Design and Storage Optimization of GPU-based Parallel Program of Image Registration for Remote Sensing

Haifang Zhou [1] Rulin Xu and Jingfei Jiang.

Abstract. Image registration is a crucial step of many remote sensing related applications. As the scale of data and complexity of algorithm keep growing, image registration faces great challenges of its processing speed. In recent years, the computing capacity of GPU improves greatly. Taking the benefits of using GPU to solve general propose problem, we research on GPU-based remote sensing image registration algorithm. A mutual information based wavelet registration algorithm is proposed on the GPU parallel programming model, and storage optimization strategy is applied on the registration process. Using CUDA language, we tested our proposed methods with nVIDIA Tesla M2050 GPU. The experiment results prove that the parallel programming model and storage optimization strategy can well adapt to the field of remote sensing image registration, with a speedup of 19.9x. Our research also shows that the GPU-based general propose computing has a bright future in the field of remote sensing image processing.

Keywords: GPU Mutual information Image registration Parallel algorithm.

1.1 Introduction

Processing two or more images and other information sources about the same target, remote sensing image registration performs space transformation on them based on the same reference frame. All these information being processed is gotten at different time and different angle, which is collected by similar or diverse sensors [1]. Remote sensing image registration is an important step of the remote

[1] Haifang Zhou (✉)
College of Computer, National University of Defense Technology, Changsha, China
e-mail: haifang_zhou@163.com

sensing related applications, the outcome of which leads a direct impact on its following processes..

As remote sensing technology develops steadily, the time, space and spectral resolutions of remote sensing images become higher, and the scale of remote sensing data increases greatly. Meanwhile, with the growing requirements on accuracy of the result of remote sensing image registration, the complexity of such process continues to rise. Therefore, the processing speed of remote sensing image registration meets great challenges. Traditional methods accelerate the process with multi-CPU and perform it in parallel. Le Moigne et al. [2] designed a fine-grained parallel registration algorithm based on multi-resolution analysis, while Ozkan et al. [3] proposed a genetic algorithm based fine-grained parallel algorithm.

In recent years, GPU (Graphic Process Unit) have gained rapid development in the field of general propose computing. It provides a new way of researching and exploring remote sensing images registration. Since GPU was produced, its processing capability grows with a speed beyond the Moore law [4]. The FP (Floating-Point) computing capability of the existing nVIDIA Geforce GTX 590 GPU has reached 4.96Tflops. Using GPU to solve general propose computing problems has the following advantages: (1) GPU contains lots of the parallel computing core, which provides more resources for computing; (2) With its high storage bandwidth, it can speed up data transmission; (3) Its overhead of switching threads is low, which can improve overall performance; (4) It adopts read-only cache, accelerating the processes to obtain data. Recently, CPU-GPU heterogeneous platform has made rapid development too, and it has already been well used in many fields as mentioned in [5][6].

To solve the processing speed problem of remote sensing image registration, in this paper, we research on some GPU-based accelerate methods which combines the research hotspot of GPU-based general propose computing.

1.2 CUDA Programming Model and Storage Structure

The GPU accelerate platform we used in our research is from nVIDIA, and we adopted nVIDIA CUDA language to develop GPU applications.

The CUDA langrage [7] developed by nVIDIA company is an extension of the C language. It mainly uses APIs to call for some basic services, which makes the programmer who is familiar with C langrage could design and develop general propose computing applications quickly. In the CUDA programming model, CPU is considered as host while GPU is treated as device, and program is operated in the SIMT (Single Instruction Multiple Thread) manner. Device creates many threads during its operation. Many threads form block, and then blocks form grid.

CUDA improves its performance by obtaining program level parallelism with lots of threads. This kind of parallelism includes two types: fine-grained thread parallelism and coarse-grained thread parallelism. Fine-grained thread parallelism

indicates the parallel relationship between threads in a block, while coarse-grained thread parallelism is the parallel relationship between blocks.

