Parallel Algorithms for Image Enhancement and Segmentation by Region Growing with an Experimental Study (Extended Abstract)

David A. Bader[*] Joseph JáJá[†] David Harwood[‡] Larry S. Davis[§]

Institute for Advanced Computer Studies
University of Maryland, College Park, MD 20742
E-mail: {dbader, joseph, harwood, lsd}@umiacs.umd.edu

Abstract

This paper presents efficient and portable implementations of a useful image enhancement process, the Symmetric Neighborhood Filter (SNF), and an image segmentation technique which makes use of the SNF and a variant of the conventional connected components algorithm which we call δ-Connected Components. We use efficient techniques for distributing and coalescing data as well as efficient combinations of task and data parallelism. The image segmentation algorithm makes use of an efficient connected components algorithm based on a novel approach for parallel merging. The algorithms have been coded in SPLIT-C *and run on a variety of platforms, including the Thinking Machines CM-5, IBM SP-1 and SP-2, Cray Research T3D, Meiko Scientific CS-2, Intel Paragon, and workstation clusters. Our experimental results are consistent with the theoretical analysis (and provide the best known execution times for segmentation, even when compared with machine-specific implementations.) Our test data include difficult images from the Landsat Thematic Mapper (TM) satellite data.*

1. Problem Overview

Given an $n \times n$ image with k gray levels on a p processor machine ($p \leq n^2$), we wish to develop efficient and portable parallel algorithms to perform various useful image processing computations. Image segmentation algorithms cluster pixels into homogeneous regions, which, for example, can be classified into categories with higher accuracy than could be obtained by classifying the individual pixels. Region growing is a class of techniques used in image segmentation algorithms in which, typically, regions are constructed by an agglomeration process that adds (merges) pixels to regions when those pixels are both adjacent to the regions and similar in property (most simply intensity) (e.g. [7, 12, 26]). Each pixel in the image receives a label from the region growing process; pixels will have the same label if and only if they belong to the same region. Our algorithm makes use of an efficient and fast parallel connected components algorithm [2, 3] based on a novel approach for merging.

In real images, natural regions have significant variability in gray level. Noise, introduced from the scanning of the real scene into the digital domain, will cause single pixels outliers. Also, lighting changes can cause a gradient of gray levels in pixels across the same region. Because of these and other similar effects, we preprocess the image with a stable filter, the Symmetric Neighborhood Filter (SNF) [13], that smooths out the interior pixels of a region to a near-homogeneous level. Also, due to relative motion of the camera and the scene, as well as aperture effects, edges or regions are usually blurred so that the transition in gray levels between regions is not a perfect step over a single pixel, but ramps from one region to the other over several pixels. Our filter is, additionally, an edge-preserving filter which detects blurred transitions such as these and sharpens them while preserving the true border location as best as possible. Most preprocessing filters will smooth the interior of regions at the cost of degrading the edges, or conversely, detect edges while introducing intrinsic error on previously homogeneous regions. However, the SNF is an edge-preserving smoothing filter which performs well for both edge-sharpening and region smoothing. It is an

[*]Also affiliated with Department of Electrical Engineering. The support by NASA Graduate Student Researcher Fellowship No. NGT-50951 is gratefully acknowledged.

[†]Also affiliated with Department of Electrical Engineering. Supported in part by NSF grant No. CCR-9103135 and NSF HPCC/GCAG grant No. BIR-9318183.

[‡]Supported by NSF HPCC/GCAG grant No. BIR-9318183.

[§]Also affiliated with the Department of Computer Science and the Center for Automation Research; supported by NSF HPCC/GCAG grant No. BIR-9318183.

iterative filter which also can be tuned to retain thin image structures corresponding, e.g., to rivers, roads, etc. A variety of SNF operators have been studied, and we chose a single parameter version which has been shown to perform well on remote sensing applications.

The majority of previous parallel implementations of the SNF filter are architecture- or machine-specific and do not port well to other platforms (e.g. [10, 18, 19, 21]). For example, [22] gives an implementation of a 15×15 SNF filter on the CMU Warp, a 10-processor linear systolic array, which takes 4.76 seconds on a 512×512 image. We present our SNF filter execution timings in Figure 7. In comparison, on a 32-processor TMC CM-5, we take less than 165 milliseconds per iteration operating on an image of equivalent size.

After the image is enhanced by the SNF, we use a variant of the connected components algorithm for gray level images, called δ-Connected Components, to combine similar pixels into homogeneously labeled regions producing the final image segmentation. As with the SNF implementations, most previous parallel algorithms for segmentation do not port well to other platforms (e.g. [9, 17, 20, 23, 24]).

2. Block Distributed Memory Model

We use the Block Distributed Memory (BDM) Model ([15, 16]) as a computation model for developing and analyzing our parallel algorithms on distributed memory machines. Each of our hardware platforms can be viewed as a collection of powerful processors connected by a communication network that can be modeled as a complete graph on which communication is subject to the restrictions imposed by the latency and the bandwidth properties of the network. We view a parallel algorithm as a sequence of local computations interleaved with communication steps, and we allow computation and communication to overlap. The complexity of parallel algorithms will be evaluated in terms of two measures: the computation time $T_{comp}(n,p)$, and the communication time $T_{comm}(n,p)$.

