Article

International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

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

The International Journal of High Performance Computing Applications
2014, Vol. 28(1) 50–60
© The Author(s) 2013
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/1094342013478640
hpc.sagepub.com
SAGE

Liang Zheng[1,2,3], Huai Zhang[1,2], Taras Gerya[4], Matthew Knepley[5], David A Yuen[3,6] and Yaolin Shi[1,2]

Abstract

The Stokes equations are frequently used to simulate geodynamic processes, including mantle convection, lithospheric dynamics, lava flow, and among others. In this study, the multigrid (MG) method is adopted to solve Stokes and continuity equations with strongly temperature-dependent viscosity. By taking advantage of the enhanced computing power of graphics processing units (GPUs) and the new version of Matlab 2010b, MG codes are optimized through Compute Unified Device Architecture (CUDA). Herein, we illustrate the approach that implements a GPU-based MG solver with Red–Black Gauss–Seidel (RBGS) smoother for the three-dimensional Stokes and continuity equations, in a hope that it helps solve the synthetic incompressible sinking problem in a cubic domain with strongly variable viscosity, and finally analyze our solver's efficiency on a GPU.

Keywords

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

1 Introduction

Graphics processing units (GPUs) are increasingly being used to solve numerical problems, since NVIDIA first released the Compute Unified Device Architecture (CUDA) in 2007. Modern GPUs enjoy a laudable performance in parallel computing thanks to the unique architecture of single instruction multiple thread (SIMT) it applies. With a different design philosophy to the central processing unit (CPU), the GPU is oriented by throughput design with many cores, but without powerful or adequate cache, branch prediction or data forwarding functionalities. GPUs consume much less energy, compared with PC clusters of the same computing capacity. For example, Tesla 2050 has a peak performance of 1.03 Tflops for single precision floating point operation and 515 Gflops for double precision floating point operation on a single GPU card with the max power consumption at 238 W, while a single i7-960 CPU with the limited performance of 3.2 Gflops needs a power consumption at 130 W. This is one critical issue because the enhanced power efficiency economizes the device for a noticeably reduced electricity bill.

CUDA is a parallel programming model designed to manage thousands of threads running on the streaming multiprocessors (SMs) and the streaming processors (SPs) at the hardware level of GPU. CUDA threads are organized in blocks and grids at the software level, where one grid contains numerous blocks, and one block contains numerous threads. This architecture hierarchy allows the threads in the same block to communicate with each other through the shared memory using SMs and barrier synchronization. The number of threads in each block and the number of blocks in each grid have to be defined explicitly in CPU codes (host codes) which are usually written in C or other conventional languages such as Python and Fortran. CUDA, as a high-level language, allows a programmer to use CUDA C to define GPU kernel's functionalities. As a

[1] Key Laboratory of Computational Geodynamics, University of Chinese Academy of Sciences, China
[2] College of Earth Science, University of Chinese Academy of Sciences, China
[3] Minnesota Supercomputing Institute, University of Minnesota, MN, USA
[4] Institute of Geophysics, ETH-Zurich, Switzerland
[5] Computational Institute, University of Chicago, Chicago, IL, USA
[6] School of Environmental Sciences, China University of Geosciences, Wuhan, China

Corresponding author:
Huai Zhang, Yaolin Shi, Key Laboratory of Computational Geodynamics, University of Chinese Academy of Sciences, 19A Yuquan Road, Shijingshan, Beijing 100049, China.
Email: huaizhang@gmail.com; shiyl@ucas.ac.cn

result, CUDA C kernel functions (device codes), which are called by host codes in principal, could be able to be executed in parallel through CUDA threads (NVIDIA, 2011).

Developing CUDA applications is still challenging, especially when most codes are written in other languages rather than C. On the one hand, script codes, such as Matlab and Python, are often used to implement and optimize the algorithms. Furthermore, using script language as an interface to call C and Fortran codes is commonplace in scientific computation, as it reduces the workload of programming without compromising the core performance. Fortunately, some script programming tools, such as Jacket Matlab toolbox[1] and PyCUDA (Klöckner et al., 2009), have already supported porting CUDA kernels. The latest version of Matlab (Matlab 2010b or newer versions) released a parallel computing toolbox available to support GPU computing. Compared with the Jacket Matlab toolbox, Matlab 2010b can call parallel thread execution (PTX) directly. PTX provides a low-level instruction set for CUDA programming, similar to the assemble language applied in the x86 architecture. The handwriting CUDA kernels can be transformed into PTX codes with –ptx flag at command line while compiling the codes , which can be called as Matlab functions as well. In this paper, we discuss how to solve the three-dimensional Stokes flow problem in detail using PTX kernels called by the Matlab 2010b.

2 Background of Stokes flow problem

The Stokes flow can be applied to study geodynamical phenomena, as the Earth behaves like an incompressible creeping flow that would last for hundreds of millions of years in its history. For example, Earth's mantle can be deemed as a flow with very high viscosity that has sustained for millions of years. Stokes flow, also named as creeping flow, has a relatively low Reynolds number when the advective transport term in the Navier–Stokes equations is negligible. In general, we utilize the conservation of mass and momentum to describe the stable status of Stokes flow problem as follows:

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{1}$$

$$\frac{\partial \sigma'_{ij}}{\partial x_j} - \frac{\partial P}{\partial x_i} + \rho g_i = 0 \tag{2}$$

where u_i is velocity, σ'_{ij} is the deviatoric stress, P is pressure, ρ is density and g_i represents acceleration of body force such as the gravity in most cases. Equation (1) is the continuity equation, and Equation (2) is indeed the Stokes equation. Einstein's summation convention is used here. In fact, the coupled equations of (1) and (2) lead the system to saddle point problem, which needs to meet the LBB criteria (also known as the Ladyzhenskaya–Babuska–Breezi condition) when a numerical scheme is implemented. LBB criteria is a compatibility condition between the velocity and pressure spaces, which is necessary to yield stable pressure approximations. The constitutive relationship bridging the velocity u_i and the deviatoric stress σ'_{ij} in Equations (1) and (2) is defined as

$$\sigma'_{ij} = 2\eta\dot{\epsilon}_{ij} = \eta\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{3}$$

in which, $\dot{\epsilon}_{ij}$ is the strain rate and the viscosity η describes the rheology of the fluid.

