clear in which mode a sequence of instruction should be executed in a mixed-mode machine, this is not the case when counteracting trade-offs are involved. For example, a data-conditional statement may contain instructions that perform network transfers. Choosing the mode of an operation is not straightforward; conditional statements should be performed in MIMD mode while network transfers should be performed in SIMD mode. In such cases, the programmer may choose to code more than one version of the algorithm to determine the optimal approach.

When using the mixed-mode approach, new problems can arise such as the macro level "max of sums"/"sum of maxs" effect. In mixed mode, an entire block of instructions whose execution time varies on different PEs due to data conditional statements and/or variable execution time instructions may exhibit the same performance characteristics on a macro level if synchronization is required after the block (e.g., MIMD instructions in an SIMD loop) [23].

In summary, SIMD, MIMD, and mixed mode all have advantages and disadvantages. Consequently, the programmer of a mixed-mode system must be aware of the trade-offs between executing in SIMD mode and MIMD mode. Finding the optimal implementation involves algorithm analysis and experimentation.

4. Data distribution

The overall segmentation algorithm can be divided into stages, with each stage having its own optimal mapping on a parallel machine. A common parallel implementation issue among all the stages is the distribution of the range data to the processors. The way data is distributed among the PEs dictates the number of inter-PE transfers performed, and the source/destination pairs for each transfer. If establishing a new communication path between PEs is more costly than continuing to use the current setting, then a distribution method that promotes few such path creations (network settings) should be considered. System performance is determined in part by the amount of data transferred during each network setting.

For some algorithms (e.g., range data segmentation), the distribution method dictates the number of calculations performed on each PE. The method often used in parallel implementations of image processing algorithms distributes a square subimage to each PE. This minimizes the number of inter-PE transfers. For some algorithms, minimizing transfers is not as advantageous as minimizing the number of calculations required between transfers. In this section, an alternative method that minimizes the number of calculations is presented. It is based on distributing consecutive rows rather than square subimages of data to the PEs.

Many calculations in segmentation algorithms involve w-by-w windows of data (e.g., convolutions, local minima). For some of the pixels in the image, specifically those located on the perimeter, window calculations need not be performed because they lack the necessary data over the entire w-by-w window. These *border* pixels play an important role in the selection of the distribution method. The proposed *horizontal stripe* method allows window algorithms to take advantage of the fact that no

calculations need to be performed for the border pixels, thereby decreasing the overall number of required calculations.

Let the original image be of size M-by-M, the number of PEs on the target machine be N and the window size for the calculations be w-by-w, where w is a positive odd integer. A pixel is processed using data from a w-by-w window with that pixel as the center. In the following discussion, it is assumed that $M \geqslant N$, and typically $N \geqslant 64$, $M \geqslant 128$ and $w > 3$. For simplicity it is also assumed that M is a multiple of N, although the obtained results can be adopted to cases where this restriction does not apply. The first two stages of the algorithm, the coefficient and squared error calculations, and the subsequent data transfers, are chosen as an example of a general processing scenario in which a sequence of calculations followed by data transfers are performed.

Typically, image processing algorithms that rely heavily on inter-PE transfers have used the square subimage distribution method as shown in Fig. 3. This method distributes a unique square portion of size $(M/\sqrt{N})$-by-$(M/\sqrt{N})$ from the original image to every PE. The method succeeds in minimizing the number of data transfers by using subimages with a square perimeter. For the algorithm stages under consideration, each PE must transfer $4\lfloor w/2 \rfloor \times (M/\sqrt{N}) + 4(\lfloor w/2 \rfloor)^2$ floating point data elements (Fig. 4). All N PEs can send to unique destination PEs simultaneously, allowing up to N concurrent data item transfers. Each PE must share data with a

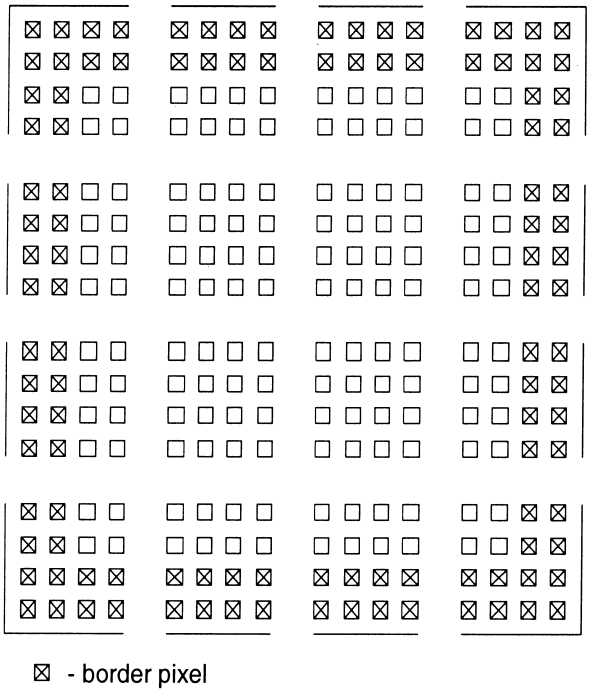

Fig. 3. Square distribution example, $M = 16$, $N = 16$, $w = 5$.

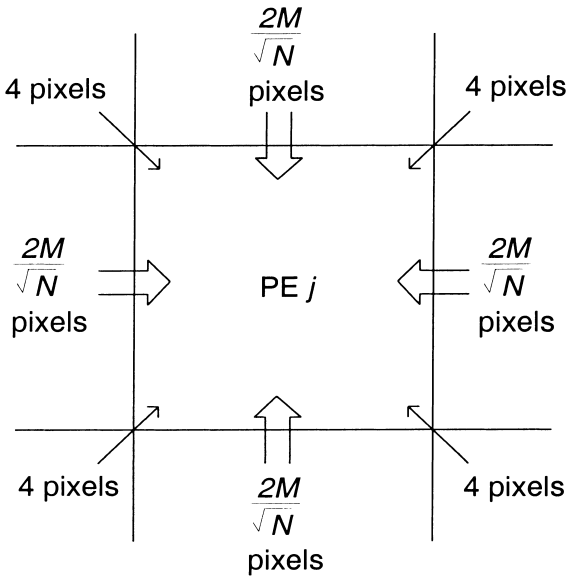

Fig. 4. Pixel transfers for $w = 5$.

maximum of eight neighboring PEs. A minimum of four network settings are required to perform a transfer operation using this method (e.g., in Fig. 4, the data elements to be sent from PEs on the diagonal from PE j can be sent through the PEs to the left and right of PE j).

