A Parallel Approach to Hybrid Range Image Segmentation

Nicholas Giolmas Daniel W. Watson David M. Chelberg Howard Jay Siegel

Parallel Processing Laboratory, School of Electrical Engineering
Purdue University, West Lafayette, IN 47907-1285 USA

Abstract

Parallel processing methods are an attractive means to achieve significant speedup of computationally expensive image understanding algorithms, such as those applied to range images. Mixed-mode parallel systems are ideally suited to this area because of the flexibility in using the different modes of parallelism. The trade-offs of using different parallel modes are examined through the implementation of hybrid range segmentation operations, characteristic of a broad class of low level image processing algorithms. Alternative means of distributing data among the processing elements that achieve improved performance are considered. Results comparing different implementations on a single reconfigurable parallel processor, PASM, indicate some generally applicable guidelines for the effective parallelization of vision algorithms.

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 and mapping the algorithm onto that particular parallel architecture [9,11]. In this paper, implementation approaches are presented that are applicable to many low level image understanding algorithms on a variety of parallel architectures. Mixed-mode parallel systems are well suited to the implementation of low level image understanding algorithms because of the flexibility in using the different modes of parallelism. The trade-offs of using different parallel modes and different data distribution schemes are examined through the implementation of a particular range segmentation algorithm. This algorithm is characteristic of a broad class of image understanding algorithms.

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 segmentation is the 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 [9]. Selection of the parallel architecture that is best suited to the algorithm is a critical step in the algorithm mapping process. Furthermore, the optimal implementation for different portions of an algorithm may require using different parallel architectures. This work adopts a phase optimized approach for the entire algorithm, although this method is not guaranteed to produce the optimal implementation [4].

The PASM (partitionable SIMD/MIMD) parallel processing system [13], designed at Purdue, is being used to study the application of parallel processing to image processing algorithms. PASM is a multiprocessor system that is capable of mixed-mode parallelism, i.e., it can operate in either the SIMD or

This research was supported by the Naval Ocean Systems Center under the High Performance Computing Block, ONT, by the Office of Naval Research under grant number N00014-90-J-1937, by the National Science Foundation under grant number CDA-9015696, and by the Digital Equipment Corporation Incentives for Excellence Grant.

MIMD mode of parallelism, and can dynamically switch between modes at instruction level granularity. A 30-processor small-scale prototype (16 processors in the computational engine) has been built and is being used as a testbed for application studies.

Background information about range image processing and parallel processing issues is included in Sections 2 and 3, respectively. Section 4 examines the distribution of data among the PEs. Section 5 compares different parallel implementations and provides results obtained from the study. A summary and concluding remarks are given in Section 6.

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. Alternatively, an edge-based approach detects discontinuities in depth values and in surface orientations. The algorithm study in Sections 4 and 5 examines the parallel implementation of a hybrid approach [15] to the problem of range image segmentation, which is a combination of region-based and edge-based approaches.

There are several motivations for the choice of this algorithm. First, the algorithm is characteristic of many image processing algorithms, so that results for this study are applicable to a broad class of image processing algorithms. Additionally, the hybrid algorithm employs both region-based and edge-based segmentation methodologies, and the combining portion of the algorithm is representative of other kinds of algorithms (e.g., connected-component labeling, contracting, and expanding algorithms). 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 call for different parallel methodologies.

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 fitting algorithms, such as [3], which uses low-order bivariate polynomials, and [14], 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 algorithms, because the computations involved in each stage of the algorithm are common to many segmentation and image processing algorithms.

The algorithm consists of three major stages (Figure 1). In the first stage, differential geometric properties of a surface (e.g., surface normal, Gaussian curvature, and mean curvature) are locally estimated. Object surfaces are locally approximated using second-order bivariate polynomials. In the first 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 that provides a minimum fitting error, referred to as E^2 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. 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.

In the second stage, three types of initial segmentations are computed. A region-based segmentation is obtained in the form of the surface type (KH) map. Surface points are classified according to the sign of K and H into one of eight possible surface primitives [3]. To eliminate small surface regions, that are typically due to noise, the surface type map is contracted and expanded once. The resulting map is called a refined KH sign map. 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 refined map can then be used in higher level image understanding algorithms.

