Proceedings of the 5th International Conference on Information Technology and Application in Biomedicine, in conjunction with
The 2nd International Symposium & Summer School on Biomedical and Health Engineering
Shenzhen, China, May 30-31, 2008

Implementation of Medical Image Segmentation in CUDA

Lei Pan[1], Lixu Gu[1*], Jianrong Xu[2*]
[1]*School of software, Shanghai Jiaotong University, P.R.China,*
[2]*Shanghai Renji Hospital, Shanghai, P.R.China*

***Abstract*— As the fast development of GPU, people tend to use it for more general purposes than its original graphic related work. The high parallel computation capabilities of GPU are welcomed by programmers who work at medical image processing which always have to deal with a large scale of voxel computation. The birth of NVIDIA® CUDA™ technology and CUDA-enabled GPUs brought a revolution in the general purpose GPU area. In this paper, we propose the implementation of several medical image segmentation algorithms using CUDA and CUDA-enabled GPUs, compare their performance and results to the previous implementation in old version of GPU and CPU, indicate the advantages of using CUDA technology and how to design algorithm to make full use of it.**

***Keywords*— CUDA, GPU, Segmentation, Region Growing, Watershed**

I. INTRODUCTION

Compared to CPU, GPU has more powerful performance in parallel processing and lots scholars had studied the GPU-based image processing[1] and visualization[2] recent years. It is showed in the related researches that GPU technologies (Cg, HLSL, etc.) are powerful weapons in dealing with computation of large scales of data in the medical image processing field. However, such previous GPU technologies always bring difficult and massive programming work. The limitation in the languages and the native insufficiency of the design may cause unclear coding, imprecise computation and instable system. For example, in previous GPU and related technologies, data types are based on the size of the registers in GPU memory. Compared to CPUs they offer only a limited instruction set consisting primarily of mathematics operations which are often graphics specific and in general accept as input a limited number of 32-bit floating point 4-vectors. The fragment processor can output only 4 floating point 4-vectors, usually representing colors [3]. This defect may result in a large scale of waste of memory allocation for unnecessary computation and lead to inefficiency. Codes, variables and procedures that handle data input/output transfer between host and device are fixed. Data computation and parallel arrangements inside the device are rigid and difficult to learn.

In 2007, the birth of NVIDIA® CUDA™ technology and CUDA-enabled GPUs brought a revolution in the general purpose GPU area. NVIDIA® CUDA™ technology is a fundamentally new computing architecture that enables the GPU to solve complex computational problems in consumer, business, and technical applications. CUDA (Compute Unified Device Architecture) technology gives computationally intensive applications access to the tremendous processing power of NVIDIA graphics processing units (GPUs) through a revolutionary new programming interface. CUDA also provides orders of magnitude more performance and simplifies software development by using the standard C language [4].

Segmentation is a fundamental problem in the medical image analysis. The general segmentation problem is the process of partitioning an image or data-set into a number of homogeneous segments [5]. Although the methods of image segmentation have been improved significantly recently, it is still a very difficult problem in practice [6].

In this paper, CUDA technology is employed to implement different segmentation algorithms over real medical images in the experiments and the results are listed. A comparison between CUDA and previous GPU technologies are held. Advantages and disadvantages over CUDA are analyzed.

II. METHODOLOGY

A. Memory arrangement

In CUDA framework, the GPU is viewed as a compute device capable of executing a very high number of threads in parallel. The large scale of original resource data in different format (float, double, int, long, short, etc.) are uploaded from host to device at a higher speed than previous GPU version and then stored as an array in the global memory of the device. It is totally up to programmer to manipulate and access the data array agilely through thread block and grid of thread blocks, which are organized inside the GPU device by CUDA. For example, we can assign a grid of 512*256 (2 dimension) blocks with 512*1*1 (viewed as 1 dimension) threads inside each of its block for a 512*512*256 resource brain image.

Once the threads and blocks assignment is done, the position of each thread can be directly located by using *threadIdx* and *blockIdx* where *threadIdx.x*, *threadIdx.y* and *threadIdx.z* respectively stand for the current thread index of X, Y, Z dimension inside the current block and *blockIdx.x*, *blockIdx.y* respectively stand for the current block index of X and Y dimension inside the current grid. In above example, let *length* be 512 (the first dimension in the block), let *width*

Lei. Pan, Lixu. Gu(e-mail: gu-lx@cs.sjtu.edu.cn) are with the Laboratory of Image Guided Surgery Therapy (IGST), Shanghai JiaoTong University(SJTU), China.
*Lixu Gu and Jianrong Xu are Corresponding Authors.

