Implementation of a multigrid solver on GPU for Stokes equations with strongly variable viscosity based on Matlab and CUDA

Liang ZHENG∗§, Taras Gerya†, Matthew Knepley‡, David A. Yuen† Huai ZHANG; Yaolin SHI∗

March 17, 2011

Abstract

Stokes equations have been used in numerical simulations of geodynamic processes such as mantle convection, lithospheric deformation and lava flow, etc. In order to implement a solver for these equations, multigrid method is introduced to our solve. Multigrid method is commonly used in reducing the iteration steps for solving the elliptic partial differential equation with the ill-conditioned matrix due to the saddle points in the matrix system coupling mass and momentum equations and strongly variable viscosity due to rheology. Taking the advantages of the current Graphic Processing Units’ (GPU) computing ability and the new Matlab 2010b we utilize the massive programming method to speed up the original Matlab codes with Compute Unified Device Architecture (CUDA). In this paper we will introduce the implementation of a Matlab based multigrid solver for three dimensional Stokes equations with strongly variable viscosity using Red-Black Gauss-Seidel method on GPU.

Keywords: GPU, Matlab, multigrid, Stokes flow, strongly variable viscosity

1 Introduction

Since NVIDIA released the Compute Unified Device Architecture (CUDA) in 2007, utilizing Graphics Processing Units (GPU) for solving numerical problem has become more and more popular. Modern GPU has high performance in parallel computing thanks to its unique architecture with the model of Single Instruction Multiple Thread (SIMT). Moreover, GPU’s energy consumption is very low compared to PC cluster. For example, the current Tesla 2050’s peak

∗Laboratory of Computational Geodynamics, Chinese Academy of Sciences, China
†Institute of Geophysics, ETH-Zurich, Switzerland
‡Computational Institute, University of Chicago, Chicago, Illinois, U.S.
§Minnesota Supercomputing Institute, University of Minnesota, Minnesota, U.S.
single precision floating point performance can reach 1.03 Tflops for a single GPU card which is similar to a small PC cluster while its power consumption is only 238W. This is very important because improved power efficiency allow the device to be utilized with normal power output requirement and large electricity bill.

CUDA uses the model of Grid and Block for software level to manage hundreds of threads which run on GPU’s Streaming Multiprocessors (SM) and Scalar Processor (SP) for hardware level. By now, developing a CUDA applications is still not easy especially when you already have a lot of finished codes which is not written in C. Translating all the codes into C and CUDA is usually inefficient. For one thing, scientists can write script codes first to test their algorithms and then translate them into traditional compiling language later. For another using script language to call C and Fortran codes is very common in the community of scientific computation because it can reduce programming work and the performance has not decreased very much. For GPU we already have PyCUDA (Python call CUDA kernels) and since Mathworks released Matlab 2010b in September 2010 we can easily test the real CUDA codes with Matlab interface by calling Parallel Thread execution (PTX) instructions with Matlab’s parallel computing toolbox. In this paper we focus on using Matlab interface to call PTX kernels compiled from CUDA codes to run on GPU for solving Stokes flow problem.

2 Background

Solving Stokes flow problem is very important in Computational Geodynamics which can describe a lot of geodynamic phenomenal because our earth behaves as incompressible creeping flow for very long-time scale. For example the mantle and subduction can be regarded as flow with very high viscosity for millions of years. Stokes flow is also named creeping flow for which Renold Number is extremely to zero so that we can ignore advective transport term in Navier-Stokes equations. Normally we use the conservation of mass and momentum to denote the steady statues of Stokes flow problem which is indeed Stokes equations showing as following:

\[
\frac{\partial u_i}{\partial x_i} = 0 \tag{1}
\]

\[
\frac{\partial \sigma_{ij}}{\partial x_j} + \frac{\partial P}{\partial x_i} + \rho g_i = 0 \tag{2}
\]

where \( u_i \) is velocity, \( \sigma_{ij} \) is stress, \( P \) is pressure, \( \rho \) is density and \( g_i \) represents acceleration of gravity. Although sometimes we can get the analytic solutions to study the Stokes flow problems [Turcotte and Schubert, 2002; Schubert et al., 2001; Payne and Pell, 1960; Zhong, 1996], in most situations using modern high performance computers to get the numerical solutions is the only choice. In fact
the coupled equations of (1) and (2) cause the system to saddle point problem which makes the equations hard to solve. Equation (3) is the constitutive relationship of stress and strain rate \( \varepsilon_{ij} \) which is related to the partial derivative of velocity with respect to the directions. \( \mu \) is the viscosity which can describe the rheology of the physical earth. Generally speaking for the effective viscosity of rocks there is an empirical formula as :

