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

Liang Zheng¹,²,³, Huai Zhang¹,², Taras Gerya⁴, Matthew Knepley⁵, David A Yuen³,⁶ and Yaolin Shi¹,²

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: huaiizhang@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 exe-
cuted in parallel through CUDA threads (NVIDIA, 2011).

Developing CUDA applications is still challenging, espe-
cially 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 algo-
rithms. Furthermore, using script language as an interface
to call C and Fortran codes is commonplace in scientific com-
cputation, as it reduces the workload of programming without
compromising the core performance. Fortunately, some
script programming tools, such as Jacket Matlab toolbox1 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 tool-
box 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 instruc-
tion set for CUDA programming, similar to the assemble lan-
guage 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 phe-
nomena, 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 trans-
port 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:

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

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

where \( u_i \) is velocity, \( \sigma_{ij} \) is the deviatoric stress, \( P \) is press-
ure, \( \rho \) 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 condi-
tion) when a numerical scheme is implemented. LBB crit-
eria is a compatibility condition between the velocity
and pressure spaces, which is necessary to yield stable pres-
sure approximations. The constitutive relationship bridging
the velocity \( u_i \) and the deviatoric stress \( \sigma_{ij} \) in Equations (1)
and (2) is defined as

\[
\sigma_{ij} = 2\eta \dot{e}_{ij} = \eta \left( \frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right)
\]  
(3)
in which, \( \dot{e}_{ij} \) is the strain rate and the viscosity \( \eta \) 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 condi-
tions (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
dynamical modeling, the Stokes equations will pro-
duce difficulties numerically due to strongly variable coeffi-
cients, 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

\[
\eta_{\text{eff}} \propto \exp \left( \frac{E_a + V_a P}{RT} \right)
\]  
(4)

where, \( \eta_{\text{eff}} \) 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 itera-
tive 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 stud-
ies (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 coeffi-
cients from finer grids to the coarser one, while prolonga-
tion 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,
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

---

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.
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 applicable 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:

\[
P_{\text{new}} = P + \eta \text{Res}_{\text{continuity}} \text{Res}_{\text{continuity}}
\]

where \( P_{\text{new}} \) is the pressure to be solved, \( P \) is the pressure obtained from the previous iteration, \( \eta \) is the local viscosity, \( \text{Res}_{\text{continuity}} \) is the residual of continuity equation, and \( \text{Res}_{\text{continuity}} \) is the relaxation coefficient of continuity equation.

For the strongly variable viscosity problem, the big contrast of coefficients hinders 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. Together with the gradual increasing in the 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 \( \eta_n, \eta_{xy}, \eta_{xz} \) and \( \eta_{yz} \) corresponding to the components of normal stress and shear stress:

\[
\begin{align*}
\eta_{ij} &= \eta_{\text{computational}} \exp \left[ \frac{\ln (\eta_{\text{computational}})}{\ln (\eta_{\text{max}})} \right] \eta_{ij} \\
\eta_{ij} &= \eta_{\text{original}} \eta_{ij} \\
\end{align*}
\]

where \( \eta_{\text{computational}} \) and \( \eta_{\text{original}} \) represent current and original viscosity, respectively, with \( \eta_{\text{min}} \) and \( \eta_{\text{max}} \) for the minimal and maximal viscosity of the model. \( \eta_{ij} \) 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.
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:

```c
1 global void function ( double x, double y, ... )
2 {
3     // calculate the index using the thread id and block id
4     int i = blockIdx.x;
5     int j = threadIdx.x.x;
6     ... ...
7 }
```

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.**

1: initialize the density, viscosity and unknowns;
2: smoother(ithernum=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: while (residual > tolerance) do
10:    for n = 1 to levelnum do
11:       smoother(ithernum=ithernum(n))
12:     run restriction
13:    end for
14:    for n = levelnum to 1 do
15:       smoother(ithernum=ithernum(n))
16:     run prolongation
17:    end for
18: end while

---
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:

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

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:

```matlab
1 //filename : rb_vx_r.cu
2 #include "Index.h"
3 #include "Index.h"
4 #include "Index.h"
5 #include "Index.h"
6 #include "Index.h"
7 #include "Index.h"
8 #include "Index.h"
9 #include "Index.h"
10 #include "Index.h"
11 #include "Index.h"
12 //set the kernel in the main function
13 set global rb_vx_r_kernel ;
14 % define the kernel object connecting the ptx and cu files we have
15 rb_vx_r_kernel= parallel . gpu .CUDAKernel ( 'rb_vx_r . ptx'
16 , 'rb_vx_r . cu ') ;
17 % set the size of each block
18 rb_vx_r_kernel . Thread Block Size=[xnum-1 ynum-1 znum-1] ;
19 % set the size of each grid
20 rb_vx_r_kernel . Grid Size=[xnum-1 ynum-1 znum-2] ;
21 % call the kernel in the smoother function
22 (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.
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 $\gamma$-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.
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.

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

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

References


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.


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 in Geomatics from China University of Geosciences and another BS in Computer Science from Huazhong University of Sciences and Technology.

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.