The square subimage method does not take advantage of the fact that border pixel calculations can be omitted because the border pixels are distributed unevenly among the PEs. Some PEs will contain no border pixels and consequently perform the maximum number of pixel operations, given by $(M/\sqrt{N}) \times (M/\sqrt{N})$ or M^2/N. These PEs dictate the amount of time required to complete the parallel calculation task.

Using the proposed horizontal stripe method shown in Fig. 5, each PE initially contains M/N rows of data. Each PE transfers data to its two neighboring PEs. One way to do this is to have PE j send the bottom $\lfloor w/2 \rfloor$ rows of the subimage to PE $j+1$ and the top $\lfloor w/2 \rfloor$ rows to PE $j-1$. As discussed later, the actual direction of the transfers varies depending on the chosen implementation, but the total amount of data transferred is $2\lfloor w/2 \rfloor$ rows (i.e., $2\lfloor w/2 \rfloor \times M$ pixels). Although the number of data elements transferred is increased, the number of network settings is decreased to a maximum of two. The border pixels located on the left and right side of the image are uniformly distributed over all PEs, and operations on these border pixels for each PE are not performed. When the transfers are complete, each PE will calculate the six coefficient values and the squared error value for at most $(M/N) \times (M - 2\lfloor w/2 \rfloor)$ or $(M^2/N) - 2\lfloor w/2 \rfloor \times (M/N)$ pixels. Let T_t denote the time required to transfer one data element, and T_0 denote the time required to perform an operation for one pixel, which may consist of many calculations. The *operation transfer ratio* is defined as

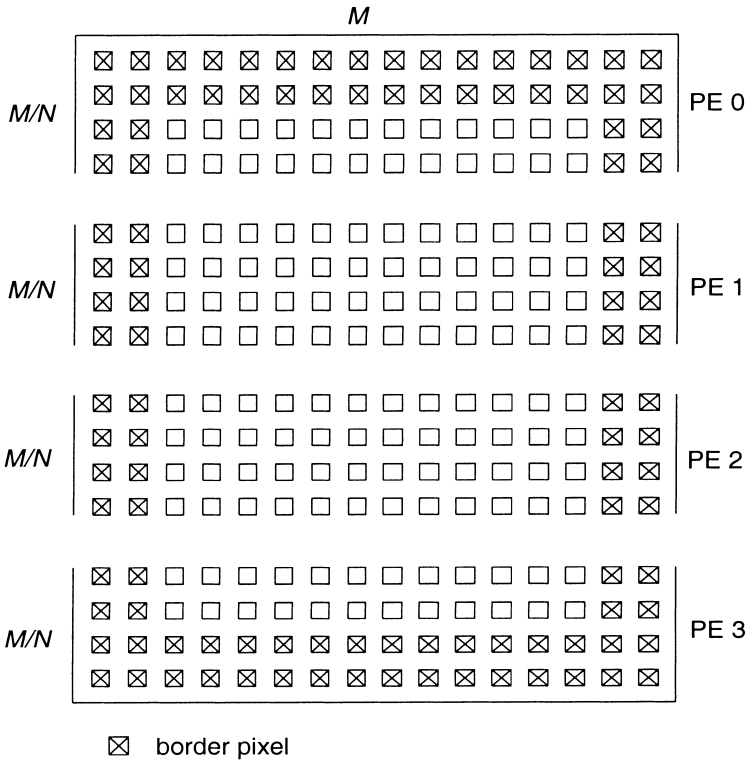

Fig. 5. Striped distribution example, $M = 16$, $N = 4$, $w = 5$.

$\rho = T_0/T_t$. Experimental results on PASM have shown that $\rho \approx 2500$ for these segmentation operations (i.e., calculating the six coefficient values and squared error value for each pixel).

The time penalty, T_p, for processing the extra $2\lfloor w/2 \rfloor \times (M/N)$ pixels using the square method is

$$T_p = 2\left\lfloor \frac{w}{2} \right\rfloor \times \frac{M}{N} \times T_0.$$

If network setups are time consuming, the time required by the square method for the extra needed setups should be included in T_p. Just counting the number of pixels sent, the time penalty, T_h, for performing the extra intra-PE transfers by using the horizontal stripe method versus the square method is

$$T_h = \left[2\left\lfloor \frac{w}{2} \right\rfloor M - \left(4\left\lfloor \frac{w}{2} \right\rfloor \times \frac{M}{\sqrt{N}} + 4\left(\left\lfloor \frac{w}{2} \right\rfloor \right)^2 \right) \right] \times T_t.$$

The extra time required by the square method to complete this stage of the algorithm is $T_p - T_h$.

The value of ρ depends on the particular algorithm and machine used for execution. For a given image size, window size, and number of PEs, ρ will uniquely

determine the optimal distribution method for the algorithm under consideration. Let $\rho_{\text{breakeven}}$ designate the value of ρ such that if $\rho > \rho_{\text{breakeven}}$, then the striped distribution method should be chosen over the square distribution method. Fig. 6 shows the value of $\rho_{\text{breakeven}}$ as a function of the number of PEs (N) for which the stripe method should be chosen, given an image size of $M = 1024$ and window size of $w = 5$. Consider a target system with 256 PEs and an image size of $M = 1024$. If $\rho > 223$, then the stripe distribution method should be used to implement this algorithm. This analysis technique is quite general, and can be applied to other algorithms and machines.

Thus, distribution of image data should not always be done in squares, as is usually the case. In general, the horizontal stripe method performs faster whenever calculations on border pixels can be omitted, and transfers are relatively fast compared to calculations.

When network setups are time consuming, there is further gain in using the stripe method. The number of network settings can be reduced from four settings needed for the square subimage distribution scheme to only two for the striped subimage distribution scheme. This can be beneficial if network setup times are significant, as can be the case for circuit-switched networks and networks that employ wormhole routing [26]. With these routing approaches, there is a relatively higher cost for establishing a path between a given source PE and a given destination PE, and a relatively smaller cost for each pixel transfer. In these cases, it is possible to further reduce the number of network settings by modifying the set of pixels that are considered by each PE.

