V European Conference on Computational Fluid Dynamics
ECCOMAS CFD 2010
J. C. F. Pereira and A. Sequeira (Eds)
Lisbon, Portugal,14-17 June 2010

GENERIC FINITE ELEMENT PROGRAMMING FOR MASSIVELY PARALLEL FLOW SIMULATIONS

Timo Heister*, Martin Kronbichler‡, Wolfgang Bangerth†

*NAM, University of Göttingen
Lotzestrasse 16-18, 37083 Göttingen, Germany
e-mail: heister@math.uni-goettingen.de

‡Uppsala University
e-mail: martin.kronbichler@it.uu.se

†Department of Mathematics, Texas A&M University
e-mail: bangerth@math.tamu.edu

Key words: Finite Element Software, Parallel Algorithms, Massive Parallel Scalability

Abstract. *We present design issues, data structures and algorithms which allow us to leverage the power of high performance computer clusters in generic finite element codes. Large scale flow simulations require parallel algorithms to scale to thousands of processor cores.*

Simple parallelization in generic finite element codes often reduces to implementing parallel linear algebra and solvers or interfacing with an existing parallel linear algebra library. Other data structures and algorithms, e.g. mesh handling, are not designed for parallel computations and introduce bad scaling and memory constraints. Scaling to more than about one hundred cores and a few million of unknowns is not feasible. Today's computer clusters have up to tens of thousands of cores and are the foundation to deal with large scale numerical problems. Generic finite element codes, like `deal.II` *[4, 5], feature methods for rapid and flexible development, higher order elements and adaptivity. To take advantage of those features in large parallel computer clusters, e.g. in flow simulations, many parts of the library have to be adapted and tailored to scale.*

We describe the steps done in enabling `deal.II` *to scale from less than a hundred cores and a few million unknowns up to thousands of cores and billions of unknowns: data structures for distributed fully-adaptive mesh handling, data structures for efficient indexing of degrees of freedom, algorithms for efficiently distributing the degrees of freedom, and more. Distributing storage of data and local – instead of global – communication plays a key role in this process. The parallelization is done in a general setting applicable to any generic finite element library, c.f. [3]. We show numerical results that demonstrate the applicability of our design to generic finite element problems, and that our algorithms scale very efficiently to billions of unknowns.*

1 INTRODUCTION

Today's computer clusters have up to tens of thousands of cores and are the foundation to deal with large scale numerical problems such as simulating complex problems in fluid mechanics. Computational software must be designed to efficiently use the parallel hardware in order to address these problems.

There is a huge gap between highly specialized codes designed to run on those huge machines and general finite element codes. The former are mostly hand-tailored to the numerical problem to be solved and often only feature methods of low order time and spatial discretization. General finite element codes like `deal.II`[4, 5] do not scale to larger clusters but provide many features: higher order finite elements, mesh adaptivity, flexible coupling of different elements and equations, and more.

To close this gap we are working on enhancing the parallel scalability of `deal.II` while maintaining the advanced numerical features listed above. The goal is to do numerical simulations with `deal.II` on massively parallel machines with a distributed memory architecture.

While we describe our progress with `deal.II` in [3], the algorithms outlined here are applicable to any generic finite element code. The modifications discussed in this paper are still in progress but will become available as open source in the near future.

In Section 2 we describe the data structures and algorithms for the parallelization of finite element software. In Section 3 we describe our model problem simulating thermal convection in the earth's mantle[11]. We conclude with numerical results in Section 4 for a simple Poisson equation which shows the parallel scalability and results for the more involved mantle convection problem.

2 MASSIVELY PARALLEL FINITE ELEMENT SOFTWARE DESIGN

For finite element simulations to scale to a large number of processors, one needs linear scalability of the compute time with respect to the number of processors and the problem size. Additionally, the local memory consumption must not increase in step with the problem size. The former requires avoiding global communication and to use only algorithms that scale with problem size. The latter requires the distribution of data structures to store only information that is needed locally.

