GPU-Based Relative Fuzzy Connectedness Image Segmentation

Ying Zhuge[a)]

Radiation Oncology Branch, National Cancer Institute, National Institutes of Health, Bethesda, MD 20892

Krzysztof C. Ciesielski[b)]

Department of Mathematics, West Virginia University, Morgantown, WV 26506 and Medical Image Processing Group, Department of Radiology, University of Pennsylvania, Philadelphia, PA 19104

Jayaram K. Udupa[c)]

Medical Image Processing Group, Department of Radiology, University of Pennsylvania, Philadelphia, PA 19104

Robert W. Miller[d)]

Radiation Oncology Branch, National Cancer Institute, National Institutes of Health, Bethesda, MD 20892, USA

(Dated: 28 September 2012)

Purpose: 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. The purpose of this paper is to present a parallel version of an algorithm that belongs to the family of Fuzzy Connectedness (FC) algorithms, to achieve an interactive speed for segmenting large medical image data sets.

Methods: The most common FC segmentations, optimizing an ℓ_∞-based energy, are known as Relative Fuzzy Connectedness (RFC) and Iterative Relative Fuzzy Connectedness (IRFC). Both RFC and IRFC objects (of which IRFC contains RFC) can be found via linear time algorithms, linear with respect to the image size. The new algorithm, P-ORFC (for Parallel Optimal RFC), which is implemented by using NVIDIA's Compute Unified Device Architecture (CUDA) platform, considerably improves the computational speed of the above mentioned CPU based IRFC algorithm.

Results: Experiments based on four data sets of small, medium, large, and super data size, achieved speed-up factors of 32.8×, 22.9×, 20.9×, and 17.5×, correspondingly, on the NVIDIA Tesla C1060 platform. Although the output of P-ORFC need not precisely match that of IRFC output, it is very close to it and, as we prove, always lies between the RFC and IRFC objects.

Conclusions: A parallel version of a top-of-the-line algorithm in the family of FC has been developed on the NVIDIA GPUs. An interactive speed of segmentation has been achieved, even for the largest medical image data set. Such GPU implementations may play a crucial role in automatic anatomy recognition in clinical radiology.

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

I. INTRODUCTION

In spite of several decades of research, image segmentation remains a challenging problem in medical image analysis[1]. Graph-based algorithms make up a prominent class of purely image based segmentation algorithms. Among these, as summarized in[2] and[3], the graph cut methods[4] employ the ℓ_1-norm while random walker[5] optimizes the ℓ_2-norm. The fuzzy connectedness[6–9] and the shortest path (geodesics)[10,11] use the ℓ_∞-norm. The fuzzy connectedness (FC) framework has some unique properties such as robustness to seed points and computational speed. As such, it has been extensively utilized in many medical applications, including multiple sclerosis lesion detection and quantification via Magnetic Resonance Imaging (MRI)[12], upper airway segmentation via MRI for studying pediatric obstructive sleep apnea[13], automatic brain segmentation in MRI with the assistance of an atlas[14], clutter-free volume rendering and artery-vein separation in MR angiography[15], in brain tumor delineation via MRI[16], etc. Several different forms of the mathematical definition of FC and the associated algorithms are reported in the literature[17–20]. The theoretical framework of FC has been compared rigorously to the popular level set and graph cut methods[21–24] 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.

Both graph cut and random walk methods have been implemented on the GPU[3],[25–27], achieving the speed improvement, with respect to their CPU counterparts and using different platforms, of the order of 0.7–19 fold. Several parallel implementations have been developed to improve the efficiency of the FC algorithms. A parallel implementation of the scale-based FC algorithm has been developed for implementation on a Cluster of Workstations (COW) by using the message passing interface (MPI) parallel-processing standard[28]. 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 FC algorithm has been reported in[29]. 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. A parallel implementation of the absolute FC method, AFC[6], on the NVIDIA GPU has also been developed achieving a speed increase of more than 10 over the CPU version[30].

This paper focusses on a GPU implementation of an logarithm optimizing the ℓ_∞-based energy $\varepsilon^{\max}$, in the ℓ_p-norm formulation proposed in[2]. The energy optimization is what truly distinguishes this work from that presented in[30], as AFC objects have no build-in energy optimization factor.

The objects optimizing the energy $\varepsilon^{\max}$ are carefully discussed in[24]. The most efficient currently existing CPU algorithm returning an $\varepsilon^{\max}$-optimizer, $\mathrm{GC}_{\max}$ from[24] (compare also[31]), is a version of Dijkstra algorithm. It returns, in a linear time with respect to the image size, an object known as the Iterative Relative FC, IRFC, object. (Compare[8,9].) The smallest (in the sense of inclusion) among all $\varepsilon^{\max}$-optimizers always exists and is called the Relative FC, RFC, object. The RFC object can be strictly smaller than the IRFC object. The RFC object can also be found in a linear time with respect to the image size (by a simple modification of the $\mathrm{GC}_{\max}$ algorithm); however such modification runs slower (by a factor of 2) than the original $\mathrm{GC}_{\max}$ algorithm.

The new algorithm, called P-ORFC (for Parallel Optimal Relative FC), is implemented on NVIDIA's Compute Unified Device Architecture (CUDA) platform for segmenting medical image data sets and achieves a speedup factor of 17-32 over the top-of-the-line CPU algorithm $\mathrm{GC}_{\max}$ discussed above. The output of P-ORFC is close to the IRFC object and, as we prove here, always lies between the RFC and IRFC objects; however, it may be strictly between these two objects. Certainly, it would be more desirable to have a fast GPU-based algorithm that returns precisely the known IRFC (or RFC) object. This was our initial idea, as reported in the conference proceedings version of this paper[32], which describes the CPU-based implementation CUDA-IRFC-IFT of the $\mathrm{GC}_{\max}$ algorithm. However, the experiments of segmenting large size image data with CUDA-IRFC-IFT (not included in the study reported in[32]) did not achieve the performance substantially better the CPU based $\mathrm{GC}_{\max}$ algorithm. Therefore, for the study presented in this paper, we redesigned the CUDA-IRFC-IFT algorithm to the new algorithm P-ORFC, which clearly outperforms (with respect to running time) the $\mathrm{GC}_{\max}$ and CUDA-IRFC-IFT algorithms, while it has a similar quality of the output.

The paper is organized as follows. In Section II.A, we first summarize the principles of fuzzy connectedness and describe some of the CPU-based FC algorithms. Section II.B describes our new algorithm and its implementation on the NVIDIA Tesla C1060 GPU by using CUDA. It also contains the proof of its correctness. The experimental results are presented in Section III. Finally, we state our concluding remarks in Section IV.

II. MATERIALS AND METHODS

II.A. Principles of Fuzzy Connectedness

In this section, the principles of *Fuzzy Connectedness*, FC, are briefly summarized (following[6,8,9,22,24]), to make this paper self-contained.

II.A.1. Digital image and its scene

We will identify a *digital image* $I = \langle C, f \rangle$ with its *intensity function* $f \colon C \to \mathbb{R}^\ell$, that is, a map from its *domain* — a finite set C, whose elements will be referred to as *spels*, short for space elements — into $\mathbb{R}^\ell$. The value $f(c)$ of f at c represents image intensity, an ℓ-dimensional vector, at this spel.