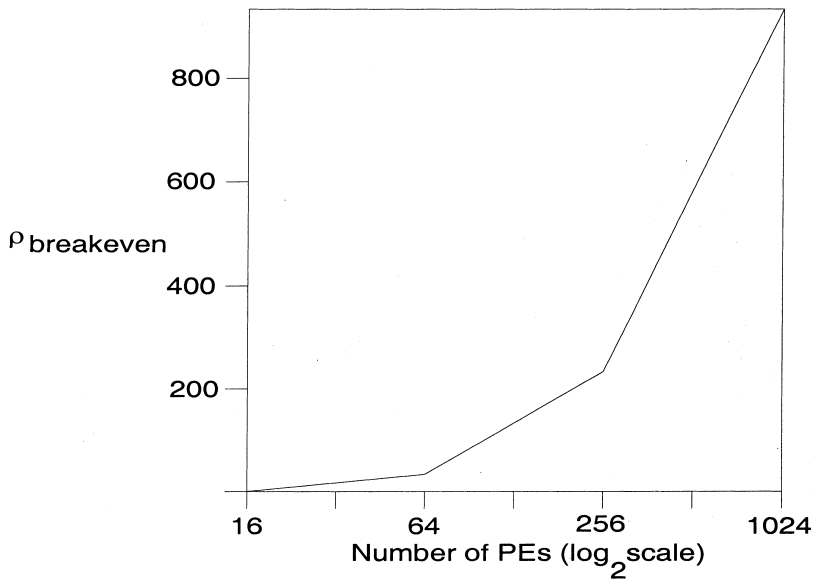

Fig. 6. $\rho_{\text{breakeven}}$ vs. N for $M = 1024$ and $w = 5$.

Consider the case using the striped subimage distribution scheme with N PEs operating on an M-by-M image, such that each PE initially contains an M/N row by M column subimage. Rows and columns are numbered 0 to $M-1$. The first row of pixel data contained in PE j is then row $j \times M/N$ of the original subimage, and the last row in PE j is row $((j+1) \times M/N) - 1$. Instead of sending $\lfloor w/2 \rfloor$ rows of M pixels to each of two neighboring PEs, let PE j send its first $w-1$ rows (i.e., $2 \times \lfloor w/2 \rfloor$ rows) of pixel data to one neighbor, PE $j-1$ *modulo* N (the pixels received by PE $N-1$ from PE 0 are ignored). This is illustrated in Fig. 7 using 3-by-3 windows. Only one network setting is required, because only one destination is selected for each PE. PE j then has data for all the pixels in rows $j \times M/N$ through $((j+1) \times M/N + w - 2)$ *modulo* M, and can then perform w-by-w window calculations on the M/N image rows $(j \times M/N) + \lfloor w/2 \rfloor$ through $(j+1) \times M/N + \lfloor w/2 \rfloor - 1$. In effect, the problem has been shifted "down" $\lfloor w/2 \rfloor$ rows,

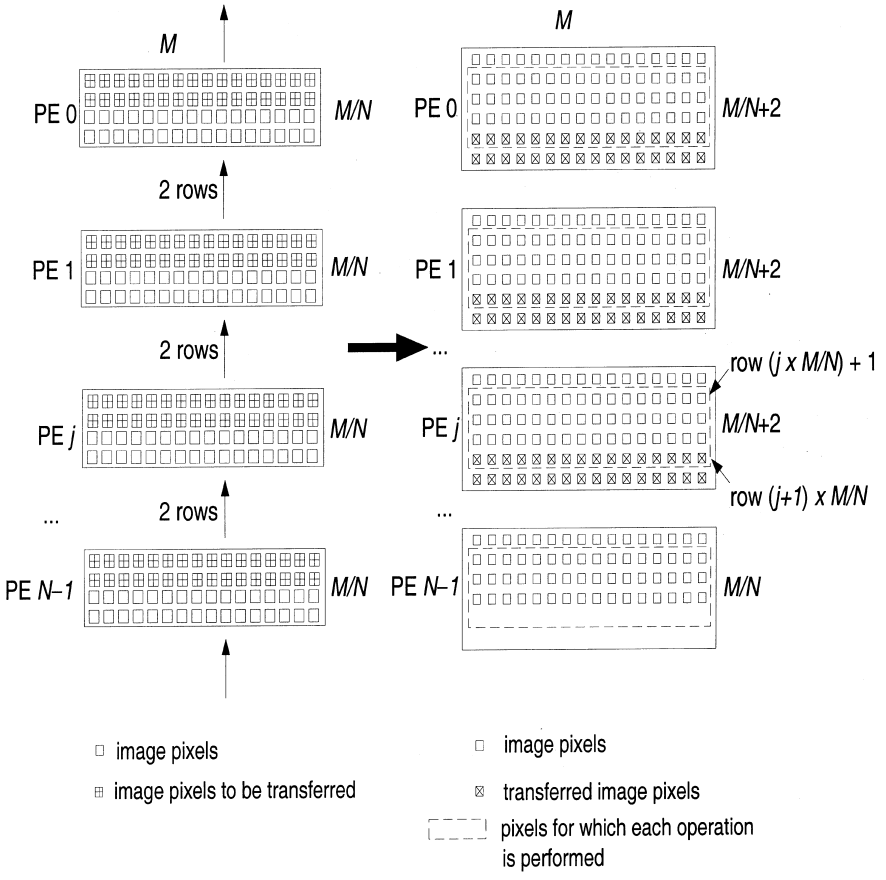

Fig. 7. Method for performing all transfers needed to calculate 3-by-3 window operation on image using a single network setting.

making it possible to perform all the necessary transfers with one network setting. PE $N-1$ is the only exception; it has to process only $M/N-w+1$ image rows. For Fig. 7, where $w=3$, PE j then has data for all the pixels in row $j\times M/N$ through $((j+1)\times M/N+1)$ *modulo* M, and can then perform 3-by-3 window calculations on the M/N image rows $(j\times M/N)+1$ through $(j+1)\times M/N$.

