Geochemistry Geophysics Geosystems G3
AN ELECTRONIC JOURNAL OF THE EARTH SCIENCES
Published by AGU and the Geochemical Society

Technical Brief
Volume 7, Number 6
1 June 2006
Q06001, doi:10.1029/2005GC001155
ISSN: 1525-2027

GeoFramework: Coupling multiple models of mantle convection within a computational framework

E. Tan and E. Choi
Seismological Laboratory, California Institute of Technology, Pasadena, California 91125, USA (tan2@gps.caltech.edu)

P. Thoutireddy
Center for Advanced Computer Research, California Institute of Technology, Pasadena, California 91125, USA

Now at Parametric Technologies Corporation, 2730 San Tomas Expressway, Santa Clara, California 95051, USA

M. Gurnis
Seismological Laboratory, California Institute of Technology, Pasadena, California 91125, USA

M. Aivazis
Center for Advanced Computer Research, California Institute of Technology, Pasadena, California 91125, USA

[1] Solver coupling can extend the capability of existing modeling software and provide a new venue to address previously intractable problems. A software package has been developed to couple geophysical solvers, demonstrating a method to accurately and efficiently solve multiscale geophysical problems with reengineered software using a computational framework (Pyre). Pyre is a modeling framework capable of handling all aspects of the specification and launching of numerical investigations. We restructured and ported CitcomS, a finite element code for mantle convection, into the Pyre framework. Two CitcomS solvers are coupled to investigate the interaction of a plume at high resolution with global mantle flow at low resolution. A comparison of the coupled models with parameterized models demonstrates the accuracy and efficiency of the coupled models and illustrates the limitations and utility of parameterized models.

Components: 7031 words, 6 figures, 4 tables, 1 animation.

Keywords: GeoFramework; solver coupling; Pyre; CitcomS; CitcomS.py.

Index Terms: 0545 Computational Geophysics: Modeling (4255); 1213 Geodesy and Gravity: Earth's interior: dynamics (1507, 7207, 7208, 8115, 8120); 8121 Tectonophysics: Dynamics: convection currents, and mantle plumes.

Received 30 September 2005; **Revised** 8 March 2006; **Accepted** 31 March 2006; **Published** 1 June 2006.

Tan, E., E. Choi, P. Thoutireddy, M. Gurnis, and M. Aivazis (2006), GeoFramework: Coupling multiple models of mantle convection within a computational framework, *Geochem. Geophys. Geosyst.*, *7*, Q06001, doi:10.1029/2005GC001155.

1. Introduction

[2] Geological processes encompass a broad spectrum of length and timescales, often with different physical processes dominating at either different locations or scales. Traditionally, a modeling code (a solver) is developed for a problem of specific length and timescale, but its utility beyond the original purpose is often limited. Modeling the dynamics of geophysical systems of all relevant scales is challenging with present-day tools. Writing a completely new solver covering such broad temporal and spatial scales is a substantial investment and may be undesirable. Leveraging existing, benchmarked, single-scale solvers and coupling them to solve multiscale problem would be a more viable alternative. The GeoFramework software addresses this need through creating and maintain-

ing a suite of reusable and combinable tools for solid earth problems.

[3] GeoFramework extends Pyre, a Python-based modeling framework. Pyre is originally developed to link solid (Lagrangian) and fluid (Eulerian) solvers, as well as mesh generators, visualization packages, and databases, with one another for engineering applications [*Cummings et al.*, 2002]. Within the Pyre framework, a solver is aware of the presence of other solvers and can interact with each other via exchanging information across adjacent mesh boundaries. Such interaction is termed "solver coupling." There are four advantages of solver coupling for multiscale problems in geophysics:

[4] 1. Natural boundary conditions (BCs): Often BCs are set a priori on only one of the multi-boundaries available (such as sidewalls). Reflecting or periodic BCs can result in unrealistic deformation. However, if a regional solver is coupled with a solver of a larger domain (but of coarser resolution), the deformation field of the later solver can be used as the BCs of the former solver, while the response of the former solver can be fed back to the later solver. Therefore the regional solver can have more natural BCs. Alternatively, an uncoupled model with traditional mesh refinement, i.e., the study area in high resolution and a vast surrounding area in low resolution, can achieve similar goals.

[5] 2. Computational efficiency: The stable time step size is proportional to the smallest grid resolution, linear in hyperbolic equations and quadratic in parabolic equations. In an uncoupled model with mesh refinement, each step can advance in time only by a small amount, dictated by the finest grid. Computation on the coarser grid, which does not require such a small time step, must use the same small time step as the finest grid. In the case of a coupled, multiresolution model, since different solvers can have time steps of different sizes, the coarser-resolution solver can have a larger time step, resulting in a substantial improvement in computational efficiency over traditional mesh refinement.

[6] 3. Multiphysics models: A geophysical process can involve the coupling of a wide suite of physical processes. For example, the mechanism of postseismic deformation can be either elastic or plastic. A solver that can handle all aspect of the relevant physics may not be available, or if available, the code would be complicated and difficult to maintain and develop. On the other hand, the problem can be handled by multiple solvers coupled together, with each solver responsible for fewer physical processes, so that the code for each solver is simple and manageable.

[7] 4. Data assimilation and prediction: Data output by one solver can be seamlessly passed as the input to another solver. For example, the result of a mantle convection model, when converted to seismic velocity with the aid of a mineral physics database, can be fed into a seismological code to generate synthetic seismograms, which can be compared with observation to further improve the convection model.

[8] In geodynamics, one can imagine several examples where solver coupling would have considerable utility. Solver coupling can simulate the interaction between: large-scale and small-scale mantle convection, the viscous mantle and elastic crust, mantle flow and the thermodynamics of mineral phase relations, tectonic stress loading and earthquake rupture, and earthquake rupture and seismic wave propagation. In this paper, we approach the problem of mantle convection interacting at two different length scales. In a companion paper (E. Choi et al., manuscript in preparation, 2006), we will demonstrate the linkage between long-term crustal deformation and mantle convection.

