The Fifteenth International Symposium on Artificial Life and Robotics 2010 (AROB 15th '10),
B-Con Plaza, Beppu,Oita, Japan, February 4-6, 2010

Fast processing method for PIV using GPGPU

Koji Miyazaki and Kikuhito Kawasue

University of Miyazaki, 1-1 Gakuen Kibanadai-Nishi, Miyazaki 889-2192, Japan
(Email: ng3804u@student.miyazaki-u.ac.jp)

***Abstract*:** In the present study, the ability to apply general purpose computation on graphics processing units (GPGPUs) to particle image velocimetry (PIV) is confirmed, and the processing speed of the PIV is accelerated by GPGPUs. Our code is based on the direct cross-correlation method, where one of the PIV algorithms is rewritten for GPGPU computing using the CUDA tool kit. The results of a performance test indicate that GPU computing for PIV demonstrated an excellent acceleration rate of more than 100 times greater than CPU computing while maintaining acceptable precision.

Keywords: GPGPU, CUDA, PIV, Parallel computing, Image processing

I. INTRODUCTION

In our research, the processing speed of particle image velocimetry (PIV) was accelerated by general-purpose computation on graphics processing units (GPGPUs), which has been considered as a new acceleration technique for computing. Computations using GPGPUs use the graphics processing unit (GPU) on a graphics card for not only image processing but also general purposes such as numerical simulation [1]. The acceleration of processing using GPGPUs has been attempted for various numerical simulations, such as computational fluid dynamics and medical image processing [1]-[4]. However, reports on the application of GPGPUs to PIV are rare. Therefore, we applied GPGPU to PIV, which is an image-based measurement method, and evaluated the resulting performance. The computational errors related to GPGPUs should be also examined. Since most GPUs do not have double-precision arithmetic units, the results of computing on GPUs for application to PIV may have problems related to precision. Therefore, the results of the PIV analysis obtained using a GPGPU were compared to those obtained using a CPU.

Particle image velocimetry is a useful tool in the study of transient fluid flow phenomena [5][6]. The velocity vectors of a fluid are obtained by measuring the translational displacement of tracer particle during a short time interval without interfering with the flow. In general, the direct cross-correlation (DCC) method, which is a PIV algorithm, offers improved accuracy as compared with the FTT-based cross-correlation method. However, the DCC method requires several process iterations, unlike the FTT-based cross-correlation method. In the case of actual analysis, as the size of the analysis region becomes larger, the processing time increases, often reaching several minutes. Since this is problematic in the case of practical application, the processing speed using the DCC method should be accelerated. In the DCC method, in order to determine the displacement of a small region, cross-correlations between the interrogation region and several candidate regions must be calculated, and the associated processing comprises the majority of the calculation costs for the whole process. The calculation of all of the cross correlations was parallelized and accelerated using a GPGPU with CUDA, which is one of the software development environments for GPGPUs used in the present study. CUDA is freely distributed by NVIDA and extends the C language for use with the GPU, which has an excellent architecture for parallelism. As a result of a performance test, computing on the GPU for application to PIV exhibits an excellent acceleration rate of more than 100 times greater than CPU computing. In this case, approximately 95% of the entire process was executed on the GPU.

II. GPGPU

1. GPU hardware architecture

Figure 1 describes the architecture of the NVIDIA GeForce GTX 275 GPU used in this study, and Table 1 lists its hardware specifications. The GPU chipset of the GeForce GTX 275 is the GT200, which can operate in graphics mode or parallel computing mode. In GPGPU computing, the parallel computing mode is used. In graphics processing mode, the GPU architecture consists of ten texture processing clusters, and, in parallel computing mode, the GPU architecture consists of ten thread processing clusters. Here, TPC stands for texture processing cluster in graphics processing mode and thread processing cluster in parallel computing mode. Each TPC is made up of three streaming multiprocessors (SMs), each of which contains eight streaming processors (SPs), which are either processor cores or thread processors. The total number of SMs on the GT200 GPU is thirty. Unlike the general CPU architectures, the GPU can perform zero-cost hardware-

based context switching, i.e., the GPU can immediately switch to another thread process, and it supports over thirty thousand concurrent threads in hardware (see Table 1). However, the maximum number of concurrent threads depends on environmental variables such as the executed kernel size (program size) in the GPU. Although double-precision arithmetic units have been implemented in the GT200 GPU, only one unit is implemented per SM and thirty units are implemented per GPU. Therefore, the performance of the double-precision arithmetic may be poor.