Authorized licensed use limited to: Shanghai Jiao Tong University. Downloaded on July 27,2010 at 03:10:35 UTC from IEEE Xplore. Restrictions apply.

be 512 (the first dimension in the grid), then concurrent access to each voxel of the image can be easily realized by

$$offset = (blockIdx.y \cdot width + blockIdx.x) \cdot length + threadIdx.x \quad (1)$$

GPU data allocation schemes vary according to different segment methods. Multiple blocks schemes reduce thread cooperation, because threads in different thread blocks from the same grid cannot communicate and synchronize with each other. In this situation, all data operations are not completely parallel. A device usually runs all the blocks of a grid combining both sequentially and in parallel. Compared to the previous example, a balance factor n, which stands for the voxel number that each thread handles, is introduced to control the amount of serial processing in each parallel processing thread.

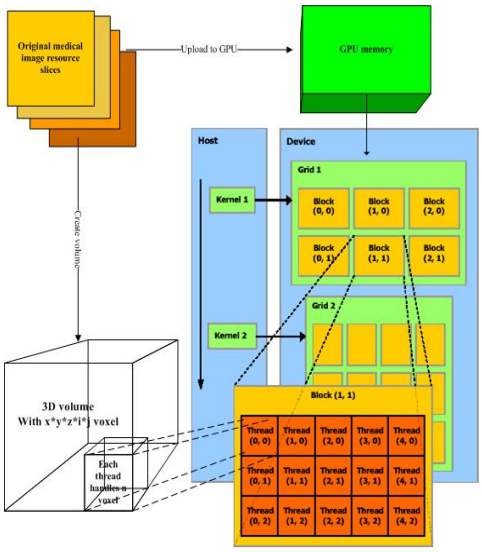

Fig. 1. Medical image data allocation for memory arrangement in CUDA. Three dimensional original medical images (CT, MR, etc.) composed by a series of slices are organized in blocks of threads and grids of blocks. The number of elements that handled by a single thread is easily and completely decided by programmers.

$$T = sync_{thread(x,y,z)}(sync_{thread(i,j)}(n \cdot l \cdot t))$$

$$\max(K_{x,y}) < sync_{thread(i,j)}(K) < \sum_{x=1}^{i}\sum_{y=1}^{j} K_{x,y}$$

Here K stands for the time cost of each thread operation, $sync_{thread(i,j)}(K)$ stands for the total time cost in synchronizing a block with $i \cdot j$ threads. Another symbol l stands for the number of the operation that each voxel requires and symbol t stands for the average time cost of a single operation. The restriction on block dimension (x,y,z) and thread dimension (i,j) varies on different NVIDIA GPU hardware types. Basically there are fixed upper bounds for x*y*z blocks and i*j threads. Let the total voxel number in the resource medical image be v, then $\frac{v}{n} = x \cdot y \cdot z \cdot i \cdot j$. Since x, y, z, i, j depend on the hardware and l, t depend on the algorithms we design, our goal is to find a proper n which results a minimum time cost T in both the parallel and serial processing.

B. Segmentation algorithms implementation

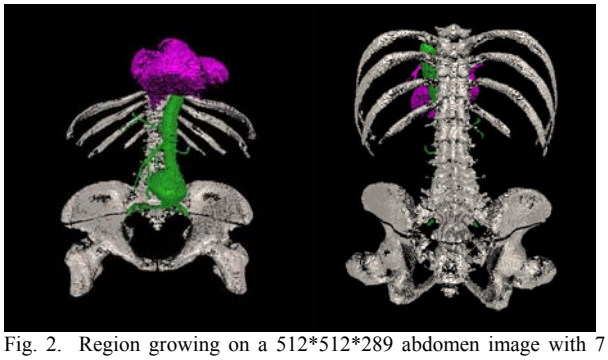

Fig. 2. Region growing on a 512*512*289 abdomen image with 7 seeds located in different parts of the image after 120 iterations by CUDA on NVIDIA Geforce 8500 GT. The result includes heart(red), artery(green) and bone(silver). In this experiment, when n=3*3*3, the total time cost reaches the minimum 12.875 seconds.

Segmentation algorithms can be divided in to two major groups as model based and region based [7]. We select and implement region growing and watershed methods of region based group as a test of CUDA implementation in segmentation field. The level-set method [8] of model based group and other complex algorithms are still under test.

