Numerical Investigations of Variable Preconditioned GCR with Mixed Precision on GPU

Soichiro IKUNO, Shuntaro DEKI and Norihisa FUJITA

Tokyo University of Technology, Katakura 1404-1, Hachioji, Tokyo 192-0982, Japan

(Received 9 December 2009 / Accepted 8 February 2010)

The Variable Preconditioned GCR (VPGCR) with mixed precision on Graphics Processing Unit (GPU) using CUDA is numerically investigated. The convergence theorem of VPGCR is guaranteed that the residual equation can be solved in the range of single precision, which means that VPGCR is applicable method to elicit the high performance of GPU. The results of computations show that VPGCR with mixed precision demonstrated significant achievement than that with double precision operation. Especially, the hybrid VPGCR on GPU is 10.10 times faster than that of CPU, and the standard VPGCR on GPU is 10.36 times faster than that of CPU.

© *2010 The Japan Society of Plasma Science and Nuclear Fusion Research*

Keywords: VPGCR method, GPGPU, CUDA, mixed precision, linear system

DOI: 10.1585/pfr.5.S2115

1. Introduction

Generally, it is necessary to solve a large simultaneous linear system in the numerical calculation of equilibrium configuration of Spheromak plasma or Tokamak plasma [1]. Moreover, the calculation speed and high accuracy are required for solving the system.

Recently, the performance of Graphics Processing Unit (GPU) surpasses that of CPU and various researches of General Purpose for GPU (GPGPU) have been proposed aggressively [2,3]. However, GPU can calculate in the case of single precision fast, but becomes slow in double precision calculation because GPU is a device used for drawing of the graphics. Although the performance in double precision is still superior to the other commodity processors, high performance single precision operations of GPU should be used effectively. Moreover, there are many difficult points in coding the floating operations using GPU.

K. Abe *et al*. developed new preconditioning strategy which is called the Variable Preconditioned Generalized Conjugate Residual (VPGCR) method [4]. In VPGCR, the residual equation is solved in each iteration instead of preconditioned matrix calculation. The convergence theorem of VPGCR is guaranteed that the residual of VPGCR converges if the relative residual norm of variable precondition calculation r satisfies the criterion $r < 1$. The residual equation can be solved in the range of single precision, which means that VPGCR is applicable method to elicit the high performance of GPU.

In this study, we apply the hybrid scheme that uses single precision and double precision operations to VPGCR method using GPU. In the precondition process, single precision operations can be used for faster calculation since only approximate results are needed. The reduction of execution time in the precondition that occupies a large part of calculation is dramatically effective although the main GCR process must be executed in double precision as same as usual.

2. Compute Unified Device Architecture, CUDA

In recent years, commodity graphics processing units (GPUs) have been obtained large computation power since applications like 3D games needs a realistic visualization. Furthermore, GPU architectures have been changed from fixed operation to flexible organization for programmability; therefore, GPUs are capable of scientific computing more than the specific graphics operation. For example, GeForce GTX 285 can perform up to 1063 GFLOPS by using single precision and 88.6 GFLOPS by using double precision. This performance is about a performance of 53 times of Core2 Duo by using single precision. However, computations on GPU are restricted to single precision floating point arithmetics, and rounding operation is fixed to truncate. Although a double precision floating point value can be emulated with two single precision values, long execution time is needed.

There are two difficult points in operations using GPU. First, the results of operation can be written to the frame buffer memory by reading out data from the texture memory. However, the opposite process cannot be done. Another difficulty is to achieve interdependence with data processing in the back and forth because GPU specializes in the parallel processing in the input data operation.

CUDA is architecture with new hardware and the software to manage the calculation on GPU as a parallel com-

author's e-mail: ikuno@nal.ikulab.org

© 2010 The Japan Society of Plasma Science and Nuclear Fusion Research

puter developed by the NVIDIA corporation [5]. In addition, when CUDA is used, we do not have to move the data to graphics API. The concept related to the graphics like the texture memory and the frame buffer, etc. did not worry when CUDA is used, and it is possible to treat comparatively with the operation in CPU. CUDA is correspondence since the Geforce 8 series.

