1994

A Data Parallel Algorithm for Solving the Region Growing Problem on the Connection Machine

Nawal Copty
Syracuse University, School of Computer and Information Science

Sanjay Ranka
Syracuse University, School of Computer and Information Science

Geoffrey C. Fox
Syracuse University, School of Computer and Information Science

Ravi V. Shankar
Syracuse University, School of Computer and Information Science

A Data Parallel Algorithm for Solving the Region Growing Problem on the Connection Machine [1]

Nawal Copty Sanjay Ranka Geoffrey Fox
Ravi V. Shankar
School of Computer and Information Science
Syracuse University
Syracuse, NY 13244

Published in:

Journal of Parallel and Distributed Computing, Special Issue on Data Parallel Languages and Programming **21**, 1
(April 1994), pp. 160-168

[1]This work was supported in part by NSF under CCR-9110812 and DARPA under contract # DABT63-91-C-0028. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

Abstract

Region growing is a general technique for image segmentation, where image characteristics are used to group adjacent pixels together to form regions. This paper presents a parallel algorithm for solving the region growing problem based on the split and merge approach, and uses it to test and compare various parallel architectures and programming models. The implementations were done on the Connection Machine, models CM-2 and CM-5, in the data parallel and message passing programming models. Randomization was introduced in breaking ties during merging to increase the degree of parallelism, and only one and two-dimensional arrays of data were used in the implementations.

Keywords: Region growing, Split and merge, Parallel processing, Data parallelism, Message passing, and Connection machine.

1 The Region Growing Problem

Region growing is a general technique for image segmentation. Image characteristics are used to group adjacent pixels together to form regions. Regions are merged with other regions to *grow* larger regions. A region might correspond to a world object or a meaningful part of one [2].

The merging of pixels or regions to form larger regions is usually governed by a **homogeneity criterion** that must be satisfied. A variety of homogeneity criteria have been investigated for region growing. If $f(x, y)$ is the image intensity at the pixel with coordinate (x, y), then the **pixel range** homogeneity criterion for a region R is *true* whenever $\|f(x_1, y_1) - f(x_2, y_2)\| < T$ for all point pairs (x_1, y_1) and (x_2, y_2) in R, and *false* otherwise. This particular homogeneity criterion requires that the range between the minimum and maximum intensities within a region R not exceed a threshold value T.

There are many algorithms for solving the region growing problem [1, 2, 6, 7, 10, 15]. The effectiveness of a particular algorithm depends on the application area and the input image. In this paper, we present a parallel algorithm for solving the region growing problem based on the split and merge approach proposed by Horowitz and Pavlidis [8].

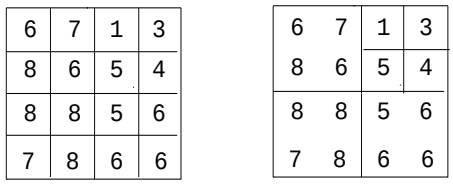
While previous parallel implementations [13, 14] of the split and merge approach have used dynamic or tree structures to represent the regions in the image, our implementations use only one and two-dimensional arrays. We also introduce an element of randomness to the algorithm whenever a tie occurs during merging; this has significantly reduced the execution time. A component labeling algorithm proposed by Hambrusch et al [6] defines as initial regions adjacent pixels that have the same intensity value, instead of using a split stage. Moreover, Hambrusch et al use extra selection criteria that improve the quality of the solution for some input images and reduce the possibility of a tie during merging.

2 The Split and Merge Approach

The split and merge approach solves the region growing problem in two stages: the split stage and the merge stage. The split stage is a preprocessing stage that aims to reduce the number of merge steps required to solve the problem.

2.1 The Split Stage

In the **split stage**, an $N \times N$ image is partitioned into square regions which conform to the homogeneity criterion. At first, each pixel is considered a homogeneous square region of size 1×1. Then every group of four adjacent pixels are tested for homogeneity. If the homogeneity criterion is satisfied, the pixels are combined into one larger square region of size 2×2, and so on... The split stage terminates when the whole image is one square region of size $N \times N$, or when no more square regions can be merged. Figure 1 shows the square regions produced by the split stage for a 4×4 image, where the threshold value $T = 3$. The numbers in the image represent pixel intensities.



Square regions: (a) at start of the split stage; (b) after first and final split iteration

Figure 1: The Split Stage

2.2 The Merge Stage

In the **merge stage** of the split and merge approach, the square regions determined by the split stage are iteratively merged into larger and larger regions which conform to the homogeneity criterion. The merge continues until no more merges are possible.