The communication time $T_{comm}(n,p)$ refers to the total amount of communications time spent by the overall algorithm in accessing remote data. The transfer of a block consisting of m contiguous words, assuming no congestion, takes $\tau + \sigma m$ time, where τ is an upper bound on the latency of the network and σ is the time per word at which a processor can inject or receive data from the network. The algorithms in this paper utilized two collective communication primitives, **transpose**, **bcast**, and **reduce**, where **transpose** is an all-to-all communication of equal sized buffers and **reduce** is a prefix-sum, both modeled by $\tau + \sigma \max(m,p)$, and **bcast** is a one-to-all communication, again with equal sized buffers, modeled by $2(\tau + \sigma \max(m,p))$, where m is the maximum amount of data transmitted or received by a processor [15, 16, 2, 3, 4, 5]. Using this cost model, we can evaluate the communication time $T_{comm}(n,p)$ of an algorithm as a function of the input size n, the number of processors p, and the parameters τ and σ.

We define the computation time $T_{comp}(n,p)$ as the maximum time it takes a processor to perform all the local computation steps. In general, the overall performance $T_{comp}(n,p) + T_{comm}(n,p)$ involves a tradeoff between $T_{comm}(n,p)$ and $T_{comp}(n,p)$. Our aim is to develop parallel algorithms that achieve $T_{comp}(n,p) = O\left(\frac{T_{seq}}{p}\right)$ such that $T_{comm}(n,p)$ is minimum, where T_{seq} is the complexity of the best sequential algorithm. Such optimization has worked very well for the problems we have looked at, but other optimization criteria are possible. The important point to notice is that, in addition to scalability, our optimization criterion requires that the parallel algorithm be an efficient sequential algorithm (i.e., the total number of operations of the parallel algorithm is of the same order as T_{seq}).

3. Image (Data) Layout and Test Images

A straightforward data layout is used in these algorithms for all platforms. The input image is an $n \times n$ matrix of integers. We assign tiles of the image as equally as possible among the processors. If p is an even power of two, i.e. $p = 2^d$, for even d, the processors will be arranged in a $\sqrt{p} \times \sqrt{p}$ logical grid. For future reference, we will denote the number of rows in this logical grid as v and the number of columns as w. For odd d, we assign the number of rows of the logical processor grid to be $v = 2^{\lfloor \frac{d}{2} \rfloor}$, and the number of columns to be $w = 2^{\lceil \frac{d}{2} \rceil}$. Each processor initially owns a tile of size $\frac{n}{v} \times \frac{n}{w}$. For future reference, we assign $q = \frac{n}{v}$ and $r = \frac{n}{w}$. We assume that the p processors are labeled consecutively from 0 to $p-1$ and are assigned in row-major order to the logical processor grid just described.

Our test images are shown in [5] and Appendix B. We use Landsat satellite data to represent real images; Figure 8, taken from band 4 of a view of New Orleans, is a 256 gray level, 512×512 pixel array from Landsat Thematic Mapper (TM) satellite data. A 128×128 subimage of these scene is shown in Figure 9.

4. Image Segmentation - Overview

Images are segmented by running several phases of the SNF enhancement algorithm, followed by several iterations of the 1-Nearest Neighbor filter, and finally, δ-Connected Components. See Figure 1 for a dataflow diagram of the complete segmentation process.

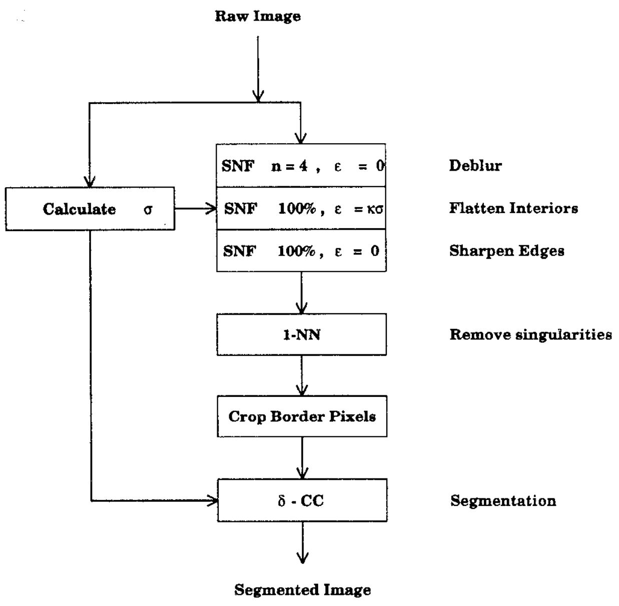


Figure 1. Segmentation Process

4.1. Symmetric Neighborhood Filter

Due to interior variation as well as noise and blur, regions in real images are seldom homogeneous in gray level and sharp along their borders. Preprocessing the image with an enhancement filter that reduces these effects will yield better segmentation results.

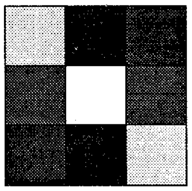


Figure 2. Symmetric Pairs of Pixels

The SNF filter compares each pixel to its 8-connected neighbors. (Note that the 1-pixel image boundary is ignored in our implementation.) The neighbors are inspected in symmetric pairs around the center, i.e. N $\sim$ S, W $\sim$ E, NW $\sim$ SE, and NE $\sim$ SW; see Figure 2 for diagram of a 3×3 neighborhood centered around a pixel, with the symmetric pairs colored the same. Using each pair and the center pixel, we select one of the three in each of the four comparisons using the following criteria. Assume without loss of generality that the pair of pixels are colored A and B, and $A > B$ (see Figure 3). If the center pixel (with value x) falls within region R_A, that is, $\frac{A+B}{2} < x \leq A + \epsilon$, then we select A. Likewise, if the center pixel falls within region R_B, i.e., $B - \epsilon \leq x < \frac{A+B}{2}$, then we select B. And if x is midway between A and B, we simply select x which is the average. Finally, if x is an outlier with respect to A and B, so that $x > A + \epsilon$ or $x < B - \epsilon$, we select x to leave it fixed. The collection of four selected pixels are then averaged together, and finally, the center pixel is replaced by the mean of this average and the center pixel's current gray level value. This latter average is similar to that of a damped gradient descent which yields a faster convergence.