[9] A challenging mantle convection problem is the tilting of a plume conduit in large-scale mantle flow. Hot material rising from a hot thermal boundary layer forms a low viscosity plume conduit. The tilting of a plume conduit has a substantial influence on the location of a hot spot. Global flow models, in which the motions of plumes are parameterized, show that hot spot locations are influenced by large-scale flow [*Steinberger and O'Connell*, 1998]. However, the parameterized model assumes that the presence of plumes does not change the background mantle flow. It also assumes that the motion of a plume conduit can be parameterized by the vector sum of the ambient flow and the rising velocity of the plume conduit, which is inversely proportional to the ambient viscosity and not affected by the presence of the top or bottom boundaries. The validity of these assumptions is unclear and unverified, because of several difficulties. Since the rising velocity of a plume conduit is not easy to measure, the effect of boundaries on the flow is difficult to quantify. On the other hand, numerical calculation of whole mantle flow with sufficient resolution to resolve a plume conduit remains beyond the capability of the

most powerful computers, while numerical calculation of regional models is inadequate because of the missing large-scale flow. This motivates us to apply the Pyre framework to this geophysical problem. Here, we use the interaction of a plume at high resolution with global mantle flow (each computed by an instance of the finite element code, *CitcomS*) as a test case demonstrating the utility of the Pyre framework.

[10] In this paper, we describe the science-neutral Pyre framework and then introduce a new software package that has been developed for coupling geophysical solvers. We then present the results from the plume-global flow coupling. In the appendices, we demonstrate the numerical veracity of the methods.

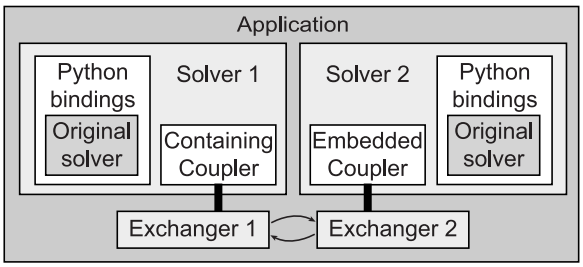

Figure 1. The architecture of a coupled *Application*. The *Application* has two *Solvers*. The original code of the solver (in C/C++/Fortran) is complied into a library, which is called by the Pyre *Solver* via the Python bindings. *Solver 1* is the containing solver and has a *ContainingCoupler*; *Solver 2* is the embedded solver and has an *EmbeddedCoupler*. The *Couplers* communicate via the *Exchangers*, which are external to the *Solvers*.

2. Overview of Pyre

[11] Pyre is a full featured, object-oriented environment that is capable of handling all aspects of the specification and launching of numerical investigations. Pyre operates on massively parallel supercomputers including both shared memory computers and Beowulf clusters. Pyre is written in the Python programming language, an open source, well maintained and widely used interpretive environment.

[12] Pyre leverages the extensibility of the Python interpreter to allow for the seamless integration of rather diverse computational facilities. The framework provides enough flexibility to allow the dynamic discovery of available facilities as part of simulation staging. There is a well defined and well documented method by which a new solver or a new material model can be made available to the framework, while the flexibility allows the user to specify solvers and algorithms in the simulation script, without the need for recompilation or relinking. The combination promotes experimentation with new algorithms by lowering the overall overhead associated with trying out new approaches.

[13] Each simulation model under Pyre is called an *Application*. An *Application* could contain one or more *Solvers*. An *Application* and its *Solver*(*s*) can run on multiple processors, but each processor has only one *Application* and one *Solver* on it (Figure 1). The role of the *Application* is to assign each processor a *Solver* and orchestrate the simulation staging of the *Solver*(*s*), such as initialization (including memory allocation and variable assignment), time marching, and output (Appendix C, section Cl).

3. CitcomS.py

[14] We restructured *CitcomS*, a finite element code for mantle convection in a 3-D full spherical shell [*Zhong et al.*, 2000], and its regional variant (a cut out bounded by lines of constant latitude and longitude) [*Tan et al.*, 2002; *Conrad and Gurnis*, 2003], ported to Pyre, and renamed the code to *CitcomS.py* (available under the GNU General Public License at http://geodynamics.org). The ported version can execute as a stand-alone program, like the old version, or as a *Solver* under a Pyre *Application* (Appendix C, section C2). The later case is a prerequisite of coupled models. The restructuring involves a few top-level functions, leaving the numerical algorithm and internal data structure unchanged.

4. Coupler and Exchanger

[15] To restrict the scope of this paper, we assume that two *Solvers* are coupled in a Pyre *Application* (Figure 1). The domain of one *Solver* is completely immersed within the domain of the other *Solver* (Figure 2a). The former is called the *embedded Solver*, and the later the *containing Solver*. The containing *Solver* has a *ContainingCoupler*, while the embedded *Solver* has an *EmbeddedCoupler*. The interactions between the *Solvers* are simulated by sharing physical quantities (such as velocity, temperature, or traction) on the interfaces, which

has the form of sending and receiving information between *Solvers*. The *Couplers* drive the information exchange and synchronize the *Solvers* (Appendix C, section C3).

[16] The actual information exchange occurs in the *Exchanger* (Figure 1), which consists of a number of C++ classes. The *Exchanger* of a *Solver* can communicate with another *Exchanger* of a different *Solver*. An *Exchanger* is specific to its host *Solver*, but independent from the *Solver* that it is coupled to. The detail and complexity of the coupling mechanism is isolated inside the *Coupler*, leaving the *Exchanger* flexible and extensible. For example, although the *Exchanger* of *CitcomS.py* is developed to couple with another *CitcomS.py*, it can couple with an elastic *Solver* for crustal dynamics problems (E. Choi et al., manuscript in preparation, 2006). Since a goal is to leverage existing modeling code, the *Exchanger* is external to the *Solver* and not required for uncoupled applications.

[17] For simplicity, let us first consider the case of a single-processor *Solver* coupled with another single-processor *Solver*. *Solver A*, which is going to send a message, has an *Outlet*, while *Solver B*, which is going to receive, has an *Inlet*. First, *Solver B* has a *BoundedMesh*. The *BoundedMesh* contains a set of nodes at the interfaces of coupling *Solvers* and maintains a bounding box of those nodes, hence its name. Here we use "interface" in a loose sense. The set can be the whole collection of boundary nodes of an embedded *Solver* (Figure 2b), or only part of it, or the nodes in the overlapping region (Figure 2c). The *Inlet* sends the *BoundedMesh* to the *Outlet*. The *Outlet* uses the bounding box as an efficient check on whether the *BoundedMesh* overlaps with the domain of *Solver A*. The *Outlet* then assembles the requested data (usually by interpolation of a local field variable to the nodes in the *BoundedMesh*) and sends them to the *Inlet*. With finite elements, the interpolation involves finding the corresponding element and computes the shape functions of each node in the *BoundedMesh* and is the most time-consuming procedure. If both meshes are static (Eulerian), this procedure is computed only once and the shape functions are stored for subsequent use. If one of the meshes changes with time (i.e., Lagrangian), this procedure repeats at every time step. The *Inlet* then