As with the striped subimage scheme that uses two network settings, PEs 1 to $N-2$ each operate on an M/N-by-M subimage. However, in the version that uses a single network setting, PE 0 operates on $M/N\times M$ meaningful pixels (as opposed to $(M/N-\lfloor w/2\rfloor)\times M$ for the two-setting case), while PE $N-1$ calculates $(M/N-w+1)\times M$ meaningful pixels (as opposed to $(M/N-\lfloor w/2\rfloor)\times M$). No PE operates using more than $M/N\times M$ meaningful pixels (as in the two-setting case). Thus, if changing the network setting takes extra time for a machine, the single network setting version can be executed in less time than its two-setting counterpart.

For iterative algorithms, the same network setting will allow PEs to receive the information they need to compute the next iteration. Each PE sends the top $w-1$ rows of information to PE $j-1$. Each PE now must compute a different portion of the image (shifted slightly), but each PE has enough data to compute its share of the overall workload, with the exception of the PEs at the bottom. Because edge pixels are ignored, the overall image size shrinks with each iteration, and therefore, there is less work for the PEs at the bottom of the image (i.e., no PE uses more than $M/N\times M$ meaningful pixels). Fuller utilization of the PEs would require redistribution of the data, entailing additional network settings, and communication costs. The one network setting scheme is most beneficial when network settings are relatively expensive.

Even if the square subimage distribution scheme is used, the number of network settings can still be reduced using this method. Consider the case using the square subimage distribution scheme for N PEs operating on an M-by-M image, such that each PE initially contains an $M/\sqrt{N}$ row by $M/\sqrt{N}$ column subimage. Assume the PEs are logically arranged as a two-dimensional array. Let PE j be denoted PE(k, l), where $k=\lfloor j/\sqrt{N}\rfloor$ (i.e., the row of PE j) and $l=j$ *modulo* $\sqrt{N}$ (i.e., the column of PE j). Then PE(k, l) has rows $k\times(M/\sqrt{N})$ through $(k+1)\times(M/\sqrt{N})-1$ for columns $l\times(M/\sqrt{N})$ through $(l+1)\times(M/\sqrt{N}-1)$. Each PE($k$, l) sends its first (top) $w-1$ rows of pixel data to PE ($k-1$ *modulo* $\sqrt{N}$, l), and then sends its first (leftmost) $w-1$ columns of pixel data to PE(k, $l-1$ *modulo* $\sqrt{N}$). PE(k, l)'s subimage rows $k\times(M/\sqrt{N})$ through $k\times(M/\sqrt{N})+w-2$ of columns $l\times(M/\sqrt{N})$ through $(l\times(M/\sqrt{N})+w-2$ can be sent to PE ($k-1$ *modulo* $\sqrt{N}$, $l-1$ *modulo* $\sqrt{N}$) through PE($k-1$ *modulo* $\sqrt{N}$, l). This occurs as part of the second data transfer. For all transfers, data received by PEs (0, l) and $(k,\sqrt{N}-1), 0\le l,\ k<\sqrt{N}$ is ignored.

In this case, only two network settings are required, because only two destinations are selected for each PE. PE(k, l) then has data for all the pixels in the subimage containing rows $k\times(M/\sqrt{N})$ through $(k+1)\times M/\sqrt{N}+w-2$ and columns $l\times(M/\sqrt{N})$ through $(l+1)\times M/\sqrt{N}+w-2$. (The PEs on the edges of the logical array have some special conditions). Thus, PE(k, l) can then perform w-by-w window calculations on a subimage with rows $k\times(M/\sqrt{N})+\lfloor w/2\rfloor$ through $(k+1)\times$

$M/\sqrt{N} + \lfloor w/2 \rfloor - 1$, and columns $l \times (M/\sqrt{N} + \lfloor w/2 \rfloor$ through $(l+1) \times M/\sqrt{N} + \lfloor w/2 \rfloor - 1$. Analogous to the previous example, the problem has been shifted "down" $\lfloor w/2 \rfloor$ rows and "right" $\lfloor w/2 \rfloor$ columns, making it possible to perform all the necessary transfers with two network settings.

This is illustrated graphically in Fig. 8 for a 3-by-3 window operation when $M/\sqrt{N} = 4$ (PE numbers and row/column numbers have been omitted for clarity). In Fig. 8(a) each PE sends its top two rows of pixels to the PE "above" it. As each PE receives the pixels from the PE "below," it forms an $(M/\sqrt{N}) + 2$ row by $M/\sqrt{N}$ column subimage. Then, as illustrated in Fig. 8(b), each PE sends its two leftmost columns of pixels to the PE to the "left." As each PE receives pixels from its "right" neighbor, it forms a square subimage of size $(M/\sqrt{N}) + 2$ rows by $(M/\sqrt{N}) + 2$ columns. Each PE can then perform a 3-by-3 window operation on a $M/\sqrt{N}$ by $M/\sqrt{N}$ subimage, effectively moving the problem "down" and to the "right".

For iterative algorithms, the process may be repeated. With each iteration the problem moves "down" and to the "right." Because edge pixels are ignored, the image is shrinking with each iteration and no PE uses more than $M/\sqrt{N} \times M/\sqrt{N}$ meaningful pixels (i.e., the problem never hits the top and left edges of the image, and PEs on the opposite edges just have less work to do).

5. Parallel implementation

The serial algorithms that compute the phases of the overall segmentation algorithm were mapped onto the target parallel machine. Due to its greater efficiency for our task, the striped distribution method was used. Theoretical analysis and experimental results were used to determine the best mapping for each phase. The ability of PASM to switch between parallel modes can be exploited to obtain the optimal mode selection in a single-mode implementation or the best combination of parallel modes in a mixed-mode implementation. Examples of implementation studies on the PASM prototype for other types of applications (with different computational characteristics) include [10,27–29].

In Ref. [10], a practical image-processing algorithm, an edge-guided thresholding algorithm, was used to study the phase optimization technique, where the programmer makes an implicit assumption that by combining the the best version of each phase the optimal implementation will be achieved. Results of the study demonstrate that the phase-optimized approach can result in a suboptimum mixed-mode implementation, due to the effects of the temporal juxtaposition of phases of the algorithm at execution, resulting in the macro-level "max of sums"/"sum of maxs" effect discussed in Section 3. Thus, if a phase optimization approach is used, it is important to be sure this detrimental effect does not occur.

The phase-optimized approach is used here to find an implementation that makes effective use of computational modes to reduce overall execution time. For the computation required for this task and the assignment of modes discussed below, the phase optimization was beneficial.

