Comput. Methods Appl. Mech. Engrg. 198 (2009) 1691–1700

Contents lists available at ScienceDirect

Comput. Methods Appl. Mech. Engrg.

journal homepage: www.elsevier.com/locate/cma

# Parallel scalable adjoint-based adaptive solution of variable-viscosity Stokes flow problems

Carsten Burstedde [a], Omar Ghattas [a,b,*], Georg Stadler [a], Tiankai Tu [a], Lucas C. Wilcox [a]

[a] *Institute for Computational Engineering & Sciences (ICES), The University of Texas at Austin, Austin, TX 78712, USA*
[b] *Jackson School of Geosciences and Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX 78712, USA*

A R T I C L E   I N F O

*Article history:*
Received 19 June 2008
Received in revised form 6 October 2008
Accepted 31 December 2008
Available online 7 January 2009

*Keywords:*
Adaptive mesh refinement
Stokes equations
Parallel computing
Octree algorithms
Adjoint error estimation
Algebraic multigrid

A B S T R A C T

We present a framework for parallel adaptive solution of variable-viscosity Stokes flow problems. We focus on data structures, algorithms, and solvers that can scale to thousands of processor cores. The problem is discretized by octree-based finite elements with explicit enforcement of continuity constraints at hanging nodes. The parallel octree structure allows for fast neighbor-finding and facilitates local coarsening and refinement of the mesh. Mesh adaptivity is driven by *a posteriori* error indicators, including adjoint-based goal-oriented techniques. Dynamic load-balancing is achieved by dynamically partitioning a Morton-ordered space-filling curve. The Stokes system is solved iteratively using the minimum residual method (MINRES), preconditioned by a Schur-complement-based approximate inverse that employs algebraic multigrid V-cycle approximations of the inverses of the Poisson-like operators. We demonstrate the effectiveness of this framework on several testbed problems with up to 6 orders of magnitude variation in viscosity and up to 1.7 billion unknowns, on up to 4096 cores. The results indicate that the overhead due to all AMR components is less than 3% of the overall solve time, the solver exhibits very good algorithmic and parallel implementation scalability, the solver is insensitive to the magnitude of viscosity variation, and adjoint-based adaptivity results in over two orders of magnitude reduction in number of unknowns and up to an order of magnitude improvement in runtime relative to a uniform mesh, for the same level of error.

Published by Elsevier B.V.

## 1. Introduction

Variable-viscosity Stokes equations play an important role in models of creeping flows arising in several geophysical areas, including magma migration [21], mantle convection [22,32], and ice sheet dynamics [18]. In many cases, the presence of local fine-scale features requires adaptive mesh refinement/coarsening (AMR) to make the Stokes flow models tractable. Furthermore, even with the use of AMR, problem sizes are often so large that solution on multi-thousand processor supercomputers is necessary, which typically presents difficulty for AMR algorithms. Moreover, viscosities can vary by several orders of magnitude, posing challenges for popular scalable solvers.

Here, we present a framework for parallel adaptive finite element solution of variable-viscosity Stokes equations that scales to thousands of processor cores. The framework combines ALPS, our library for parallel octree-based AMR with a multigrid-preconditioned Krylov method for the solution of variable-viscosity Stokes systems. At each cycle, the Stokes equations are solved on the current mesh, and *a posteriori* error estimates are used to mark elements for refinement. After refinement, the solution on the old mesh is interpolated to the new mesh and used as initialization for the next Stokes solve.

Our goal is to achieve scalability and performance for the entire adaptive process. This requires efficient mathematical methods and careful design and implementation of algorithms. First, efficient and scalable algorithms for error estimation, local refinement, and repartitioning of meshes are needed. Ideally, the time needed for AMR components should remain small compared to solver time, so that the gains accrued from having fewer degrees of freedom are not offset by inefficiencies of the algorithms for adaptivity. Second, the numerical components of the discretization and solver must be constructed carefully so that we achieve optimal (or nearly optimal) algorithmic scalability. This results when the number of iterations of the solver remains nearly constant as the mesh is refined or the viscosity variation increases. Third, algorithms for the discretization, Stokes solver, and AMR components must be implemented with careful attention to parallel scalability.

* Corresponding author.
*E-mail addresses:* carsten@ices.utexas.edu (C. Burstedde), omar@ices.utexas.edu (O. Ghattas), georgst@ices.utexas.edu (G. Stadler), ttu@ices.utexas.edu (T. Tu), lucasw@ices.utexas.edu (L.C. Wilcox).

0045-7825/$ - see front matter Published by Elsevier B.V.
doi:10.1016/j.cma.2008.12.015

In this paper, we describe the design and construction of parallel algorithms for discretization, iterative solution, and adaptivity that are aimed at achieving high performance as well as algorithmic and parallel scalability. While here we discretize the Stokes equations using pressure-stabilized trilinear finite elements, our framework naturally accommodates higher-order finite elements. Mesh adaptation is driven by *a posteriori* error indicators [1,3,5,11,25], targeting either the global discretization error or certain quantities of interest. We use algebraic constraints on hanging nodes to impose continuity of the solution field across coarse-to-fine element transitions. This results in a conforming approximation, allowing the use of standard finite element ideas. The finite element discretization, stabilization, and adjoint-based goal-oriented error estimates we use are presented in Section 2. Our AMR library ALPS uses parallel octree-based hexahedral finite element meshes and dynamic load-balancing based on space-filling curves, and is described in Section 3. We use the iterative Krylov solver MINRES (minimum residual method) for the solution of the discrete Stokes saddle point problem. Preconditioning is carried out by approximate block factorization and algebraic multigrid V-cycle approximation (using *BoomerAMG* from the *hypre* [10] package) of the inverse of viscous and pressure Schur complement operators. The solver and preconditioner are described in Section 4. Finally, Section 5 presents the results of a number of tests involving two geophysical problems, one from mantle convection and another from magma migration. These tests are designed to assess the performance and scalability of the AMR and solver algorithms presented in this paper. These include tests of the overhead imposed by the AMR components, the algorithmic and parallel scalability of the solver, the sensitivity of the solver to the magnitude of viscosity variation, and the overall efficiency of the adaptive Stokes solver.

## 2. Stokes discretization and error estimates

We consider the stationary incompressible Stokes equations with variable-viscosity $\mu(\cdot) \geqslant \mu_0 > 0$ in $\Omega \subset \mathbb{R}^3$:

$$-\nabla \cdot \left(\mu(\nabla \mathbf{u} + \nabla \mathbf{u}^\top)\right) + \nabla p = \mathbf{f}, \tag{1a}$$
$$\nabla \cdot \mathbf{u} = 0, \tag{1b}$$

where $\mathbf{u} = (u_1, u_2, u_3)$ denotes the velocity vector, $p$ the pressure, and $\mathbf{f}$ a given vector body force. Moreover, we assume Dirichlet (i.e. zero-slip) boundary conditions on $\emptyset \neq \Gamma \subset \partial\Omega$ and homogeneous Neumann (i.e., traction-free) conditions elsewhere, i.e.,

$$\mathbf{u} = \mathbf{u_0} \quad \text{on } \Gamma, \tag{1c}$$
$$\left(\mu(\nabla \mathbf{u} + \nabla \mathbf{u}^\top) - pI\right)\mathbf{n} = \mathbf{0} \quad \text{on } \partial\Omega \setminus \Gamma. \tag{1d}$$

Note that the viscous term in (1a) is often replaced by the vector Laplacian, having the computational advantage that the velocity components are coupled only through the incompressibility condition (1b). While the two formulations are equivalent only for constant viscosity, in Section 4 we will employ the vector Laplacian as a preconditioner for the viscous term.

In this section, we describe a stabilized finite element discretization and derive *a posteriori* error estimates for (1). Then we describe how we use these estimates to iteratively adapt finite element meshes.