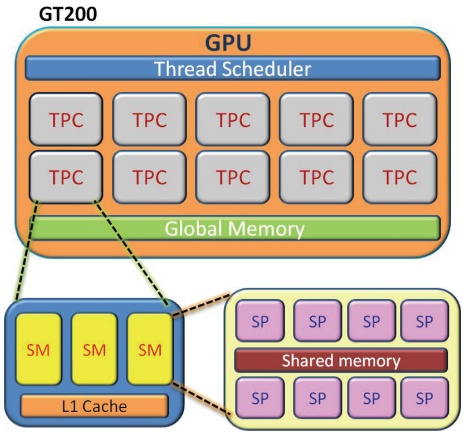

Fig. 1 NVIDIA GeForce GTX 275 GPU architecture in parallel computing mode

Table 1 NVIDIA GeForce GTX 275 hardware specifications

Texture processor clusters (TPCs)	10
Streaming multiprocessors (SMs) per TPC	3
Super function units (SFUs) per SM	2
Streaming processors (SPs) per SM	8
Total SPs (Cores)	240
Maximum concurrent threads per SM	1,024
Total maximum concurrent threads	30,720
Double-precision arithmetic units per SM	1
Peak floating point performance GFLOPS	933
Global memory size	869 MB
Shared memory per SM	16 KB
Registers per SM	16,384 registers

2. Software development environment for GPGPU

Cg is a conventional software development tool kit for GPU application software. Because the Cg is the specifically designed programming tool kit for computer graphics, it is not suitable for the development of GPGPU application software. However, recently, software development environments, such as CUDA and OpenCL, which consist of a complier, libraries, and useful tools for GPGPUs have been made freely available. CUDA is a C language development environment for NVIDA CUDA-enabled GPUs but OpenCL can be used for both NVIDA and ATI GPUs, which support the OpenCL architecture. In the present study, we used CUDA as the development tool of the program code for GPGPU computing.

III. PIV ALGORITHM

1. Direct cross-correlation (DCC) method

In PIV, the velocity information is estimated using two consecutive images detected by a camera. Figure 2 shows an example of standard PIV consecutive images [7], and illustrates the procedure of the direct cross-correlation method. In order to extract the velocity at each location, a normalized cross-correlation coefficient between the location (x, y) in the first image and the location ($x+dx$, $y+dy$) in the second image is defined as follows:

$$R_{fg}(dx,dy)=\frac{\sum_{i=1}^{N}\sum_{j=1}^{N}\{f(x_i,y_j)-f_m\}\{g(x_i+dx,y_j+dy)-g_m\}}{\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N}\{f(x_i,y_j)-f_m\}^2\sum_{i=1}^{N}\sum_{j=1}^{N}\{g(x_i+dx,y_j+dy)-g_m\}^2}} \quad (1)$$

where $f(x,y)$ and $g(x,y)$ are the intensities at location (x,y) in the first image and the second image, respectively, and N denotes the size of a side of the interrogation region, which is determined considering the concentration of tracer particles and the maximum velocity of the flow. This cross-correlation coefficient is used to estimate the translational displacements of tracer particles between two consecutive images. The displacement at location (x,y) between two images is estimated as follows. First, the values of the above cross-correlation coefficient are calculated between location (x,y) in the first field and the possible point $(x+dx,x+dy)$ in the second field. Second, the dx and dy values of the displacement are changed, and the cross-correlation coefficient is recalculated. By repeating this procedure, the best estimate of the translational displacement of the tracer particles is determined as the pair (dx,dy) that maximizes the correlation coefficient. The global velocity profile of the flow can be obtained by repeating the above computation at every location. Figure 3 shows a flow chart of the DCC method.

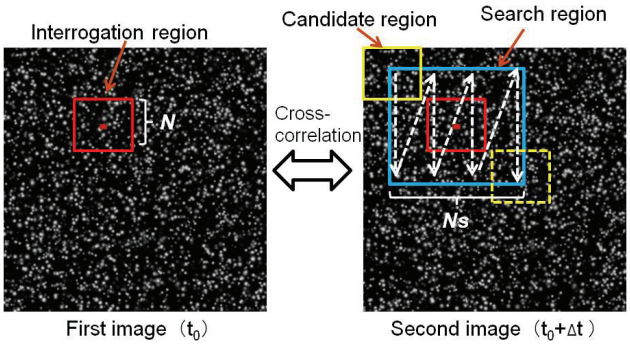

Fig. 2 Overview of the procedure of the DCC method

2. Amount of computations in the DCC method

In Fig. 2, Ns denotes the size of one side of the search region. In order to obtain one velocity vector in the DCC method, the correlation coefficient is calculated $Ns \times Ns$ times. When Ns = 33 and the number

of locations required in order to determine the velocity vectors is 256, the number of calculations per location reaches 1,089, and the calculation is performed a total of 278,784 times during processing. Table 2 shows an example of the increase in the number of calculations. As *Ns* increases, the required calculation amount increases extremely (proportional to *Ns*×*Ns*). In the DCC method, the computational cost required to calculate all of the cross-correlation coefficients between the interrogation region and all of the candidate regions in the search region is enormous. In the case of our code for the DCC PIV, the processing time required to calculate the cross-correlation coefficients accounts for approximately 99% of the total processing time. Since the cost is expected to be substantially reduced and the processing speed is accelerated by parallelizing the calculation process, the associated part of our code based on the DCC algorithm was rewritten to the GPGPU computing code using the CUDA tool kit. A flow chart of the parallelized DCC method using GPGPU is shown in Fig. 4.

