31st Annual International Conference of the IEEE EMBS
Minneapolis, Minnesota, USA, September 2-6, 2009

GPU Accelerated Fuzzy Connected Image Segmentation by using CUDA

Ying Zhuge, Yong Cao, and Robert W Miller

Abstract— **Image segmentation techniques using fuzzy connectedness principles have shown their effectiveness in segmenting a variety of objects in several large applications in recent years. However, one problem of these algorithms has been their excessive computational requirements when processing large image datasets. Nowadays commodity graphics hardware provides high parallel computing power. In this paper, we present a parallel fuzzy connected image segmentation algorithm on Nvidia's Compute Unified Device Architecture (CUDA) platform for segmenting large medical image data sets. Our experiments based on three data sets with small, medium, and large data size demonstrate the efficiency of the parallel algorithm, which achieves a speed-up factor of 7.2x, 7.3x, and 14.4x, correspondingly, for the three data sets over the sequential implementation of fuzzy connected image segmentation algorithm on CPU.**

I. INTRODUCTION

Image segmentation is one of the most crucial tasks in image processing and computer vision. In spite of nearly four decades of research, image segmentation remains a challenging problem. Recently developed fuzzy connectedness framwork and it extensions have been extensively utilized in many medical applications [1], [2], [3], [4]. These include multiple sclerosis lesion detection and quantification via MR imaging [5], upper airway segmentation in children via MRI for studying obstructive sleep apnea [6], electron tomography segmentation [7], abdominal segmentation [8] and automatically brain segmentation [9] in MRI images with the assistance of an atlas of their corresponding regions, clutter-free volume rendering and artery-vein separation in MR angiography [10], and in brain tumor delineation via MR imaging [11], and automatic breast density estimation [12] via digitized mammograms for breast cancer risk assessment. However, one problem with the fuzzy connected image segmentation algorithms has been their time-consuming requirements for large image data [13].

Graphics Processing Unit (GPU) computing presents one of the techniques that can address this problem. The GPUs substantial arithmetic and memory bandwidth capabilities, coupled with its recent addition of user programmability, has allowed for general-purpose computation on graphics hardware (GPGPU)[14]. Many non-graphics-oriented computationally expensive algorithms have been implemented on the GPU. Developers prefer GPUs over other alternative parallel processors such as cluster of workstations due to several advantages including their low cost and wide availability. Owens et al. presented a comprehensive survey of latest research in GPU computing [15]. Since medical imaging applications intrinsically have data-level parallelism with high compute requirements, they are very suitable to be implemented on the GPU. Several research work of medical imaging applications on GPU has been reported recently [16], [17].

The purpose of this paper is to develop a parallel fuzzy connected image segmentation algorithm on GPU by using CUDA to achieve interactive speed when segmenting large medical image data. The paper is organized as follows. In Section II, we first briefly present the fuzzy connectedness principles and it sequential algorithm. In Section III, we describe the NVIDIA GPU architecture and the CUDA programming model, and explain how to implement parallel fuzzy connected image segmentation by using CUDA. The experimental results are presented in Section IV. Finally, we state our concluding remarks in Section V.

II. FUZZY CONNECTEDNESS PRINCIPLES AND SEQUENTIAL ALGORITHM

We shall briefly describe the concepts related to fuzzy connectedness in this section to make this paper self-contained. We refer to a volume image as a *scene* and represent it by a pair $\mathcal{C} = (C, f)$, where C is a rectangular array of cuboidal volume elements, usually referred to as voxels, and f is the scene intensity function which assigns to every voxel $c \in C$ an integer called the intensity of c in $\mathcal{C}$ in a range $[L, H]$.

A. Fuzzy adjacency and affinity

Independent of any image data, we think of the digital grid system defined by the voxels as having a *fuzzy adjacency relation*. This relation assigns to every pair (c, d) of voxels a value between zero and one. The closer c and d are spatially to each other, the greater is this number. This is intended to be a "local" phenomenon. How "local" it ought to be should perhaps depend on the blurring property of the imaging device. We denote the fuzzy adjacency relation by α and the degree of adjacency assigned to any voxels (c, d) by $\mu_\alpha(c, d)$.