### 2.1. Discretization and stabilization

We begin by stating (1) in a mixed variational form. For $\mathbf{u}, \mathbf{v} \in \mathbf{V} := (H^1(\Omega))^3$ and $p, q \in W := L^2(\Omega)$ we introduce the bilinear forms $A(\cdot,\cdot) : \mathbf{V} \times \mathbf{V} \to \mathbb{R}$ and $B(\cdot,\cdot) : \mathbf{V} \times W \to \mathbb{R}$ defined by

$$A(\mathbf{u},\mathbf{v}) := \int_\Omega \frac{\mu}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^\top) : (\nabla \mathbf{v} + \nabla \mathbf{v}^\top)\,\mathbf{dx},$$
$$B(\mathbf{u},p) := \int_\Omega -(\nabla \cdot \mathbf{u})p\,\mathbf{dx}.$$

Defining the bilinear form $Q$ and the linear form $F$

$$Q([\mathbf{u},p],[\mathbf{v},q]) = A(\mathbf{u},\mathbf{v}) + B(\mathbf{v},p) + B(\mathbf{u},q), \quad F(\mathbf{v}) := \int_\Omega \mathbf{f} \cdot \mathbf{v}\,\mathbf{dx},$$

the mixed variational formulation of the Stokes equation becomes: Find $\mathbf{u} \in \mathbf{V}_{\mathbf{u}_0} := \{\mathbf{u} \in \mathbf{V} : \mathbf{u} = \mathbf{u}_0 \text{ on } \Gamma\}$ and $p \in W$ such that

$$Q([\mathbf{u},p],[\mathbf{v},q]) = F(\mathbf{v}) \quad \text{for all } (\mathbf{v},q) \in \mathbf{V}_0 \times W, \tag{2}$$

where $\mathbf{V}_0 := \{\mathbf{u} \in \mathbf{V} : \mathbf{u} = \mathbf{0} \text{ on } \Gamma\}$. It is well known that a solution to (2) exists and that it is unique if $\Gamma \neq \partial\Omega$, whereas it is unique only up to a constant for $p$ if $\Gamma = \partial\Omega$ [15]. In the latter case uniqueness can be enforced by restricting the pressure space to $p, q \in W_0 := \{u \in W : \int_\Omega u\,\mathbf{dx} = 0\}$.

We discretize the variational form (2) using the finite element method with a hexahedral mesh and $Q_1$-$Q_1$ elements, i.e., trilinear elements for both velocity and pressure. To be precise, we partition $\bar{\Omega} = \bigcup_{\Omega^e \in \mathscr{M}} \bar{\Omega}^e$ and define the finite element spaces

$$V^h := \{u \in C_0(\Omega) : u|_{\Omega^e} \in Q_1 \text{ for all } \Omega^e\}, \quad \mathbf{V}^h = (V^h)^3,$$

where $Q_1$ denotes the space of trilinear (i.e., linear in each variable) element functions. The corresponding spaces that incorporate Dirichlet boundary conditions are

$$\mathbf{V}^h_{\mathbf{u}_0} := \{\mathbf{u} \in \mathbf{V}^h : \mathbf{u} = \mathbf{u}_0 \text{ on } \Gamma\} \quad \text{and} \quad \mathbf{V}^h_0 := \{\mathbf{u} \in \mathbf{V}^h : \mathbf{u} = 0 \text{ on } \Gamma\}.$$

It is well known that this equal-order discretization does not satisfy the inf–sup (or Babuška–Brezzi) condition for stability of numerical methods for saddle point problems [15]. As a remedy, we employ the polynomial pressure stabilization from [14] (see also [6,15]). Here, one adds $L^2$ pressure projections on the element level to the mixed variational equation. Define for $p, q \in W$ the mesh-dependent bilinear form

$$C(p,q) := \int_{\Omega^e} \frac{1}{\mu}(p - \Pi p)(q - \Pi q)\,\mathbf{dx}, \tag{3}$$

where $\Pi : W \to P_0$ denotes the $L^2$ projection from $W$ onto the space $P_0$ of element-wise constant functions. This term is added to the left hand side of (2), resulting in the modified bilinear form

$$Q^h([\mathbf{u}^h,p^h];[\mathbf{v}^h,q^h]) := Q([\mathbf{u}^h,p^h];[\mathbf{v}^h,q^h]) + C(p^h,q^h). \tag{4}$$

for $\mathbf{u}^h, \mathbf{v}^h \in \mathbf{V}^h$ and $p^h, q^h \in V^h$. Thus, the discrete stabilized Stokes problem becomes: Find $(\mathbf{u}^h, p^h) \in \mathbf{V}^h_{\mathbf{u}_0} \times V^h$ such that

$$Q^h([\mathbf{u}^h,p^h],[\mathbf{v}^h,q^h]) = F(\mathbf{v}^h) \quad \text{for all } (\mathbf{v}^h,q^h) \in \mathbf{V}^h_0 \times V^h. \tag{5}$$

The implementation of the stabilization (3) corresponds to a simple modification of the element mass matrix. This modification guarantees that constants are in the null space of $C$ while penalizing the spurious modes in $p$. The method achieves optimal accuracy with respect to the solution regularity, i.e., linear convergence for the velocity in the $H^1$-norm and for the pressure in the $L^2$-norm; for proofs see [6,15]. Other appealing features of this stabilization are that it does not require the specification of a stabilization parameter, it leads to symmetric systems, and it is completely local to the element. The latter property is especially attractive for parallel implementations, since no communication is necessary (unlike macroelement-based stabilization [15]). Section 3 provides more details on our parallel implementation, in particular in the context of adaptive mesh refinement.

## 2.2. A posteriori error estimation

*A posteriori* error estimates can be used to manage the numerical error in finite element approximation by adaptively refining and coarsening finite element meshes. Often, one is not interested in minimizing error in a global norm, since this might not provide efficient control of the error in the quantities of interest. Adaptive methods based on goal-oriented error estimation reduce the error in these quantities of interest, often making them superior to adaptation based on global error indicators [1,5]. Generally, goal-oriented error estimation requires the solution of adjoint PDEs.

Below we derive a goal-oriented error indicator for the pressure-stabilized Stokes equation. At the end of the section we also give two simpler error indicators, the residual error indicator, which is based on the complete residual of the Stokes equation, and the divergence error indicator, which is based on the residual in the mass continuity equation only.

Note that the stabilization (3) leads to a non-standard Galerkin method, since in the discrete problem a term is added to the continuous bilinear form $A(\cdot,\cdot)$. Thus, Galerkin orthogonality for the errors $(\mathbf{e}_u, e_p) := (\mathbf{u}-\mathbf{u}^h, p-p^h)$ no longer holds, i.e.,

$$Q([\mathbf{e}_u, e_p],[\mathbf{v}^h, q^h]) = C(p^h, q^h) \text{ for } (\mathbf{v}^h, q^h) \in \mathbf{V}^h \times V^h, \tag{6}$$

which in general is nonzero. This slightly complicates the usual procedure for goal-oriented *a posteriori* error estimates. Hence, we briefly sketch the derivation of the error estimates below.

For simplicity, we assume the quantity of interest $J(\mathbf{u},p)$ is linear in velocity and pressure. To obtain weights for the residuals that drive the refinement, the following adjoint problem is defined: Find $(\mathbf{z},y) \in \mathbf{V}_0 \times W$ such that

$$Q([\mathbf{v},q];[\mathbf{z},y]) = J(\mathbf{v},q) \text{ for all } (\mathbf{v},q) \in \mathbf{V}_0 \times W. \tag{7}$$

Denoting again $(\mathbf{e}_u, e_p) = (\mathbf{u}-\mathbf{u}^h, p-p^h)$, we wish to minimize the error in the quantity of interest $|J(\mathbf{u},p) - J(\mathbf{u}^h,p^h)| = |J(\mathbf{e}_u,e_p)|$. Using (6) and the adjoint problem (7), we obtain for arbitrary $(\mathbf{z}^h,y^h) \in \mathbf{V}^h \times V^h$

$$J(\mathbf{e}_u,e_p) = Q([\mathbf{e}_u,e_p],[\mathbf{z},y]) \tag{8a}$$
$$= Q([\mathbf{e}_u,e_p],[\mathbf{z}-\mathbf{z}^h, y-y^h]) + C(p^h,y^h). \tag{8b}$$

In what follows, $(\mathbf{z}^h,y^h)$ are chosen as approximations of $(\mathbf{z},y)$ in the finite element spaces $\mathbf{V}_0^h$ and $V^h$. The error in the quantity of interest is further split such that

$$\begin{aligned} J(\mathbf{e}_u,e_p) &= Q([\mathbf{u},p],[\mathbf{z}-\mathbf{z}^h,y-y^h]) - Q([\mathbf{u}^h,p^h],[\mathbf{z}-\mathbf{z}^h,y-y^h]) \\ &\quad + C(p^h,y^h) \\ &= F([\mathbf{z}-\mathbf{z}^h,y-y^h]) - A(\mathbf{u}^h,\mathbf{z}-\mathbf{z}^h)s - B(\mathbf{z}-\mathbf{z}^h,p^h) \\ &\quad - B(\mathbf{u}^h,y-y^h) + C(p^h,y^h). \end{aligned}$$

