Scalable Adaptive Mantle Convection Simulation on Petascale Supercomputers

Carsten Burstedde*, Omar Ghattas*¶, Michael Gurnis†, Georg Stadler*, Eh Tan‡,
Tiankai Tu*, Lucas C. Wilcox* and Shijie Zhong§
*Institute for Computational Engineering & Sciences, The University of Texas at Austin, Austin, Texas, USA
¶Depts. of Geological Sciences and Mechanical Engineering, The University of Texas at Austin, Austin, Texas, USA
†Seismological Laboratory, California Institute of Technology, Pasadena, California, USA
‡Computational Infrastructure for Geodynamics, California Institute of Technology, Pasadena, California, USA
§Department of Physics, University of Colorado, Boulder, Colorado, USA

***Abstract*—Mantle convection is the principal control on the thermal and geological evolution of the Earth. Mantle convection modeling involves solution of the mass, momentum, and energy equations for a viscous, creeping, incompressible non-Newtonian fluid at high Rayleigh and Peclet numbers. Our goal is to conduct global mantle convection simulations that can resolve faulted plate boundaries, down to 1 km scales. However, uniform resolution at these scales would result in meshes with a trillion elements, which would elude even sustained petaflops supercomputers. Thus parallel adaptive mesh refinement and coarsening (AMR) is essential.**

We present RHEA, a new generation mantle convection code designed to scale to hundreds of thousands of cores. RHEA is built on ALPS, a parallel octree-based adaptive mesh finite element library that provides new distributed data structures and parallel algorithms for dynamic coarsening, refinement, rebalancing, and repartitioning of the mesh. ALPS currently supports low order continuous Lagrange elements, and arbitrary order discontinuous Galerkin spectral elements, on octree meshes. A forest-of-octrees implementation permits nearly arbitrary geometries to be accommodated. Using TACC's 579 teraflops Ranger supercomputer, we demonstrate excellent weak and strong scalability of parallel AMR on up to 62,464 cores for problems with up to 12.4 billion elements. With RHEA's adaptive capabilities, we have been able to reduce the number of elements by over three orders of magnitude, thus enabling us to simulate large-scale mantle convection with a finest local resolution of 1.5 km.

I. INTRODUCTION

Mantle convection is the principal control on the thermal and geological evolution of the Earth. It is central to our understanding of the origin and evolution of tectonic deformation, the evolution of the thermal and compositional states of the mantle, and ultimately the evolution of the Earth as a whole. Plate tectonics and volcanism are surface manifestations of mantle convection, which in turn exert significant control on multiple surface processes (such as mountain building, sedimentary basin formation, and sea level change). By controlling the thermal state of the mantle, mantle convection dictates the cooling of the Earth's core and has a direct impact on the geodynamo. Understanding convection within Earth has been designated one of the "10 Grand Research Questions in Earth Sciences" in a recent National Academies report [1].

Mantle convection modeling involves solution of mass, momentum, and energy equations for a viscous, creeping, incompressible non-Newtonian fluid at high Rayleigh and Peclet numbers. Despite its central importance to our understanding of the dynamics of the solid Earth, simulation of global mantle convection down to the scale of faulted plate boundaries is currently intractable, due to the wide range of time and length scales involved. The length scales of interest range from several kilometers at faulted plate boundaries (requiring a minimum $O(1)$ km mesh), to the $O(10^4)$ km scale of the globe. Similarly, time scales range from $O(10^4)$ years to capture transport at the smallest length scales, to global transit time scales of $O(10^8-10^9)$ years. Taken together, this implies a space–time problem with $O(10^{16}-10^{17})$ degrees of freedom, which is well beyond the reach of contemporary supercomputers.

The advent of petascale computing promises to make such simulations tractable. However, uniform discretization of the mantle at 1 km resolution would result in meshes with nearly a trillion elements. Mantle convection simulations on such meshes are estimated to require one year on a sustained one petaflops system; moreover, mantle convection inverse problems, which we ultimately target, will require hundreds of such simulations. Adaptive mesh refinement and coarsening (AMR) methods can reduce the number of unknowns drastically by dynamically placing resolution only where needed. For the mantle convection problems we target, this can result in three to four orders of magnitude reduction in number of elements (as seen for example in the problem solved in Section VI). Thus, AMR has the potential to reduce the simulation wallclock time for

the mantle convection problems we target to hours on a sustained one petaflops system.

Unfortunately, parallel AMR methods can also impose significant overhead, in particular on highly parallel computing systems, due to their need for frequent re-adaptation and repartitioning of the mesh over the course of the simulation. Because of the complex data structures and frequent communication and load balancing, scaling dynamic AMR to petascale systems has long been considered a challenge. For example, the 1997 Petaflops Algorithms Workshop [2] assessed the prospects of scaling a number of numerical algorithms to petaflops computers. Algorithms were classified according to Class 1 (scalable with appropriate effort), Class 2 (scalable provided significant research challenges are overcome), and Class 3 (possessing major impediments to scalability). The development of dynamic grid methods—including mesh generation, mesh adaptation and load balancing—were designated Class 2, and thus pose a significant research challenge. In contrast, static-grid PDE solvers were classified as Class 1. The assessments of scalability that resulted from this workshop have withstood the test of time, and remain largely accurate today.

The main contribution of this paper is to present a new library for parallel dynamic octree-based finite element adaptive mesh refinement and coarsening that is designed to scale to hundreds of thousands of processors, and to describe its enablement of the first mantle convection simulations that achieve local 1 km resolution to capture yielding at faulted plate boundaries. ALPS (**A**daptive **L**arge-scale **P**arallel **S**imulations), our new library for parallel dynamic AMR, incorporates parallel finite element octree-based mesh adaptivity algorithms, including support for dynamic coarsening, refinement, rebalancing, and repartitioning of the mesh. RHEA is our new generation adaptive mantle convection code that builds on ALPS, and includes a parallel variable-viscosity nonlinear Stokes solver, based on Krylov solution of the (stabilized) Stokes system, with preconditioning carried out by approximate block factorization and algebraic multigrid V-cycle approximation (via the `BoomerAMG` package) of the inverse of viscous and pressure Schur complement operators, and adjoint-based error estimators and refinement criteria.

We discuss parallel scalability and performance on Ranger, the 579 teraflops, 62,976-core Sun system at the Texas Advanced Computing Center (TACC) that went into production in February 2008. Parallel dynamic AMR solutions of the advection-diffusion component of RHEA with up to 12.4 billion elements, on up to 62,464 cores of Ranger, indicate just 11% of the overall time is spent *on all components related to adaptivity* (the remainder is consumed by the PDE solver), while maintaining 50% parallel efficiency in scaling from *1 core to 62K cores*. The algorithmic scalability of the parallel variable-viscosity Stokes solver is also excellent: the number of iterations is almost insensitive to a 16,384-fold increase in number of cores, and to a problem size of over 3.2 billion unknowns.

All of the timings and performance evaluations presented here correspond to *entire end-to-end simulations*, including mesh initialization, coarsening, refinement, balancing, partitioning, field transfer, hanging node constraint enforcement, and PDE solution. The results show that, with careful attention to algorithm design and parallel implementation, the cost of dynamic adaptation can be made small relative to that of PDE solution, even for meshes and partitions that change drastically over time. Scalability to $O(10^4)$ cores follows. Figure 1 depicts dynamically evolving plumes from a typical regional mantle convection simulation, along with associated dynamically-adapted meshes, computed using RHEA and ALPS.

Most of the scalability and parallel performance results—as well as the AMR-enabled mantle convection simulations of yielding at faulted plate boundaries—presented in this paper are based on the initial version of ALPS, which employed second-order accurate trilinear hexahedral finite elements on Cartesian geometries. ALPS has recently been extended to support adaptivity using high-order spectral elements with discontinuous Galerkin discretizations on a very general class of geometries. These geometries are domains that can be decomposed into non-overlapping subdomains, each of which can be mapped to a hexahedron. The octree algorithms described in this paper thus operate on each subdomain, with appropriate handshaking between subdomains; the result is a *forest of octrees*. In addition to the results using trilinear finite elements on Cartesian geometries, we also give preliminary results for parallel high order spectral element discontinuous Galerkin AMR simulations using forest-of-octree adaptivity on spherical geometries.

The remaining sections of this paper are organized as follows. In Section II, we give a brief description of the challenges involved in modeling global mantle convection. Section III describes the numerical discretization and solution algorithms employed, while Section IV overviews the octree algorithms we employ. Parallel performance and scalability results are presented in Section V, and Section VI presents results from the first large-scale mantle convection simulations with 1 km local resolution at faulted plate boundaries. In Section VII, we give a brief description, and performance evaluation, of extensions to ALPS that support high order spectral element discretizations and forest-of-octrees adaptivity on general geometries. Finally, we end with conclusions in Section VIII.