3. Models of parallelism

A partitionable-SIMD/MIMD machine can operate as one or more independent or cooperating submachines, where each submachine may operate in either the SIMD or MIMD mode of parallelism. TRAC [10] and PASM [13] are examples of partitionable-SIMD/MIMD systems that have been prototyped. Each PASM submachine can switch modes at instruction level granularity with negligible overhead, independent of the other submachines. As previously stated, the use of both modes of parallelism to implement an algorithm is referred to as mixed-mode parallelism. Each segment of a parallel algorithm is examined to determine the mode in which it should be executed. Consequently, the programmer of a mixed-mode system must be aware of the trade-offs between executing in SIMD mode and MIMD mode.

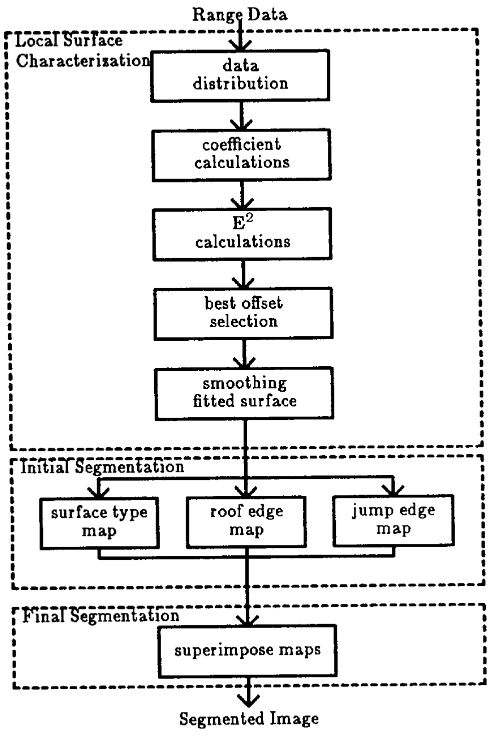


Figure 1: Range image segmentation algorithm flow diagram.

A limited mixed-mode machine is OPSILA [1]. OPSILA is an existing system and can switch between the SIMD and SPMD (single program - multiple data stream) modes of parallelism. SPMD mode is a form of MIMD mode where all processing elements (PEs) independently execute the same program. The mapping of tasks onto OPSILA is presented in [2,7]. Unlike PASM and TRAC, OPSILA is not a partitionable machine.

There are trade-offs that exist between the SIMD and MIMD modes of parallelism that explain why some sequences of instructions are better performed in one mode than in the other [5]. Some of the advantages and disadvantages of each mode, as they apply to image processing algorithms, are discussed 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. Variable-time instructions execute more efficiently in MIMD mode than in SIMD mode. In SIMD mode, the control unit (CU) broadcasts the next instructions to the PEs only after they have all completed the current instruction. Therefore, each instruction takes as long as it takes the PE that executes it most slowly. 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 $\max_P(\sum_i T_i^P)$ (Figure 2). Because $\max_P(\sum_i T_i^P) \leq \sum_i \max_P(T_i^P)$, 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 instructions 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 [12].

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 enabled. C is then executed. This serializes the execution of B and C. Conversely, in MIMD mode those PEs where A is true can execute B while the other PEs execute C. In MIMD mode, the maximum time to execute the if-then-else 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 becomes apparent when inter-PE data transfers 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 must execute explicit synchronization and identification protocols for each inter-PE transfer. While the details of the inter-PE transfer protocol 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, the CU can be used to overlap operations with PEs. 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 array address calculations, can be performed in the CU while the PEs are performing other computations. In MIMD mode, CU/PE overlap does not occur, and the PEs must perform all of the instructions.

The "max of sums"/"sum of maxs" property is not limited to single instructions. 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 [5] (e.g., MIMD instructions in an SIMD loop).

From the discussion above it is evident that there are many trade-offs between operating in SIMD mode and operating in MIMD mode. Although it is often clear in which mode a sequence of instructions should be implemented, 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 operation is not straightforward; i.e., 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.