Element-wise integration by parts for the second and third term yields

$$\begin{aligned} J(\mathbf{e}_u,e_p) = \sum_{\Omega^e\in\mathscr{M}} \Bigg\{ & \int_{\Omega^e} \left(\mathbf{f} + \nabla\cdot\frac{\mu}{2}(\nabla\mathbf{u}^h + (\nabla\mathbf{u}^h)^\top) - \nabla p^h\right)(\mathbf{z}-\mathbf{z}^h)\,\mathbf{dx} \\ & + \int_{\Omega^e} (\nabla\cdot\mathbf{u}^h)(y-y^h)\,\mathbf{dx} \\ & + \int_{\partial\Omega^e} \left(\frac{\mu}{2}(\nabla\mathbf{u}^h + (\nabla\mathbf{u}^h)^\top) - p^h\mathbf{I}\right)\mathbf{n}\cdot(\mathbf{z}-\mathbf{z}^h)\,\mathbf{dx} \\ & + \int_{\Omega^e} (p^h - \Pi p^h)(y^h - \Pi y^h)\,\mathbf{dx} \Bigg\} \\ = \sum_{\Omega^e\in\mathscr{M}} \Bigg\{ & \int_{\Omega^e} \mathbf{R}_1(\mathbf{u}^h,p^h)(\mathbf{z}-\mathbf{z}^h)\,\mathbf{dx} + \int_{\Omega^e} R_2(\mathbf{u}^h)(y-y^h)\,\mathbf{dx} \\ & + \int_{\partial\Omega^e} \mathbf{R}_3(\mathbf{u}^h,p^h)(\mathbf{z}-\mathbf{z}^h)\,\mathbf{dx} + \int_{\Omega^e} R_4(p^h)(y^h-\Pi y^h)\,\mathbf{dx} \Bigg\} \end{aligned}$$

with

$$\begin{aligned} \mathbf{R}_1(\mathbf{u}^h,p^h)_{|\Omega^e} &= \mathbf{f} + \nabla\cdot\frac{\mu}{2}(\nabla\mathbf{u}^h + (\nabla\mathbf{u}^h)^\top) - \nabla p^h, \\ R_2(\mathbf{u}^h)_{|\Omega^e} &= \nabla\cdot\mathbf{u}^h, \\ \mathbf{R}_3(\mathbf{u}^h)_{\partial\Omega^e} &= \frac{1}{2}\left[\left[\left(\frac{\mu}{2}(\nabla\mathbf{u} + (\nabla\mathbf{u})^\top) - p^h I\right)\mathbf{n}\right]\right], \\ R_4(p^h) &= \frac{1}{\mu}(p^h - \Pi p^h). \end{aligned}$$

Here, $[\![\cdot]\!]$ denotes the jump across faces of element boundaries. Note that the residuals $\mathbf{R}_1, R_2, R_4$ can be calculated for each element from its nodal values only, while $\mathbf{R}_3$ requires information from the element's face neighbors. The above expressions provide an estimate of the error in the quantity of interest in terms of the element error indicators $\eta^e$,

$$|J(\mathbf{e}_u,\mathbf{e}_p)| \leqslant \sum_{\Omega^e\in\mathscr{M}} \eta^e, \quad \text{with } \eta^e = \sum_{i=1}^{4} \rho_i^e \omega_i^e, \tag{9}$$

where

$$\begin{aligned} \rho_1^e &= \|\mathbf{R}_1(\mathbf{u}^h,p^h)\|_{\Omega^e}, & \omega_1^e &= \|\mathbf{z}-\mathbf{z}^h\|_{\Omega^e}, \\ \rho_2^e &= \|R_2(\mathbf{u}^h)\|_{\Omega^e}, & \omega_2^e &= \|y-y^h\|_{\Omega^e}, \\ \rho_3^e &= \|\mathbf{R}_3(\mathbf{u}^h,p^h)\|_{\partial\Omega^e}, & \omega_3^e &= \|\mathbf{z}-\mathbf{z}^h\|_{\partial\Omega^e}, \\ \rho_4^e &= \|R_4(p^h)\|_{\Omega^e}, & \omega_4^e &= \|y^h - \Pi y^h\|_{\Omega^e}. \end{aligned}$$

The additional term $\rho_4^e\omega_4^e$ stems from the absence of Galerkin orthogonality for the discretization errors (see (6)). Note that in computing the traction jump term $\mathbf{R}_3(\mathbf{u}^h,p^h)$, the pressure $p^h$ can be neglected since it is continuous across element faces.

Given a finite element solution $(\mathbf{u}^h,p^h)$, the residuals $\rho_i^e$ can be computed according to the expressions above. However, the weights $\omega_i^e$ involve the continuous adjoint solution $(\mathbf{z},y)$ and thus their evaluation requires sufficiently accurate approximate solution of the adjoint problem. Using the same discretization and mesh as used in the primal problem fails, since this leads to zero weighs $\omega_i^e$. Hence, one needs to employ global or local higher-order approximations: global approximations are based on solutions on a finer mesh or higher-order finite elements; local higher-order approximation can be obtained, for instance, by patch-wise higher-order interpolation. In our numerical tests, we simply solve the adjoint problem on a mesh obtained by one global refinement of the primal mesh.

Cheaper-to-compute error indicators attempt to decrease global norms of the error and do not require the solution of an adjoint equation, but are usually less effective for quantities of interest than goal-oriented techniques. These indicators can be recovered from (9) by choosing appropriate weights: If all $\omega_i^e$ $(i = 1,\dots,4)$ in (9) are chosen equal to 1, the residual error indicator is obtained. Similarly, for the divergence error indicator one sets $\omega_1^e = \omega_3^e = \omega_4^e = 0$ and $\omega_2^e = 1$ for all elements $\Omega^e \in \mathscr{M}$.

## 2.3. Mesh adaptation based on a posteriori error indicators

Element-wise *a posteriori* error indicators can be used to successively adapt finite element meshes to more effectively resolve physical phenomena of varying spatial scales. A typical nested-iteration cycle for mesh adaptation, known as a *Solve–Estimate–Mark–Refine* cycle, is given by:

1. Choose an initial mesh $\mathscr{M}_0$, a maximum element error $\eta_{\max}$, a maximum number of refinement cycles $k_{\max}$ and set $k = 0$.
2. **Solve**: Compute the finite element solution $(\mathbf{u}_k^h, p_k^h)$ of (5) on the mesh $\mathscr{M}_k$.

3. **Estimate**: Compute the error indicator $\eta^e$ for each element $\Omega^e \in \mathscr{M}_k$ and stop if for all elements $\eta^e \leqslant \eta_{\max}$.
4. **Mark**: Mark elements that have large indicators $\eta^e$ for refinement.
5. **Refine**: Refine the marked elements to obtain a new mesh $\mathscr{M}_{k+1}$.
6. If $k < k_{\max}$ let $k := k + 1$ and go to Step 2.

In Step 4 several marking strategies are possible. For example, we may wish to refine all elements with an error indicator larger than a given threshold $\eta_{\max}$. Another strategy is to refine the $\alpha\%$ of elements with the largest error indicators, for $\alpha \in (0, 100)$. Note that the latter strategy requires communication since the error indicators are available locally only. In practice, $\eta_{\max}$, $\alpha$, and $k_{\max}$ are chosen with a final mesh size and a target number of cores in mind so that the refinement process does not exceed the available memory.

## 3. Parallel octree-based mesh adaptation and load-balancing

In this section, we describe the essential components of ALPS. The design of our library supports many mesh-based PDE discretization schemes, such as low- and high-order variants of finite element, finite volume, spectral element, and discontinuous Galerkin methods, though only finite element methods on trilinear hexahedral elements are currently implemented. We build on prior approaches to parallel octree mesh generation [30,31], and extend them to accommodate solution-adaptive refinement (and coarsening). This requires separating the octree from the mesh data structures. Specifically, adaptation and partitioning of the mesh are handled through the octree structure, and a distinct mesh is generated from the octree every time the mesh changes.

