A volume integral equation Stokes solver for problems with variable coefficients

Dhairya Malhotra
The University of Texas at Austin,
Austin, TX 78712
dhairya.malhotra@gmail.com

Amir Gholami
The University of Texas at Austin,
Austin, TX 78712
i.amirgh@gmail.com

George Biros
The University of Texas at Austin,
Austin, TX 78712
gbiros@acm.org

ABSTRACT

We present a novel numerical scheme for solving the Stokes equation with variable coefficients in the unit box. Our scheme is based on a volume integral equation formulation. Compared to finite element methods, our formulation decouples the velocity and pressure, generates velocity fields that are by construction divergence free to high accuracy and its performance does not depend on the order of the basis used for discretization. In addition, we employ a novel adaptive fast multipole method for volume integrals to obtain a scheme that is algorithmically optimal. Our scheme supports non-uniform discretizations and is spectrally accurate. To increase per node performance, we have integrated our code with both NVIDIA and Intel accelerators. In our largest scalability test, we solved a problem with 20 billion unknowns, using a 14-order approximation for the velocity, on 2048 nodes of the Stampede system at the Texas Advanced Computing Center. We achieved 0.656 petaFLOPS for the overall code (23% efficiency) and one petaFLOPS for the volume integrals (33% efficiency). As an application example, we simulate Stokes flow in a porous medium with highly complex pore structure using a penalty formulation to enforce the no slip condition.

1. INTRODUCTION

We propose an algorithm for the Stokes equations in the unit cube with variable coefficients, which can be stated as

$$\rho u - \operatorname{div}\left(\mu(\nabla u + \nabla u^T)\right) + \nabla p = f, \quad \operatorname{div} u = 0. \qquad (1)$$

Here, $u = u(x)$ is the (vector-valued) velocity, $p = p(x)$ is the (scalar-valued) pressure, $f = f(x)$ is a (vector-valued) momentum source, and $x \in [0,1]^3$. The first equation is the conservation of momentum and the second the conservation of mass—also known as the incompressibility condition. Periodic, free-space, Dirichlet, Neumann, or mixed boundary conditions on the faces of the unit cube can be applied. The **variable coefficients** $\rho = \rho(x)$ and $\mu = \mu(x)$ are related to the fluid density and viscosity respectively.

SC14 November 16–21, 2014, New Orleans

Equation (1) models nearly incompressible flows (steady or unsteady upon temporal discretization) in which the convective inertia is neglected. Such flows are found in microfluidics, biofuels, emulsions, polymers, porous and fractured media flows, and geophysical flows. Also, (1) can be used to model incompressible elastic materials with a homogeneous matrix phase and a random distribution of micron-sized particles in the matrix like reinforced elastomers, microgel suspensions, and biological tissues. Finally, solvers for equation (1) can be used as part of implicit-explicit time-stepping schemes for Navier-Stokes problems, in which the nonlinear convection is treated explicitly.

Designing numerical methods for (1) is challenging. The main difficulties are summarized below.

- It requires four unknowns per spatial grid point in three dimensions (three velocities and one pressure).
- Satisfying the incompressibility condition accurately is hard but crucial for obtaining the correct results.
- We cannot use arbitrary discretization spaces for the velocity and pressure because the inf sup condition [17] must be satisfied.
- Discretizations of (1) result in ill-conditioned systems. The need for different discretization spaces for velocity and pressure necessitates block preconditioners.
- Equation (1) is an elliptic but indefinite problem, which further complicates the construction of fast linear algebraic solvers and preconditioners, especially for problems with highly variable coefficients or high-order discretizations [6].

Due to the importance of Stokes solvers, sophisticated techniques have emerged that can tackle the challenges described above. Discretizing and solving (1) is typically done using finite element methods (FEM) and, to a lesser extent, using finite-difference or finite volume methods [17]. Many theoretically-optimal technologies have been developed for constant and variable coefficients. In practice, however, most existing codes that have been scaled to large core counts have demonstrated scalability only for low-order implementations, typically first- or second-order accurate [7, 8].