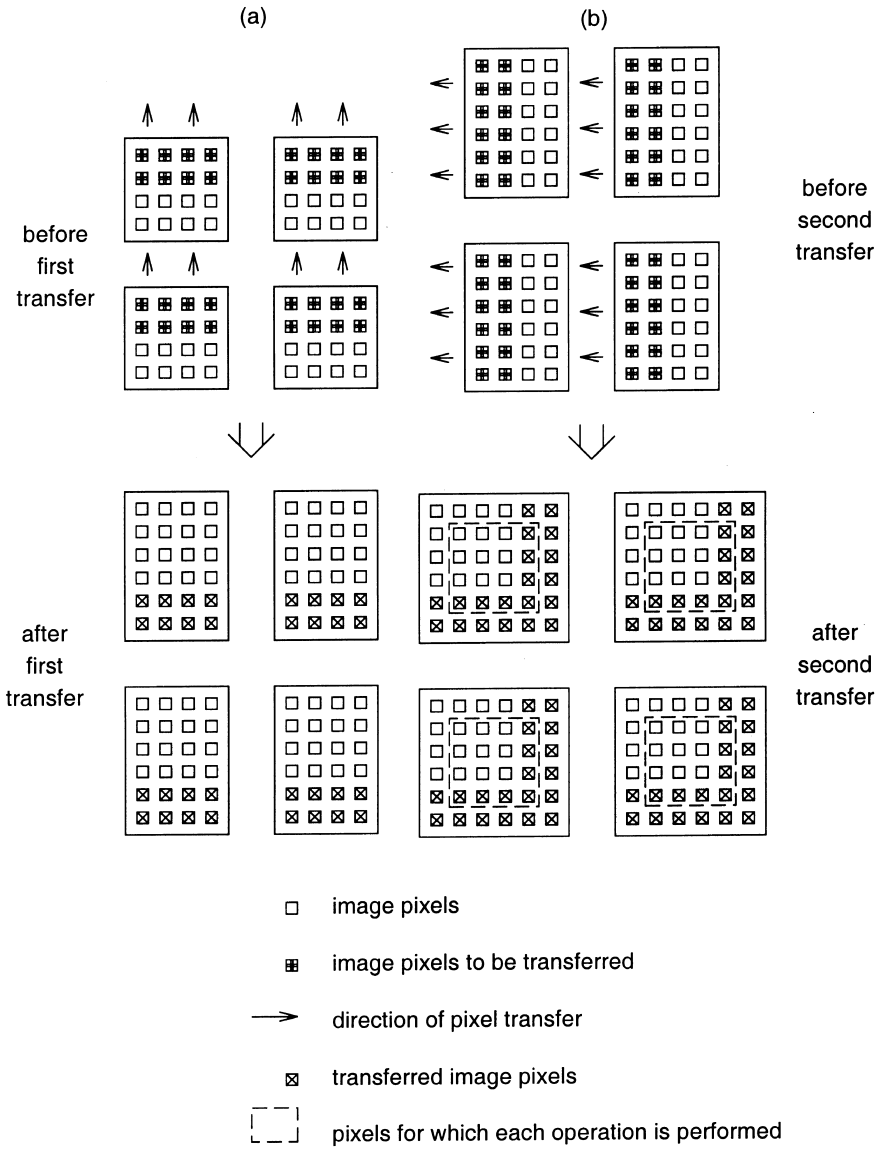

Fig. 8. Method for performing all transfers needed to calculate a 3-by-3 window operation on square subimages using two network settings (PE and row/column numbers omitted for clarity).

In determining the optimal parallel mode of execution for each phase of the segmentation algorithm, four basic trade-offs were considered (described in Section 3:

1. Variable instruction execution times ("max of sums"/"sum of maxs").
2. Non-uniform program flow (includes *if – then – else* statements).

3. CU/PE overlap.
4. Inter-PE transfer synchronization advantage.

While (3) and (4) favor SIMD mode, (1) and (2) favor MIMD mode.

Counteracting trade-offs must be quantified. For example, most of the calculations in the segmentation algorithm involve floating point data that may require data-dependent instruction execution times, as is the case for the PASM prototype, invoking trade-off (1). Conversely, many of the calculations are required for every pixel in the image, rendering the algorithms that perform them highly iterative. The CU/PE overlap (3) ability of SIMD or mixed-mode can be utilized by letting the CU execute the loop iterations while the PEs are calculating the desired result.

Table 1 lists the various phases of the segmentation algorithm in the required order of execution along with their optimal parallel mapping mode, determined through experimentation. The underlying reasons behind the mode choice are also listed. To demonstrate how this was done, five of the phases are described below, particularly, coefficient calculation, squared error calculation, smoothing and type selection, and the contraction and expansion of the surface-type map.

In the segmentation algorithm, after a particular phase is completed data transfers may be performed prior to the execution of the next phase (e.g., prior to the best offset selection, squared error data must be transferred). Transfers are better performed in the SIMD mode of operation, as described in Section 3.

In the first phase, object surfaces in the range image are locally approximated using second-order bivariate polynomials of the form

$$z = ax^2 + by^2 + cxy + dx + ey + f.$$

The coefficients a, b, c, d, e, and f are determined for each pixel in the original image by a least-squares method. For each pixel (i, j) in the original image, six convolution operations are performed for a u-by-v window centered at pixel (i, j) (for this study, $u = v = 5$). Each convolution is a running sum of element-wise multiplications between a distinct operator and the original range data [20]. The range image is initially distributed among the PEs, and necessary data transfers are performed before the coefficient calculations. Results of operations for pixels on the top

Table 1
Optimal parallel mode selection for each phase of the segmentation algorithm

Phase	Optimal mode	Reason
Coefficient calculation	MIMD	1
Squared error calculation	MIMD	1
Inter-PE data transfers	SIMD	4
Best offset selection	MIMD	2
Smoothing and type selection	MIMD	1, 2
Inter-PE data transfers	SIMD	4
Contraction of surface map	mixed-mode	2, 3
Expansion of surface map	mixed-mode	2, 3, 4
Boundary pixel selection	mixed-mode	2, 3
Roof and jump edge selection map superimposing	MIMD	1, 2

and bottom borders of the original image are discarded (only the bottom border is computed unnecessarily with the one network setting approach).

