GPU-Based Iterative Relative Fuzzy Connectedness Image Segmentation

Ying Zhuge[a], Jayaram K. Udupa[b], Krzysztof C. Ciesielski[b,c], Alexandre X. Falcão[d], Paulo A. V. Miranda[e], and Robert W. Miller[a]

[a]Radiation Oncology Branch, National Cancer Institute, National Institutes of Health, Bethesda, MD 20892;
[b]Medical Image Processing Group, Department of Radiology, University of Pennsylvania, Philadelphia, PA 19104;
[c]Department of Mathematics, West Virginia University, Morgantown, WV 26506;
[d]Institute of Computing, University of Campinas, Campinas, SP, Brazil
[e]Department of Computer Science, IME - University of São Paulo, São Paulo, SP, Brazil

ABSTRACT

This paper presents a parallel algorithm for the top of the line among the fuzzy connectedness algorithm family, namely the iterative relative fuzzy connectedness (IRFC) segmentation method. The algorithm of IRFC, realized via image foresting transform (IFT), is implemented by using NVIDIA's compute unified device architecture (CUDA) platform for segmenting large medical image data sets. In the IRFC algorithm, there are two major computational tasks: (i) computing the fuzzy affinity relations, and (ii) computing the fuzzy connectedness relations and tracking labels for objects of interest. Both tasks are implemented as CUDA kernels, and a substantial improvement in speed for both tasks is achieved. Our experiments based on three data sets of small, medium, and large data size demonstrate the efficiency of the parallel algorithm, which achieves a speed-up factor of 2.4x, 17.0x, and 42.7x, correspondingly, for the three data sets on the NVIDIA Tesla C1060 over the implementation of the algorithm in CPU.

Keywords: Image segmentation, fuzzy connectedness, graph-based methods, GPU implementations

1. INTRODUCTION

In spite of several decades of research, image segmentation remains a challenging problem in medical image analysis.[1] Image segmentation methods based on the fuzzy connectedness (FC) framework and its extensions[2–5] have been extensively utilized in many medical applications, including multiple sclerosis (MS) lesion detection and quantification via MR imaging,[6] upper airway segmentation in children via MRI for studying obstructive sleep apnea,[7] automatic brain segmentation in MRI images with the assistance of an atlas,[8] clutter-free volume rendering and artery-vein separation in MR angiography,[9] in brain tumor delineation via MR imaging,[10] etc. The theoretical framework of FC has been compared rigorously to the popular level set and graph cut methods[11–13] and has been shown to have some theoretical and practical advantages over the latter including computational efficiency. However, when processing large image data sets, the run times for FC algorithms are still too high to meet practical clinical demands.

Several parallel implementations have been developed to improve the efficiency of the fuzzy connectedness algorithms. A parallel implementation of the scale-based FC algorithm has been developed for implementation on a cluster of workstations (COWs) by using the message passing interface (MPI) parallel-processing standard.[14] A manager-worker scheme has been used in this implementation. A speed-up factor of approximately three has been achieved on a COW with six workstations. An OpenMP-based parallel implementation of the fuzzy connectedness

Further author information:
Y.Z..: E-mail: zhugey@mail.nih.gov, Telephone: 1 301 451 7313
J.K.U.: E-mail: jay@mail.med.upenn.edu, Telephone: 1 215 662 6780
R.W.M.: Email:rwmiller@mail.nih.gov, Telephone: 1 301 451 8952.

Medical Imaging 2012: Image-Guided Procedures, Robotic Interventions, and Modeling, edited by David R. Holmes III, Kenneth H. Wong, Proc. of SPIE Vol. 8316, 831604

algorithm has been reported.[15] A speed increase of approximately five has been achieved relative to the sequential implementation on an SGI Altix 4700, an expensive shared memory multiprocessor system (MPS). A parallel implementation of the absolute FC method[2] on the NVIDIA GPU has also been developed.[16] A speedup factor of more than 10 has been achieved over an optimized CPU implementation for various clinical medical images. The successful implementation of the earliest and very basic algorithm in the FC framework led us to develop parallel implementations of more advanced FC algorithms. In this paper, we present a parallel version of the top of the line FC family of algorithms, namely the iterative relative fuzzy connectedness algorithm via image foresting transform (IRFC-IFT),[4,13,17] implemented on NVIDIA's Compute Unified Device Architecture (CUDA) platform for segmenting medical image data sets.

The paper is organized as follows. In Section 2, we first summarize the FC principles and the sequential IRFC algorithm; we then describe parallelized version of this algorithm and its implementation on the NVIDIA Tesla C1060 GPU by using CUDA in Section 3. The experimental results are presented in Section 4. Finally, we state our concluding remarks in Section 5.

2. FUZZY CONNECTEDNESS PRINCIPLES AND SEQUENTIAL ALGORITHM

In this section, we briefly describe the principles of fuzzy connectedness to make this paper self-contained. Please see the original papers for more details.[2,4,12]

We refer to a 3-D digital image as a *scene* and represent it by a pair $\mathcal{C} = (C, f)$, where C, called the *scene domain*, is a rectangular array of *voxels*, and f is the *scene intensity function* which assigns to every voxel $c \in C$ an integer intensity value in $[L, H]$.