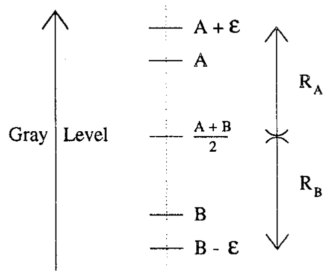


Figure 3. Selection of SNF Pixels

The first phase of segmentation is a combination of three iterative SNF filters. The first step runs for a small number of iterations (e.g. four) with $\epsilon = 0$ and is used to preserve edges. We define σ to be the median of the standard deviations of all 3×3 neighborhoods centered around each non-border pixel in the image. See [4] for a parallel median algorithm. To flatten the interior of regions, SNF is iterated with $\epsilon = \kappa\sigma$, where κ is typically set to 2.0 for this application. The stopping criteria for this iterative filter occurs when the percentage of "fixed" pixels reaches 100.0 %, this percentage has not changed for three iterations, or when we reach 200 iterations, whichever comes first. Finally, we sharpen the borders of regions with SNF using $\epsilon = 0$, again stopping the iterative process when the pixels have fixed, as defined above. The resulting image has near-homogeneous regions with sharp transitions between bordering regions.

4.2. 1-Nearest Neighbor Filter

Single pixel regions rarely can be classified, even under the best circumstances. Therefore, we prefer to filter these out as our last enhancement stage. A typical 1-Nearest Neighbor filter removes single pixel outliers by replacing each pixel in the image with the value of one of its adjacent pixels which is closest to its own gray level. Note that one application of the 1-Nearest Neighbor filter may cause small neighborhoods of pixels to oscillate. For example, two adjacent pixels with values A and $A + \Delta$ surrounded by a region of more than Δ levels above or below would never stabilize. Therefore, we apply the 1-Nearest Neighbor as an iterative filter, stopping when the input and output images are identical. For faster convergence, we use a damped approach (similar to the SNF) which assigns an output pixel to the mean of its original and nearest neighbor values. Typically, we converge in roughly six to eight iterations.

Since no image enhancement occurs along the pixels of image borders, we crop the border so that additional segmentation techniques will not use this raw data to merge dissimilar regions via paths through the noisy, uncorrected pixels. For this application, we crop the border by a width of three pixels.

4.3. δ-Connected Components

The image processing problem of determining the connected components of images is a fundamental task of imaging systems (e.g. [1, 6, 11, 14]). All pixels with gray level (or 'color') 0 are assumed to be background, while pixels with color > 0 are foreground objects. A connected component in the image is a maximal collection of uniformly colored pixels such that a path exists between any pair of pixels in the component. Note that we are using the notion of 8-connectivity, meaning that two pixels are adjacent if and only if one pixel lies in any of the eight positions surrounding the other pixel. Each pixel in the image will receive a label; pixels will have the same label if and only if they belong to the same connected component. Also, all background pixels will receive a label of 0.

It is interesting to note that, in the previous paragraph, we defined connected components as a maximal collection of uniform color pixels such that a **path** existed between any pair of pixels. The conventional algorithm assumes that there is a connection between two adjacent pixels if and only if their gray level values are identical. We now relax this connectivity rule and present it as a more general algorithm called **δ-Connected Components**. In this approach, we assume that two adjacent pixels with values x and y are connected if their absolute difference $|x - y|$ is no greater than the threshold δ. Note that setting the parameter δ to 0 reduces the algorithm to the classic connected components approach. This algorithm is identical in analysis and complexity to the conventional connected components algorithm, as we are merely changing the criterion for checking the equivalence of two pixels.

For the final phase in the segmentation process, δ-Connected Components is applied to the enhanced image, using $\delta = \kappa\sigma$, where the values of κ and σ are the same as those input to the enhancement filters. The analysis for the δ-Connected Components algorithm is given in Section 6, equation (5). Thus, we have an efficient algorithm for image segmentation on parallel computers.

4.4. Test Images

We use the Landsat Thematic Mapper (TM) raw satellite data for our test images. Each test image is a 512×512 pixel subimage from a single TM band. A subimage from band 5, an image from South America, is given in [5], and Figure 8 is taken from band 4 of New Orleans data. These images have 256 gray levels and also have post-processing enhancement of the brightness for visualization purposes in this paper. We have applied SNF enhancement to these images, and the results appear below the original images. A further segmentation with $\delta = \kappa\sigma$ using the δ-Connected Components algorithm is given at the bottom of Figures 8 and 9.

5. Symmetric Neighborhood Filter - Parallel Implementation

A useful data movement needed for the 3×3 local SNF filter is the fetching of tile-based **ghost cells** ([8, 25]) which contain shadow copies of neighboring tiles' pixel borders. These ghost cells are used in the selection process when recalculating our tile's border. Suppose each tile of the image allocated to a processor is $q \times r$ pixels. We have four ghost cell arrays, **ghostN** and **ghostS** which hold r pixels each, and **ghostW** and **ghostE** which hold q pixels each. In addition, four single pixel ghost cells for diagonal neighboring pixels are **ghostNW**, **ghostNE**, **ghostSE**, and **ghostSW**. An example of these ghost cells is pictured in Figure 4.

