A GPU Implementation of a 3D Poisson Solver for Fluid Simulation Using The IDR(s) Algorithm

A GPU Implementation of a 3D Poisson Solver

Huynh Quang Huy Viet

Graduate School of Environmental and Life Science, Okayama University

Email: hqhviet@ems.okayama-u.ac.jp
Phone: +81 (086)-251-8835

岡山大学大学院
環境生命科学研究科
GRADUATE SCHOOL OF
ENVIRONMENTAL AND LIFE SCIENCE
OKAYAMA UNIVERSITY

Abstract

The bi-conjugate gradient stabilized algorithm (Bi-CGSTAB) has often been used as an iterative solver to compute approximate solutions of a large and sparse non-symmetric system of linear equations derived from Poisson equations in fluid simulation. Recently, the IDR(s) algorithm proposed by Sonneveld and Gijzen has been proven to be more efficient than the Bi-CGSTAB algorithm. We proposed a simplified version of the IDR(s) algorithm in which the number of inner products is minimized. In this presentation, we describe our GPU parallel implementation of a solver of 3D Poisson equations using the simplified IDR(s) algorithm.

Introduction

A system of linear equations derived from the discretization of the Poisson equation with Dirichlet and Neumann boundary conditions 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 BICGSTAB is a good algorithm especially when the available memory is relatively small in relation to the system memory size. Recently, the IDR(s) [1] algorithm proposed by Sonneveld and Gijzen has been proven to be more efficient than the Bi-CGSTAB. In this presentation, we describe our GPU parallel implementation of a solver of 3D Poisson equations using the IDR(s) algorithm. We show that our implementation can perform 16 times faster than a CPU-based implementation.

The IDR(s) algorithm

The IDR(s) algorithm [1] is summarized as follows.

Algorithm 1 The IDR(s) algorithm

Require: $A \in C^{N\times N}; x_0, b \in C^N; TOL \in (0,1); MAXIT \geq 0$
Ensure: x_n such that $\|b - Ax_n\| \leq TOL$
  $r_0 = b - Ax_o$;
  for $n = 0$ to s **do**
    $v = Ar_n; \omega = (v^T r_n)/(v^T v)$;
    $\Delta x_n = \omega r_n; \Delta r_n = -\omega v$;
    $r_{n+1} = r_n + \Delta r_n; x_{n+1} = x_n + \Delta x_n$;
  end for
  $\Delta R_{n+1} = (\Delta r_n \cdots \Delta r_0); \Delta X_{n+1} = (\Delta x_n \cdots \Delta x_0)$;
  for $k = s$ to $MAXIT$ **do**
    Solve c from $P^T \Delta R_n c = P^T r_n$;
    $v = r_n - \Delta R_n c$;
    if $mod(k, s+1) = s$ **then**
      $t = Av; \omega = (t^T v)/(t^T t)$;
      $\Delta x_n = -\Delta X_n c + \omega v; \Delta r_n = -\Delta R_n c - \omega t$;
    else if then
      $\Delta x_n = -\Delta X_n c + \omega v; \Delta r_n = -A\Delta x_n$;
    end if
    $r_{n+1} = r_n + \Delta r_n; x_{n+1} = x_n + \Delta x_n$;
    $\Delta R_n = (\Delta r_{n-1} \cdots \Delta r_{n-s})$;
    $\Delta X_n = (\Delta x_{n-1} \cdots \Delta x_{n-s})$;
  end for

In the algorithm, the matrix $P \in C^{N\times s}$ is defined as orthogonalization of a set of random vectors. In order to reduce the number of inner products, we propose to construct the matrix $P \in C^{N\times s}$ as a set of one-element vectors as follows.

$$P = \begin{bmatrix} p_0 & 0 & 0 & 0 & 0 \\ 0 & p_1 & 0 & 0 & 0 \\ 0 & 0 & p_2 & 0 & 0 \\ 0 & 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & 0 & p_s \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (1)$$

We call the IDR(s) algorithm with this P matrix as "the simplified IDR(s) algorithm" [3].

The GPU Implimentation of the Simplified IDR(s) Algorithm

The simplified IDR(s) 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
- setcol - assign a column of a matrix by a vector

These basic operations are implemented by using the Thrust parallel algorithms library. We refered [2] to implement the MV operation. Figure 1 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 1 (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.