\[
\eta_{\text{eff}} \propto \exp \left( \frac{E_a + V_a P}{RT} \right)
\]

where \( E_a \) is activation energy, \( V_a \) is activation volume, \( R \) is gas constant, \( P \) is pressure and \( T \) is temperature. From this empirical formula we can find that effective viscosity may vary many orders of magnitudes even with small changes of environmental properties like temperature or pressure (Gerya, 2010). This so called strongly variable viscosity causes the system of Stokes equations very ill-conditioned so that there are not many proven methods can be used to solve it. (Deubelbeiss and Kaus, 2008; Moresi et al., 1996)

\[
\sigma_{ij} = 2\eta \varepsilon_{ij} = \eta \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \tag{3}
\]

Iterative method is considered to be suitable for solving large scale sparse linear system. However, due to the ill-conditioned matrix system, traditional iterative method is often hard to get the convergence solutions. Using multigrid method to accelerate the iterative convergence is a better way for large classes of numerical problems that can solve the \( N \) unknowns problems with \( O(N) \) time complexity. The first multigrid method was formulated for the standard five-point finite difference method of the Possion equation in 1964 by Bachvalov. Although understanding the meaning of multigrid is not difficult, it had not been widely used for practice because it lacked good theoretical analysis. Since 1976 when Hackbusch proved it with mathematical theory, an increasing number of computational scientists started putting multigrid method into serious research. (Hackbusch, 1977, 1978; Bachvalov, 1966; Wesseling, 1991). By using multigrid method the residuals with longer wavelength can decay faster than traditional method. In another word, iterative functions run on grids with different resolutions so that the iterative information can propagate faster than those only run on the grids with highest resolution. For this reason multigrid method is also used in simulating mangle convection for solving variable-viscosity problem (Gerya, 2010; Kameyama et al., 2005; Auth and Harder, 1999; Oosterlee and Lorenz, 2006). In this paper we use a sinking model which is like a block with high viscosity (also with the higher density) sinks into the medium part with low viscosity (figure 1). Here we set a contrast of \( 10^6 \) for the viscosity structure between the block part and the medium part. All the velocity boundaries are set to free slip boundary conditions.

3 Implementation

In order to avoid odd-even decoupling between the pressure and velocity, the conservative finite differences are applied for the discrete equations. This
method can satisfy the conservation of stresses between nodal points for the three dimensional staggered grids. Equations (4) and (5) are given below as an example showing the details of discrete continuum equation and discrete x-direction Stokes equation with the stencil of staggered grids.

\[
\begin{align*}
\frac{u_{i+1,j+1,l-1} - u_{i+1,j+1,l+1}}{\Delta x_{j+1/2}} + \frac{u_{i+1,j+1,l} - u_{i,j+1,l+1}}{\Delta y_{l+1/2}} + \frac{u_{i+1,j+1,l+1} - u_{i+1,j+1,l}}{\Delta z_{l+1/2}} &= 0 \\
(4)
\end{align*}
\]

\[
\begin{align*}
4\eta_n(i-1,j,l-1) &\left( \frac{u_{i,j+1,l} - u_{i,j+1,l-1}}{\Delta x_{j+1/2}(\Delta x_{j+1/2} + \Delta x_{j+1/2})} - \frac{u_{i,j+1,l} - u_{i,j+1,l}}{\Delta y_{l+1/2}(\Delta y_{l+1/2} + \Delta y_{l+1/2})} \right) + \\
2\eta_x(i-1,j,l-1) &\left( \frac{u_{i,j+1,l} - u_{i,j+1,l-1}}{\Delta y_{l+1/2}(\Delta y_{l+1/2} + \Delta y_{l+1/2})} - \frac{u_{i,j+1,l} - u_{i+1,j,l}}{\Delta y_{l+1/2}(\Delta y_{l+1/2} + \Delta y_{l+1/2})} \right) + \\
2\eta_y(i+1,j,l) &\left( \frac{u_{i+1,j,l} - u_{i+1,j,l}}{\Delta z_{l+1/2}(\Delta z_{l+1/2} + \Delta z_{l+1/2})} - \frac{u_{i+1,j,l} - u_{i+1,j+1,l}}{\Delta z_{l+1/2}(\Delta z_{l+1/2} + \Delta z_{l+1/2})} \right) = 0 \\
(5)
\end{align*}
\]