2.1 Fuzzy adjacency and affinity

Independent of any image data, we think of the digital space defined by the voxels as having a fuzzy adjacency relation. We denote the fuzzy adjacency relation by α and the degree of adjacency assigned to a pair (c, d) of voxels by $\alpha(c, d)$. The *fuzzy adjacency* relation assigns to every pair (c, d) of voxels a value between zero and one. The closer c and d are to each other, the greater is this number. In this paper we will use the hard 6-adjacency for α.

A path p in a subset A of C is any finite seqence $\langle c_1, ..., c_k \rangle$ of voxels in A such that any consecutive voxels c_i, c_{i+1} in p are adjacent, that is, with $\alpha(c_i, c_{i+1}) > 0$. The family of all paths in A is denoted by $\mathbb{P}^A$. Voxels c and s are connected in A provided that there exists a path $p = \langle c_1, ..., c_k \rangle$ in A from c to s such that $c_1 = c$ and $c_k = s$. The family of all paths in A from c to d is denoted by $\mathbb{P}^A_{cd}$.

Fuzzy affinity, denoted κ, is a local fuzzy relation defined on the scene domain C, which assigns to every pair (c, d) of nearby voxels a strength of local hanging togetherness which has a value between zero and one. The strength $\kappa(c, d)$ between two voxels c and d depends on $\alpha(c, d)$ as well as on how similar their scene intensities $f(c)$ and $f(d)$ (homogeneity-based affinity), and intensity-derived features are for voxels c and d (object feature-based affinity). The affinity functions are discussed in detail in previous papers.[3,18] In this paper, the following functional form for κ is used:

$$\kappa(c, d) = \alpha(c, d)\sqrt{\psi(c, d)\phi(c, d)}, \tag{1}$$

where $\alpha(c, d) = 1$ if $\|c - d\| \leq 1$ and $\alpha(c, d) = 0$ if $\|c - d\| > 1$; the homogeneity based affinity $\psi(c, d)$ is given by

$$\psi(c, d) = e^{-\frac{\|f(c) - f(d)\|}{\sigma_h^2}}, \tag{2}$$

where σ_h^2 is the variance of the intensity difference for all c and d such that $\alpha(c, d) > 0$; and the object feature based affinity $\phi(c, d)$ is given by

$$\phi(c, d) = e^{-\frac{\max\{\|f(c) - m\|, \|f(d) - m\|\}^2}{\sigma_o^2}}. \tag{3}$$

For the single object case, m and σ_o^2 are related to the mean and variance of the intensity of the object that we wish to define in the scene.

2.2 Absolute fuzzy connectedness (AFC)

In this section, we will define an AFC object, denoted $P_{S\theta}$, containing a non-empty set $S \subset C$ of seeds and indicated by a threshold $\theta < 1$.

The strength of a path $p = \langle c_1, ..., c_k \rangle, k > 1$, is defined as $\mu(p) = \min\{\kappa(c_{i-1}, c_i) : 1 < i \leq k\}$, that is, the strength of the κ-weakest link of p. For $k = 1$ we associate with p the strongest possible value: $\mu(p) = 1$. For $c, d \in A \subseteq C$, the (global) *fuzzy κ-connectedness* strength in A between c and d is defined as the strength of a strongest path in A between c and d; that is, $\mu^A(c, d) = \max\{\mu(p) : p \in \mathbb{P}^A_{cd}\}$. We will refer to the function μ^A as a *connectivity measure* (on A) induced by κ. For $c \in A \subseteq C$ and a non-empty $D \subset A$, we also define $\mu^A(c, D) = \max_{d \in D} \mu^A(c, d)$. We then define the *absolute fuzzy connectedness*, AFC, object $P_{S\theta}$ as $P_{S\theta} = \{c \in C : \theta < \mu^C(c, S)\}$.

If a set of seeds S contains only one seed s, then we will write $P_{s\theta} = P_{\{s\}\theta}$. It is easy to see that $P_{S\theta}$ is a union of all objects $P_{s\theta}$ for $s \in S$, that is, $P_{S\theta} = \bigcup_{s \in S} P_{s\theta}$. One of the most important properties of the AFC objects (as well as of RFC and IRFC objects defined below) is their robustness to seeds placement. Intuitively, this property states that the FC delineation results do not change if the seeds S indicating an object are replaced by another nearby set U of seeds. The standard algorithm $\kappa\theta FOEMS$, which, given a scene $\mathcal{C} = (C, f)$, a set $S \subset C$ of seeds indicating the object, and a threshold $\theta < 1$, returns the AFC object $P_{S\theta}$, is described in.[2]

2.3 Relative fuzzy connectedness (RFC)

Instead of defining an object on its own based on connectivity strength, in RFC, all (important) objects in the scene are considered in defining each object. The objects compete among themselves in terms of connectivity strength to win over voxels in the scene and to have them as their members. In this competition, every pair of voxels in the scene will have a strength of connectedness in each object. The object in which this strength is highest will claim membership of the voxels. This approach of fuzzy object definition, using relative strengths of connectedness, eliminates the need for the threshold θ. All specified objects are defined simultaneously in this approach.