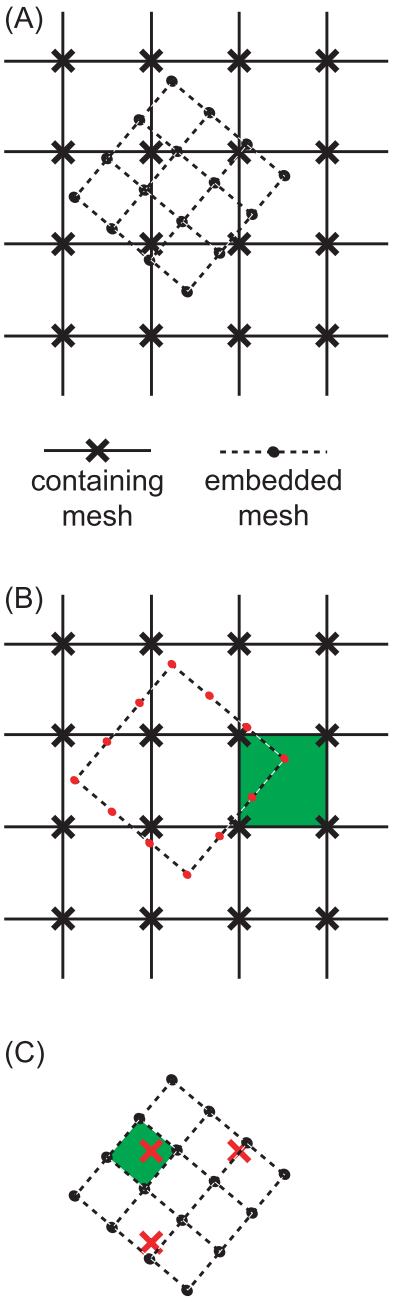

Figure 2. An example of a 2-D embedded mesh and a portion of the containing mesh. (a) The embedded grid is in dashed line, and the containing grid is in solid line. The embedded nodes are shown as dots, and the containing nodes are shown as crosses. The meshes reside on separated *Solvers* but overlap in the modeling space. The embedded mesh is completely immersed within the containing mesh. The two meshes are not required to be parallel to each other. Two scenarios of *BoundedMesh* are presented. (b) The embedded *Solver* is *Solver B*. Its *BoundedMesh* consists of 12 boundary nodes (red). The coordinates of these nodes are sent to *Solver A* (the containing *Solver*) to find the corresponding elements (green, only one element is colored) and shape functions. (c) The containing *Solver* is *Solver B*. Its *BoundedMesh* is the 3 nodes (red) in the overlapping region. One corresponding element is shown in green.

imposes the data received to the interface nodes. Depending on the use of the data, the action of "impose" can have different meanings. If it is used as BCs, the BC arrays are updated; otherwise, it might simply replace the field variable.

[18] In the case of multiprocessor coupling, the procedure becomes more complicated (see function *initialize* in Appendix C, section C3). Each processor still has an *Inlet* (for *Solver B*) or an *Outlet* (for *Solver A*). Additionally, each processor of *Solver A* has a *Source*, but only the leading processor of the *Solver B* has a *Sink*. Each processor of *Solver B* constructs a *BoundedMesh* according to its local mesh. Those local *BoundedMeshes* are broadcasted out by the *Sink* to all *Sources*. Each *Source* passes the received *BoundedMesh* coordinates to the *Outlet*, which performs the same interpolation procedure as the single processor case. The *Outlet* passes the interpolated results to the *Source*. The *Sink* collects the results from all *Sources* in *Solver A* and distributes the collected data to all *Inlets*. *Inlets* then impose these data to the interface nodes. In general, *Solvers* have their own domain decomposition scheme, and the decomposition boundaries of two *Solvers* do not coincide. Therefore nodes in a local *BoundedMesh* of an *Inlet* might be interpolated by different *Outlets*. The *Sources* and *Sink* maintain the bookkeeping of overlapping nodes.

[19] During different stages of a coupled computation, a *Solver* can act as *Solver A* or *Solver B*, i.e., either send or receive data. One advantage of coupled computation is for the containing *Solver* sending the BCs to the embedded *Solver*. If data are sent from the containing *Solver* to the embedded *Solver* only, it is called *one-way communication*. If data are sent from the embedded *Solver* to the containing *Solver* as well, it is called *two-way communication*. For one-way communication, there is no feedback from the embedded *Solver* to the containing *Solver*, and the containing *Solver* nearly executes like a stand-alone computation, but providing BCs to the embedded *Solver*. Only for two-way communication is the response of the embedded *Solver* fed back to the containing *Solver*.

[20] Different *Solvers*, depending on their design, usually have different coordinate systems and units to represent the physical quantities internally. To facilitate information exchange, we require that any quantities be exchanged in Cartesian coordinates and SI units. Conversion from and to the native coordinate system and units is carefully handled within the *Inlet* and *Outlet*. An option of skipping conversion is available if the *Solvers* use the same coordinate system and units.

[21] During a coupled computation, the *Coupler* monitors the model times of both *Solvers* (see function *clip_stable_time_step* in Appendix C, section C3). If the model times of both *Solvers* are equal, they are *synchronized.* For example, in Figure 3, step $M + 3$ and step $N + 1$ are synchronized, but step $M + 2$ and step $N + 1$ are not. Only when the times are synchronized, is the containing *Solver* allowed to march forward to the next time step. Generally, the containing *Solver* has a coarser mesh than the embedded *Solver* and has a larger stable time step. As a result, at the end of a time step, the containing *Solver* will be ahead of the embedded *Solver* (Figure 3a). The containing *Solver* must wait until the embedded *Solver* catches up (Figures 3b and 3c). The embedded *Solver*, if necessary, will clip the size of its stable time step so as to synchronize with the containing *Solver* (Figure 3d).

5. CitcomS-CitcomS Coupling

[22] Having examined the coupling mechanism in general, we now describe the physical quantities exchanged for a specific application of *CitcomS-CitcomS* coupling. For the embedded solver, velocity or traction BCs are required to solve the continuity and momentum equations (see function *solve_velocity* in Appendix C, section C2). We found that imposing three components of velocity as BCs on all boundary nodes leads to poor convergence because of mesh locking [*Hughes*, 2000]. With normal velocity and shear traction imposed as BCs, the stiffness matrix is the same as those of uncoupled problems, and we can find convergent solutions. The embedded *Solver* also needs temperature BC to solve the energy equation. We impose temperature on every boundary node. For the two-way communication, we use the temperature field of the embedded *Solver*, which is more accurate, to override the temperature field of the containing *Solver* (see function *new_time_step* in Appendix C, section C2). The veracity of the coupling method has been extensively tested in a series of benchmarks. The results of the benchmarks are given in Appendices A and B.