Nonconforming hexahedral meshes of a given rectangular domain are generated for use with a trilinear finite element discretization. Solution fields are made conforming via algebraic continuity constraints on hanging nodes, that is, nodes on edges and faces that are not vertices of all the elements sharing those edges or faces. These algebraic constraints are eliminated at the element level, so variables at the hanging nodes are no longer degrees of freedom for the solver. We maintain a global 2-to-1 balance condition, i.e., the edge lengths of face- and edge-neighboring elements may differ by at most a factor of 2. This ensures smooth gradations in mesh size, and simplifies the incorporation of algebraic constraints. Octree-based refinement/coarsening of hexahedral finite element meshes with hanging node constraints has been employed in such parallel finite element libraries as deal.II [4], libMesh [19], *hp*3d [11], and AFEAPI [20], and have been demonstrated to scale to well to hundreds of processors. Here, our focus is on parallel algorithms and implementations that can scale to $\mathcal{O}(10^4)$ cores. These are discussed in the remainder of this section.

### 3.1. 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 elements of the mesh (see Fig. 1, left). 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, see also [8].

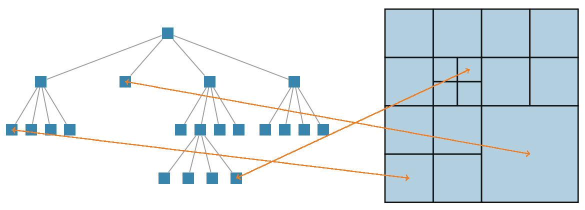
Most of the AMR functions in ALPS operate on the octree from which the mesh is generated. Since we target large parallel systems, we cannot store the full octree on each core. Thus, the tree is partitioned across cores. As we will see below, cores must be able to determine which core owns a given leaf octant. To this end we rely on a space-filling curve [2,9,12], 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. This is the only global information that is required to be stored. 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 the right of Fig. 1.

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 [31] for details. The inverse of this operation, determining the element corresponding to a given leaf, can be made efficient as well.

### 3.2. Mesh generation and adaptation

The generation of the mesh comprises several distinct steps. There are two scenarios in which a mesh is generated: the first is the initial generation of the mesh, and the second is the generation of a mesh from an adapted octree. As we will see, the adaptation of the mesh in conjunction with the transfer of data fields requires an intermediate mesh to be generated.

When generating a mesh from an adapted octree, the interpolation of element fields between old and new meshes necessitates additional functions. The procedure for adapting the mesh proceeds as follows. First, a given octree is coarsened and refined



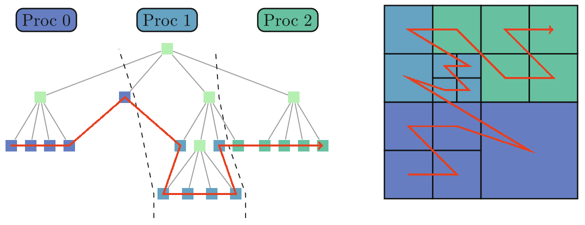


**Fig. 1.** Left: illustration of the distinct octree and mesh data structures used in ALPS. The data structures are linked logically by a 1-to-1 correspondence between octree leaves and elements. Right: a pre-order traversal of the leaves of the octree in the sequence of triples $(z, y, x)$ creates a space-filling curve in $z$-order. This imposes a total ordering of the mesh elements, known as a Morton ordering. A load-balanced partition of the octree is determined by partitioning the space-filling curve into segments of equal length. The globally shared information required for this operation amounts to one long integer per core. Note that in both figures a quadtree is show for display purposes.

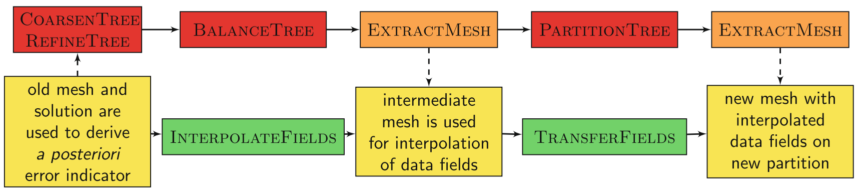

CoarsenTree
RefineTree
BalanceTree
ExtractMesh
PartitionTree
ExtractMesh
old mesh and solution are used to derive *a posteriori* error indicator
InterpolateFields
intermediate mesh is used for interpolation of data fields
TransferFields
new mesh with interpolated data fields on new partition

**Fig. 2.** Functions for mesh adaptation. Red boxes correspond to functions that operate on the octree only; orange boxes denote functions that act between the octree and the mesh; mesh and data field operations are enclosed in yellow boxes; green boxes are used for 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. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

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 following their associated leaf partition. Fig. 2 illustrates this process.

### *3.3. AMR functions*

Below we highlight the key features of the functions used to build and adapt the octree and mesh in an application code. In this paper we use the functionality of the ALPS library to adaptively refine the mesh for static Stokes equations. Due to its coarsening capabilities, the library can also be used for dynamic AMR simulations [8].

**NewTree**. This algorithm is used to construct a new octree in parallel. Each core grows an octree to an initial coarse level, which is 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 COARSENTREE and REFINETREE work directly on the octree and are completely local operations that require no communication. REFINETREE traverses the leaves of the local partition of the octree on each core, 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 is a minor restriction, since the number of such leaf sets is at most one less than the number of cores. Both COARSENTREE and REFINETREE work recursively; that is, multiple levels of leaves can be removed or added in one invocation of the function.

**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 [31]. 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 has to be performed frequently throughout a simulation as the mesh is adapted. The goal of this function 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 way to partition 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 be found 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 orderings 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 an existing 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**. The way this function works on the data fields is similar to the way PARTITIONTREE works on the octree. Following the Morton ordering among the degrees of freedom, the data associated with element nodes are transferred between cores to complete the partitioning 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.

## 4. Numerical solution of the discrete stokes system

In this section, we present our solver for the solution of the discrete form of the Stokes equation, i.e. (5). The resulting discrete Stokes problem can be written as the following saddle point problem:

$$Q\begin{pmatrix}\hat{\mathbf{u}}\\ \hat{\mathbf{p}}\end{pmatrix}=\begin{pmatrix}\hat{\mathbf{f}}\\ 0\end{pmatrix}\quad\text{with } Q=\begin{pmatrix}A & B^{\top}\\ B & -C\end{pmatrix},\tag{10}$$

where $\hat{\mathbf{u}}, \hat{\mathbf{p}}$ and $\hat{\mathbf{f}}$ denote the coefficient vectors for the functions $\mathbf{u}^h, p^h$ and $\mathbf{f}^h$ and the matrices $A, B$ and $C$ correspond to the bilinear forms $A(\cdot,\cdot)$, $B(\cdot,\cdot)$ and $C(\cdot,\cdot)$, respectively. The blocks $A$ and $C$ are symmetric and positive definite and, thus, (10) is an indefinite symmetric system.

### *4.1. Iterative solution by Krylov method*

Since the coefficient matrix $Q$ is symmetric and indefinite, we employ the preconditioned minimum residual method (MINRES) [28] for its solution. MINRES is a generalization of the conjugate gradient method to indefinite systems. Each MINRES iteration requires one application of the matrix $Q$ to a vector, two inner products, and storage of two vectors. Each inner product requires a collective reduction operation.

### 4.2. Choice of preconditioner

To obtain a mesh-independent (or almost mesh-independent) number of iterations, i.e., a constant number of iterations as the problem size increases, one needs to employ a suitable preconditioner for (10). Note that MINRES requires a symmetric and positive definite preconditioner. The block factorization

$$\begin{pmatrix} A & B^\top \\ B & -C \end{pmatrix} = \begin{pmatrix} I & 0 \\ BA^{-1} & I \end{pmatrix}\begin{pmatrix} A & 0 \\ 0 & -(BA^{-1}B^\top + C) \end{pmatrix}\begin{pmatrix} I & A^{-1}B^\top \\ 0 & I \end{pmatrix} \tag{11}$$

shows that $Q$ is congruent to a block diagonal matrix. Neglecting the off-diagonal terms $A^{-1}B$ on the right hand side of (11) motivates the use of the symmetric and positive definite matrix

$$P = \begin{pmatrix} A & 0 \\ 0 & S \end{pmatrix}, \quad \text{with } S = BA^{-1}B^\top + C \tag{12}$$

as preconditioner. However, since the Schur complement $S$ involves $A^{-1}$, systems of the form $P\hat{\mathbf{z}} = \hat{\mathbf{r}}$ cannot be solved easily, which makes $P$ unsuitable as a preconditioner. Thus, we replace the Schur complement $S$ by a lumped mass matrix (e.g., [17]) weighted by the inverse viscosity $\mu^{-1}$. For instance in [15] it is shown that in the case of constant viscosity the resulting diagonal matrix is spectrally equivalent to $S$. For varying viscosity and interface Stokes problems, similar results are obtained in [27,26]. Note that, when lumped, the pressure stabilization matrix $C$ drops out. This is due to the fact that at the element level, constants are in the null space of $C$. The resulting diagonal matrix $\widetilde{M}$ reflects the local element size as well as the local viscosity. This is essential for favorable scalability of the MINRES iterations as the problem grows, and is particularly important for adaptively refined meshes.