Beside a set S of seeds indicating the object, let T be a non-empty set of seeds, disjoint with S, indicating the background (co-object). The actual RFC object $P_{S,T}$ is defined via competition of seed sets S and T for attracting a given voxel $c \in C$: $P_{S,T} = \{c \in C : \mu^C(c, S) > \mu^C(c, T)\}$. Notice that $P_{S,T} = \bigcup_{s \in S} P_{\{s\},T}$, since $P_{S,T} = \{c \in C : (\exists s \in S)\mu^C(c, s) > \mu^C(c, T)\} = \bigcup_{s \in S} P_{\{s\},T}$, as $\mu^C(c, S) = \max_{s \in S} \mu^C(c, s)$.

The above RFC delineation procedure can be easily generalized to $m > 2$ objects. For any scene $\mathcal{C} = (C, f)$, let $\mathcal{S} = \{S_1, \ldots, S_m\}$ denote sets of disjoint non-empty seeds in C, where each S_i is specified for an associated object P_i. If, for each i, we put $T_i = (\bigcup_{j=1}^m S_j) \backslash S_i$, then the RFC segmentation is defined as a family $\mathcal{P} = \{P_{S_i,T_i} : i = 1, ..., m\}$. It is easy to see that the different objects in $\mathcal{P}$ are disjoint.

It is important to note that we confined ourselves to a single affinity for different objects. Indeed, single affinity is a necessary condition to insure nice properties of the relative fuzzy connectedness to hold.[4] Here we describe how different object-oriented affinities may be combined into a single affinity. Let $\kappa_1, \kappa_2, \ldots, \kappa_m$ be affinities specified for the m object regions, respectively. These affinities may be combined into a single affinity κ that retains as much as possible the individual object-specific information.[4] In this paper, we use the fuzzy union of the individual affinities. That is,

$$\kappa(c, d) = \max_i \kappa_i(c, d). \tag{4}$$

2.4 Iterative relative fuzzy connectedness (IRFC)

The RFC segmentation $\mathcal{P} = \{P_{S_i,T_i} : i = 1, ..., m\}$ of a scene, associated with a family of seeds $\mathcal{S} = \{S_1, \ldots, S_m\}$, can still leave a set $B = B_{\mathcal{P}}$ of voxels outside of any objects wherein the strengths of connectedness are equal with respect to the seeds. The *iterative relative fuzzy connectedness* (IRFC) paradigm was proposed to circumvent this problem.[4,5] It is an iterative refinement strategy that imposes additional constraints based on the results from previous iterations. We treat the RFC segmentation objects P_{S_i,T_i} as the first iteration $P^1_{S_i,T_i}$ of the final segmentation, while the next iteration is to redistribute some of the voxels $c \in B_{\mathcal{P}}$, for which $\mu^C(c, S_i) = \mu^C(c, T_i)$

for some i. Such a tie can be resolved if the strongest paths justifying $\mu^C(c, S_i)$ and $\mu^C(c, T_i)$ cannot pass through the voxels already assigned to another object. That is, we like to add voxels from the set $P^* = \{c \in B : \mu^{B \cup P_{S_i,T_i}}(c, S_i) > \mu^{B \cup P_{S_j,T_i}}(c, S_j)$ for all $j \neq i$, to a new generation $P^{k+1}_{S_i,T_i}$ of $P^k_{S_i,T_i}, k = 1, 2, \ldots, \infty$.

An efficient implementation of the IRFC approach that has been recently proposed,[13] based on the Image Foresting Transform (IFT),[17,19,20] is summarized below.

For a given scene $\mathcal{C} = (C, f)$ and fuzzy adjacency α, *a spanning forest on* $\mathcal{C}$ is any family $\mathbb{F}$ of directed paths such that: (1) for every $c \in C$ there is a unique $p_c = \langle c_1, \ldots, c_k \rangle \in \mathbb{F}$ with $c_k = c$, and (2) an initial segment of any path in $\mathbb{F}$ is also in $\mathbb{F}$. A spanning forest $\mathbb{F}$ is often identified with its predecessor map $Pr_{\mathbb{F}} : C \to C \cup \{nil\}$ that is defined as follows: if $p_c = \langle c_1, ..., c_k \rangle \in \mathbb{F}$ is a unique path with $c_k = c$, then $Pr_{\mathbb{F}}(c) = nil$ for $k = 1$ and $Pr_{\mathbb{F}}(c) = c_{k-1}$ for $k > 1$. Let $S_{\mathbb{F}} = \{c \in C : Pr_{\mathbb{F}}(c) = nil\}$. A root map $R_{\mathbb{F}} : C \to S_{\mathbb{F}}$ associated with the spanning forest $\mathbb{F}$ is defined as $R_{\mathbb{F}}(c) = c_1$, for any $c \in C$, where $p_c = \langle c_1, ..., c_k \rangle$ is the unique path in $\mathbb{F}$ which terminates at c. For any seed set $S \in C$, a path $p = \langle c_1, ..., c_k \rangle$ is optimal (with respect to S and a path cost function μ) provided that $c_1 \in S$ and $\mu(p) = \mu^C(c_k, S)$. A spanning forest $\mathbb{F}$ is optimal if every path in $\mathbb{F}$ is optimal.

The *Image foresting transform* (IFT)[19] takes a given scene $\mathcal{C}$, an optimal path cost function (connectivity measure μ^C), and an adjacency relation α, and then produces an optimum spanning forest. During the process, a cost map and root map are also built.