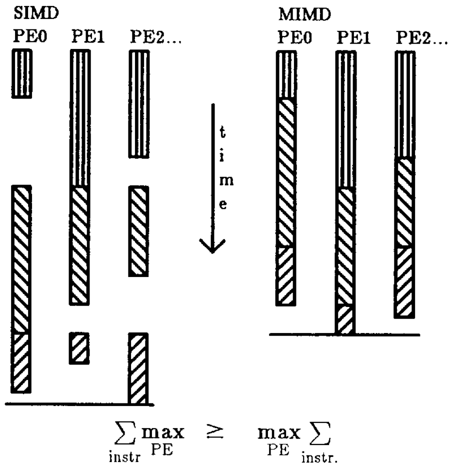
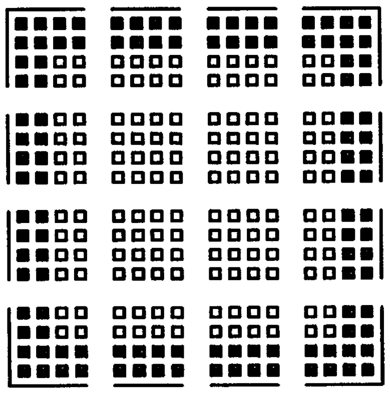


Figure 2: Execution of variable-time instructions in SIMD and MIMD mode.

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 conventionally 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 is presented, based on distributing consecutive rows rather than square subimages of data to the PEs, that minimizes the number of calculations.



■ - border pixel

Figure 3: Square distribution example, M=16, N=16, w=5.

Many calculations in segmentation algorithms involve $w \times w$ windows of data (e.g., convolutions, local minimum). 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 $w \times w$ 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 be performed for the border pixels, thereby decreasing the overall number of required calculations.

Let the original image be of size $M \times M$, the number of PEs on the target machine be N, and the window size for the calculations be $w \times w$. In the following discussion, it is assumed that $M \geq N$, and typically $N \geq 64$, $M \geq 128$, and $w > 3$. For simplicity it is also assumed that M is a multiple of N, although the obtained results can be adapted to cases where this restriction does not apply. The first two stages of the algorithm, the coefficient and E^2 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.

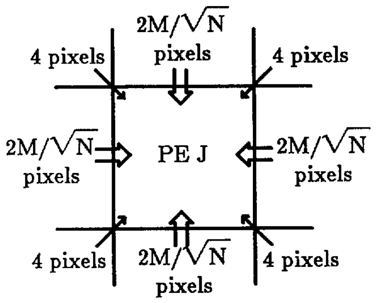


Figure 4: Pixel transfers for $w = 5$.

Conventionally, image processing algorithms that rely heavily on inter-PE transfers have used the square subimage distribution method as shown in Figure 3. This method distributes a unique square portion of size $(M / \sqrt{N}) \times (M / \sqrt{N})$ 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 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 (Figure 4). Each PE must share data with a maximum of eight neighboring PEs. A minimum of four network settings are required to perform a transfer operation using this method (e.g., in Figure 4, the data elements to be sent from PEs catercorner to 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 receive 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 Figure 5, each PE initially receives M/N rows of data. Each PE transfers data to its neighboring PEs. The 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). The number of data elements transferred is increased due to the wider perimeter. 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 border pixels for each PE are not performed. When the transfers are complete, each PE will calculate the six coefficient values and the E^2 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.

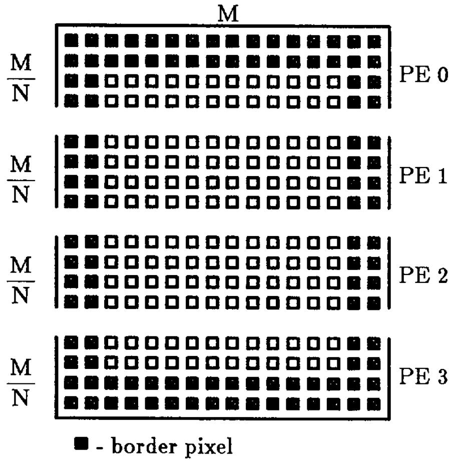


Figure 5: Striped distribution example, M=16, N=4, w=5.

