On Optimization and Parallelization of Fuzzy Connected Segmentation for Medical Imaging

Christopher Gammage
Computer Science Dept.
Wayne State University, Detroit, MI
damage@wayne.edu

Vipin Chaudhary
Institute for Scientific Computing
Wayne State University, Detroit, MI
vipin@wayne.edu

Abstract

Fuzzy Connectedness is an important image segmentation routine for image processing of medical images. It is often used in preparation for surgery and sometimes during surgery. It is important to have an algorithm which can execute very fast, especially in the intra-operative environment. We have taken code from a popular image processing toolkit called ITK and ported it to a C environment. We optimized the implementation to give maximal performance (giving speedup of 23 times). We attempted three different levels of parallelization. We found that MPI was not an efficient method of parallelization as the algorithm is data dependant and large amounts of communication must be done. This communication overshadows the speed increase from doing computation on multiple processors, or nodes in a cluster. However, some limited speedup over the optimizations was obtained using OpenMP on an SMP system leading to a speedup of fifty using four processors over the original ITK implementation.

1. Introduction

The goal of image segmentation is to seperate an image into several different regions. Each region satisfies some criterion. The criterion and method for finding features that fit it are all dependant on the type of segmentation being done. Segmentation of nontrivial images is one of the most difficult tasks in image processing [3].

1.1. Segmentation in Medical Imaging

Segmentation is a very important factor in the medical field [1]. In many medical applications it is important to classify an image into different anatomical or pathological regions. Usually, the modality of image data is either MRI (Magnetic Resonance Imaging), CT (Computed Tomography), PET (Positron Emission Tomography), SPECT (Single Photon Emission Computed Tomography) or ultrasound. However, the quality of segmentation can be improved by combining the images with multimodal registration [4] and doing multi-spectral segmentation [6, 8, 10]. In MRI and CT brain images, surgeons often want to divide the image into gray matter, white matter, and CSF (Cerebral Spinal Fluid).

The need for high speed segmentation plays a very important role in Computer Assisted surgery. It is important to have a real-time robust method of segmentation available at the operation table. Segmentation allows for the fast analysis of critical information. It also serves as an important preprocessing step for the registration of Pre-Operative MRI data to Intra-Operative MRI data.

2. Fuzzy Connectedness

Fuzzy connectedness is based on the notion of fuzzy sets[13]. Fuzzy sets are a mathematical way to represent the uncertainty that we encounter in everyday life [11]. It was first proposed by Prewitt in [7] that fuzzy subsets are useful for the segmentation of medical images. Many have realized that the fuzzy subsets and object notions have significant implications in image segmentation [12, 9]. Segmentation using fuzzy subsets is categorized as a clustering algorithm, which is automated and uses global methods. It is said that fuzzy subsets represent the best hope for unsupervised segmentation [2, 11].

Segmentation based on fuzzy subsets tries to find the connectedness between a seed point and every other point in the image. This connectedness is called the fuzzy connectedness. The resulting set of values of connectivity is called the fuzzy scene. As Udupa [12] puts it, "Fuzzy Connectedness" captures the "Hanging Togetherness" of pixels.

This research was supported in part by Michigan Life Science Corridor Grant

Authorized licensed use limited to: SUNY Buffalo. Downloaded on October 22, 2008 at 13:08 from IEEE Xplore. Restrictions apply.

With the level of 'Fuzziness' in imaging devices, it is natural to think of segmentation in terms of fuzziness. Unfortunately, the process of finding the fuzzy connectedness of every pixel in the image with the seed point is not a fast process [11]. Along each path p there is a "weakest link" that determines the strength of the connectivity. The connectivity between two pixels can be defined differently depending on the type of data being segmented. The method we used in our testing is shown in equation (1). Udupa designed two algorithms to overcome the computation cost of finding fuzzy connectedness. The first, $\kappa\theta_x FOE$ is based on dynamic programming and takes a threshold θ_x as a parameter. The second, κFOE, shown in Algorithm 1, does not use a predetermined threshold. $\kappa\theta_x FOE$ is faster because there is no need to find a best path, as it can terminate upon reaching the threshold value. However, the second algorithm does hold a powerful advantage. After the fuzzy scene values have been computed, the θ_x can interactively be adjusted to fine tune the segmentation. We have found that in computer assisted surgery, this sort of interactivity is an absolute necessity.