For image segmentation, let $\mathcal{S} = \{S_1, \ldots, S_m\}$ be sets of disjoint non-empty seeds in C where S_i is specified for an associated object P_i. We define $P(S_i, \mathbb{F})$ as the set of all $c \in C$ with $R_{\mathbb{F}}(c) \in S_i$. A label map is a function defined as $\lambda : C \to \{l_1, \ldots, l_m\}$. From the segmentation point of view, the same label l_i is assigned to all $c \in P(S_i, \mathbb{F})$, that is, $\lambda(c) = \lambda(R_{\mathbb{F}}(c)) = l_i$.

The sequential algorithm of IRFC-IFT[13] on CPU, which we choose to parallelize, is presented below for the special case of object and background ($m = 2$).

Algorithm 1 IRFC-IFT

Input: Affinity function κ; Non-empty sets $S, T \subset C$ of seeds, indicating object and background, respectively;
Output: Functions $h : C \to \{-1\} \cup [0, 1]$ approximating $\mu^C(\cdot, S \cup T)$; Predecessor map Pr; Root map R; A labeling map $\lambda : C \to \{0, 1\}$;
Auxiliary: An ordered priority queue Q.
begin

```
Set h(c) ← 1, R(c) ← c, Pr(c) ← nil for all c ∈ S ∪ T; h(c) ← −1, R(c) ← c, Pr(c) ← c for all c ∉ (S ∪ T);
Set λ(c) ← 0 for all c ∈ T, and λ(c) ← 1 for all c ∈ S;
Insert c in Q for all c ∈ S ∪ T;
while Q is not empty do
   Remove a voxel c from Q, such that h(c) is maximal;
   for every d such that κ(c, d) > 0 do
      if h(d) < min{h(c), κ(c, d)} then
         Set h(d) ← min{h(c), κ(c, d)};
         Set R(d) ← R(c); Pr(d) ← c; λ(d) ← λ(c);
         if d ∈ Q then
            update the location of d in Q;
         else
            insert d in Q;
         end if
      end if
   end for
end while
```

end

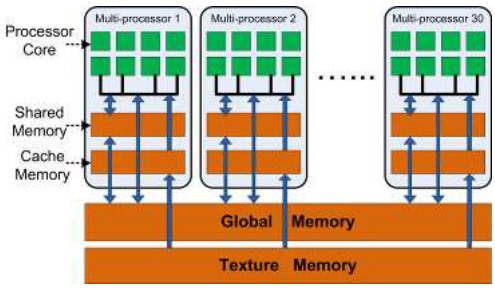


Figure 1. NVIDIA GPU hardware architecture.

3. GPU IMPLEMENTATION

In this section, we first briefly describe the NVIDIA GPU hardware architecture and the CUDA programming model. For a full description on NVIDIA GPU and CUDA, readers are referred to the CUDA programming guide.[21] We then describe how we implement the IRFC-IFT algorithm using the CUDA model. Note that, because of the limited size of GPU device memory and latency of global memory access, the predecessor map and root map in the algorithm of IRFC-IFT are not implemented in the GPU parallelized program. For comparison purposes, these two maps are also not built in the sequential program.

3.1 NVIDIA GPU architecture

The underlying hardware architecture of a NVIDIA GPU is illustrated in Figure 1. The NVIDIA Tesla C1060 GPU is used as an example to provide a brief overview of the architecture. The Tesla C1060 GPU has 240 processing cores with a clock rate of 1.3GHz for each core, delivering nearly 1 Tera FLOPS of computational power. To support an intuitive and flexible programming environment to access such computing power, NIVIDA provides the CUDA framework,[21] which is based on a C-language model. CUDA enables the generation and management of a massive number of processing threads, which can be executed in parallel on GPU cores with efficient hardware scheduling.

The 240 cores of Tesla C1060 GPU are grouped into 30 multi-processors. Each multi-processor has 8 processing cores, organized in a SIMD (Single Instruction Multiple Data) fashion. Each core has its own register file and arithmetic logic unit which allows it to accomplish a specific computational task. The Tesla C1060 has 4 GB of on-board device memory, which can be used as read-only texture memory or read-write global memory. The GPU device memory features very high bandwidth, recorded at 102 GB per second, but it suffers from high access latency. In each multi-processor unit, there is 16 KB of user-controlled L1 cache, called shared memory. If it is used efficiently, it can be employed to hide the latency in global memory access.

The CUDA programming model is based on concurrently executed threads. CUDA manages threads in a hierarchical structure. Threads are grouped into a thread block, and thread blocks are grouped into a grid. All threads in one grid share the same functionality, as they are executing the same kernel code. Each thread block is mapped on to one multi-processor unit, and threads in each block are scheduled to run on 8 processing cores of the multi-processor unit, using a scheduling unit of 32-thread warp. Since the threads in a block are executed on the same multi-processor, they can use the same shared memory space for data communication. On the other hand, the threads between different blocks can communicate only through the low-speed global memory.

3.2 CUDA implementation of IRFC

In CUDA, programs are expressed as kernels. In order to map a sequential algorithm to the CUDA programming environment, developers should identify data-parallel portions of the application and isolate them as CUDA kernels. In the IRFC-IFT algorithm, there are two major computational tasks: (C1) computing the fuzzy affinity relations, and (C2) computing the fuzzy connectivity measure and tracking labels for objects of interest. We shall refer to (C1) as "affinity computation" and (C2) as "tracking" connectivity. These two tasks are implemented as CUDA kernels, and a significant improvement in speed for both tasks is achieved as a result.