IV. PERFORMANCE TESTS

1. Testing conditions

Tables 3 and 4 show, respectively, the testing environments and test cases used to evaluate the performance of the GPU computing for PIV. In Cases 1 through 5, the search region size was varied as indicated in Table 4, and the total number of cases is 15. Cases 1 through 4 were performed to confirm the acceleration rate of GPGPU over conventional CPU computing, and Case 5 was tested in order to evaluate the performance of the new double-precision arithmetic implemented on the GT200 GPU. The CPU computing and GPU computing were performed with double precision and single precision, respectively, except in Case 5.

Table 2 Example of the increase in the number of calculations

Locations	*Ns*	Calculations per location (*Ns*×*Ns*)	Total calculations (*Ns*×*Ns*× *locations*)
256	33	1,089	278,784
256	65	4,225	1,081,600
256	97	9,409	2,408,704

Table 3 Hardware testing environments

Environment	Env. 1	Env. 2
CPU	Phenom 9750	Core i7 920
Memory	4 GB	6 GB
GPU	GeForce GTX275	GeForce GTX275

Table 4 Lists of test cases

Case	Env.	Computing	Precision	Search region size (*Ns*×*Ns*)
1	Env. 1	CPU	double	1,089 (33×33), 4,225 (65×65), 9,409 (97×97)
2	Env. 1	GPU	single	
3	Env. 2	CPU	double	
4	Env. 2	GPU	single	
5	Env. 2	GPU	double	

2. Results of the performance tests

Figure 5 and 6 show the elapsed times for the above test cases. Here, the elapsed time indicates the time just for the process execution after the process was created. In cases of processing by only the CPU (Cases 1 and 3), the elapsed time increases linearly with the search region size, whereas in cases of processing by the GPGPU (Cases 2 and 4), the elapsed time increases only slightly. On the other hand, the difference in the results for the GPGPU computing between the Phenom CPU and Core i7 CPU suggests that the performance of the GPGPU depends not only on the GPU architecture but also on the CPU or system architecture. The results for Case 5 indicate that double-floating-point

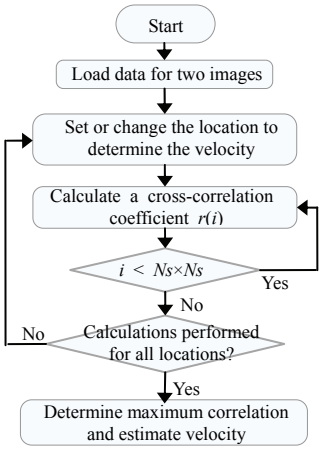

Fig. 3 Flow chart of the DCC method for CPU computing

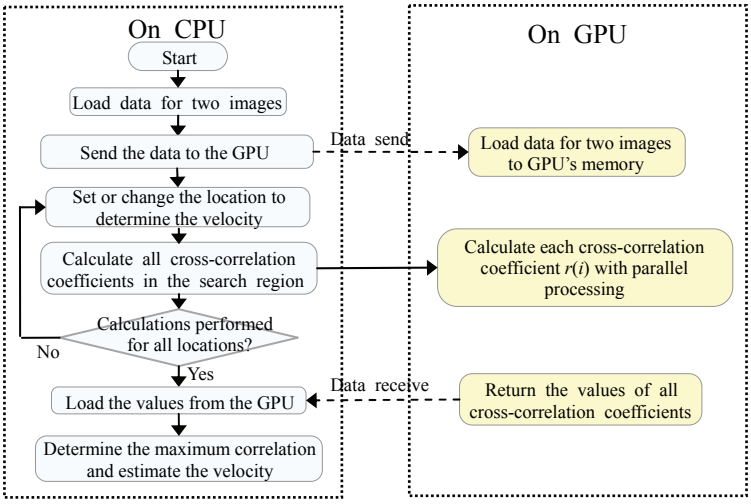

Fig. 4 Flow chart of the parallelized DCC method for GPU computing

calculation requires greater processing time than the single-floating-point calculation, because the number of double-precision arithmetic units of the GPU is not sufficient to execute a large number of parallel threads. Figure 7 shows the fractions of the elapsed time used for CPU and GPU processing. The CPU processing time includes the data translation time between the host memory and the GPU memory. Although there is a difference related to the search region size, approximately 95% of the elapsed time was used for GPU processing. Figure 8 shows the acceleration rates of the GPU computing over CPU-only computing, where rates of over 100 times and about 80 times can be seen for the Phenom CPU and the Core i7 CPU, respectively.