The domain C of the image comes with an adjacency relation function $\alpha \colon C \times C \to \{0, 1\}$, independent of the image intensity function, and the structure $\mathcal{C} = \langle C, \alpha \rangle$ is referred to as a *digital scene*. The spels $c, d \in C$ are *adjacent*, when $\alpha(c, d) > 0$. Intuitively, the adjacent spels are defined as spatially close to each other and are considered to be connected (in a topological and graph-theoretical sense). Recall[22,24] that a scene $\mathcal{C}$ is often identified with the directed graph $G = \langle C, E \rangle$, where the sets E of graph directed edges are defined as $\{\langle c, d \rangle \in C \times C \colon \alpha(c, d) > 0\}$. (Since α is usually a symmetric function, the graph can be also naturally identified with a non-directed graph.)

In the experimental part of this paper, Sec. III, we will concentrate on the gray scalar images (i.e., having range $\mathbb{R}^1$) defined on the 3D domains of the rectangular form $C = C_1 \times C_2 \times C_3$, each C_i being the set of integers $\{1, \ldots, m_i\}$. (Thus, in what follows, we often refer to spels as *voxels*.) Also, we will use the 6-adjacency relation of voxels. However, none of the results presented in Sec. II.B (including the algorithm P-ORFC) requires these additional restrictions. In fact, the results even apply to a more relaxed definition of the adjacency α, which allows its range to contain the fractional values in $[0, 1]$.

A *path* p in the scene $\mathcal{C}$ is any finite sequence $\langle c_1, ..., c_k \rangle$ of elements of C such that any consecutive voxels in p are adjacent, that is, with $\alpha(c_i, c_{i+1}) > 0$ for all $i = 1, \ldots, k-1$. For $A \subset C$ and $c \in C$, we say a path $p = \langle c_1, ..., c_k \rangle$: *is in* A, provided $c_i \in A$ for all i; and *is* *from* A *to* c, when $c_1 \in A$ and $c_k = c$.

II.A.2. Affinity and connectivity functions

The *affinity* function (also, referred to as a graph *weight* or *cost* function) is a map $\kappa\colon E \to [0,1]$ which assigns to every pair $\langle c, d\rangle$ of adjacent voxels a strength $\kappa(c,d)$ of local hanging togetherness. The strength $\kappa(c,d)$ usually depends on: the value of $\alpha(c,d)$ (important only, when we allow fractional values of α), the similarity between the intensities $f(c)$ and $f(d)$ (this is a *homogeneity-based affinity* component ψ), and the *object feature-based affinity* component ϕ defined below. The affinity functions are discussed in detail in[7,33]. In the experimental part of 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 $\psi(c,d)$ is given by

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

with σ_h^2 being the variance of the intensity difference $\|f(c) - f(d)\|$ for all adjacent c and d, and the object feature based affinity $\phi(c,d)$ is given by

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

Here, m and σ_o^2 are related to the mean and variance of the intensity of the object that we wish to find in $\mathcal{C}$.

For a fixed affinity κ, we define a *strength* $\mu(p)$ *of a path* $p = \langle c_1, ..., c_k\rangle$ in $\mathcal{C}$, with $k > 1$, as $\mu(p) = \min\{\kappa(c_{i-1}, c_i)\colon 1 < i \le k\}$, 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 $c \in C$ and a non-empty $S \subset C$, we define the *connectivity* value $\mu(c,S)$ (of c with respect to S) as the maximum strength $\mu(p)$ among all paths from S to c. In addition, if a set $A \subset C$ contains both c and S, then $\mu^A(c,S)$ is defined as the maximum of $\mu(p)$ among all paths from S to c in A. Finally, if $T \subset C \setminus S$ is non-empty, then $\mu(S,T) = \max_{s\in S}\mu(s,T)$ is the maximum of the strength of all possible paths from S to T.

II.A.3. Absolute and Relative Fuzzy Connectedness

For a non-empty set $S \subset C$ of seeds, indicating the foreground, and a threshold $\theta < 1$, the *Absolute Fuzzy Connectedness, AFC*, object is defined as

$$P_{S\theta} = \{c \in C \colon \mu(c, S) > \theta\}.$$

Notice that $S \subset P_{S\theta} = \bigcup_{s\in S} P_{\{s\}\theta}$ and that each object $P_{\{s\}\theta}$ is connected (in the topological and graph-theoretical sense). The AFC object is robust with respect to seed choice in the sense that $P_{\{c\}\theta} = P_{\{s\}\theta}$ for every $c \in P_{\{s\}\theta}$.

The necessity of specifying the threshold vanishes in the *Relative Fuzzy Connectedness, RFC*, definition. The RFC object is defined in terms of the disjoint non-empty sets $S, T \subset C$ of seeds, the former indicating an object, the latter the background:

$$P^1_{S,T} = \{c \in C \colon \mu(c, S) > \mu(c, T)\}.$$

The RFC object is always disjoint with T. It contains S, when $\mu(S, T) < 1$. Notice also that, if S is a singleton, then $P^1_{S,T}$ equals the AFC object $P_{S\theta}$ with $\theta = \mu(S, T)$. The RFC object is also robust with respect to seed choice.

It has been recently noticed (see[22,24]) that the RFC object $P^1_{S,T}$ minimizes an ℓ_∞-*energy* $\varepsilon^{\max}$ in the family

$$\mathcal{P}(S, T) = \{P \subset C \colon S \subset P \subset C \setminus T\},$$

where $\varepsilon^{\max}(P) = \|F_P\|_\infty = \max\{\kappa(c, d) \colon \langle c, d\rangle \in \mathrm{bd}(P)\}$ and the boundary $\mathrm{bd}(P)$ of P is defined as the set of adjacent pairs $\langle c, d\rangle$ for which P contains precisely one of c and d. Here F_P is a mapping from E to $[0, \infty)$, defined as $F_P(c, d) = \kappa(c, d)$ for $\langle c, d\rangle \in \mathrm{bd}(P)$ and $F_P(c, d) = 0$ for other adjacent pairs. The minimal energy $\min\{\varepsilon^{\max}(P) \colon P \in \mathcal{P}(S, T)\}$ is equal to $\mu(S, T)$ and the RFC object $P^1_{S,T}$ is the smallest (with respect to inclusion) element of the collection $\{P \in \mathcal{P}(S, T) \colon \varepsilon^{\max}(P) = \mu(S, T)\}$. It is worth mentioning here that the standard min cut/max flow graph cut algorithm produces an object that minimizes the ℓ^1-energy $\|F_P\|_1$ on $\mathcal{P}(S, T)$.