To reduce the cost of the preconditioner in (12), we replace the $3 \times 3$ block matrix $A$ in (12) by the discrete vector Laplacian with variable viscosity for the preconditioner. For constant viscosity and Dirichlet boundary conditions, $A$ is equivalent to the vector Laplacian, which motivates this replacement. Thus, the preconditioner reduces to

$$\widetilde{P} = \begin{pmatrix} \widetilde{A} & 0 \\ 0 & \widetilde{M} \end{pmatrix}, \quad \text{with } \widetilde{A} = \begin{pmatrix} L_1 & 0 & 0 \\ 0 & L_2 & 0 \\ 0 & 0 & L_3 \end{pmatrix}. \tag{13}$$

Here, for $i = 1,2,3$ the block matrix $L_i$ denotes the discretization of the bilinear form $L_i(u_i, v) := \int_\Omega \mu \nabla u_i \cdot \nabla v \, dx$, taking into account possibly different boundary conditions for $u_1, u_2, u_3$. Note that due to the block diagonal structure of $\widetilde{P}$, $u_1, u_2, u_3$, and $p$ are decoupled and thus each component of $\widetilde{P}\hat{\mathbf{z}} = \hat{\mathbf{r}}$ can be solved independently.

### 4.3. Implementation of the preconditioner by algebraic multigrid (AMG)

While a solve with the lumped mass matrix $\widetilde{M}$ is trivial, $L_1, L_2$ and $L_3$ are discretizations of Poisson operators on highly heterogeneous meshes with large variations in the viscosity $\mu$. To approximately calculate $L_i^{-1}\hat{\mathbf{r}}_i$ for given $\hat{\mathbf{r}}_i$, we use one V-cycle of an algebraic multigrid (AMG) method (e.g. [7]). Compared to geometric multigrid, AMG can have advantages due to its ability to account for variations in viscosity and adaptively refined meshes in the grid hierarchy. AMG requires a setup phase, in which a coarse grid hierarchy and corresponding restriction and interpolation operators are constructed. Parallel implementations of AMG require communication for this setup step. Generally, there is a trade-off between increased time/memory and the effectiveness of the coarse grid hierarchy. For our tests we use the parallel AMG implementation *BoomerAMG* from the *hypre* package [10,13,16]. *BoomerAMG* allows the user to choose among various coarsening strategies, and to set parameters that influence the complexity of the coarse grid hierarchy and the interpolation and restriction operators. The settings we used in our test are summarized in the next section.

## 5. Numerical results

In this section, we study the parallel performance and scalability of the parallel adaptive mesh refinement method and variable-viscosity Stokes solver described in the previous sections. All of our tests are performed on Ranger, the 504 teraflops, 62,976-core Sun/AMD parallel supercomputer at the Texas Advanced Computing Center (TACC). Each compute core of Ranger has a 2.0 GHz clock rate and 2 GB of memory.

In our tests, we assess isogranular (or weak) scalability, i.e., we simultaneously increase the problem size and the number of cores while keeping the problem size per core constant. Since the problem size grows as the core count increases, isogranular scaling stresses not only the parallel implementation but also the algorithmic scalability. For a Krylov solver, optimal algorithmic scalability requires that the work per Krylov iteration, as well as the number of iterations, remains constant as the problem size increases. This property is of course predicated on effective and cheap-to-apply preconditioners.

Two geophysical test problems are used in this section to study isogranular and algorithmic scalability. The first is motivated by simulation of convection within Earth's mantle. We study the parallel performance and scaling of the adaptive Stokes solver and its dependence on the viscosity variation. Moreover, we show that the overhead due to parallel mesh refinement is negligible. The second test problem is a benchmark Stokes flow problem relevant to magma dynamics. Here the Stokes flow field is driven by velocity boundary conditions that represent diverging tectonic plates. The solver is nested within the adaptive refinement loop described in Section 2.3. Adaptive refinement is controlled by several different *a posteriori* error indicators, including the adjoint estimators developed in Section 2.2. Since a semi-analytical spectral solution for this benchmark is available, we are also able to study convergence of finite element approximations of a quantity of interest functional as a function of the number of degrees of freedom and the overall run time of the code.

Before presenting the examples, we summarize the settings used in our numerical tests. Unless otherwise specified, the MINRES iteration is terminated when the residual drops by a factor of $10^6$ relative to the initial residual. Note that the residual occurring naturally in the preconditioned MINRES algorithm is $\sqrt{\hat{\mathbf{r}}^\top \widetilde{P}^{-1}\hat{\mathbf{r}}}$ rather than the equation residual $\|\hat{\mathbf{r}}\| = \sqrt{\hat{\mathbf{r}}^\top\hat{\mathbf{r}}}$. Table 1 summarizes the settings used in *BoomerAMG*. The complexity of the grid hierarchy is controlled by the choice of the basic coarsening algorithm and the parameters for truncation and interpolation. For a detailed description of how these settings influence the AMG setup phase

**Table 1**
Settings used in *BoomerAMG* from the *hypre* package. We use the parallel coarsening method PMIS, extended interpolation, and a maximum of 5 matrix entries per row for the interpolation matrices. The truncation factor and the threshold for strong matrix connections influence the complexity of the grid hierarchy.

| Coarsening | Interpolation | Truncation factor | Strong threshold | Max entries per row for interp. |
|---|---|---|---|---|
| PMIS | Extended | 0.3 | 0.5 | 5 |

and the residual reduction rate of the AMG solver we refer to [10,13].

### 5.1. Example 1: mantle convection

Mantle convection is the principal driving mechanism for the thermal and geological evolution of the Earth's surface. The dynamics of mantle convection are governed by equations for the balance of mass, linear momentum, and energy, e.g., [32]. A simplified model is given by a time-dependent advection–diffusion equation for temperature, coupled with a stationary Stokes equation with temperature-dependent, and hence spatially-variable, viscosity. Using an operator splitting approach, the Stokes problem is solved at each time step, given the temperature field. This yields an updated velocity field for the advection–diffusion equation. Typically, the Stokes problems are characterized by viscosities that vary by $10^3$ to $10^7$ orders of magnitude. Moreover, to resolve the wide range of spatial scales frequently encountered, adaptively refined meshes are often required.

In this section, we study the Stokes solver for a model problem of rising thermal blob (see Fig. 3, left). The domain is $\Omega = [0,1]^3$ and we use free-slip boundary conditions, i.e., zero normal velocity and zero tangential traction. The right hand side $\mathbf{f}$ and the viscosity $\mu$ depend on the temperature field $T(x,y,z) := \exp(-\beta((x-0.5)^2 + (y-0.5)^2 + (z-0.2)^2))$ as

$$\mathbf{f} = (0, 0, 10^6 T), \quad \mu = \exp(-\alpha T). \tag{14}$$

The constants $\alpha, \beta \geqslant 0$ above are used to control the viscosity variation. With the exception of the cases reported in Table 4, we use $\alpha = 7.5$ and $\beta = 200$, which results in a viscosity contrast of approximately $5 \times 10^3$.

Table 2 shows the time needed for the Stokes solver and all AMR components as the problem size and number of cores are scaled in isogranular fashion. Each case is initialized on a uniform mesh with 32.7 K elements per core. We perform three mesh refinement cycles, as follows. At each cycle, the Stokes problem is solved and a global error estimator is used to refine the 7% of elements with the largest error. After refining the mesh, the coarser mesh solution is interpolated onto the refined mesh, and used as an initial guess for the MINRES iterative solver on the refined mesh. After three refinements, this results in a mesh with approximately 110 K elements per core. On this final mesh, which contains four sizes of elements, a final Stokes solve is performed. Table 2 shows that for a range from 1 to 4096 cores, *all* AMR components (including error estimation, marking/refinement, mesh extraction, 2:1 balance condition enforcement, interpolation and solution transfer, and repartitioning of the mesh) consume less than 3% of the overall solve time. The most costly AMR components are the mesh extraction algorithm, in which the finite element mesh is constructed from the octree, and the repartitioning of the mesh among the cores, which is needed for load-balancing. Nevertheless, despite the large communication volumes required by these components, they require negligible time relative to the solver. Of course, one could always make the AMR components look good by employing a poor solver. The next two tables demonstrate that this is not the case: the solver has nearly-ideal algorithmic scaling and insensitivity to viscosity variation.