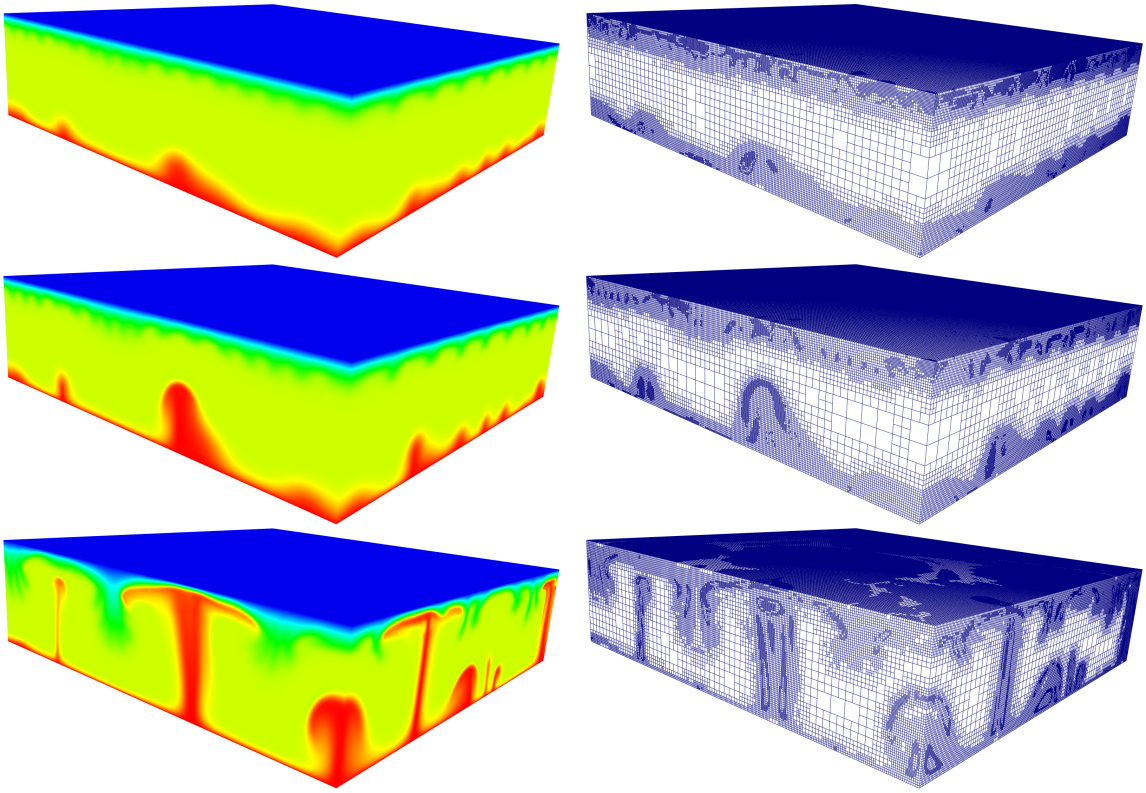

Fig. 1. Illustration of dynamic mesh adaptation for mantle convection. The figure shows snapshots of the thermal field T at three time instants (left column) and corresponding adapted meshes (right column). The mesh resolves the rising plumes as well as the instabilities at the top layer. The elements span levels 4 to 9 in octree depth.

II. Mantle convection modeling

The dynamics of mantle convection are governed by the equations of conservation of mass, momentum, and energy. Under the Boussinesq approximation for a mantle with uniform composition and the assumption that the mantle deforms as a viscous medium, a simplified version of these equations reads (e.g. [3], [4]):

$$\boldsymbol{\nabla}\cdot\boldsymbol{u}=\mathbf{0}, \tag{1}$$

$$\boldsymbol{\nabla}p-\boldsymbol{\nabla}\cdot\left[\eta(T,\boldsymbol{u})\left(\boldsymbol{\nabla}\boldsymbol{u}+\boldsymbol{\nabla}\boldsymbol{u}^{\mathsf{T}}\right)\right]=\mathrm{Ra}\,T\,\boldsymbol{e}_r \tag{2}$$

$$\frac{\partial T}{\partial t}+\boldsymbol{u}\cdot\boldsymbol{\nabla}T-\boldsymbol{\nabla}^2T=\gamma, \tag{3}$$

where $\boldsymbol{u}$, p, η, and T are the velocity, pressure, temperature- and strain-rate-dependent viscosity, and temperature, respectively; γ is the internal heat generation; $\boldsymbol{e}_r$ is the unit vector in the radial direction; and Ra is a Rayleigh number that controls the vigor of convection and is defined as $\mathrm{Ra}=\alpha\rho_0 g\Delta T D^3/(\kappa\eta_0)$, where α, ρ_0, η_0, and κ are the reference coefficient of thermal expansion, density, viscosity, and thermal diffusivity, respectively, ΔT is the temperature difference across the mantle with thickness D, and g is the gravitational acceleration. Mantle convection modeling requires numerical solutions to these governing equations. However, the highly nonlinear and multiscale physical processes caused by complex deformation mechanisms, the significant compositional heterogeneities (e.g., continents) associated with melting and mantle differentiation, and large Rayleigh numbers ($\mathcal{O}(10^8-10^9)$) pose significant numerical and computational challenges [5].

Mantle viscosity at high temperatures ($>1000°$C) depends on temperature, pressure, stress, and grain size, while at intermediate temperatures (between $450°$C and $1000°$C) the deformation is controlled by plasticity, and at low temperatures ($<450°$C) by faulting. While the high temperature deformation mechanism can be reasonably approximated as Newtonian flow, the latter two are highly nonlinear and may lead to extreme localization in deformation such as through tectonic faults at plate boundaries where earthquakes occur. At faulted plate boundaries, one may need $\sim$1 km resolution to resolve the physics of deformation [6], [7].

There are three difficulties in modeling highly nonlinear and multiscale mantle convection problems. The first is with the highly nonlinear deformation and large variations ($\mathcal{O}(10^5)$ or greater) in material properties that are caused by nonlinear deformation-, temperature-, and grain size-dependent viscosity. This makes the linearized

Stokes systems of equations arising at each iteration and time step extremely ill-conditioned and not easily solved with standard iterative solvers. Second, to fully resolve the physics at the smallest scale of faulted plate boundaries in global models with a uniform mesh of 1 km resolution, one may need $\mathcal{O}(10^{12})$ elements, beyond the reach of any contemporary supercomputer. Third, the system is highly time-dependent with plate boundaries and mantle plumes migrating, often with the same velocities as plates. This rules out static mesh methods that are widely used for steady-state and instantaneous mantle convection problems. Dynamic AMR appears to be the only viable approach that can accommodate the wide range of minimal spatial scales required. For a variable resolution of 1 km near plate boundaries, 5 km within thermal boundary layers and plumes, and 15 km for the rest of the mantle, we estimate the total number of elements needed for AMR to be $\mathcal{O}(10^9)$, three orders of magnitude smaller than using a uniform mesh. This is an approximate figure, and will depend on the number of plumes that will be realized in simulations under mantle conditions.

III. Numerical discretization and solution

We give a brief overview of the numerical algorithms used to discretize and solve the governing equations of mantle convection (1)–(3) on highly adapted meshes. The discussion here is limited to low order discretization on Cartesian geometries; extensions to high order and more general geometries are discussed in Section VII. We employ trilinear finite elements for all fields (temperature, velocity, pressure) on octree-based hexahedral meshes. These meshes contain hanging nodes on nonconforming element faces and edges. To ensure continuity of the unknowns, algebraic constraints are used to eliminate the hanging nodes at the element level. Galerkin finite element discretizations of Equation (3) require stabilization to suppress spurious oscillations for advection-dominated flows, for which we use the streamline upwind/Petrov-Galerkin (SUPG) method [8]. For realistic mantle convection regimes, Equation (3) is strongly advection-dominated, so we use an explicit predictor-corrector time integrator [9] for its solution. It is well known that equal-order velocity-pressure discretization of the Stokes equation does not satisfy the inf-sup condition for numerical stability of saddle point problems. We circumvent this condition via polynomial pressure stabilization [10], [11]. Solution of the coupled system (1)–(3) is split into an advection-diffusion time step to update temperature, followed by a variable-viscosity Stokes solve to update the flow variables. The nonlinearity imposed by strain-rate-dependent viscosity is addressed with a Picard-type fixed point iteration.

The discretization of the variable-viscosity Stokes equation leads to a symmetric saddle point problem, which we solve iteratively by the preconditioned minimal residual (MINRES) method [12]. Each MINRES iteration requires one application of the Stokes matrix to a vector, storage of two vectors, and two inner products. The stabilized Stokes matrix can be block-factored into the form

$$\begin{pmatrix} A & B^\top \\ B & -C \end{pmatrix} = \begin{pmatrix} I & 0 \\ BA^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & -S \end{pmatrix} \begin{pmatrix} I & A^{-1}B^\top \\ 0 & I \end{pmatrix}$$

where $S = (BA^{-1}B^\top + C)$ is the Schur complement, A is the discrete tensor divergence operator, B is the discrete vector divergence, and C is the stabilization matrix. The factorization shows that the Stokes operator is congruent to a block-diagonal matrix. Neglecting the off-diagonal blocks motivates use of the symmetric positive definite preconditioner

$$P = \begin{pmatrix} \tilde{A} & 0 \\ 0 & \tilde{S} \end{pmatrix}.$$

Here, $\tilde{A}$ is a variable-viscosity discrete vector Laplacian approximation of A, which is motivated by the fact that for constant viscosity and Dirichlet boundary conditions, A and $\tilde{A}$ are equivalent. $\tilde{S}$ is an approximation of the Schur complement given by a lumped mass matrix weighted by the inverse viscosity η^{-1}. The resulting diagonal matrix $\tilde{S}$ is spectrally equivalent to S [11]. The preconditioner P captures the variability in element size as well as viscosity, which is essential for algorithmic scalability of MINRES for highly heterogeneous viscosities and adaptively refined meshes. Moreover, P replaces the indefinite Stokes system with a much-easier-to-solve block-diagonal system consisting of Poisson-like or Gram-type blocks.