II.A.4. Iterative Relative Fuzzy Connectedness

The RFC procedure leads naturally to two objects: the foreground $P^1_{S,T}$ and the background $P^1_{T,S}$. These two objects, however, can still leave a sizable leftover zone $B = \{c \in$

$C\colon \mu(c,S) = \mu(c,T)\} = C \setminus (P^1_{S,T} \cup P^1_{T,S})$, where there is tie in strength of connectedness. The goal of *Iterative Relative Fuzzy Connectedness, IRFC,* is to find a way to naturally redistribute some of the spels from B to a new version of objects, $P^\infty_{S,T}$ and $P^\infty_{T,S}$. The IRFC object is defined as $P^\infty_{S,T} = \bigcup_{k=1}^\infty P^k_{S,T}$, where the objects $P^k_{S,T}$ are found iteratively, starting from the RFC object $P^1_{S,T}$:

$$P^{k+1}_{S,T} = P^k_{S,T} \cup \left\{ c \in C \setminus P^k_{S,T} \colon \mu(c,S) > \mu^{C \setminus P^k_{S,T}}(c,T) \right\}.$$

Objects $P^\infty_{S,T}$ and $P^\infty_{T,S}$ are disjoint and $P^\infty_{S,T}$ still minimizes the energy $\varepsilon^{\max}$ on $\mathcal{P}(S,T)$, as proved in[24]. The paper[24] describes also a linear time (with respect to image size) algorithm $\mathrm{GC}_{\max}$, which returns $P^\infty_{S,T}$. (Compare also[11,31,34].) Actually, $\mathrm{GC}_{\max}$ returns the Optimal Path Forest for $S \cup T$, that is, a family $\mathcal{P} = \{p_c \colon c \in C\}$ of paths such that for every $c \in C$, $p_c = \langle c_1, ..., c_k \rangle$ is from $S \cup T$ to c, $\mu(p_c) = \mu(c, S \cup T)$, and any initial restriction $\langle c_1, ..., c_j \rangle$ $(1 \le j < k)$ of p_c is also in $\mathcal{P}$. For this family, $P^\infty_{S,T}$ equals $\{c \in C \colon p_c \text{ starts at } S\}$. The IRFC object is also robust with respect to seed choice.

The $\mathrm{GC}_{\max}$ algorithm can be also used to find, still in a linear time with respect to the image size, the RFC object[24].

II.B. The new GPU-based algorithm

In this section we present the new GPU-based algorithm P-ORFC. We start with a brief description of the NVIDIA GPU hardware architecture and the CUDA programming model. For a full description of NVIDIA GPU and CUDA, readers are referred to the CUDA programming guide[35]. Then, we describe the new algorithm, P-ORFC, devised

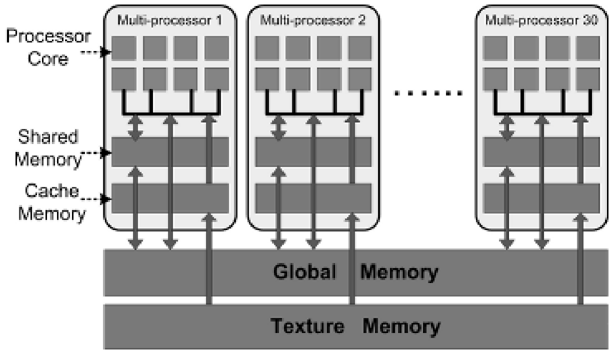


FIG. 1 NVIDIA GPU hardware architecture.

for the CUDA model and implemented in NVIDIA GPU. Finally, we state and prove the theoretical properties of P-ORFC and relate the P-ORFC-returned object to the RFC and IRFC objects.

II.B.1. NVIDIA GPU architecture and CUDA programming

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 access to this computing power, NVIDIA provides the CUDA programming framework[35], based on the C-language model, which is briefly described below.

The 240 cores of the 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.

In CUDA programming, the parts of the algorithm that are executed in parallel mode on P-ORFC are referred to as *kernels*. A CUDA kernel is executed by an array of threads. All threads run the same kernel code and each thread has an ID that it uses to compute memory addresses make control decisions.

II.B.2. The new algorithm, P-ORFC*, implemented in* CUDA

The P-ORFC algorithm is presented below.

Algorithm 1 P-ORFC

Input: An image $\langle C, f\rangle$; Non-empty sets $S, T \subset C$ of seeds, indicating the foreground (object) and background, respectively;

Output: Functions: $h\colon C \to \{-1\} \cup [0,1]$ approximating $\mu(\cdot, S \cup T)$ and a labeling map $\lambda\colon C \to \{0,1\}$ (1 for the background, 0 for the foreground);

Auxiliary: A global boolean variable β, to help in deciding on stopping of the main loop;

```
begin
1: Allocate GPU global memory for the affinity $\kappa$;
2: Invoke AFFINITY-KERNEL on GPU to compute $\kappa$;
3: Allocate GPU global memory for $h$ and $\lambda$;
4: $h(c) \leftarrow 1$ for all $c \in S \cup T$ and $h(c) \leftarrow -1$ for all $c \notin (S \cup T)$;
5: $\lambda(c) \leftarrow 1$ for all $c \in T$, and $\lambda(c) \leftarrow 0$ for all $c \in S$; $\beta \leftarrow$ "true";
6: while $\beta =$ "true" do
7:     $\beta \leftarrow$ "false";
8:     Invoke TRACKING-KERNEL($h, \lambda, \beta$);
9:     Transfer $\beta$ back to CPU;
10: end while
11: Copy $h$ and $\lambda$ from GPU to CPU;
12: Return $\{c \in C\colon \lambda(c) = 0\}$ as the foreground;
end
```

In any FC algorithm, including P-ORFC, there are two major computational tasks: (C1) computing the fuzzy affinity relations, and (C2) computing the fuzzy connectivity measure $\mu(\cdot, S \cup T)$. In addition, we record in (C2) the labels for the objects of interest. We shall refer to (C1) as "affinity computation" and (C2) as "tracking" connectivity. These two tasks are implemented as CUDA kernels in the P-ORFC algorithm illustrated in the flow chart of Figure 2.

II.B.2.a. Affinity computation kernel The CUDA implementation of fuzzy affinity computation is straightforward. The fuzzy affinity computation of every pair $\langle c, d\rangle$ of adjacent voxels (i.e., 6-adjacent or, more generally, with $\alpha(c,d) > 0$) is totally independent of other pairs of voxels. Thus, for the pair $\langle c, d\rangle$, one thread is assigned to compute the corresponding affinity components $\psi(c,d)$ and $\phi(c,d)$ from (2) and (3), and the fuzzy affinity $\kappa(c,d)$ result,

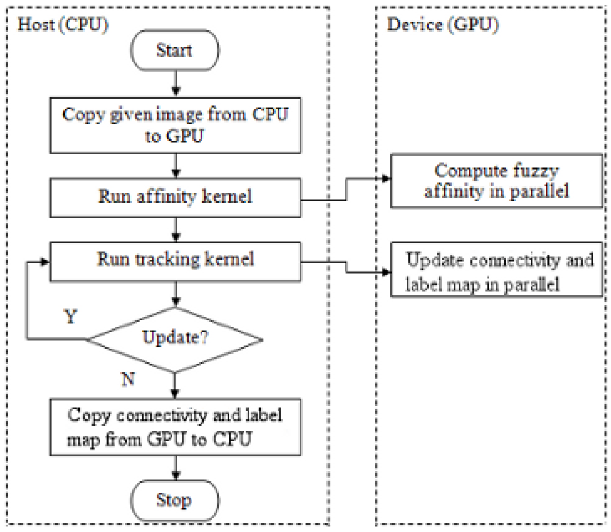

Host (CPU)
Device (GPU)
Start
Copy given image from CPU to GPU
Run affinity kernel
Compute fuzzy affinity in parallel
Run tracking kernel
Update connectivity and label map in parallel
Update?
Y
N
Copy connectivity and label map from GPU to CPU
Stop

FIG. 2 The flow chart of the CUDA implementation of the P-ORFC algorithm.

given by the formula (1), is written to the specific allocated GPU device memory.