1) Affinity computation kernel: The CUDA implementation of fuzzy affinity computation is straightforward. The fuzzy affinity computation of every pair (c, d) of voxels where $\alpha(c, d)$ is greater than zero is totally independent

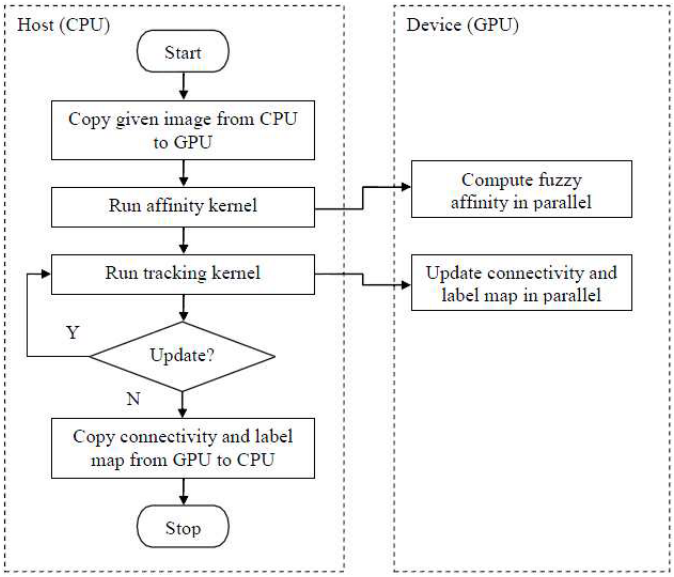


Figure 2. The flow chart of the CUDA implementation.

of other pairs of voxels. Thus for the pair (c, d), one thread is assigned to compute corresponding $\psi(c, d)$ and $\phi(c, d)$ in Equations (2) and (3), and the fuzzy affinity $\kappa(c, d)$ result of Equation (1) is written to the specific allocated GPU device memory.

2) Tracking kernel: The sequential IRFC-IFT algorithm is essentially Dijkstra's algorithm, with a slight modification for multiple sources. Parallel implementation of the Dijkstra's algorithm is quite challenging.[22,23] Particulary, the priority queue structure in the algorithm is very hard to implement on the GPU. In this paper, we use an alternative strategy to implement the functionality of the priority queue. The idea here is to try parallelize Step 5 in the sequential IRFC-IFT algorithm. That is, all voxels having maximum connectivity measure are independently processed by different threads simultaneously. We exploit the computing capability of concurrent executed threads on the GPU. For any $c \in C$, the connectivity measure $h(c)$ is mapped to an integer value which is used as an index in the queue. A global variable g is maintained on the GPU to track the maximal index value in the queue. Each GPU thread independently checks the connectivity value of the voxel it processes and compares with the maximal index. Only those threads associated with maximal index are allowed to update the connectivity and label map information of neighboring voxels. The atomic read/write operations in the device memory are used for concurrent memory access. The flow chart of our CUDA implementation is illustrated in Figure 2, and the algorithm is presented below.

Algorithm 2 CUDA-IRFC-IFT

Input: Given scene $\mathcal{C} = (C, f)$; Non-empty sets $S, T \subset C$ of seeds, indicating object and background, respectively;
Output: Functions $h : C \to \{-1\} \cup [0, 1]$ approximating $\mu^C(\cdot, S \cup T)$; A labeling map $\lambda : C \to \{0, 1\}$;
Auxiliary: A function $\eta : C \to \{0, 1\}$ indicating which voxels of C are in the queue; A global index variable g.
begin

1: Set $h(c) \leftarrow 1, \eta(c) \leftarrow 1$ for all $c \in S \cup T$ and $h(c) \leftarrow -1, \eta(c) \leftarrow 0$ for all $c \notin (S \cup T)$;
2: Set $\lambda(c) \leftarrow 0$ for all $c \in T$, and $\lambda(c) \leftarrow 1$ for all $c \in S$, $g \leftarrow 1$;
3: Copy h and λ from CPU to GPU;
4: Invoke AFFINITY-KERNEL on grid to compute fuzzy affinity κ;
5: **while** $g > 0$ **do**
6: Invoke TRACKING-KERNEL(h, λ, g);
7: Transfer g back to CPU;
8: **end while**
9: Copy h and λ from GPU to CPU;

end

The above algorithm is executed on the host (CPU) side while the kernels are executed on the device (GPU) side, as shown in Figure 2. The values of h, κ, λ and η are stored in GPU global memory, which are accessible by all threads on GPU. As explained above, the different threads that meet the criteria operate on voxels for updating connectivity and label map information simultaneously. In one invocation, they all update connectivity information as much as they can on the voxels in their purview. The CPU determines if there are any voxels left in (virtual) queue through the global index value. If so, the kernel is invoked again with updated information of g and η. The CPU terminates the run of the algorithm when the global index g has value of zero. The value of g is updated in the following manner. A variable l_i is maintained in shared memory of each block B_i, which indicates the local maximum index of voxels processed by threads in B_i. When a thread in B_i updates connectivity and label map information, the value of l_i is updated accordingly. After all threads in B_i finish their work, the value of g is then updated in terms of l_i. We take this strategy to update the value of g because the memory access in shared memory is much faster than that in global memory. The affinity kernel and the tracking kernel algorithms are presented below.