6. Model Setup

[23] We will show an example of two *CitcomS.py Solvers* coupled in two-way communication (Figure 4). The embedded *Solver* is a high-

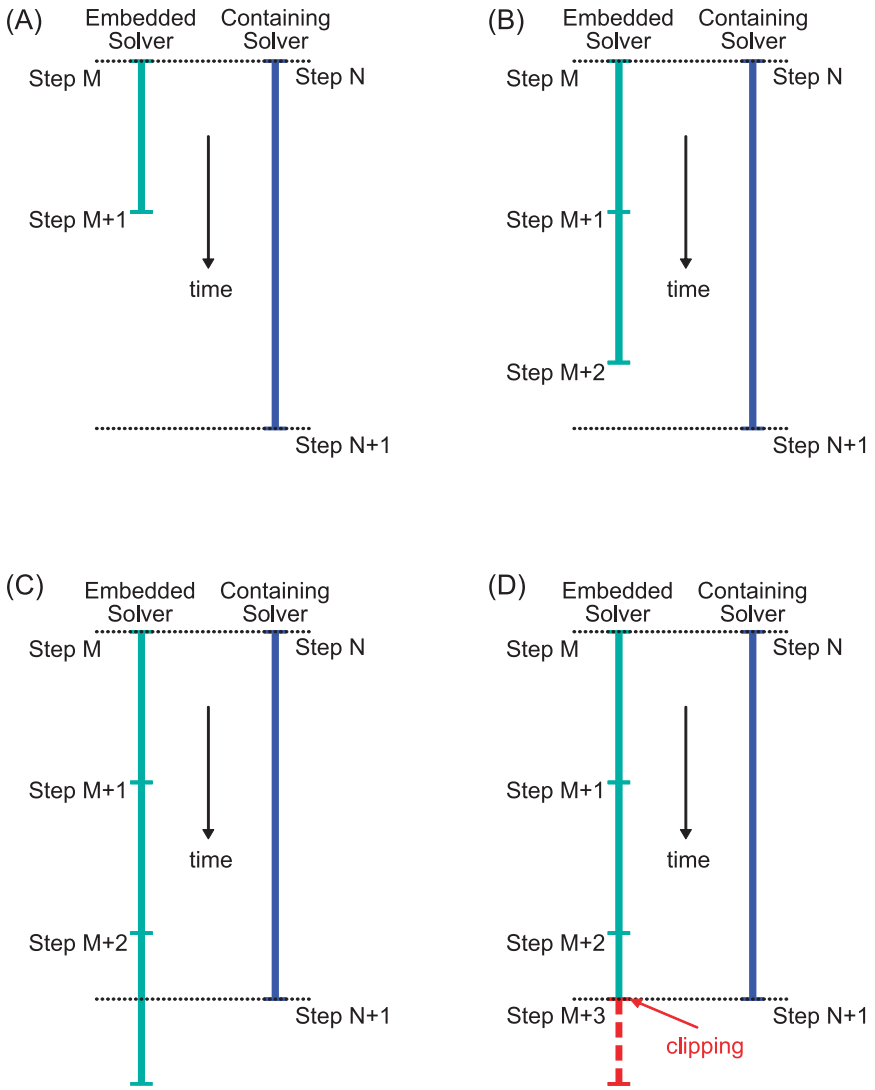

Figure 3. Synchronizing time steps of two *Solvers*. (a) After a synchronized step, the containing *Solver* will be ahead of the embedded *Solver*. (b and c) The embedded *Solver* keeps marching forward. The containing *Solver* waits until the embedded *Solver* catches up. (d) The embedded *Solver* clips the size of its stable time step so as to synchronize with the containing *Solver*.

resolution regional *CitcomS.py*. The containing *Solver* is a global *CitcomS.py*. The resolution of the containing mesh is 180 km horizontally and 40–100 km vertically with mesh refinement near the bottom boundary. The embedded mesh has a resolution of 40 km in each direction and is centered near Hawaii. Both meshes have an inner radius 0.55 and an outer radius 1. The ambient viscosity, η_a?, is 100 for the lithosphere (with the base at 90 km depth), 1 for the upper mantle, and 30 for the lower mantle. The nondimensional viscosity is temperature-dependent according to $\eta = 0.1\eta_a \exp(1.74/(T + 0.5) - 1.74/1.5)$, where T is the temperature. The model has a Rayleigh number 3×10^7. The temperature BC at the core-mantle boundary (CMB) is $T = 0$ except in a small region beneath Hawaii, where a circular region centered at 20°N, 155°W has elevated temperature $T = \exp(-s/s_0)$, where S is the distance to the center, and $s_0 = 750/6371$. The temperature BC at the surface is $T = 0$. The mantle is isothermal ($T = 0$) with a no-slip top surface initially. A plume develops from the heated region and rises vertically. After the plume impinges the surface, we impose plate motion from 80 Ma to the present using the plate motion model of *Lithgow-Bertelloni and Richards* [1998]. The resultant mantle temperature is zero everywhere except the plume. Since two-