1.1 Contributions

We propose a scheme that circumvents most of the challenges associated with stencil-based discretizations. Writing $\rho(x)$ and $\mu(x)$ as perturbations around constant values ρ_0 and μ_0, that is $\rho(x) = \rho_0 + \tilde{\rho}(x)$ and $\mu(x) = \mu_0 + \tilde{\mu}(x)$, it is possible to transform (1) to a second-kind volume integral equation for the velocity u *only*:

$$u + \mathcal{G}[\tilde{\rho} u] + \mathcal{D}[\tilde{\mu}(\nabla u + \nabla u^T)] = \mathcal{G}[f], \qquad (2)$$

where $\mathcal{G}$ is a convolution operator with a boundary condition-dependent Green's function for the Stokes problem and $\mathcal{D}$

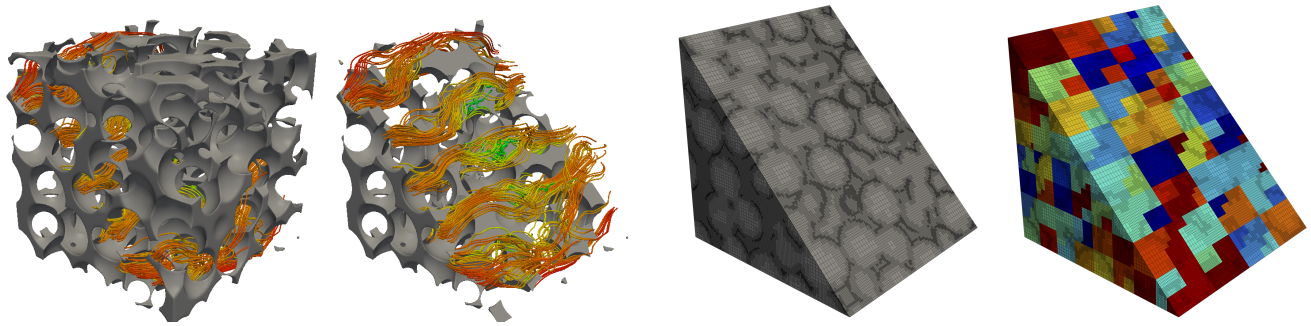


Figure 1 *Here we illustrate the capabilities of our solver. We simulate Stokes flow through a porous medium. From left: in the **first** figure, the grey color indicates the solid phase geometry and the space in between is the pore space. We also visualize the velocity field using streamlines. In the **second** figure we show the same geometry with clipping to better visualize the streamlines. The **third** figure shows the leaves of the octree and the fourth one the spatial partitioning across different nodes. Weak scalability results for this problem are reported below.*

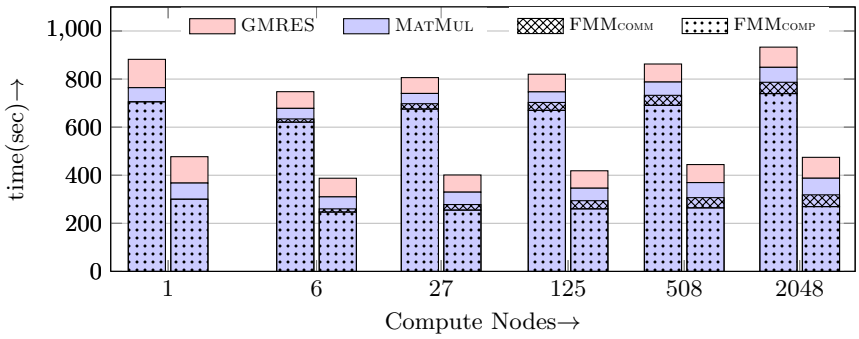


Figure 2 *Breakdown of time in seconds for different stages of the solver for CPU and CPU+Phi runs, respectively. We use the PETSc library for the GMRES solver. The remaining implementation is done in our group. The 'MatMult' phase is the application of the integral equation matrix operator to a vector. We break it down to computation and communication. We observe excellent scalability from one to 2048 nodes. Each node has 16 x86 cores, and a Phi co-processor. The clock per core is 2.7GHz. We report more information about this run in Table 1.*

p	N_{dof}/p	N_{iter}	T_{solve}	TFLOPS	η
1	8.0E+6	155	477	0.36	1.00
6	7.8E+6	115	388	2.27	1.04
27	8.6E+6	101	401	10.3	1.05
125	8.5E+6	98	419	45.3	0.99
508	8.9E+6	92	444	173	0.94
2048	9.1E+6	90	474	656	0.88