Now consider the voxels as having scene intensities assigned to them. We define another local fuzzy relation called *affinity* on voxels denoted by κ. The strength of this relation between any voxels c and d, denoted $\mu_\kappa(c, d)$, lies between zero and one, and indicates how the voxels "hanging together" locally in the scene. $\mu_\kappa(c, d)$ is determined on $\mu_\alpha(c, d)$, as well as on how similar the intensities or intensity-based properties at c and d. The properties of fuzzy

Ying Zhuge and Robert Miller are with Radiation Oncology Branch, National Cancer Institute, National Institutes of Health, Bethesda, MD 20892, USA {zhugey, rwmiller}@mail.nih.gov

Yong Cao is with Computer Science Department, Virginia Polytechnic Institute and State University, Blacksburg, VA 24060, USA yongcao@vt.edu

affinity relations are studied extensively and a guidance as to how to setup fuzzy affinities in practical applications is given in [2]. In this paper, we used the following functional form for μ_κ,

$$\mu_\kappa(c,d) = \mu_\alpha(c,d)\sqrt{g_1(f(c),f(d))g_2(f(c),f(d))}, \quad (1)$$

where g_1 and g_2 are Gaussian functions of $\frac{f(c)+f(d)}{2}$ and $\frac{|f(c)-f(d)|}{2}$, respectively. In this equation, g_1 is a Gaussian with mean and variance that are related to the mean and variance of the intensity of the object we with to define in the scene. That is, this component of affinity takes on a high value when c and d are both close to an expected intensity value for the object. g_2 is a 0-mean Gaussian, the underlying idea being to capture the degree of local hanging togetherness of c and d based on intensity homogeneity.

B. Fuzzy connectedness and fuzzy objects

Our aim it to capture the global phenomenon of "hanging togetherness" in a global fuzzy relation on voxels called *fuzzy connectedness*, denote K. The strength of this relation $\mu_K(c,d)$ between any voxels c and d, indicating the strength of their connectedness, lies between zero and one, and is determined as follows: There are numerous possible "paths" within the scene domain C between c and d. Each path for our purposes is a sequence of voxels, starting from c and ending in d, with the successive voxels being nearby. We think of each pair of successive voxels as constituting a link and the whole path to be a chain of links. We assign a strength (between zero and one) to every path which is simply the smallest pairwise voxels affinity along the path. Finally, the strength of connectedness between c and d is the strength associated with the strongest of all paths between c and d.

Let θ be any number in [0,1], a fuzzy connected object $\mathcal{O}$ in $\mathcal{C}$ of strength θ, and containing a voxel o, consists of a pool $O \subset C$ of voxels together with a value indicating "objectness" assigned to every voxel. O is such that $o \in O$, and for any voxels $c \in O$ and $d \in O$, the strength of connectedness between them $\mu_K(c,d) \geq \theta$, and for any voxel $c \in O$ and $e \notin O$, the strength $\mu_K(c,e) \leq \theta$.

The fuzzy connectedness algorithm is presented below. For any voxel affinity κ in a scene $\mathcal{C} = (C, f)$, we define the *κ-connectivity scene* of $\mathcal{C}$ with respect to a voxel $o \in C$ by $\mathcal{C}_{Ko} = (C, f_{Ko})$, where, for any $c \in C$, $f_{Ko} = \mu_K(o,c)$. The algorithm uses Dijkstra's implementation of dynamic programming to find the best path from o to each voxel in C.

Algorithm κFOE

Input: $\mathcal{C} = (C, f)$, any $o \in C$ and any fuzzy affinity κ.
Output: A κ-connectivity scene $\mathcal{C}_{Ko} = (C, f_{Ko})$ of $\mathcal{C}$ with respect to o.
Auxiliary Data Structures: A 3D array representing the connectivity scene $\mathcal{C}_{Ko} = (C, f_{Ko})$ and a queue Q containing voxels to be processed. We refer to the array itself by $\mathcal{C}_{Ko}$ for the purpose of the algorithm.

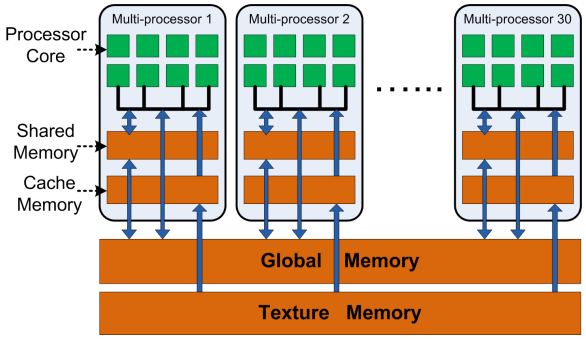

Fig. 1. Architecture of the NVIDIA GTX280 GPGPU in detail.