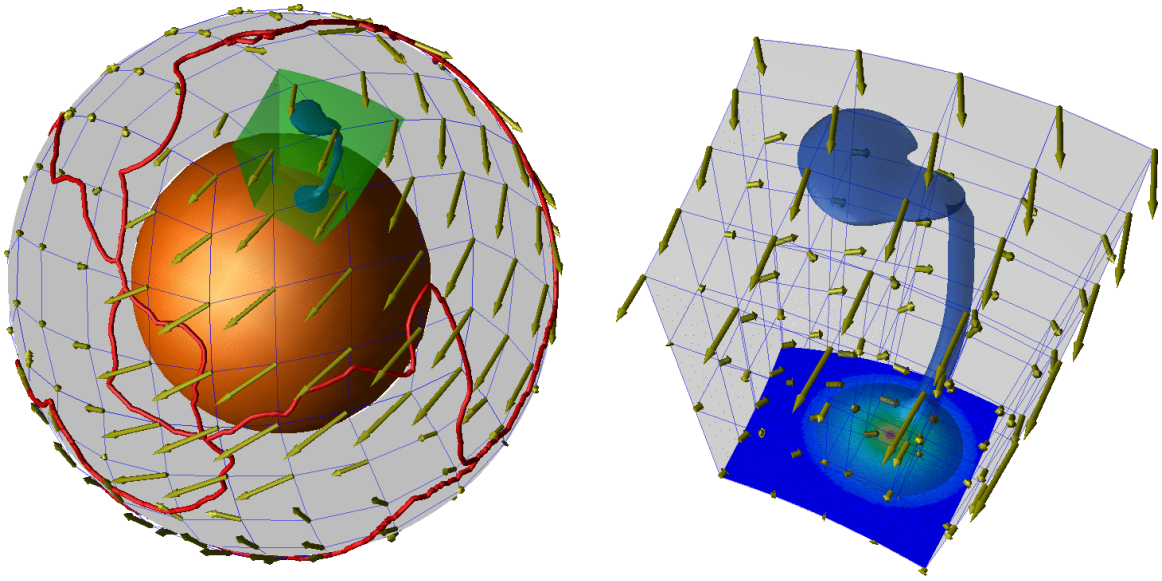

Figure 4. *CitcomS-CitcomS* coupling. (left) The containing *Solver* is a global model, with plate motion imposed on the top surface. The view is centered on western Pacific. Red lines are plate boundaries. Yellow arrows are imposed plate motion. The orange sphere is the core. The small green box is the domain of the embedded *Solver*. (right) The embedded *Solver* is a high-resolution regional model, with boundary conditions retrieved from the containing *Solver*. The black line is the past hot spot location. The red segment is the assumed melt conduit, starting at 160 km depth. The velocity vector is in yellow. The temperature BC at the CMB is shown in color. The plume is visualized as an isosurface (T = 0.08). The numbers of grid points of both meshes are reduced for visualization purposes.

way communication is used, the temperature fields are consistent across the meshes.

[24] We defined the hot spot position by locating the temperature maxima at 160 km depth, implicitly assuming that partial melting occurs at this depth and the melt rises and escapes to the surface through a vertical conduit. At shallower depth, the temperature anomaly tends to be attached to the lithosphere and translates with the plate, which is not representing the motion of the plume.

[25] The two-way communication model is compared with another two models. One model is similar to the two-way communication model, except using one-way communication. The containing mesh in the one-way communication model is driven purely by the plate motion and has no temperature heterogeneity. The comparison between the two-way and one-way communication models will address the influence of the plume buoyancy on the global mantle flow, which, in turn, affects the motion of the plume conduit. The plume only resides in the embedded mesh. Therefore the temperature fields are inconsistent with one another.

[26] Another model is the parameterized model of plume ascent, following the method of *Steinberger and O'Connell* [1998]. The plume conduit starts from a fixed point at 20°N, 155°W and 200 km above the CMB and ends at 180 km below the top surface. The conduit is advected according to the vector sum of the ambient flow and the rising velocity of a plume conduit, $w = w_0/\eta_a$, which is inversely proportional to the ambient viscosity. Since w_0 is difficult to determine from a dynamic model, we use a range of w_0 to find the best fitting model. The ambient flow is taken from an uncoupled global-scale dynamic model. This global flow model is identical to that in the one-way communication model. The comparison between the one-way communication and parameterized models will address the validity of the parameterized plume motion.

7. Results

[27] The evolution of the plume conduit in the two-way communication model is shown in Animation 1. The initial plume conduit is vertical. When the plate motion is imposed at 80 Ma, the plume is swept laterally by the mantle flow. The hot spot progresses to the northwest during 80–74 Ma, then, it progresses to the north until 65 Ma (thick red line in Figure 5). At this stage, the movement of the hot spot is parallel to the plate

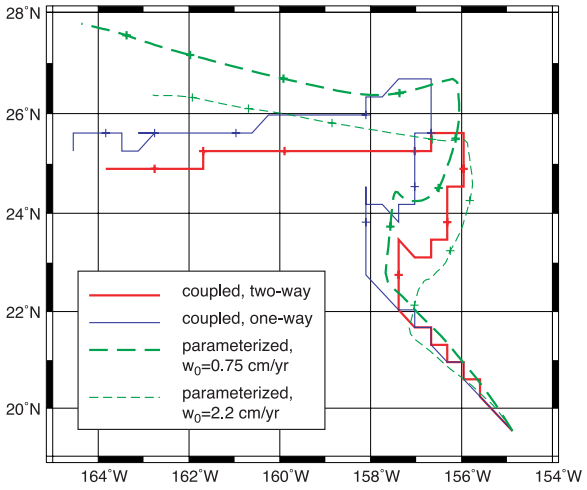

Figure 5. The hot spot locations from different models. Each line represents the motion of the hot spot, starting from the lower right corner, over 80 Myr. A tick (+ symbol) is plotted at intervals of 10 Myr. The zigzagged hot spot locations of the coupled models are artifacts from locating temperature maxima on a discrete grid.

motion, and the plume conduit is tilted toward the northwest too. Between 65–43 Ma, the plate motion is generally to the north, while the hot spot progresses to the northeast. The plume conduit, which was tilted to the northwest, becomes tilted to the north. This readjustment causes the apparent eastward hot spot motion. After 42 Ma, the plate motion changes to the northwest, but with reduced northward component. The readjustment of the plume conduit induces a southward movement to the hot spot. As a result, the hot spot motion becomes westward. At the end of simulation, the hot spot has been displaced 1000 km northwest away from its original location.

[28] The results of the one-way communication model (blue line in Figure 5) are close to the results of two-way communication model. The former model slightly over-estimates the hot spot motion by about 110 km in the 80–74 Ma period, when the plume head has not yet dissipated. After the period, both models agree well if the prior overestimate is removed. At the final step, the separation between the hot spots is about 110 km, and the azimuths to the original location are similar. This suggests that the global flow is not significantly altered by the presence of the plume conduit. The result is not surprising. When the temperature field is interpolated from the embedded mesh and fed back to the containing mesh, the plume conduit is not well resolved by the containing mesh. As a consequence, the temperature anomaly of the plume is weak and perturbs the flow only slightly.