The primary bottlenecks to parallel scalability in finite element codes are the mesh handling, the distribution and numbering of the degrees of freedom, and the numerical linear algebra. After giving data structures and algorithms for these three components, we summarize the necessary steps for the complete parallelization. We focus on the central aspects of parallelization, and refer to [3] for the technical details.

2.1 Distributed adaptive meshing

The current algorithmic design of `deal.II` (and most other generic finite element libraries if they support distributed parallel computing at all) duplicates the computational

mesh on each computational node. processor. This approach is algorithmically straightforward: all complex data structures are replicated on all nodes, but simpler ones, like the linear system, are stored in a distributed way. This approach is not feasible for massively parallel computations because the generic description of a mesh involves a significant amount of data for each cell which must then be replicated on each processor. This results in a huge and unnecessary memory overhead on each machine.

Most of the memory used to replicate the mesh is wasted as each processor only needs to access a small subset of all cells. We will say that the processor "owns" these cells. In addition, a processor also needs to store all cells touching the cells it owns, but that are in fact owned by neighboring machines. We call the neighboring cells ghost cells. The information about the ghost cells is needed for several reasons, the most obvious of which is because continuous finite elements share degrees of freedom on the lines and vertices. Ghost cells are also needed for adaptive refinement, error estimation, and more.

In adaptive finite elements there are not only the active cells on which the computation is done (as discussed above), but also the hierarchy of the refinement step from a given base mesh in the h-adaptive case. Our data structure distinguishes three different kinds of information for mesh storage:

1. The coarse mesh. This consists of a number of coarse cells describing the problem domain. We assume that the coarse mesh only consists of a relatively small number of cells (up to maybe a tens of thousands) compared to the number of active cells in the parallel computation (which can number in the billions). The refinement process starts with the coarse mesh.

2. Refinement information. This can be stored in a (sparse) octree (a quadtree in two spatial dimensions) of refinement flags for each coarse cell. Each flag either states that this cells has been refined into eight (four in two dimensions) children or that it is an active cell if not.

3. Active cells. This is the set of local cells discussed above, on which the finite element calculation is done. There is a lot of information attached to each cell: vertex coordinates, connectivity information to faces, lines, corners, and neighboring cells, material indicators, boundary indicators, etc. We include the ghost cells in this set; the only difference is that they are flagged to belong to a different machine.

Storing the coarse mesh (1.) on each machine is not a problem, and it simplifies the algorithms enormously. For the refinement information (2.) we interface to an external library called `p4est`[7]. This library handles the abstract collection of octrees describing the refinement from the coarse mesh and handles coarsening, refinement and distribution of the cells between the machines. It also allows queries about the local mesh and the ghost layer. Internally `p4est` enumerates all terminal cells in the collection of octrees with a space filling curve. This allows rapid operations and scalability to billions of cells.

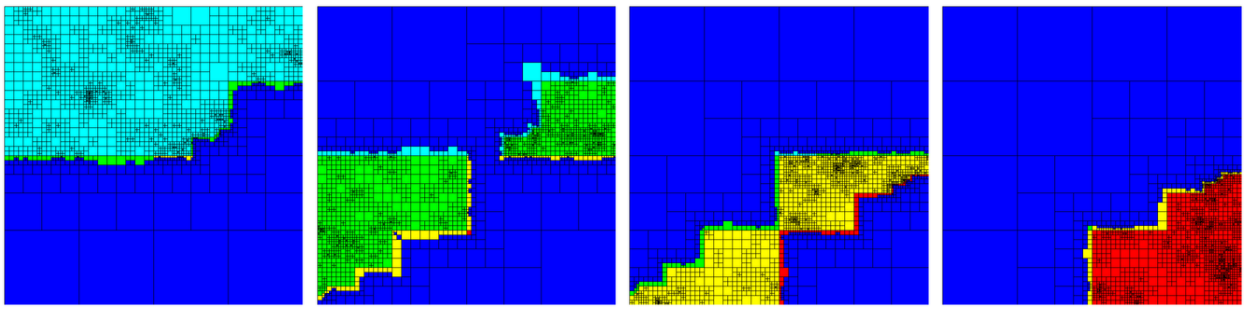

Figure 1: Example of a distributed mesh from the viewpoint of four different machines. Colors indicate ownership of a cell. Dark blue denotes cells that are inactive on a given machine. Note the ghost layer of cells owned by other processors.