Each MINRES iteration requires an approximate solve with the preconditioner P. $\tilde{A}$ contains, for each velocity component, a highly-variable-viscosity discrete Poisson operator. For the approximate solution of these systems, we employ algebraic multigrid (AMG). Compared to geometric multigrid, AMG can have advantages due to its ability to mitigate heterogeneities in mesh size and viscosity. In its setup phase, AMG derives a coarse grid hierarchy and corresponding restriction and interpolation operators. We use the parallel AMG implementation `BoomerAMG` from the `hypre` package [13], [14]. `BoomerAMG` allows the user to choose among various coarsening strategies and to trade off effectiveness of the grid hierarchy against setup time. Using one V-cycle of AMG as a preconditioner leads to almost optimal algorithmic scalability, as reflected by the nearly-constant number of MINRES iterations as the problem size and number of cores increases in Figure 2. On the other hand, AMG can impose significant communication requirements, particularly at very large numbers of processors;

#cores	#elem	#elem/core	#dof	MINRES #iterations
1	67.2K	67.2K	271K	57
8	514K	64.2K	2.06M	47
64	4.20M	65.7K	16.8M	51
512	33.2M	64.9K	133M	60
4096	267M	65.3K	1.07B	67
8192	539M	65.9K	2.17B	68

Fig. 2. Weak scalability of variable-viscosity Stokes solver, which involves MINRES iterations preconditioned by approximate block factorization of the Stokes system based on one V-cycle of `BoomerAMG` applied to vector Poisson and Schur complement blocks. Table demonstrates excellent algorithmic scalability of solver, despite strong ill-conditioning due to severe heterogeneity in material properties. Number of MINRES iterations is essentially insensitive to an 8192-fold increase in number of cores, and problem size to over 2 billion unknowns.

we defer a discussion of the parallel scalability of the variable-viscosity Stokes solver to Section V.

IV. The ALPS library for parallel AMR

In this section we briefly describe parallel data structures, parallel algorithms, and implementation issues underlying ALPS, our library for parallel finite element octree-based AMR that supports dynamic coarsening, refinement, rebalancing, and repartitioning of the mesh. In this section we describe the initial version of ALPS, which supports trilinear hexahedral finite elements on Cartesian geometry and employs the Octor library [15], [16]. Extensions to high order discretizations on general geometries via forest-of-octree adaptivity are discussed in Section VII.

We begin first by contrasting octree-based finite element AMR methods with other AMR methods. These generally fall into two categories, structured (SAMR) and unstructured (UAMR) (see the review in [17] and the references therein). SAMR methods represent the PDE solution on a composite of hierarchical, adaptively-generated, logically-rectangular, structured grids. This affords reuse of structured grid sequential codes by nesting the regular grids during refinement. Moreover, higher performance can be achieved due to the locally regular grid structure. SAMR methods maintain consistency and accuracy of the numerical approximation by carefully managing interactions and interpolations between the nested grids, which makes high-order-accurate methods more difficult to implement than in the single-grid case. This results in communications, load balancing, and grid interaction challenges that must be overcome to achieve scalability to large numbers of processors. Exemplary SAMR implementations that have scaled to $O(10^3)$ processors include Chombo [18], PARAMESH [19], [20], SAMRAI [21], and Cart3D [22].

In contrast to SAMR, UAMR methods typically employ a single (often conforming) mesh that is locally adapted by splitting and/or aggregating (often) tetrahedral elements. As such, high-order accuracy is achieved naturally with, e.g., high order finite elements, and the unstructured nature of the tetrahedra permits boundary conformance. Furthermore, the conforming property of the mesh eliminates the need for interpolations between grid levels of SAMR methods. The challenge with UAMR methods is to maintain well-shaped elements while adapting the mesh, which is particularly difficult to do in parallel due to the need to coordinate the coarsening/refinement between processors. As in SAMR, dynamic load-balancing, significant communication, and complex data structures must be overcome. Exemplary UAMR implementations include Pyramid [23] and PAOMD [24], [25].

ALPS uses parallel octree-based hexahedral finite element meshes, which can be thought of as being intermediate between SAMR and UAMR. We build on prior approaches to parallel octree mesh generation [16], [26], and extend them to accommodate dynamic solution-adaptive refinement and coarsening. Adaptation and partitioning of the mesh are handled through the octree structure and a space-filling curve, and a distinct mesh is generated from the octree every time the mesh changes. Flexibility of the mesh is attained by associating a hexahedral finite element to each octree leaf. This approach combines the advantages of UAMR and SAMR methods. Like UAMR, a conforming approximation on a single mesh (as opposed to a union of structured grid blocks) is constructed, though here the mesh itself is not conforming; instead, algebraic constraints on hanging nodes impose continuity of the solution field across coarse-to-fine element transitions. These constraints are imposed at the element level, making use of interpolations from the element shape functions. A global 2-to-1 balance condition is maintained, i.e., edge lengths of face- and edge-neighboring elements may differ by at most a factor of two. This ensures smooth gradations in mesh size, and simplifies the incorporation of algebraic constraints. High-order approximations and general geometries are naturally accommodated as in any finite element method (see Section VII). Like SAMR, element quality is not an issue, since refinement and coarsening result from straightforward splitting or merging of hexahedral elements, but communication is reduced significantly compared to SAMR: interpolations from coarse to fine elements are limited to face/edge information and occur just once each mesh adaptation step, since a single multi-resolution mesh is used to represent the solution between adaptation steps. (Forest-of-) octree-based AMR on hexahedral finite element meshes with hanging node constraints has been employed in such li-

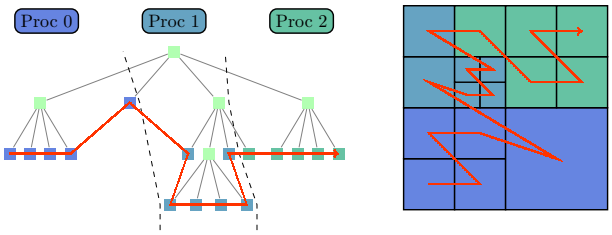

Fig. 3. Illustration of the distinct octree and mesh parallel data structures used in ALPS and their partitioning (quadtree shown for display purposes). The data structures are linked logically by a 1-to-1 correspondence between leaves of the tree and elements of the mesh. A pre-order traversal of the leaves of the octree in the sequence of triples (z, y, x) creates a space-filling curve in Morton (or z) order, shown in red. A load-balanced partition of the octree is determined by partitioning the space-filling curve into segments of equal length; the partitions are shown as shades of blue.

braries as deal.II [27], libMesh [28], *hp*3d [29], AFEAPI [30], and SAMRAI [31], and has been demonstrated to scale to well to $\mathcal{O}(100 - 1000)$ processors. Our goal is the design and implementation of parallel data structures and algorithms that enable octree-based finite element AMR methods to scale to the $\mathcal{O}(10^5)$ cores (and beyond) characteristic of sustained petaflops architectures. In the remainder of this section, we discuss the distributed data structures, parallel algorithms, and implementations underlying ALPS.

A. Octrees and space-filling curves

All coarsening and refinement information is maintained within an octree data structure, in which there is a one-to-one correspondence between the leaves of the octree and the hexahedral finite elements of the mesh (see Figure 3). The root of the octree represents an octant of the size of the computational domain. The leaves of the octree represent the elements that are present in the current mesh. The parents of these leaves are used to determine the relationships between the leaves. When an element is refined, it is split into eight equal-sized child elements. This is represented in the octree by adding eight children to the leaf octant representing the element being divided. A coarsening operation amounts to removing all children with a common parent. The operations defined on the octree and the mesh are detailed below.

Most of the AMR functions in ALPS operate on the octree from which the mesh is generated. Since we target large parallel systems, we cannot store the full octree on each core. Thus, the tree is partitioned across cores. As we will see below, cores must be able to determine which other core owns a given leaf octant. For this, we rely on a space-filling curve, which provides a globally unique linear ordering of all leaves. As a direct consequence, each core stores only the range of leaves each other core owns. This can be determined by an `MPI_Allgather` call on an array of long integers with a length equal to the number of cores. Thus the only global information that is required to be stored is one long integer per core. We use the Morton ordering as the specific choice of space-filling curve. It has the property that nearby leaves tend to correspond to nearby elements given by the pre-order traversal of the octree, as illustrated in Figure 3.

The basic operations needed for mesh generation and adaptation require each core to find the leaf in the octree corresponding to a given element. If the given element does not exist on the local core, the remote core that owns the element must be determined. This can be done efficiently given the linear order of the octree; see [16] for details. The inverse of this operation, determining the element corresponding to a given leaf, can be made efficient as well.

B. AMR functions for mesh generation and adaptation

The procedure for adapting a mesh involves the following steps. First, a given octree is coarsened and refined based on an application-dependent criterion, such as an error indicator. Next, the octree is "balanced" to enforce the 2-to-1 adjacency constraint. After these operations, a mesh is extracted so that the relevant finite element fields can be transferred between meshes. Following this, the adapted mesh is partitioned and the finite element fields are transferred to neighboring cores according to their associated leaf partition. Figure 4 illustrates this process. The key features of the AMR functions in this figure are described below. Here, "application code" refers to a code for the numerical discretization and solution of PDEs built on meshes generated from the octree.

NEWTREE. This function is used to construct a new octree in parallel. This is done by having each core grow an octree to an initial coarse level. (This level is several units smaller than the level used later in the simulation.) At this point, each core has a copy of the coarse octree, which is then divided evenly between cores. The cores finish by pruning the parts of the tree they do not own, as determined by the Morton order. This is an inexpensive operation that requires no communication.

