A Multi-GPU Implementation of a Parallel Solver for Incompressible Navier-Stokes Equations Discretized by Stabilized Finite Element Formulations

HUYNH Quang Huy Viet, SUITO Hiroshi
Graduate School of Environmental and Life Science, Okayama University
E-mail: hqhviet@okayama-u.ac.jp, suito@okayama-u.ac.jp

1 Introduction

For solving partial differential equations problems in Computational Fluid Dynamics (CFD), finite element methods are conventional numerical methods that are particularly used because of their accuracy. In solving the Navier–Stokes (NS) equations using finite element methods for simulation of incompressible flows, some instabilities arise from the presence of advection terms or the high Reynolds number. Hughes and Tezduyar et al. [1, 2, 3, 4] proposed stabilized finite element formulations for incompressible flows. Stabilization in solving Navier–Stokes equations is achieved by adding two stabilization terms to the Galerkin formulations of the Navier–Stokes equations. The first stabilization term is the streamline upwind/Petrov-Galerkin (SUPG) term. The second stabilization term is the pressure stabilizing/Petrov-Galerkin (PSPG) term. This stabilized finite element method has been shown to be very effective for the simulation of incompressible flows.

In engineering applications, when computational conditions become stiff, NS solvers might not converge to solutions within an allowed time limitation. Therefore it is necessary to develop fast and accurate NS solvers using parallel processing techniques. Traditionally, parallel NS solvers are developed using supercomputers or PC clusters with parallel programming platforms such as OpenMP and MPI. Recently, because modern graphics processing units (GPUs) have many processors or cores, GPU computing has been recognized as a powerful platform to achieve high performance in simulation and modeling in the CFD. GPU computing is the use of GPUs in association with the use of CPUs to speed up computations. A desktop machine or a workstation with a powerful GPU inside can achieve extremely high levels of performance for computation and simulation. The necessity exists to develop parallel NS solvers on GPUs for various engineering applications.

In a recent report [5], we briefly described our implementation of a solver based on the GPBi-CG algorithm for 3D unsteady Navier-Stokes equations discretized by the SUPG/PSPG stabilized finite element formulation using a single GPU. In this paper, we report a new multi-GPU implementation of the solver and shows performance results.

2 Stabilized Finite Element Formulations

2.1 Governing Equations

We consider the following dimensionless form of the Navier–Stokes equations in a spatial domain $\Omega \subset \mathbf{R}^3$:

$$\frac{\partial u_i}{\partial t} + u_j\frac{\partial u_i}{\partial x_j} = -\frac{\partial p}{\partial x_i} + \frac{1}{Re}\frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \quad in \quad \Omega, \tag{1}$$

$$\frac{\partial u_i}{\partial x_i} = 0 \quad in \quad \Omega. \tag{2}$$

Here, we adopt the summation convention on repeated indices that have values 1, 2, and 3. The x, y, z axes in the Cartesian coordinate system are designated as $x_i, i = 1, 2, 3$. Here, u_i represents the component of the velocity vector field $\mathbf{u}$ in the i^{th} dimension, p stands for the scalar pressure field, and Re denotes the Reynolds number.

Let us discretize the spatial domain Ω by elements Ω^e, $e = 1, 2..., n_{el}$. Let $\mathcal{S}_{\mathbf{u}}, \mathcal{V}_{\mathbf{u}}$ be the trial and test function spaces for velocity and $\mathcal{S}_p, \mathcal{V}_p$ ($\mathcal{V}_p = \mathcal{S}_p$) be trial and test function spaces for pressure. The stabilized finite element formulation of the equations (1)-(2) with the SUPG/PSPG stabilization terms can be expressed as follows [1]: Find $\mathbf{u} \in \mathcal{S}_{\mathbf{u}}$ and $p \in \mathcal{S}_p$ such that $\forall \mathbf{w} \in \mathcal{V}_{\mathbf{u}}$ and $\forall q \in \mathcal{V}_p$:

$$\int_{\Omega} w_i \left(\frac{\partial u_i}{\partial t} + \bar{u}_j \frac{\partial u_i}{\partial x_j} \right) d\Omega - \int_{\Omega} \frac{\partial w_i}{\partial x_i} p d\Omega + \int_{\Omega} \frac{1}{Re} \frac{\partial w_i}{\partial x_j} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) d\Omega$$

$$+ \sum_{e=1}^{n_{el}} \int_{\Omega_e} \tau \bar{u}_k \frac{\partial w_i}{\partial x_k} \left(\frac{\partial u_i}{\partial t} + \bar{u}_j \frac{\partial u_i}{\partial x_j} + \frac{\partial p}{\partial x_i} \right) d\Omega = 0, \tag{3}$$

$$\int_{\Omega} q \frac{\partial u_i}{\partial x_i} d\Omega + \sum_{e=1}^{n_{el}} \int_{\Omega_e} \tau \frac{\partial q}{\partial x_i} \left(\frac{\partial u_i}{\partial t} + \bar{u}_j \frac{\partial u_i}{\partial x_j} + \frac{\partial p}{\partial x_i} \right) d\Omega = 0, \tag{4}$$

where τ is the SUPG/PSPG stabilization parameter. The formulation to calculate the parameter τ is given in details in the paper [1].

3 GPBi-CG Algorithm

The discretization of the Navier-Stokes equation by stabilized finite element method leads to a large and sparse non-symmetric system of linear equations. This linear equation system is solved by using the GPBi-CG algorithm [6].

Algorithm 1 The Unpreconditioned GPBi-CG

```
$\boldsymbol{x}_0$ is an initial guess, $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_o$;
Set $\boldsymbol{r}_0^* = \boldsymbol{r}_0, \boldsymbol{t}_{-1} = \boldsymbol{w}_{-1} = 0, \beta_{-1} = 0$;
for $n = 0, 1, ...$ until $||\boldsymbol{r}_n|| \leq \epsilon ||\boldsymbol{b}||$ do
  $\boldsymbol{p}_n = \boldsymbol{r}_n + \beta_{n-1}(\boldsymbol{p}_{n-1} - \boldsymbol{u}_{n-1})$,
  $\alpha_n = \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}{(\boldsymbol{r}_0^*, \boldsymbol{A}\boldsymbol{p}_n)}$,
  $\boldsymbol{y}_n = \boldsymbol{t}_{n-1} - \boldsymbol{r}_n - \alpha_n \boldsymbol{w}_{n-1} + \alpha_n \boldsymbol{A}\boldsymbol{p}_n$,
  $\boldsymbol{t}_n = \boldsymbol{r}_n - \alpha_n \boldsymbol{A}\boldsymbol{p}_n$,
  $\zeta_n = \frac{(\boldsymbol{y}_n, \boldsymbol{y}_n)(\boldsymbol{A}\boldsymbol{t}_n, \boldsymbol{t}_n) - (\boldsymbol{y}_n, \boldsymbol{t}_n)(\boldsymbol{A}\boldsymbol{t}_n, \boldsymbol{y}_n)}{(\boldsymbol{A}\boldsymbol{t}_n, \boldsymbol{A}\boldsymbol{t}_n)(\boldsymbol{y}_n, \boldsymbol{y}_n) - (\boldsymbol{y}_n, \boldsymbol{A}\boldsymbol{t}_n)(\boldsymbol{A}\boldsymbol{t}_n, \boldsymbol{y}_n)}$,
  $\eta_n = \frac{(\boldsymbol{A}\boldsymbol{t}_n, \boldsymbol{A}\boldsymbol{t}_n)(\boldsymbol{y}_n, \boldsymbol{t}_n) - (\boldsymbol{y}_n, \boldsymbol{A}\boldsymbol{t}_n)(\boldsymbol{A}\boldsymbol{t}_n, \boldsymbol{t}_n)}{(\boldsymbol{A}\boldsymbol{t}_n, \boldsymbol{A}\boldsymbol{t}_n)(\boldsymbol{y}_n, \boldsymbol{y}_n) - (\boldsymbol{y}_n, \boldsymbol{A}\boldsymbol{t}_n)(\boldsymbol{A}\boldsymbol{t}_n, \boldsymbol{y}_n)}$,
  (if $n = 0$, then $\zeta_n = \frac{(\boldsymbol{A}\boldsymbol{t}_n, \boldsymbol{t}_n)}{(\boldsymbol{A}\boldsymbol{t}_n, \boldsymbol{A}\boldsymbol{t}_n)}, \eta_n = 0$),
  $\boldsymbol{u}_n = \zeta_n \boldsymbol{A}\boldsymbol{p}_n + \eta_n(\boldsymbol{t}_{n-1} - \boldsymbol{r}_n + \beta_{n-1}\boldsymbol{u}_{n-1})$,
  $\boldsymbol{z}_n = \zeta_n \boldsymbol{r}_n + \eta_n \boldsymbol{z}_{n-1} - \alpha_n \boldsymbol{u}_n$,
  $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \alpha \boldsymbol{p}_n + \boldsymbol{z}_n$,
  $\boldsymbol{r}_{n+1} = \boldsymbol{t}_n - \eta_n \boldsymbol{y}_n - \zeta_n \boldsymbol{A}\boldsymbol{t}_n$,
  $\beta_n = \frac{\alpha_n}{\zeta_n} \frac{(\boldsymbol{r}_0^*, \boldsymbol{r}_{n+1})}{(\boldsymbol{r}_0^*, \boldsymbol{r}_n)}$,
  $\boldsymbol{w}_n = \boldsymbol{A}\boldsymbol{t}_n + \beta_n \boldsymbol{A}\boldsymbol{p}_n$;
end for
```