Let T_t denote the time required to transfer one data element, and T_o denote the time required to perform an operation, which may consist of many calculations, for one pixel. The operation transfer ratio, is defined as $\rho = T_o / T_t$. Experimental results on PASM have shown that $\rho \simeq$ 2,500 for these segmentation operations, (i.e., calculating the six coefficient values and E^2 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

$$T_p = 2\lfloor \frac{w}{2} \rfloor \times (\frac{M}{N}) \times T_o$$

If network setups are time consuming, the time required by the square method for the extra two setups should be included in T_p. The time penalty, T_h, for performing the extra transfers by using the horizontal stripe method versus the square method is

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

Let $T_{extra} = T_p - T_h$, or the extra time required by the square method to complete this stage of the algorithm.

The value of ρ depends on the particular algorithm. 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 $\rho_{breakeven}$ designate the value of ρ such that if $\rho > \rho_{breakeven}$, then the striped distribution method should be chosen over the square distribution method. Figure 6 shows the value of ρ as a function of the number of PEs (N) for which the stripe method should be chosen, given an image size of 1024 and window size of 5. Consider a target system with 256 PEs and an image size of 1024. If ρ is greater than 223, then the stripe distribution method should be used to implement this algorithm.

In conclusion, 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 are not to be performed, and transfers are relatively fast compared to calculations. When network setups are time consuming there is further gain to using the stripe method.

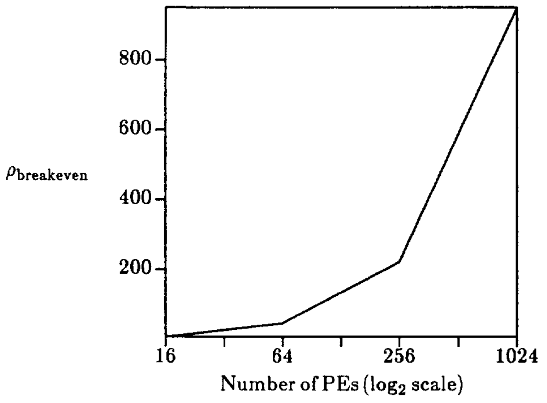


Figure 6: $\rho_{breakeven}$ vs. N for M=1024.

5. Parallel implementation

The serial algorithms that compute the stages of the overall segmentation algorithm were mapped onto the target parallel machine. Theoretical analysis and experimental results were used to determine the best mapping for each stage. 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 include [4,6,8].

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

(a) Variable instruction execution times.

(b) Non-uniform program flow (includes *if – then – else* statements).

(c) CU/PE overlap.

(d) Synchronization advantage.

While (c) and (d) favor SIMD mode, (a) and (b) favor MIMD mode. 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. 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 (c) 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. These counteracting trade-offs must be quantified.

Table 1 lists the various stages 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 further illustrate the mode selection, detailed experimental results are presented in Figures 7, 8, and 9 for the coefficient calculation, surface smoothing and selection, and contraction stages, respectively, using 16 PEs and a 64×64 image. Processing time for all but the expansion stage is primarily input image independent.

The coefficient calculation involves convolution operations between six predetermined 5×5 operators and the window of range data centered at each pixel(x,y). The floating point data calculations introduce variable instruction execution times on different PEs (a) and thus MIMD is chosen as the optimal mode (Figure 7). 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 "sum of maxs" rule dictates.

Stage	Optimal Mode	Reason
Coefficient calculation	MIMD	a
E^2 calculation	MIMD	a
Inter-PE data transfers	SIMD	d
Best offset selection	MIMD	b
Smoothing & type selection	MIMD	a,b
Inter-PE data transfers	SIMD	d
Contraction of surface map	mixed-mode	b,c
Expansion of surface map	MIMD	b
Boundary pixel selection	mixed-mode	b,c
Roof & jump edge selection/ map superimposing	MIMD	a,b

Table 1: Optimal parallel mode selection for each stage of the segmentation algorithm.

The surface smoothing and type selection algorithm is also iterative and includes both floating point calculations and conditional statements. Four implementations were considered. In the first, the entire algorithm was mapped to MIMD mode. In the second implementation (labeled "mixed-mode-Loop" in Figure 8) the outer loop was performed in SIMD on the CU and the contents of the loop were performed on the PEs in MIMD mode. In the third implementation (labeled "mixed-mode-Cond in the Figure 8) the outer loop was performed in SIMD mode on the CU and floating point calculations were also performed in SIMD on the PEs. Only the conditional statements were performed in MIMD on the PEs. These attempts 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. The pure SIMD version suffered from both inefficient conditional statement execution and the "sum of maxs" effect on the data dependent execution times.