COARSENTREE/REFINETREE. Both of these work directly on the octree and are completely local operations requiring no communication. On each core, REFINETREE traverses the leaves of the local partition of the octree, querying the application code whether or not a given leaf should be refined. If so, eight new leaves are added to the level beneath the queried octant. COARSENTREE follows a similar approach, examining the local partition of the octree for eight leaves from the same parent that the application code has marked for coarsening. Note that we do not permit coarsening of a set of leaf octants that are distributed across cores. This

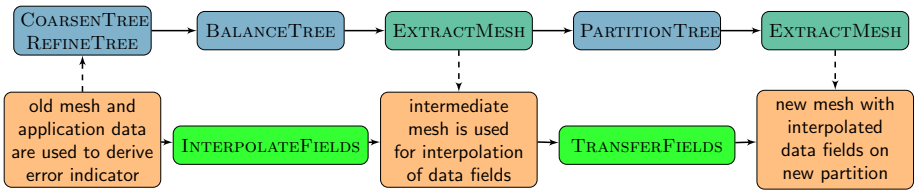

Fig. 4. Functions for mesh adaptivity. Blue boxes correspond to functions that operate on the octree only; cyan boxes denote functions that act between the octree and the mesh; mesh and data field operations are enclosed in orange boxes; green boxes represent functions that act on the mesh and the application data fields only. Solid arrows represent the flow of function calls; dashed arrows signify the input and output of mesh and/or data fields.

is a minor restriction, since the number of such leaf sets is at most one less than the number of cores.

BALANCETREE. Enforcing the 2-to-1 size difference constraint between adjacent elements, also known as balancing the tree, is done with the parallel prioritized ripple propagation algorithm described in [16]. The algorithm uses a buffer to collect the communication requests as it balances the octree one refinement level at a time. This buffering aggregates all of the communication so that the number of communication rounds scales linearly with the number of refinement levels.

PARTITIONTREE. Dynamic partitioning of the octree for load balance is a key operation that must be performed frequently throughout a simulation as the mesh is adapted. The goal is to assign an equal number of elements to each core while keeping the number of shared mesh nodes between cores as small as possible. The space-filling curve offers a natural mechanism for partitioning the octree, and hence mesh, among cores. The curve is divided into one segment per core according to the total ordering. The result is a partition with good locality properties, i.e., neighboring elements in the mesh tend to reside on the same core.

EXTRACTMESH. This function builds the mesh from a given octree and sets up the communication pattern for the application code. Unique global ordering of the elements and degrees of freedom of the mesh are determined and the relationship between the elements and nodes is established. Hanging nodes do not have unknowns associated with them, and therefore are not part of the global degrees of freedom. Their dependence on the global degrees of freedom, which is required to enforce the continuity of the finite element data fields, is also determined in this function. Ghost layer information (one layer of elements adjacent to local elements) from remote cores is also gathered.

INTERPOLATEFIELDS. This function is used to interpolate finite element data fields from one mesh to a new mesh that has been created by at most one level of coarsening and refinement. For simple interpolation between two trilinear finite element meshes, there is no global communication required to execute this step, given the value of ghost degrees of freedom. Once finished, the cores gather the information for their ghost degrees of freedom by communicating with their neighboring cores.

TRANSFERFIELDS. This function is the operation on the data fields analogous to PARTITIONTREE. Following the Morton ordering among the degrees of freedom, the data associated with element nodes is transferred between cores to complete the load-balancing stage. At the end of this process every core has obtained the data for all elements it owns and discarded what is no longer relevant due to the changed partition.

MARKELEMENTS. We target PDE applications in which solution features evolve spatially over time, necessitating dynamic mesh adaptivity. Given an error indicator η_e for each element e, MARKELEMENTS decides which elements to mark for coarsening and refinement. Since we wish to avoid a global sort of all elements according to their error indicators, we employ a strategy that adjusts global coarsening and refinement thresholds through collective communication. This process iterates until the overall number of elements expected after adaptation lies within a prescribed tolerance around a target.

V. PERFORMANCE OF PARALLEL AMR ON RANGER

In this section we study the performance of the ALPS parallel AMR library on Ranger, the 579 teraflops, 123 terabyte, 62,976-core Sun/AMD system at TACC. In our analysis below, we study both isogranular (i.e., weak) and fixed-size (i.e., strong) scalability. The *complete end-to-end run time* is always used to calculate speedups and parallel efficiencies; this includes mesh initialization, coarsening, refinement, 2:1 octree balancing, octree/mesh partitioning, field transfer, hanging node constraint enforcement, and PDE solution.

We conduct most of our AMR experiments on solutions of the time-dependent advection-diffusion equation (3) in the high-Peclet number regime, for several reasons. First, this represents a challenging test problem for parallel mesh adaptivity when advection dominates, since the equation acquires hyperbolic character: sharp fronts are transported and frequent coarsening/refinement and repartitioning of the mesh for load balance are required. Second, since we use explicit time-stepping

for the advection-diffusion component, the work done per time step is linear in problem size and has the usual surface/volume communication-to-computation ratio, and therefore the numerical components should exhibit good scalability. This allows us to focus our assessment on scalability of the AMR components, without diluting their role by mixing in the implicit nonlinear Stokes solver. Finally, the low-order-discretized, explicitly-solved, scalar, linear transport equation is a severe test of AMR, since there is very little numerical work over which to amortize AMR: vector unknowns, nonlinear equations, implicit solvers, and high-order discretizations increase the computational work by orders of magnitude, making it difficult to assess the scalability of the standalone AMR components. Indeed, results presented at the end of this section demonstrate that—thanks to the scalable design of the underlying algorithms—all AMR components together consume negligible time on $O(10^4)$ cores when running side-by-side with our full (nonlinear, implicit, vector) mantle convection code. For all of the studies in this section, the frequency of adaptation is fixed at once every 32 time steps for the advection-diffusion component, and once every 16 time steps for the full mantle convection code.

A. Extent of mesh adaptation

Figure 5 illustrates the extent of mesh adaptation in a typical simulation on 4096 cores with about 131,000 elements per core. The left image shows the number of elements that are coarsened, refined, and unchanged within each mesh adaptation step. The figure illustrates the aggressiveness of mesh adaptation when driven by the advection-dominated transport problem: typically half the elements are coarsened or refined at each adaptation step. The figure also demonstrates that the MARKELEMENTS function is able to keep the total number of elements about constant. The right image depicts the distribution of elements across the refinement levels for selected mesh adaptation steps. By the 8th adaptation step, the mesh contains elements at 10 levels of the octree, leading to a factor of 512 variation in element edge length. These results illustrate the significant volume and depth of mesh adaptation over the course of a simulation. This is worth keeping in mind when examining the scalability results in subsequent sections.

B. Fixed-size scaling

Figure 6 shows the speedups for small, medium-sized, large, and very large test problems as the number of cores increases while the problem size remains the same (strong scaling). The small, medium, and large meshes on 1, 16, and 256 cores (respectively) contain

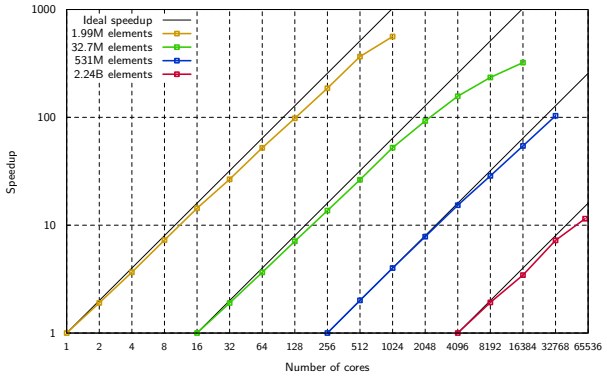

Fig. 6. Fixed-size scalability: Speedups based on total runtime plotted against the number of cores for four different problem sizes. Baseline granularity of the small, medium, and large problems is the same, at 2 million elements per core. The very large problem has a baseline granularity of 547,000 elements per core.

approximately 2.1 million elements per core. The largest test problem has 0.54 million elements per core at 4096 cores. Using the MARKELEMENTS function, these numbers are kept approximately constant during the simulation. We selected these problem sizes based on the multicore architecture of Ranger. Each compute node has 32 GB of main memory and 16 cores. The size of each problem is chosen such that it uses one core per node for the initial number of cores (1, 16, 256). This provides optimal conditions for these initial runs since every core has exclusive access to processor memory and the node's network interface. The first four scaling steps keep the number of nodes constant and increase the number of cores used per node to 16, while at the same time increasing the sharing of resources. Once all 16 cores are used on a node, we increase the number of nodes in powers of two.

The fixed-size scaling speedups are nearly optimal over a wide range of core counts. For instance, solving the small test problem (yellow line) on 512 cores is still 366 times faster than the solution on a single core. Similarly, the overall time for solving the medium-size problem (green line) on 1024 cores results in a speedup of more than 52 over the runtime for 16 cores (optimal speedup is 64), the large problem (blue line) exhibits a speedup on 32,768 cores of 101 over the same problem running on 256 cores (128 is optimal) and solving the very large problem (red line) on 61440 compared to 4096 cores is a factor of 11.5 faster (optimal speedup is 15).