begin
1 set all voxels of $\mathcal{C}_{Ko}$ to 0 except o which is set to 1;
2 push o to Q;
3 **while** Q is not empty **do**
4 remove a voxel c from Q for which $f_{Ko}(c)$ is maximal;
5 **for** each voxel e such that $\mu_\kappa(c,e) > 0$ **do**
6 set $f_{\min} = \min\{f_{Ko}(c), \mu_\kappa(c,e)\}$;
7 **if** $f_{\min} > f_{Ko}(e)$ **then**
8 set $f_{Ko}(e) = f_{\min}$;
9 **if** e is already in Q **then**
10 update the location of e in Q;
11 **else**
12 push e in Q;
end

III. PARALLEL FUZZY CONNECTED IMAGE SEGMENTATION ON GPU

A. NVIDIA GPU Architecture and Programming Model

The performance of a state-of-art GPU is compatible to a supercomputer. For example, a NVIDIA GTX280 GPU has 240 processing cores with a clock rate of 1.3G Hz 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 CUDA framework [18], which is based on a C-language model instead of Graphics *Shader-Programming* concept. CUDA enables 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 GTX280 GPU are grouped into 30 multi-processors, as shown in Figure 1. Each multi-processor has 8 processing cores, organized in a SIMD (Single Instruction Multiple Data) fashion. GTX280 has 1 GB of onboard device memory, which can be used as read-only texture memory or read-write global memory. GPU device memory features a very high bandwidth, recorded at 141 GB per second, but is suffered from high access latency. In each multi-processor, there is 16 KB of user-controlled L1 cache, called shared memory. If it is used efficiently, it can be used to hide the latency for global memory access.

CUDA programming model is based on concurrently executed threads. CUDA manages threads in a hierarchical

structure. *Threads* are grouped into thread *block*, and thread blocks are grouped into *grid*. All threads in one grid share the same functionality, as they are executing the same *kernel* code. Each thread block is mapped onto one multi-processor, and threads in each block are scheduled to run on 8 processing cores of the multi-processor, 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 only communicate through low-speed global memory.

B. CUDA Implementation

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 fuzzy connectedness segmentation method, there are two major computational tasks: (i) computing the fuzzy affinity relations, and (ii) computing the fuzzy connectedness relations. We shall refer to (i) as "affinity computation" and (ii) as "tracking" a fuzzy object. we implement these two tasks as CUDA *kernels*, and a dramatic improvement for both tasks can be achieved in their speed.

1) *Affinity computation kernel* : The CUDA implementation of fuzzy affinity computation is straightforward. The fuzzy affinity computation of every pair (c, d) of voxel c and d where $\mu_\alpha(c, d)$ is greater than zero is totally independent of other pair of voxels. Thus for the pair (c, d) of voxels of c and d, one thread computes corresponding $g_1(c, d)$ and $g_2(c, d)$ in equation 1, and write the fuzzy affinity $\mu_\kappa(c, d)$ to GPU global memory. For computational simplicity, we use the six-adjacency relation for α.

2) *Tracking kernel*: Computing the fuzzy connectedness values for a fuzzy object is a variation of the single-source-shortest-path (SSSP) problem. Dijkstra's algorithm is an optimal sequential solution to SSSP problem. Parallel implementations of Dijkstra's SSSP algorithm is quite challenging [19]. As far as we know, there is no efficient parallel algorithm of the SSSP in a SIMD model. Harish and Narayanan [20] proposed using CUDA to accelerate large graph algorithms (including SSSP) on the GPU, however they implemented only a very basic version and did not gain much performance improvement. We use similar implementation, but take advantage of newer version of CUDA hardware which supports atomic read/write operations in the device global memory. Our CUDA implementation is presented below.

Algorithm CUDA-κFOE

Input: $\mathcal{C} = (C, f)$, any $o \in C$ and any fuzzy affinity κ.
Output: A κ-connectivity scene $\mathcal{C}_{Ko} = (C, f_{Ko})$ of $\mathcal{C}$ with respect to o.
Auxiliary Data Structures: Two 3D arrays representing binary scenes $\mathcal{C}_{m1} = (C, f_{m1})$ and $\mathcal{C}_{m2} = (C, f_{m2})$. We refer to arrays themselves by $\mathcal{C}_{m1}$ and $\mathcal{C}_{m2}$ for the purpose of the algorithm.

begin
1 set all voxels of $\mathcal{C}_{Ko}$ to 0 except o which is set to 1;
2 set all voxels of $\mathcal{C}_{m1}$ to 0 except o which is set to 1;
3 **while** $\mathcal{C}_{m1}$ is not all zero **do**
4 set all voxels of $\mathcal{C}_{m2}$ to 0;
5 **for** each voxel in parallel **do**
6 Invoke TRACKING-KERNEL($\mathcal{C}_{Ko}, \mathcal{C}_{m1}, \mathcal{C}_{m2}, \mu_\kappa$) on grid ;
7 Copy $\mathcal{C}_{m2}$ to $\mathcal{C}_{m1}$;
end