`deal.II` recreates the active cells with the ghost layer using the information supplied by `p`4est.

Figure 1 shows an example mesh structure as it is seen on four participating machines. Because we chose to recreate a local triangulation on each machine, most other parts of the finite element library do not need to be changed. The global numbering of the degrees of freedom is the major exception and is discussed in the next subsection.

2.2 Handling degrees of freedom

Finite element simulation requires a global numbering of all degrees of freedom. This is much more complicated to achieve fully in parallel; it is much easier when the complete mesh information is available on each machine. We developed an algorithm with mostly local communication in [3]. The main steps are

1. Local numbering of the degrees of freedom on locally owned cells.
2. A decision process for ownership of degrees of freedom on the interface between machines.
3. Ensuring a globally unique numbering.
4. Local communication of the indices of degrees of freedom on interfaces and within the ghost layer to neighboring machines in an efficient manner.

2.3 Linear algebra

With a global numbering of the degrees of freedom all that remains to perform parallel finite element calculations is the distributed storage of the linear system. There are existing, extensively tested, and widely used libraries like PETSC[1, 2] and Trilinos[10, 9]. PETSC and Trilinos supply row-wise distributed matrices, vectors and algorithms, like iterative Krylov solvers, and also preconditioners. `deal.II`, and most other finite element libraries, have interfaces to these libraries and through them they gain the ability to solve

large linear systems on parallel computers. We have tested both PETSc and Trilinos solvers up to many thousand cores and have obtained excellent scaling results (see below).

2.4 Summary of the algorithm

We summarize the overall structure of a parallel adaptive finite element computation:

1. Rebuild the local mesh on each machine from the refinement information.
2. Distribute degrees of freedom and produce a globally unique enumeration.
3. Assemble the linear system. This is done by looping only over the local cells. Entries that are written to matrix rows owned by a different processor are automatically shipped by the linear algebra package. Everything else in the assembly process remains unchanged compared to serial programs.
4. The linear system is solved with iterative Krylov solvers using preconditioners tailored to the respective problem.
5. An error estimator is run on the local part of the mesh (for example measuring the jumps in the gradients) and all machines agree on cells to mark for refinement and coarsening. Efficient algorithms exist for most refinement strategies such as when a fixed fraction of *global* cells is to be refined[3].
6. The mesh – modified according to the flags – is repartitioned between the machines and the solution is transferred from the previous owner of a cell to its new owner.
7. The adaptive loop continues with the new mesh by returning to step 1.

In an actual implementation, some technical details must be addressed:

1. Hanging Nodes. Adaptive refinement with quadrilaterals or hexahedra must handle cases of T-junctions inside the mesh on the interface between two differently refined cells. Here the additional degrees of freedom need to be constrained algebraically. While this is relatively straightforward in the h-refinement context in sequential computations, the necessary algorithms are not trivial if each processor only knows a fraction of the global mesh.
2. Local Indexing. We need an efficient data structure to describe subsets of the set of all degrees of freedom. In particular, we need efficient search and indexing operations in this data structure while maintaining a compact representation.
3. The solution transfer. Moving a finite element solution from the old to the new mesh in the distributed case involves shipping data between machines. When using thousands of processors or more, this can only efficiently be done using point-to-point communication, without global communication steps.

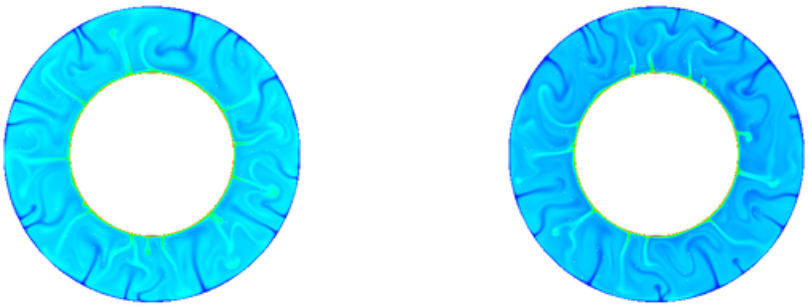

Figure 2: Two time steps in the evolution of turbulent mixing in a simulation of the Earth's mantle.