These speedups are remarkable given the extensive mesh adaptation (as illustrated in Figure 5), and the associated need for frequent and aggressive load balancing (and hence communication),

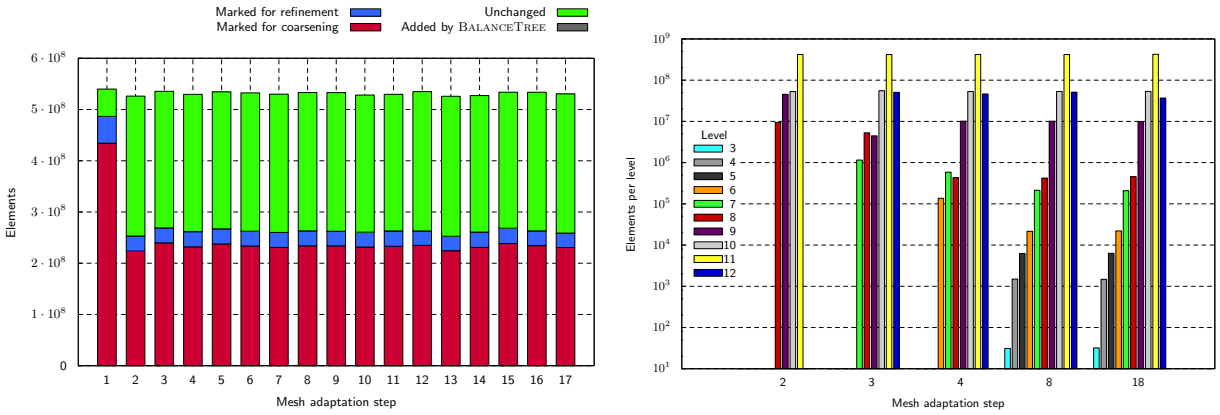

Fig. 5. Left: number of elements that are refined (blue), coarsened (red), created to respect the 2:1 balance condition (gray, barely visible), and unchanged (green), at each mesh adaptation step. Right: number of elements within each level of the octree for selected mesh adaptation steps. The octree has elements across 10 levels for late-time meshes. Simulation ran on 4096 cores with approximately 131,000 elements per core.

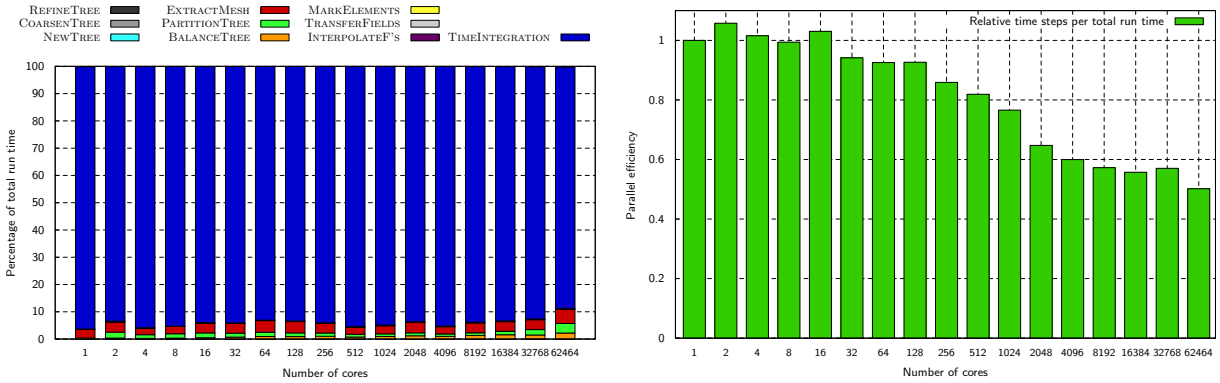

Fig. 7. Weak scalability for advection-diffusion using trilinear elements. On the left is a breakdown of total run time into different components related to numerical PDE integration (blue) and AMR functions (all other colors), with increasing number of cores from 1 to 62,464. Problem size increases isogranularly at roughly 131,000 elements per core (largest problem has approximately 7.9 billion elements). The overall runtime is dominated by the numerical integration. The most expensive operation related to AMR is ExtractMesh, which uses up to 6% of the runtime. Overall, AMR consumes 11% or less of the overall time. On the right is the parallel efficiency measured in total processed elements per core per total run time, normalized by the total processed elements per total run time for a single core, with increasing number of cores from 1 to 62,464. Despite a 62K-fold increase in problem size and number of cores, parallel efficiency remains above 50%.

C. Isogranular scaling

Figure 7 provides evidence of the isogranular scalability of parallel AMR on the advection-diffusion problem using the Alps library. The left image depicts the breakdown of the overall runtime into the time consumed by the explicit numerical time-stepping (blue) and all of the AMR functions (all other colors), for weak scaling from 1 to 62,464 cores. Problem size is held approximately constant at 131,000 elements per core, i.e. 7.9 billion elements for the largest problem. The results indicate that the mesh adaptation functions—including refinement, coarsening, interpolation, field transfer, rebalancing, repartitioning, and mesh extraction—together impose little overhead on the PDE solver. Only for 62K cores does the total cost of AMR exceed 10% of end-to-end run time, and even then just barely. This, despite the fact mentioned above, that we are solving a scalar, linear, low-order-discretized, explicitly-time-advanced advection-diffusion problem. PartitionTree completely redistributes the octree (and thus the mesh) among all cores to balance the load (we do not impose an explicit penalty on data movement in the underlying partitioning algorithm). Although this requires sending and receiving large amounts of data using one-to-one communication, Figure 7 demonstrates that the time for PartitionTree remains more-or-less constant for all core counts beyond one (for which partitioning is unnecessary). This is also reflected in the timings for TransferFields, which do not show up in Figure 7 even though complete finite element data fields are redistributed across the network.

The right image of Figure 7 displays the parallel efficiency for isogranular scaling from 1 to 62,464 cores. Here, parallel efficiency is defined as the total processed

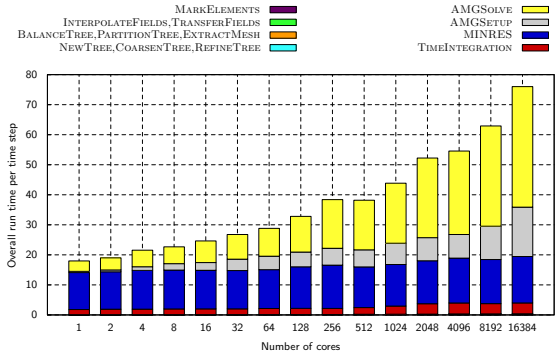

Fig. 8. Breakdown of overall run time per time step into different components related to AMG setup (gray), AMG V-cycle (yellow), MINRES iterations (blue) and numerical PDE integration (red). All other operations are AMR functions and have negligible cost, even though the mesh is adapted every 16 time steps. The number of cores increases from 1 to 16,384. Problem size increases isogranularly at roughly 50,000 elements per core (largest problem has approximately 815 million elements). The Stokes solve dominates the runtime (more than 95%). The AMR, explicit advection-diffusion time integration, and MINRES iterations (including element matvecs and inner products) exhibit nearly ideal scaling, while the AMG setup and V-cycle times grow with problem size.

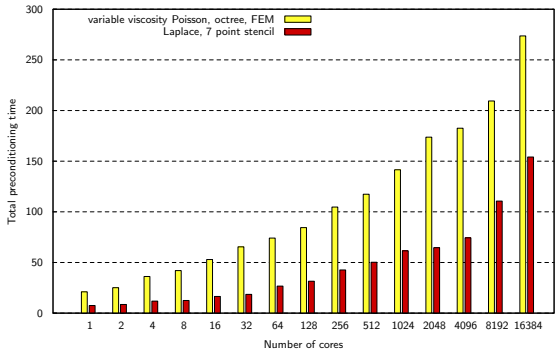

Fig. 9. Overall time for AMG setup and 160 V-cycles as the core count increases from 1 to 16,384. Problem size increases isogranularly at roughly 50,000 elements per cores. Timings are given for the Laplacian discretized on a regular grid with a 7-point stencil, and for a variable-viscosity Poisson operator on an adapted hexahedral finite element mesh.

elements per core per total run time for a given core count, normalized by the total processed elements per total run time for a single core. A parallel efficiency of 1.0 indicates perfect isogranular scalability. For example, if the adaptivity process resulted in exactly twice as many elements created and the same number of time steps when doubling the core count and tightening the refinement criteria (to increase problem size), then our parallel efficiency measure would reduce to simply observing whether the total run time remained constant. Otherwise, normalizing by the number of elements per core and the number of time steps accounts for possible vagaries of the adaptivity process. As can be seen in the figure, despite extensive adaptivity and a problem that maximally stresses parallel adaptivity, parallel efficiency remains above 50% in scaling from 1 to 62,464 cores. This excellent scalability is no surprise, given the small fraction of overall time consumed by AMR relative to PDE solution.

D. Isogranular scaling of full mantle convection code

The isogranular scalability of the full mantle convection code is shown in Figure 8. The AMR functions, explicit advection-diffusion time-stepping, and MINRES iterations (which are dominated by element-level matvecs) scale nearly optimally from 1 to 16,384 cores, as evidenced by the nearly-constant runtime as the problem size and number of cores increase. Indeed, the explicit time-stepping and AMR operations (the mesh is adapted every 16 time steps) are negligible relative to the MINRES iterations. On the other hand, the times for the AMG setup and V-cycle (employed for the diagonal blocks of the preconditioner) do increase with increasing problem size, due to communication overheads and growth in complexity. To analyze this further, in Figure 9 we compare isogranular scalability of the main building block of our Stokes preconditioner, namely a highly variable-viscosity, finite element-discretized, adapted-grid Poisson solve, with the best possible situation to test AMG scalability: a Laplace solve discretized on a regular grid with a 7-point finite difference stencil. To provide for a fair comparison with Figure 8 for the Stokes solves, where the preconditioner is reused as long as the mesh remains unadapted, i.e. one AMG setup is performed per 16 time steps at an average of 10 V-cycles required per time step, the reported timings in Figure 9 include one AMG setup and 160 V-cycles. Note that while the Laplace preconditioner application has lower run time, it yields similar parallel scalability as the more complex Poisson preconditioner application. Thus, we cannot expect better performance from the latter, given the scaling of the former. Finally, in Figure 10 we show the time used by the specific AMR functions in each adaptation step compared to the time used in the solver. Note that the time spent for mesh adaptation remains well below 1% of the overall run time over a range of 1 to 16,384 cores. Thus, in comparison with the full mantle convection solver, AMR consumes negligible time.