The analysis for the prefetching of ghost cells is simple. We can divide the prefetching into eight separate data movements, one for each direction. Since each movement is a permutation, i.e. it has a unique source and destination, it can be routed with little or no contention. Thus, the entire ghost cell prefetching takes

$$\begin{cases} T_{comm}(n,p) & \leq & 8\tau + 4\frac{n}{\sqrt{p}} + 4; \\ T_{comp}(n,p) & = & O\left(\frac{n}{\sqrt{p}}\right). \end{cases} \tag{1}$$

A second data movement needed for SNF is the reduction operation using the **reduce** collective communication primitive. Each processor i has a data value, Z_i, and we need the value of $Z_0 \oplus Z_1 \oplus \ldots \oplus Z_{p-1}$, where $\oplus$ is any associative operator. The complexity of this can be shown to be: [4, 5]

$$\begin{cases} T_{comm}(n,p) & \leq & \tau + p - 1; \\ T_{comp}(n,p) & = & O(p). \end{cases} \tag{2}$$

An SPMD algorithm for an iteration of SNF on Processor i:

Algorithm 1 *Symmetric Neighborhood Filter*

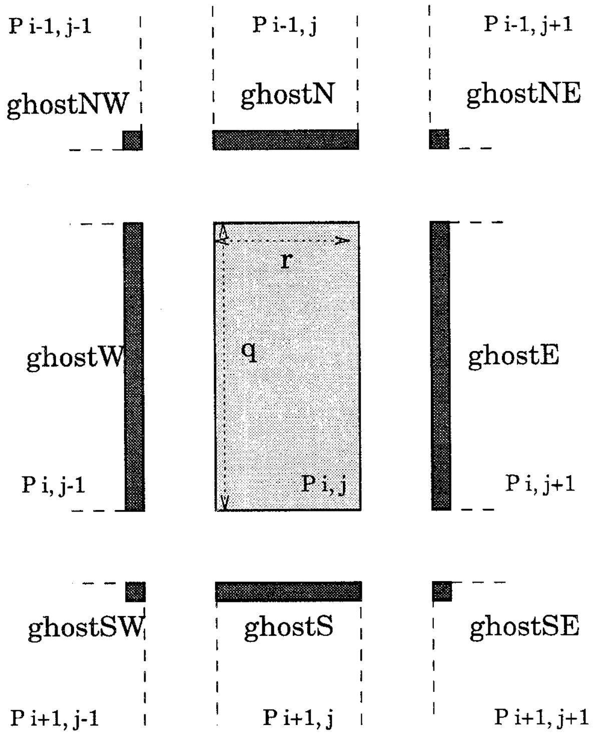


Figure 4. An example of Ghost Cells

Block Distributed Memory Model Algorithm.

```
Input:
    { $i$ } is my processor number;
    { $p$ } is the total number of processors, labeled from 0 to $p-1$;
    { $A$ } is the $n \times n$ input image.
    { $\epsilon$ } is input parameter.
begin
    0. Processor $i$ gets an $\frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}}$ tile of image $A$,
          denoted $A_i$.
    1. Prefetch Ghost Cells.
    2. For each local pixel $A_{i,<x,y>}$ that has not
          fixed yet, using $\epsilon$, compute $B_{i,<x,y>}$, the updated
          pixel value. Decide if local pixel position $< x, y >$
          is now fixed.
    3. Set $f_i$ equal to the number of local pixels that
          have remained fixed.
    4. $f$ = reduce($f_i$, +),
          that is, $f = \sum_{i=0}^{p-1} f_i$.
    5. Output $\frac{f}{n^2} \times 100\%$.
end
```

For each iteration of the SNF operator on a p-processor machine, the theoretical analysis is as follows. The complexities for Step 1 and Step 4 are shown in (1) and (2), respectively. Steps 2 and 3 are completely local and take $O\left(\frac{n^2}{p}\right)$. Thus, for $p \le n$, the SNF complexities are

$$\begin{cases} T_{comm}(n,p) & \le & 9\tau + 4\frac{n}{\sqrt{p}} + 3 + p; \\ T_{comp}(n,p) & = & O\left(\frac{n}{\sqrt{p}} + p\right) + O\left(\frac{n^2}{p}\right) \\ & = & O\left(\frac{n^2}{p}\right). \end{cases} \quad (3)$$

Figure 7 in Appendix A shows the convergence of the SNF enhancement during the second phase of the smoothing filter. As can be seen, there is a fast convergence of the pixels asymptotically close to 100% fixed. Because fixed pixels are not recalculated, the time per iteration quickly ramps down from approximately 165 ms/iteration to 26 ms/iteration on a 512×512 TM image.

The complexity of an iteration of the 1-Nearest Neighbor filter is simple, namely, a fetch of ghost cells and one pass through the image tile on each processor. The ghost cell analysis is given in (1), and the update of pixels takes $O\left(\frac{n^2}{p}\right)$. Therefore, the 1-Nearest Neighbor algorithm has complexities

$$\begin{cases} T_{comm}(n,p) & \le & 8\tau + 4\frac{n}{\sqrt{p}} + 4; \\ T_{comp}(n,p) & = & O\left(\frac{n^2}{p}\right). \end{cases} \quad (4)$$

6. δ-Connected Components of Grayscale Images

The high-level strategy of our connected components algorithm uses the well-known divide and conquer technique. Divide and conquer algorithms typically use a recursive strategy to split problems into smaller subproblems and, given the solutions to these subproblems, merge the results into the final solution. It is common to have either an easy splitting algorithm and a more complicated merging, or vice versa, a hard splitting, following by easy merging. In our parallel connected components algorithm, the splitting phase is trivial and implicit, while the merging process requires more work.