In most cases, the numerical algorithms built on high-performance computing philosophy are proved more realistic, although analytic solutions can be used to address the flow problem given simple geometries and boundary conditions (Payne and Pell, 1960; Schubert et al., 2001; Turcotte and Schubert, 2002). To date, many numerical methods have been developed and applied to solve the Stokes equations, including the finite difference method, the finite volume method, and the finite element method (Patankar, 1980; Elman et al., 2005; Lynch, 2005). However, in the context of geodynamical modeling, the Stokes equations will produce difficulties numerically due to strongly variable coefficients, and hence need more wisdom to be handled (Moresi et al., 1996; Deubelbeiss and Kaus, 2008). This effects from temperature-dependent viscosity can be summarized as

$$\eta_{eff} \propto \exp\left(\frac{E_a + V_a P}{RT}\right) \tag{4}$$

where, η_{eff} represents the effective viscosity, E_a represents activation energy, and V_a activation volume, R is the gas constant, with P for the pressure and T the temperature. Apparently, the effective viscosity may vary by orders of magnitudes even with a small change of environmental properties such as temperature or pressure. Some previous studies reported using MG methods to accelerate the iterative convergence in the Stokes flow problem with strongly variable viscosity (Auth and Harder, 1999; Kameyama et al., 2005; Tackley, 2009; Oosterlee and Lorenz, 2006; Gerya, 2010), which solves the N unknown problems approximately with $O(N)$ time complexity. The MG method was recast for the first time by Bachvalov and Fedorenko in 1964, based on the standard five-point finite difference scheme applied in the Poisson equation. The method has been applied in a wide range of previous studies (Fedorenko, 1964; Bachvalov, 1966; Hackbusch, 1977, 1978; Wesseling, 1991). An MG method allows people to run iterations between coarse and refined grids. As a result, iterative information propagates faster, and the residuals with a longer wavelength decay faster, compared with the one running on the finest grids. MG methods are capable to solve the problems across several grids with different resolutions, mainly through three operations: restriction, smoothing, and prolongation. Restriction projects the coefficients from finer grids to the coarser one, while prolongation interpolates the coefficients simply in an opposite manner. Smoothing (smoother) runs limited iterations at each point of different grids. There are two MG methods that are frequently used: geometric multigrid (GMG) and algebraic multigrid (AMG). AMG is more applicable,

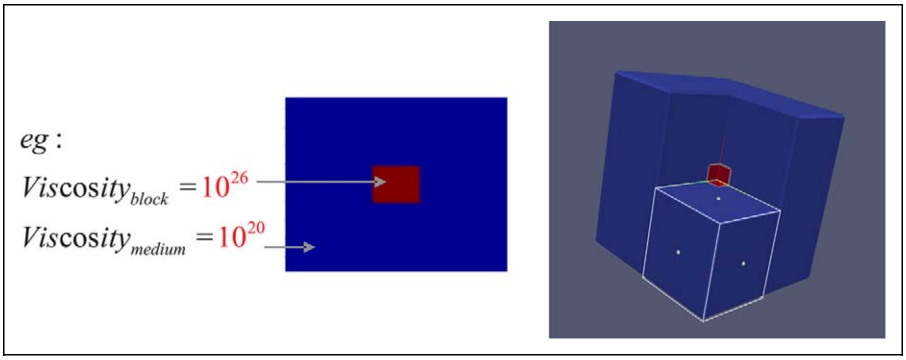


Figure 1. Testing model (SINKER): a block sinks into the medium with strongly variable viscosity. In this model, we set the sinker's density as 3100 kg/m^3 and the viscosity as 10^{26} Pa·S The medium's density and viscosity are 3000 kg/m^3 and 10^{20} Pa·S, respectively.

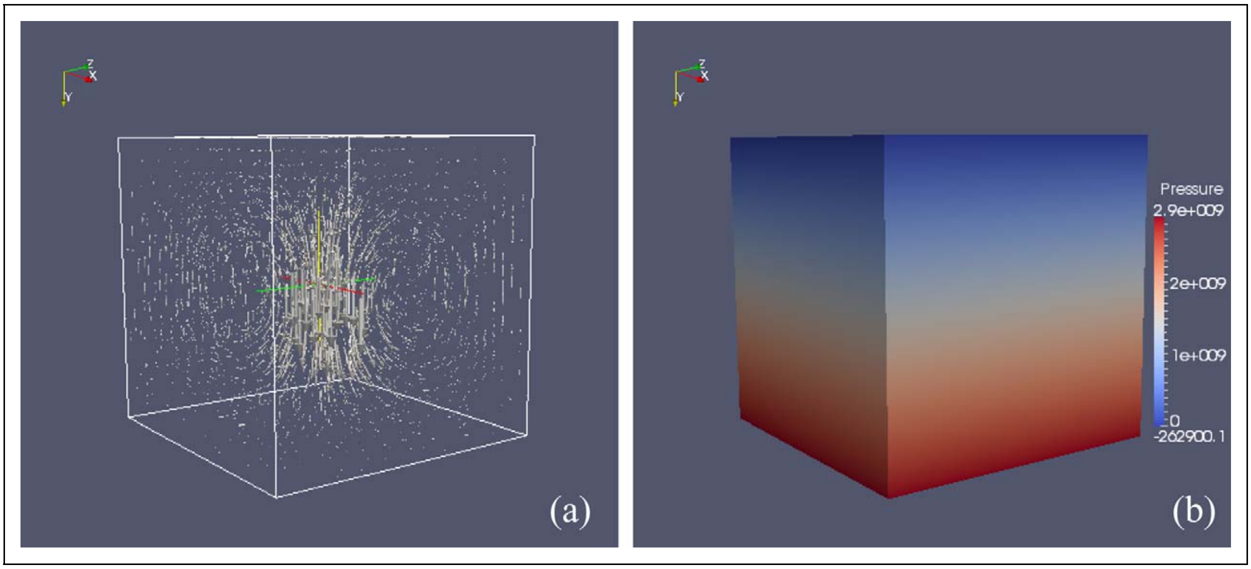


Figure 2. Computed result: (a) velocity; (b) pressure.

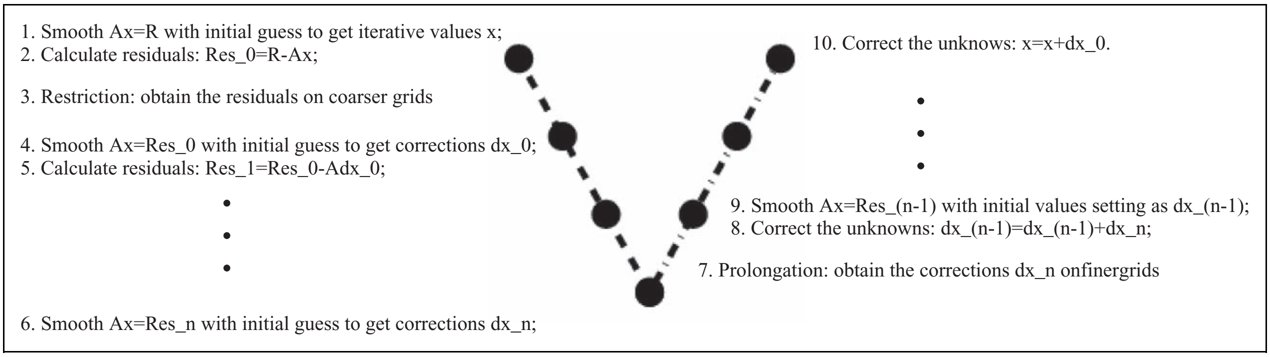
compared with GMG, especially for the finite element method that needs an explicitly built-up linear matrix system, but less efficient in performance compared with the latter. A GPU-based GMG solver has already been released, but fully rewritten in C++ that we cannot reuse the existing codes (Cohen and Molemaker, 2009). We also implemented a two-dimensional GMG solver fully rewritten in CUDA, and a fairly primitive three-dimensional version with MATLAB and CUDA (Zheng et al., 2013).