Actually, in our experiments, we use as the affinity function $\bar{\kappa}(c,d) = D\,\hat{\kappa}(c,d)$, with $D = 2^{12} = 4096$, where $\hat{\kappa}(c,d)$ is the greatest number in the set $Z = \{i/D\colon i = 0,1,\ldots,D\}$ less than or equal to $\kappa(c,d)$. (Thus, $\bar{\kappa}(c,d)$ is the integer part of $D\,\kappa(c,d)$.) Although replacing κ with its approximation $\hat{\kappa}$ can slightly change the segmentation output, it considerably speeds up the running time of the algorithm. On the other hand, according to the results from the paper[33], the affinities $\bar{\kappa}$ and $\hat{\kappa}$ produce identical FC results.

II.B.2.b. Tracking kernel The top of the line in the sequential IRFC algorithm $\mathrm{GC_{max}}$[24] (compare also[31]) is a (relatively simple) modification of the Dijkstra's algorithm DA. In particular, the order in which the voxels are accessed by DA (so, also by $\mathrm{GC_{max}}$) is strictly regulated, and so this regulation does not allow full exploitation of the parallel processing features offered by the GPU. (We tried GPU implementation of $\mathrm{GC_{max}}$, as reported in the conference proceedings version of this paper[32], and found that such a version of the algorithm runs considerably slower than the P-ORFC algorithm presented here. This seems to be due to the fact that, during the execution of the GPU version of $\mathrm{GC_{max}}$, most of the CUDA threads actually do not update a voxel's connectivity values.) The challenges of parallel implementation of the Dijkstra's algorithm are also reported in[36,37].

To avoid the difficulties caused by the priority queue structure required by DA, a "brute-force" method was used to update the connectivity value $\mu(c, S \cup T)$ of each voxel c. In particular, all threads are made to be actively involved in the updating operations: each GPU thread independently checks the connectivity value of the voxel it processes and compares it with that of its neighbors. A global boolean variable β, indicating whether any connectivity update was made during each kernel run, is maintained on the GPU to track the update status. Its local counterpart, b, is kept on the shared memory of each multiprocessor and the b's are combined to yield β at the end of each kernel execution.

The atomic read/write operations in the device memory are used for concurrent memory access. Actually, the read/write conflict can occur only in Algorithm 3 in lines 4 and 5, when the values of h and λ are accessed/modified. The multiple write/write can appear only for line 6, but then the same modifications are made, so there is no conflict, as at least one of the modifications will be performed.

The Algorithm 1 was executed on the host (CPU) side while the kernels were executed on the device (GPU) side, as shown in Figure 2. The values of h, κ, and λ were stored in GPU global memory, which was accessible by all threads on GPU. The different threads of TRACKING-KERNEL operate on voxels for updating connectivity and label information simultaneously. In one invocation, they all updated connectivity information as much as they could on the voxels in their purview. The CPU boolean variable β determines whether any updating has been done in the last invocation of the tracking kernel. If so, the kernel was invoked again with the updated information on the voxels. The CPU terminates the run of the algorithm when no updates were performed during the last run of the tracking kernel.

The affinity and tracking kernels are presented below. The kernels start with computing thread index $t(id)$, which determines which thread handles each voxel.

Algorithm 2 AFFINITY-KERNEL

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

Note that in Algorithm 2, each pair $\langle c, d\rangle$ is considered only once. Thus, different threads independently compute affinities for different pairs of voxels.

In the tracking kernel, the value of the global variable β is updated in the following manner. A boolean variable b_i is maintained in the shared memory of each block B_i, i denoting an index of the associated multi-processor, which indicates if there are any changes of connectivity in the voxels processed by threads in B_i. When a thread in B_i updates connectivity and label map information, the value of b_i is set to "true". After all threads in each B_i finish their work, the value of β is then updated in terms of b_i's. We take this strategy to update the value of β because the memory access in the shared memory is much faster than that in the global memory.

Algorithm 3 TRACKING-KERNEL(h, λ, β)

Auxiliary: A boolean variable b in a shared memory of each block B, initialized as $b \leftarrow$ "false".

```
1: Compute thread index $t(id)$;
2: for each voxel $c$ processed by $t(id)$ do
3:     for each voxel $e$ such that $\alpha(c,e) > 0$ do
4:         if $(h(c) < \min\{h(e), \kappa(c,e)\})$ or $(h(c) = \min\{h(e), \kappa(c,e)\}$ and $\lambda(c) < \lambda(e))$ then
5:             $h(c) \leftarrow \min\{h(e), \kappa(c,e)\}$, $\lambda(c) \leftarrow \lambda(e)$;
6:             $b \leftarrow$ "true";
7:         end if
8:     end for
9: end for
10: Synchronize all threads in the block;
11: if $b =$ "true" for at least one block then
12:     $\beta \leftarrow$ "true";
13: end if
```

To understand the meaning of TRACKING-KERNEL, one needs to put it in the perspective of the entire algorithm P-ORFC, which is an iterative procedure. At the first iteration, only the threads which process the voxels c neighboring $e \in S \cup T$ 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 collects the global index information from each block on GPU that is updated by each thread in the block through shared memory and decides when to terminate calling the TRACKING-KERNEL. Each thread checks the connectivity and label values of each voxel under its control, and the values of its neighbors, to see if it is allowed to operate on the current voxel c for updating connectivity and label information.

In line 5 of TRACKING-KERNEL, atomic operation was used for consistency. Otherwise, write/read conflict can occur. (Actually, a pair $\langle h(c), \lambda(c)\rangle$ was coded as a single integer atomic variable, to insure that the update was always done simultaneously by the same thread.) In lines 6 and 12, there may be multiple writings happening simultaneously. However, atomic operation was not used here since, according to CUDA manual, it is guaranteed that at least one writing operation will be performed.

The algorithm P-ORFC terminates when all voxels are fully processed and no further changes to connectivity or label can be made.

II.B.3. Properties of P-ORFC

Theorem 1 *The object P returned by the algorithm P-ORFC contains the RFC object and is contained in the IRFC object. Moreover, P minimizes the ℓ^∞-energy $\varepsilon^{\max}$.*

Proof. Assume that during the execution of P-ORFC we keep track of the path function p such that, at any voxel c and at any time of the program execution, $p(c)$ represents the path from $S \cup T$ to c which justifies the current values h and λ. To do this formally, we would need to initialize p as

- set $h(c) \leftarrow \langle c\rangle$ for all $c \in S \cup T$ and $h(c) \leftarrow \emptyset$ for all $c \notin (S \cup T)$;

and, between lines 5 and 6 in the tracking kernel, add the command

- set $p(c) \leftarrow p(e)\hat{}\langle c\rangle$, i.e., the extension of $p(e)$ by c.

(This would not change the algorithm's output and, since we do not need the variable p except for this proof, we do not actually implement it.)

Let $\mathcal{P} = \{p_c \colon c \in C\}$ be the family of paths forming an Optimal Path Forest for $S \cup T$ such that $\{c \in C \colon p_c \text{ starts at } S\}$ is the IRFC object P^∞. (See Subsection II.A.4.) Notice that at the end of the execution of P-ORFC, for every voxel c we have

($\star$) $h(c) = \mu(c, S \cup T)$ and, whenever $c \notin P^\infty$, also $\lambda(c) = 1$.

To see this, first notice that, if ($\star$) holds for some c after any execution of the tracking kernel, then this property is preserved by any further kernel's execution, since in any consecutive execution the condition from line 4 is not satisfied. Thus, to show that ($\star$) holds, it is enough to prove, by induction on the length $|p_c|$ of p_c, that ($\star$) holds for c after $|p_c|$-many executions of the tracking kernel. Indeed, this is clearly true for $|p_c| = 1$. Also, if ($\star$) is true for any c with $|p_c| = k - 1$, then for any $p_c = p_e\hat{}\langle c\rangle$ of length k, during the kth execution of TRACKING-KERNEL, ($\star$) already holds for e and the execution of lines 4–7 insures that ($\star$) holds also for c. This concludes the argument for ($\star$).