[29] Two parameterized models with different w_0 are shown (dashed green lines in Figure 5). The model with $w_0 = 0.75$ cm/yr best fits with the one-way communication model. Both hot spot locations generally agree with each other during 80–40 Ma, with a separation about 50 km, but diverge after then. At the final step, the separation is about 270 km, and the azimuths are off by about 10°. If we use $w_0 = 2.2$ cm/yr, which is used by *Steinberger and O'Connell* [1998] too, the final hot spot will be off by about 210 km and 10°. On the other hand, this model agrees better with the two-way communication model. We consider this agreement fortuitous, because the models are driven by different global flows. The hot spot separation between the parameterized and coupled models is a metric to the accuracy to the parameterized model. The relative error of the parameterized model, defined as the ratio between the separation to the total displacement of hot spots, is about 20%. We find the accuracy of the parameterized models acceptable, considering its small requirement in computational cost.

[30] From the comparisons, we conclude (1) the plume conduit in our model does not change the plate-driven flow significantly and (2) the parameterized model can approximate the hot spot location with an appropriate w_0. Nevertheless, the choice of w_0 is not self-evident and should best be guided by the coupled model.

[31] The uncoupled global flow model required ≈40 hours of computation on a Beowulf cluster, while the coupled model requires ≈64 hours. With a 60% increase in computation time, we find a solution with a fourfold increase in resolution in the plume region of the coupled model. To achieve the same resolution, an uncoupled model with mesh refinement would have required a time step size 1/4 the size of the coupled case and would have taken ≈256 hours to compute. With such large savings in computational resources, problems that were too expensive can become manageable with solver coupling.

8. Conclusion

[32] As our understanding of Earth's deformation improves, more sophisticated models are needed to explore the deformation process. However, the growing complexity of future models will exceed the capabilities of current generation solvers.

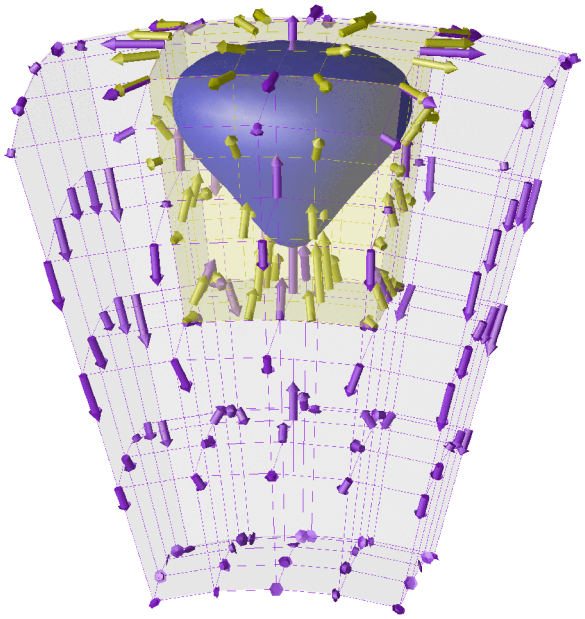

Figure A1. Regional-regional *CitcomS.py* coupling of case a1 at the 100th step. The containing mesh and its velocity vectors are in purple, while the embedded mesh and its velocity vectors are in yellow. The magnitude and direction of the velocity fields are consistent for the two meshes. Also plotted are the temperature iso-surfaces (blue for the containing mesh and green for the embedded mesh) at $T = 0.05$. The two iso-surfaces are so close that only one is visible. The numbers of grid points are reduced for visualization purpose.

Solver coupling can extend the capability of existing solvers with moderate investment. We have developed a software package to couple geophysical solvers and demonstrated the feasibility to solve a multiscale problem efficiently via solver coupling. In a companion paper, we will demonstrate the feasibility to solve multiscale, multiphysics problems using the same technique (E. Choi et al., manuscript in preparation, 2006). We believe that this new technique will provide a new venue to address problems that were too expensive or too complicated to solve before. The software is freely available to the community.

Appendix A: Benchmarks of Regional-Regional CitcomS.py Coupling

[33] Two regional *CitcomS.py Solvers* are coupled. The size of the embedded domain is half of that of the containing domain in each direction (Figure A1). The two domains share the top surface, and the grids are aligned. This configuration does not incur errors when interpolating exchanged data. Therefore any discrepancy in the velocity solutions can be attributed the embedded *Solver*. Moreover, when the element number ratio of the containing to the embedded meshes is 2 (case a1 and a5), every embedded node is collocated with another containing node, so that the solutions on the two meshes should be identical. For the other cases, only portions of the embedded nodes are collocated with the containing nodes. The initial temperature field has a hot, spherical anomaly sitting below the embedded domain. Therefore the initial temperature field in the embedded mesh is 0 everywhere, and the flow inside is purely driven by the BCs. We compare the velocity fields at the 0th time step on the collocated nodes (Table A1). The purpose of this benchmark is to confirm the consistency of the velocity field on both meshes, which is the basic requirement of solve coupling.

[34] We allow the containing *Solver* to execute for 100 time steps and the embedded *Solver* for corresponding time steps. One-way communication is used, i.e., no temperature feedback from the embedded *Solver* to the containing *Solver*. Discrepancy in temperature fields will accumulate over time. Then, we compare the temperature fields at the 100th time step on the collocated nodes (Table A2; Figure A1). Alternatively, if two-way

Table A1. Results of Velocity Fields at the 0th Time Step of Regional-Regional Coupling[a]

Case	No. of Elements (Containing)	No. of Elements (Embedded)	RMS(u)	RMS(d_u)	Discrepancy, %
a1	32	16	87.4848	0.374317	0.4279
a2	32	32	87.4848	0.497431	0.5686
a3	32	48	87.4848	0.532460	0.6087
a4	32	64	87.4848	0.935975	1.0698
a5	64	32	84.2790	0.203668	0.2417

[a] The second column is the number of elements in each direction of the containing mesh. The third column is the number of elements in each direction of the embedded mesh. u is the velocity field in the overlapping region. d_u is the difference in the velocity fields of the two meshes. RMS is the root mean square. Discrepancy is defined as RMS(d_u)/RMS(u).

Table A2. Results of Temperature Fields at the 100th Time Step of Regional-Regional Coupling[a]

Case	No. of Elements (Containing)	No. of Elements (Embedded)	RMS(T) ($\times 10^{-2}$)	RMS (d_T) ($\times 10^{-4}$)	Discrepancy, %	Time Step (Embedded)
a1	32	16	5.0029	1.9009	0.3799	109
a2	32	32	5.0029	5.6182	1.1229	237
a3	32	48	5.0029	5.6161	1.1225	338
a4	32	64	5.0029	7.5698	1.5130	469
a5	64	32	10.133	5.0963	0.5029	113

[a] We only compare the temperature field in the overlapping region. d_T is the difference in the temperature field of the two meshes. Discrepancy is defined as RMS(d_T)/RMS(T). The fifth column is the time step of the embedded mesh when the time step of the containing mesh is 100.

communication is used instead, the temperature fields of both *Solvers* will be consistent after each synchronized time step. Therefore the temperature discrepancy of a two-way communication model will be less than one hundredth of the value in Table A2.

