A GPU Implementation of GPBiCGSafe-Algorithm-Based Navier-Stokes Solvers for 3D Incompressible Flows

A GPU Implementation of 3D Navier-Stokes Solvers

Huynh Quang Huy Viet & Hiroshi Suito

Graduate School of Environmental and Life Science, Okayama University

Contact Information:
Graduate School of Environmental and Life Science
Okayama University
3-1-1 Tsushima-naka, Okayama, 700-8530, Japan
Email 1: hqhviet@ems.okayama-u.ac.jp
Email 2: suito@okayama-u.ac.jp

Introduction

A system of linear equations derived from the discretization of the Navier-Stokes equation with mixed Dirichlet and Neumann boundary conditions for pressure, or with non-uniform grid spacing, is a large and sparse non-symmetric system of linear equations. The numerical solution of non-symmetric linear systems has often been solved by using iterative methods such as the Bi-conjugate gradient stabilized method (Bi-CGStab) or the Generalized minimal residual method (GMRES). The Bi-CGStab is a good algorithm especially when the available memory is relatively small in relation to the size of system memory. Among the variations of the Bi-CGStab algorithm [2] proposed by various researchers, the GPBiCGSafe algorithm proposed by Fujino et al., has been proven to have very good convergence behavior. In this poster, we describe our CUDA parallel implementation of a solver of 3D Navier-Stokes equations using the GPBiCGSafe algorithm.

Numerical Solution

The Navier-Stokes Equations

1. The momentum equation

$$\frac{\partial \vec{u}}{\partial t} + (\vec{u}\cdot\nabla)\vec{u} = -\frac{1}{\rho}\nabla p + \nu\nabla^2\vec{u}, \tag{1}$$

2. The continuity equation

$$\nabla\cdot\vec{u} = 0, \tag{2}$$

where $\vec{u}$ is the velocity vector field, p is the scalar pressure field, ρ is the density and ν is the kinematic viscosity coefficent of the fluid. In this study, the density and kinematic viscosity coefficent are constants. The Simplified MAC (SMAC) method [1] is used to find numerical solutions of the governing equations. At each time step n, the next velocity field $\vec{u}^{n+1}$ is computed as follows:

1. Computing an intermediate velocity vector field $\vec{u}^*$

$$\vec{u}^* = \vec{u}^n - \delta t[(\vec{u}^n\cdot\nabla)\vec{u}^n + \frac{1}{\rho}\nabla p^n - \nu\nabla^2\vec{u}^n], \tag{3}$$

2. Solving the Poisson equation for the pressure correction term Φ^{n+1} and computing the scalar pressure field p^{n+1}

$$\nabla^2\Phi^{n+1} = \frac{\rho}{\delta t}\nabla\cdot\vec{u}^*, \tag{4}$$

$$p^{n+1} = p^n + \Phi^{n+1}, \tag{5}$$

3. Updating the velocity vector field $\vec{u}^{n+1}$ by using the computed scalar pressure field so as it satisfies the continuity equation

$$\vec{u}^{n+1} = \vec{u}^* - \frac{\delta t}{\rho}\nabla\Phi^{n+1}. \tag{6}$$

Discretization of the above Poisson equation (4) leads to a very large, sparse and banded linear equation system. This linear equation system is solved by using the GPBiCGSafe algorithm [2].