The merge is achieved by reformulating the region growing problem as a weighted, undirected graph problem, where the vertices of the graph represent the regions in the image, and the edges represent the neighboring relationships between these regions. That is, an edge e exists between two vertices v and w of the graph, if and only if the regions represented by v and w share a common boundary. The weight of the edge e is the difference between the maximum and minimum pixel intensities in the union of the two regions represented by v and w.

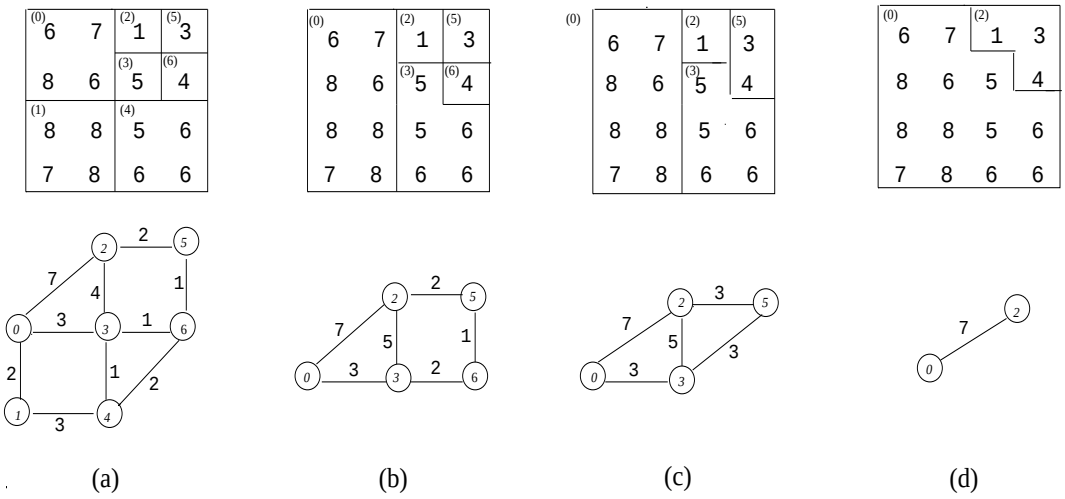
Obviously, only vertices connected by edges satisfying the homogeneity criterion can be merged. In one merge iteration, each region selects for merging the neighbor that *best satisfies* the homogeneity criterion, namely the neighbor connected to it by the edge of least weight. This "best merge" approach yields better results by minimizing the increase in range with each merge [14]. A tie between two or more neighbors may be broken by selecting the neighbor with the smallest (largest) ID, or by using some other criteria.

Two regions *actually merge* if they *select each other* for merging. Once two regions merge, the region with the smaller ID becomes the representative of the two, and the vertices and edges of the graph are updated. The merge stage terminates when no more edges satisfying the homogeneity criterion exist in the graph.

Figure 2 shows the different regions obtained and their corresponding graphs in each iteration of the merge stage, for the 4×4 image of Figure 1. Ties are broken by selecting the neighbor with the smallest ID. The small numbers in parenthesis in the corners of the regions denote the region IDs.

Resolving Ties at Random: The region growing problem is a representative of a type of loosely synchronous problems, known as adaptive irregular problems, whose data objects evolve during the computation in a time synchronized manner [5]. The problem exhibits a dynamic behavior that starts with a high degree of parallelism that very rapidly diminishes to a much lower degree of parallelism.

In order to increase the degree of parallelism in the algorithm, we introduced an element of randomness to our parallel implementations. Whenever a tie occurs during the merging of regions, the tie is broken by selecting one of the tied neighbors *at random* instead of selecting



Regions: (a) at start of the merge stage; (b) after first merge iteration; (c) after second merge iteration; (d) after third and final merge iteration

Figure 2: The Merge Stage When Ties are Broken by Choosing Neighbor With Smallest ID

the neighbor with the smallest (largest) ID. In Figure 2(a), both regions 3 and 5 tie for merging with region 6, as they best satisfy the homogeneity criterion for region 6. Region 6 chooses to merge with region 3, since ties in Figure 2 are broken by choosing the neighbor with the smallest ID. However, no merge actually takes place, since region 3 chooses to merge with region 4. If, instead, ties were broken *at random*, then, in the first merge iteration, the three region pairs: 0 and 1, 3 and 4, 5 and 6 could merge at the same time, and the merge stage could take 2 iterations instead of 3.

Experimentally, the random approach in breaking ties proved to be significantly faster than the approach of selecting the neighbor with the smallest (largest) ID, as shown in Table I. This is due to the fact that the random approach generally results in a larger number of merges per merge iteration, while the approach of selecting the neighbor with the smallest (largest) ID imposes a serialization on the order of merges.

3 The Parallel Implementations

The region growing problem was implemented on two distinct models of the Connection Machine: the CM-2 and CM-5.