4. For finite element systems with several components (like velocity, pressure and temperature as used in Section 3) the global indices must be sorted by vector components, even if different kinds of elements (e.g. quadratic elements for the velocity, but linear ones for the pressure) are used in each component.

3 APPLICATION TO MANTLE CONVECTION

Let us briefly outline the physical basis of our test problem, the simulation of fluid flow in the Earth's mantle due to thermal convection. More details and motivation for the discretization and solver choices are given in [11].

In the Earth's mantle, fluid flow is strongly dominated by viscous stresses and driven (among other factors) by temperature differences in the material, while inertia is negligible at realistic velocities of a few centimeters per year. Thus the buoyancy-driven flow can be described by the Boussinesq approximation:

$$\begin{aligned} -\nabla\cdot(2\eta\varepsilon(\mathbf{u}))+\nabla p &= -\rho\,\beta\,T\,\mathbf{g}, \\ \nabla\cdot\mathbf{u} &= 0, \\ \frac{\partial T}{\partial t}+\mathbf{u}\cdot\nabla T-\nabla\cdot\kappa\nabla T &= \gamma. \end{aligned} \tag{1}$$

Here, $\mathbf{u}, p, T$ denote the three unknowns of velocity, pressure, and temperature in the Earth's mantle. The first two equations form a Stokes system for velocity and pressure with a forcing term on the right hand side stemming from the buoyancy of the fluid through the temperature T. The third equation is of advection–diffusion type, describing how the temperature is advected by the velocity field.

Let η be the viscosity of the fluid and κ the diffusivity coefficient for the temperature (both assumed to be constant here for simplicity), $\varepsilon(u) = \frac{1}{2}(\nabla u + (\nabla u)^t)$ denotes the symmetrized gradient, ρ is the density, β the thermal expansion coefficient, g the gravity vector, and γ describes the external heat sources. Fig. 2 shows two snapshots from the evolution of the turbulent mixing within the Earth's mantle due to the heating from below and cooling from above.

The temperature equation is solved semi-explicitly: the diffusion is treated implicitly, while the advection uses an extrapolated velocity from the last two time steps. For the time discretization of the temperature equation we choose the backward differentiation formula of second order (BDF2). The Stokes part is stationary and solved separately. We use the inf-sup stable finite element pair Q_{k+1}^d-Q_k, $k > 1$ for the Stokes part and discretize the temperature with continuous Q_{k+1} elements. We use an artificial viscosity model to stabilize the temperature. The nonlinearity in this stabilization is handled explicitly.

The discretized Stokes problem forms a saddle point system

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} f \\ g \end{pmatrix} \tag{2}$$

which is preconditioned with an operator P^{-1} of block triangular type:

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} P^{-1} \begin{pmatrix} v \\ q \end{pmatrix} = F \quad \text{with} \quad P^{-1} = \begin{pmatrix} \widetilde{A} & B^T \\ 0 & \widetilde{S} \end{pmatrix}^{-1}.$$

Here approximations $\widetilde{A}^{-1} \approx A^{-1}$ and $\widetilde{S}^{-1} \approx S^{-1}$ for the Schur complement $S := -BA^{-1}B^T$ are used. This choice is motivated by the observation that with exact evaluations of A^{-1} and S^{-1}, the number of outer (F)GMRES steps is at most two[6]. This preconditioner can be written as

$$P^{-1} = \begin{pmatrix} \widetilde{A}^{-1} & -\widetilde{A}^{-1}B^T\widetilde{S}^{-1} \\ 0 & \widetilde{S}^{-1} \end{pmatrix} = \begin{pmatrix} \widetilde{A}^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & B^T \\ 0 & -I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -\widetilde{S}^{-1} \end{pmatrix}.$$

Consequently, each application of the preconditioner requires the solution of two inner problems[8]: the applications of $\widetilde{A}^{-1}$ and $\widetilde{S}^{-1}$, and there is one matrix-vector product with the matrix B^T. For the numerical results in section 4, $\widetilde{A}^{-1}$ is approximated with a BiCGStab solver preconditioned with the algebraic multigrid preconditioner ML from Trilinos and $\widetilde{S}^{-1}$ is approximated by a mass matrix in the pressure space solved by an ILU-preconditioned CG method.