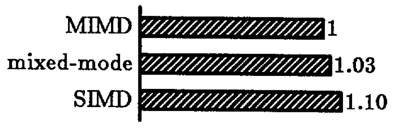


Figure 7: Normalized execution times for coefficient calculation.

The contraction routine requires a conditional statement to determine the contraction status of each pixel. This is best performed in MIMD (b). The rest of the instructions have constant execution times. Mixed-mode performs better than MIMD in this case (Figure 9) because the CU can execute the loop operations and some required array index calculations in overlapped fashion with the PEs. The PEs execute the constant time instructions in SIMD, then switch to MIMD to perform the conditional statement (*then* or *else* depending on local data) and return to SIMD mode for the rest of the loop instructions. The mixed-mode implementation is better than pure SIMD due to the inefficient execution of conditional statements (as discussed in Section 3) in the later mode.

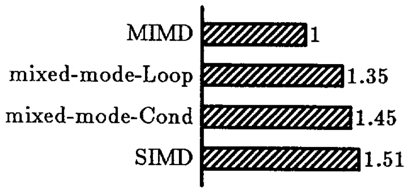


Figure 8: Normalized execution times for surface smoothing and type selection algorithm.

Between the E^2 calculation and best offset selection stages 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 E^2 calculation stage 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 [4,5], experimental results have shown that the variability in completion times among PEs after the E^2 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.

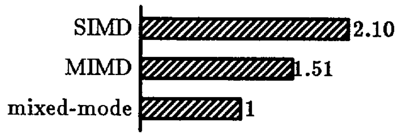


Figure 9: Normalized execution times for contraction algorithm.

The execution time for this portion of the algorithm (coefficient and E^2 calculations plus transfers) is 6% faster than the pure MIMD version when executed on a 64×64 image with 16 PEs, as shown in Figure 10. The same is true for the second data transfer phase and the two mixed-mode stages in the segmentation algorithm. Thus, the entire algorithm is implemented using the phase optimization approach. Figure 11 shows the obtained speed-up from executing the segmentation algorithm on the PASM prototype, using the phase optimized parallel implementation, for various numbers of processors and an image of size 128×128.

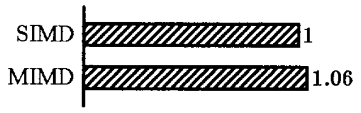


Figure 10: Normalized execution times for coefficient and E^2 calculations plus data transfers.

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 (a) and thus limits the MIMD advantages in SPMD implementations to (b). Stages that benefit from (a) can now benefit from (c) and stages that incorporate (a) (e.g., surface smoothing and type selection in the segmentation algorithm) can be implemented in mixed-mode to also benefit from (c).

6. Summary

Parallel processing methods are an attractive means to achieve significant speedup of

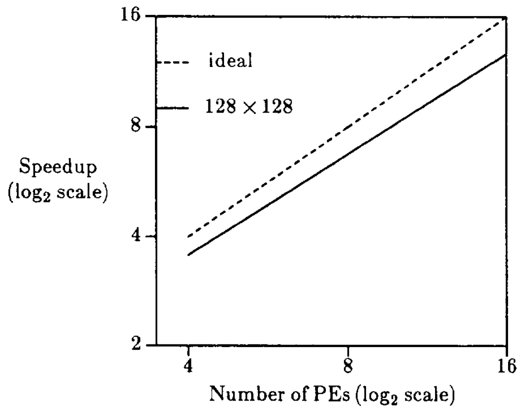


Figure 11: Obtained speedup for segmentation algorithm (M=128).

computationally expensive image understanding algorithms. Through the study of a characteristic range image segmentation algorithm the trade-offs of different modes of parallelism were examined. An alternative method for distributing data among PEs that achieves a reduction in execution time was presented. The use of mixed-mode parallelism has a distinct advantage over static architectures in the implementation of parallel image understanding algorithms. Given equivalent processing power, a mixed-mode system is capable of achieving greater performance when the tasks to be performed benefit from the use of more than one mode of parallelism. The study of a hybrid segmentation algorithm showed that a variety of different parallel modes would be chosen if each computational task to be performed is considered in isolation. Given a mixed-mode system, these tasks can be implemented in their preferred mode, in contrast to a static architecture that would be less effective at performing some of these tasks.