In this study, we introduce an optimized three-dimensional GMG implementation in detail by solving the Stokes flow problem using a cubic sinking model (SINKER) under the Cartesian coordinate system, which is supposed that a high-viscosity and high-density block sinks in the fluid medium part with a low viscosity (Figure 1). A contrast of the viscosity structure at 10^6 is set between the block part and the medium part. All of the velocity boundaries are assigned to be free slip boundary conditions. This SINKER problem has been widely discussed as a benchmark in geodynamic modeling (May and Moresi, 2008; Gerya, 2010; Furuichi et al., 2011).. Figure 2 shows our SINKER problem.

3 Implementation

3.1 CPU version

To solve the Stokes system, the Matlab codes on the single CPU version have already been implemented using an applicable V-cycle MG based finite difference method (Gerya, 2010). Our work for the GPU version is also based on it. V-cycle (named after the letter 'V'), as shown in Figure 3, is the simplest MG method featured with a restriction operation that runs directly from the finest level to the coarsest level, a prolongation operation running in the opposite manner, and a smoother running at different levels. The original Matlab codes use GMG, taking advantage of the regular nature of the cubic model. The GMG method computes the residuals using the initial guess or computed unknowns on the finest grids, and takes the residuals as the right-hand side on the coarser grids to calculate the corrections for the unknowns on the finer grids. In another word, on coarser grids, GMG explicitly runs some iterations to get the corrections for the unknowns on finer grids. On different grids, the residuals and corrections are obtained by projection that restriction plays the role of computing the



1. Smooth Ax=R with initial guess to get iterative values x;
2. Calculate residuals: Res_0=R-Ax;
3. Restriction: obtain the residuals on coarser grids
4. Smooth Ax=Res_0 with initial guess to get corrections dx_0;
5. Calculate residuals: Res_1=Res_0-Adx_0;

•
•
•

6. Smooth Ax=Res_n with initial guess to get corrections dx_n;
7. Prolongation: obtain the corrections dx_n onfinergrids
8. Correct the unknowns: dx_(n-1)=dx_(n-1)+dx_n;
9. Smooth Ax=Res_(n-1) with initial values setting as dx_(n-1);

•
•
•

10. Correct the unknowns: x=x+dx_0.

Figure 3. V-cycle GMG: A represents the coefficients of the discrete equations, R represents the right-hand side, x represents the unknowns, Res_0,1, . . . , n represents the residuals at each level, and dx_0,1, . . . , n represents the corrections on finer grids respectively. (For example, dx_1 is the correction of dx_0.) The black dots are smoothing operations. The dotted lines on the left-hand side and right-hand side are restrictions and prolongations, respectively.

residuals, and prolongation calculates the corrections. A brief description of V-cycle GMG is shown in Figure 3.

For the Stokes system, the smoother must be applicative to deal with pressure which does not show up in the continuity equations. To explicitly compute the pressure, a computational compressibility approach is used by updating the pressure using the computed residuals of the continuity equation at each iteration:

$$P^{new} = P + \eta Res^{continuity}\theta^{continuity} \tag{5}$$

where P^{new} is the pressure to be solved, P is the pressure obtained from the previous iteration, η is the local viscosity, $Res^{continuity}$ is the residual of continuity equation, and $\theta^{continuity}$ is the relaxation coefficient of continuity equation.

For the strongly variable viscosity problem, the big contrast of coefficients hinder the convergence rate, especially for the initial iterations. We applied the approach to gain the initial guess of the unknowns with a gradually increased computational viscosity contrast to overcome it, which is named as a continuation method. In the beginning of the V-cycle, the viscosity is rescaled to a low-viscosity contrast. After some iterations the computational viscosity contrast gradually increases until the original viscosity values of the problem are restored. The rescaling operations are shown as follows:

$$\eta_{i,j} = \eta_{min}^{computational} \exp\left[\frac{\ln\left(\eta_{max}^{computational}/\eta_{\mathrm{min}}^{computational}\right)}{\ln\left(\eta_{max}^{original}/\eta_{min}^{original}\right)} \ln\left(\frac{\eta_{i,j}}{\eta_{min}^{original}}\right)\right] \tag{6}$$

where $\eta^{computational}$ and $\eta^{original}$ represent current and original viscosity, respectively, with η_{min} and η_{max} for the minimal and maximal viscosity of the model. $\eta_{i,j}$ depicts the viscosity on each grid. One alternative is to use a small modification of multigrid (MG) that right-hand side on the finest grids is also assigned with the residual obtained in the previous cycle. Together with the gradual increasing in a computational viscosity contrast, the convergence performance for large viscosity contrasts can be achieved efficiently. This method is also named as 'multi-multigrid' (Gerya, 2010).

In addition, we applied the staggered schemes to avoid decoupling of odd–even problem, which meets the requirement of LBB condition at the same time, as we have mentioned. The conservative finite difference scheme is applied for variable viscosity case, which satisfies the conservation of stress between the nodal points on the three-dimensional staggered grids. Equations (7) and (8) are given below as an example of discretizing the continuum equation and discrete Stokes equation in the x-direction. In case of utilizing the staggered grids, that different variables have individual index systems, respectively (see Figure 4), where viscosity is defined as η_{n}, η_{xy}, η_{yz} and η_{xz} corresponding to the components of normal stress and shear stress:

$$\begin{aligned} &\frac{u_{x(i+1,j+1,l+1)} - u_{x(i+1,j,l+1)}}{\Delta x_{j+1/2}} + \frac{u_{y(i+1,j+1,l+1)} - u_{y(i,j+1,l+1)}}{\Delta y_{i+1/2}} \\ &\quad + \frac{u_{z(i+1,j+1,l+1)} - u_{z(i+1,j+1,l)}}{\Delta z_{l+1/2}} = 0 \end{aligned} \tag{7}$$