The CM-2 is a massively parallel computer that belongs to the range of SIMD (Single Instruction Multiple Data) machines. The CM-2 operates under the programmed control of a front end computer that provides the program development and execution environment. All CM-2 programs execute on the front end; during the course of the execution, the front end issues instructions to the CM-2 processors. The CM-2 supports the data parallel model of programming, and provides the CM Fortran language which is essentially standard Fortran 77 supplemented with the array processing extensions of Fortran 90.

The CM-5, on the other hand, is an MIMD machine composed of a control processor and

tens or hundreds of node processors connected together in the form of a *fat tree* [9]. Every node processor is a general-purpose computer that can fetch and interpret its own instruction stream, execute arithmetic and logical instructions, calculate memory addresses, and perform interprocessor communication. The CM-5 supports both the data parallel and message passing models of programming. For the data parallel model, the CM-5 provides the CM Fortran language. For the message passing model, the CM-5 provides the CMMD library, which is a collection of routines that permit cooperative message passing among the node processors. CMMD supports a version of message passing known as *host/node* programming, where a *host* program runs on the control processor, and independent copies of a *node* program run on each of the node processors.

3.1 The Data Parallel Implementation

In the data parallel model of execution, the same CM Fortran program can be executed on both the CM-2 and the CM-5 without modification. The data parallel implementation of the split and merge region growing algorithm consists of the following steps:

1. The two-dimensional pixel image is repeatedly split into homogeneous square regions. The split stage stops when the whole image is one homogeneous square region, or when no more merges are possible.

2. For each square region in the pixel image, a corresponding graph vertex is created, and for each pair of neighboring square regions, an edge is created. Edges that do not satisfy the homogeneity criterion are de-activated.

3. A region determines its neighboring region that best satisfies the homogeneity criterion. In the case of a tie, the region chooses one of the tied neighboring regions *at random*. Two regions merge if their merge choices are mutual. In one merge iteration, several region pairs can merge at the same time without conflicting with each other.

4. The vertices and edges of the graph are updated to reflect the new regions in the image. Edges that do not satisfy the homogeneity criterion are de-activated.

5. If there still exist any active edges, then steps 3 and 4 are repeated. Otherwise, the program terminates.

3.2 The Message Passing Implementation

In contrast to the data parallel model of execution, the message passing model requires the programmer to explicitly specify the detailed behavior of individual processors operating asynchronously. The message passing implementation of the split and merge algorithm is a *hand-coded translation* of the data parallel one. It consists of the following steps:

0. The pixel image is partitioned equally among the node processors. Given a pixel image of size $N \times N$ and $P1 \times P2$ node processors, the pixel image is mapped to the processor grid such that each processor receives an $\frac{N}{P1} \times \frac{N}{P2}$ sub-image of the original image.

1. Each node processor independently splits its $\frac{N}{P1} \times \frac{N}{P2}$ sub-image and determines the homogeneous square regions within it. If the sub-image within a processor is rectangular in shape, it is divided into square sections and the split stage is applied independently to each of these sections in turn.

2. Each node processor sets up the vertices and edges of the graph associated with its sub-image. Boundary information is exchanged so that edges connected to vertices in other processors are created.

3. The node processors cooperate to merge the regions determined so far in the image.

4. The node processors cooperate to update the vertices and edges of their graphs.

5. If there still exist any active edges in any of the node processors, then steps 3 and 4 are repeated. Otherwise, the host and node programs terminate.

Irregular Communication: At several points in the message passing implementation, irregular communication is required, where each of the node processors sends zero or more messages to other processors in an irregular fashion. An efficient communication scheme is needed whereby messages are sent and received without causing deadlock.

Two different communication schemes were investigated. The first, called **Linear Permutation (LP)** [12], uses synchronous (blocked) message passing. In this scheme, each processor obtains a copy of the communication matrix, using a global concatenation operation. Then, in step i, $0 < i < Q$, processor p_k sends a message to processor $p_{(k+i) \ MOD \ Q}$ and receives a message from processor $p_{(k-i) \ MOD \ Q}$, where Q is the total number of node processors. The sender and receiver processors are blocked until the message is transmitted. The steps of the Linear Permutation algorithm are as follows:

For all processors p_k, $0 \leq k \leq Q-1$, *in parallel do*
for $i = 1$ *to* $Q-1$ *do*
  Processor p_k sends a message to processor $p_{(k+i) \ MOD \ Q}$
  Processor p_k receives a message from processor $p_{(k-i) \ MOD \ Q}$
endfor

The second communication scheme uses **asynchronous** message passing. In this scheme, a processor that wishes to send or receive a message does not block while waiting for its partner. A processor announces its intention to send or receive a message, and then pursues other computation until the message is ready to be sent and received. When both the sender and receiver are ready, the system interrupts whatever else is happening on the processors and the message is transmitted. The steps of the asynchronous communication algorithm are as follows:

1. Using a global reduction operation, each processor determines the number of messages it must receive from the other processors.