$$fuzzy(a,b) = max_{short} * e^{\frac{-0.5*((a+b)/2-\bar{x})^2}{\sigma^2}} \tag{1}$$

Algorithm 1 κFOE

Input : A 3D Image and a 3D seedpoint
Output: A 3D Fuzzy Scene S representing connectedness of each arbitrary pixel to the seedpoint

```
1:  Allocate a 3D image $S$ the same size as the input
2:  Set each image value to 0
3:  Set the seedpoint voxel to the highest value possible
4:  Create an empty queue $Q$
5:  Push indexes of the seed's 6 neighbors to $Q$
6:  while $Q$ is not empty do
7:      Pop an Index $I$ from $Q$
8:      In $Neighbors(I)$, find $max$ in $S$ such that ... †
9:      if $max > S(I)$ then
10:         Set $S(I) = max$
11:         for all $Neighbors(I)$ do
12:             Push to $Q$
13:         end for
14:     end if
15: end while
```

† see udupa [12] for details

3. Optimizations

We have based our algorithm on the implementation of κFOE by the ITK Image Segmentation and Registration Toolkit (Available at www.itk.org) [5]. Before parallelizing any algorithm, it is alway best to first make that algorithm's implementation as efficient as possible.

The implementation by ITK is a heavily templated C++ code. It was coded for flexibility and not for optimal performance. Our first step in optimization was to port the code to C. We compared the speeds of the C Code and ITK Code. We chose a test seedpoint in the center of the 3D volume to segment the white matter from gray matter. The volume used was a $256 \times 256 \times 75$ MRI brain image, shown in Fig. 1. The output fuzzy scene is shown in Fig. 4. The scene can be used to very quickly generate binary images at different threshold values, shown in Figs. 2 and 3.

Using the ITK code, in one case, we found it takes 251 seconds to compute the segmentation results. The C code replaces the linked-list based queue with an array based queue. The new code took 136 seconds to complete. This resulted in a speedup of 1.77 times, where speedup is defined in (2) with $T =$ execution time. The next step was to remove the possibility of duplicate items in the Queue, as suggested by Udupa [12]. This was achieved by creating a 3D array which represents whether a specific pixel is in the queue already. Iterating the queue to check for duplicates proved to be very slow in comparison. Checking the 3D array takes the same amount of time regardless of the size of the queue. This modification allowed the algorithm to complete in 72 seconds and yeilded the same result as the original ITK results. To further speed the processing when changed the exponent calculation in equation (1) to use a lookup table. By changing the size of the lookup table we get more accurate results. Through experimentation, we found the correct size of the lookup table needed in order to get results very similar to what would be givin without a lookup table. This also gave a significance speedup. By using the duplicate protection on the queue, the lookup table and compiler optimizations (loop unrolling and fast math) we realised a total speedup of 21.47 times from ITK's code by without ever using any parallelization techniques (completing in 11.69 seconds). The speedup without using any sort of parallelization seems good, however, the demands of the medical field require real time results.

$$S = \frac{T_{original}}{T_{new}} \tag{2}$$

4. Parallelization

We looked at the κFOE parallelization from three angles. The first was from the distributed computing prespective, using MPI. Then we looked at using SMP, with OpenMP. Lastly, we looked at the hardware level using SSE2. The first thing to notice about the κFOE algorithm is that it is not data independent. In one example

Authorized licensed use limited to: SUNY Buffalo. Downloaded on October 22, 2008 at 13:08 from IEEE Xplore. Restrictions apply.