Table 1 *Solve time, total* FLOP *rate and parallel efficiency of the solver for porous media flow, for weak scaling up to 2K compute nodes on Stampede using CPUs and the Intel Phi accelerator. The CPU code and Phi portions of the code are multithreaded and vectorized. For the GMRES solve we use the PETSc library; the GMRES calculation is not accelerated with Phi. The GMRES iterations do not depend on the mesh size (in fact they are reducing with increasing mesh size). Here $\rho = 0$ in the pore phase and $\rho = 1E + 9$ in the solid phase.*

is the symmetric part of its gradient. The Green's functions correspond to any constant-coefficient values ρ_0 and μ_0, e.g., the average values of μ and ρ. The derivation and precise expressions are stated in §2. In other words, to solve (1) we solve (2) for the velocity u. Once the velocity has been computed, the pressure and overall stress can also be computed by evaluating appropriate convolution integrals.

Our formulation has several nice characteristics. It results in a *second-kind Fredholm equation* with a condition number that is independent of the mesh size; it depends only on the magnitude of $\tilde{\rho}$ and $\tilde{\mu}$. The *incompressibility condition* is satisfied pointwise due to the Green's function formulation. The equations for velocity and pressure are *decoupled*. We can evaluate the pressure as a post-processing step. There is *no* inf sup-*type restriction* on approximating p.

Formulations like (2) for variable-coefficient boundary value problems are well known [13] but have not been used for Stokes problems. One possible reason is the lack of technologies to evaluate $\mathcal{G}$ accurately and efficiently. $\mathcal{G}$ and $\mathcal{D}$ are formally dense operators. They are convolutions, but for non-uniform discretizations, fast Fourier transforms can not be used. A second possible reason is that the singularities in $\mathcal{G}$ and $\mathcal{D}$ make their computation extremely expensive, resulting in huge constants in the complexity estimates. The third possible reason is that the method requires the knowledge of a Green's function that accounts for the boundary conditions. Thus, the method is restricted only to simple geometries. In this paper, we try to address these issues. We demonstrate that the quadratures and the convolution can be done accurately and efficiently. Also, we show how one can approximate complex geometries using penalization. In particular,

- We present a volume integral formulation of Stokes equation and demonstrate the feasibility of the approach (§2, §2.3).
- We conduct a performance study and report time-to-solution for various smooth and discontinuous problems (§3). In section §2.2, for the constant coefficient case, we compare our results with finite-element discretizations using `Deal.II` an award-winning state-of-the-art finite-element library [3].
- We demonstrate scalability on single core, GPUs, MIC, and MPI architectures (§3) and report, to our knowledge, one of the largest, high-order Stokes runs.
- We apply it to porous medium flows in complex geometries using a penalization approach. An example is shown in Figure 1 and scalability results in Figure 2 and Table 1.
- We make our code freely available[1].

Our method consists of a fast volume integral evaluation scheme that efficiently computes the convolution operators $\mathcal{G}$ and $\mathcal{D}$ using a volume fast multipole method. We use 2:1 balance restriction of the tree to enable precomputation of all singular or near-singular integrals offline. Also, we use

[1] `http://padas.ices.utexas.edu/sc14stokes.tgz`

highly optimized kernels for x86, Intel Phi, and NVIDIA Kepler architectures. We use PETSc's [2] Krylov iterative solvers for the solution of the integral equation.

Since our solver supports variable coefficients, the case of complex geometries can be treated with several methods. For low-order approximations a penalty formulation can be used. We present an example in this paper. For high-order accuracy ideas similar to overset grids [12] and domain decomposition methods [10] can be used. The geometry is first decomposed into subdomains that are diffeomorphic to the unit cube and matched with appropriate boundary conditions. Each subdomain can be solved with our scheme. Algorithms for high-order accuracy and complex geometries are ongoing work in our group.