For solving this coupled system the original codes implemented with Multi-grid method based on Matlab was already developed by Taras. By analyzing the time consumption (figure 2) of each part of the original Matlab codes which

\*http://www.cambridge.org/gb/knowledge/isbn/item2709959

Figure 1: Testing model
was written by Taras we can find that the smoother, restriction and prolongation functions almost take all of the time. To see what happened with a real compiled language, these most reused functions was first translated into C code which can be called by Matlab with mexfunction. Table 1 shows the comparison of the time consumption between the original codes of Matlab and the codes of Matlab calling C function on a Intel Core i7 CPU (3.07GHz). From Table 1 we can see even rewriting those most reused functions with C code can speed up the original codes but it’s not enough. No matter the original Matlab codes or the Matlab codes calling C functions, the smoother which takes most part of time consumption still need to be improved. Using GPU the sequential inner cycle over all the grid points can be modified to parallel threads with the SIMT model. This is the original intention of this paper.

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Original Matlab</th>
<th>Matlab with C</th>
</tr>
</thead>
<tbody>
<tr>
<td>25<em>25</em>25</td>
<td>160 sec</td>
<td>15 sec</td>
</tr>
<tr>
<td>49<em>49</em>49</td>
<td>1108 sec</td>
<td>125 sec</td>
</tr>
</tbody>
</table>

The V-cycle multigrid method has two interpolation operations one is the restriction and the other is prolongation (figure 3). In fact in order to speed up the convergence for strongly variable viscosity we applied an approach named multi-multigrid approach. It rescales the viscosity with gradual increase for the contrast to run the iterations with different viscosity on each level for the both restriction and prolongation procedures. At the end of the V-cycle we add the results with the computed corrections at the the finest grid and run the iterations again on the finest grid for every given cycles. Equation 6 shows the detail of viscosity rescaling formula in which \( \eta_{\text{computational}} \) and \( \eta_{\text{original}} \) are current and original viscosity respectively.
As we mentioned before, we will use Matlab’s parallel computing toolbox which supports GPU computing with script language. There is indeed no difference for writing CUDA kernel codes between the traditional CUDA and our implementation. What we need to do is to rewrite functions with CUDA codes and compile them into PTX file not the binary file. Matlab can call the PTX codes using the Kernel Object in which we can set the size of Grid and Block as usual.

To avoid the disordered threads running on GPU, we will apply Red-Black Gauss-Seidel (RBGS) (figure 4) algorithm for GPU codes. The RBGS divides the Gauss-Seidel iteration into two parts by red and black colors so that one point can always use the newest information around itself. In order to reduce logic operations (there is only one logic unit on SM which means the logic operations may run 16 times (half wrap) without doing anything) boundary points should be set before the CUDA kernels. We can describe the whole workflow as algorithm 1.

Because of the difference of the index system between Matlab and C, Matlab’s index begins from 1 and uses column-major while C language’s index begins from 0 and uses row-major, we defined a series of macros to transfer index as following:

```cpp
1 #define vx(i, j, k) vx[(i-1)+(j-1)*(ynum+1)+(k-1)*(xnum)*(ynum+1)]
2 #define RX(i, j, k) RX[(i-1)+(j-1)*(ynum+1)+(k-1)*(xnum)*(ynum+1)]
3 #define vy(i, j, k) vy[(i-1)+(j-1)*(ynum)+(k-1)*(xnum+1)*(ynum)]
4 #define RY(i, j, k) RY[(i-1)+(j-1)*(ynum)+(k-1)*(xnum+1)*(ynum)]
5 #define vz(i, j, k) vz[(i-1)+(j-1)*(ynum+1)+(k-1)*(xnum+1)*(ynum+1)]
6 #define RZ(i, j, k) RZ[(i-1)+(j-1)*(ynum-1)+(k-1)*(xnum-1)+(ynum+1)]
7 #define pr(i, j, k) pr[(i-1)+(j-1)*(ynum-1)+(k-1)*(xnum-1)+(ynum-1)]
8 #define RC(i, j, k) RC[(i-1)+(j-1)*(ynum-1)+(k-1)*(xnum-1)+(ynum-1)]
```