4 NUMERICAL RESULTS

4.1 A simple testcase

We first present some results to demonstrate the scalability of the different parts implemented in the library. In Figure 3 we show the weak scalability from 8 to 1000 processors with about 500,000 degrees of freedom per processor. We measure timings for different parts and average memory consumption on each machine. The problem is a Poisson problem on a three dimensional unit cube with regular refinement (no adaptivity). All parts but the solver (BiCGStab preconditioned with an algebraic multigrid) scale linearly. Figure 4 shows individual iterations in a fully adaptive refinement loop for a two dimensional Poisson equation on 1024 processors (left). The problem size increases over several

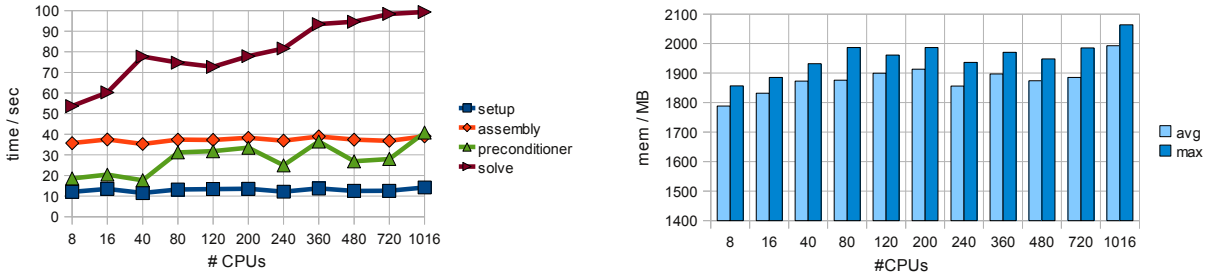

Figure 3: 3D Poisson Problem, regular refinement. Left: weak scaling up to 1016 processors. Right: average peak memory for the same data.

refinement cycles from 1.5 million to 1.5 billion degrees of freedom. On the right you see the strong scalability starting from 256 and going to 4096 processors for the same cycle with a fixed problem size within the adaptive iteration loop.

Both results show the excellent scalability with respect to problem size and number of processors. The memory consumption stays nearly constant even when increasing the problem size by over a factor of one hundred.

4.2 Results for the mantle convection problem

Finally we show some early numerical results for the mantle convection example outlined in section 3. The domain is a two dimensional shell modeling the earth. In figure 5 we present timing of seven adaptive refinement steps for a single fixed time step. We observe good scalability, although the construction of the preconditioner and the solver itself do not scale linearly. Surprisingly, the compute time spent in these parts of the program does increase sub-linearly with the number of degrees of freedom; this is currently under investigation.

5 CONCLUSIONS

We have presented a general framework for massively parallel finite element simulation. The early results shown here are convincing, showing that even complex problem with more than a billion unknowns can be solved on a large cluster of machines. The

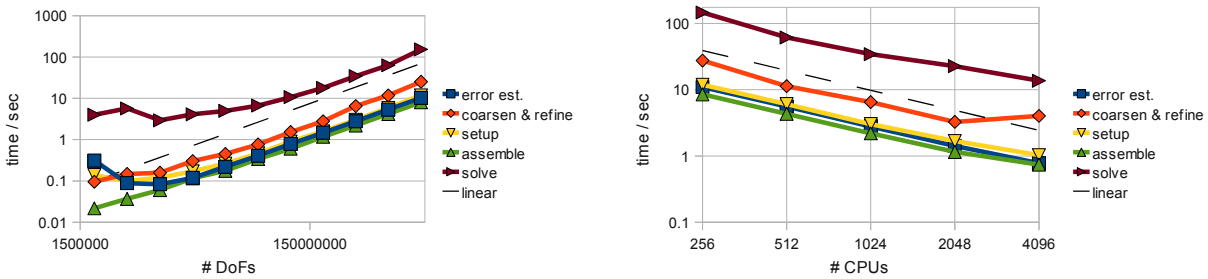

Figure 4: 2D Poisson Problem, fully adaptive. Left: weak scaling on 1024 processors. Right: strong scaling, problem size of around 400 million DoFs.