Next, we will show that the RFC object $P^1 = \{c \in C\colon \mu(c, S) > \mu(c, T)\}$ is contained in P. For this, fix a $c \in P^1$. Then, at the end of the execution of P-ORFC, we have $\mu(p(c)) = h(c) = \mu(c, S \cup T) = \mu(c, S) > \mu(c, T)$, insuring that $\lambda(c) = 0$, as $p(c)$ must start at S. So, indeed, $c \in P$.

To see that P is contained in the IRFC object P^∞, fix a c from $C \setminus P^\infty$. We need to show that $c \notin P$. Indeed, by ($\star$), $c \in C \setminus P^\infty$ implies that $\lambda(c) = 1$ and so $c \notin P$.

Finally, we prove that P minimizes the ℓ^∞-energy $\varepsilon^{\max}$. Recall, that this energy is minimized at $\theta = \mu(S, T)$. So, take adjacent voxels v_0 and v_1, one from P, another from its complement. Let $i \in \{0, 1\}$ be such that $h(v_i) \geq h(v_{i-1})$ and put $c = v_i$, $e = v_{i-1}$. Then, $h(c) \geq h(e)$. By definition of $\varepsilon^{\max}$, we need to show that $\kappa(c, e) \leq \theta$. So, by way of contradiction, assume that $\kappa(c, e) > \theta$. Then, $h(c) \geq h(e) \geq \kappa(c, e)$ is impossible, since the concatenation of the paths $p(c)$, $\langle c, e\rangle$, and $p(e)$ would form a path from S to T of strength $\geq \kappa(c, e) > \theta = \mu(S, T)$. The inequality $\min\{h(c), \kappa(c, e)\} > h(e)$ is also impossible, since then the path $p(c)\hat{}e$ from $S \cup T$ to e would have a strength greater than $h(e) = \mu(e, S \cup T)$, contradicting maximality of $h(e)$. The only remaining possibility is that of the inequality $\kappa(c, e) \geq h(c) = h(e)$. However, in this case the algorithm would insure that both $\lambda(c)$ and $\lambda(e)$ are 1, meaning that neither c nor e is in P, a contradiction.

The above three cases show, that the assumption $\kappa(c, e) > \theta$ leads to contradiction, so that indeed we must have $\kappa(c, e) \leq \theta$, as required. ∎

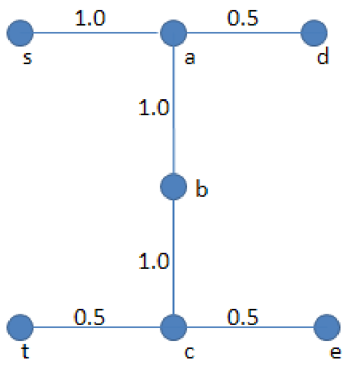


FIG. 3 Graph used to illustrate the segmentation difference between the RFC $P^1_{S,T}$ and IRFC $P^\infty_{S,T}$ objects and the output P of P-ORFC. The seed sets for the object and background are $S = \{s\}$ and $T = \{t\}$, respectively.

II.B.4. RFC object via P-ORFC algorithm

As Figure 3 shows, the output of P-ORFC does not need to coincide with either RFC or IRFC object. However, the following algorithm shows how to use P-ORFC to return the RFC object. 330

Algorithm 4 P-RFC

Input: An image $\langle C, f\rangle$; Non-empty sets $S, T \subset C$ of seeds, indicating the foreground (object) and background, respectively;

Output: The RFC object $P^1_{S,T} = \{c \in C \colon \mu(c,S) > \mu(c,T)\}$;

```
begin
  1: Invoke P-ORFC with $\hat{S} = S$ and $\hat{T} = \emptyset$ to compute $h_S(\cdot) = \mu(\cdot, S)$;
  2: Invoke P-ORFC with $\hat{S} = T$ and $\hat{T} = \emptyset$ to compute $h_T(\cdot) = \mu(\cdot, T)$;
  3: Return $\{c \in C \colon h_S(c) > h_T(c)\}$ as the foreground;
end
```

The output of P-RFC agrees with the RFC object $P^1_{S,T}$, since P-ORFC returns $h(\cdot) = \mu(\cdot, S \cup T)$. However, we invoke P-ORFC twice, so the computation time of P-RFC will be greater than (roughly twice) the computation time of P-ORFC.

Finally, an example is presented in Figure 3, for which none of the objects RFC $P^1_{S,T}$, IRFC $P^\infty_{S,T}$, and the output P of P-ORFC, coincide. Indeed, it is easy to see that $P^1_{S,T} = \{s,a,b,c\}$, since $\mu(v,S) = 1 > 0.5 = \mu(v,T)$ for $v \in \{s,a,b,c\}$ and $\mu(w,S) = \mu(w,T) = 0.5$ for $w \in \{d,e\}$. Also, $P^2_{S,T} = P^\infty_{S,T} = \{s,a,b,c,d,e\}$, since there is no path from T to $w \in \{d,e\}$ disjoint with $P^1_{S,T}$, which means that $\mu(w,S) = 0.5 > \mu(w,T) = 0$. Finally, $P = \{s,a,b,c,d\}$, since after the third invocation of the tracking kernel, the last when any change is made, we have $\langle h(v), \lambda(v)\rangle = \langle 1, 0\rangle$ for $v \in \{s,a,b,c\}$, $\langle h(d), \lambda(d)\rangle = \langle 0.5, 0\rangle$, and $\langle h(e), \lambda(e)\rangle = \langle 0.5, 1\rangle$. (After the second invocation of the tracking kernel these parameters are: $\langle h(v), \lambda(v)\rangle = \langle 1, 0\rangle$ for $v \in \{s,a,b\}$, $\langle h(d), \lambda(d)\rangle = \langle 0.5, 0\rangle$, and $\langle h(w), \lambda(w)\rangle = \langle 0.5, 1\rangle$ for $w \in \{c,e\}$.) In summary, for this example, we have $P^1_{S,T} \subsetneq P \subsetneq P^\infty_{S,T}$.

TABLE I Data set information and performance of the GPU implementation with respect to an optimal CPU implementation.

Dataset	*Small*	*Medium*	*Large*	*Super*
Protocol	PD MRI	T1 MRI	T1 MRI	T1 MRI
Scene domain	$256 \times 256 \times 55$	$256 \times 256 \times 124$	$512 \times 512 \times 192$	$512 \times 512 \times 576$
Voxel size (mm)	$0.98 \times 0.98 \times 3.0$	$0.94 \times 0.94 \times 1.5$	$0.5 \times 0.5 \times 1.0$	$0.5 \times 0.5 \times 0.33$
CPU time (s)	8.21	19.24	101.45	312.24
GPU time (s)	0.25	0.84	4.85	17.85
Speed-up	32.8	22.9	20.9	17.5

III. EXPERIMENTAL RESULTS

In this section, the running times of the GPU and CPU implementations of the P-ORFC algorithm are compared for image data of different sizes, and the similarity and accuracy of the segmentations are assessed.