2. Every processor sends, asynchronously, all the messages it wishes to send to other processors.

3. Every processor receives the required number of messages.

In order to reduce the communication overhead in both schemes, whenever a processor needs to send more than one message to the same destination, all the messages are concatenated together and sent as one large message.

3.3 Data Structures

In implementing the split and merge algorithm for solving the region growing problem, no sophisticated data structures were needed to solve the problem. Two-dimensional arrays were used to store the intensities as well as other information pertaining to the pixels, such as whether a pixel is a region representative or not. One-dimensional arrays were used to store information about the vertices and edges of the graph modeling the problem.

To illustrate the way in which data is stored in the various arrays, consider Figure 2(a) which shows the regions in the image at the start of the merge stage where the threshold value $T = 3$. Information on vertices corresponding to these regions is stored in one-dimensional arrays, as follows:

Region ID:	0	1	2	3	4	5	6
Min. pixel value:	6	7	1	5	5	3	4
Max. pixel value:	8	8	1	5	6	3	4

Information on edges is stored in one-dimensional arrays, as follows:

ID of first region of Edge:	0	0	0	1	2	2	3	3	4	5
ID of second region of Edge:	1	2	3	4	3	5	4	6	6	6
Min. pixel value in union of 2 regions:	6	1	5	5	1	1	5	4	4	3
Max. pixel value in union of 2 regions:	8	8	8	8	5	3	6	5	6	4
Edge active?	Yes	No	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes

4 Complexity

Given an $N \times N$ pixel image, the complexity of the parallel split and merge algorithm depends on the number of processors used and the number of iterations required to find the regions in the image. The number of iterations in turn depends on the shape and size of those regions.

4.1 The Split Stage

In the best case, when every pixel is a region by itself, only one split iteration is required. In the worst case, when the whole image is one homogeneous square region, $log(N)$ split iterations are required.

CM-2 Implementation: Suppose that P processors are used by the data parallel implementation on the CM-2, and P is smaller than N^2. At the beginning of the split stage, each pixel

is considered a square region and the first split iteration can be done in $\frac{N^2}{P}$ steps. In the second split iteration, there are $O(\frac{N^2}{4})$ square regions and this iteration can be done in $O(\frac{N^2}{4\times P})$ steps, and so on, until the number of square regions becomes $\leq P$. When this occurs, each iteration can be done in one step and there will be at most $log(P)$ of these iterations. So, the complexity of the split stage in the data parallel implementation on the CM-2 is given by $O(\frac{N^2}{P} + logP)$.

CM-5 Implementations: In both the data parallel and message passing implementations on the CM-5, the first $log\frac{N^2}{P}$ split iterations are done locally, while the last $logP$ iterations require communication. Assuming that communication in each of the last $logP$ split iterations requires $O(\tau)$ time units, where τ is the setup time, then the total time for the split stage is $O(\frac{N^2}{P} + (\tau \times logP))$. If the split stage is stopped after $log\frac{N^2}{P}$ iterations, then the time is $O(\frac{N^2}{P})$.

4.2 The Merge Stage

The number of iterations needed to complete the merge stage of the algorithm is upper bounded by the maximum number of sub-regions that must be merged to connect any single region in the image. If a region consists of r sub-regions, then it will require at least $log(r)$ merge iterations. In the worst case, when only one pair of regions is merged in each iteration, it will require $r-1$ merge iterations.

The total time for the merge stage depends on the number of regions in the image at the beginning and at the end of the merge stage. Let R_i and R_f denote these two numbers, respectively. Suppose that the number of regions is reduced by a factor of k at every step in the merge stage ($1 \leq k \leq 2$). Then the number of iterations required is $log_k\frac{R_i}{R_f}$. The exact value of k depends on the input image and the approach used in resolving ties. As the timings in Table I show, the random tie breaking approach generally results in a greater value of k than the smallest (largest) ID approach.

The number of edges, E, and the number of regions, R_i, at the beginning of the merge stage can be derived by Euler's formula [4]: $V + R_i - E = 2$, where V is the total number of corners of the square regions. Since $E = V + R_i - 2$ and $V \leq 4 \times R_i$, then $R_i \leq E \leq 5 \times R_i$. Thus, the number of edges is linearly proportional to the number of regions.

CM-2 Implementation: Suppose that P processors are used by the data parallel implementation on the CM-2. Then the total time required for any step of the merge stage in which E edges are active is $\frac{E}{P}\times$ (Cost of a Random Access Write + Cost of a Random Access Read).