Thus, the results of this study are useful for both image processing and parallel processing researchers. It demonstrated and quantified the usefulness of the striped data distribution technique and showed how mixed-mode parallelism can be exploited for this type of image processing task.

Acknowledgment: The authors wish to acknowledge Juneho Yi for his useful comments.

References

[1] M. Auguin and F. Boeri, "The OPSILA computer," in *Parallel Languages and Architectures*, M. Consard, ed., Elsevier Science, Holland, 1986, pp. 143-153.

[2] M. Auguin and F. Boeri, "Experiments on a parallel SIMD/SPMD architecture and its programming," *France-Japan Artificial Intelligence and Computer Science Symp. 87*, November 1987, pp. 385-411.

[3] P. Besl and R. C. Jain, "Segmentation through variable-order surface fitting," *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. PAMI-11, No. 2, Mar. 1988, pp. 167-192.

[4] T. B. Berg, S. D. Kim, and H. J. Siegel, "Limitations imposed on mixed-mode performance of optimized phases due to temporal juxtaposition," *J. Parallel and Distributed Computing*, Vol. 13, No. 2, Oct. 1991, pp. 154-169.

[5] T. B. Berg, and H. J. Siegel, "Instruction execution trade-offs for SIMD vs. MIMD vs. mixed-mode parallelism," *5th Int'l Parallel Processing Symp.*, May 1991, pp. 301-308.

[6] E. C. Bronson, T. L. Casavant, and L. H. Jamieson, "Experimental application-driven architecture analysis of an SIMD/MIMD parallel processing system," *IEEE Trans. Parallel and Distributed Systems*, Vol. 1, No. 2, Apr. 1990, pp. 195-205.

[7] P. Dulcos, F. Boeri, M. Auguin, and G. Giraudon, "Image processing on a SIMD/SPMD architecture: OPSILA," *9th Int'l Conf. on Pattern Recognition*, Nov. 1988, pp. 430-433.

[8] S. A. Fineberg, T. L. Casavant, and H. J. Siegel, "Experimental analysis of a mixed-mode parallel architecture using bitonic sequence sorting," *J. Parallel and Distributed Computing*, Vol. 11, No. 3, Mar. 1991, pp. 239-251.

[9] L. H. Jamieson, "Characterizing parallel algorithms," in *The Characteristics of Parallel Algorithms*, L. H. Jamieson, D. B. Gannon, and R. J. Douglass, eds., MIT Press, Cambridge, MA, 1987, pp. 65-100.

[10] G. J. Lipovski, M. Malek, "Parallel Computing: Theory and Comparisons," John Wiley & Sons, New York, NY, 1987.

[11] C. Reinhart and R. Nevatia, "Efficient parallel processing in high level vision," *DARPA Image Understanding Workshop*, Sept. 1990, pp. 829-839.

[12] H. J. Siegel, J. B. Armstrong, and D. W. Watson, "Mapping computer vision related tasks onto reconfigurable parallel processing systems," to appear in *Computer*, Feb. 1992.

[13] H. J. Siegel, T. Schwederski, J. T. Kuehn, and N. J. Davis IV, "An overview of the PASM parallel processing system," in *Computer Architecture*, D. D. Gajski, V. M. Milutinovic, H. J. Siegel, and B. P. Furht, eds., IEEE Computer Society Press, Washington, DC, 1987, pp. 387-407.

[14] H. S. Yang and A. C. Kak, "Determination of the identity and position and orientation of the topmost object in a pile," *Computer Vision, Graphics, and Image Processing*, Vol. 36, Dec. 1986, pp. 229-255.

[15] N. Yokoya and M. D. Levine, "Range image segmentation based on differential geometry: a hybrid approach," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. PAMI-11, No. 6, June 1989, pp. 643-649.