$$\begin{aligned} &\left(4\eta_{\mathrm{n}(i-1,j,l-1)} \frac{u_{x(i,j+1,l)} - u_{x(i,j,l)}}{\Delta x_{j+1/2}\left(\Delta x_{j-1/2} + \Delta x_{j+1/2}\right)}\right. \\ &\qquad \left. -4\eta_{\mathrm{n}(i-1,j-1,l-1)} \frac{u_{x(i,j,l)} - u_{x(i,j-1,l)}}{\Delta x_{j-1/2}\left(\Delta x_{j-1/2} + \Delta x_{j+1/2}\right)}\right) \\ &+ \left(\begin{array}{l} 2\eta_{xy(i,j,l-1)}\left(\frac{u_{x(i+1,j,l)} - u_{x(i,j,l)}}{\Delta y_{i-1/2}\left(\Delta y_{i-1/2}+\Delta y_{i+1/2}\right)} + \frac{u_{y(i,j+1,l)} - u_{y(i,j,l)}}{\Delta y_{i-1/2}\left(\Delta x_{j-1/2}+\Delta x_{j+1/2}\right)}\right) \\ -2\eta_{xy(i-1,j,l-1)}\left(\frac{u_{x(i,j,l)} - u_{x(i-1,j,l)}}{\Delta y_{i-1/2}\left(\Delta y_{i-3/2}+\Delta y_{i-1/2}\right)} + \frac{u_{y(i-1,j+1,l)} - u_{y(i-1,j,l)}}{\Delta y_{i-1/2}\left(\Delta x_{j-1/2}+\Delta x_{j+1/2}\right)}\right) \end{array}\right) \\ &+ \left(\begin{array}{l} 2\eta_{xz(i-1,j,l)}\left(\frac{u_{x(i,j,l+1)} - u_{x(i,j,l)}}{\Delta z_{l-1/2}\left(\Delta z_{l-1/2}+\Delta z_{l+1/2}\right)} + \frac{u_{z(i,j+1,l)} - u_{z(i,j,l)}}{\Delta z_{l-1/2}\left(\Delta x_{j-1/2}+\Delta x_{j}+1/2\right)}\right) \\ -2\eta_{xz(i-1,j,l-1)}\left(\frac{u_{x(i,j,l)} - u_{x(i,j,l-1)}}{\Delta z_{l-1/2}\left(\Delta z_{l-3/2}+\Delta z_{l-1/2}\right)} + \frac{u_{z(i,j+1,l-1)} - u_{z(i,j,l-1)}}{\Delta z_{l-1/2}\left(\Delta x_{j-1/2}+\Delta x_{j+1/2}\right)}\right) \end{array}\right) \\ &-2\frac{P_{i-1,j,l-1} - P_{i-1,j-1,l-1}}{\Delta x_{j-1/2} + \Delta x_{j+1/2}} \\ &+\frac{1}{4}\left(\rho_{i-1,j,l-1} + \rho_{i,j,l-1} + \rho_{i-1,j,l} + \rho_{i,j,l}\right)g_x = 0 \end{aligned} \tag{8}$$

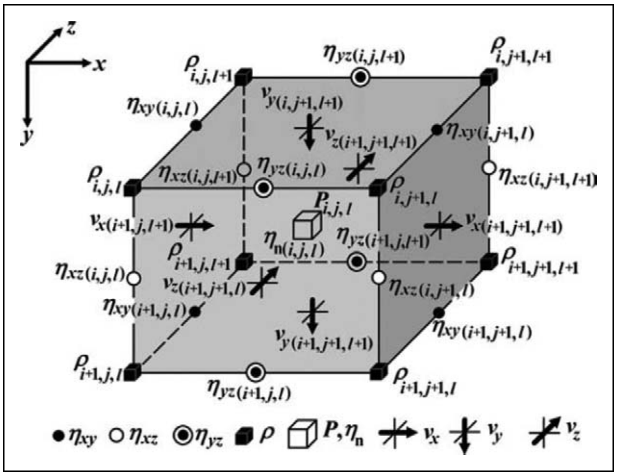


Figure 4. Indexing of different variables for a three-dimensional staggered grid. (Reproduced with permission from Gerya (2010).)

3.2 GPU version

In the next step, we will put forward the GPU implementation. An analysis of the time consumed at each individual part of the original Matlab codes (Figure 5) shows that the three components, namely smoother, restriction, and prolongation, take the lion's share.

To figure out what may happen with a real compiled language, most of the reused functions were translated in the first place into C codes to be called by Matlab with mexfuntion. Table 1 shows the comparison of running time consumption between Matlab codes and its calling C codes using an Intel i7 CPU (3.07 GHz).

One can see from Table 1 that simply rewriting the majority of reused functions in the C language, including smoother, restriction and prolongation components, can accelerate the performance of original Matlab codes, though it's not sufficiently fast enough at current stage. In both of the cases of original Matlab codes and its calling C codes, the smoother takes up most part of the time consumed, which apparently needs optimization.

On GPU, the sequential inner cycle can be modified into parallel threads across all grid points under the SIMT model. A common style of the SIMT model can be written using CUDA as:

```
_global_ void function ( double x, double y, . . . )
{
// calculate the index using the thread id and block id
int i=blockIdx . x;
int j=threadIdx . x;
. . . . . .
}
```

where the instruction in the C_like function we defined is indeed the single instruction to manage many threads running simultaneously to fulfill one function. As what we introduced in the previous section, CUDA organizes the threads in block and grid style at the software level. The thread index can be calculated using the CUDA defined structural variables blockIdx and threadIdx (NVIDIA, 2011). In our implementation, the setting of boundary conditions, restriction and prolongation can apply this SIMT model conveniently. For the smoother, the original CPU based Matlab codes utilize the Gauss–Seidel iterations, in which computing depends on the newest updated neighborhood nodes. It is difficult to implement this strongly inter-dependent algorithms on a GPU by multi-threading stylish, as all of the threads are running simultaneously in a disordered manner. To overcome this hurdle during GPU implementation, the RBGS algorithm is adopted in our application. The RBGS splits the Gauss–Seidel iteration into two parts by red and black colors, allowing the computing point to keep using the newest information associated with it (Figure 6).

Next, we will talk about the workflow of our implementation. As we mentioned before, the pressure is updated with the residuals of continuity equation which requires an initial smoothing loop to compute the residuals on the finest level. During the iteration of the smoother (inner iteration), there are three major procedures: setting boundaries, running the kernels of red and black colors. All of them run on a GPU. One thing that should be noted in advance is that the values on the boundary points have to be imposed ahead of CUDA kernels, for the sake of reducing the logic operations (there is only one logic unit on the SM, which means logic operations may run 16 times or half warp without doing anything). After the initial smoothing loop, we run the main V-cycle loop (outer iteration) containing smoother together with the restriction and prolongation procedures until the tolerance criteria is meet. The workflow can be described as in Algorithm 1.