The time taken by a Random Access Read and a Random Access Write of B data elements on a P-processor hypercube is $O(logP)$ if $B \leq P$, and $O(\frac{B\times logB}{P})$ if $B \geq P^{1+\epsilon}, \epsilon > 0$.

If we assume that the number of active edges decreases by a factor of k in each iteration of the merge stage (same as for number of regions), then the total time required for the merge stage, assuming $B \leq P$ in every iteration, is $O(logP \times log_k\frac{R_i}{R_f})$. The total time required in the general case is $O(\frac{R_i\times logR_i}{P} + logP \times log_k\frac{R_i}{R_f})$. Note that this is a very loose complexity analysis.

CM-5 Implementations: In the data parallel and message passing implementations on the CM-5, each merge step of the algorithm requires a many-to-many communication. The complexity of the many-to-many communication is difficult to analyze, since it depends on the number of the messages sent by every processor, which in turn depends on the image.

5 Performance

The data parallel implementation (CM Fortran) of the split and merge algorithm was executed on both a 16K CM-2 and a 32-node CM-5, while the message passing implementation (F77 + CMMD) was executed on a 32-node CM-5 only. Several images were used to test the various implementations. These images are shown in Figure 5 in the Appendix.

5.1 Smallest-ID vs. Random Approach in Resolving Ties

Table I compares the smallest-ID and random approaches in resolving ties during the merge stage. The table presents the execution time and the number of iterations required by the merge stage of the data parallel implementation (CM Fortran) on the CM-5, using each of the two approaches. Invariably, in all of the images, the random approach in resolving ties proved to be significantly faster than the approach of selecting the region with the smallest ID. Similar results were obtained for the message passing implementation on the CM-5, as well as the data parallel implementation on the CM-2.

	Merge Stage (Smallest-ID Approach)		Merge Stage (Random Approach)	
	Time (sec)	Iterations	Time (sec)	Iterations
Image 1:	334.948	290	33.013	19
Image 2:	151.670	153	31.615	20
Image 3:	1406.099	809	42.570	27
Image 4:	622.980	549	37.588	25
Image 5:	186.834	226	24.471	16
Image 6:	1754.254	1062	75.582	45

Table I: Comparison of Smallest-ID and Random Approaches in Breaking Ties in the Data Parallel Implementation on the CM-5 (32 nodes)

5.2 Comparison of the Parallel Implementations

The bar chart of Figure 3 gives a visual comparison of the times taken by the merge stage in the various implementations. **LP** refers to the Linear Permutation communication scheme and **Async** refers to the asynchronous one.

Figure 4 presents the execution time and speedup of the merge stage in the message passing implementation on the CM-5 using asynchronous communication, as a function of the number of processors used.

The detailed timings of the various implementations (using the random approach in resolving ties) is presented in Table II in the Appendix.