The GPBiCGSafe Algorithm

Algorithm 1 The GPBiCGSafe Algorithm

$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0$;
Choose $\boldsymbol{r}_0^*$ such that $(\boldsymbol{r}_0^*, \boldsymbol{r}_0) \neq 0, \beta_{-1} = 0$;
$\boldsymbol{p}_{-1} = \boldsymbol{u}_{-1} = \boldsymbol{z}_{-1} = 0$;
for $n = 0, 1, \ldots$ until $||\boldsymbol{r}_n||/||\boldsymbol{r}_0|| \leq \epsilon$ **do**
  $\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}(\boldsymbol{p}_{n-1} - \boldsymbol{u}_{n-1})$;
  $\boldsymbol{A}\boldsymbol{p}_n = \boldsymbol{A}\boldsymbol{r}_n + \beta_{n-1}(\boldsymbol{A}\boldsymbol{p}_{n-1} - \boldsymbol{A}\boldsymbol{u}_{n-1})$;
  $\alpha_n = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}{(\boldsymbol{r}_0^*, \boldsymbol{A}\boldsymbol{p}_n)}$;
  $\boldsymbol{a}_n = \boldsymbol{r}_n, \boldsymbol{b}_n = \boldsymbol{A}\boldsymbol{z}_{n-1}, \boldsymbol{c}_n = \boldsymbol{A}\boldsymbol{r}_n$;
  $\zeta_n = \frac{(\boldsymbol{b}_n,\boldsymbol{b}_n)(\boldsymbol{c}_n,\boldsymbol{a}_n) - (\boldsymbol{b}_n,\boldsymbol{a}_n)(\boldsymbol{c}_n,\boldsymbol{b}_n)}{(\boldsymbol{c}_n,\boldsymbol{c}_n)(\boldsymbol{b}_n,\boldsymbol{b}_n) - (\boldsymbol{b}_n,\boldsymbol{c}_n)(\boldsymbol{c}_n,\boldsymbol{b}_n)}$;
  $\eta_n = \frac{(\boldsymbol{c}_n,\boldsymbol{c}_n)(\boldsymbol{b}_n,\boldsymbol{a}_n) - (\boldsymbol{b}_n,\boldsymbol{c}_n)(\boldsymbol{c}_n,\boldsymbol{a}_n)}{(\boldsymbol{c}_n,\boldsymbol{c}_n)(\boldsymbol{b}_n,\boldsymbol{b}_n) - (\boldsymbol{b}_n,\boldsymbol{c}_n)(\boldsymbol{c}_n,\boldsymbol{b}_n)}$;
  (if $n = 0$, then $\zeta_n = \frac{(\boldsymbol{c}_n,\boldsymbol{a}_n)}{(\boldsymbol{c}_n,\boldsymbol{c}_n)}$, $\eta_n = 0$) ;
  $\boldsymbol{u}_n = \zeta_n\boldsymbol{A}\boldsymbol{p}_n + \eta_n(\boldsymbol{A}\boldsymbol{z}_{n-1} + \beta_{n-1}\boldsymbol{u}_{n-1})$;
  $\boldsymbol{t}_n = \boldsymbol{r}_n - \alpha_n\boldsymbol{A}\boldsymbol{p}_n$;
  $\boldsymbol{z}_n = \zeta_n\boldsymbol{r}_n + \eta_n\boldsymbol{z}_{n-1} - \alpha_n\boldsymbol{u}_n$;
  $\boldsymbol{A}\boldsymbol{z}_n = \zeta_n\boldsymbol{A}\boldsymbol{r}_n + \eta_n\boldsymbol{A}\boldsymbol{z}_{n-1} - \alpha_n\boldsymbol{A}\boldsymbol{u}_n$;
  $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha\boldsymbol{p}_n + \boldsymbol{z}_n$;
  $\boldsymbol{r}_{n+1} = \boldsymbol{t}_n - \boldsymbol{A}\boldsymbol{z}_n$;
  $\beta_n = \frac{\alpha_n}{\zeta_n}\frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}$;
end for

The GPU Implementation

The GPBiCGSafe algorithm is implemented by using the following basic operations:

- MV - matrix vector product
- DOT - inner product
- AXPBY - add a multiple of one vector to another
- COPY - copy one vector to another
- SCAL - scale a vector by a constant

These basic operations are implemented by using the Thrust parallel algorithms library [4]. Figure 2 shows the representation of the MV operation of a sparse banded matrix A with a vector b. Each band of an $N \times N$ dimensional matrix A can be seen as an one-dimensional array of length N which is stored in the global memory of a GPU. The vectors b and the result vector Ab of dimension N are also stored in the global memory. Zeroes are appended or prepended to the bands depending on the position of the band in the banded matrix A. Figure 2 (bottom) shows a process of multiplication of a tri-banded matrix A with a vector b. Each band of the matrix A is appended or prepended with zeroes corresponding the position of the band and then multiplied with the vector b which is shifted to left or right similarly. The products are summed to obtain the result vector Ab.