VI. Mantle convection AMR simulations

In this section, we present end-to-end simulations that demonstrate the full power of AMR to resolve the range of length scales, nonlinearities, and time-dependent phenomena that we expect to be present in global models of mantle convection. Although these calculations are in a Cartesian regional domain, they approximate the range of processes encountered in spherical domain models,

#cores	NewTree	Solve time	CoarsenT RefineT	BalanceT	PartitionT	ExtractM	InterpolateF TransferF	MarkE	AMR time / Solve time
1	0.16	269.00	0.01	0.03	0.00	0.48	0.05	0.04	0.23%
8	0.45	355.42	0.02	0.10	0.35	0.77	0.06	0.08	0.39%
64	0.53	434.75	0.02	0.20	0.46	0.97	0.07	0.14	0.43%
512	0.62	578.53	0.02	0.61	0.67	1.83	0.09	0.17	0.59%
4096	0.84	813.43	0.02	1.04	0.95	2.73	0.15	0.23	0.63%
16384	1.61	1134.30	0.01	1.23	1.22	2.85	0.20	0.32	0.51%

Fig. 10. Breakdown of timings (in seconds) from Figure 8 for AMR solution of the mantle convection equations over a range of 1 to 16,384 cores with isogranular increase in problem size. The second column shows the time for mesh and tree initialization, which is performed just once in the simulation. All other timings are provided per mesh adaptation step, i.e., per 16 time steps. The last column shows the percentage of overall time spent in AMR components relative to the solve time, which is less than 1%.

which will be pursued using the extensions described in the next section. We have verified RHEA with the widely used, validated, static mesh mantle convection code CitcomCU [32].

We present a mantle convection simulation for which adaptivity is essential for capturing localization phenomena. We adopt a viscosity law that displays yielding under high stress. This simplified approach dynamically achieves large-scale motions with narrow zones of low viscosity such as those associated with plate tectonics and faulted plate boundaries, an intermediate mode of convection associated with instability of the low viscosity thermal boundary layers, and the smallest modes of secondary convection associated with the instability of the tectonic plates. The models are in a $8\times4\times1$ Cartesian non-dimensional domain (equivalent to a dimensional domain approximately 23,200 km across and 2,900 km deep) with a temperature- and pressure-dependent viscosity that yields under high deviatoric stress according to

$$\eta = \begin{cases} \min\left\{10\exp(-6.9T), \frac{\sigma_y}{2\dot{e}}\right\}, & z > 0.9 \\ 0.8\exp(-6.9T), & 0.9 \geq z > 0.77 \\ 50\exp(-6.9T), & z \leq 0.77 \end{cases}$$

where σ_y is the yield stress and $\dot{e}$ is the second invariant of the deviatoric strain rate tensor. The three-layered viscosity structure simulates the lithosphere, the aesthenosphere, and the lower mantle. Shallow material can yield, while deeper material is subject only to a temperature-dependent viscosity. This is an approximation to laboratory-derived constitutive relations, but demonstrates our ability to incorporate weak plate boundaries and strong downwellings (i.e. slabs) that are recognized as essential parts of the dynamics of plate tectonics (e.g. [5]). Here, the viscosities range over four orders of magnitude.

Our AMR method well resolves both the evolving fine scale thermal structures as well as zones of plastic failure associated with the boundaries of plate-like structures (Figures 1 and 11). The high Rayleigh number of the mantle together with temperature-dependent viscosity leads to sharp thermal and material boundaries that rapidly move through the convecting domain, often at the apex of rising plumes and descending slabs. Both the hot rising and cold descending structures are ideally captured by AMR (Figure 1). Fine meshes dynamically follow the apex of plumes and slabs and resolve the temperature and viscosity contrasts around deep seated, low viscosity plume conduits. Substantial volumes of the isothermal (and relatively slow moving) interior are sufficiently resolved with large elements. The AMR efficiently coarsens the mesh as it tracks plumes ascending out of previously finely-resolved regions (Figure 1).

Within the stiff, cold upper thermal boundary layer, the strain rates are localized in zones above the descending sheets and cause yielding. Within the region of yielding, the mesh has been refined down to about 3.0 km with some parts down to 1.5 km (Figure 11, lower right plot). These zones of yielding have high strain rates and low viscosity. The rest of the upper thermal boundary layer has substantially lower strain rates and little internal deformation.

This mechanical and numerical behavior has essential components for the simulation of plate tectonics: large coherent blocks of the stiff thermal boundary layer that move with little internal strain (geologically, oceanic plates), low viscosity zones as the thermal boundary layer bends into a downwelling sheet (a subducted slab), and a high resolution mesh that continuously tracks this weak zone as the position of bending moves laterally (tracking the plate boundary).

The simulation required 19.2 million elements at 14 levels of octree refinement on 2400 Ranger cores. A mesh with uniform resolution requires 34 billion elements on level 13, and 275 billion on level 14. Even if we only compare with a uniform mesh on level 13, AMR enabled a more than 1,000-fold reduction in the number of unknowns and permitted a calculation on 2400 cores to be carried out in a reasonable time, while the uniform mesh would be too large to fit in the memory of all 62,976 cores of Ranger.

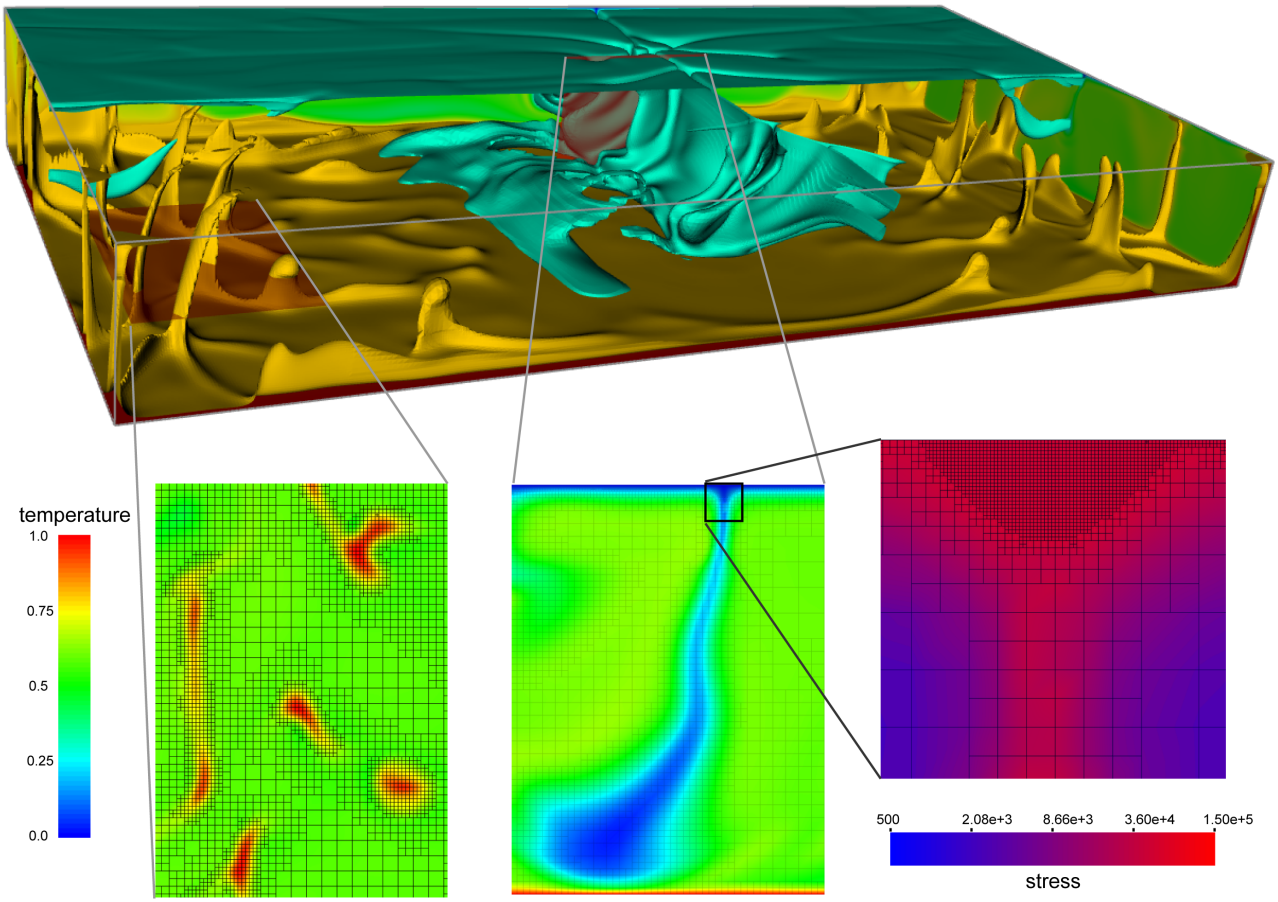

Fig. 11. Mantle convection with yielding. Top: Temperature isosurfaces at $T = 0.3$ (cyan), $T = 0.8$ (orange). Bottom left: Horizontal slice of temperature showing that grid refinement follows the temperature gradient. Bottom center: Vertical slice of temperature showing the downwelling slab and the yielding zone. Bottom right: A zoom-in of the yielding zone, where the finest grid of ~ 1.5 km resolution covers the region of highest stress.