The convolution operators exhibit coherence in their structure, and partial results from one convolution for a pixel can be used on subsequent convolutions for the same pixel. It becomes advantageous to perform all six convolutions for the same pixel in one routine, because the number of floating point calculations is decreased due to the use of partial results. Additionally, if the individual loops for each convolution are unrolled (i.e., separate sections of code are used, one for each convolution), the extra effort for the loop increment and test is eliminated. The routine contains a sequence of floating point operations, whose execution time is dependent on the data. The sum of all the execution times varies considerably between PEs, which makes synchronization between loop iterations expensive.

Because floating-point multiplication and addition operations are data-dependent, an MIMD implementation can take advantage of the "sum of maxs" vs. "max of sums" rule, i.e., no synchronization is required between successive operations. Conversely, because loop overhead operations can be performed on the CU while calculations are being performed on the processing elements, an SIMD implementation may also be beneficial. Both an MIMD and an SIMD version of the coefficient calculation phase were coded. Timing measurements for this phase indicate that the MIMD advantages outweigh the SIMD advantages for this portion of the algorithm with the SIMD version taking 10% longer than the MIMD.

In the next phase, the squared error value for each pixel is calculated using the obtained coefficients. Let $z(x,y)$ denote the original image value for pixel (x, y). Then

$$[\textit{squared error}](x,\ y\colon a,\ b,\ c,\ d,\ e,\ f) = \sum_{v=-2}^{2}\sum_{u=-2}^{2}[au^2 + bv^2 + cuv + du + ev + f - z(x+u,\ y+v)]^2.$$

The structure of the operation can be exploited and the inner summation loops can be unrolled, as is the case for the coefficient calculations. This restructuring produces an algorithm that closely resembles the coefficient calculation phase, and thus similar considerations about the choice of operating mode apply.

One possible mixed-mode approach for these two phases is to execute the loop control overhead on the CUs in SIMD mode, and perform the loop body in MIMD mode. In this case, the mixed-mode implementation performs faster than the SIMD version, but more slowly than the MIMD version. No synchronization occurs among the PEs within the loop body, but each PE must still wait at the beginning of the next loop iteration until all of the PEs have executed the current loop iteration, because loop control is performed in SIMD mode. Although the operation is iterative, the penalty for synchronizing the PEs after each iteration to overlap the loop operation on the CU becomes significant as the macro "max of sums"/"sum of maxs" rule dictates.

In the "smoothing and type selection" phase, the smoothing, surface normal computation, and surface type selection for each pixel are performed. Each PE has all the necessary data required to perform these three operations. Synchronization of

the PEs between these operations is not necessary because no transfers are required. The smoothing operation for each pixel (x, y) is obtained by first computing the relative offset (u, v) from pixel (x, y) of the pixel within its 5-by-5 neighborhood that has the minimum squared error value determined in the best offset selection phase. The new "fitted" value, $z(x, y)$, for pixel (x, y) is computed using the coefficient values of the pixel$(x-u,\ y-v)$:

$$z(x,y) = a(x-u,y-v)u^2 + b(x-u,y-v)v^2 + c(x-u,y-v)uv + d(x-u,y-v)u + e(x-u,y-v)v + f(x-u,y-v).$$

From the derivatives of the above equation, the Gaussian and mean curvature values are obtained for each pixel (x, y). The surface type for pixel (x, y) can now be computed from these curvature values through a series of conditional statements. The normal vector for each pixel, $\vec{n}(x, y)$, is also computed from the derivatives.

The surface type classification requires conditional clauses to determine which of the eight surface primitives to use for the pixel. For this portion of the algorithm, SIMD mode has the disadvantage of requiring each conditional *then* and *else* clause to be performed serially. Because there is only one control stream, only those PEs for which a condition clause is to be executed are active, while the other PEs in the system are disabled.

The operations are combined in one routine and performed in MIMD mode because of the large number of floating point calculations involved in the smoothing and normalization phases, the conditional statements in the surface type selection, and the fact that no synchronization is required. The PEs are synchronized at the completion of these operations by switching to SIMD mode, to perform inter-PE transfers of the newly computed data.

Four implementations for the smoothing and type selection phase were considered. In the first, the entire algorithm was mapped to MIMD mode. In the second implementation the outer loop was performed in SIMD on the CU and the contents of the loop were performed on the PEs in MIMD mode; the mixed-mode took 35% longer than the MIMD. In the third implementation the outer loop was performed in SIMD mode on the CU, floating point calculations were also performed in SIMD on the PEs, and only the conditional statements were performed in MIMD on the PEs. These mixed-mode implementations, intended to overlap the PE execution with the CU loop operations in SIMD, failed to produce faster execution times than the pure MIMD version. This is once again due to the macro "sum of maxs" effect observed in the coefficient calculation phase. The pure SIMD version suffered from both inefficient conditional statement execution and the "max of sums"/"sum of maxs" effect on the data-dependent execution times, resulting in a time 51% longer than MIMD.

In the contraction and expansion phases of the range image segmentation algorithm, portions of the range image are grouped together to form segments consisting of contiguous sets of pixels that have the same surface primitive classification. In the contraction operation, any pixel in the image that has as its neighbor a pixel with a surface primitive classification different from its own is removed from the image and replaced with a void surface classification. In the expansion operation, the opposite

occurs: any void-classified pixel is replaced with the classification of a non-void neighbor. Contraction may be performed multiple times before the first expansion occurs.

The obtained initial surface type map is contracted and expanded to eliminate small regions typically due to noise (this occurs in the "surface type map" component of Fig. 1). A conditional test is required to determine the contraction status of each pixel. The instructions that precede the test are bit-wise operations and are executed in SIMD mode. The mode-switching capability provided by PASM is used to switch to MIMD mode, perform the conditional test, and then switch back to SIMD mode.

Although mixed-mode operation is used, only two instructions are performed in MIMD, and the rest in SIMD. This allows the loop execution to be performed on the CU in an overlapped fashion, providing an additional degree of parallelization and enhancing the performance of the algorithm.