The CPU version 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 of device memory. CUDA SDK 3.2 is used in our GPU implementation. Four image

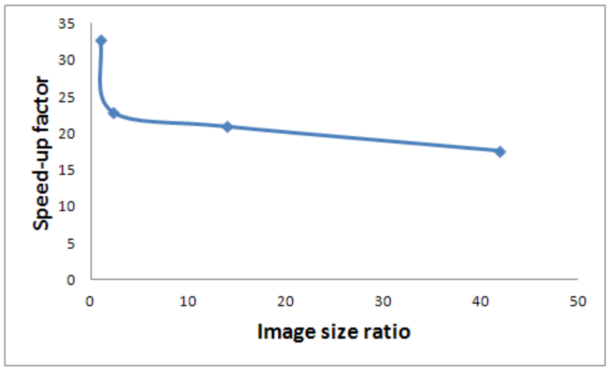


FIG. 4 Curve of the speed-up factor vs image size ratio.

data sets—small, medium, large, and super—are utilized to test the performance of the GPU and CPU implementations. Table I lists the image data set information and shows the performance of the GPU versus CPU implementation. (The "super" data set was obtained by interpolating the "large" data set.) The parameters and seeds used in both CPU and GPU implementations are identical. The parameters m, σ_h, and σ_o are estimated from a small training region.

A speed-up factor of 32.8×, 22.9×, 20.9×, and 17.5×, respectively, has been achieved, for the four data sets over the optimal CPU implementation. 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. Seemingly there is some loss of achievable speed-up as the data size increases. This is mainly because, for larger data sets, typically a larger number of iterations in the algorithm of P-ORFC will be required, which means more costly transfers between GPU and CPU. In addition, the affinity and tracking kernels require much more device global memory access, which has high latency. However, the loss seems to level off once the data set size crosses the medium size, as depicted in Figure 4, where the value on the horizontal axis represents the ratio of image data size to the small data size. Note that, in all our experiments, we segmented the white matter object (foreground) from the other co-objects (background). For each scene in our experiment, the white matter object covers almost the whole scene domain.

Figure 5 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 was used. Figures 5(a) and (b) show one slice of the original PD-weighted scene and the corresponding white

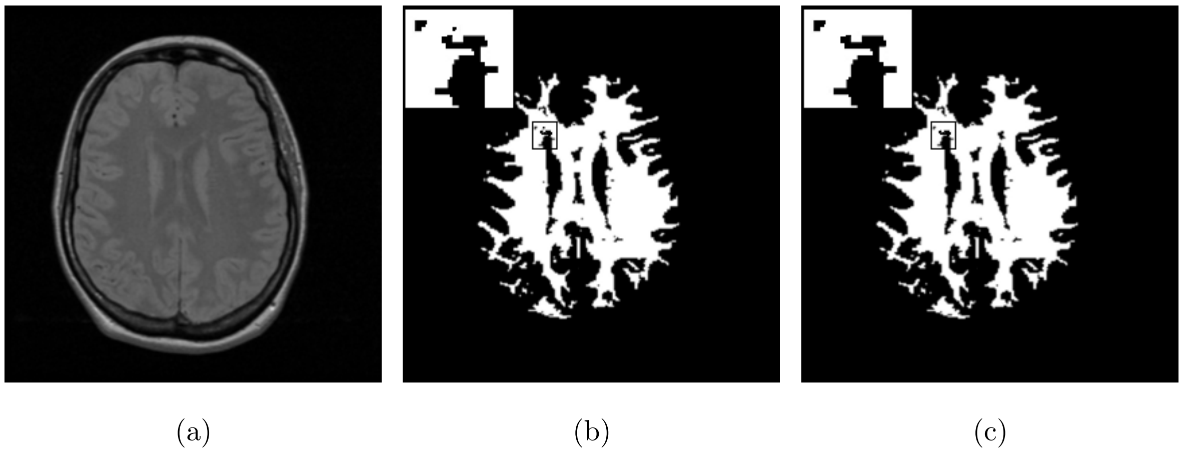


FIG. 5 (a) A slice of PD-weighted MRI scene from the small data set, (b) the white matter segmented by the GPU implementation, and (c) the white matter segmented by the CPU implementation.

matter produced by the algorithm P-ORFC. Figure 5(c) shows the white matter segmented by the CPU implementation. A region of interest is shown magnified to demonstrate how segmentation results may differ qualitatively between CPU and GPU implementations.

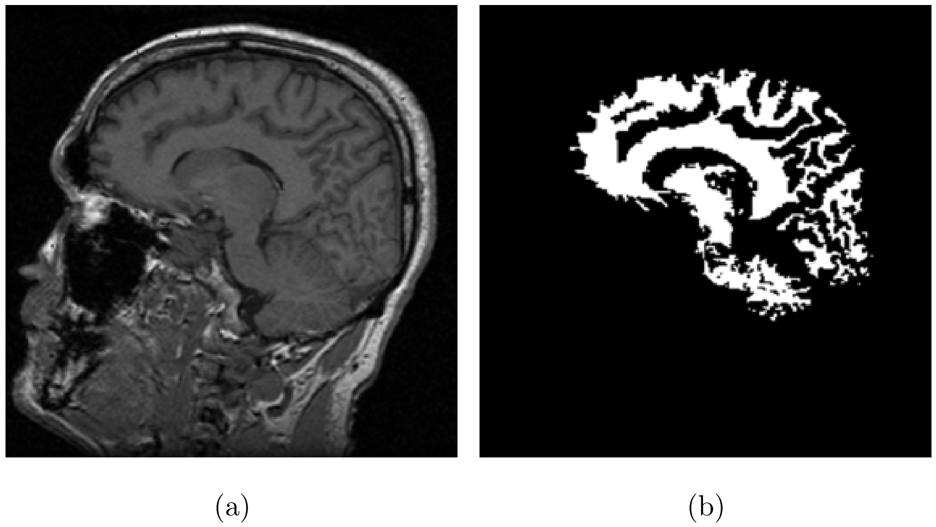
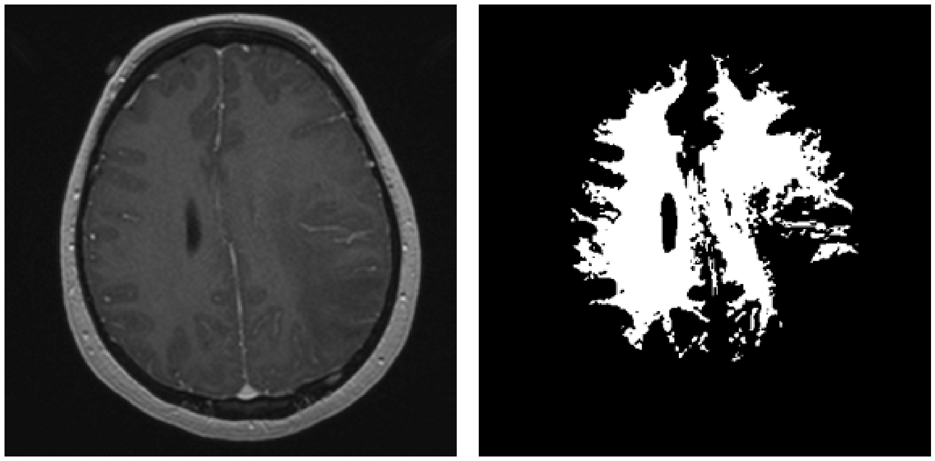


FIG. 6 (a) A slice of T1-weighted MRI scene from the medium data set, and (b) the final label map.