Algorithm 3 AFFINITY-KERNEL

1: Compute thread index $t(id)$;
2: **for** each voxel c processed by $t(id)$ **do**
3: **for** all d such that $\alpha(c, d) > 0$ **do**
4: Compute affinity $\kappa(c, d)$;
5: Write $\kappa(c, d)$ to corresponding GPU memory;
6: **end for**
7: **end for**

Note that in Algorithm 3, each pair (c, d) is considered only once. Thus different threads independently compute affinities for different pairs of voxels.

The algorithm CUDA-IRFC-IFT is an iterative procedure. At the first iteration, only those threads which process the voxels $c \in S \cup T$ are active. TRACKING-KERNEL is called to update the connectivity measure h and the label map λ. More threads will be involved and become active for the connectivity and label map information update at the successive iterations. Because of the limited communication capability among threads from different blocks, the CPU side needs to collect the global index information from each block on GPU that is updated by each thread in the block through shared memory, and it decides when to terminate calling the TRACKING-KERNEL. Each thread checks the connectivity and η value of each voxel under its control to see if it is allowed to operate on neighboring voxels for updating connectivity and label map information. Note in lines 6 and 7 of the Algorithm TRACKING-KERNEL, atomic operation was used for consistency, because update

Algorithm 4 TRACKING-KERNEL(h, λ, g)

```
Compute thread index t(id);
for each voxel c processed by t(id) such that h(c) = g do
   Set η(c) ← 0;
   for all e such that κ(c, e) > 0 do
      if h(e) < min{h(c), κ(c, e)} then
         Set h(e) ← min{h(c), κ(c, e)},λ(e) ← λ(c);
         Set η(e) ← 1;
      end if
   end for
end for
update g;
```

operations for one voxel by multiple threads might happen simultaneously. The atomic operation insures that no conflict occurs. When voxel c has been processed by one thread, it doesn't need to be further processed in the next iteration. The algorithm CUDA-IRFC-IFT terminates when all voxels are processed.

4. EXPERIMENTAL RESULTS

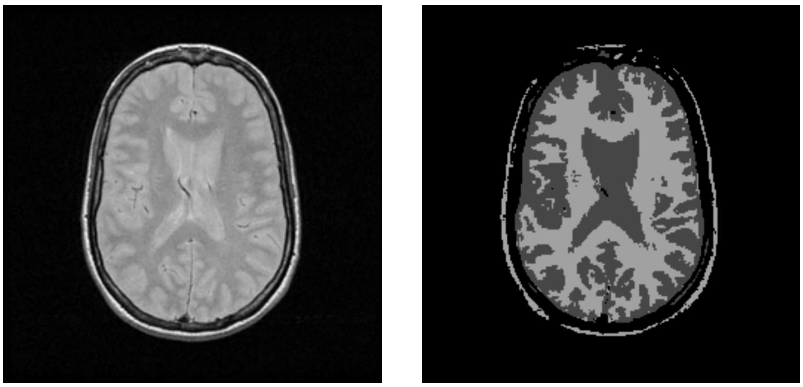
In this section, the running times of the GPU and CPU implementations of the IRFC-IFT algorithm are compared for image data of different sizes. The CPU version of FC is implemented in C++. The computer used is a DELL PRECISION T7400 with a quad-core 2.66 GHz Intel Xeon CPU. It runs Windows XP and has 2 GB of main memory. The GPU used is the NVIDIA Tesla C1060 with 240 processing cores and 4 GB device memory. CUDA SDK 3.2 is used in our GPU implementation. Three image data sets – small, medium, and large – are utilized to test the performance of the GPU and CPU implementations. Table 1 lists the image data set information and shows the performance of the GPU implementation versus the CPU implementation. A speed-up factor of 2.4x, 17.0x, and 42.7x, respectively, has been achieved, for the three data sets over the CPU implementation. It is noted that the segmentation results produced from both GPU and CPU implementations are identical. Here, the speed-up factor is defined as ts/tp, where ts and tp are the times taken for the sequential and parallel implementations, respectively. It seems that the larger the size of the testing data set, the more speedup we can achieve. This is mainly because for larger data sets, there are more voxels might having maximum index value such that more threads will be involved for update operations in one iteration. Note that in all of our experiments, we try to separate the object of interest (foreground) from the other co-object (background).

Table 1. Data set information and performance of the GPU implementation with respect to the optimal CPU implementation.