To implement the GPBi-CG algorithm on the GPU platform, we used the Nvidia's GPU linear algebra libraries cuSPARSE [7] and cuBLAS [8] to implement four basic vector operations of the algorithm: SpMV, DOT, AXPY and SCAL as described below.

- SpMV: the sparse matrix-vector product,
- DOT: the inner product,
- AXPY: add a multiple of one vector to another,
- SCAL: scaling a vector by a constant.

The SpMV operation is implemented by using the cuSPARSE library. The DOT, AXPY and SCAL operations are implemented by using the cuBLAS library.

4 Multi-GPU Implementation

To extend to multi-GPU implementation of the GPBi-CG algorithm, we implemented the multi-GPU version of SpMV, DOT, AXPY, and SCAL operations by using the MPI library, the cuSPARSE library and the cuBLAS library. It is trivial to implement the multi-GPU version of the DOT, AXPY, and SCAL operations by using the MPI library and cuBLAS library. However, the implementation of the multi-GPU version of the SpMV operation is not straightforward. It is carried out by subdividing the computational domain into subdomains that are distributed over the GPUs by using the MPI library. To reduce communication cost, we develop an efficient procedure that each GPU receives from other GPUs only data each GPU needs. We consider the problem of matrix-vector multiplication of a sparse matrix $\boldsymbol{A}$ and a dense vector $\boldsymbol{b}$. We partition the matrix $\boldsymbol{A}$ into groups of adjacent complete rows and assign one such group to one GPU. Each GPU is now obligated to compute entries of the result vector $\boldsymbol{Ab}$ that match with the partitioned rows of the matrix $\boldsymbol{A}$ to form a partial result of the result vector $\boldsymbol{Ab}$. This submatrix-vector computation can be carried out after each GPU receives the entries of the vector $\boldsymbol{b}$ that correspond to non-zero entries in the rows of the submatrix $\boldsymbol{A}$ which are assigned to each GPU. A simple implementation for this problem is to implement a code that all the GPUs receives all the entries of the entire vector $\boldsymbol{b}$. However, this implementation is not efficient because the number of non-zero entries per a row of a sparse matrix generated by finite element methods is small. To solve this problem, we develop an efficient procedure that each GPU receives from other GPUs only the entries of the vector $\boldsymbol{b}$ which each GPU needs to compute submatrix-vector multiplications.