1.2 Related work

An integral equation for Navier-Stokes in two dimensions was reported in [16], but the formulation and algorithms were specific to a disk geometry and do not generalize to arbitrary geometries in 2D or to 3D. We are not aware of any other work on volume integral equation formulations for Stokes with variable coefficients. Of course, there is a lot of work on solvers for Stokes *boundary integral* equations [23, 24] but they are not applicable for discretizing (1) for arbitrary ρ and μ. A key component of our solver is the efficient evaluation of the convolution operators $\mathcal{G}$ and $\mathcal{D}$ in (2). In 2D, a fast scheme was proposed in [14] and a 3D extension is discussed in [20]. We use a scheme that uses a novel algorithm for traversing near and far-field integrals. Our volume fast multipole scheme along with the new traversals for the far and near interactions are described in [22]. We are not aware of any other work on scalable algorithms for evaluating $\mathcal{G}$ or for solving volume integral equations.

For finite element methods for steady Stokes problems we refer the reader to [5] and [17]. For unsteady Navier-Stokes, most solvers are based on pressure-projection schemes, which do not work well for stationary problems. For Stokes solvers, state-of-the art implementations include [7,8], and [19]. The latter is done with `Deal.II`, an open source package with which we compare our code for the constant coefficient case on a single core. One of the most scalable Stokes runs is reported in [9] in which the authors solve problems with up to two billion unknowns on 120K cores using linear elements.

The majority of scalable finite element codes for the Stokes equation use low-order discretizations. An exception is the work in [6] in which the author studies the convergence rates of different high-order discretizations along with the costs of solving the related algebraic system of equations. Multigrid-accelerated block preconditioners result in mesh-independent behavior for all orders. However, the constants deteriorate with increasing approximation order. A cubic velocity-linear pressure ($Q3-Q1$) discretization required 41 iterations for six orders of magnitude reduction in the Krylov residual, whereas a seventh-order velocity-fifth order pressure ($Q7-Q5$) discretization required 95 iterations.

For porous media flow there is a lot of work for Darcy models, but less work on scalable algorithms for Stokes flow. Typically meshing is prohibitively expensive or not scalable and a penalty formulation similar to ours (described in §2.1.1) is used. A discussion on the need and importance of solvers for porous media flows can be found in [25] which also uses a penalty formulation (with a finite volume scheme).

1.3 Limitations

Our work is the first study of its kind and as such is not comprehensive. We consider the formulation, analysis, convergence tests and scalability studies only for the case of variable density in free space. Here we are not considering high-order accurate scheme for complex geometries, generic boundary conditions, or variable viscosity problems. Also, we are not considering the design of preconditioners for (2). As we can see in Table (1), although the condition number of (2) is mesh-independent, it does depend on the magnitude of the variable coefficients. We briefly comment on preconditioning in §4.

2. METHODOLOGY AND ALGORITHMS

G	kernel (Green's) function
ω	computational domain: $[0,1]^3$
N_{oct}	number of leaf octants
L	maximum tree depth
m	multipole order
q	Chebyshev polynomial degree
ϵ_{tree}	tolerance for adaptive refinement
$N_{\text{cell}} = (q+1)(q+2)(q+3)/2$	degrees of freedom per leaf
$N_{\text{dof}} = N_{\text{oct}} N_{\text{cell}}$	number of unknowns
p	number of processes
T	total solve time
ϵ_{gmres}	GMRES residual
N_{iter}	GMRES iterations

Table 2 *Basic notation used in the paper.*

To explain our scheme we will consider the case with free-space boundary conditions. Other boundary conditions, for example Dirichlet, Neumann, or periodic can be implemented using either the same free-space Green's function by tiling $\mathbb{R}^3$ or by using a problem-specific Green's function [14]. We assume that $\tilde{\rho}$, $\tilde{\mu}$, and f are compactly supported in $\omega = [0,1]^3$. We only consider the variable density case and set $\mu = 1$. Under these simplifications, (1) becomes

$$\rho_0 u + \tilde{\rho} u - \Delta u + \nabla p = f, \quad \operatorname{div} u = 0.$$