3. VPGCR with Mixed Precision

As is well known that a preconditioning strategy can improve the performance for solving a linear system $A\boldsymbol{x} = \boldsymbol{b}$ using the Krylov subspace method. Here, A, $\boldsymbol{x}$ and $\boldsymbol{b}$ denote a coefficient matrix, an unknown vector and a known vector, respectively. Generally, a preconditioned matrix M is determined by incomplete LU decomposition, and a vector $M^{-1}\boldsymbol{r}_k$ is calculated at k-th iteration by using backward substitution or incomplete Cholesky factorization. Here, $\boldsymbol{r}_k$ denotes residual vector at k-th iteration. The calculation time of solving linear system is relatively large for the preconditioned part.

K. Abe *et al.* developed new preconditioning strategy which is called variable preconditioning method [4]. The algorithm of Variable Preconditioned Generalized Conjugate Residual method (VPGCR) is shown in Fig. 1. VPGCR has two nested iterations for Generalized Conjugate Residual method (GCR) and variable preconditioning for GCR are called as outer-loop and inner-loop, respectively. The inner-loop is shown as surrounded by the box (see Fig. 1).

In VPGCR, the following residual equations are solved in each outer-loop and inner-loop instead of preconditioned matrix calculation.

$$A\boldsymbol{p}_0 = \boldsymbol{r}_0, \tag{1}$$

$$A\boldsymbol{z}_{k+1} = \boldsymbol{r}_{k+1}. \tag{2}$$

Here, subscript denotes a iteration number. Naturally VPGCR takes much CPU time than that of general GCR method if the residual equation is solved in high accuracy. However, VPGCR has the advantageous character. The convergence theorem of VPGCR is guaranteed that the residual of VPGCR converges if the relative residual norm of inner-loop satisfies the following condition.

$$r_{\text{in}} = \frac{\|\boldsymbol{r}_{k+1} - A\boldsymbol{z}_{k+1}\|_2}{\|\boldsymbol{r}_{k+1}\|_2} < 1. \tag{3}$$

Here, $\|\boldsymbol{x}\|_2$ denotes 2-norm of vector $\boldsymbol{x}$. That is to say; the residual equation can be solved roughly by using iterative method with only a few iteration. Generally, a stationary iterative method such as Gauss-Seidel method, is adopted for variable preconditioning procedure.

To elicit the high performance of GPU, it is necessary to use single precision calculations in a meaningful way because GPU is developed as a device used for drawing of the graphics. Additionally, the convergence theorem of VPGCR is guaranteed that the residual equation can be solved in the range of single precision, which means that VPGCR is applicable method to elicit the high performance of GPU. Therefore, VPGCR with mixed precision that uses single precision operation for inner-loop and double precision operation for outer-loop is adopted for the solver on GPU and the method is called the hybrid precision VPGCR method.

begin procedure VPGCR
  $\boldsymbol{x}_0 \leftarrow$ Set a initial value.
  begin loop
    $\boldsymbol{r}_0 \leftarrow \boldsymbol{b} - A\boldsymbol{x}_0$
    $\boldsymbol{p}_0 \leftarrow$ roughly solve $A\boldsymbol{p}_0 = \boldsymbol{r}_0$
    $\boldsymbol{q}_0 \leftarrow A\boldsymbol{p}_0$
    begin for $k \leftarrow 1, 2, \cdots, \kappa_{\text{GCR}}$
      $\alpha_k \leftarrow \dfrac{(\boldsymbol{r}_k, \boldsymbol{q}_k)}{(\boldsymbol{q}_k, \boldsymbol{q}_k)}$
      $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
      $\boldsymbol{r}_{k+1} \leftarrow \boldsymbol{r}_k - \alpha_k \boldsymbol{q}_k$
      begin if $\|\boldsymbol{r}_k\|_2 \le \varepsilon_{\text{out}} \|\boldsymbol{r}_0\|_2$
        return $\boldsymbol{x}_k$
      end if
      $z_{k+1} \leftarrow$ roughly solve $A\boldsymbol{z}_{k+1} = \boldsymbol{r}_{k+1}$
      begin for $i \leftarrow 0, 1, \cdots, k$
        $\beta_{k,i} \leftarrow -\dfrac{(A\boldsymbol{z}_{k+1}, \boldsymbol{q}_i)}{(\boldsymbol{q}_i, \boldsymbol{q}_i)}$
      end for
      $\boldsymbol{p}_{k+1} \leftarrow \boldsymbol{z}_{k+1} + \sum_{i=0}^{k} \beta_{k,i} \boldsymbol{p}_i$
      $\boldsymbol{q}_{k+1} \leftarrow A\boldsymbol{z}_{k+1} + \sum_{i=0}^{k} \beta_{k,i} \boldsymbol{q}_i$
    end for
    $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}_k$
  end loop
end procedure