To analyze the isogranular scalability of the solver, Table 3 provides a breakdown of the timings for the Stokes solve on the final (i.e., the three-times-refined) mesh. To make the results independent of the solutions on the coarser meshes, for this test we initialize the MINRES iteration with a zero solution. The table reports the number of MINRES iterations as well as the time needed for the AMG setup, MINRES solve excluding the preconditioner (which is dominated by a matrix–vector product), and V-cycle preconditioner. The number of MINRES iterations is seen to be almost insensitive to a 4096-fold increase in number of degrees of freedom. The AMG setup time is the time used by *BoomerAMG* to construct the coarse grid hierarchy and the interpolation operators. Due to the decoupling of the velocity components in the preconditioner,

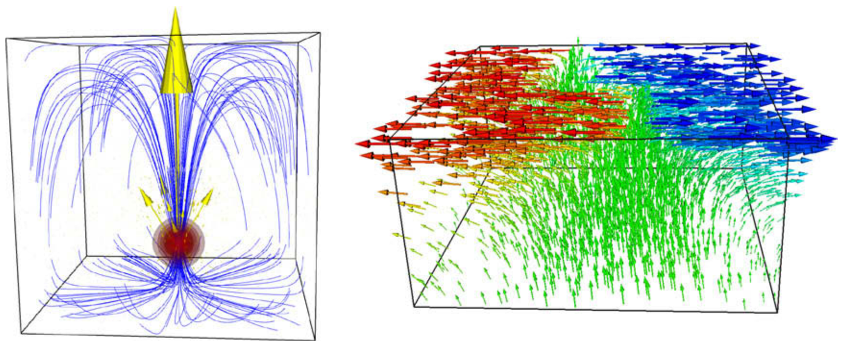


**Fig. 3.** Left: thermal blob and streamlines for Example 1. Right: velocity field for Example 2.

**Table 2**
Timings (in seconds) for adaptive solution of Example 1 (mantle convection) problem for isogranular (weak) scaling. Problem size increases with number of cores, maintaining 32.7 K elements per core. The mesh undergoes three refinements, beginning from a uniform coarse mesh. At each refinement step, the Stokes system is solved and the mesh is refined based on the global *a posteriori* error indicator. The table gives the total time taken by the Stokes solver and by the different AMR components. Columns 3–8 report the time taken for the complete AMR process, i.e. for error estimation, marking and refining elements, extracting the new mesh, 2:1-balancing of the octree, interpolation and transfer of the solution fields to the new mesh, and repartitioning of the octree. The last column shows the percentage of overall time spent in AMR components relative to the solve time, which is less than 3% in all cases.

| # Cores | Solver time | Error estimate | Mark & refine | Extract mesh | Balance tree | Interp. & transfer | Partition tree | $\frac{\text{AMR time}}{\text{solve time}}$ (%) |
|---|---|---|---|---|---|---|---|---|
| 1 | 345.6 | 1.78 | 0.08 | 2.05 | 0.12 | 0.13 | 0.00 | 1.2 |
| 8 | 374.8 | 2.29 | 0.22 | 3.38 | 0.27 | 0.16 | 1.77 | 2.2 |
| 64 | 497.6 | 2.66 | 0.36 | 6.21 | 1.00 | 0.22 | 2.51 | 2.6 |
| 512 | 696.5 | 2.89 | 0.84 | 9.64 | 2.05 | 0.43 | 3.26 | 2.8 |
| 4096 | 1095.8 | 3.04 | 1.41 | 10.44 | 2.39 | 0.64 | 10.92 | 2.6 |

**Table 3**
Isogranular (weak) scaling of the solver for Example 1 with varying viscosity $\mu$ on the triply-adapted mesh. The number of cores, number of degrees of freedom, number of MINRES iterations, AMG setup time, MINRES iteration time excluding multigrid V-cycle, and V-cycle preconditioner time are shown in the table. Also shown are the algorithmic parallel efficiency $\eta_A$ based on the number of MINRES iterations ($\eta_A = 1.00$ implies number of iterations remain constant with increasing problem size), the implementation parallel efficiency $\eta_I$ of one MINRES iteration excluding the V-cycle ($\eta_I = 1.00$ means MINRES runtime is independent of problem size), the parallel efficiency of the V-cycle preconditioner $\eta_V$ (1.00 means V-cycle runtime is independent of problem size), and the overall parallel efficiency $\eta$ ($\eta_V = 1.00$ means the end-to-end execution time, including the setup phase, is independent of the problem size).

| # Cores | # Dofs | MINRES # iterations | AMG setup (s) | MINRES matvec (s) | AMG V-cycle (s) | $\eta_A$ | $\eta_I$ | $\eta_V$ | $\eta$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 403K | 63 | 8.2 | 174.8 | 49.9 | 1.00 | 1.00 | 1.00 | 1.00 |
| 8 | 3.3M | 66 | 14.8 | 215.2 | 78.1 | 0.95 | 0.85 | 0.67 | 0.76 |
| 64 | 26.8M | 75 | 20.6 | 240.2 | 143.9 | 0.84 | 0.87 | 0.41 | 0.58 |
| 512 | 216M | 90 | 28.4 | 295.4 | 222.2 | 0.70 | 0.85 | 0.32 | 0.43 |
| 4096 | 1.7B | 106 | 50.2 | 349.5 | 378.2 | 0.59 | 0.84 | 0.22 | 0.34 |

the AMG setup phase is carried out for three different scalar systems. Table 3 shows the parallel efficiencies for these different components as well as the overall parallel efficiency. The AMG setup and V-cycle preconditioner times grow faster than the times for the other parts of the solver, which scale almost optimally. This is due to the extensive communication needed in the setup phase and the coarse grid solve within the V-cycle preconditioner. For more discussion of these well-known bottlenecks of parallel AMG implementations, we refer to [16,13]. Ultimately, however, an overall parallel efficiency of 34% in scaling from 1 to 4096 cores should be regarded as excellent performance for implicit solution of a highly variable coefficient saddle point problem.

Finally, we study the dependence of the solver on the magnitude of viscosity variation. As before, we consider the Stokes solve on the final mesh, which has undergone three cycles of refinement. Changing the parameters $\alpha$ and $\beta$ in (14) leads to different contrasts in the viscosity. Table 4 shows the resulting minimum and maximum values of the viscosity throughout the mesh and the maximum viscosity gradient. The table reports the number of MINRES iterations needed for solution of a problem with 216 M degrees of freedom on 512 cores, the AMG setup time, and the average solver time per MINRES iteration. The number of MINRES iterations remains essentially constant, independent of the range of viscosity variation. Moreover, the AMG setup time, which takes into account viscosity when building the coarse grid hierarchy, takes approximately the same amount of time in all cases.

### 5.2. Example 2: benchmark for melt migration

The second example is a benchmark problem from magma dynamics. Magma dynamics can be modeled by a coupling of Darcy's law for porous flow of melt within a viscously deforming solid date represented by Stokes flow [21]. The pressure gradient from the Stokes equation affects the melt flow (see e.g. [24]), so for this problem it is critical to compute an accurate approximation of the pressure with the Stokes solver. We solve the benchmark Stokes flow problem proposed in [23], which models flow of the mantle driven by a mid-ocean ridge-transform-ridge spreading center. Fig. 4a illustrates the geometry of the driving plates, while the right image in Fig. 3 gives the velocity field. Large pressures are expected at the ridge, which will drive adaptivity. A semi-analytical spectral solution of the benchmark problem [29] is used as a reference solution to calculate the error in the finite element approximations.