Figure 6 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. A spoiled gradient recalled (SPGR) acquisition was used. This data set was obtained from the web site of National Alliance for



(a) (b)

FIG. 7 (a) A slice of T1-weighted MRI scene from the large data set, and (b) the final label map.

Medical Image Computing (http://www.na-mic.org). Again, Figures 6(a) and (b) show one slice of the original scene, and the separated white matter.

Figure 7 shows the example of the large size data set, which is a T1-weighted MRI scene of the head of a patient with a large brain tumor. The original scene slice and the corresponding segmented white matter are shown in Figures 7 (a) and (b).

It is noted that the connectivity measures h produced by the GPU and CPU implementations are identical. However, the label maps λ may be slightly different in these two implementations. The differences are at the level of label assignment of voxels when the strengths of connectedness are equal with respect to object seeds S and background seeds T, as illustrated in the previous section via Figure 3. Note that the optimal CPU implementation outputs an IRFC object $P^{\infty}_{S,T}$[24].

The dice coefficient[38] is used to quantitatively assess the similarity between segmentation results obtained from the CPU and GPU implementations (objects P and $P^{\infty}_{S,T}$). Given two segmented binary scenes X and Y from two methods, the dice coefficient is defined as

$$d_{\nu}(X,Y) = \frac{2|(X \bigcap Y|)}{(|X| + |Y|)}, \tag{4}$$

where $|X|$ represents the number of voxels in scene X with value 1. The Dice coefficient varies from 0 to 1 and it measures the degree of agreement between the two segmented regions. It is 1 when the two regions are identical and 0 when they are completely disjoint.

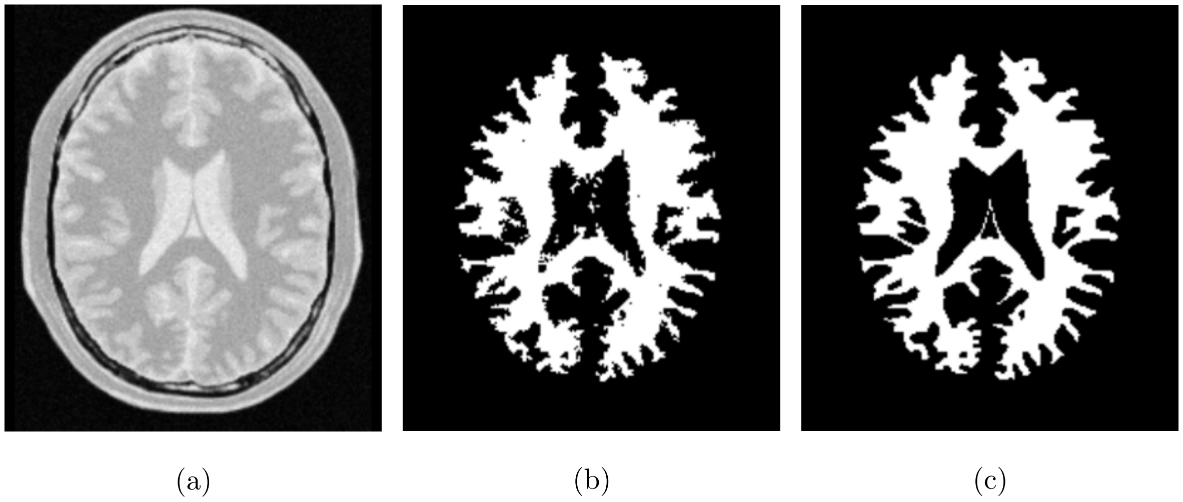


FIG. 8 (a) A slice of PD-weighted MRI scene with 3% noise from the simulated BrainWeb database, (b) the segmented white matter by the algorithm P-ORFC, and (c) the ground truth.

The values of d_ν are listed in Table II for segmented white matter objects produced by the two algorithms for the four data sets. The high value of d_ν (> 0.99) for all data sets indicates the high degree of agreement between the segmentation results of the two algorithms.

TABLE II Comparison between segmentation results obtained from the CPU and GPU versions of the algorithm, for the data sets in Table I.

Dataset	*Small*	*Medium*	*Large*	*Super*
d_ν	0.992	0.995	0.994	0.994

The accuracy of the algorithm P-ORFC is evaluated on the BrainWeb simulated brain database with several noise levels[39]. PD-weighted MRI scenes of normal brain with 1%, 3%, and 5% noise levels are used in our experiment. Each scene has a size of $181 \times 217 \times 181$ and voxel size of $1.0\text{mm} \times 1.0\text{mm} \times 1.0\text{mm}$. The white matter tissue is segmented by using the algorithm P-ORFC and the result compared to known ground truth is expressed via the dice coefficient in Table III. Note that we did not employ any preprocessing step for noise suppression in segmentation, and even for the highest noise level, a d_ν value of 0.93 was achieved. Figures 8(a) and (b) show one slice of the original PD-weighted scene with 3% noise and the corresponding white matter segmented by the algorithm P-ORFC. Figure

8(c) shows ground truth of the white matter.

TABLE III Accuracy of the algorithm P-ORFC on the simulated PD-weighted MRI scene of normal brain with different levels of noise.

Noise level	1%	3%	5%
d_ν	0.968	0.943	0.926

IV. DISCUSSION AND 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.

A parallel version of an algorithm that optimizes an ℓ_∞-based energy and belongs to the family of FC algorithms, has been developed on the NVIDIA GPUs, which provides a far more cost and speed-effective solution than both clusters of workstations and multiprocessing systems. The parallel implementation achieves speed increases by factors ranging from 17.5$\times$ to 32.8$\times$ on the Tesla C1060 GPU over the top-of-the-line CPU implementation of the IRFC algorithm, optimized and efficient (near-linear time), for image data sets with a wide range of sizes. An interactive speed of segmentation has been achieved, even for the largest data set. For some specific applications, several free parameters (e.g. fuzzy affinity parameter) in fuzzy connected image segmentation may be difficult to optimize. The interactive speed of segmentation can give users immediate feedback on parameter settings, thus allowing them to fine-tune free parameters and produce more accurate segmentation results. The accuracy of the parallel algorithm on GPU has been evaluated on the BrainWeb simulated PD-weighted MRI scenes with 1%, 3%, and 5% level of noise, and has been shown to yield a dice coefficient of 0.93-0.97, with greater than 99% similarity to the results from CPU algorithms. The algorithm can be generalized to GPU implementation of other ℓ_∞-norm minimization methods such as the image foresting transform[11].

In the current implementation, the tracking kernel is iteratively launched, which is computationally expensive. The performance of the parallel implementation can 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[40]. In the future, we will study clinical applications of the GPU-based RFC image segmentation method, and the integration of this algorithm with the method of fuzzy model-based automatic anatomy recognition[41]. Such GPU implementations may play a crucial role in automatic anatomy recognition in clinical radiology.

Note that in the algorithm P-ORFC, the number of calls to TRACKING-KERNEL may be sensitive to the spatial distribution of seed locations; thus its running time will be affected by the distribution of seed voxels relative to the shape of the object being segmented. In our experiments, sets S and T were specified as those voxels whose intensities were equal to the mean intensity for the white matter and gray matter tissues, respectively. The relationship between the number of iterations in P-ORFC and the spatial distribution of seed voxels will need to be investigated in the future to study optimum ways of specifying seeds for the best possible algorithm speed. In fuzzy model-based FC, seed specification tailored to the particular object shape can be potentially implemented.

Acknowledgement

This research was supported by the Intramural Research Program of the National Cancer Institute, NIH.