Fig. 1 The algorithm of variable preconditioned generalized conjugate residual (VPGCR) method. Here, ε_{out}, $\|\boldsymbol{x}\|_2$ and κ_{GCR} denote termination condition for iterations, 2-norm of vector $\boldsymbol{x}$ and restart parameter of GCR, respectively.

4. Evaluation

VPGCR with mixed precision on GPU is evaluated by means of the linear system $A\boldsymbol{x} = \boldsymbol{b}$. In the present study, the following Toeplitz matrix is adopted for the coefficient matrix.

$$A = \begin{bmatrix} 2 & 1 & & & 0 \\ 0 & 2 & 1 & & \\ \gamma & 0 & 2 & 1 & \\ & \gamma & 0 & 2 & 1 \\ 0 & & \gamma & 0 & 2 \end{bmatrix}. \tag{4}$$

Here, γ denotes a parameter. It is known that the values of condition number increase as the values of γ increase. In other words, the linear system becomes ill-posed problem as the values of γ increase. Thus, we can control the ill-posedness of the problem by controlling the parameter γ. The known vector $\boldsymbol{b}$ is generated so that all the solutions $\boldsymbol{x}$

Table 1 Evaluation environment.

OS	Ubuntu Linux 9.04 x86_64
CPU	Intel Core2Quad Q9550
CPU Memory	8GB
CPU Compiler	gcc 4.3.3
CPU Compiler Option	-O3
GPU	GeForce GTX 285
GPU Memory	1GB
GPU Compiler	nvcc 2.3
GPU Compiler Option	-O -arch=sm_13

might become one (i.e., $\boldsymbol{x} = [1, 1, \cdots, 1]^{\mathrm{T}}$).

As we mentioned above, a stationary iterative method is adopted for the solver for equations (1) and (2). Moreover, the coefficient matrix A must be diagonally dominant matrix. However, the exact solution is not required in the inner-loop of VPGCR. The condition of the requirement is only $r_{\mathrm{in}} < 1$. In the present study, the Jacobi method is employed for solver. This is because, the Jacobi method is easy to parallelized, and the parallel performance is better than that of other stationary iterative method. In addition, iteration of inner-loop is terminated if the condition $r_{\mathrm{in}} < \varepsilon_{\mathrm{in}}$ is satisfied, where $\varepsilon_{\mathrm{in}}$ denotes a constant parameter.

Although a number of threads in an one block is fixed as 128, number of block changes its value as the following equation.

$$M_{\mathrm{block}} = \lceil \frac{N}{M_{\mathrm{thread}}} \rceil. \tag{5}$$

Here, N denotes a dimension size of the linear system and M_{thread} denotes a number of threads. Furthermore, $\lceil X \rceil$ denotes a ceiling function.

At the procedure of inner product calculation in the GPU code, vectors are divided into some blocks and the blocks are scattered to each thread in order to parallel processing. After finishing the calculation in all threads, we have to gather the results from each thread and to add each other to get the solution of inner product. It is well known that it takes large access time from GPU to main memory so that it is a standard tactic to avoid the access to the main memory as much as possible. However, it is faster to execute the procedures of gathering the results from each thread and adding each results by CPU than that of GPU. From the above reasons, a part of convolution operation is operated not only by GPU but also by CPU in the inner product calculation.

Throughout this paper, parameters are fixed as follows: $N = 2048$, $M_{\mathrm{thread}} = 128$, $\varepsilon_{\mathrm{out}} = 1.0^{-12}$. Besides, the evaluation environment is shown in Table 1.