1) Region Growing: In order to make full use of the parallel computation capability of GPU, it is recommended to select as many seeds of different parts of the image as possible. Region growing starts from each seed inside the image. We compare and label all six neighbors (up, down, left, right, front, back) of each voxel based on intensity or other defined rules. Data of each voxel can be conveniently located in CUDA by using (1) since the image data has been allocated into GPU memory according to a specific scheme. In the example of (1), the offset of six neighbors of each voxel in the GPU memory can be easily calculated by

$$\begin{aligned} left &= offset - 1 \\ right &= offset + 1 \\ up &= offset - width \cdot length \\ down &= offset + width \cdot length \\ front &= offset - length \\ back &= offset + length \end{aligned} \quad (2)$$

However, multiple seeds can lead to over growing problem and growing competition. In order to solve the over growing problem in some images where different parts still have connections after the thresholding, several times of erosion and dilation can be of great help[9]. Both erosion and dilation operation need six neighbors information from (2) in order to decide whether current voxel is eroded or dilated.

Since the cost of adding seeds are not as expensive as in CPU, we can easily regain the lost parts in the branches which are missing during the erosion by adding a seed there. New Seeds

$$S_{n+1} \in G(S_1, S_2, \cdots S_n) - G'(S_1, S_2 \cdots S_n)$$

Here S_n stands for seed voxel, $G(S_1,S_2 \ldots S_n)$ stands for the result voxel set after region growing without erosion and dilation from seeds $S_1, S_2 \ldots S_n$. $G'(S_1,S_2 \ldots S_n)$ stands for the result voxel set after region growing using erosion and

Authorized licensed use limited to: Shanghai Jiao Tong University. Downloaded on July 27,2010 at 03:10:35 UTC from IEEE Xplore. Restrictions apply.

dilation from seeds S_1, S_2 … S_n.

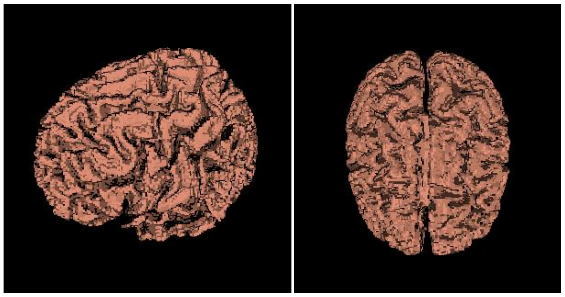

Fig. 3. Region growing on a 256*256*256 brain image with 6 seeds after 40 iterations by CUDA on NVIDIA Geforce 8500 GT. In this experiment, when n=1*1*1, the total time cost reaches the minimum 2.435 seconds

The final segmentation result is stored in an array of specific data type in GPU memory, which is to be transferred back to host CPU when display is needed.

2) Watershed: Like common watershed algorithm, gradient values of the original image are obtained and gauss blurring is deployed before immerse the lowest gradient. Both operations can be conveniently achieved in GPU by some widespread methods. Watershed algorithm starts with immersing from the lowest gradient value voxel instead of region growing from seeds [10]. The immersing rule also takes neighbor voxel into consideration:

(1) When all neighbors has not been labeled (at the higher level), then current voxel is labeled as a new basin.

(2) When all labeled neighbors belong to the same basin or watershed line, label the current voxel as the same basin with its neighbor.

(3) When two labeled neighbors belong to different basin exist, label the current voxel as the watershed line.

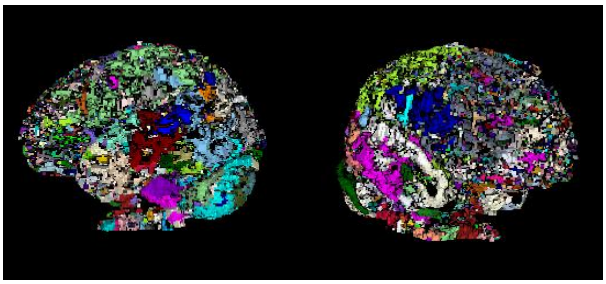

Fig. 4. Multi-degree immersing watershed method on the same image of Fig. 3. Superabundant regions (2871 regions in this image) caused by over segmented is still a serious problem in this image although multi-degree immersing watershed method is employed and efficiency is improved. However, such problem is more acceptable in small images like brain than in large images like abdomen. In order to get the specific region, post processes on this result is needed. Interactive and manual operation is also provided to users.