To derive the integral equation we need to introduce the Green's function $G(x,y)$. By construction, $G(x,y)$ satisfies

$$\begin{aligned} \rho_0 G(x,y) - \Delta_x G(x,y) + \nabla_x G_p(x,y) &= \delta(x-y), \\ \operatorname{div}_x G(x,y) &= 0, \end{aligned} \tag{3}$$

where $G_p(x,y)$ is the corresponding pressure and δ is the Dirac delta function. For a function f, the convolution of G with f is denoted by $\mathcal{G}[f]$. For the free-space case and with $\rho_0 = 0$, $\mathcal{G}$ is given by [23]

$$\mathcal{G}[f](x) := \int_{\omega(y)} G(x-y) f(y) = \int_{\omega(y)} \frac{1}{8\pi} \left(\frac{1}{\|r\|_2} + \frac{r r^T}{\|r\|_2^3} \right) f(y),$$

where $r = x - y$. We also set $\rho_0 = 0$ so that $\rho = \tilde{\rho}$. Then, by taking the convolution of the momentum equation in (1) with G, integrating by parts the $G\Delta u$ and $G\nabla p$ terms, and using (3), we obtain

$$u(x) + \mathcal{G}[\rho u](x) = \mathcal{G}[f](x), \ \forall x \in \omega. \tag{4}$$

Equation (4) is the main equation we will be considering here. The expressions for G and its gradient for the case with variable viscosity or for the case with non-zero ρ_0 are quite cumbersome and beyond the scope of this paper. Nothing changes in the formulation for those cases, but the case of $\rho_0 \neq 0$ is computationally more expensive because G is not scale invariant anymore.

As mentioned in the introduction, equation (4) is a second-kind Fredholm integral equation. Thus, it has a bounded condition number [18]. The size of the condition number increases with increasing the norm of ρ.

2.1 Discretization

The numerical problem can be stated as follows: given evaluator functions for ρ, f and a target accuracy, compute an evaluator function for u.

To do this, we use a Galerkin scheme in which we represent ρ, f and u in the same basis. This basis is constructed by decomposing ω into a set of disjoint cells ω_i (in our case the leaf-octants of an octree partition ω). In each ω_i, we approximate $\rho(x)$, $u(x)$, and $f(x)$ using Chebyshev polynomials of degree q. We call this representation the **Chebyshev tree** of a function. Using a Galerkin projection on (4), we obtain a finite-dimensional algebraic system for the coefficients of u at each ω_i. This system is non-symmetric. We solve it with unrestarted, unpreconditioned, Generalized Minimum Residual method (GMRES). The evaluator function for u will use the Chebyshev coefficients of u at each ω_i. We review these steps in detail below.

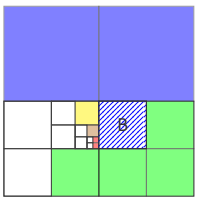
Building the Chebyshev tree. We partition ω to ω_i so that ρ and f are both resolved to a prespecified accuracy in the Chebyshev space. Let's consider the case for f. A Chebyshev octree representation of f is a tree in which at every leaf ω_i, we represent f by its Chebyshev coefficients $\hat{f}_{nm\ell}$ so that

$$f(x) = \sum_{n=0,m=0,\ell=0}^{n+m+\ell \le q} \hat{f}_{nm\ell} T_{nm\ell}(x), \quad x \in \omega_i,$$

where $T_{nm\ell}(x) = T_n(x_1)T_m(x_2)T_\ell(x_3)$ and T_n is the n-th degree Chebyshev polynomial. We proceed in a top-down fashion. Recall that we are given an evaluator, which for any x returns $f(x)$. To construct the tree, we start at the root level and we evaluate $\hat{f}$ by sampling f at Chebyshev points and then taking a discrete Chebyshev transform [26]. That is, we use Chebyshev quadrature to evaluate $\hat{f}_{nm\ell} = \int_{\omega_i} T_{nm\ell} f(x) w(x)$, where $w(x) = (1-x^2)^{-1/2}$ is required for orthogonality. Then, we check the size of the tails: if $\sum_{n+m+\ell=q} |\hat{f}_{nm\ell}| \le \epsilon$ we terminate the recursion, otherwise we subdivide and continue recursively for each child octant of the root. Once the tree is constructed, we have a representation for f in terms of piecewise Chebyshev polynomials. If f is discontinuous we stop when we reach a prespecified maximum tree level. Notice that we are *not* using a tensor product basis. Instead we truncate so that the highest degree of the polynomial is q. This results in N_{cell} coefficients per leaf octant (roughly $(q+2)^3/2$ instead of $3(q+1)^3$).