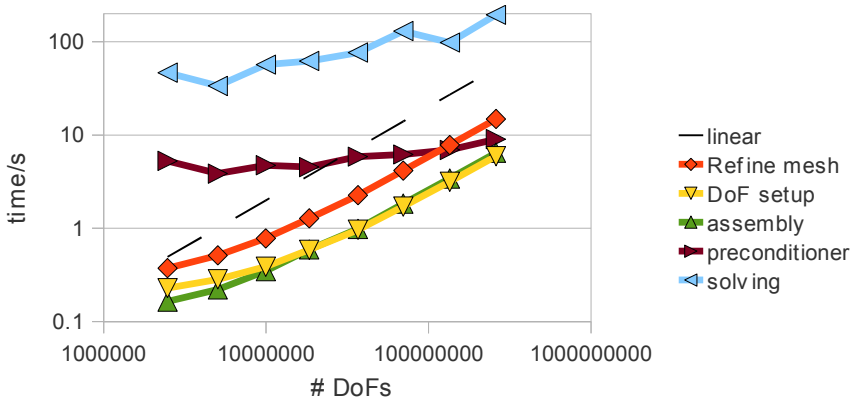

Figure 5: 2D mantle convection on 512 processors.

developments in `deal.II` outlined here enable us to do calculations that are two orders of magnitude larger than previously possible.

6 Acknowledgments

Timo Heister is partly supported by the German Research Foundation (DFG) through GK 1023. Martin Kronbichler is partially supported by the Graduate School in Mathematics and Computation (FMB). Wolfgang Bangerth was partially supported by Award No. KUS-C1-016-04 made by King Abdullah University of Science and Technology (KAUST), by a grant from the NSF-funded Computational Infrastructure in Geodynamics initiative through Award No. EAR-0426271, and by an Alfred P. Sloan Research Fellowship.

Part of the computations for this paper were done on the Hurr cluster of the Institute for Applied Mathematics and Computational Science (IAMCS) at Texas A&M University. Hurr is supported by Award No. KUS-C1-016-04 made by King Abdullah University of Science and Technology (KAUST).

REFERENCES

[1] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.0.0, Argonne National Laboratory, 2008.

[2] S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, B. F. Smith, and H. Zhang. PETSc Web page, 2009. http://www.mcs.anl.gov/petsc.

[3] W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler. Algorithms and Data Structures for Massively Parallel Generic Finite Element Codes. *In preparation.*

[4] W. Bangerth, R. Hartmann, and G. Kanschat. `deal.II` *Differential Equations Analysis Library, Technical Reference.* `http://www.dealii.org`.

[5] W. Bangerth, R. Hartmann, and G. Kanschat. deal.II — a General Purpose Object Oriented Finite Element Library. *ACM Transactions on Mathematical Software*, 33(4):27, August 2007.

[6] M. Benzi, G. H. Golub, and J. Liesen. Numerical Solution of Saddle Point Problems. *Acta Numerica*, 14:1–137, 2005.

[7] C. Burstedde, L. C. Wilcox, and O. Ghattas. p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *In preparation.*

[8] T. Heister, G. Lube, and G. Rapin. On robust parallel preconditioning for incompressible flow problems. In *Numerical Mathematics and Advanced Applications, ENUMATH 2009*. Springer, Berlin, 2010.

[9] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the Trilinos project. *ACM Trans. Math. Softw.*, 31:397–423, 2005.

[10] M. A. Heroux et al. Trilinos Web page, 2009. http://trilinos.sandia.gov.

[11] M. Kronbichler and W. Bangerth. Advanced numerical techniques for simulating mantle convection. *In preparation.*