Each processor holds a unique tile of the image, and hence can find the initial connected components of its tile by using a standard sequential algorithm. Next, the algorithm iterates $\log p$ times[1], alternating between combining the tiles in **horizontal merges** of vertical borders and **vertical merges** of horizontal borders. Our algorithm uses novel techniques to perform the merges and to update the labels. We will attempt to give an overview of this algorithm; for a complete description, see [2, 3, 5].

We merge the p subimages into larger and larger image sections with consistent labelings. There will be $\log p$ iterations since we cut the number of uncombined subimages

[1]Note that throughout this paper "$\log x$" will always be the logarithm of x to the base $b = 2$, i.e. $\log_2 x$.

in half during each iteration. Unlike previous connected components algorithms, we use a technique which identifies processors as **group managers** or **clients** during each phase. The group managers have the task of organizing the retrieval of boundary data, performing the merge, and creating the list of label changes. Once the group managers broadcast these changes to their respective clients, all processors must use the information to update their **tile hooks**, data structures which point to connected components on local tile borders. See Figure 5 for an illustration of the **tile hook** data structure in which three tile hooks contain the information needed to update the border pixels. The clients assist the group managers by participating in the coalescing of data during each merge phase. Finally, the complete relabeling is performed at the very end using information from the tile hooks.

During each merge, a subset of the processors will act as **group managers**. These designated processors will prefetch the necessary border information along the column (or row) that they are located upon in the logical processor grid, setting up an equivalent graph problem, running a sequential connected components algorithm on the graph (with the modification that two adjacent nodes are connected if they differ in color by no more than δ), noting any changes in the labels, and storing these changes ((α_i, β_i) pairs) in a shared structure. The **clients** decide who their current group manager is and wait until the list of label changes is ready. They retrieve the list, and all processors make the necessary updates to a proper subset of their labels.

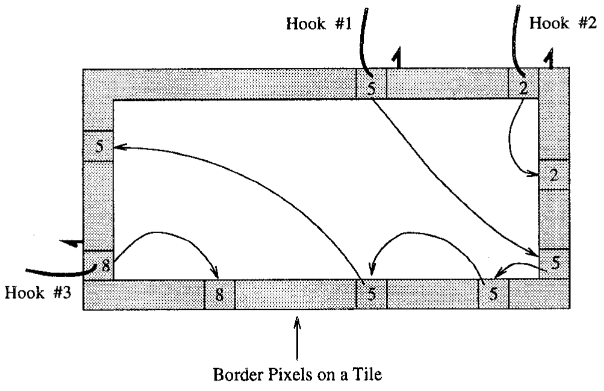


Figure 5. An example of Tile Hooks

At the conclusion of each of the $\log p$ merging steps, only the labels of pixels on the border of each tile are updated. There is no need to relabel interior pixels since they are not used in the merging stage. Only boundary pixels need their labels updated. Taking advantage of this, we do not need to propagate boundary labels inward to recolor a tile's interior pixels after each iteration of the merge. This is one of the attractive highlights of our newly proposed algorithm; namely, the drastically limited updates needed during the merging phase. Finally, after the last step of merging, each processor updates its interior pixel labels.

6.1. Parallel Complexity for δ-Connected Components

Thus, for $p \leq n$, the total complexities for the parallel δ-Connected Components algorithm are [2, 3, 5]

$$\left\{ \begin{array}{lcl} T_{comm}(n,p) & \leq & (4\log p)\tau + (24n + 2p) \\ & = & (4\log p)\tau + O\left(\frac{n^2}{p}\right); \\ T_{comp}(n,p) & = & O\left(\frac{n^2}{p}\right). \end{array} \right. \qquad (5)$$

Clearly, the computational complexity is the best possible asymptotically. As for the communication complexity, intuitively a latency factor τ has to be incurred during each merge operation, and hence the factor $(\log p)\tau$.

6.2. Experimental Results

Researcher(s)	Machine	PE's	Time	work/pix
Copty et al. [7]	TMC CM-2	8192	13.911 s *	217 ms
		16384	9.650 s *	302 ms
1994	TMC CM-5	32	42.931 s *	83.9 ms
			9.567 s †	18.5 ms
			5.537 s ‡	10.8 ms
Bader et al.	TMC CM-5	16	81.6 ms	79.7 μs
(this paper)		32	72.0 ms	141 μs
	IBM SP-2-WD	4	62.9 ms	15.4 μs
1995	Meiko CS-2	4	99.6 ms	24.3 μs
		8	90.9 ms	44.4 μs

Table I. Implementation Results of Segmentation Algorithm on Image 3 from [7], seven gray circles (128×128 pixels)

Researcher(s)	Machine	PE's	Time	work/pix
Copty et al. [7]	TMC CM-2	8192	20.538 s *	80.2 ms
		16384	13.955 s *	109 ms
1994	TMC CM-5	32	77.648 s *	37.9 ms
			12.290 s †	6.00 ms
			7.334 s ‡	3.58 ms
Bader et al.	TMC CM-5	16	223 ms	54.4 μs
(this paper)		32	175 ms	85.5 μs
	IBM SP-2-TH	4	202 ms	12.3 μs
		8	187 ms	22.8 μs
1995	IBM SP-2-WD	4	194 ms	11.8 μs
		8	176 ms	21.5 μs
	Meiko CS-2	4	414 ms	25.3 μs
		8	274 ms	33.5 μs
		16	204 ms	49.8 μs
	Cray T3D	4	396 ms	24.2 μs