Since CUDA can exactly handle and locate each voxel in the device at the same time. It is no need to sort the gradient value and create a FIFO queue to handle the waiting voxel in the iteration. However, the previous serial watershed algorithm is not suitable for the parallel computational ability of CUDA. Over segmented problems are more serious after immersing operation in CUDA because basin number increases during the parallel labeling. Some data structures have been abandoned and improvement and adjustment in immersing step has been made.

In order to solve the common over segmented problems in watershed method, the multi-level immersing watershed method [7] can be improved to be implemented in our experiment.

$$h' = h + Diff(p,h)$$

The new height *h'* of the gradient value of each voxel to be immersed in current iteration is the previous h plus *Diff(h,p)*, where

$$Diff(p,h) = \sum_{q \in \{p, N_G(p)\} \cap I(p,h) \geq I(q,h)} (I(p,h) - I(q,h))/n \quad [11]$$

Here *Diff(p,h)* is dynamically calculated after each immersing iteration. *I(p,h)* and *I(q,h)* respectively stand for the intensity or gradient value of the voxel set *p* and their neighbor voxel sets *q* at the current height h. $N_G(p)$ stands for the neighbor area within a specific distance from voxel *p* in the graphic *G*. The computation of large scale sum of the *I(p,h)* and *I(q,h)* of the whole image makes full use of the parallel capability of CUDA.

After immersing to the highest gradient voxel, watershed algorithm stops. Post processing includes merging some basins and remove irrelevant watershed lines.

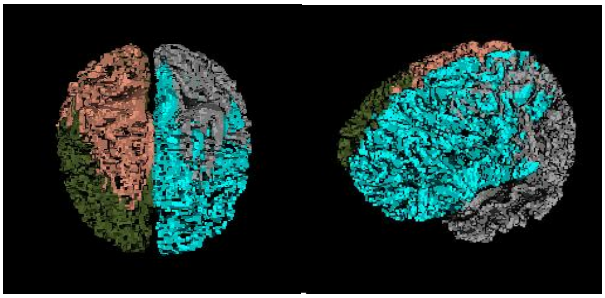

Fig. 5. Result after manually combining specific regions of Fig. 4. Related regions are selected by users in the post process step. Selected regions are combined and marked with different colors (blue, green, gray, brown) according to different requirements for diagnostic uses.

III. EXPERIMENTAL RESULTS

Our experiments of different segmentation algorithms are employed on NVIDIA Geforce 8500 GT with 256MB global memory. Other configurations are Windows-XP Professional operation system, Intel Pentium D 2.80GHz, 1.50GB RAM.

A. Region Growing

In region growing algorithm, thresholding, erosion and dilation are employed on abdomen image (Fig. 2) and brain image (Fig. 3). It is indicated in the experimental result that CUDA takes little advantage on efficiency over Cg on large size data. However, CUDA has an obviously better performance on multiple seeds growing for multiple parts in a single image than serial CPU.

B. Multi-Degree Immersing Watershed

In multi-degree immersing watershed algorithm, immersing rules are strictly followed although the current immersing height is dynamically calculated every iteration. In

Authorized licensed use limited to: Shanghai Jiao Tong University. Downloaded on July 27,2010 at 03:10:35 UTC from IEEE Xplore. Restrictions apply.

this experiment, Cg technology was not employed because of lack of resource. Experiment on abdomen image returns an unsatisfied result with large amount of over-segmented regions and needs additional post process. However, the result of CUDA and CUDA-enabled GPU on brain image (Fig. 4) apparently exceeds the result of CPU that proposed by Shengcai Peng[11]. It is showed in the result (2871 regions left) that although CUDA technology can improve the parallel computation, this feature is not entirely suitable for multi-degree immersing algorithm. Post process[11], which includes region selection, combination and other operations (Fig. 5), is recommended for diagnostic uses.

TABLE I
Experiment on Region Growing Method

Hardware configuration	ABDOMEN IMAGE / 512*512*289	Brain image / 256*256*256
CUDA / NVIDIA Geforce 8500 GT	12.875 sec (7 seeds, 120 iterations)	2.435 sec (6 seeds, 40 iterations)
Cg / NVIDIA Geforce 6800	13.62 sec (16 seeds, 120 iterations)	4.42 sec (16 seeds, 60 iterations)
Intel Pentium D 2.80GHz	21.034 sec(7 seeds, 100 iterations)	Not implemented

Experiment using Intel Pentium D CPU on brain image was not implemented.

In the experiment of region growing method, the number and position of seeds are selected by user.