[35] The results in Tables A1 and A2 confirm that the solution on the embedded mesh is consistent to that on the containing mesh. When all nodes are collocated (case a1 and a5), the discrepancy is minimized. Doubling the resolution of both meshes will decrease the discrepancy by half. Refining the resolution in the embedded mesh while keeping the same resolution in the containing mesh increases the discrepancy gradually (case a2, a3, and a4). The consistency achieved is encouraging, considering that grid spacing of the embedded mesh in case a4 is 4 times smaller than that of the containing mesh.

Appendix B: Benchmarks of Full-Regional CitcomS.py Coupling

[36] The containing mesh is a spherical shell extending from an inner radius of 0.5 to an outer radius of 1, and is divided into 12 caps. The side of each cap is ≈55°. The embedded mesh extends from 0°N to 22.5°N, 45°E to 90°E, and 0.75 to 1 in radius. These two meshes share the same top surface. The containing grid is not parallel to the embedded grid. As a result, interpolation error is unavoidable. The grid spacing of the embedded mesh in case b4 is 4.9 times smaller than that of the containing mesh. The initial temperature field has a hot, spherical anomaly sitting below the embedded domain. Therefore the initial temperature field in the embedded mesh is 0 everywhere, and the flow within is purely driven by the BCs. The embedded velocity field is interpolated to the coordinates of the containing nodes. The interpolated velocity field is compared with the containing velocity field at the 0th time step (Table B1).

[37] We allow the containing *Solver* to execute for 70 time steps and the embedded *Solver* for the corresponding time steps. One-way communication is used, i.e., no temperature feedback from the embedded *Solver* to the containing *Solver*. Discrepancy in temperature fields can accumulate over time. The embedded temperature field is interpolated to the coordinates of the containing nodes. The interpolated temperature field is compared with the containing temperature field at the 70th time step (Table B2). The temperature discrepancy of a two-way communication model will

Table B1. Results of Velocity Fields at the 0th Time Step of Full-Regional Coupling[a]

Case	No. of Elements (Containing)	No. of Elements (Embedded)	RMS(u)	RMS(d_u)	Discrepancy, %
b1	32	16	28.3647	0.308470	1.0781
b2	32	32	28.3647	0.346116	1.2165
b3	32	48	28.3647	0.500369	1.7622
b4	32	64	28.3647	0.500971	1.7662
b5	64	32	28.6791	0.145538	0.5075

[a] The second column is the number of elements horizontally in a spherical cap of the containing mesh. The third column is the number of elements along the latitude of the embedded mesh. u is the velocity field in the overlapping region. d_u is the difference in the velocity fields of the two meshes. RMS is the Root Mean Square. Discrepancy is defined as RMS(d_u)/RMS(u).

Table B2. Results of Temperature Fields at the 70th Time Step of Full-Regional Coupling[a]

Case	No. of Elements (Containing)	No. of Elements (Embedded)	RMS(T) ($\times 10^{-2}$)	RMS(d_T) ($\times 10^{-3}$)	Discrepancy, %	Time Step (Embedded)
b2	32	32	4.1186	2.9344	7.1248	165
b3	32	48	4.1499	3.4362	8.2802	230
b4	32	64	4.1662	4.6017	11.045	289
b5	64	32	2.5045	3.3190	13.252	198[b]

[a] We only compare the temperature field in the overlapping region. d_T is the difference in the temperature field of the two meshes. Discrepancy is defined as RMS(d_T)/RMS(T). The fifth column is the time step of the embedded mesh when the time step of the containing mesh is 70. Case b5 is terminated early because the temperature fields diverge too much.
[b] The containing mesh is at 44th time step.

be less than one seventieth of the value in Table B2.

[38] The results in Table B1 and Table B2 show a similar trend as the regional-regional coupling benchmark. When the resolutions of the meshes are similar (case b1 and b5), the discrepancy is minimized. Doubling the resolution of both meshes will decrease the discrepancy by half. Refining the resolution in the embedded mesh while keeping the same resolution in the containing mesh increases the discrepancy gradually (case b2, b3, and b4). The results in Table B2 show large discrepancy due to accumulated interpolation error over time. However, if two-way communication is used, the temperature fields of both meshes are synchronized every time step, so that discrepancy never accumulates.

Appendix C: Code Listing

[39] All of the codes listed are written in Python. For the sake of simplicity, the codes are greatly simplified, so that the definitions of nonessential functions are omitted and some variables are treated global in scope.

C1. Simplified Structure of *Application*

[40] Application:

```
def main():
    # assign a solver to the current processor
    # (definition omitted)
    solver = assign_solver()

    # allocates memory, initializes
    # variables, etc..
    solver.initialize()

    # solve the initial field if necessary
    # (definition omitted)
    solver.solve_0th_time_step()

    # loop until finished
    while True:
        # notify solver to begin a new time
        # step
        solver.new_time_step()

        # determine the size of the time step
        dt = solver.find_stable_time_step()

        # move forward by dt
        solver.solve_next_time_step(dt)

        # notify the solver to end the time
        # step
        # (definition omitted)
        solver.end_time_step()

        # check the end-of-simulation
        # conditions
        # (definition omitted)
        if is_finished():
            break

    # release memory, etc..
    # (definition omitted)
    solver.finalize()
```

C2. Simplified Structure of *CitcomS.py* Solver

[41] CitcomS.py:

```
### Remark: coupler is a "global" variable

def initialize():
    # (definitions omitted)
    allocate_memory()
    init_variables()
    # assign a coupler (either a
    # ContainingCoupler
    # or an EmbeddedCoupler) to the solver
    # (definition omitted)
    coupler = assign_coupler()
```

```
    # initialize the Exchanger package
    coupler.initialize()


def find_stable_time_step():
    # calculate proposed_dt from grid
    # spacing
    # and velocity field (definition omitted)
    proposed_dt = find_local_stable_time_
      step()

    # exchange proposed_dt with another
    # solver,
    # clip it if necessary
    dt = coupler.clip_stable_time_step
      (proposed_dt)

    return dt


def solve_next_time_step(dt):
    # solve the energy equation by a time
    # step
    # of size dt (definition omitted)
    solve_temperature(dt)
    # solve the momentum equation
    solve_velocity()

    # save the result to disk
    # (definition omitted)
    output()


def solve_velocity():
    # EmbeddedCoupler receives velocity
    # BCs
    # from ContainingCoupler and imposes
    # the BC
    scoupler.pre_solve_velocity()
    # solve the Stokes flow problem
    # (definition omitted)
    solve_stokes_ flow()

    # ContainingCoupler sends velocity
    # BCs
    # to EmbeddedCoupler
    coupler.post_ solve_velocity()


def new_time_step():
    # update the temperature field of
    # containing
    # solver if using two-way
    # communication
    coupler.new_time_step()
```