5 Performance Results

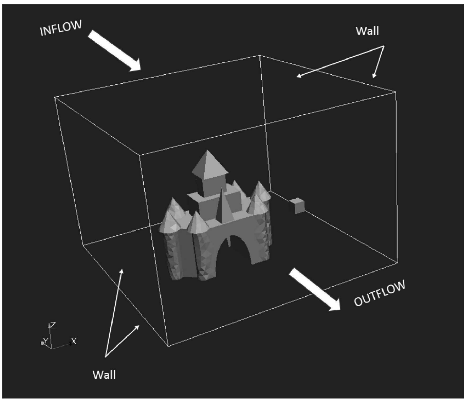

Figure 1: Computational domain of the test problem

We consider a test problem which consists of an object immersed in a fluid domain as shown in Fig. 1. Starting from a surface mesh, we created three tetrahedral meshes with different resolutions. We carried out the computation by using a GPU system with the following specification:

- Intel Xeon CPU E5630, 2.53GHz,
- 2 GPUs - Tesla C2070,
- 24 GB System Memory.

Table 1: Speed-up ratios and execution times

Mesh Size		Two GPUs	Two CPUs	Speed
#Node	#Element	Time	Time	Up
28279	144786	161 s	223 s	1.4
59572	307509	354 s	574 s	1.6
115810	638447	765 s	1396 s	1.8

We measured the time of the execution on two GPU devices and that of two CPU threads. As shown in Tab. 1, the GPU execution is 1.8 times faster than the CPU execution for the large mesh with 638447 elements. The speedup ratio increases as the number of elements of meshes increases.

6 Conclusions

We propose an efficient implementation of a multi-GPU parallel solver based on the GPBi-CG algorithm for 3D unsteady Navier–Stokes equations discretized by the stabilized finite element formulations. We develop a hybrid CPU-GPU strategy that distributes the computation across GPUs by using the MPI library. In future research, we plan to evaluate the scalability of the program (speedup factor versus the number of GPU devices) in multi-GPU high-performance computing systems. We also plan to improve the current implementation by using multigrid preconditioners.

Acknowledgment

This work was supported by Japan Science and Technology Agency, Core Research for Evolutional Science and Technology (JST-CREST). The first author would like to acknowledge Mr. Ryota Yamane (Okayama University) for providing the STL file used for test computations.

References

[1] T. E. Tezduyar, Stabilized finite element formulations for incompressible flow computations, Advances in Applied Mechanics 28 (1992) 1–44.

[2] T. E. Tezduyar, S. Mittal, S. E. Ray, R. Shih, Incompressible flow computations with stabilized bilinear and linear equal order interpolation velocity pressure elements, Computer Methods in Applied Mechanics and Engineering 95 (1992) 221–242.

[3] F. Shakib, T. J. R. Hughes, Z. Johan, A new finite element formulation for computational fluid dynamics: X. the compressible Euler and Navier–Stokes equations, Computer Methods in Applied Mechanics and Engineering 89 (1991) 141–219.

[4] L. P. Franca, S. L. Frey, T. J. R. Hughes, Stabilized finite element methods: I. application to the advective–diffusive model, Computer Methods in Applied Mechanics and Engineering 95 (1992) 235–276.

[5] V. Huynh, H. Suito, A GPU Parallel Solver for 3D Incompressible Navier-Stokes Equations Discretized by the SUPG/PSPG Stabilized Finite Element Formulation, GPU Technology Conference 2016, http://on-demand.gputechconf.com/gtc/2016/posters/GTC_2016_Computational_-Fluid_Dynamics_CFD_01_P6160_WEB.pdf (Accessed Oct. 2016).

[6] S. L. Zhang, GPBi-CG: Generalized product-type methods based on Bi-CG for solving nonsymmetric linear systems, SIAM J. Sci. Comput. 18 (1997) 537–551.

[7] CuSPARSE, https://developer.nvidia.com/cusparse (accessed Oct. 2016).

[8] CuBLAS, https://developer.nvidia.com/cublas (Accessed Oct. 2016).