With the macros we defined the Matlab codes can be translated into C and CUDA codes in the style of three dimensional arrays which are indeed one

---

Figure 3: V-cycle multigrid

Figure 4: Red-Black method
Algorithm 1 V-cycle multigrid with Red-Black Gauss-Seidel

1: initialize the density, viscosity and unknowns;
2: smoother(iternum=1){
3: for $k = 1$ to iternum do
4: set velocity boundaries
5: run kernels of red color
6: run kernels of black color
7: end for
8: compute the residuals
9: }
10: while (The mean residual > $10^{-4}$) do
11: for $n = 1$ to levelnum do
12: smoother(iternum=iternum(n))
13: run restriction
14: end for
15: for $n = levelnum$ to 1 do
16: smoother(iternum=iternum(n))
17: run prolongation
18: end for
19: end while

dimensional arrays. Taking vx of red color for example we write the kernel codes as following:

```cpp
#include "Index.h"

global void rb_vx_r( ... )
{
    int i=blockIdx.x;
    int j=blockIdx.y;
    int k=threadIdx.x;

    //+2 means start from 1 and skip the boundary points
    i+=2; j+=2; k+=2;

    //decide if it's the red nodes
    if ((i+j+k)%2!=0) return;

    double resxcur, kfxcur;
    double vx0=vx(i, j, k);

    resxcur=RX(i, j, k)+(pr(i-1, j, k-1)-pr(i-1, j-1, k-1))/xstp;
    resxcur=resxcur-(xkf2*(etan(i-1, j, k-1)*(vx(i, j+1, k)-vx0)-etan(i-1, j-1, k-1)*(vx0-vx(i, j-1, k))));
    resxcur=resxcur-(etaxy(i, j, k-1)*(ytfv*(vx(i+1, j, k)-vx0)+xykf*(vy(i, j+1, k-1)-vy(i, j, k)))-etaxy(i-1, j, k-1)*(ytfv*(vx0-vx(i-1, j, k))+xykf*(vy(i-1, j-1, k-1)-vy(i-1, j, k))));
    resxcur=resxcur-(etaxz(i-1, j, k)*(zkf*(vx(i, j, k+1)-vx0)+xzkf*(vz(i, j, k+1)-vz(i, j, k)))-etaxz(i-1, j, k-1)*(zkf*(vx0-vx(i, j, k-1))+xzkf*(vz(i, j-1, k-1)-vz(i, j, k))));
    kfxcur=-xkf2*(etan(i-1, j, k-1)+etan(i-1, j-1, k-1)-ytfv*etaxy(i, j, k))
```

7
−1)+etaxy(i−1,j,k−1)−zkf∗(etaxz(i−1,j,k)+etaxz(i−1,j,k−1));

vx(i,j,k)=vx0+resxcur/kxcur*krelaxs;

To call the CUDA kernels, defining the Kernel Object in Matlab and setting the size of block and grid are needed as what we do in CUDA C languages. The example for vx of red color is also given in the following:

1 % set the kernel in the main function
2 global rb.vx.r_kernel;
3 rb.vx.r_kernel=parallel.gpu.CUDAKernel(‘rb.vx.r.ptx’,‘rb.vx.r.cu’);
4
5 % call the kernel in the smoother function
6 rb.vx.r_kernel.ThreadBlockSize=[znum−1 1 1];
7 rb.vx.r_kernel.GridSize=[ynum−1 xnum−2];
8 [vx,vy,vz,pr]=feval(rb.vx.r_kernel,vx,vy,vz,pr,...
9 RX,RY,RZ,RC,etaxy,etaxz,etayz,etan,...
10 krelaxs,krelaxc,...
11 xnum,ynum,znum,xstp,ystp,zstp,...
12 xkf,ykf,zkf,xkf2,ykf2,zkf2,xykf,xzkf,yzkf);  

Although the new version of Matlab can manage the variables on GPU with gpuArray and CPU with workspace together, that means we do not need to use cudaMalloc or cudaMemcpy any more, GPU and CPU uses different memory systems which are communicated by PCIE bus. The bandwidth of PCIE is very limited compared to GPU that all the data should be operated on GPU all the time including restriction and prolongation with different grids.