Each PE "tags" the contracted pixels, counts the number of tagged pixels, and stores the final count in a variable to be used in the expansion stage. The global sum of these counts are examined by the CU in SIMD mode to determine whether to continue processing. The expansion routine is executed in MIMD on each PE where the count is greater than zero. Each contracted pixel must be expanded, based on the most common classification value of its neighboring pixels. A series of conditional statements is performed to obtain the most common value. Thus, MIMD mode is chosen for this part of the phase due to the conditional nature of the operation.

Several iterations of the expansion algorithm may be required to remove all existing void-classified pixels, and inter-PE data transfers are required between each iteration. Because MIMD transfers require expensive synchronization protocols, the transfers are performed in SIMD mode.

For this case, a mixed-mode implementation performs better than MIMD, because in mixed-mode the loop operations and array index calculation are performed on the CU in SIMD mode, and the PEs switch to MIMD to perform the conditional statement (*then* or *else* depending on local data). The mixed-mode implementation is better than pure SIMD due to the inefficient execution of conditional statements (as discussed in Section 3 in the latter mode. Specifically, MIMD took 51% longer than mixed mode and SIMD took 110% longer than mixed mode.

Between the squared error calculation and best offset selection phases of the segmentation algorithm, data must be transferred between neighboring PEs. The optimal mode for data transfers (as discussed in Section 3) is SIMD. However, an advantage of using MIMD transfers here is that the PEs are not forced to synchronize at the end of the squared error calculation phase and may proceed with the best offset selection without paying the synchronization cost of switching from MIMD to SIMD mode. Although this cost may make the phase optimized approach unsuitable for some algorithms [10,23], experimental results have shown that the variability in completion times among PEs after the squared error calculation is not large. Thus, the synchronization cost of switching to SIMD mode is minimal, and the benefit of performing the transfers in SIMD is more significant for this algorithm.

The execution time for this portion of the algorithm (coefficient and squared error calculations plus transfers) is 6% faster than the pure MIMD-with-SIMD-transfers version when executed on a 64-by-64 image with 16 PEs. The same is true for the second data transfer phase and the two mixed-mode phases in the segmentation algorithm. Thus, the entire algorithm is implemented using the phase optimization approach.

Results on a 128-by-128 image are shown in Fig. 9. Because PASM is a partitionable machine, experiments could be conducted on 4-PE and 8-PE mixed-mode submachines, as well as on an all 16 PEs. Results for larger images are expected to be similar. Most phases of the algorithm, with the exception of the surface-type map expansion, can be performed without the collection of global data. Additionally, all data transfers performed in the algorithm can be performed in parallel, i.e., all PEs send and receive data at the same time. As the image size *increases*, the amount of data that must be collected at a global level, and the amount of image data transferred between PEs, relative to the total amount of image data, *decreases*. Thus, the speedup obtained is closer to ideal for larger images, if the number of processors used remains fixed.

With the recent inclusion of pipelined arithmetic units in microprocessors, data-dependent instruction execution times are becoming less prevalent. A mixed-mode parallel machine that incorporates such processors can benefit from this fact that eliminates variable instruction execution times and thus limits the MIMD advantages in SPMD (single program, multiple data) implementations to non-uniform program flow (i.e., conditional statements). Phases that benefit from the variable instruction execution times can now benefit from CU/PE overlap. The phases that

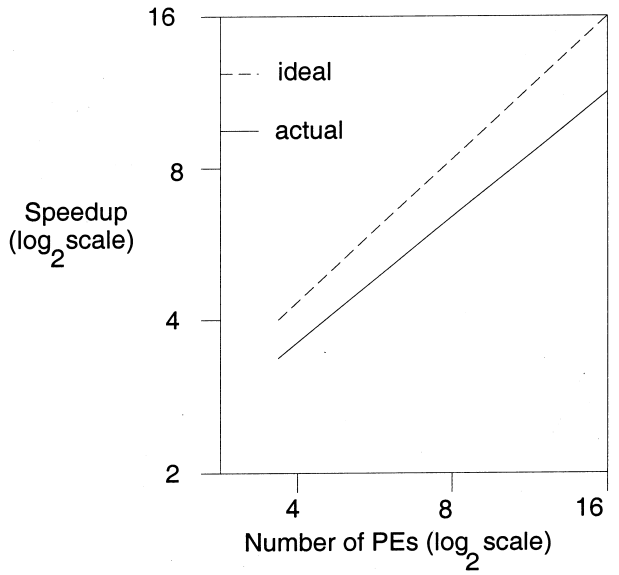

Fig. 9. Obtained speedup for segmentation algorithm ($M = 128$).

incorporate the variable instruction execution times (e.g., smoothing and type selection in the segmentation algorithm) can be implemented in mixed mode to also benefit from CU/PE overlap.

6. Summary

The use of parallel processors to reduce the execution time of image processing tasks such as range image segmentation must be conducted with attention to detail if the full potential of the target parallel system is to be achieved. Mapping the algorithm onto the architecture is not always straightforward. Among other factors, the initial distribution of data, allocation of operations to individual processors, choice of parallel architecture, and choice of computational mode (for the entire algorithm or for individual phases) can impact the overall execution time of the algorithm. In this study, the usefulness of a striped data distribution technique was demonstrated and quantified by examining the trade-offs between minimizing the number of data elements transferred between PEs and minimizing the number of calculations for the given application due to the homogeneous distribution of edge pixels, independent of the mode of parallelism. A new general technique was developed that uses a particular allocation of work to PEs in the machine to reduce the number of network settings by one half for window-based operations. Finally, the trade-offs of different modes of computational parallelism were quantitatively examined for different phases of a range image segmentation algorithm, and the advantages of the mixed-mode approach were demonstrated. The results of this study are useful for both image processing and parallel processing researchers.

Acknowledgements

The authors thank Janet M. Siegel and the referees for their comments. A preliminary version of portions of this study was presented at the Sixth International Parallel Processing Symposium.

References

[1] L.H. Jamieson, Characterizing parallel algorithms, in: L.H. Jamieson, D.B. Gannon, R.J. Douglass (Eds.), The Characteristics of Parallel Algorithms, MIT Press, Cambridge, MA, 1987, pp. 65–100.
[2] C. Reinhart, R. Nevatia, Efficient parallel processing in high level vision, Image Understanding Workshop Sponsored by DARPA, Information Science and Technology Office, September, 1990, pp. 829–839.
[3] M. Proesmans, A. Oosterlinck, Parallel segmentation algorithms, in: I. Pitas (Ed.), Parallel Algorithms for Digital Image Processing, Computer Vision Neural Networks, Chichester, England, Wiley, 1993, pp. 123–146.
[4] L. Sousa, M. Piedade, Low level parallel image processing, in: I. Pitas (Ed.), Parallel Algorithms for Digital Image Processing, Computer Vision Neural Networks, Chichester, England, Wiley, 1993, pp. 25–52.