Matlab index starts from 1 by column-major, while the C language starts from 0 through row-major. Due to the index difference between Matlab and C, macros can be defined to convert the indices. We take vx for example:

Algorithm 1 V-cycle multigrid with RBGS.

```
initialize the density, viscosity and unknowns;
smoother(iternum=1){
for $k = 1$ to $iternum$ do
    set velocity boundaries
    run kernels of red color
    run kernels of black color
end for
compute the residuals
}
while (residual > tolerance) do
for $n = 1$ to $levelnum$ do
    smoother(iternum=iternum(n))
    run restriction
end for
for $n = levelnum$ to 1 do
    smoother(iternum=iternum(n))
    run prolongation
end for
end while
```

```
#define vx( i , j , k) vx [( i –1)+(j –1)* (ynum+1)+(k–1)* (xnum)
    *(ynum+1)]
```

where xnum and ynum represent the number of nodes for each axis. With the macros defined, Matlab codes can be translated into C or CUDA codes in three-dimensional arrays. However, they are literally one-dimensional in nature. The following is an example where we write the CUDA kernel codes for vx with red color:

```
//filename : rb_vx_r .cu
#include ”Index .h”

//_global_ is a key word to define CUDA kernels
_global_ void rb_vx_r( ... )
{
 //obtain the index in 3D using the index of thread
   int i=blockIdx .x;
   int j=blockIdx . y ;
   int k=threadIdx . x ;

 //+2 means starting from 1 and skipping the boundary points
   i+=2;j+=2;k+=2;

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

 //compute the vx
   vx ( i , j ,k) =...;
}
```

The CUDA kernel code rb_vx_r.cu will be compiled into the PTX file rb_vx_r.ptx with nvcc compiler. Then we need to define Kernel Object and the size of block and grid in Matlab. The following is an example using vx in red color:

```
% set the kernel in the main function
global rb_vx_r_kernel ;
% define the kernel object connecting the ptx and cu files we
    have
rb_vx_r_kernel=parallel . gpu .CUDAKernel ( ’ rb_vx_r . ptx
    ’ , ’ rb_vx_r . cu ’ ) ;
% set the size o f each block
rb_vx_r_kernel . Thread Block Size=[znum–1 1 1 ] ;
% set the size o f each grid
rb_vx_r_kernel . Grid Size=[ynum–1 xnum–2] ;
% call the kernel in the smoother function
[ vx , vy , vz , pr ]= feval ( rb_vx_r_kernel , ... ) ;
```

Then we can reconstruct all the functions needed to run on GPU with CUDA and Matlab as what we introduced. The new version of Matlab manages GPU variables using gpuArray to transfer data from the MATLAB workspace. It means that cudaMalloc or cudaMemcpy is no longer needed. However, GPU and CPU use different memory systems that are communicated with one another through PCIE bus. PCIE is very limited in bandwidth compared with GPU. As a result, all of the data have to be operated on GPU throughout the time, including restriction and prolongation at different grid points.

4 Performance analysis

The current simulation runs on a single NVIDIA Tesla 2070C GPU and an Intel Core i7 3.04 GHz CPU with 12 GB memory. The tolerance of average $\|residual\|$ is set at 10^{-5}. Six-level grids are created to use the V-cycle scheme, with the iteration numbers of 10, 20, 40, 80, 160, and 320 for each level. When dealing with a large-scale problem, it would be undesirable to compare the performance between the original Matlab codes and the Matlab calling CUDA codes. On the one hand, the original Matlab codes are so slow that it is hardly in a position to simulate a large-scale problem. On the other hand, GPU is selectively good for simulating a high-throughput problem.

The comparison between the Matlab calling CUDA codes and the Matlab calling C codes is given in Figure 7. We should point out that the deployment of RBGS smoother reduces iterations, which implies that the GPU-based RBGS smoother benefits not only the smoother itself but also the entire cycle. The 256*128*128 model registered a speedup up to a factor of 13.5. When the resolution exceeds 256*256*128, it becomes difficult to record time consumption without GPU computing, suggesting that GPU can be used as an enhancer to raise the resolution of Matlab codes on a simple device at a tolerable level of time consumption

Meanwhile, it seems that the performance improvement does not obey the scaling law from the time consuming aspect. First, the reduced V-cycles do not increase the speedup when the model's resolution is over 128*128*128. This may be caused by the algorithm itself. Most probably, it is because the long-wavelength iterative information propagates faster on coarse grids than on fine grids, which makes it easier to affect the convergence rate given the model's resolution is not so high. Furthermore, the speedup increasing stops at the 128*128*128 model. However, the improvement in smoother's performance does not stop at the 128*128*128 model. To judge the GPU-based smoother's improved performance with different resolutions, we summarized the time consumption by the smoother as shown in Figure 8 It seemed that the speedup increases with an enhanced resolution until the level of 256*256*128, possibly due to the limitation of GPU catch size. In other words, the scale of the problem is a bottleneck that confines the potential performance of a single GPU card, and hence solving a large problem needs an even more powerful multi-GPU system

Evidently, the smoother still takes up most of the running time on the GPU (Figure 9), but restriction and prolongation consume limited time. The reasons why GPU's smoother takes a larger share may be explained that the GPU codes for others parts are more effective than smoother. Because the smoother needs a more complex implementation, somehow we cannot optimize it

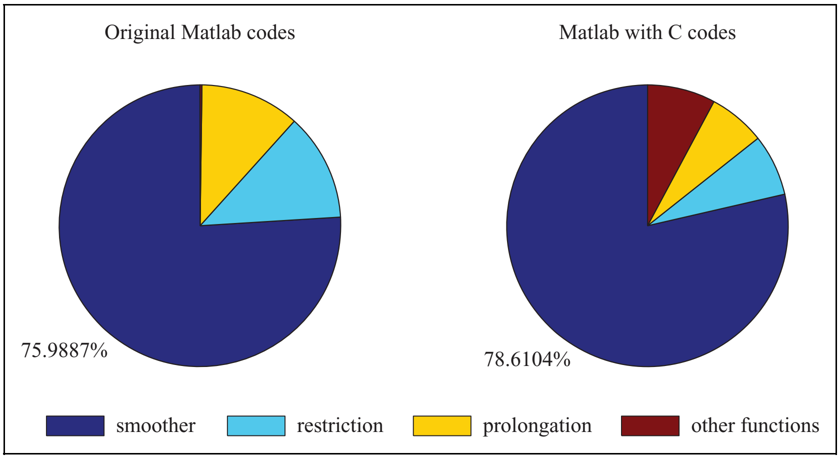


Figure 5. Time consumption analysis for CPU version. For the original Matlab codes, all of the parts are written in Matlab; for the Matlab with C codes, the smoother, restriction, and prolongation parts are rewritten in C.

Table 1. Time consumption of original Matlab and Matlab calling C codes.

Resolution	Original Matlab	Matlab calling C
25*25*25	160 s	15 s
49*49*49	1108 s	125 s

efficiently in the current version. In other words, the speedup of prolongation, restriction, and other parts such as boundary condition setting is better than the smoother itself, also suggesting that even a limited improvement of the smoother's performance would result in a noticeable improvement to the entire codes.