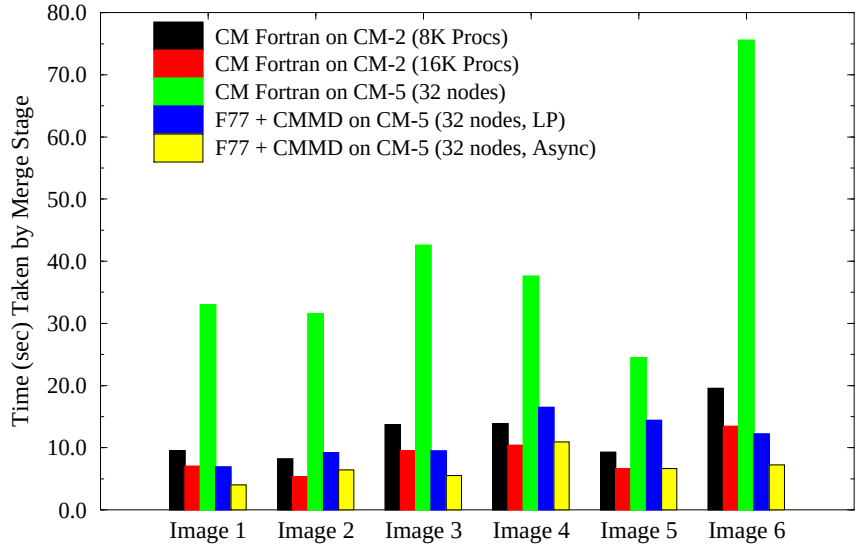


Figure 3: Execution time of the Merge Stage in the Various Implementations

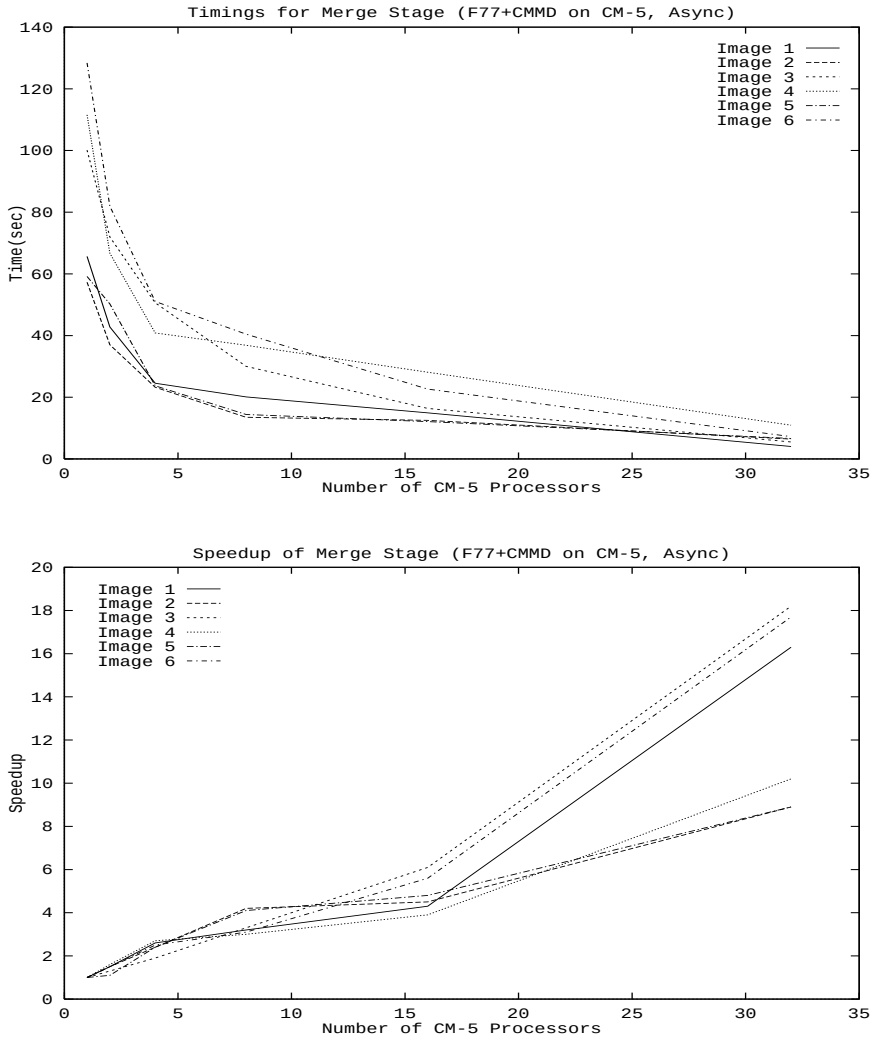


Figure 4: Execution Time and Speedup of the Merge Stage on the CM-5 as a Function of the Number of Processors

As was previously mentioned, the message passing version of the region growing algorithm is essentially a straightforward, hand-coded translation of the data parallel CM Fortran version. The CM Fortran version was easier to program than the message passing one (the number of lines of source code was 2525 and 1128 in the message passing and CM Fortran versions, respectively). In the message passing version, the programmer explicitly specifies synchronization, data partitioning, and communication, while, in the CM Fortran version, the compiler and the run-time system insert synchronization, lay out the data, and provide communication among the node processors.

The experimental results presented in Table II and Figures 3, 4 show that the message passing version exhibits reasonable speedup on the CM-5, an MIMD machine. However, the compiled CM Fortran version on the same machine runs significantly slower. We believe that with a more efficient implementation of a CM Fortran/High Performance Fortran (HPF) compiler, the performance of the data parallel version should be closer to that of the hand-coded message passing one. We are developing a Fortran 90D/HPF compiler which supports the above conversion [3]. We plan to test the performance of the region growing algorithm with this compiler and present results at a later stage.

Of the two communication schemes investigated on the CM-5, the asynchronous scheme is faster. In the Linear Permutation scheme, the processors must iterate Q times, where Q is the number of processors used, until all the required sends and receives are completed. In the asynchronous scheme, however, the number of iterations is $\leq Q$ and is dependent on the number of messages to be sent and received.

The graph that models the region growing problem constantly evolves during the course of the computation. In the current message passing implementation, the vertices and edges of the graph remain in the same processors throughout the merge stage. This, in general, leads to load imbalance. A potential approach would be to let the active vertices and edges migrate between the processors, so the load is more evenly distributed. We are currently investigating various load balancing schemes and their tradeoffs.

6 Conclusions

We have presented a parallel algorithm for solving the region growing problem based on the split and merge approach. Ties during merging were resolved by selecting a partner at random. The algorithm was implemented on the Connection Machine, models CM-2 and CM-5, in both the data parallel and message passing programming paradigms. The performance of the algorithm using the different architectures and programming models was analyzed and compared.