Convolution with $\mathcal{G}$. Once we have built the Chebyshev trees for ρ and f, we merge them to a single tree that represents both functions accurately. Then, we must evaluate the action of $\mathcal{G}$ on Chebyshev tree functions. This is the most expensive step in our method. Let us describe the basic mathematical structure of computing $u = \mathcal{G}[f]$, for a given function f. This evaluation corresponds to solving a constant-coefficient Stokes problem with right-hand side equal to f. One key part in this convolution is the quadrature. Since we also want to represent $u(x)$ as a Chebyshev polynomial, we need to evaluate $u(x)$ at Chebyshev quadrature nodes at each leaf octant. (If we need $u(x)$ at arbitrary target locations, we can use Chebyshev interpolation as a post-processing step.) The number of evaluation points, for each source octant is equal to N_{dof} and thus it is exceptionally expensive. Instead, we compute near-field integrals exactly and approximate far field integrals. By near-field integrals we mean integrals in which the source and target octants share a face, edge, or vertex. All other integrals are marked as far-field.

If we were to use a uniform tree with tensor product polynomials for the near-field integrals, we would need to precompute $243(q+1)^6$ integrals *per level*. For a 15-order approximation, we would need roughly 20GB per level, which is not practical. What makes the calculation possible is the invariance properties of the kernel G. For most problems this kernel is (up to a scalar prefactor) scale and rotationally invariant. *By avoiding the tensor product polynomial basis and using the symmetry properties of the kernel*, we can reduce the storage to $4N_{\text{cell}}^2$ (roughly $(q+2)^6$, or 128MB for $q = 14$). If the Green's function is not scale invariant we need to precompute near field interactions *for every* level that has leaf octants.



If the tree is non-uniform, the situation is more complex because we need to precompute integrals for every level combination and directionality of source and target octant pairs. For example, see the inset figure in which B is the target box. Every color represents interactions at a different level. Unlike particle FMM where the matrix entries are simple kernel evaluations, *now the entries are singular or near-singular integrals which cannot be computed on the fly.* We need to restrict the possible combinations. For this reason we use 2:1 balancing in which we create additional octants to satisfy the following requirement for the tree: the level of two octants that share a vertex, edge, or face can differ by no more than one.

Volume FMM: The evaluation of $\mathcal{G}$ is done with a volume FMM. Similar to particle FMM codes [11], the basic idea behind volume FMM is to construct a hierarchical decomposition of the computational domain using an octree. Then, the solution at each point x can be evaluated by summing over contributions from all octants in the octree. This summation is split into near and far interactions:

$$u(x) = \sum_{\omega_i \in \text{NEAR}(x)} \int_{\omega_i} G(x-y) f(y) + \sum_{\omega_i \in \text{FAR}(x)} \int_{\omega_i} G(x-y) f(y)$$

As mentioned the near interactions (from $\omega_i \in \text{NEAR}(x)$) are computed through direct integration. The far interactions (from $B \in \text{FAR}(x)$) are low-rank and can be approximated either using analytic expansions or source equivalent representations. We use the kernel-independent scheme [27] to approximate the far interactions. The only difference from particle FMM is that the source to equivalent interactions are precomputed since f (the source) is sampled at Chebyshev grid which is the same for every octant (up to translations and scalings). Similar modifications are required for the X-list and W-list in FMM terminology. More details about integrating volume potentials with FMM can be found in [22] and details on performance optimizations in §2.3.

Evaluating $\mathcal{G}[\rho u]$. We assume that we have an evaluator for $\rho(x)$ and N_{cell} Chebyshev coefficients per octant for u. To evaluate the convolution of their product, we first evaluate u at the $(q+1)^3$ Chebyshev points at every octant,

we multiply the values pointwise to get $\rho(x_k)u(x_k)$ at the Chebyshev points, we take their Chebyshev transform to compute the Chebyshev coefficients of $\rho(x)u(x)$ and then, we apply $\mathcal{G}$ as described above. To compute ρu, we pad the reduced Chebyshev polynomial basis, and then we use tensor-product transformations with $\mathcal{O}(q^4)$ complexity. The savings are not significant, since the integrals already require $\mathcal{O}(q^6)$ calculations.

Overall scheme: The overall scheme can be summarized as follows

- Create a Chebyshev tree based on approximating f and ρ to a desired accuracy. The resulting tree has N_{oct} octants.
- Evaluate $\mathcal{G}[f]$.
- Solve (4) using GMRES, in which for each mat-vec we use the quadrature rules and FMM acceleration described above.