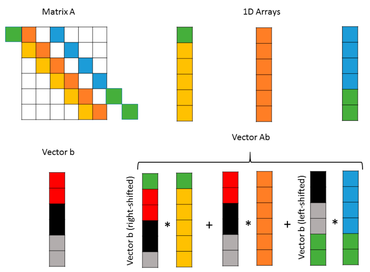

Figure 1: The GPU representation of the banded matrix vector multiplication

Results

Calculation Environment

- Intel(R) Xeon(R) CPU E5506 2.13GHz, 10 GB system memory
- NVIDIA Tesla C2050, CUDA 6.5
- NVIDIA's CUDA Compiler (NVCC), GCC 4.4.7
- Compiler options: -O2 -arch=sm_20
- Time measured by the command cudaEventElapsedTime
- All computations carried out in double precision

Speed-up Ratios

Grid Size	GPU Time	CPU Time	Speed-up
50x60x70	1260 ms	9355 ms	7.4
100x110x120	8079 ms	115283 ms	14.3
150x160x170	33032 ms	609519 ms	18.5

Table 1: Speed-ups (Poisson Equation)

We measured the speed-up ratios for a 3D Poisson equation with Dirichlet boundary conditions on the unit cube $[0,1] \times [0,1] \times [0,1]$: $-\nabla^2 u(x,y,z) = f(x,y,z)$, where $f(x,y,z) = (x^2y^2 + y^2z^2 + z^2x^2)sin(xyz)$. The discretization on a non-uniform grid generates a large and sparse non-symmetric system of linear equations. The diagonal pre-conditioner was applied to the GPBiCGSafe algorithm. The iteration was terminated when $\|r_n\|/\|r_0\| \leq \epsilon = 10^{-12}$.

Grid Size	GPU Time	CPU Time	Speed-up
50x60x70	98s	344 s	3.5
100x110x120	562 s	5478 s	9.7
150x160x170	2014 s	25462 s	12.6

Table 2: Speed-ups (Navier-Stokes Equations)

Table (2) shows the computation time during first 10 time steps on GPU and CPU of the GPBiCGSafe based solver for Navier-Stokes equations with mixed Dirichlet and Neumann boundary conditions.

Conclusions

We have presented the efficient GPU implementation of a 3D Navier-Stokes solver using the GP- BiCGSafe algorithm. We have also introduced a parallel way of multiplication of a sparse banded matrix and a vector which can be easily carried out by using the Thrust parallel algorithms library. In next works, we will improve the current implementation by employing multigrid pre-conditioners. Our main goal is to develop a fast parallel Navier-Stokes solver based on GPU for simulation of blood flows.

References

1. F.H. Harlow, A.A. Amsden, A simplified MAC technique for incompressible fluid flow calculations, J. Comput. Phys. Vol.6 , pp.322-325 (1970).
2. Seiji Fujino, Takashi Sekimoto, "Performance Evaluation of GPBiCGSafe Method without Reverse-Ordered Recurrence for Realistic Problems", Lecture Notes in Engineering and Computer Science, pp.1673-1677 (2012).
3. Kurger, J., Westermann, R., "Linear algebra operators for gpu implementation of numerical algorithms", ACM Trans. Graph., Vol.22, No.3, pp.908-916 (2003).
4. http://docs.nvidia.com/cuda/thrust/

Acknowledgements

This work is supported by JST-CREST (Japan Science and Technology Agency, Core Research for Evolutional Science and Technology).