1.3 Mutual information wavelet registration algorithm and its parallel programming model

1.3.1 General idea

The complexity of the serial wavelet registration algorithm based on the similarity of mutual information focuses on two points, i.e. four-level wavelet decomposition and mutual information calculation. These two processes will be used to correct registration parameter constantly.

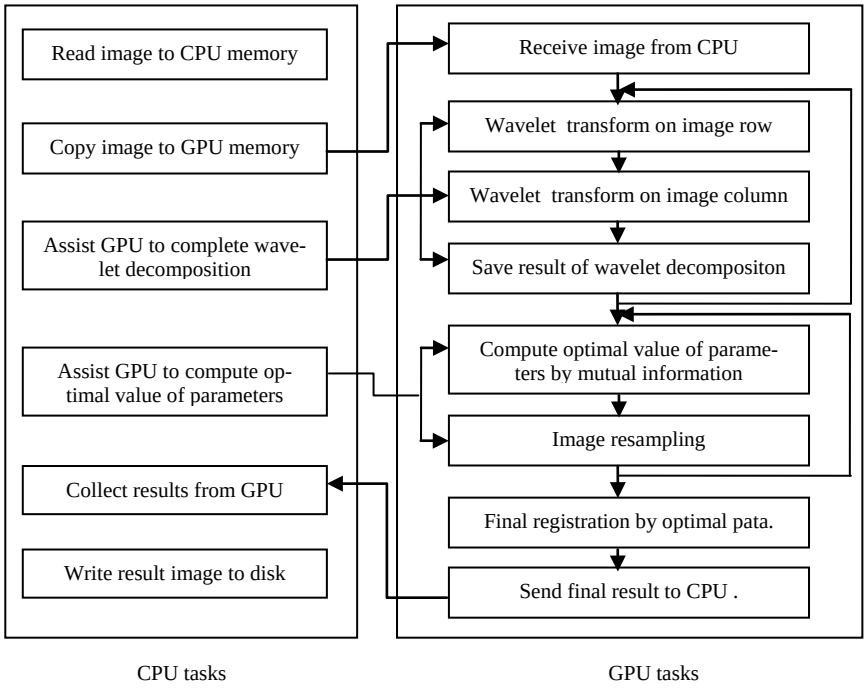

Fig. 1.1 The task partition of CPU and GPU.

While designing the parallel program of the mutual information based serial wavelet registration algorithm, there are many functions whose complexities are

directly related to the size of the remote images. These functions include column wavelet decomposition, line wavelet decomposition, re-sampling, mutual information calculation, and image registration using the most effective parameters. We map all these functions to GPU functions, so as to accelerate the computing process.

Figure1.1 shows how we assign tasks between CPU and GPU. Firstly, CPU loads the image data into memory, and then, it transfers the data to the GPU Global Memory. After that, CPU assists GPU to perform wavelet decomposition by calling GPU's wavelet decomposition kernel at each level of wavelet decomposition. CPU takes part in the process of calculating the most effective parameters mainly by calling GPU kernels (such as the kernels of processing mutual information) and performing some complex operations like comparison and accumulation. At last, CPU stores the results generated from GPU and writes them back to some specified locations in memory.

In our implement, we design individual GPU kernel for each process, i.e. column wavelet decomposition, line wavelet decomposition, re-sampling and image registration. All these kernels are called by GPU to accomplish computation. Functions of column wavelet decomposition and line wavelet decomposition are similar, which need extra constrain about the width of column and line, so as to limit the number of threads. While processing mutual information, as the whole process includes many sub-processes, such as judging the maximal and minimal greyscale of the reference image and target image, performing statistical analysis on greyscale, calculating the marginal distribution and each point's greyscale and so on, we divide the function into four kernels. GPU first divides its task according to the blocks of each grid using corresponding kernel, and then, it judges the local maximal and minimal greyscale of each block, which will be transferred to CPU later. CPU operates complex process to obtain the overall maximal and minimal greyscale, and send the result back to GPU. GPU uses another kernel to generate the joint histogram of each pixel of these two images, and CPU will finish the final statistic with the outcome from GPU. After that, GPU calculates the marginal distribution of the images and use that to generate the mutual information of each pixel. At last, CPU takes accumulation operation to process the local mutual in-formation generated by GPU and obtains the overall result which will be used to correct the registration parameters.