The time consumed by other functionalities that are not run on the GPU saw no change, compared with the modified Matlab codes with C. The important reason may be explained that the improvement of GPU parts makes the weight of other part increase, while the reduced V-cycles makes the weight of other part decrease more than the GPU parts. These two functions may lead to a balance that we cannot see the change of 'other' component's share clearly.

However, the current GPU's (even on Fermi) double precision computing capacity remains unideal, compared with single precision case. In this study, double precision is needed, as the residuals cannot be guaranteed to convergence when using single precision. In Figure 10 where single precision is used, the y-Stokes equation shows a divergence that is possibly caused by the limited word length of single precision. Apparently, a mixed-precision scheme (Furuichi et al., 2011) may solve the problem, as it gives a balanced consideration to both efficiency and precision. In this paper, only the codes with double precision are used.

5 Conclusions

In this paper, a GPU-based MG solver has been proposed to simulate the Stokes flow problem. Time efficiency can be enhanced on a GPU with the parallel RBGS smoother. Matlab's parallel computing toolbox allows user to quickly implement hybrid programming using both script language and CUDA C. In addition to MG method, the Krylov subspace method is often applied to solve the Stokes flow problem. One can accelerate the Krylov subspace-based iterative solver or the preconditioned iterative linear solver using GPU (Bell and Garland, 2012; Li and Saad, 2011). Meanwhile, the GPU-based GMG solver is applicable as a preconditioner of Krylov subspace method for the cubic model (Furuichi et al., 2011; Kameyama et al., 2005). In summary, the hybrid GPU–CPU architecture, such as the combination of MPI and CUDA, is able to enhance the resolution, and can be considered as a useful alternative architecture.

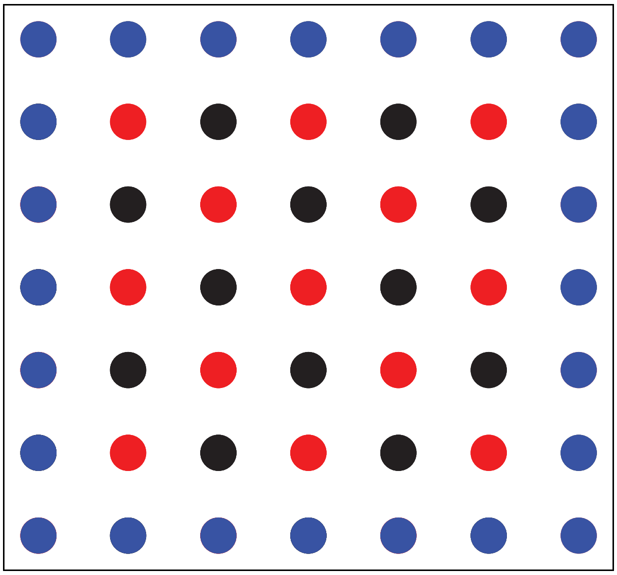


Figure 6. Red–black method: red and black cycles denote red and black nodes, blue cycles denote boundary nodes.

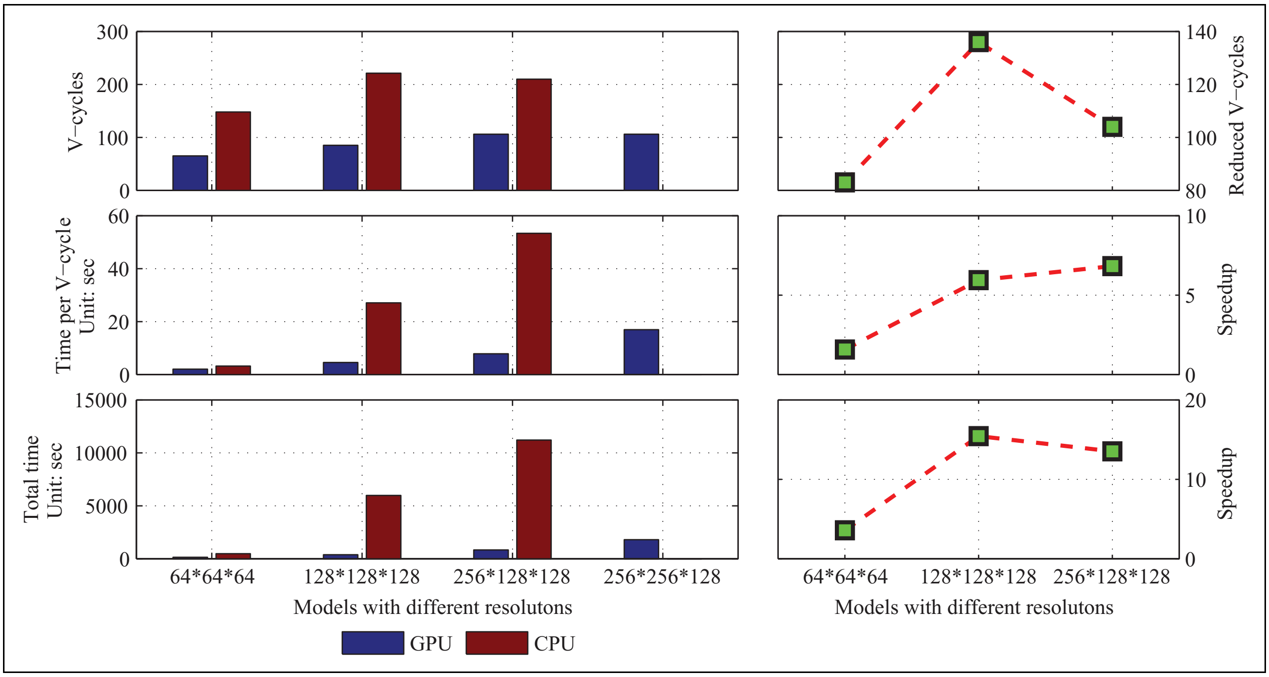


Figure 7. Iterations and time for different models: blue bars denote the Matlab with CUDA codes running on GPU, red bars denote the Matlab with C codes running on CPU. For the model 256*256*128, CPU-based Matlab with C codes are difficult to time.