**Table 4**
Performance of Stokes solver for varying viscosity given by (14) for $\alpha$ and $\beta$ as given in the table. As in Table 3 we use a mesh that has undergone three cycles of refinement, and examine only the final Stokes solve (which is initialized with a zero solution). The table reports the minimum and maximum viscosity values ($\mu_{\min}$ and $\mu_{\max}$), the maximum viscosity gradient norm $\|\nabla\mu\|_{\max}$, the number of MINRES iterations, the AMG setup time, and the average time per MINRES iteration. Each case has approximately 216M degrees of freedom and is solved on 512 cores.

| $\alpha$ | $\beta$ | $\mu_{\min}$ | $\mu_{\max}$ | $\|\nabla\mu\|_{\max}$ | # MINRES iterations | AMG setup time (s) | Solve time per iteration (s) |
|---|---|---|---|---|---|---|---|
| 0 | – | 1.00e–0 | 1.00 | 0.00e+0 | 86 | 25.29 | 5.82 |
| 3 | 200 | 4.98e–2 | 1.00 | 2.05e+1 | 80 | 28.02 | 5.80 |
| 7.5 | 20 | 5.53e–4 | 1.00 | 8.33e+0 | 75 | 25.26 | 5.62 |
| 7.5 | 200 | 5.53e–4 | 1.00 | 2.63e+1 | 90 | 28.44 | 5.75 |
| 7.5 | 2000 | 5.53e–4 | 1.00 | 8.28e+1 | 91 | 26.97 | 5.35 |
| 12 | 200 | 6.14e–6 | 1.00 | 2.89e+1 | 95 | 28.42 | 5.70 |
| 15 | 200 | 3.06e–7 | 1.00 | 3.14e+1 | 93 | 31.35 | 6.46 |

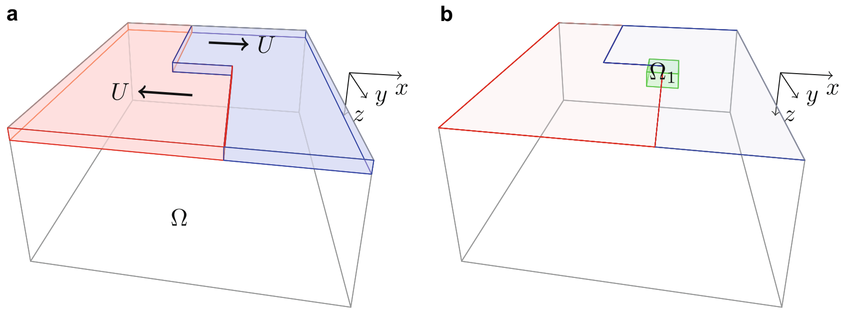


**Fig. 4.** (a) Geometry of plates driving the ridge-transform-ridge benchmark problem. (b) The region $\Omega_1$ used in the quantity of interest.

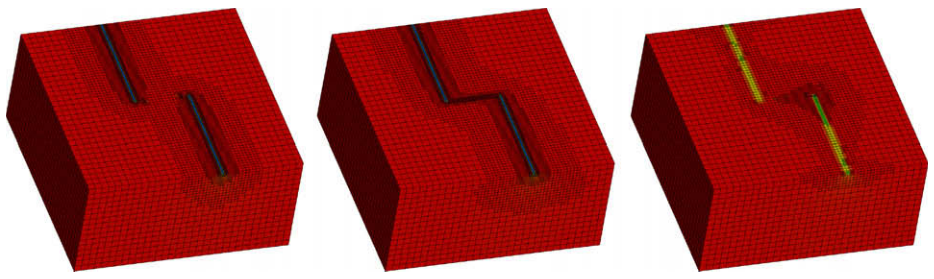


**Fig. 5.** Adaptively refined meshes for Example 2 using divergence error indicator (left), global error indicator (middle), and adjoint error indicator (right). The divergence indicator does not refine in the shearing zone, while the adjoint error indicator places elements mainly in or near the region of interest. The surface color indicates the pressure field.

For the results given in this section, we assume a constant viscosity $\mu = 1$ and use the non-dimensionalized domain $\Omega = [-6,6] \times [0,12] \times [0,6]$, where the $z$-axis is directed downward as seen in Fig. 4a. The boundary conditions on the top of the domain (i.e., where $z = 0$) are given by

$$\begin{aligned} u_1(x,y,0) &= \mathrm{erf}\left(\frac{x+1.5}{\lambda_0}\right) + \mathrm{erf}\left(\frac{x-1.5}{\lambda_0}\right) - \frac{1}{2}\mathrm{erf}\left(\frac{x+1.5}{\lambda_0}\right) \\ &\quad \times \mathrm{erf}\left(\frac{y-6}{\lambda_0}+1\right), \qquad (15) \\ u_2(x,y,0) &= 0, \\ u_3(x,y,0) &= 0, \end{aligned}$$

where $\lambda_0$ controls the smoothness of the velocity transformation at the ridge. A smaller $\lambda_0$ gives rise to a steeper pressure gradient near the ridge.

As quantity of interest we consider an integral of the pressure over the rectangular region $\Omega_1 = [0.75, 2.25] \times [5.25, 6.75] \times [0, 0.75]$, which is placed around one of the central singularities of the ridge; see Fig. 4b. We conduct a medium-scale and a large-scale test, in which we compare uniform refinement with several adaptive refinement strategies. In the adaptive cases, we mark those elements for refinement whose error indicator is larger than the mean error indicator plus 1/2 of its standard deviation.

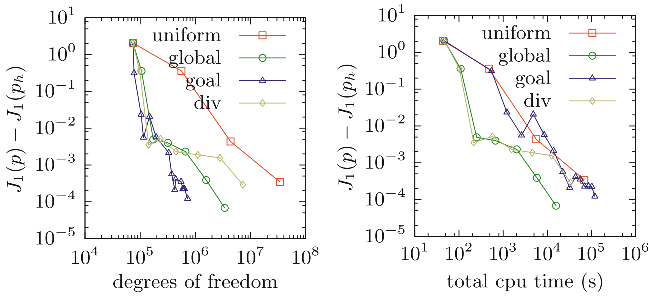


**Fig. 6.** Comparisons between refinement strategies for Example 2 (melt migration problem) with smoothing parameter $\lambda_0 = 0.04$ in (15), i.e. the medium-scale problem. The error in the quantity of interest is computed using the semi-analytical solution. A uniform mesh solution is compared with adaptive refinement strategies based on global, divergence, and goal-oriented *a posteriori* error indicators. The left image plots the error in the quantity of interest against the degrees of freedom. Note that since the quantity of interest $J_1$ is the mean pressure in $\Omega_1$ and not a global norm, one cannot expect monotonicity of the error. The right image plots the error versus the total run time. For the uniform mesh cases, the run time is based on solution of a single mesh problem, initializing the MINRES solver with the zero solution guess. For the runs on adaptive meshes the run time includes error estimation, mesh adaptation, and the solves on all coarser meshes. Moreover, for the goal-oriented error indicator, the run time also includes the solves of adjoint problems on (by a factor of 8) finer meshes. The problems are solved on different numbers of cores (8 for all adaptive runs and 1, 4, 32, and 256 cores for the uniform runs). To compensate for this difference, we report the total cpu time, i.e., the total wall clock time multiplied by the number of cores.

For the medium-scale problem, which uses the ridge smoothing parameter $\lambda_0 = 0.04$, we are able to compute the semi-analytical solution in reasonable time, and thus use velocity boundary conditions on all faces with values given by this solution. This solution is also used to compute the error in the quantity of interest. In Fig. 5, we show adapted meshes after three cycles of refinement, using different error indicators. The results are summarized in Fig. 6, where we plot not only the error versus the degrees of freedom, but also versus overall run time. The run time includes the solve time on all coarser grids as well as the mesh adaptation time. To account for the very different sizes of the problems, they are solved on different numbers of cores. This is why we report a "total cpu time" in the figure, which is the wall clock time multiplied by the number of cores.

The results for the large-scale problem are reported in Fig. 7. Here we choose $\lambda_0 = 10^{-5}$ in (15), which is why this problem requires much higher resolution around the ridge. We can no longer compute a semi-analytical solution in reasonable time, and therefore we use (15) as a boundary condition only on the top surface, while employing zero traction conditions on all other boundaries. Moreover, the exact value of the quantity of interest is estimated by extrapolating the results obtained on uniform meshes.