VII. Extensions

In this section we discuss extensions of the hexahedral finite element, Cartesian octree-based AMR algorithms and data structures in ALPS, which were described in Section IV, to high-order spectral element discretizations and to general geometries via a forest-of-octrees decomposition. Whenever a domain can be decomposed into non-overlapping subdomains that are mappable to cubes (as is the case for spherical shell geometries that describe the mantle), we can treat each of the cubes as the root of an adaptive octree, thus creating a forest of (oc)trees. Due to the topological relations between these trees, their connecting faces involve transformations between the coordinate systems of each of the neighboring trees. A general-purpose adaptive code that implements this philosophy is deal.II [27]. That code, however, replicates the global mesh structure on each processor, which limits the scalability to a few hundred processors.

We have created the P4EST library, which implements a forest of arbitrarily connected parallel adaptive octrees. P4EST extends the $z-$order technique for parallel single octrees described in Section IV and in [15], [16], [26], [33] by a connectivity structure that defines the topological relations between neighboring octrees. As in the single-octree case, each core stores only the elements it owns, minimizing global information. The main issue is to enforce the 2:1 balance condition across faces, edges and corners, not only inside an octree but also between connected trees. This is complicated by the arbitrary number of trees that can share any particular edge or corner, each existing in a different coordinate system. We have scaled P4EST up to 32,768 cores and will report performance results elsewhere.

Exploiting the geometric flexibility of P4EST, we have built MANGLL, a high-order nodal discontinuous Galerkin (DG) discretization library for hexahedral elements, on top of the octree data structure. The implementation is an extension of Hesthaven and Warburton's

nodal DG framework [34] to hexahedral elements. The elements, also used in the spectral element method [35], have nodes at the tensor product of Legendre-Gauss-Lobatto (LGL) points. All integrations are performed using the LGL quadrature, which reduces the block diagonal DG mass matrix to a diagonal. The numerical flux is integrated on nonconforming interfaces between elements (where the mesh is refined) by introducing a face integration mesh that integrates the contributions from each smaller face individually using the two-dimensional tensor LGL quadrature.

In addition to offering greater accuracy per grid point, high order discretizations generally achieve substantially greater per-processor performance (because of the large dense elemental matrices they generate in conventional implementations) and greater parallel performance (due to the increased work for a given communication volume, since most of the additional degrees of freedom are on the interior of elements; and due to the smaller number of elements for a given number of unknowns). The dense matrices generated by high order methods are associated with application of the derivative operator at the element level. Explicit application of the element derivative matrix requires $6(p+1)^6$ flops, where p is the element polynomial order. This matrix-based implementation is extremely cache friendly, since the application can be done in one large matrix-matrix multiply. Alternatively, the tensor product structure of basis function can be exploited to carry out the same operation in $6(p+1)^4$ flops. While this tensor product-based implementation is asymptotically work-optimal, unfortunately it is not as cache friendly since it produces smaller matrices (one for each spatial dimension). Clearly, as the polynomial order increases, the tensor-product implementation will be preferable to the matrix implementation. The crossover point depends on how much faster large dense matrix operations execute on a given architecture. For the 2.3 GHz AMD Barcelona nodes on Ranger, and using the highly-optimized GotoBLAS library for matrix-matrix multiplications, we found the crossover point to occur between $p = 2$ and $p = 4$ for the scalar advection problem solved in this section. However, on other systems for which a wider gulf exists between execution rates for small and large dense element-level matrices—such as heterogeneous systems with attached accelerators, like Roadrunner—the crossover can be at a higher polynomial order. Thus, in this section, we give performance results for both matrix-based and tensor-product based implementations for a variety of polynomial orders.

To illustrate ALPS' MANGLL library for high-order discretization and P4EST library for forest-of-octree adaptivity on general geometries, we solve a pure advection problem on a spherical shell using high-order adaptivity to dynamically resolve an advecting spherical front. The arbitrary order nodal DG discretization is combined with upwind numerical fluxes to discretize the advection problem in space. A five-stage fourth-order explicit Runge-Kutta (RK) method is used to integrate the solution in time. Figure 12 shows snapshots from a solution of the advection equation using linear (i.e. $p = 1$) elements on 1024 cores. The spherical shell is split into 6 caps as usual in a cubed-sphere decomposition. Each cap consists of 4 octrees, resulting in 24 adaptive octrees overall. The figure shows a hemispherical slice for illustrative purposes. The right column shows the adapted mesh and associated temperature field for two time steps that are relatively close in time. Comparison of the top right and bottom right images shows that the mesh has adapted to the advecting temperature concentrations. The left column displays the partitioning of the mesh onto cores, where the color indicates MPI rank. As can be seen in the figure, the partitioning changes drastically from one time step to the next.

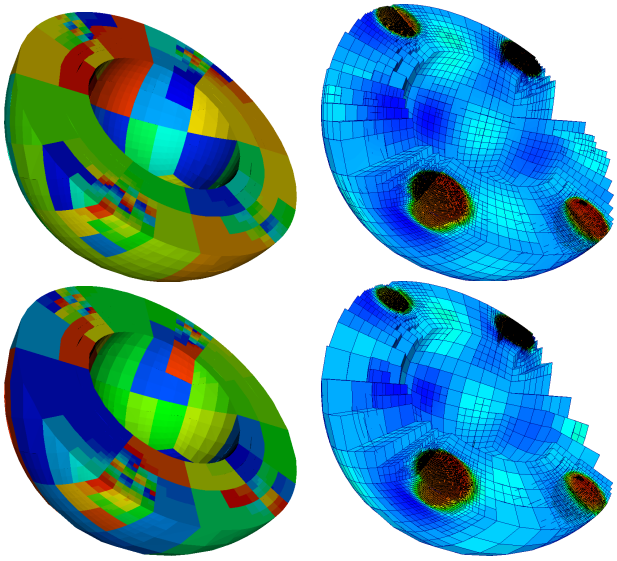

Fig. 12. Partitioning of spherical shell on 1024 cores for two neighboring time steps (left column) and corresponding adapted mesh with temperature field (right column).

Excellent performance is observed for high-order DG AMR solution of the advection equation. For example, for order $p = 4$, we observe 90% parallel efficiency on 16,384 cores relative to 64 cores, and for order $p = 6$ we found 83% parallel efficiency on 32,768 cores compared to 32 cores, both with adapting the mesh every 32 time steps. We next measure sustained floating point performance using PAPI for both the matrix- and tensor product-based implementations of the element derivative operator. For order $p = 2$, both approaches are within 15% of one another in runtime. For $p = 4$, the matrix approach is approximately 23% slower than

the tensor approach on 64, 512 and 16,384 cores. On the latter number of cores, the matrix version sustained 30 teraflops. Increasing the order to $p = 8$ on the same number of cores, the sustained rate increases to 71 teraflops. Finally, on 32,768 cores, order $p = 6$ sustains 100 teraflops, while order $p = 8$ sustains 145 teraflops. We stress that for these higher order discretizations, the less cache-friendly tensor product implementation of the element derivative operator executes at a substantially lower rate, yet still runs more quickly. For example, for $p = 6$ on 32K cores, the sustained rate is just 9.3 teraflops, but the tensor product implementation performs 20 times fewer flops, and therefore runs twice as fast as the matrix implementation. Clearly, the tensor implementation is the method of choice for higher polynomial order on Ranger. However, we provide the matrix-based performance numbers as indicators of performance on heterogeneous systems that can take even greater advantage of the matrix-based implementation's numerous large dense element-level matrix operations than a homogeneous cache-based system can.

VIII. Conclusions

Because of the complex data structures and large volumes of communication required, the scalability of dynamic AMR to tens of thousands of processors has long been considered a challenge. The main contributions of this paper have been to present ALPS, an AMR framework that uses parallel octree-based hexahedral finite element meshes and dynamic load balancing based on space-filling curves designed to scale to sustained petascale systems and beyond; and RHEA, an adaptive mantle convection code built on ALPS aimed at kilometer local resolution in global mantle convection simulations necessary to resolve faulted plate boundaries.

The main achievements of this paper can be summarized as follows:

- Scalability studies of ALPS on Ranger for the advection-dominated transport component of RHEA, which maximally stresses AMR, have demonstrated excellent weak and strong scalability on up to 62,464 cores and 7.9 billion elements. A speedup of over 100 is observed in strong scaling of a 531M element problem from 256 to 32,768 cores, while 50% parallel efficiency is maintained in isogranular scaling from 1 core to 62K cores. The total time consumed by all AMR components is less than 11% of the overall run time for the low-order discretized transport component, which provides few flops to amortize AMR over.
- For the full mantle convection simulation, which augments the explicitly-solved transport equation with the implicitly-solved nonlinear Stokes equations, the overhead of AMR is even smaller. In scalings to 16,384 cores, the cost of the AMR components is in fact negligible (less than 1%).
- The multiple orders of magnitude variation in viscosity is addressed by a parallel preconditioner for the nonlinear Stokes equation that exhibits a number of iterations that is essentially independent of problem size when scaling weakly from 1 to 8192 cores.
- High order spectral element discontinuous Galerkin AMR solution of the advection equation on spherical shells using the high order library MANGLL and the forest of octrees library P4EST from ALPS exhibits 90% parallel efficiency in scaling weakly from 64 to 16,384 cores.
- On 62K cores, AMR with a low order discretization for the transport component of RHEA sustains 36 teraflops, despite having very little numerical work within the (small) element kernels. For a high order DG discretization ($p = 8$), the matrix-based variant of MANGLL sustains 145 teraflops on 32K cores while carrying out full AMR. While the matrix-based implementation is not the method of choice for this order of polynomial, for Ranger, and for our current level of optimization, it is suggestive of the high performance that can be obtained even with full AMR on systems with dedicated hardware that can execute dense matrix operations at substantially higher rates than host processors.
- The mantle convection simulations of Section VI have used the AMR capabilities described in this paper to model plastic yielding at plate boundaries within large-scale mantle convection models with more than three orders of magnitude fewer elements, while achieving a resolution of about 1.5 km in the regions of yielding.