Let us first investigate the execution time of the hybrid VPGCR and the standard VPGCR on GPU and CPU. The values of the execution time are plotted as functions of $\varepsilon_{\mathrm{in}}$ for various method in Fig. 2. The parameter of the Toeplitz matrix is fixed as $\gamma = 1.0$ in this case. This figure indicates

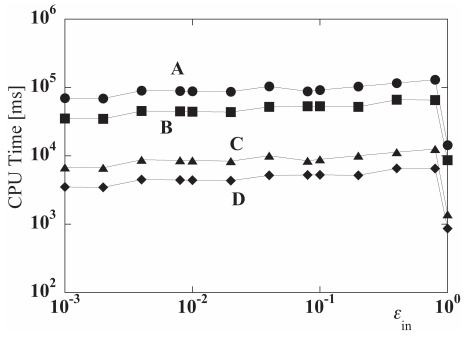

Fig. 2 The execution time of the hybrid VPGCR and the standard VPGCR on GPU and CPU as functions of $\varepsilon_{\mathrm{in}}$ for the case with $\gamma = 1.0$. A: Standard VPGCR on CPU, B: Hybrid VPGCR on CPU, C: Standard VPGCR on GPU, D: Hybrid VPGCR on GPU.

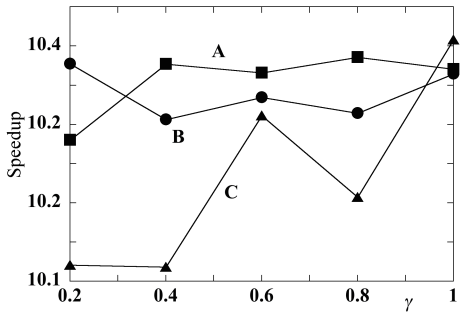

Fig. 3 The speedup ratio of the standard VPGCR calculated on GPU is plotted as a function of γ. The origin of the speedup is the standard VPGCR calculated on CPU. A: $\varepsilon_{\mathrm{in}} = 10^{-2}$, B: $\varepsilon_{\mathrm{in}} = 10^{-3}$, C: $\varepsilon_{\mathrm{in}} = 10^{-1}$.

that the hybrid VPGCR on GPU is faster than that of the standard VPGCR or the standard VPGCR on CPU. Especially, the hybrid VPGCR on GPU is 10.10 times faster than that of CPU, and the standard VPGCR on GPU is 10.36 times faster than that of CPU at the case of $\varepsilon_{\mathrm{in}} = 1.0$. Moreover, the hybrid VPGCR on GPU is 16.59 times as fast as the standard VPGCR on CPU.

Next, the speedup ratio of the standard VPGCR on GPU is shown in Fig. 3. This figure indicates that the standard VPGCR on GPU is about 10 times faster than that of CPU in both cases. This results lead to the conclusion that standard VPGCR on GPU is effective for linear system solver.

The speedup ratio of the hybrid VPGCR on GPU is also shown in Fig. 4. We can see from this figure that the values of speedup ratio increase as the value of γ is increase in both cases. This results indicate that the problem becomes ill-posed as the value of γ increase, and as a result the number of inner-loop iteration has increased (see Table 2). In the hybrid VPGCR, inner-loop procedures are calculated by single precision. Therefore, increasing single precision operation bringing out the performance of GPU. Table 2 also indicates that the tighten up the termination

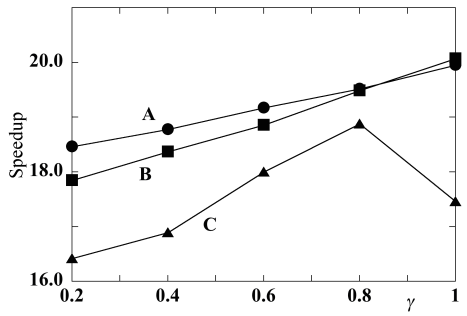

Fig. 4 The speedup ratio of the hybrid VPGCR calculated on GPU is plotted as a function of γ. The origin of the speedup is the standard VPGCR calculated on CPU. A: $\varepsilon_{\rm in} = 10^{-3}$, B: $\varepsilon_{\rm in} = 10^{-2}$, C: $\varepsilon_{\rm in} = 10^{-1}$.

Table 2 Number of iteration for inner and outer (in parentheses) loop which is calculated on GPU.