The results in Figs. 6 and 7 show that adaptive solutions require orders of magnitude fewer degrees of freedoms for the same accuracy than uniform mesh solutions. Even though the timings for the adaptive mesh cases take into account all overheads including the solves on all coarser meshes, we observe an improvement in the total cpu time. Note also that the adaptive cases require less memory, which makes it possible to run them on fewer cores. The goal oriented error indicator results in the fewest degrees of freedom. However, this indicator requires solution of an adjoint problem at each iteration, which adds to the overall run time, as can be seen in the right plots in Figs. 6 and 7. The adjoint solve could be accel-

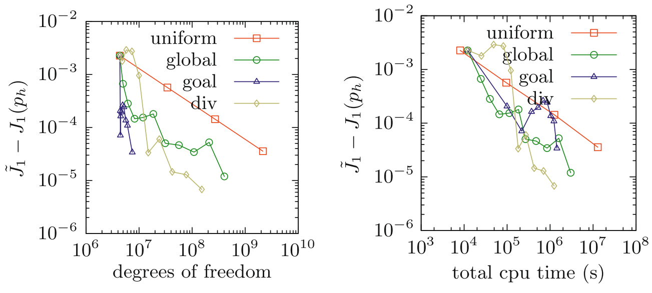


**Fig. 7.** Same as Fig. 6 but with $\lambda_0 = 10^{-5}$. Since the smoothing parameter $\lambda_0$ is too small to compute an accurate semi-analytical solution in reasonable time, the exact value for the quantity of interest is estimated by extrapolating the results of the uniform mesh solutions. Since the problems are solved on different numbers of cores (128 cores for all adaptive case, and 16, 128, 1024, and 8192 cores for the uniform cases), we again report the total cpu time.

erated significantly by not solving the adjoint problem on a finer mesh (requiring 8 times more elements), but on the same mesh as the primal Stokes problem and then using solution reconstruction techniques.

## Acknowledgements

This work was partially supported by NSF (Grants OCI-0749334, DMS-0724746, CNS-0619838, CCF-0427985), DOE SC's SciDAC program (Grant DE-FC02-06ER25782), DOE NNSA's PSAAP program (cooperative agreement DE-FC52-08NA28615), and AFOSR's Computational Math program (Grant FA9550-07-1-0480). We acknowledge many helpful discussions with the *hypre* developers and with George Biros and Serge Prudhomme. We thank TACC for their outstanding support, in particular Bill Barth, Karl Schulz, and Victor Eijkhout. We also thank Marc Spiegelman for referring us to the benchmark example used in Section 5.2. Finally, we dedicate this paper to Professor J. Tinsley Oden, who provided inspiration and encouragement of this work, on the occasion of his 70th birthday.

## References

[1] M. Ainsworth, J.T. Oden, A Posteriori Error Estimation in Finite Element Analysis, Pure and Applied Mathematics (New York), John Wiley & Sons, New York, 2000.
[2] S. Aluru, F.E. Sevilgen, Parallel domain decomposition and load balancing using space-filling curves, in: Proceedings of the Fourth IEEE Conference on High Performance Computing, 1997, pp. 230–235.
[3] I. Babuška, W. Rheinboldt, A posteriori error estimates for the finite element method, Int. J. Numer. Method Engrg. 12 (1978) 1597–1615.
[4] W. Bangerth, R. Hartmann, G. Kanschat, deal.II – a general-purpose object-oriented finite element library, ACM Trans. Math. Software 33 (2007).
[5] R. Becker, R. Rannacher, An optimal control approach to a posteriori error estimation in finite element methods, Acta Numer. 10 (2001) 1–102.
[6] P. Bochev, C. Dohrmann, M. Gunzburger, Stabilization of low-order mixed finite elements for the Stokes equations, SIAM J. Numer. Anal. 44 (2006) 82–101.
[7] W.L. Briggs, V.E. Henson, S. McCormick, A Multigrid Tutorial, second ed., SIAM, 2000.
[8] C. Burstedde, O. Ghattas, M. Gurnis, G. Stadler, E. Tan, T. Tu, L.C. Wilcox, S. Zhong, Scalable adaptive mantle convection simulation on petascale supercomputers, in: Proceedings of ACM/IEEE SC08, 2008.
[9] A. Caglar, M. Griebel, M.A. Schweitzer, G. Zumbusch, Dynamic load-balancing of hierarchical tree algorithms on a cluster of multiprocessor PCs and on the Cray T3E, in: H.W. Meuer (Ed.), Proceedings 14th Supercomputer Conference, Mannheim, Mateo, 1999.
[10] Center for Applied Scientific Computing, LLNL, Hypre. High performance preconditioners, User Manual, 2007. <https://computation.llnl.gov/casc/linear_solvers/>.
[11] L. Demkowicz, J. Kurtz, D. Pardo, M. Paszyński, W. Rachowicz, A. Zdunek, Computing with hp Finite Elements II. Frontiers: Three-Dimensional Elliptic and Maxwell Problems with Applications, CRC Press, Taylor and Francis, 2007.
[12] J.M. Dennis, Partitioning with space-filling curves on the cubed-sphere, in: IPDPS'03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing, Washington, DC, USA, 2003, IEEE Computer Society, p. 269.1.
[13] H. De Sterck, U.M. Yang, J.J. Heys, Reducing complexity in parallel algebraic multigrid preconditioners, SIAM J. Matrix Anal. Appl. 27 (2006) 1019–1039.
[14] C. Dohrmann, P. Bochev, A stabilized finite element method for the Stokes problem based on polynomial pressure projections, Int. J. Numer. Methods Fluids 46 (2004) 183–201.
[15] H.C. Elman, D.J. Silvester, A.J. Wathen, Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics, Oxford University Press, Oxford, 2005.
[16] R.D. Falgout, An introduction to algebraic multigrid, Comput. Sci. Engrg. 8 (2006) 24–33.
[17] T.J.R. Hughes, The Finite Element Method, Dover, New York, 2000.
[18] K. Hutter, Theoretical Glaciology, Mathematical Approaches to Geophysics, D. Reidel Publishing Company, Dordrecht, Holland, 1983.
[19] B. Kirk, J.W. Peterson, R.H. Stogner, G.F. Carey, libMesh: A C++ library for parallel adaptive mesh refinement/coarsening simulations, Engrg. Comput. 22 (2006) 237–254.
[20] A. Laszloffy, J. Long, A.K. Patra, Simple data management, scheduling and solution strategies for managing the irregularities in parallel adaptive hp finite element simulations, Parallel Comput. 26 (2000) 1765–1788.
[21] D. McKenzie, The generation and compaction of partially molten rock, J. Petrol. 25 (1984) 713–765.
[22] L.N. Moresi, S. Zhong, M. Gurnis, The accuracy of finite element solutions of Stokes' flow with strongly varying viscosity, Phys. Earth Planet. Interiors 97 (1996) 83–94.
[23] J.P. Morgan, D.W. Forsyth, Three-dimensional flow and temperature perturbations due to a transform offset: effects on oceanic crustal and upper mantle structure, J. Geophys. Res. 93 (1988) 2955–2966.
[24] J.P. Morgan, Melt migration beneath mid-ocean spreading centers, Geophys. Res. Lett. 14 (1987) 1238–1241.
[25] J.T. Oden, S. Prudhomme, Goal-oriented error estimation and adaptively for the finite element method, Comput. Method Appl. Mech. Engrg. 41 (2001) 735–756.
[26] M.A. Olshanskii, J. Peters, A. Reusken, Uniform preconditioners for a parameter dependent saddle point problem with application to generalized stokes interface equations, Numer. Math. 105 (2006) 159–191.
[27] M.A. Olshanskii, A. Reusken, Analysis of a stokes interface problem, Numer. Math. 103 (2006) 129–149.
[28] C.C. Paige, M.A. Saunders, Solution of sparse indefinite systems of linear equations, SIAM J. Numer. Anal. 12 (1975) 617–629.
[29] M. Spiegelman, SpecRidge: a spectral ridge benchmark for pressure–velocity Stokes solvers, 2007. <http://geodynamics.org/hg/magma/3D/SpecRidge>.
[30] H. Sundar, R.S. Sampath, G. Biros, Bottom-up construction and 2:1 balance refinement of linear octrees in parallel, SIAM J. Sci. Comput. 30 (2008) 2675–2708.
[31] T. Tu, D.R. O'Hallaron, O. Ghattas, Scalable parallel octree meshing for terascale applications, in: Proceedings of ACM/IEEE SC05, 2005.
[32] S. Zhong, D.A. Yuen, L.N. Moresi, Numerical Methods in Mantle Convection, Treatise on Geophysics, Elsevier, 2007. pp. 227–252, (Chapter 7).