If ρ or f are discontinuous, the tails do not decay. For this reason we always specify a maximum depth that corresponds to the minimum octant size allowed in our approximation.

To summarize, let P_N be the projection operator that restricts f, u and ρ, to the space spanned by N_{dof} polynomials. Then the discretized system can be written as

$$u_N + \mathcal{G}_N[\rho u_N] = \mathcal{G}_N[f], \tag{5}$$

where $\mathcal{G}_N[\cdot] = P_N\mathcal{G}[P_N\cdot]$ denotes the discretization of $\mathcal{G}$ using the quadrature scheme we described. Notice the degrees of freedom depend on q but for notational simplicity we suppress this dependence. We remark again that this linear system is not symmetric, unlike the original Stokes problem. (Many FEM-based solvers also use GMRES to allow for optimal preconditioning [6].)

Error analysis: We follow standard error analysis for projection methods for second kind operators [18]. Let u_N to be the solution of (4). Then the overall error can be estimated by

$$\|u_N - u\| = \mathcal{O}(\|P_N u - u\| + \|\mathcal{G}_N - P_N\mathcal{G}\| + \|P_N f - f\|),$$

in the L^2 norm. The constant in the estimate is proportional to the norm of $\mathcal{G}$, which in turn depends on $\|\rho\|$. The first term is the approximation error due to the projection, the second term is error due to quadratures (related to the smoothness of ρ) and FMM, and the last term is the approximation error of f.

Assuming standard regularity, i.e., $\rho(x) \in L^\infty$ and $f \in L^2$, then $u \in H^2$. If ρ and f are in C^∞ the convergence is of order q. Otherwise the convergence depends on the regularity of u (in our assumptions at least C^1) and ρ (due to the quadrature error). For constant coefficients $\rho = 0$ and the middle term drops. Then the error in u becomes directly proportional to the error in f.

2.1.1 *Formulation for the porous medium flow*

Let $\xi(x)$ be the characteristic function of the fluid phase and $1 - \xi(x)$ the characteristic function of the solid phase. Then (4) is satisfied in the fluid phase and $u = 0$ in the solid phase. For regular pore geometries the right approach is to use a double-layer potential formulation using boundary integral equations. But for complex geometries like the one in Figure 1, constructing surface meshes can be complicated and expensive due to the scalability of meshing and constructing multilevel preconditioners. When engineering accuracy are acceptable (say 1% error), one can approximate the solution using a penalty formulation. This is a classical approach similar to fictitious domain methods, immersed boundary methods, embedded methods and others. To force the fluid to have zero velocity we use a volume penalty method in which $\rho(x) = \eta(1 - \xi(x))$, where η is the penalty parameter. This can easily be derived by a constrained variational formulation of the Stokes equations in which $u(x) = 0$ for $\xi(x) = 0$. With this approximation, the formulation becomes

$$\eta(1 - \xi(x))u(x) - \Delta u + \nabla p = f, \quad \operatorname{div} u = 0,$$

and its volume integral formulation becomes

$$u(x) + \eta\mathcal{G}[(1 - \xi)u] = \mathcal{G}[f]. \tag{6}$$

The theoretical analysis of the scheme for Stokes equations can be found in [1]. Let us denote the solution of (6) as u_η and u_* be the exact solution. It can be shown that u_η converges to u_* as η goes to infinity, in the L^2 norm. If the solution u is regular (in the true domain), the convergence rate is expected to be $\mathcal{O}(1/\eta)$. If the solution is not regular the convergence rate can deteriorate to $1/\eta^{1/a}$, with $a = 2$ or even $a = 4$. Let us remark that the use of penalty formulations for porous media is not new, for example it has been used in [25].

2.2 Convergence

To demonstrate the convergence of the scheme, we consider three problems with analytic solutions. The first two test cases correspond to C^∞ velocity fields. For the third test the solution is in H^2. We consider convergence as a function of the polynomial degree q, the GMRES tolerance, the discretization error $\epsilon_{\text{tree}} = \|P_N f - f\|$ and the order of multipole expansions m. All the reported times here are for single-core of x86 only code compiled with the same compiler (Intel) with O2 flag.