Dataset	*Small*	*Medium*	*Large*
Protocol	PD MRI	T1 MRI	T1 MRI
Scene domain	$256 \times 256 \times 46$	$256 \times 256 \times 124$	$512 \times 512 \times 192$
Voxel size	$0.98 \times 0.98 \times 3.0$	$0.94 \times 0.94 \times 1.5$	$0.5 \times 0.5 \times 1.0$
CPU time (s)	13.45	293.62	10123.48
GPU time (s)	5.55	17.30	236.98
Speed-up	2.4	17.0	42.7

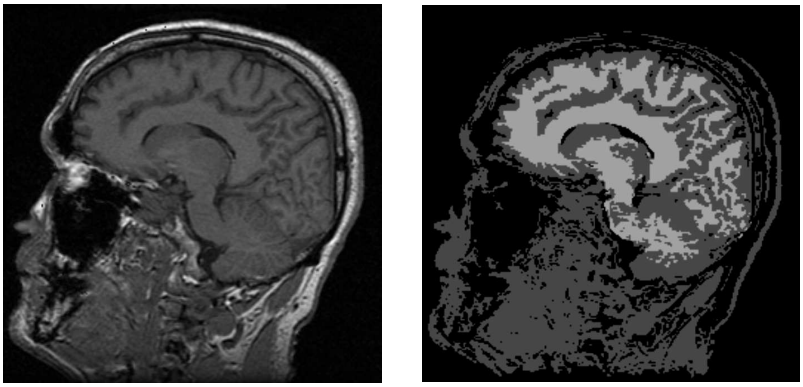
Figure 3 shows the example of the small size data set, which comes from MRI of the head of a clinically normal human subject. A fast spin-echo dual-echo protocol is used. Figure 3(a) shows one slice of the original PD-weighted scene, Figure 3(b) shows corresponding label map that separates the white matter from the other tissues in the brain.

Figure 4 shows the example of the medium size data set, which is a T1-weighted MRI scene of the head of a clinically normal human subject. The spoiled gradient recalled (SPGR) acquisition was used. This data set was



(a) (b)

Figure 3. (a) A slice of PD-weighted MRI scene from the small data set, and (b) the final label map.



(a) (b)

Figure 4. (a) A slice of T1-weighted MRI scene from the medium data set, and (b) the final label map.

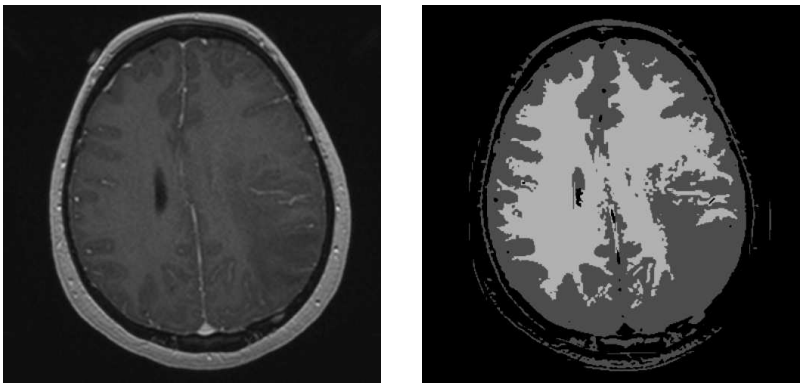
obtained from the web site of National Alliance for Medical Image Computing (http://www.na-mic.org). Figure 4(a) shows one slice of the original scene, and Figure 4(b) shows the separated white matter label from other tissues.

Figure 5 shows the example of the large size data set, which is a T1-weighted MRI scene of the head of a patient with brain tumor. Figure 5(a) shows one slice of the original scene, and Figure 5(b) shows the corresponding label map depicting the white matter label and that of the other tissues.

5. CONCLUDING REMARKS

Recently, clinical radiological research and practice are becoming increasingly quantitative. Further, images continue to increase in size and volume. For quantitative radiology to become practical, it is crucial that image segmentation algorithms and their implementations are rapid and yield practical run time on very large data sets. This paper describes an example of a practical and cost-effective solution to the problem.

We developed a parallel algorithm of the iterative relative fuzzy connectedness algorithm via the image foresting transform (IRFC-IFT), the top of the line among the fuzzy connectedness algorithm family, on the NVIDIA GPUs, which are far more cost and speed-effective than both COWs and multiprocessing systems. The parallel implementation achieves speed increases by factors ranging from 2.4x to 42.7x on the Tesla C1060 GPU over an optimized CPU implementation for three image data sets with a wide range of sizes. A near-interactive speed of segmentation has been achieved, even for the large data set. For some specific applications, several free parameters (e.g. fuzzy affinity parameter) in fuzzy connected image segmentation might be difficult to optimize. The interactive speed of segmentation could give users immediate feedback on parameter settings; thus allowing them



(a) (b)

Figure 5. (a) A slice of T1-weighted MRI scene from the large data set, and (b) the final label map.

to fine-tune free parameters and produce more accurate segmentation results. In the current implementation, the tracking kernel is iteratively launched which is computationally expensive. The performance of the parallel implementation could be further improved by devising a better mechanism for inter-block communication on the GPU. In addition, other parallel implementation methods of the Dijkstra's algorithm need to be investigated.[24] On the other hand, a more efficient (near-linear time) implementation of the sequential IRFC-IFT algorithm has also been available,[25] and so the actual speed up factor may differ from those reported in this work. However, we are certain that such GPU implementations will play a crucial role in automatic anatomy recognition in radiology.

REFERENCES

[1] Duncan, J. S. and Ayache, N., "Medical image analysis: Progress over two decades and the challenges ahead," *EEE Transactions on Pattern Analysis and Machine Intelligence* **22**, 85–105 (2000).

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

[3] Saha, P. K., Udupa, J. K., and Odhner, D., "Scale-based fuzzy connectedness image segmentation: Theory, algorithms, and validation," *Computer Vision and Image Understanding* **77**, 145–174 (2000).

