Matrix Decomposition Based Conjugate Gradient Solver for Poisson Equation

Hang Liu
Department of Electrical and Computer Engineering
George Washington University
Washington DC, USA
Email: {asherliu, howie}@gwu.edu

Jung-Hee Seo Rajat Mittal
Department of Mechanical Engineering
Johns Hopkins University
Baltimore MD, USA
Email: {jhseo, mittal}@jhu.edu

H. Howie Huang
Department of Electrical and Computer Engineering
George Washington University
Washington DC, USA
Email: howie@gwu.edu

Abstract

Finding a fast solver for the Poisson equation is important for many scientific applications. In this work, we design and develop a matrix decomposition based Conjugate Gradient (CG) solver, which leverages Graphics Processing Unit (GPU) clusters to accelerate the calculation of the Poisson equation. Our experiments show that the new CG solver is highly scalable and achieves significant speedup over a CPU-based Multi-Grid (MG) solver.

***Keywords*-Poisson Equation; Graphics Processing Unit (GPU); Conjugate Gradient (CG) solver;**

I. Introduction

The Poisson equation is widely used in scientific and engineering fields, and remains computationally challenging. For example, in computational fluid dynamics, this equation is part of the Navier-Stokes equations which can be utilized to describe a wide range of dynamic fluid phenomena. In this work, we focus on solving the Poisson equation on an integrated domain. In our previous work [1], we use an MG solver on traditional CPU clusters to calculate the Poisson equation. Because Poisson equation is a second derivative partial differential equation, the MG solver may take a lot of CPU time. In this work, we design and develop a GPU-based CG solver that leverages row-oriented matrix decomposition and achieves good scalability and speedup over the MG solver.

In the simulation, we consider a three-dimensional driven cavity problem [2]. The baseline flow solver used in this study is 'Vicar3D' which is developed by our group at JHU [1]. The Poisson equation (3D) can be written as:

$$\Delta p = \nabla^2 p = (\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2})p = f(x, y, z) \quad (1)$$

To solve this Poisson equation numerically, the first step is discretizing the 3-D Poisson equation into a set of difference equations on the domain. Due to the simulation requirement, we discretize Eq. (1) based on non-uniform mesh domain:

$$\begin{aligned}&\frac{1}{\Delta x_i}(\frac{-p_{i+1,j,k}+p_{i,j,k}}{0.5(\Delta x_{i+1}+\Delta x_i)} + \frac{p_{i,j,k}-p_{i-1,j,k}}{0.5(\Delta x_i+\Delta x_{i-1})}) + \\ &\frac{1}{\Delta y_i}(\frac{-p_{i,j+1,k}+p_{i,j,k}}{0.5(\Delta y_{j+1}+\Delta y_j)} + \frac{p_{i,j,k}-p_{i,j-1,k}}{0.5(\Delta y_j+\Delta y_{j-1})}) + \\ &\frac{1}{\Delta z_i}(\frac{-p_{i,j,k+1}+p_{i,j,k}}{0.5(\Delta z_{k+1}+\Delta z_k)} + \frac{p_{i,j,k}-p_{i,j,k-1}}{0.5(\Delta z_k+\Delta z_{k-1})}) \\ &= rhs_{i,j,k} \end{aligned} \quad (2)$$

Further, this equation can be written as:

$$\begin{aligned}&a_{px}p_{i+1,j,k} + a_{mx}p_{i-1,j,k} + a_{py}p_{i,j+1,k} + a_{my}p_{i,j-1,k} + \\ &a_{pz}p_{i,j,k+1} + a_{mz}p_{i,j,k-1} + a_c p_{i,j,k} = rhs_{i,j,k}\end{aligned} \quad (3)$$

where the parameters are the following:

$$a_{px} = \frac{-2}{\Delta x_i(\Delta x_{i+1}+\Delta x_i)},\ a_{mx} = \frac{-2}{\Delta x_i(\Delta x_{i-1}+\Delta x_i)},$$
$$a_{py} = \frac{-2}{\Delta y_j(\Delta y_{j+1}+\Delta y_j)},\ a_{my} = \frac{-2}{\Delta y_j(\Delta y_{j-1}+\Delta y_j)},$$
$$a_{pz} = \frac{-2}{\Delta z_k(\Delta z_{k+1}+\Delta z_k)},\ a_{mz} = \frac{-2}{\Delta z_k(\Delta z_{k-1}+\Delta z_k)},$$
$$a_c = -1 * (a_{px} + a_{mx} + a_{py} + a_{my} + a_{pz} + a_{mz})$$

where (i, j, k) are the indices in (x, y, z) direction respectively. Combining all the difference equations, we can get:

$$\mathbf{A}\vec{x} = \vec{b}, \quad (4)$$

where $\vec{x}$ is the solution vector of which element is $p_{i,j,k}$ and the element of $\vec{b}$ is $rhs_{i,j,k}$.

II. Fast CG Solver

As the problem size grows, one has to decompose the problem into smaller sub-problems and solve them in parallel. In our solver, we first look into the most popular decomposition methods, domain decomposition and functional decomposition [3]. Domain decomposition splits the whole domain into sev-

eral smaller sub-domains and iteratively searches the solution by coordinating the calculation of adjacent sub-domains [4][5]. As a first attempt, we implement the domain decomposition-based CG method [5]. However, the convergence speed is very slow, because our problem does not have a physically separable domain. On the other side, functional decomposition aims to process the functions on the domain in parallel, and it is not applicable for the Poisson equation.