Table II. Implementation Results of Segmentation Algorithm on Image 6 from [7], a binary tool (256×256 pixels)

Our implementation performs better compared with other recent parallel region growing codes ([7]). Note that this

*data parallel algorithm
†message passing code, communication scheme 1
‡message passing code, communication scheme 2

implementation uses data parallel Fortran on the TMC CM-2 and CM-5 machines, and lower-level implementations on the CM-5 using Fortran with several message passing schemes. Image 3 from [7] is a 256-gray level 128 $\times$ 128 image, containing seven homogeneous circles. Image 6 from [7] is a binary 256 $\times$ 256 image of a tool. Tables I and II show the comparison of execution times for Images 3 and 6, respectively. Because these images are noise-free, our algorithm skips the image enhancement task.

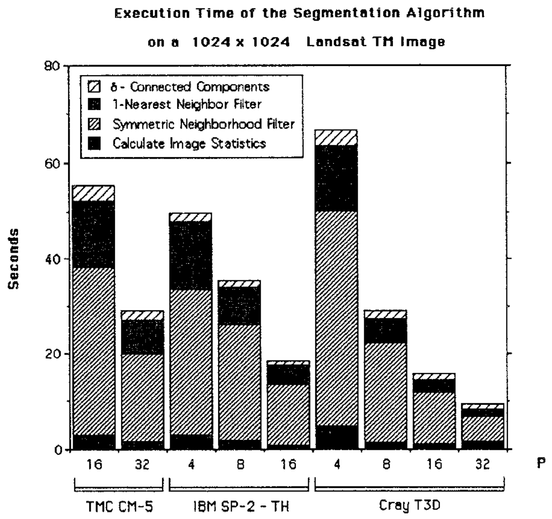


Figure 6. Scalability of the Segmentation Algorithm

Figure 6 shows scalability of the segmentation algorithm running on a 1024 $\times$ 1024 TM band 5 subimage, with various machine configurations of the CM-5, SP-2, and T3D. For this image, the first, second, and third phases of SNF iterate 4, 56, and 47 times, respectively. Also, the 1-Nearest Neighbor task contains 11 iterations. Table III compares the best-known sequential code for SNF to that of the parallel implementation. This test uses the 1024 $\times$ 1024 band 5 image, and iterates with the counts specified above. The sequential tests are performed on fast workstations dedicated to a single user and reflect only the time spent doing the filter calculations. These empirical results show our segmentation algorithm scaling with machine and problem size, and exhibiting superior performance on several parallel machines when compared with state-of-the-art sequential platforms.

Machine	PE's	Time (seconds) for 107 iterations
Sun Sparc 10 - Model 40	1	104
Sun Sparc 20 - Model 50	1	83.6
IBM SP-2-TH	1	78.2
DEC AlphaServer 2100 4/275	1	48.1
TMC CM-5	16	35.2
	32	18.5
IBM SP-2-TH	4	30.9
	8	24.4
	16	12.5
Cray T3D	4	45.3
	8	20.9
	16	10.6
	32	5.35

Table III. Total SNF Execution Time (in seconds) for 1024 $\times$ 1024 **band 5 image**

7. Acknowledgements

We would like to thank the CASTLE/Split-C group at UC Berkeley, especially for the help and encouragement from David Culler, Arvind Krishnamurthy, Lok Tin Liu, Steve Luna, and Rich Martin. Computational support on UC Berkeley's 64-processor TMC CM-5 and 8-processor Intel Paragon was provided by NSF Infrastructure Grant number CDA-8722788. We also thank the Numerical Aerodynamic Simulation Systems Division of NASA's Ames Research Center for use of their 128-processor CM-5 and 128-node IBM SP-2-WD.

We recognize Charles Weems at the University of Massachusetts for providing the DARPA test image suite, and Nawal Copty at Syracuse University for providing additional test images.

Also, Klaus Schauser, Oscar Ibarra, and David Probert of University of California, Santa Barbara, provided access to the 64-node UCSB Meiko CS-2. The Meiko CS-2 Computing Facility was acquired through NSF CISE Infrastructure Grant number CDA-9218202, with support from the College of Engineering and the UCSB Office of Research, for research in parallel computing.

The Jet Propulsion Lab/Caltech Cray T3D Supercomputer used in this investigation was provided by funding from the NASA Offices of Mission to Planet Earth, Aeronautics, and Space Science. We also acknowledge William Carlson and Jesse Draper from the Supercomputing Research Center for writing the parallel compiler AC on which the T3D port of Split-C has been based.

We acknowledge the use of the UMIACS 16-node IBM SP-2-TN2, which was provided by an IBM Shared University Research award and an NSF Academic Research Infrastructure Grant No. CDA9401151.

Please see `http://www.umiacs.umd.edu/~dbader` for additional performance information. In addition, all the code used in this paper is freely available for interested parties from our anonymous ftp site, `ftp://ftp.umiacs.umd.edu/pub/dbader`. We encourage other researchers to compare with our results for similar inputs.