γ	S(10^{-3})	H(10^{-3})	S(10^{-1})	H(10^{-1})
0.2	57 (3)	57 (3)	36 (6)	36 (6)
0.4	79 (3)	79 (3)	48 (6)	48 (6)
0.6	127 (3)	127 (3)	78 (6)	78 (6)
0.8	279 (3)	279 (3)	204 (7)	204 (7)
1.0	13688 (2)	13703 (2)	17933 (6)	20623 (6)

NOTE: The Character S and H in first row denote standard VPGCR and hybrid VPGCR and the values in parentheses in first row shows $\varepsilon_{\rm in}$.

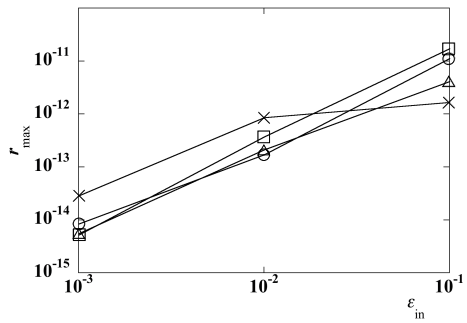

Fig. 5 The values of $r_{\rm max}$ are plotted as a function of $\varepsilon_{\rm in}$. ○: $\gamma = 0.2$, □: $\gamma = 0.4$, △: $\gamma = 0.6$, ×: $\gamma = 0.8$.

condition leads to increasing inner-loop iteration. As the results, the values of speed up increase in both cases (see Fig. 3 and Fig. 4).

Finally, the error of VPGCR is investigated. The values of $r_{\rm max}$ are plotted as a function of $\varepsilon_{\rm in}$ and show in Fig. 5. Here, the values of $r_{\rm max}$ is defined as follows

$$r_{\rm max} = \frac{||\boldsymbol{x}^{\rm D} - \boldsymbol{x}^{\rm H}||_\infty}{||\boldsymbol{x}^{\rm D}||_\infty}, \tag{6}$$

where, $\boldsymbol{x}^{\rm D}$ denotes a solution of linear system solved by the standard VPGCR on CPU and $\boldsymbol{x}^{\rm H}$ denotes a solution of linear system solved by the hybrid VPGCR on GPU. We can see from this figure that the values of $r_{\rm max}$ increase as the value of $\varepsilon_{\rm in}$ increase. This result indicates that preconditioned matrix does not derive correctly because the termination condition of inner-loop ease up.

5. Conclusion

We implemented the Variable Preconditioned-GCR method to GPU with the hybrid scheme that applies single precision operation within the precondition and uses double precision in remaining part.

To elicit the high performance of GPU, VPGCR with mixed precision that uses single precision operation for inner-loop and double precision operation for outer-loop is adopted for the solver on GPU.

Conclusions obtained in the present study are summarized as follows.

- The Jacobi method is employed for inner-loop solver because the method is easy to parallelized, and the parallel performance is better than that of other stationary iterative method.
- A part of convolution operation is operated not only by GPU but also by CPU in the inner product calculation because it is faster to execute the procedures of gathering the results from each thread and adding each results by CPU than that of GPU.
- The hybrid VPGCR on GPU is faster than that of the standard VPGCR or the standard VPGCR on CPU. Especially, the hybrid VPGCR on GPU is 10.10 times faster than that of CPU, and the standard VPGCR on GPU is 10.36 times faster than that of CPU at the case of $\varepsilon_{\rm in} = 1.0$.
- The problem becomes ill-posed as the value of γ increase, and as a result the number of inner-loop iteration has increased. Therefore, increasing single precision operation bringing out the performance of GPU.
- The tighten up the termination condition leads to increasing inner-loop iteration. As the results, the values of speed up increase in both cases.

[1] S. Ikuno, M. Natori and A. Kamitani, Fusion Energy 1998, **4**, 1497 (IAEA, Vienna, 1999).
[2] http://gpgpu.org/
[3] A. Cevahir, A. Nukada and S. Matsuoka, Lecture Notes in Computer Science, **5544**, 893 (Springer Berlin, 2009).
[4] K. Abe and S. L. Zhang, Int. J. Numer. Anal. Model. **22**, 147 (2005).
[5] http://www.nvidia.com/object/cuda_develop.html