[5] A.M. Wallace, G.J. Michaelson, P. McAndrew, K.G. Waugh, W.J. Austin, Dynamic control and prototyping of parallel algorithms for intermediate- and high-level vision, Computer 25 (2) (1992) pp. 43–53.
[6] K. Banerjee, P.S. Sastry, K.R. Ramakrishnan, Y.V. Venkatesh, An SIMD machine for low-level vision, Information Sciences 44 (1988) pp. 19–50.
[7] H.J. Siegel, M. Maheswaran, D.W. Watson, J.K. Antonio, M.J. Atallah, Mixed-mode system heterogeneous computing, in: M.M. Eshaghian (Ed.), Heterogeneous Computing, Artech House, Norwood, MA, 1996, pp. 19–65.
[8] M.M. Eshaghian, R. Miller, C. Weems, Multimode system heterogeneous computing, in: M.M. Eshaghian (Ed.), Heterogeneous Computing, Artech House, Norwood, MA, 1996, pp. 67–100.
[9] R.B. Fisher, E. Trucco, M.D. Brown, A.C. Hume, MIMD and SIMD parallel range data segmentation, in: I. Pitas, (Ed.), Parallel Algorithms for Digital Image Processing, Computer Vision Neural Networks, Chichester, England, Wiley, 1993, pp. 147–173.
[10] T.B. Berg, S.D. Kim, H.J. Siegel, Limitations imposed on mixed-mode performance of optimized phases due to temporal juxtaposition, Journal of Parallel and Distributed Computing 13 (2) (1991) pp. 154–169.
[11] D.W. Watson, H.J. Siegel, J.K. Antonio, M.A. Nichols, M.J. Atallah, A block-based mode selection mode for SIMD/SPMD parallel environments, Journal of Parallel and Distributed Computing 21 (3) (1994) pp. 271–288.
[12] H.J. Siegel, T. Schwederski, W.G. Nation, J.B. Armstrong, L. Wang, J.T. Kuehn, R. Gupta, M.D. Allemang, D.G. Meyer, D.W. Watson, The design and prototyping of the PASM reconfigurable parallel processing system, in: A.Y. Zomaya (Ed.), Parallel Computing: Paradigms and Applications, International Thomson Computer Press, London, UK, 1996, pp. 78–114.
[13] H.J. Siegel, T.D. Braun, H.G. Dietz, M.B. Kulaczewski, M. Maheswaran, P. Pero, J.M. Siegel, J.J.E. So, M. Tan, M.D. Theys, L. Wang, The PASM project: A study of reconfigurable parallel computing, Second International Symposium on Parallel Architectures, Algorithms and Networks, (I-SPAN '96), June 1996, pp. 529–536.
[14] R.N. Kapur, U.V. Premkumar, G.J. Lipovski, Organization of the TRAC processor-memory subsystem, in: Proceedings of the AFIPS 1980 National Computer Conference, 1980, pp. 623–629.
[15] M. Auguin F. Boeri, The OPSILA computer, in: M. Consard (Ed.), Parallel Languages and Architectures, Elsevier, Holland 1986, pp. 143–153.
[16] C.G. Herter, T.M. Warschko, W.F. Tichy, M. Philippsen, Triton/1: A massively-parallel mixed-mode computer designed to support high level languages, 1993 Heterogeneous Computing Workshop, 1993, pp. 65–70.
[17] P.M. Kogge, EXECUBE-a new architecture for scalable MPPS, International Conference on Parallel Processing, August 1994, pp. I–77–94.
[18] Integrated Computing Engines Inc. The MeshSP, tech. rep. ICE, Inc., Waltham, MA, July 1995.
[19] B. Sabata, F. Arman, J.K. Aggarwal, Segmentation of 3d range images using pyramidal data structures, CVGIP: Image Understanding 57 (3) (1993) pp. 373–387.
[20] N. Yokoya, M.D. Levine, Range image segmentation based on differential geometry: A hybrid approach, IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-11(6) (1989) pp. 643–649.
[21] P. Besl, R.C. Jain, Segmentation through variable-order surface fitting, IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-11(2) (1988) pp. 167–192.
[22] H.S. Yang, A.C. Kak, Determination of the identity and position and orientation of the topmost object in a pile, Computer Vision Graphics and Image Processing 36 (2/3) (1986) pp. 229–255.
[23] T.B. Berg, H.J. Siegel, Instruction execution trade-offs for SIMD vs MIMD vs mixed-mode parallelism, in: Fifth International Parallel Processing Symposium, May 1991, pp. 301–308.
[24] H.J. Siegel, J.B. Armstrong, D.W. Watson, Mapping computer vision related tasks onto reconfigurable parallel processing systems, Computer 25 (2) (1992) pp. 54–63.
[25] S.D. Kim, M.A. Nichols, H.J. Siegel, Modeling overlapped operation between the control unit and processing elements in an SIMD machine, Journal of Parallel and Distributed Computing 12 (4) (1991) pp. 329–342.

[26] L.M. Ni, P.K. McKinley, A survey of wormhole routing techniques in direct networks, Computer 26 (2) (1993) pp. 62–76.
[27] E.C. Bronson, T.L. Casavant, L.H. Jamieson, Experimental application-driven architecture analysis of an SIMD/MIMD parallel processing system, IEEE Transactions on Parallel and Distributed Systems 1 (2) (1990) pp. 195–205.
[28] M.-C. Wang, W.G. Nation, J.B. Armstrong, H.J. Siegel, S.D. Kim, M.A. Nichols, M. Gherrity, Multiple quadratic forms: A case study in the design of data-parallel algorithms, Journal of Parallel and Distributed Computing 21 (1994) pp. 124–139.
[29] S.A. Fineberg, T.L. Casavant, H.J. Siegel, Experimental analysis of a mixed-mode parallel architecture using bitonic sequence sorting, Journal of Parallel and Distributed Computing 11 (3) (1991) pp. 239–251.