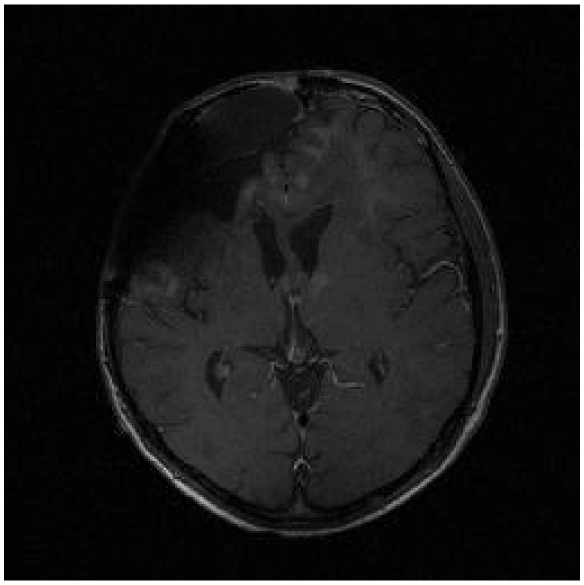

Figure 1. Slice 37 of a $256 \times 256 \times 75$ post operative MRI of a brain

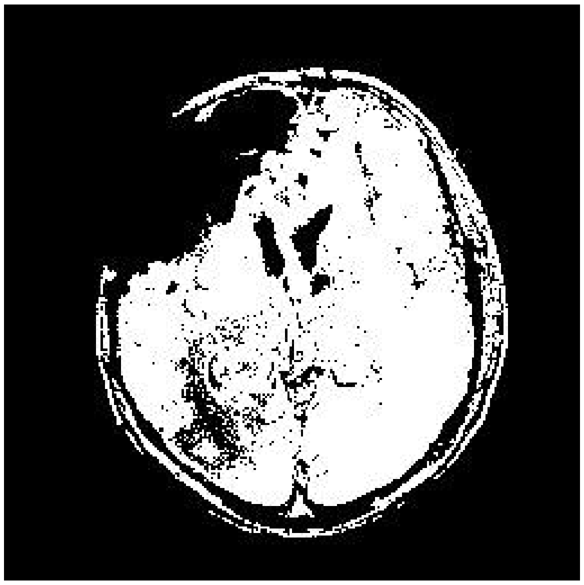

Figure 2. Binary Image, from $threshold = 0.25$

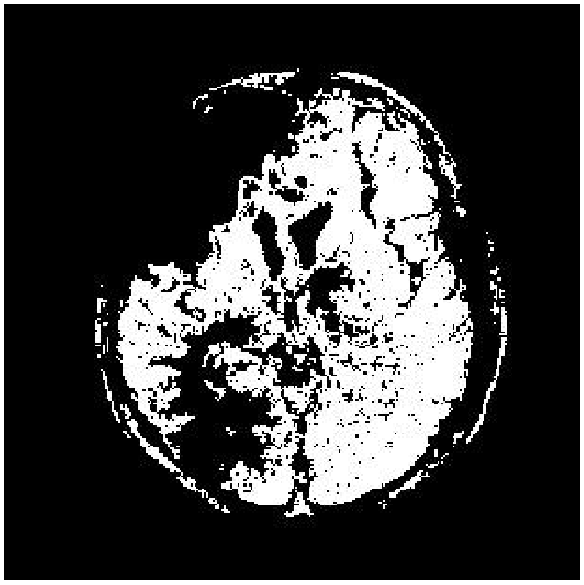

Figure 3. Binary Image, from $threshold = 0.75$

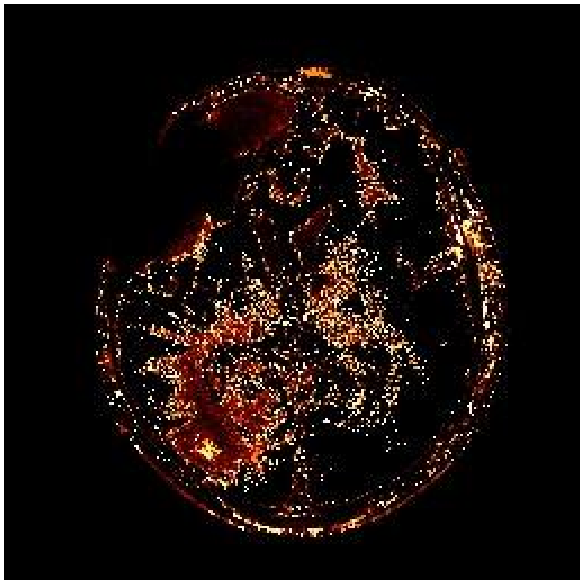

Figure 4. Fuzzy Scene after segmentation at $\{128, 128, 37\}$

Authorized licensed use limited to: SUNY Buffalo. Downloaded on October 22, 2008 at 13:08 from IEEE Xplore. Restrictions apply.