[4] Udupa, J. K., Saha, P. K., and Lotufo, R. A., "Relative fuzzy connectedness and object definition: Theory, algorithms, and applications in image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **24**, 1485–1500 (2002).

[5] Ciesielski, K. C., Udupa, J. K., Saha, P. K., and Zhuge, Y., "Iterative relative fuzzy connectedness for multiple objects with multiple seeds," *Computer Vision and Image Understanding* **107**, 160–182 (2007).

[6] Udupa, J. K., Wei, L., Samarasekera, S., Miki, Y., Buchem, M. A., and Grossman, R. I., "Multiple sclerosis lesion quantification using fuzzy connectedness principles," *IEEE Transaction on Medical Imaging* **16(5)**, 598–609 (1997).

[7] Liu, J., Udupa, J. K., Odhner, D., McDonough, J. M., and Arens, R., "System for upper airway segmentation and measurement with mr imaging and fuzzy connectedness," *Academic Radiology* **10(1)**, 13–24 (2003).

[8] Zhou, Y. and Bai, J., "Atlas-based fuzzy connectedness segmentation and intensity non-uniformity correction applied to brain mri," *IEEE Trans. Biomedical Engineering* **54**, 121–129 (2007).

[9] Lei, T., Udupa, J. K., Saha, P. K., , and Odhner, D., "Artery-vein separation via mra–an image processing approach," *IEEE Transactions on Medical Imaging* **20(8)**, 689–703 (2001).

[10] Moonis, G., Liu, J., Udupa, J. K., and Hackney, D., "Estimation of tumor volume using fuzzy connectedness segmentation of mri," *American Journal of Neuroradiology* **23(3)**, 356–363 (2002).

[11] Ciesielski, K. C. and Udupa, J. K., "A framework for comparing different image segmentation methods and its use in studying equivalences between level set and fuzzy connectedness frameworks," *Computer Vision and Image Understanding* **115**, 721–734 (2011).

[12] Ciesielski, K. C. and Udupa, J. K., "Region-based segmentation: fuzzy connectedness, graph cut, and other related algorithms," in [*Biomedical Image Processing*], Deserno, T. M., ed., 251–278, Springer-Verlag, Berlin Heidelberg (2011).

[13] Ciesielski, K. C., Udupa, J. K., Miranda, P. A., and Falcao, A. X., "Comparison of fuzzy connectedness and graph cut segmentation algorithms," in [*Medical Imaging: Image Processing*], Dawant, B. M. and Haynor, D. R., eds., *Proc. SPIE* **7962**, 796203 (2011).

[14] Grevera, G., Udupa, J. K., Odhner, D., Zhuge, Y., Souza, A., Mishra, S., and Iwanaga, T., "Cavass - a computer assisted visualization and analysis software system," *Journal of Digital Imaging* **20**, 101–118 (2007).

[15] Carvalho, B. M. and Herman, G. T., "Parallel fuzzy segmentation of multiple objects," *Int J Imaging Syst. Technol.* **18**, 336–344 (2008).

[16] Zhuge, Y., Cao, Y., Udupa, J. K., and Miller, R. W., "Parallel fuzzy connected image segmentation on gpu," *Medical Physics* **38**, 4365–4371 (2011).

[17] Miranda, P. A. and Falcao, A. X., "Links between image segmentation based on optimum-path forest and minimum cut in graph," *Journal of Mathematical Imaging and Vision* **35**, 128–142 (2009).

[18] Ciesielski, K. C. and Udupa, J. K., "Affinity functions in fuzzy connectedness based image segmentation I: Equivalence of affinities," *Computer Vision and Image Understanding* **114**, 146–154 (2010).

[19] Falcao, A. X., Stolfi, J., and Lotufo, R. A., "The image foresting transforms: Theory, algorithms and applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **26**, 19–29 (2004).

[20] Falcao, A. X. and Bergo, F. P. G., "Interactive volume segmentation with differential image foresting transforms," *IEEE Transactions on Medical Imaging* **23(9)**, 1100–1108 (2004).

[21] NVIDIA, [*NVIDIA CUDA programming guide 3.0*], NVIDIA Corporation (February 2010). http://developer.nvidia.com/cuda.

[22] Nepomniaschaya, A. S. and Dvoskina, M. A., "A simple implementation of dijkstra's shortest path algorithm on associative parallel processors," *Fundam. Info.* **43**, 227243 (2000).

[23] Harish, P. and Narayanan, P. J., "Accelerating large graph algorithms on the gpu using cuda," in [*Proceedings of the 14th international conference on High performance computing*], *HiPC'07*, 197–208, Springer-Verlag, Berlin, Heidelberg (2007).

[24] Meyer, U. and Sanders, P., "$\triangle$-stepping: A parallel single source shortest path algorithm," in [*In ESA98: Proceedings of the 6th Annual European Symposium on Algorithms*], 393–404, Springer-Verlag (1998).

[25] Ciesielski, K. C., Udupa, J. K., Falcao, A. X., and Miranda, P. A., "Fuzzy connectedness image segmentation in graph cut formulation: A linear-time algorithm and a comparative analysis," *Journal of Mathematical Imaging and Vision* (2012). Accepted.