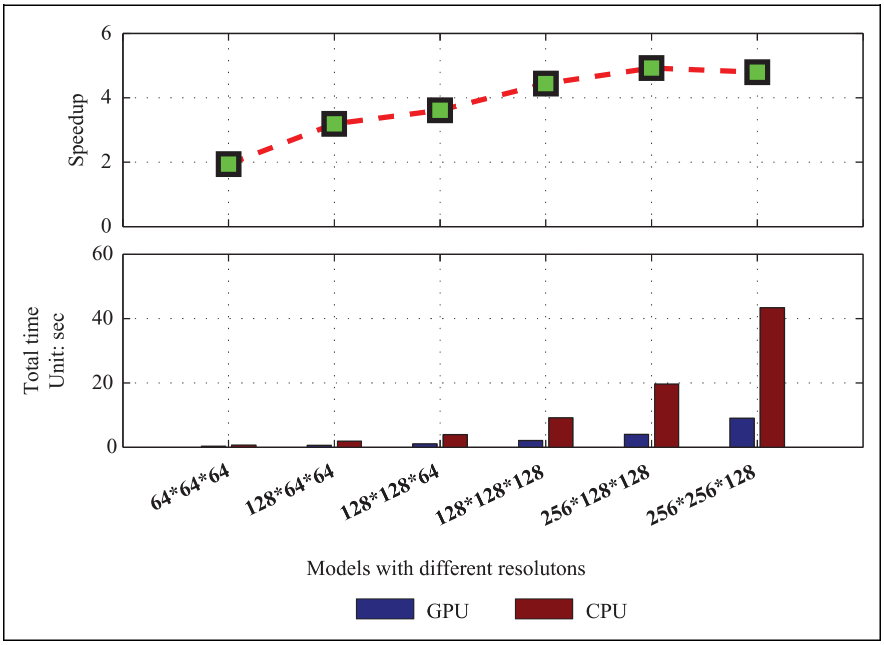


Figure 8. Time and speedup for different smoothers: blue bars denote the Matlab with CUDA codes running on GPU, red bars denote the Matlab with C codes running on CPU. Total time means 10 iterations for each timing.

Funding

This work was supported by the National High Technology Research and Development Program of China (863 Program), ‘Rapid visualization and diagnosis techniques of earth system model output data’ (grant number 2010AA012402), the NSF CMG Program and the Project SinoProbe-07 of China.

Note

1. See http://www.accelereyes.com

Acknowledgements

We would like to thank Masanori C. Kameyama for discussing the comparison between the single and double precision problems. The discussions with Dmitri Karpeev and

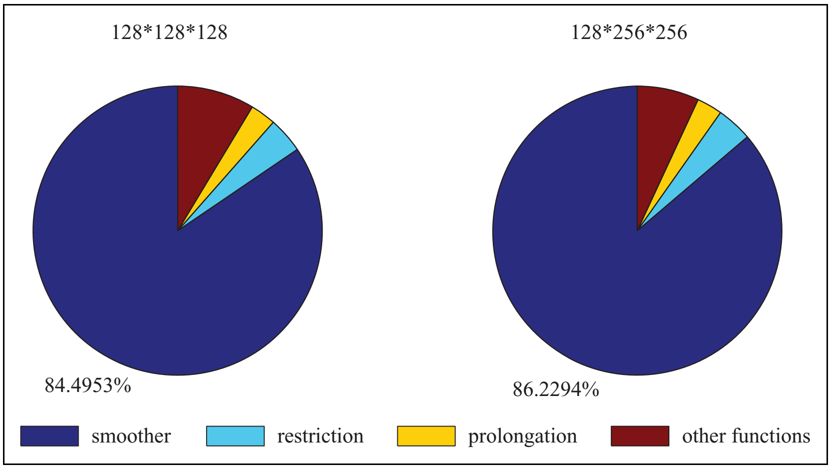


Figure 9. Time consumption analysis for GPU versions with different resolutions. The smoother, restriction, and prolongation parts run on GPU; the other part runs on CPU.

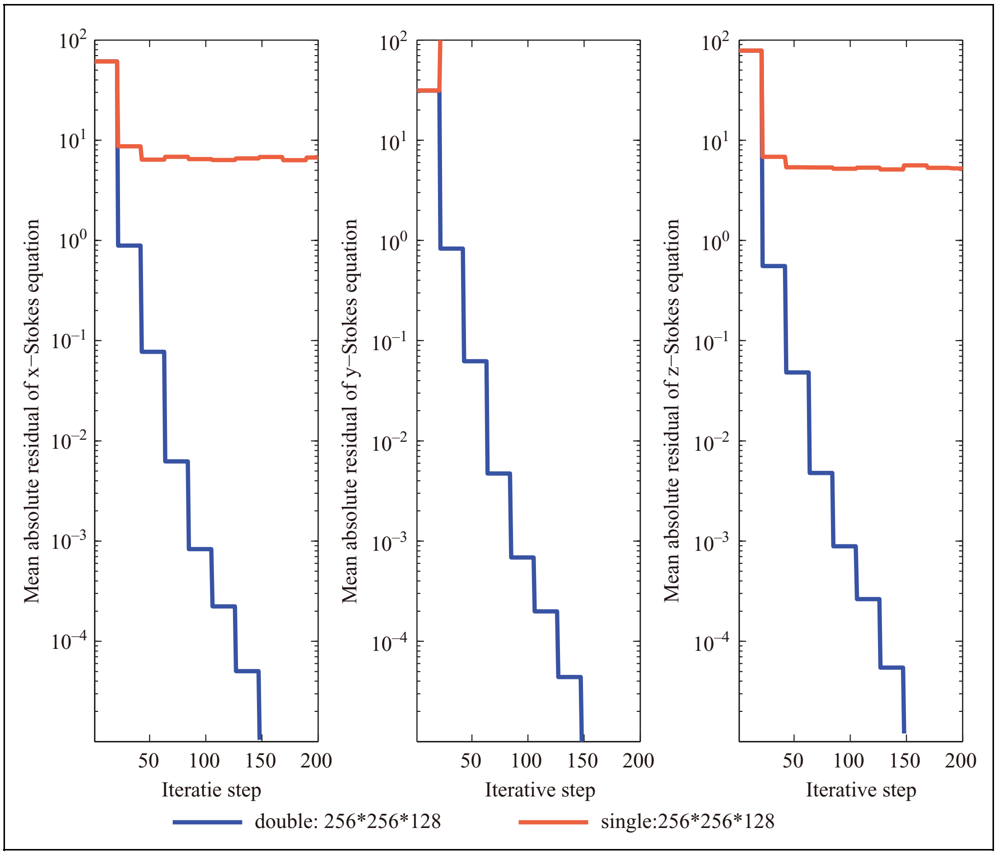


Figure 10. Mean absolute residual of double and single precision for *x*, *y*, *z* direction equations, the iterative step represents the number of V-cycles.

Michael Elgersma are appreciated. We also appreciate the reviewers' advices to this paper.

References

Auth C and Harder H (1999) Multigrid solution of convection problems with strongly variable viscosit. *Geophysical Journal International* 137: 793–804.

Bachvalov N (1966) On the convergence of a relaxation method with natural constraints on the elliptic operator. *Computational Mathematics and Mathematical Physics* 6: 101–135.

Bell N and Garland M (2012) Cusp: Generic parallel algorithms for sparse matrix and graph computations. http://cusp-library.googlecode.com.

Cohen JM and Molemaker MJ (2009) A fast double precision CFD code using CUDA. http://www.jcohen.name/papers/Cohen_Fast_2009_final.pdf.

Deubelbeiss Y and Kaus B (2008) Comparison of Eulerian and Lagrangian numerical techniques for the Stokes equations in the presence of strongly varying viscosity. *Physics of the Earth and Planetary Interiors* 171: 92–111.