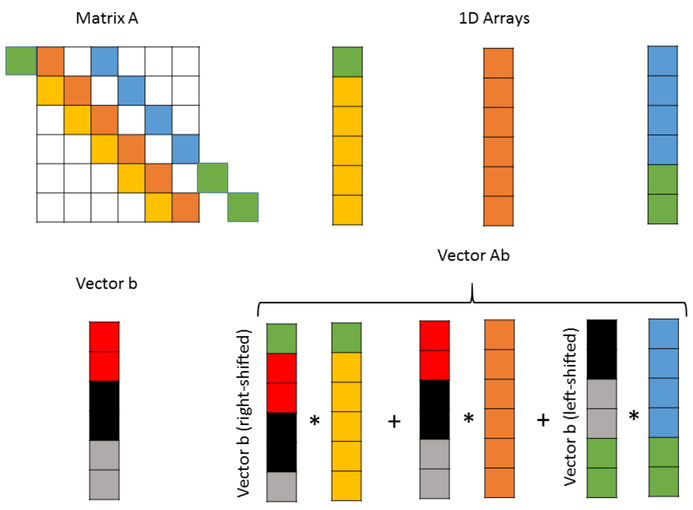

Figure 1: The GPU representation of the banded matrix vector multiplication

Computational Experiments

We present computational results obtained in our system which has a Intel(R) Xeon(R) CPU E5506 2.13GHz and a NVIDIA Tesla C2050 GPU. We consider the three-dimensional Poisson equation with Dirichlet boundary conditions on the unit cube $[0,1] \times [0,1] \times [0,1]$:

$$-\nabla^2 u(x,y,z) = f(x,y,z) \quad (2)$$

The exact solution $u(x,y,z)$ and the corresponding forcing term $f(x,y,z)$ are $u(x,y,z) = sin(xyz)$ and $f(x,y,z) = (x^2y^2 + y^2z^2 + z^2x^2)sin(xyz)$.

Grid Size	GPU Time(s)	CPU Time(s)	Speedup
50x60x70	4.848	27.164	5.6
55x65x75	4.785	39.746	8.3
60x70x80	5.223	63.358	12.2
70x80x90	6.074	86.942	14.2
80x90x100	8.171	130.459	16.0

Table 1: Speedups

The solution is approximated by discretization of the equation by the finite difference scheme with a seven point stencil on a non-uniform 3D grid. The grid sizes are $h_x = \frac{1}{n_x - 1}$, $h_y = \frac{1}{n_y - 1}$ and $h_z = \frac{1}{n_z - 1}$ where n_x,n_y, n_z are the number of grid points in x, y and z directions. The discretization on a non-uniform grid generates a large and sparse non-symmetric system of linear equations $Ax = b$ where the matrix A is of order $N = n_x n_y n_z$. In this experiment, no pre-conditioner is applied. The iteration is terminated when $\|r_n\|/\|b\| \leq TOL = 10^{-9}$.

The resulting speedups are calculated as follows:

$$speedup = \frac{Processing\ \ time\ \ on\ \ CPU}{Processing\ \ time\ \ on\ \ GPU} \quad (3)$$

Table (1) gives the observed speedups of the IDR(s) with $s = 4$ on a linear system derived from discretization of the Poisson equation on a non-uniform 3D grid with different grid sizes.

Conclusions

- We proposed a simplified version of the IDR(s) algorithm in which the number of inner product operations is minimized.
- We described the GPU implementation of a solver of the Poisson equation in three dimensions using the simplified IDR(s) algorithm.

Forthcoming Research

- We will perform many numerical experiments by choosing different combinations of the IDR(s) based Poisson solver and pre-conditioners to find a good pre-conditioner which is most efficient and scalable.
- We will implement a GPU parallel 3D Navier-Stokes solver for blood flow simulations using the simplified IDR(s) algorithm.

References

1. Peter Sonneved, Martin B. van Gijzen, "IDR(s): A Family of Simple and Fast Algorithms for Solving Large Nonsymmetric Systems of Linear Equations", SIAM Journal on Scientific Computing, Vol.31, pp. 1035-1062.
2. Jens Krger, Rdiger Westermann, "Linear algebra operators for gpu implementation of numerical algorithms", ACM Trans. Graph., Vol.22, No. 3, pp. 908-916, 2003.
3. Huynh Quang Huy Viet, Suito Hiroshi, "An MPI parallel implementation of a three-dimensional Poisson solver for fluid simulation using the IDR(s) algorithm", The 27th Computational Fluid Dynamics Symposium, D05-4, 2013.

Acknowledgements

This work is supported by Core Research for Evolutional Science and Technology (CREST) of the Japan Science and Technology Agenecy (JST).