1.3.2 Parallel mapping model of kernels

The data of remote image is organized as two-dimensional array. It is required that CUDA threads could be organized in the same way, and kernels could be mapped in parallel to support fine-grained thread parallelism. We design the parallel mapping model based on the following mathematic model.

Assume V is the range corresponding to the domain *X*, and *T* is the set of GPU threads. *X* consists of two parts, i.e. X_1 and X_2, which are organized in the same way. $X_1 \vee X_2 = X$, $X_1 \wedge X_2 = \Phi$ (Φ indicates empty set). In additional, the number of elements in X_1, X_2, V and T are equal, which is indexed from 0 to n. We define the following functions:

$$f_1 : f(x_i, y_j) = \begin{cases} t_i, & i = j \\ -1, & i \neq j \end{cases} \qquad f_2 : f(t_k) = v_k$$

$x_i \in X_1$, $y_j \in X_2$, $t_i \in T$, i and j are indexes of the element in X_1, X_2 and *T*. $t_k \in T$, $v_k \in V$, k is the index of the element in *T* and *V*. Function f_1 maps the elements with the same index in X_1 and X_2 to the thread with the same index in *T*, and -1 indicates there is no thread mapped to the element in *X*. Function f_2 shows each thread generates only one result, and the result is stored to the location with the same index in *V*.

Utilizing the relationship of the above functions to realize remote image registration, we assign the target image data stored in GPU global memory to X_1, assign the reference image data to X_2, and assign the outcome of image registration to *V*. All these date sets are organized as two-dimensional array, so is the thread set *T*. While realizing it in parallel, threads use the built-in variables of the thread grid to index the data of images and the location to store results.

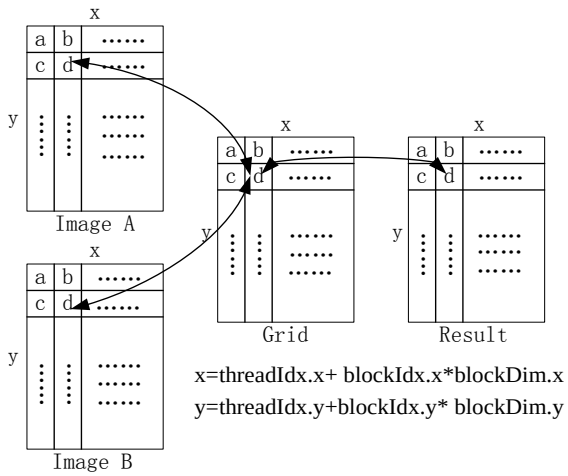

Fig. 1.2 GPU parallel mapping model of remote sensing image registration.

The thread grid has 8 built-in variables to locate the threads, i.e. threadIdx.x, threadIdx.y, blockIdx.x, blockIdx.y, blockDim.x, blockDim.y, gridDim.x and gridDim.y. Among them, threadIdx.x and threadIdx.y are the horizontal and longitudinal index of the thread in the block; blockIdx.x and blockIdx.y are the horizontal and longitudinal index of the block in the grid; blockDim.x and blockDim.y indicate the amount of threads in the block in horizontal and longitudinal; grid-

Dim.x and gridDim.y indicate the amount of blocks in the grid in horizontal and longitudinal. We define another two variables, i.e. x and y, to indicate the index of image data and thread grid in the horizontal and longitudinal direction respectively. Therefore, there are the following relationship between image data and thread grid:

$$x = threadIdx.x + blockIdx.x * blockDim.x$$
$$y = threadIdx.y + blockIdx.y * blockDim.y$$

According the equation above, each thread finds its data to be processed and the location to store result. The parallel mapping model of remote sensing image registration is shown as figure1.2.