To this end, we design and develop Matrix Decomposition (MD) to solve Eq. (1), where we construct only one large sparse matrix A that is factorized into several smaller canonical forms. Each canonical form contains an equal number of the rows from matrix A. At first glance, the row-oriented matrix decomposition would not scale because every row of the matrix would have to reach from the beginning to the end of vector $\vec{x}$ and $\vec{b}$ in Eq. (4). However, since A is a sparse matrix, every row of A reaches only part of vector $\vec{x}$ and $\vec{b}$. Therefore, this row-oriented matrix decomposition can scale and outperform the typical LU, Block LU, Column-oriented or Cholesky decompositions [6] for the following reasons:

- Row-oriented decomposition avoids the inter-machine communication because every row of A is the stencil of one cell in the domain of Eq. (1), which is computed on one machine.
- Row-oriented decomposition stores only the non-zero elements. The block LU or Column-oriented decomposition will need to store either zeros or index information.
- Row-oriented decomposition can employ multiple GPUs, each of which will solve a number of rows in parallel. On the other hand, the LU decomposition can scale only to two GPUs, and the Cholesky decomposition requires A to be symmetric which is not always the case.

We employ diagonal preconditioner (Jacobi preconditioner) for our problem because it requires no extra storage and computation for the preconditioner. At meantime, due to the decretizing time step is not very small in 'Vicar3D', diagonal preconditioner is enough for stablize A of Eq. (4). Furthermore, we utilize three GPU-based optimizations in the solver: GPU affinity [7], GPU P2P communication [8] and GPU stream launching [8].

III. Experiments

We have implemented our solver in C and CUDA code. We evaluate the solver and the new combined code on Forge, a 153-teraflop supercomputer at the National Center for Supercomputing Applications (NCSA) that is a part of National Science Foundation's Extreme Science and Engineering Discovery Environment (XSEDE) program. The GPU devices installed in Forge are Nvidia Tesla M2070 cards. In this experiment, we mainly study the scalability of our solver on one machine and on multiple machines. We test different problem sizes from 128^3 to 512^3. Note that we are running the cubic problem domains, and in the following, we use 128 to represent the 128^3 problem, 256 for the 256^3, and so on. In the evaluation, we use double precision for calculation. The precision requirement for both our CG solver and MG solver is 10^{-3}. We use gcc 4.4, mpicxx (mpich 1.4), and nvcc 4.0 for compiling.

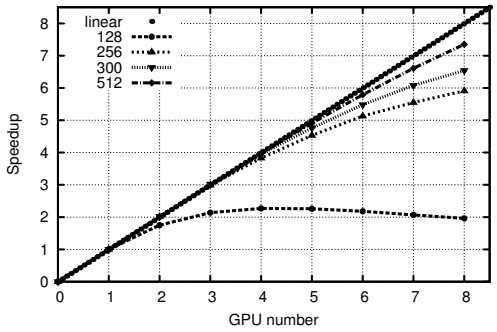

Fig. 1. Speedup on one machine

Figure 1 shows the performance when our CG solver employs different numbers of GPU devices on a single Forge node. Note that since the 512 problem cannot fit in one or two GPU devices, it runs from three to eight GPUs. For the large problems, our solver achieves close to linear speedup, e.g., the 512 problem has the highest speedup of 7.35 among all problem sizes, since the overhead ratio for small problems is larger than big problems.

Further, we compare the performance of the new CG solver and our existing CPU-based MG solver. For the uniform problem size of 512, our CG solver consumes 154 seconds when employing 40 GPU devices, while the previous MG solver consumes 9.4 hours when utilizing 128 CPUs. In other words, our new solver is able to improve the simulation performance by 219 times.

IV. Acknowledgement

This project is in part supported by National Science Foundation grant IOS-1124813, and an NVIDIA Academic Partnership Award.

References

[1] R. Mittal, H. Dong, M. Bozkurttas, F. Najjar, A. Vargas, and A. Von Loebbecke, "A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries," *Journal of Computational Physics*, vol. 227, no. 10, pp. 4825–4852, 2008.

[2] P. Shankar and M. Deshpande, "Fluid mechanics in the driven cavity," *Annual Review of Fluid Mechanics*, vol. 32, no. 1, pp. 93–136, 2000.

[3] I. Foster, *Designing and building parallel programs*. Addison-Wesley Reading, MA, 1995, vol. 95.

[4] A. Quarteroni and A. Valli, *Domain decomposition methods for partial differential equations*. Clarendon Press, 1999, vol. 10.

[5] A. Meyer, "A parallel preconditioned conjugate gradient method using domain decomposition and inexact solvers on each subdomain," *Computing*, vol. 45, no. 3, pp. 217–234, 1990.

[6] H. William and S. Teukolsky, *Numerical Recipes in C: The art of scientific computing*. Cambridge university press, 1988.

[7] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover, "Gpu cluster for high performance computing," in *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. IEEE Computer Society, 2004, p. 47.

[8] Nvidia, "Cuda programming guide," 2008.