Acknowledgements: We would like to thank Paul Coddington, Pablo Tamayo, and Jhy-Chun Wang for interesting and helpful discussions; Gregor von Laszewski for help in preparing the manuscript; and the referees for their useful and insightful comments.

References

[1] H. Alnuweiri and V. Prasanna, "Parallel Architectures and Algorithms for Image Component Labeling", *IEEE Trans. Patt. Anal. Machine Intell.*, **14**, pp. 1014-1034, 1992.

[2] D. Ballard and C. Brown, *Computer Vision*, Prentice Hall, Englewood Cliffs, NJ, 1982.

[3] Z. Bozkus et al, "Compiling Fortran 90D/HPF for Distributed Memory MIMD Computers", Tech. Report # 444, Northeast Parallel Architectures Center, Syracuse Univ., May 1993.

[4] S. Even, *Graph Algorithms*, Computer Science Press, Potomac, MD, 1979.

[5] G. Fox et al, "Software support for irregular and loosely synchronous problems", Tech. Report, Northeast Parallel Architectures Center, Syracuse Univ., May 1992.

[6] S. Hambrusch, X. He, and R. Miller, "Parallel Algorithms for Gray-Scale Digitized Picture Component Labeling on a Mesh-Connected Computer", *J. Parallel Distrib. Comput.*, to appear.

[7] R. M. Haralick and L. G. Shapiro, "Image Segmentation Techniques", *Computer Vision, Graphics, and Image Processing* **29**, pp. 100-132, 1985.

[8] S. L. Horowitz and T. Pavlidis, "Picture Segmentation By a Directed Split-and-Merge Procedure", *Proc. 2nd International Joint Conf. on Pattern Recognition*, pp. 424-433, 1974.

[9] C. Leiserson, "The Network Architecture of the Connection Machine CM-5", *Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, San Diego, CA, 1992.

[10] T. Pavlidis, "Image Analysis", *Annual Review of Computer Science* **3**, pp. 121-146, 1988.

[11] S. Ranka and S. Sahni, *Hypercube Algorithms*. Springer-Verlag, New York, 1990.

[12] S. Ranka, J. Wang, and G. Fox, "Static and runtime algorithms for all-to-many personalized communication on permutation networks", *Proc. International Conf. on Parallel and Distributed Systems*, 1992.

[13] J. C. Tilton, "Image segmentation by iterative parallel region growing with applications to data compression and image analysis", *Proc. 2nd Symposium on the Frontiers of Massively Parallel Computation*, 1988.

[14] M. Willebeek-LeMair and A. Reeves, "Solving non-uniform problems on SIMD computers: Case study on region growing", *J. Parallel Distrib. Comput.* **8**, pp. 135-149, 1990.

[15] S. W. Zucker, "Region growing: Childhood and adolescence", *Computer Graphics and Image Processing* **5**, pp. 382-399, 1976.

APPENDIX

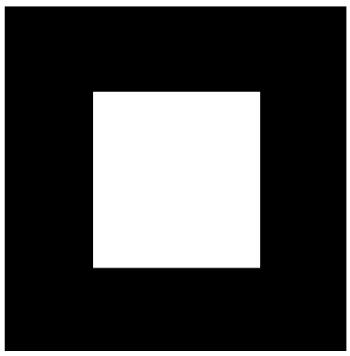


Image 1 (128 x 128 image)

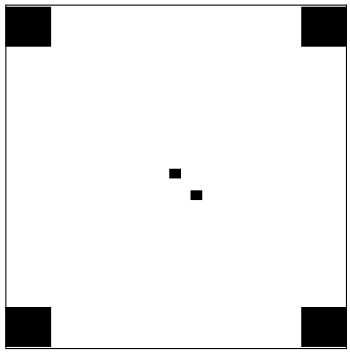


Image 2 (128 x 128 image)

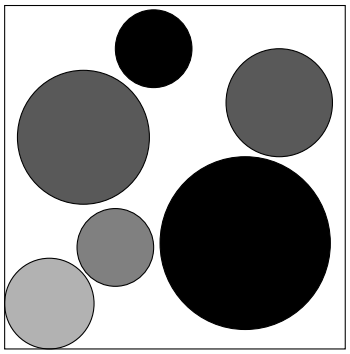


Image 3 (128 x 128 image)

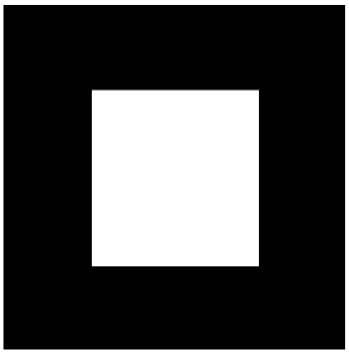


Image 4 (256 x 256 image)