REFERENCES

[a)]Electronic mail: zhugey@mail.nih.gov

[b)]Electronic mail: KCies@math.wvu.edu

[c)]Electronic mail: jay@mail.med.upenn.edu

[d)]Electronic mail: rwmiller@mail.nih.gov

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

[2]C. Couprie, L. Grady, L. Najman, H. Talbot, Power Watershed: A Unifying Graph-

Based Optimization Framework, IEEE Transactions on Pattern Analysis and Machine Intelligence 33(7), 1384–1399 (2011).

[3]A. Bhusnurmath, C. J. Taylor, Graph cuts via ℓ_1 norm minimization, IEEE Transactions on Pattern Analysis and Machine Intelligence 30(10), 1866–1871 (2008).

[4]Y. Boykov, O. Veksler, R. Zabih, Fast approximate energy minimization via graph cuts, IEEE Transactions on Pattern Analysis and Machine Intelligence 23(11), 1222–1239 (2001).

[5]L. Grady, Random walks for image segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 28(11), 1768–1783 (2006).

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

[7]P. K. Saha, J. K. Udupa, D. Odhner, Scale-based fuzzy connectedness image segmentation: Theory, algorithms, and validation, Computer Vision and Image Understanding 77(2), 145–174 (2000).

[8]J. K. Udupa, P. K. Saha, R. A. Lotufo, Relative fuzzy connectedness and object definition: Theory, algorithms, and applications in image segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence 24(11), 1485–1500 (2002).

[9]K. C. Ciesielski, J. K. Udupa, P. K. Saha, Y. Zhuge, Iterative relative fuzzy connectedness for multiple objects with multiple seeds, Computer Vision and Image Understanding 107(3), 160–182 (2007).

[10]X. Bai, G. Sapiro, A geodesic framework for fast interactive image and video segmentation and matting, Proc. ICCV'07, 1–8(2007).

[11]A. X. Falcão, J. Stolfi, R. A. Lotufo, The image foresting transforms: Theory, algorithms and applications, IEEE Transactions on Pattern Analysis and Machine Intelligence 26(1), 19–29 (2004).

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

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

[14]Y. Zhou, J. Bai, Atlas-based fuzzy connectedness segmentation and intensity non-uniformity correction applied to brain MRI, IEEE Trans. Biomedical Engineering 54(1), 121–129 (2007).

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

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

[17]Y. Zhuge, P. K. Saha, J. K. Udupa, Vectorial scale-based fuzzy connected image segmentation, Computer Vision and Image Understanding 101, 177–193 (2006).

[18]A. Pednekar, I. A. Kakadiaris, Image segmentation based on fuzzy connectedness using dynamic weights, IEEE Transaction on Image Processing 15(6), 1555–1562 (2006).

[19]G. T. Herman, B. M. Carvalho, Multiseeded Segmentation Using Fuzzy Connectedness, IEEE Transactions on Pattern Analysis and Machine Intelligence 23(5), 460–474 (2001).

[20]K. C. Ciesielski, J. K. Udupa, Affinity functions in fuzzy connectedness based image segmentation II: Defining and recognizing truly novel affinities, Computer Vision and Image Understanding 114(1), 155–166 (2010).

[21]K. C. Ciesielski, J. K. Udupa, 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(6), 721–734 (2011).

[22]K. C. Ciesielski, J. K. Udupa, Region-based segmentation: fuzzy connectedness, graph cut, and other related algorithms, in: T. M. Deserno (Ed.), Biomedical Image Processing, Springer-Verlag, Berlin Heidelberg, Part 4, 251–278 (2011).

[23]K. C. Ciesielski, J. K. Udupa, P. A. Miranda, A. X. Falcão, Comparison of fuzzy connectedness and graph cut segmentation algorithms, Proc. SPIE 7962, 796203-1–12 (2011).

[24]K. C. Ciesielski, J. K. Udupa, A. X. Falcão, P. A. V. Miranda, Fuzzy connectedness image segmentation in graph cut formulation: A linear-time algorithm and a comparative analysis, Journal of Mathematical Imaging and Vision 44(3), 375–398 (2012).

[25]V. Vineet, P. J. Narayanan, CUDA cuts: Fast graph cuts on the GPU, Proc. CVPR Workshops, 1–8 (2008).

[26]L. Grady, T. Schiwietz, S. Aharon, R. Westermann, Random Walks for Interactive Organ Segmentation in Two and Three Dimensions: Implementation and Validation, Proc.

MICCAI, 773–780(2005).

[27]M. Collins, J. Xu, L. Grady, V. Singh, Random walks based multi-image segmentation: Quasiconvexity results and GPU-based solutions, Proc. CVPR, 1656–1663(2012).

[28]G. Grevera, J. K. Udupa, D. Odhner, Y. Zhuge, A. Souza, S. Mishra, T. Iwanaga, CAVASS: a computer-assisted visualization and analysis software system, Journal of Digital Imaging 20(Suppl 1), 101–118 (2007).

[29]B. M. Carvalho, G. T. Herman, Parallel fuzzy segmentation of multiple objects, Int. J. Imaging Syst. Technol. 18(5-6), 336–344 (2008).

[30]Y. Zhuge, Y. Cao, J. K. Udupa, R. W. Miller, Parallel fuzzy connected image segmentation on GPU, Medical Physics 38(7), 4365–4371 (2011).

[31]P. A. Miranda, A. X. Falcão, Links between image segmentation based on optimum-path forest and minimum cut in graph, Journal of Mathematical Imaging and Vision 35(2), 128–142 (2009).

[32]Y. Zhuge, J. K. Udupa, K. C. Ciesielski, A. X. Falcão, P. A. V. Miranda, R. W. Miller, GPU-based iterative relative fuzzy connectedness image segmentation, Proc. SPIE 8316, 831604-1–11 (2012).

[33]K. C. Ciesielski, J. K. Udupa, Affinity functions in fuzzy connectedness based image segmentation I: Equivalence of affinities, Computer Vision and Image Understanding 114(1), 146–154 (2010).

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

[35]NVIDIA, NVIDIA CUDA programming guide 3.2, NVIDIA Corporation, 2010, http://developer.nvidia.com/cuda.

[36]A. S. Nepomniaschaya, M. A. Dvoskina, A simple implementation of dijkstra's shortest path algorithm on associative parallel processors, Fundam. Info. 43(1-4), 227–243 (2000).

[37]P. Harish, P. J. Narayanan, Accelerating large graph algorithms on the GPU using cuda, Proc. High performance computing, HiPC'07, 197–208 (2007).

[38]L. R. Dice, Measures of the amount of ecologic association between species, Ecology 26(3), 297–302 (1945).

[39]D. L. Collins, A. P. Zijdenbos, V. Kollokian, J. G. Sled, N. J. Kabani, C. J. Holmes, A. C. Evans, Design and Construction of a Realistic Digital Brain Phantom, IEEE Trans. Medical Imaging 17(3), 463–468(1998).

[40]U. Meyer, P. Sanders, $\triangle$-stepping: A parallel single source shortest path algorithm, J. Algorithms 49(1), 114–152 (2003).

[41]J. K. Udupa, D. Odhner, A. X. Falcão, K. C. Ciesielski, P. A. V. Miranda, M. Matsumoto, G. J. Grevera, B. Saboury, D. A. Torigian, Automatic anatomy recognition via fuzzy object models, Proc. SPIE 8316, 831605-1–8 (2012).