Elman H, Silvester D and Wathen A (2005) *Finite elements and fast iteratice solvers with applications to incompressible fluid dynamics*. Oxford University Press.

Fedorenko R (1964) The speed of convergence of one iterative process. *Computational Mathematics and Mathematical Physics* 4: 227–235.

Furuichi M, May DA and Tackley PJ (2011) Development of a Stokes flow solver robust to large viscosity jumps using a Schur complement approach with mixed precision arithmetir. *Journal of Computational Physics* 230: 8835–8851.

Gerya T (2010) *Introduction to numerical Geodynamic modelling*. Cambridge: Cambridge University Press.

Hackbusch W (1977) *On the Convergence of a Multi-grid Iteration Applied to Finite Element Equations*. Report, Universitat Koln, pp. 76–12.

Hackbusch W (1978) On the multi-grid method applied to difference equations. *Computing* 20: 291–306.

Kameyama M, Kageyama A and Sato T (2005) Multigrid iterative algorighm using pseudo-compressibility for three-dimensional mantle convection with strongly variable viscosity. *Journal of Computational Physics* 206: 162–181.

Klöckner A, Pinto N, Lee Y, Catanzaro B, Ivanov P and Fasih A (2009) *PyCUDA: GPU Run-time Code Generation for High-performance Computing*. Technical Report 2009-40, Scientific Computing Group, Brown University, Providence, RI, USA.

Li R and Saad Y (2011) GPU-accelerated preconditioned iterative linear solvers. http://www-users.cs.umn.edu/~saad/PDF/umsi-2010-112.pdf.

Lynch D (2005) *Numerical Partial Differential Equations for Environmental Scientists and Engineers: A Practical First Course*. Berlin: Springer-Verlag.

May D and Moresi L (2008) Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics. *Physics of the Earth and Planetary Interiors* 171: 33–47.

Moresi L, Zhong S and Gurnis M (1996) The accuracy of finite element solutions of Stokes' flow with strongly varying viscosity. *Physics of the Earth and Planetary Interiors* 97: 83–94.

NVIDIA (2011) *NVIDIA CUDA C Programming Guide*. NVIDIA Corporation.

Oosterlee C and Lorenz F (2006) Multigrid methods for the Stokes system. *Computing in Science and Enginering* 8: 34–43.

Patankar S (1980) *Numerical Heat Transfer and Fluid Flow*. New York: McGraw-Hill.

Payne L and Pell W (1960) The Stokes flow problem for a class of axially symmetric bodies. *Journal of Fluid Mechanics* 7: 529–549.

Schubert G, Turcotte DL and Olson P (2001) *Mantle Convection in the Earth and Planets*. Cambridge: Cambridge University Press.

Tackley P (2009) Effects of strongly temperature-dependent viscosity on time-dependent, three-dimensional models of mantle convection. *Geophysical Research Letters* 124: 18–28.

Turcotte DL and Schubert G (2002) *Geodynamics: Applications of Continuum Physics to Geological Problems*, 2nd edn. Cambridge: Cambridge University Press.

Wesseling P (1991) *An Introduction to Multigrid Methods*. New York: John Wiley and Sons.

Zheng L, Gerya T, Knepley M, Yuen DA, Zhang H and Shi Y (2013) GPU implementation of multigrid solver for Stokes equation with strongly variable viscosity. In: Yuen DA, Wang L, Chi X, Johnsson L, Ge W and Shi Y (eds.), *GPU Solutions to Multi-scale Problems in Science and Engineering*. New York: Springer-Verlag, pp. 293–304.

Author biographies

Liang Zheng is a PhD candidate at the University of Chinese Academy of Sciences, working in the field of high-performance computing for solid geophysics and geodynamics. He received his BS in Geomatics from China University of Geosciences and another BS in Computer Science from Huazhong University of Sciences and Technology.

Huai Zhang is a professor of geodynamics working at the University of Chinese Academy of Sciences. His current research fields include solid geophysics and modeling of geodynamics problems. He received his BS and MS from Harbin Institute of Technology in 1995 and 1997, and PhD from Institute of Mathematics, Chinese Academy of Sciences in 2000. He is the vice director of Key Laboratory of Computational Geodynamics, Chinese Academy of Sciences.

Taras Gerya is a professor at Swiss Federal Institute of Technology (ETH-Zurich) working in the field of numerical modeling of geodynamic and planetary processes. He received his undergraduate training in Geology at the Tomsk Polytechnic Institute, his PhD in Petrology at the Moscow State University and his Habilitation in Geodynamics at ETH-Zurich. In 2008 he was awarded the Golden Owl Prize for teaching of continuum mechanics and numerical geodynamic modeling at ETH-Zurich. His present research interests include subduction and collision processes, ridge-transform oceanic spreading patterns, intrusion emplacement into the crust, generation of earthquakes, fluid and melt transport in the lithosphere, precambrian geodynamics and core and surface formation of terrestrial planets. He is the author of *Introduction to Numerical Geodynamic Modeling* (Cambridge University Press, 2010).

Matthew G Knepley received his BS in Physics from Case Western Reserve University in 1994, an MS in Computer Science from the University of Minnesota in 1996, and a PhD in Computer Science from Purdue University in 2000. He was a Research Scientist at Akamai Technologies in 2000 and 2001. Afterwards, he joined the Mathematics and Computer Science department at Argonne National Laboratory (ANL), where he was an Assistant Computational Mathematician, and a Fellow in the Computation Institute at University of Chicago. In 2009, he joined the Computation Institute as a Senior Research Associate. His research focuses on scientific computation, including fast methods, parallel computing, software development, numerical analysis, and multicore architectures. He is an author of the widely used PETSc library for scientific computing from ANL, and is a principal designer of the PyLith library for the solution of dynamic and quasi-static tectonic deformation problems. He was a J. T. Oden Faculty Research Fellow at the Institute for Computation Engineering and Sciences, UT Austin, in 2008, and won the R&D 100 Award in 2009 as part of the PETSc team.

David A Yuen is a professor of geophysics and scientific computation at the University of Minnesota. He graduated from Caltech with a bachelor's degree in physical chemistry in 1969, masters in geophysics from Scripps in 1973 and PhD in geophysics and Space Physics from UCLA in 1978. He received a postdoc at the Department of Physics, University of Toronto to 1979. He was an Assistant Professor and Associate Professor at Arizona State University in 1985. From September 1985 he has worked at the University of Minnesota. He became an AGU Fellow in 2005.

Yaolin Shi is a professor of geophysics at the University of Chinese Academy of Sciences. He received a PhD from UC Berkeley in 1986. He was vice president of the Chinese Geophysical Society (2008–2012) and vice president of the Chinese Seismological Society (2007–2011). He has been a member of the Chinese Academy of Sciences since 2001.