Algorithm TRACKING-KERNEL($\mathcal{C}_{Ko}, \mathcal{C}_{m1}, \mathcal{C}_{m2}, \mu_\kappa$)

begin
1 **if** $f_{m1}(c)$ **then**
2 **for** each voxel e such that $\mu_\kappa(c, e) > 0$ **do**
3 set $f_{\min} = \min\{f_{Ko}(c), \mu_\kappa(c, e)\}$;
4 **if** $f_{\min} > f_{Ko}(e)$ **then**
5 set $f_{Ko}(e) = f_{\min}$;
6 set $f_{m2}(e) = 1$;
end

In our implementation, TRACKING-KERNEL is called in each iteration. Each voxel $c \in C$ checks if it is true in the binary array $\mathcal{C}_{m1}$. If yes, (which means its neighbor's connectivity values need to be updated,) it fetches its connectivity value $f_{Ko}(c)$ from the connectivity array $\mathcal{C}_{Ko}$ and the affinities $\mu_\kappa(c, e)$ between voxel c and its adjacent voxel e. Then the connectivity value of voxel e is updated if the minimum of $\mu_\kappa(c, e)$ and $f_{Ko}(c)$ is greater than its original connectivity value $f_{Ko}(e)$. Note in line 5 of Algorithm TRACKING-KERNEL, we used atomic operation for consistency because update operations might happen by multiple threads simultaneously. The two binary arrays $\mathcal{C}_{m1}$ and $\mathcal{C}_{m2}$ are used to avoid inconsistency too. When voxel c has been processed by one thread, if it needs to be processed further in next iteration depends on processing results of its adjacent voxels. The algorithm CUDA-κFOE terminates until there is no any update from all threads.

IV. EXPERIMENTAL RESULTS

In this section, we compare the running times of our GPU and CPU implementations of the fuzzy connectedness image segmentation for data sets with different sizes. The CPU version of fuzzy connectedness is implemented in C++. The computer used is a DELL PRECISION T7400 with quad-core 2.66GHz Intel Xeon CPU. It runs Windows XP and has 2GB of main memory.The GPU used is the NVIDIA GTX280 with 240 processing cores and 1GB device memory. CUDA SDK 2.0 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 I lists the image data sets information and shows the performance of our GPU implementation with respect to the CPU implementation. We have achieved from 7.2x to 14.4x speedup over the CPU implementation. The bigger

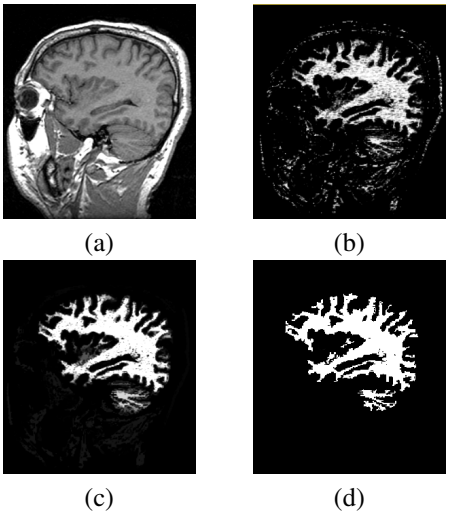

Fig. 2. A slice of T1-weighted MRI scene from the medium data set (a), the corresponding slices of the scenes depicting the affinity values (b), the connectedness values (c), and the final hard object (d).

size of the data set, the more speedup we can achieve. Note that segmentation results produced from both GPU and CPU implementations are same.

TABLE I

DATA SET INFORMATION AND PERFORMANCE OF GPU IMPLEMENTATION WITH RESPECT TO CPU IMPLEMENTATION.

Dataset	small	medium	large
Protocol	PD MRI	T1 MRI	CT
Domain	(256,256,46)	(256,256,124)	(256,256,459)
Voxel Size (mm)	(0.98,0.98,3.0)	(0.94,0.94,1.5)	(0.68,0.68,1.5)
CPU time(sec)	6.17	13.36	27.88
GPU time(sec)	0.86	1.84	1.94
speedup	7.2	7.3	14.4

Figure 2 shows one example from the medium data set, which is a SPGR T1-weighted MRI scene of the head of a clinically normal human subject we download from the web site of National Alliance for Medical Image Computing (http://www.na-mic.org). Figure 2(a) shows one slice of the original scene, and Figures 2(b), 2(c), and 2(d) show corresponding slices depicting the affinity values, connectedness values, and the final hard object for white matter in the head.