1.4 Data Flow Oriented Storage Optimization

1.4.1 Asynchronous data transfer between CPU and GPU

CUDA parallel program is executed on the CPU-GPU heterogeneous platform. In the parallel algorithm of mutual information based wavelet registration, the tasks of CPU and GPU are divided as shown in figure 2. CPU assigns memory space for GPU to store data, loads target data to GPU, controls GPU to call for kernel and stores the outcome generated by GPU. GPU is responsible to execute kernels and keep some temporary results. As there is a large amount of data needs to be transferred between CPU and GPU, if CPU starts to transfer data to GPU and wait until all the data has been transferred without processing the following tasks, it would stay idle during the transfer process and waste its computing resources. Therefore, while CPU loads the image data to the Global Memory in GPU, the data should be transferred asynchronously. In this case, CPU needs not to wait until data transfer is finished, which overlaps computing and data transferring.

1.4.2 Ooptimization of memory access for thread data

Access to Shared Memory is much faster. With the help of GPU Shared Memory, we can improve the program performance greatly. In our GPU-based algorithm proposed in this paper, the registration data is stored in the Global Memory. However, because of its slower speed of access, if threads get each data from the Global Memory while executing, the performance of the program will be degraded. For this reason, we use Shared Memory to optimize the performance. We locate

the registration data of each block in the Shared Memory, so that the threads in each block can access data through the Shared Memory. Note that if several threads access the same data, i.e. when Bank Conflict occurs, then threads would have to access in serial, which affects the program performance. When the data is transferred from Global Memory to Shared Memory, we should try to avoid Bank Conflict.

First, we should divide the image data into (n+1)×(n+1) parts, and create the required data for (n+1)×(n+1) blocks. Each block asks for its own space in the Shared Memory, in order to store the data needed by thread in block. For instance, Part(0,0) has four image pixels, and pixel 1 is stored in the same location in the Block(0,0) in the Shared Memory. The others are processed in the same way.

We adopt the following method to map the target image data to the Blocks in the Shared Memory.

__shared__ unsigned char fimage[16][16];//declare the space for reference image in Shared Memory

__shared__ unsigned char mimage[16][16];// declare the space for target image in Shared Memory

j=threadIdx.x+blockIdx.x*blockDim.x;//index x of the source data

i=threadIdx.y+blockIdx.y*blockDim.y;// index y of the source data

fimage[threadIdx.y][threadIdx.x]=fixedimage[i*width+j];//copy data

mimage[threadIdx.y][threadIdx.x]=movedimage[i*width+j];//copy data

After the image data is copied from Global Memory to Shared Memory, threads in blocks can directly use the built-in variable of the block to locate image data. Each thread accesses its own target image data in different places in the Shared Memory, avoiding the existence of Bank Conflict.

1.5 Data Flow Oriented Storage Optimization

In our experiment, we adopt Intel Xeon5670 CPU, which has 6 cores and 12MB 2-Level Cache, supporting 12 threads with 2.93GHz frequency. The GPU we used is nVIDIA Tesla M2050, which has 448 cores. The operate system is Red Hat Linux 5.0.

The benchmarks we tested are five groups of images, and the size of images is scaled up 4 times. During the experiment, the process of loading images and writing back results are necessary and it introduces the same overhead in both the serial program and CUDA parallel program. However, image registration is the main task costs more time. Therefore, we only record and analyse the overall time exclude loading images and storing results. Table 1 shows the results of our experiment.