C3. Simplified Structure of *Coupler* for *CitcomS-CitcomS* Coupling

[42] ContainingCoupler:

```
    ### Remark: inlet and outlet are "global"
    ### variables

    def initialize():
      # source will receive the boundary nodes
      # from the sink of EmbeddedCoupler
      source = create_source()

      # outlet will use source to interpolate the
      # data and send the data to the inlet
      # of EmbeddedCoupler
      outlet = create_outlet(source)

      if is_two_ way_communication:
        # overlapped is an instance of
        # BoundedMesh,
        # it contains the overlapped nodes
        # of the
        # containing solver (e.g., Figure 2c)
        overlapped = create_boundedmesh()

        # sink will broadcast the overlapped
        # nodes
        # to the sources of ContainingCoupler
          sink = create_sink(overlapped)

        # inlet will use sink to receive data sent
        # by outlet of ContainingCoupler
        inlet = create_intlet(sink)


def pre_solve_velocity():
    # do nothing
    pass


def post_solve_velocity():
    # send velocity/stress/temperature BCs
    # to EmbeddedCoupler
    outlet.send()


def new_time_step():
    if is_two_way_communication:
      # receive temperature from
      # ContainingCoupler
      inlet.recv()

      # replace the temperature field by the
      # received value
      inlet.impose()
```

```
    def clip_stable_time_step(proposed_dt):
        # exchange the value of dt with
        # EmbeddedCoupler
        ec_dt = exchange_dt(proposed_dt)
        return proposed_dt


EmbeddedCoupler:
    ### Remark: inlet, outlet, is_two_way_
    ### communication
    ### cc_dt, accumulated_dt, and
    ### was_synchronized are
    ### "global" variables


    def initialize():
        # boundary is an instance of
        # BoundedMesh, it
        # contains the boundary nodes of the
        # embedded solver (e.g., Figure 2c)
        boundary = create_boundedmesh()

        # sink will broadcast the boundary nodes
        # to the
        # sources of ContainingCoupler
        sink = create_sink(boundary)

        # inlet will use sink to receive data sent by
        # outlet of ContainingCoupler
        inlet = create_inlet(sink)

        if is_two_way_communication:
            # source will receive the overlapped
            # nodes
            # from the sink of ContainingCoupler
            source = create_source()

            # outlet will use source to interpolate
            # the
            # data and send the data to the inlet
            # of ContainingCoupler
            outlet = create_outlet(source)

        # was synchronized at the previous time
        # step?
        was_synchronized = False

        # dt of ContainingCoupler
        cc_dt = 0

        # dt accumulated since the last
        # synchronized
        # time step
        accumulated_dt = 0


def pre_solve_velocity():
    # receive velocity/stress/temperature BCs
    # from ContainingCoupler when solvers
    # were
    # synchronized at the previous time step,
    # i.e., both solvers march forward in this
    # time step
    if was_synchronized:
        inlet.recv()

    # impose BCs
    inlet.impose()


def post_solve_velocity():
    # do nothing
    pass


def new_time_step():
    if is_two_way_communication and
       is_sync():
        # send temperature to
        # Containing Coupler
        outlet.send()

    # store the sync state of the previous
    # time step
    # in a variable
    if is_sync():
        was_synchronized = True
    else:
        was_synchronized = False


def clip_stable_time_step(proposed_dt):
    if is_sync():
        # exchange the value of dt with
        # ContainingCoupler (definition
        # omitted)
        cc_dt = exchange_dt(proposed_dt)

        # reset the time
        accumulated_dt = 0

    # accumulate the time
    accumulated_dt + = dt

    # if after accumulation, the time exceeds
    # that
    # of ContainingCoupler, clip the time
    if accumulated_dt > cc_dt:
        # clip proposed_dt
        dt = proposed_dt – (accumulated_
             dt - cc_dt)
```

```
            accumulated_dt = cc_dt
        else:
            dt = proposed_dt

        return dt


def is_sync():
    if accumulated_dt = = cc_dt:
        return True
    else:
        return False
```

Acknowledgments

[43] This work has been funded by the NSF ITR program under grant number EAR-0205653. Additional support for E. Tan is provided by EAR-0215644. We thank Craig O'Neil for providing the plate motion model. This work represents contribution 9139 of the Division of Geological and Planetary Sciences, California Institute of Technology.

References

Conrad, C. P., and M. Gurnis (2003), Seismic tomography, surface uplift, and the breakup of Gondwanaland: Integrating mantle convection backwards in time, *Geochem. Geophys. Geosyst.*, *4*(3), 1031, doi:10.1029/2001GC000299.

Cummings, J., M. Aivazis, R. Samtaney, R. Radovitzky, S. Mauch, and D. Meiron (2002), A virtual test facility for the simulation of dynamic response in materials, *J. Supercomput.*, *23*(1), 39–50.

Hughes, T. J. R. (2000), *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*, 682 pp., Dover, Mineola, N. Y.

Lithgow-Bertelloni, C., and M. A. Richards (1998), The dynamics of Cenozoic and Mesozoic plate motions, *Rev. Geophys.*, *36*(1), 27–78.

Steinberger, B., and R. J. O'Connell (1998), Advection of plumes in mantle flow: Implications for hotspot motion, mantle viscosity and plume distribution, *Geophys. J. Int.*, *132*(2), 412–434.

Tan, E., M. Gurnis, and L. Han (2002), Slabs in the lower mantle and their modulation of plume formation, *Geochem. Geophys. Geosyst.*, *3*(11), 1067, doi:10.1029/2001GC000238.

Zhong, S., M. T. Zuber, L. Moresi, and M. Gurnis (2000), Role of temperature-dependent viscosity and surface plates in spherical shell models of mantle convection, *J. Geophys. Res.*, *105*(B5), 11,063–11,082.