4 Performance Analysis

Our simulation were run on a single NVIDIA Tesla 2070C GPU with Intel Core i7 3.04 GHz CPU and 12GB memory. On one hand the original Matlab code is considerably slow, it is hard to simulate large scale problem; On the other hand, GPU is just suitable for many threads with excellent time efficiency. For large scale problems, it is not applicable to compare their performance between the original Matlab codes and the codes of Matlab with CUDA. So only comparison of the codes of Matlab calling C functions (table 2). We can find the codes of Matlab with CUDA run much faster than the already modified codes of Matlab with C. At the same time using GPU can extremely increase the resolution for the Matlab codes on a simple machine within the tolerable time consumption.

In this calculation we set the tolerance of the average $\|\text{residual}\|$ as $10^{-5}$. Double precision is used here. Residual seems difficult to converge when using single precision (figure 5). We set 6 level grids for V-cycle multigrid method with iteration numbers as 10, 20, 40, 80, 160, 320 of each level respectively. Smoother still takes most of the running time for the GPU (figure 6) codes so even a bit of improvement of the smoother’s performance can make big benefit for the whole code. We summarized the time consumption of the smoother function for one step for different resolutions (10 iterations for each velocity cycle are set here).

8
Table 2: Iterations and time on different models

<table>
<thead>
<tr>
<th>Model</th>
<th>V-cycles (Matlab with CUDA)</th>
<th>Time  (Matlab with CUDA)</th>
<th>V-cycles (Matlab with C)</th>
<th>Time  (Matlab with C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64<em>64</em>64</td>
<td>65</td>
<td>133s</td>
<td>148</td>
<td>477s</td>
</tr>
<tr>
<td>128<em>128</em>128</td>
<td>85</td>
<td>387s</td>
<td>211</td>
<td>5975s</td>
</tr>
<tr>
<td>256<em>128</em>128</td>
<td>106</td>
<td>828s</td>
<td>210</td>
<td>11200s</td>
</tr>
<tr>
<td>256<em>256</em>128</td>
<td>106</td>
<td>1794s</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Figure 5: Mean absolute residual of double and single precision

Figure 6: Mean absolute residual of double and single precision
Table 3: Comparison of speedup on different models for smoother

<table>
<thead>
<tr>
<th>Model</th>
<th>Matlab with CUDA</th>
<th>Matlab with C</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>64<em>64</em>64</td>
<td>0.34s</td>
<td>0.66s</td>
<td>1.9x</td>
</tr>
<tr>
<td>128<em>64</em>64</td>
<td>0.58s</td>
<td>1.85s</td>
<td>2.2x</td>
</tr>
<tr>
<td>128<em>128</em>64</td>
<td>1.08s</td>
<td>3.90s</td>
<td>3.6x</td>
</tr>
<tr>
<td>128<em>128</em>128</td>
<td>2.06s</td>
<td>9.15s</td>
<td>4.4x</td>
</tr>
<tr>
<td>256<em>128</em>128</td>
<td>3.99s</td>
<td>19.64s</td>
<td>4.9x</td>
</tr>
<tr>
<td>256<em>256</em>128</td>
<td>9.04s</td>
<td>43.35s</td>
<td>4.8x</td>
</tr>
</tbody>
</table>

5 Conclusions and Future

Multigrid method for numerical simulating Stokes flow problem is suitable for GPU architecture. Time efficiency can be improved on GPU with parallel Red-Black Gauss-Seidel smoother. By using new version Matlab’s parallel computing toolbox we can quickly implement the hybrid Programming of script language and CUDA C. Beside multigrid, Krylov subspace method is often used to speed up the iteration. Some Krylov subspace based iterative solver such as GMRES and CG\(^\dagger\) and GPU-Accelerated preconditioned iterative linear solver\(^\dagger\) have already developed. Meanwhile, deploying multigrid method as a preconditioner of Krylov subspace is very common because multigrid can decay the residual of long wavelength in short time. Above all using some hybrid GPU-CPU methods such as using MPI combined with CUDA for further increase in the resolution can be considered.

6 Acknowledge

We thank the discussion with Masanori C. Kameyama. We also appreciate Yichen Zhou and Aiyu Zhu’s help for this paper. This research is supported from NSF CMG program and Project SinoProbe-07 of China.

References


\(^\dagger\)http://code.google.com/p/cusp-library/

\(^\dagger\)http://www-users.cs.umn.edu/~saad/PDF/umsi-2010-112.pdf