Nodes	Execution Time
1 (Serial)	27.62s
2	40.48s
3	44.18s
4	48.14s
5	55.19s
6	61.32s
7	68.17s
8	74.44s

Table 1. Execution time on MPI cluster

there were 31 million computations to be done, each taking $1\mu s$. Each one of these computations utilizes six unsigned short values from the image Scene to determine the Fuzzy Connectedness. It then changes the Scene value at that index. At each iteration, the Scene is changed, and the most recent value of the scene neighbors is needed to have correct calculation. We modified the queue to be divided into n different queues. Fig. 5 shows our method of splitting the queues. Each node $P_1, P_2, ...P_n$ gets an equal portion of the queue, $Q_{1a}, Q_{2a}, ...Q_{na}$. They then each process the data of their assigned queue and output a new queue $Q_{1b}, Q_{2b}, ...Q_{nb}$ and also a list of changes that needs to be done to the scene $S_1, S_2, ...S_n$. The Scene changes are applied, and the queues are combined for the next iteration's In Queue.

4.1. MPI

As a coarse grained parallelization, we chose to implement κFOE using MPI for cluster communication. We implemented the κFOE algorithm on the ISC cluster of the Wayne State University computing grid (www.grid.wayne.edu). The WSU ISC Cluster consists of 16 nodes, each a a dual 2.66 Ghz pentium IV processor, with 2.5GB RAM connected over a high speed low latency myrinet network and private 100bit ethernet. We tried implementing κFOE on the cluster using different parallelization methods, but none of them could overcome the communication and synchronization overhead involved. The best execution times for our MPI implementation are shown in Table 1. We can see that as we increase the number of nodes, the execution time takes longer instead of taking less time like we would expect. The reason for this is that the amount of time dedicated towards data transfer and synchronization overpowers the speedup that we should be getting. By analyzing the profiling data of a 2-node test run, we found that on a typical cycle of the algorithm approximately 50% of the time was spent communicating and synchronizing. On the first cycles and last, this ratio was even higher. By increasing the number of nodes, we also increase the ratio.

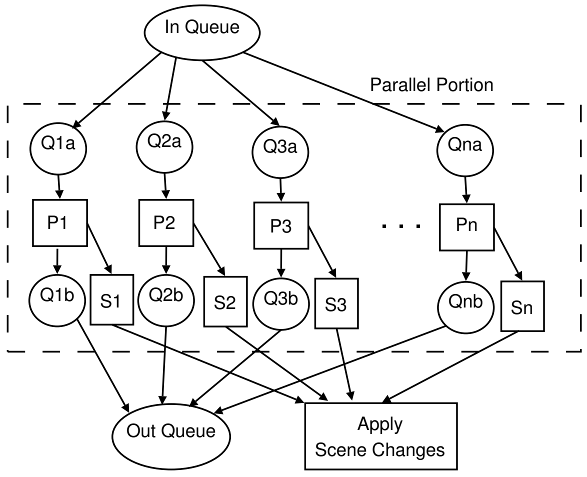

Figure 5. Parallelization Method

4.2. OpenMP

OpenMP is a directive based language that uses shared memory communication. Because of this faster method of communication we got better results than with MPI. We ran the OpenMPI version of the algorithm on a sun multiprocessor system. we show the execution timings achieved and the speedup in Fig 6. The speedups were non-linear because the ability to stop duplicate values from entering the queue is lessened as we add more parallelism, as each node is not up to date on the queue data in each other node. Other performance reductions are due to communication and synchronization overhead. The graphs show the actual speeds and the ideal parallel performance (without communication and synchronization overhead). We can see here that on four nodes, there is a two fold speedup. However, it does not scale much after that to 2.75 times speedup on eight nodes.

4.3. SSE2

Our last approach to the parallelization of the κFOE Fuzzy Connectedness algorithm was to do more than one computation at a time on each process. This involves intel's SSE2 technology. SSE2 is a set of SIMD (Single Instruction Multiple Data) instructions that adds support for 64-bit double-precision floating points. It allows for 64, 32, 16 and 8-bit integer operations on eight 128-bit MMX registers. This means that multiple computations can be done in one clock cylcle. Unfortunately, it takes time to pack and unpack the values into the registers before and after the processing is done. The amount of time it took for packing each set was longer than the performance speedup obtained. This led to no performance change at all.