V. CONCLUDING REMARKS

The data sets produced in radiological exams are growing larger everyday, which creates a severe demand of computing power for image segmentation algorithms. To address such demand, we have introduced a CUDA implementation of widely used fuzzy connected image segmentation method on low-cost GPUs. Our results show that the CUDA implementation achieves a speedup from 7.2x to 14.4x folds over an optimized CPU implementation. Interactive speed of fuzzy object segmentation is reached. We would like to further improve the performance of our CUDA implementation by taking advantage of fast GPU shared memory. In addition, we will eliminate the limitation of device memory and support processing of images larger than GPU memory.

REFERENCES

[1] J. K. Udupa and S. Samarasekera, Fuzzy connectedness and object definition: theory, algorithms, and applications in image segmentation, *Graphical Models and Image Processing*, vol. 58, 1996, pp 246-261.

[2] P. K. Saha, J. K. Udupa and D. Odhner, Scale-based fuzzy connected image segmentation: theory, algorithms, and validation, *Computer Vision and Image Understanding* , vol. 77, 2000, pp 145-174.

[3] J. K. Udupa, P. K. Saha and R. A. Lotufo, Relative Fuzzy connectedness and object definition: Theory, algorithms, and applications in image segmentation, *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 24 (11), 2002, pp 1485-1500.

[4] G.T. Herman and B.M.Carvalho, Multiseeded segmentation using fuzy connectedness, *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 23, 2001, pp 460-474.

[5] J. K. Udupa, L. Wei, S. Samarasekera, Y. Miki, M. A. Buchem and R. I. Grossman, Multiple sclerosis lesion quantification using fuzzy connectedness principles, *IEEE Trans. Med. Imag.*, vol. 16 (5), 1997, pp 598-609.

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

[7] E. Garduño, M. Wong-Barnum, N. Volkmann and M. Ellisman, Segmentation of electron tomographic data sets using fuzzy set theory principles, *Journal of Structural Biology*, vol.162, 2008, pp 368-379 .

[8] Y. Zhou and J. Bai, Multiple abdominal organ segmentation: an atlas-based fuzzy connectedness approach, *IEEE Trans. Information Technology and Biomedicine*, vol. 11, 2007, pp 348-352.

[9] Y. Zhou and J. Bai, Atlas-based fuzzy connectedness segmentation and intensity nonuniformity correction applied to brain MRI, *IEEE Trans. Biomedical Engineering*, vol. 54, 2007, pp 121-129.

[10] T. Lei, J. K. Udupa, P. K. Saha and D. Odhner, Artery-Vein Separation via MRA–An Image Processing Approach", *IEEE Trans. Med. Imag.*, vol. 20 (8), 2001, pp 689-703.

[11] G. Moonis, J. Liu, J. K. Udupa and D. Hackney, Estimation of tumor volume using fuzzy connectedness segmentation of MRI, *American Journal of Neuroradiology*, vol. 23, 2002, pp356-363.

[12] P. K. Saha, J. K. Udupa, E. F. Conant, D. P. Chakraborty and D. Sullivan, Breast Tissue density Quantification via Digitized Mammograms, *IEEE Trans. Med. Imag.*, vol. 20 (11), 2001, pp 792-803.

[13] G. Grevera, J. K. Udupa, D. Odhner, Y. Zhuge, A. Souza, S. Mishra and T. Iwanaga, CAVASS - A Computer Assisted Visualization and Analysis Software System, *Journal of Digital Imaging*, vol. 20, Suppl 1, 2007, pp 101-118.

[14] GPGPU - General Purpose Computation Using Graphics Hardware, http://www.gpgpu.org/.

[15] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. Purcell, A survey of general-purpose computation on graphics hardware, *Computer Graphics Forum*, vol. 26, 2007, pp 80-113.

[16] J. E. Cates, A. E. Lefohn, and R. T. Whitaker, GIST: an interactive, GPU-based level set segmentation tool for 3D medical images, *Medical Image Analysis*, vol. 8, 2004, pp 217-231.

[17] S. S. Samant, J. Xia, P. Muyan-Özçelik, and J. D. Owens, High performance computing for deformable image registration: Towards a new paradigm in adaptive radiotherapy, *Medical Physics*, vol. 35(8), 2008, pp 3546-3553.

[18] NVIDIA Corporation, NVIDIA CUDA compute unified device architecture programming guide, http://developer.nvidia.com/cuda, Jan. 2007.

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

[20] P. Harish, P. J. Narayanan, "Accelerating large graph algorithms on the GPU using CUDA", *High Performance Computing 2007*, vol. 4873, *Lecture Notes in Computer Science*, Springer, pp. 197-208.