A. Convergence and Execution Time for Band 5 Image

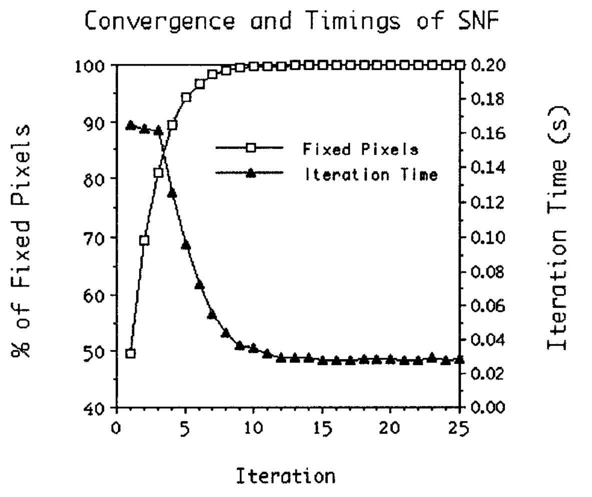
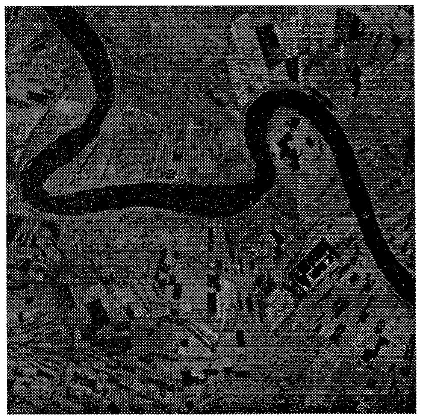
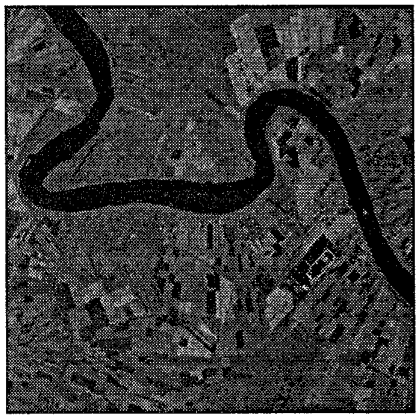
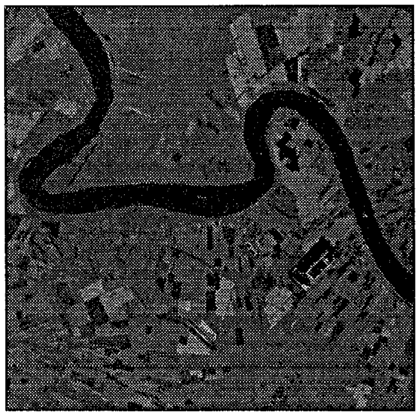


Figure 7. SNF Statistics for a 512×512 Image on a 32-processor CM-5

B. Test Images

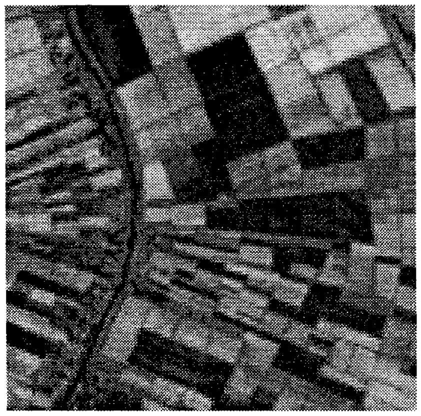

Original Image (New Orleans)

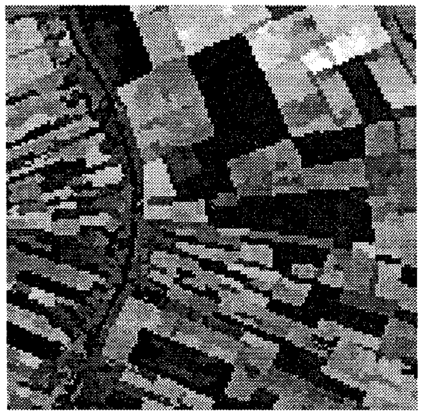

After Image Enhancement


Final Segmentation

(2270 regions)

Figure 8. Landsat TM Band 4 Images (512 × 512 pixels)


Original Image (New Orleans)


After Image Enhancement

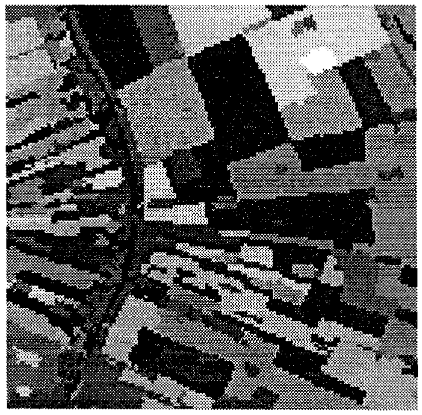

Final Segmentation

Figure 9. Landsat TM Band 4 Images (128 × 128 pixels)

References

[1] H. Alnuweiri and V. Prasanna. Parallel Architectures and Algorithms for Image Component Labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:1014–1034, 1992.

[2] D. Bader and J. JáJá. Parallel Algorithms for Image Histogramming and Connected Components with an Experimental Study. Technical Report CS-TR-3384 and UMIACS-TR-94-133, UMIACS and Electrical Engineering, University of Maryland, College Park, MD, Dec. 1994. To appear in *Journal of Parallel and Distributed Computing*.

[3] D. Bader and J. JáJá. Parallel Algorithms for Image Histogramming and Connected Components with an Experimental Study. In *Fifth ACM SIGPLAN Symposium of Principles and Practice of Parallel Programming*, pages 123–133, Santa Barbara, CA, July 1995.