In the present study, the computing results for PIV based on the DCC method for all cases were exactly the same. In the case of the PIV analysis, the computational error caused by the floating point calculation is not a significant problem. This indicates that the precision of the processing on GPU for the PIV is adequate.

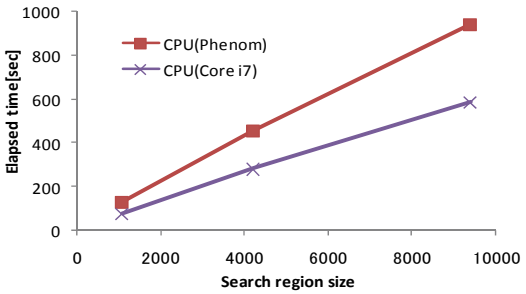

Fig. 5 Elapsed times for Cases 1 and 3

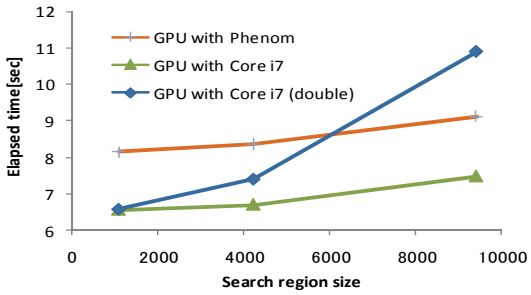

Fig. 6 Elapsed times for Cases 2, 4 and 5

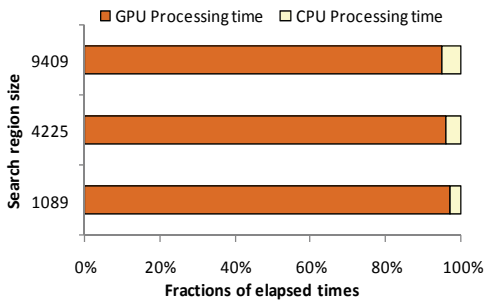

Fig. 7 Fractions of the elapsed time used for CPU and GPU processing for Case 4

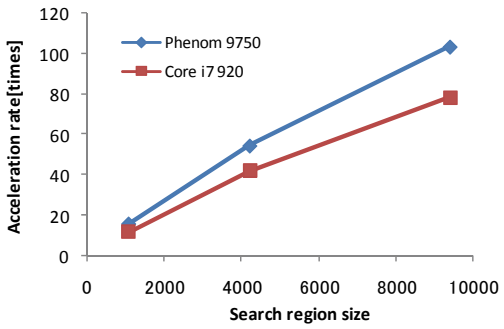

Fig. 8 Relationship between acceleration rate and search region size for GPGPU and CPU computing

V. CONCLUSION

A very high acceleration rate was obtained and the processing ability was improved by the use of a GPGPU. The performance of GPU computing depended not only on the GPU but also on the system architecture. Single-floating-point calculation precision was found to be sufficient for PIV analysis. These results indicate that the application of GPGPU to PIV is highly effective.

REFERENCES

[1] John D. Owens, David Luebke, Naga Govindaraju, et al. (2007), A Survey of General-Purpose Computation on Graphics Hardware, Computer Graphics Forum. Vol. 26, Issue 1, pp. 80-113

[2] John E. Stone, James C. Phillips, Peter L. Freddolino, et al. (2007), Accelerating molecular modeling applications with graphics processors, Journal of Computational Chemistry, Vol.28, pp. 2618-2640

[3] T. Aoki (2009), CFD Applications Fully Accelerated by GPU (in Japanese). Journal of Information Processing Society of Japan, Vol.50, Issue 2, pp.107-115

[4] F. Ino, J. Gomita, Y. Kawasaki and K. Hagihara (2006), A GPGPU approach for accelerating 2-D/3-D rigid registration of medical images. In Parallel and Distributed Processing and Applications (ISPA), LNCS, vol.4330, pp.939–950

[5] T.S. Wung and F.G.Tseng (1992), A color-coded particle tracking velocimetry with applications to natural convection, Vol.13, Experiments in Fluids, pp.217-223

[6] Y.A.Hassan and R.E.Canaan (1991), Full-field bubbly flow velocity measurements using a multi frame particle tracking technique, Vol.12, Experiments in Fluids, pp.49-60

[7] K. Okamoto, S. Nishio, T. Saga and T. Kobayashi (1997), Standard Images for Particle Imaging Velocimetry, Proc. PIV-Fukui'97