Figure1.3 shows the speedup of CUDA parallel program compared with the serial program. According to the curve in this figure, as the image scales up from 256×256 to512×512, the speedup increases slowly. Along with the image scales up to 2048×2048, the speedup increases rapidly. However, while the image scales

up from 2048×2048 to 4096×4096, the increasing trend of speedup declines a little. It is mainly because CPU has enough capacity to complete computation quickly when it processes small size images. However, the initialization and communication of GPU may lead considerable huge overhead which af1.fects the benefits GPU acceleration could bring. As the computing scale extends, the extra overhead introduced by GPU would become inconspicuous compared to the time GPU costs to process data. Meanwhile, serial program makes CPU reaches its computing limit, which leads to the executing time increases linearly. Therefore, the speedup increases linearly when the image size scales up from 512×512 to 2048×2048. Nevertheless, if the computing scale extends further, the executing time of GPU would increase linearly, which makes speedup grows steadily.

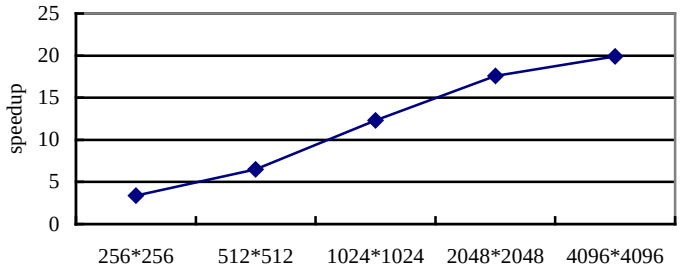

Fig. 1.3 The speedup of GPU-based mutual information wavelet registration algorithm.

Based on the analysis above, we can conclude that the time needed for executing serial or CUDA parallel program changes differently along with the image scale alters. When task could not exhaust all the CPU's computing resources, GPU speedup increases gradually. In additional, according to the test results, the CUDA parallel programming and storage optimizing method proposed in this paper can effectively accelerate the process of remote sensing image registration, which can be referred by other GPU-based accelerate algorithms.

1.6 Conclusion

Focusing on the challenges of processing speed confronted with remote sensing image registration, in this paper, we highlight the importance to accelerate its execution. Our test results show that as the size of image increases, GPU-based acceleration can bring greater improvement on performance. It is proved that the parallel model and storage optimization method proposed in this paper can well adapt GPU-based acceleration. Our research also shows that GPU-based general propose computing has a bright future in the field of remote sensing image registration. We will take further research on this topic combining with MPI technology on the multi-CPU-GPU acceleration platform in the near future.

1.6 Acknowledgment

This work was supported in part by the National Natural Science Foundation of China (NSFC) No. 61272146.

1.7 References

1. Brown L. A survey of image registration techniques. ACM Computing Surveys, 1992. 24(4): 325–375
2. Le Moigne J. Towards a Parallel Registration of Multiple Resolution Remote Sensing Data. //Proc of IGARSS'95. Piscataway, NJ: IEEE Press. 1995: 1011 - 1013 vol.2
3. Ozkan M, Fitzpatrick J. M, Kawamura K. Image Registration for a Transputer-Based Distributed System. //Proc of the 2nd International Conference on Industrial & Engineering Applications of AI & Expert Systems (IEA/AIE-89). New York, USA: ACM Press. 1989: 908-915
4. Owens J, Luebke D, Govindaraju N, et al. A survey of general-purpose computation on graphics hardware . Computer Graphics Forum. 2007,26(1):80-113
5. Cederman D, Tsigas P. A Practical Quicksort Algorithm for Graphics Processors // LNCS 5193: Algorithms - ESA 2008, 16th Annual European Symposium 2008. Berlin: Springer, 2008: 246-258
6. Giorgos Vasiliadis, Spiros Antonatos, Michalis Polychronakis. Gnort: High Performance Network Intrusion Detection Using Graphics Processors //LNCS 5230: Proc of the 11th International Symposium On Recent Advances In Intrusion Detection（RAID）. Berlin: Springer, 2008: 116-134
7. NVIDIA CUDA. [2011-05-21] http://www.nvidia.com/object/cuda_home_new.html 20 May. 2013