[4] D. Bader and J. JáJá. Practical Parallel Algorithms for Dynamic Data Redistribution, Median Finding, and Selection. Technical Report CS-TR-3494 and UMIACS-TR-95-74, UMIACS and Electrical Engineering, University of Maryland, College Park, MD, July 1995. To be presented at the 10th *International Parallel Processing Symposium*, Honolulu, HI, April 15-19, 1996.

[5] D. Bader, J. JáJá, D. Harwood, and L. Davis. Parallel Algorithms for Image Enhancement and Segmentation by Region Growing with an Experimental Study. Technical Report CS-TR-3449 and UMIACS-TR-95-44, Institute for Advanced Computer Studies (UMIACS), University of Maryland, College Park, MD, May 1995. To be presented at the 10th *International Parallel Processing Symposium*, Honolulu, HI, April 15-19, 1996.

[6] A. Choudhary and R. Thakur. Connected Component Labeling on Coarse Grain Parallel Computers: An Experimental Study. *Journal of Parallel and Distributed Computing*, 20(1):78–83, January 1994.

[7] N. Copty, S. Ranka, G. Fox, and R. Shankar. A Data Parallel Algorithm for Solving the Region Growing Problem on the Connection Machine. *Journal of Parallel and Distributed Computing*, 21(1):160–168, April 1994.

[8] D. Culler, A. Dusseau, S. Goldstein, A. Krishnamurthy, S. Lumetta, T. von Eicken, and K. Yelick. Parallel Programming in Split-C. In *Proceedings of Supercomputing '93*, pages 262–273, Portland, OR, November 1993.

[9] H. Derin and C.-S. Won. A Parallel Image Segmentation Algorithm Using Relaxation with Varying Neighborhoods and Its Mapping to Array Processors. *Computer Vision, Graphics, and Image Processing*, 40:54–78, 1987.

[10] R. Goldenberg, W. Lau, A. She, and A. Waxman. Progress on the Prototype PIPE. In *Proceedings of the 1987 Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, pages 67–74, Seattle, WA, October 1987.

[11] Y. Han and R. Wagner. An Efficient and Fast Parallel-Connected Component Algorithm. *JACM*, 37(3):626–642, 1990.

[12] R. Haralick and L. Shapiro. Image Processing Techniques. *Computer Vision, Graphics, and Image Processing*, 29:100–132, 1985.

[13] D. Harwood, M. Subbarao, H. Hakalahti, and L. Davis. A New Class of Edge-Preserving Smoothing Filters. *Pattern Recognition Letters*, 6:155–162, 1987.

[14] D. Hirschberg, A. Chandra, and D. Sarwate. Computing Connected Components on Parallel Computers. *Communications of the ACM*, 22(8):461–464, 1979.

[15] J. JáJá and K. Ryu. The Block Distributed Memory Model. Technical Report CS-TR-3207, Computer Science Department, University of Maryland, College Park, January 1994. To appear in *IEEE Transactions on Parallel and Distributed Systems*.

[16] J. JáJá and K. Ryu. The Block Distributed Memory Model for Shared Memory Multiprocessors. In *Proceedings of the 8th International Parallel Processing Symposium*, pages 752–756, Cancún, Mexico, April 1994. (Extended Abstract).

[17] J. Kistler and J. Webb. Connected Components With Split and Merge. In *Proceedings of the 5th International Parallel Processing Symposium*, pages 194–201, Anaheim, CA, April 1991.

[18] P. Narayanan and L. Davis. Replicated Data Algorithms in Image Processing. Technical Report CAR-TR-536/CS-TR-2614, Center for Automation Research, University of Maryland, College Park, MD, February 1991.

[19] M. Pietikäinen, T. Seppänen, and P. Alapuranen. A Hybrid Computer Architecture for Machine Vision. In *Proceedings of the 10th International Conference on Pattern Recognition, Volume 2*, pages 426–431, Atlantic City, NJ, June 1990.

[20] J. Tilton and S. Cox. Segmentation of Remotely Sensed Data Using Parallel Region Growing. In *Ninth International Symposium on Machine Processing of Remotely Sensed Data*, pages 130–137, West Lafayette, IN, June 1983.

[21] R. Wallace, J. Webb, and I.-C. Wu. Machine-Independent Image Processing: Performance of Apply on Diverse Architectures. *Computer Vision, Graphics, and Image Processing*, 48:265–276, 1989.

[22] J. Webb. Architecture-Independent Global Image Processing. In *Proceedings of the 10th International Conference on Pattern Recognition, Volume 2*, pages 623–628, Atlantic City, NJ, June 1990.

[23] T. Westman, D. Harwood, T. Laitinen, and M. Pietikäinen. Color Segmentation By Hierarchical Connected Components Analysis with Image Enhancement by Symmetric Neighborhood Filters. In *Proceedings of the 10th International Conference on Pattern Recognition*, pages 796–802, Atlantic City, NJ, June 1990.

[24] M. Willebeek-LeMair and A. Reeves. Region Growing on a Highly Parallel Mesh-Connected SIMD Computer. In *The 2nd Symposium on the Frontiers of Massively Parallel Computations*, pages 93–100, Fairfax, VA, October 1988.

[25] R. Williams. Parallel Load Balancing for Parallel Applications. Technical Report CCSF-50, Concurrent Supercomputing Facilities, California Institute of Technology, Nov. 1994.

[26] S. Zucker. Region Growing: Childhood and Adolescence. *Computer Graphics and Image Processing*, 5:382–399, 1976.