TABLE II
Experiment on Multi-Level Watershed Method

Hardware configuration	ABDOMEN IMAGE	Brain image
CUDA / NVIDIA Geforce 8500 GT	155.965 sec (512*512*289)	45.223 sec (256*256*256) 2871 regions
Pentium 4 2.4GHz, 2GB RAM	Not implemented	115 sec (181*217*181) 850 regions

Experimental result of Pentium 4 CPU on a 181*217*181 brain image is proposed by Shengcai Peng[11].

Result of CUDA on 512*512*289 abdomen image does not include 0.281 sec + 0.293sec data transfer times.

Different data arrangements cause different memory allocations. While it is out of memory on GPU for a whole image, user needs to separate the data into parts and transfer them between host and device one by one.

Time cost in post process of the brain image is not included in the result.

IV. CONCLUSION

In this paper, we introduced novel implementations of segmentation algorithms based on a novel technology and new type hardware. Using CUDA technology and CUDA-enabled GPUs we can easily implement basic segmentation algorithms in different operating system and integrate them into different framework. However, the advantages in performance and efficiency on CUDA over old version GPU with Cg or HLSL and simply CPU depend on several factors.

1) Allocate the voxel data of the resource medical image into threads in GPU memory properly. Try to handle as much data as possible at each computational iteration cycle and try to balance the parallel processing and serial processing.

2) Make full use of every block in the limited grids, since the limitation of number and dimension of the threads in a block is more rigid than the limitation of the blocks in a grid.

Future work includes improvement of the current segmentation algorithms to be better implemented in new CUDA standard 1.1. Another work is to design more efficient methods and implement more complex algorithms like level-set and Snake.

ACKNOWLEDGMENT

This paper is partially supported by the Chinese National Natural Science Foundation under Grant No. 30770608, Chinese National 863 research foundation under Grand No. 2007AA01Z312 and the National Fundamental Research Program (973) under Grant No. 2006CB504801 and 2007CB512701. The authors would like to thank Sizhe Lv and Pengfei Huang for their suggestions to the work.

REFERENCES

[1] I. Buck, "GPGPU: General-purpose computation on graphics hardware - GPU computation strategies & tricks." ACM SIGGRAPH Course Notes, 2004.

[2] Yang Heng and Lixu Gu, "GPU-based Volume Rendering for Medical Image Visualization", Proceedings of the 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference Shanghai, China, September 1-4, 2005

[3] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kr¨uger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," in Proceedings of Eurographics 2005 - State of the Art Reports, pp. 21–51, Aug. 2005. Dublin, Ireland, August 29 – September 2.

[4] NVIDIA® Corporation. (2007, June, 23). NVIDIA CUDA Programming guide version 1.0. Available: http://developer.download.nvidia.com/compute/cuda/1_0/NVIDIA_CUDA_Programming_Guide_1.0.pdf

[5] Yang, F., Gu, L., Xu, J., Yang, J.: The methodology of multi-level watershed 3D medical image segmentation. Int J CARS, 2006, 1:461-485

[6] Kaus, M., Warfield, S.K., Nabavi, A., Black, P.M., Jolesz, F.A., Kikinis, R.: Automated segmentation of mri of brain tumors. Radiology 218 (2001) 586–591

[7] Lefohn, A., Cates, J., Whitaker, R.: Interactive, GPU-based level sets for 3D brain tumor segmentation: Supplementary information. http://www.sci.utah.edu/~lefohn/work/rls/tumorSeg (2003)

[8] A. E. Lefohn and R. T. Whitaker, "GPUbased, three-dimensional level set solver with curvature flow," technical report uucs-02-017, School of Computing, University of Utah, 2002. Available: http://graphics.cs.ucdavis.edu/~lefohn/work/rls/gpuLevelSet3D-TR.pdf.

[9] L.Gu, T.Kaneko, "Extraction of Organs Using Three-Dimensional Mathematical Morphology", Systems and Computers in Japan, Vol.31-7, 2000, pp.29-37.

[10] S. Beucher and F. Meyer, "The morphological approach to segmentation: The watershed transform," in Mathematical Morphology in Image Processing, E. R. Dougherty, Ed. New York: Marcel Dekker, 1967

[11] Shengcai, P., Lixu, G.: A Novel Implementation of watershed Transform using Multi-Degree Immersion simulation, 2005, 27th Annual International Conference of IEEE Engineering in Medicine and Biology Society.

Authorized licensed use limited to: Shanghai Jiao Tong University. Downloaded on July 27,2010 at 03:10:35 UTC from IEEE Xplore. Restrictions apply.