We stress that all results in this paper include all overheads due to mesh adaptivity—refinement, coarsening, rebalancing, repartitioning, field transfer, redistribution, and error indication. The results show that, with careful algorithm and data structure design and implementation, the cost of dynamic AMR can be made small relative to that of PDE solution on systems with tens of thousands of cores, even for meshes that change drastically over time.

Acknowledgments

This work was partially supported by NSF's PetaApps program (grants OCI-0749334, OCI-0749045, and OCI-0748898), NSF Earth Sciences (EAR-0426271), DOE Office of Science's SciDAC program (grant DE-FC02-06ER25782), DOE NNSA's PSAAP program (cooperative agreement DE-FC52-08NA28615), and NSF grants ATM-0326449, CCF-0427985, CNS-0540372, CNS-0619838, and DMS-0724746. We acknowledge many

helpful discussions with *hypre* developers Rob Falgout and Ulrike Yang, and with George Biros. We thank TACC for their outstanding support, in particular Bill Barth, Tommy Minyard, Romy Schneider, and Karl Schulz.

REFERENCES

[1] D. J. DePaolo, T. E. Cerling, S. R. Hemming, A. H. Knoll, F. M. Richter, L. H. Royden, R. L. Rudnick, L. Stixrude, and J. S. Trefil, "Origin and Evolution of Earth: Research Questions for a Changing Planet," National Academies Press, Committee on Grand Research Questions in the Solid Earth Sciences, National Research Council of the National Academies, 2008.

[2] D. Bailey, "The 1997 Petaflops Algorithms Workshop," *Computational Science & Engineering, IEEE*, vol. 4, no. 2, pp. 82–85, Apr-Jun 1997.

[3] D. P. McKenzie, J. M. Roberts, and N. O. Weiss, "Convection in the Earth's mantle: Towards a numerical solution," *Journal of Fluid Mechanics*, vol. 62, pp. 465–538, 1974.

[4] S. Zhong, M. T. Zuber, L. N. Moresi, and M. Gurnis, "Role of temperature-dependent viscosity and surface plates in spherical shell models of mantle convection," *Journal of Geophysical Research*, vol. 105, pp. 11 063–11 082, 2000.

[5] L. N. Moresi, M. Gurnis, and S. Zhong, "Plate tectonics and convection in the Earth's mantle: Toward a numerical simulation," *Computing in Science and Engineering*, vol. 2, no. 3, pp. 22–33, 2000.

[6] M. Gurnis, C. Hall, and L. Lavier, "Evolving force balance during incipient subduction," *Geochemistry, Geophysics, Geosystems*, vol. 5, no. 7, pp. 1–31, 2004.

[7] M. I. Billen, "Modeling the dynamics of subducting slabs," *Annu. Rev. Earth Planet. Sci.*, vol. 36, pp. 325–356, 2008.

[8] A. N. Brooks and T. J. R. Hughes, "Streamline upwind Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations," *Computer Methods in Applied Mechanics and Engineering*, vol. 32, pp. 199–259, 1982.

[9] T. J. R. Hughes, *The Finite Element Method*. New York: Dover, 2000.

[10] C. Dohrmann and P. Bochev, "A stabilized finite element method for the Stokes problem based on polynomial pressure projections," *International Journal for Numerical Methods in Fluids*, vol. 46, pp. 183–201, 2004.

[11] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics*. Oxford: Oxford University Press, 2005.

[12] C. C. Paige and M. A. Saunders, "Solution of sparse indefinite systems of linear equations," *SIAM Journal on Numerical Analysis*, vol. 12, no. 4, pp. 617–629, 1975.

[13] hypre. *High Performance Preconditioners, User Manual*, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, 2007, https://computation.llnl.gov/casc/linear_solvers/.

[14] H. De Sterck, U. M. Yang, and J. J. Heys, "Reducing complexity in parallel algebraic multigrid preconditioners," *SIAM Journal on Matrix Analysis and Applications*, vol. 27, no. 4, pp. 1019–1039, 2006.

[15] C. Burstedde, O. Ghattas, G. Stadler, T. Tu, and L. C. Wilcox, "Towards adaptive mesh PDE simulations on petascale computers," in *Proceedings of Teragrid '08*, 2008.

[16] T. Tu, D. R. O'Hallaron, and O. Ghattas, "Scalable parallel octree meshing for terascale applications," in *Proceedings of ACM/IEEE SC05*, 2005.

[17] L. F. Diachin, R. Hornung, P. Plassmann, and A. Wissink, "Parallel adaptive mesh refinement," in *Parallel Processing for Scientific Computing*, M. A. Heroux, P. Raghavan, and H. D. Simon, Eds. SIAM, 2006, ch. 8.

[18] P. Colella, J. Bell, N. Keen, T. Ligocki, M. Lijewski, and B. van Straalen, "Performance and scaling of locally-structured grid methods for partial differential equations," *Journal of Physics: Conference Series*, vol. 78, pp. 1–13, 2007.

[19] A. Calder, B. Curtis, L. Dursi, B. Fryxell, G. Henry, P. MacNeice, K. Olson, P. Ricker, R. Rosner, F. Timmes, H. Tufo, J. Truran, and M. Zingale, "High-performance reactive fluid flow simulations using adaptive mesh refinement on thousands of processors," in *Proceedings of ACM/IEEE SC00*, 2000.

[20] P. MacNeice, K. M. Olson, C. Mobarry, R. de Fainchtein, and C. Packer, "PARAMESH : A parallel adaptive mesh refinement community toolkit," *Computer Physics Communications*, vol. 126, pp. 330–354, 2000.

[21] A. M. Wissink, D. A. Hysom, and R. D. Hornung, "Enhancing scalability of parallel structured AMR calculations," in *Proceedings of the International Conference on Supercomputing 2003 (ICS'03)*, San Francisco, CA, June 2003, pp. 336–347.

[22] D. J. Mavriplis, M. J. Aftosmis, and M. Berger, "High resolution aerospace applications using the NASA Columbia Supercomputer," in *Proceedings of ACM/IEEE Supercomputing 2005*. Washington, DC, USA: IEEE Computer Society, 2005, p. 61.

[23] C. D. Norton, G. Lyzenga, J. Parker, and R. E. Tisdale, "Developing parallel GeoFEST(P) using the PYRAMID AMR library," NASA Jet Propulsion Laboratory, Tech. Rep., 2004.

[24] J. E. Flaherty, R. M. Loy, M. S. Shephard, B. K. Szymanski, J. D. Teresco, and L. H. Ziantz, "Adaptive local refinement with octree load balancing for the parallel solution of three-dimensional conservation laws," *Journal of Parallel and Distributed Computing*, vol. 47, no. 2, pp. 139–152, 1997.

[25] J.-F. Remacle and M. Shephard, "An algorithm oriented mesh database," *International Journal for Numerical Methods in Engineering*, vol. 58, pp. 349–374, 2003.

[26] H. Sundar, R. S. Sampath, and G. Biros, "Bottom-up construction and 2:1 balance refinement of linear octrees in parallel," *SIAM Journal on Scientific Computing*, vol. 30, no. 5, pp. 2675–2708, 2008.

[27] W. Bangerth, R. Hartmann, and G. Kanschat, "deal.II — A general-purpose object-oriented finite element library," *ACM Transactions on Mathematical Software*, vol. 33, no. 4, p. 24, 2007.

[28] B. Kirk, J. W. Peterson, R. H. Stogner, and G. F. Carey, "`libMesh`: A C++ library for parallel adaptive mesh refinement/coarsening simulations," *Engineering with Computers*, vol. 22, no. 3–4, pp. 237–254, 2006.

[29] L. Demkowicz, J. Kurtz, D. Pardo, M. Paszyński, W. Rachowicz, and A. Zdunek, *Computing with hp Finite Elements II. Frontiers: Three-Dimensional Elliptic and Maxwell Problems with Applications*. CRC Press, Taylor and Francis, 2007.

[30] A. Laszloffy, J. Long, and A. K. Patra, "Simple data management, scheduling and solution strategies for managing the irregularities in parallel adaptive hp-finite element simulations," *Parallel Computing*, vol. 26, pp. 1765–1788, 2000.

[31] J.-L. Fattebert, R. Hornung, and A. Wissink, "Finite element approach for density functional theory calculations on locally-refined meshes," *Journal of Computational Physics*, vol. 223, pp. 759–773, 2007.

[32] S. Zhong, "Dynamics of thermal plumes in three-dimensional isoviscous thermal convection," *Geophysical Journal International*, vol. 162, pp. 289–300, 2005.

[33] C. Burstedde, O. Ghattas, G. Stadler, T. Tu, and L. C. Wilcox, "Parallel scalable adjoint-based adaptive solution for variable-viscosity Stokes flows," *Computer Methods in Applied Mechanics and Engineering*, 2008, accepted for publication.

[34] J. S. Hesthaven and T. Warburton, *Nodal Discontinuous Galerkin Methods*, ser. Texts in Applied Mathematics. Springer, 2008, no. 54.

[35] M. Deville, P. Fischer, and E. Mund, *High-Order Methods for Incompressible Fluid Flow*, ser. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, 2002, no. 9.