Authorized licensed use limited to: SUNY Buffalo. Downloaded on October 22, 2008 at 13:08 from IEEE Xplore. Restrictions apply.

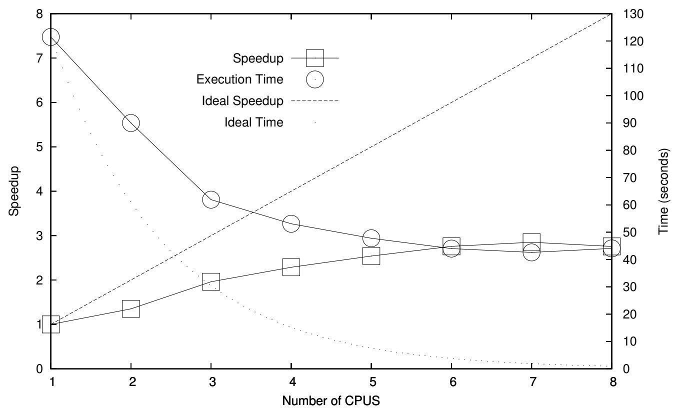

Figure 6. Execution Time vs. Num CPUS, using OpenMP

5. Conclusions

In conclusion, we have tried multiple levels of parallelism on the κFOE algorithm for fuzzy connectedness. Using optimizations we were able to get 23 times speedup. Using MPI, we were not able to increase the speed at all, because of large amounts of communication overhead. Using openMP we were able to get a speedup of 2.75 on 8 nodes over the serial version. We also found that using Intel's SSE architecture we could not increase the performance at all. This was due to the overhead of packing and upacking values into registers. We expect further improvement in OpenMP performance using load balancing. In conclusion we find that the κFOE algorithm is not very suitable for a high increase in performance via parallelization. However, we found that by making some simple coding changes to the ITK implementation we can greatly increase the performance to about fifty times using four processors.

6. Acknowledgements

We greatly appreciate the help of Dr. Joseph Landman from Scalable Informatics, LLC from discussion on parallelization to use of hardware.

References

[1] I. N. Bankman, editor. *Handbook of medical imaging*. Academic Press, Inc., Orlando, FL, USA, 2000.

[2] M. Clark, L. Hall, D. Goldgof, R. Velthuizen, F. Murtagh, and M. Silbiger. Automatic tumor segmentation using knowledge-based techniques. *IEEE Trans. on Medical Imaging*, 17(2):187–201, 1998.

[3] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[4] L. Hill and D. Hawkes. *Across-Modality Registration Using Intensity-Based Cost Functions*, chapter 34. In Bankman [1], 2000.

[5] L. Ibanez, W. Schroeder, L. Ng, and J. Cates. The itk software guid, 2003.

[6] F. Lucas-Quesada, U. Sinha, and S. Sinha. Segmentation strategies for breast tumors from dynamic mr images. *JMRI*, 6:753–763, 1996.

[7] J. Prewitt. *Object enhancement and extraction in picture processing and psychopictorics*, pages 75–149. Academic Press, New York, 1970.

[8] J. Rogowska, K. Preston, G. Hunter, L. Hamberg, K. Kwong, O. Salonen, and G. Wolf. Applications of similarity mapping in dynamic mri. *IEEE Trans. on Med. Imag.*, 14(3):480–486, 1995.

[9] A. Rosenfeld. Fuzzy digital topology. *Information and Control*, 40(1):76–87, Jan. 1979.

[10] D. Spielman, M. Sidhu, R. Herfkens, and L. Shortlife. Correlation imaging of the kidney. *International SMRM Conference*, page 373, 1995.

[11] M. Sutton, J. Bezdek, and T. Cahoon. *Image Segmentation by Fuzzy Clustering: Methods and Issues*, chapter 6. In Bankman [1], 2000.

[12] J. K. Udupa and S. Samarasekera. Fuzzy connectedness and object definition: Theory, algorithms, and applications in image segmentation. *CVGIP: Graphical Model and Image Processing*, 58(3):246–261, 1996.

[13] L. Zadeh. Fuzzy sets. *Inf. and Control*, 8:338–353, 1965.

Authorized licensed use limited to: SUNY Buffalo. Downloaded on October 22, 2008 at 13:08 from IEEE Xplore. Restrictions apply.