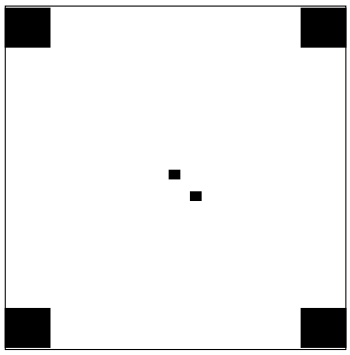


Image 5 (256 x 256 image)

Image 6 (256 x 256 image)

Figure 5: Images 1-6

Image 1: 128×128 image composed of two nested rectangular regions
No. of square regions found at end of split stage = 436
No. of regions found at end of merge stage = 2

	Split Stage		Merge Stage (Random Approach)	
	Time (sec)	Iterations	Time (sec)	Iterations
CM Fortran on :				
CM-2 (8K procs)	0.200	4	9.511	19
CM-2 (16K procs)	0.112	4	7.027	20
CM-5 (32 nodes)	0.361	4	33.013	19
F77 + CMMD on :				
CM-5 (32 nodes, LP)	0.022	4	6.914	24
CM-5 (32 nodes, Async)	0.021	4	4.025	20

Image 2: 128×128 image composed of a collection of rectangles
No. of square regions found at end of split stage = 193
No. of regions found at end of merge stage = 7

	Split Stage		Merge Stage (Random Approach)	
	Time (sec)	Iterations	Time (sec)	Iterations
CM Fortran on :				
CM-2 (8K procs)	0.200	4	8.184	18
CM-2 (16K procs)	0.112	4	5.345	17
CM-5 (32 nodes)	0.360	4	31.615	20
F77 + CMMD on :				
CM-5 (32 nodes, LP)	0.022	4	9.236	35
CM-5 (32 nodes, Async)	0.021	4	6.441	35

Image 3: 128×128 image composed of a collection of circles
No. of square regions found at end of split stage = 1732
No. of regions found at end of merge stage = 11

	Split Stage		Merge Stage (Random Approach)	
	Time (sec)	Iterations	Time (sec)	Iterations
CM Fortran on :				
CM-2 (8K procs)	0.200	4	13.711	24
CM-2 (16K procs)	0.112	4	9.538	25
CM-5 (32 nodes)	0.361	4	42.570	27
F77 + CMMD on :				
CM-5 (32 nodes, LP)	0.022	4	9.454	33
CM-5 (32 nodes, Async)	0.021	4	5.516	28

Table II: Comparison of the Performance of the Parallel Implementations

Image 4: 256×256 image composed of two nested rectangular regions
No. of square regions found at end of split stage = 823
No. of regions found at end of merge stage = 2

	Split Stage		Merge Stage (Random Approach)	
	Time (sec)	Iterations	Time (sec)	Iterations
CM Fortran on :				
CM-2 (8K procs)	1.008	5	13.882	26
CM-2 (16K procs)	0.529	5	10.381	28
CM-5 (32 nodes)	2.052	5	37.588	25
F77 + CMMD on :				
CM-5 (32 nodes, LP)	0.097	5	16.512	37
CM-5 (32 nodes, Async)	0.097	5	10.942	29

Image 5: 256×256 image composed of a collection of rectangles
No. of square regions found at end of split stage = 298
No. of regions found at end of merge stage = 7

	Split Stage		Merge Stage (Random Approach)	
	Time (sec)	Iterations	Time (sec)	Iterations
CM Fortran on :				
CM-2 (8K procs)	1.008	5	9.287	19
CM-2 (16K procs)	0.529	5	6.633	20
CM-5 (32 nodes)	2.046	5	24.471	16
F77 + CMMD on :				
CM-5 (32 nodes, LP)	0.099	5	14.388	35
CM-5 (32 nodes, Async)	0.098	5	6.640	35

Image 6: 256×256 image of a "tool"
No. of square regions found at end of split stage = 2248
No. of regions found at end of merge stage = 4

	Split Stage		Merge Stage (Random Approach)	
	Time (sec)	Iterations	Time (sec)	Iterations
CM Fortran on :				
CM-2 (8K procs)	1.008	5	19.530	34
CM-2 (16K procs)	0.529	5	13.426	33
CM-5 (32 nodes)	2.066	5	75.582	45
F77 + CMMD on :				
CM-5 (32 nodes, LP)	0.098	5	12.192	36
CM-5 (32 nodes, Async)	0.098	5	7.236	38

Table II, cont'd.: Comparison of the Performance of the Parallel Implementations