In the **first test**, we consider a constant-coefficient case with $\rho = 0$ and study the convergence of the method as we decrease the tree-refinement tolerance ϵ_{tree} and correspondingly choose the best Chebyshev degree q and multipole order m for the fastest time to solution for a given accuracy. The velocity is given by $u(x) = \exp(-125|x|^2)(x_3e_2 - x_2e_3)$, where e_2 and e_3 are orthogonal unit vectors. We report the L^∞ and L^2 errors in the velocity field and the total time to solution in Table 3.

ϵ_{tree}	q	m	N_{oct}	N_{dof}	L^∞	L^2	T
1E-1	8	2	8	4.0E+3	1.3E-1	2.1E-1	0.004
1E-2	10	4	64	5.5E+4	6.1E-3	5.6E-3	0.102
1E-3	12	6	120	1.6E+5	3.0E-5	6.7E-5	0.656
1E-4	14	8	120	2.4E+5	8.2E-7	1.9E-6	1.54
1E-5	15	10	120	2.9E+5	2.3E-7	3.2E-7	2.53

Table 3 *Convergence for* ***constant-coefficient*** *Stokes flow with decreasing tree refinement tolerance ϵ_{tree} and increasing multipole order m; L^2 and L^∞ are the relative errors of the velocity;T is the solve time (single-core run).*

Notice that the number of octants does not change significantly but the error in the solution quickly reduces. We need just 120 octants with about 300K unknowns (three velocities per grid point) to reach single precision.

N_{oct}	N_{dof}	L^∞	L^2	T
1,408	43,508	1.9E-4	1.7E-4	0.48
4,544	135,360	2.7E-5	3.2E-5	2.47
10,368	298,972	1.4E-5	1.2E-5	6.08
20,224	579,152	3.6E-6	4.4E-6	14.9

Table 4 *Convergence for* ***constant-coefficient*** *Stokes flow using $Q2-Q1$ Taylor-Hood elements with* **Deal.II**. *We report L^2 and L^∞ relative errors of the velocity and the overall wall-clock time T (excluding setup) with increasing number of unknowns. We converged GMRES to* 1E-6 *relative residual reduction. The number of GMRES iterations is mesh independent (about 20).*

To put this calculation in perspective, we solved the same problem with `Deal.II` [3, 4] a state-of-the-art, finite element package that is part of the SPEC CPU 2006 benchmark. We used non-uniform discretization with hexahedral elements for both velocities and pressures (with $q = 2$ for velocities and $q = 1$ for pressure). We use a preconditioned GMRES solver. The preconditioner is block diagonal, also known as field-split preconditioning [19]. It uses one block for the viscosity based on algebraic multigrid library [15] with one V-cycle with one step of Chebyshev smoothing. It also uses a mass preconditioner for the pressure Schur complement. The convergence rates are reported in Table 4. These timings do not include setup times. For the integral equation formulation, the setup cost is a small percentage of the constant-coefficient solve (less than 10%) and negligible for variable coefficient problems. For the FEM code the setup cost (mostly building the multigrid operators) can be substantial.

Comparing the pointwise L^∞ errors, we observe that for five digits of relative accuracy, the finite element scheme requires 135K unknowns (second row) and the integral equation requires 160K unknowns (third row). Both schemes are quite fast, with our formulation being somewhat faster since it requires just a single evaluation of $\mathcal{G}$ with FMM. For six digits of accuracy and higher, the integral equation requires less than 300K and less than three seconds. The FEM scheme requires 580K unknowns and 14.9 seconds becoming 4-5× slower than the integral equation solver. Of course, an FEM-based Stokes solver—and `Deal.II` in particular, are much more general in terms of arbitrary geometries, boundary conditions and multiphysics couplings.

The second test case is the exact smooth solution for the case in which $\rho(x)$ is a scaled Gaussian. We report the number of iterations and time to solution. For fixed ρ, the number of iterations increases with increasing mesh size because we tighten the GMRES tolerance as we refine. The dependence of the GMRES iterations on large variations of ρ is mild up to fiver orders of magnitude variations. We observe fast convergence but the conditioning deteriorates with increasing $\|\rho\|_\infty$. For four digits of relative accuracy pointwise, the number of GMRES iterations is six for $\|\rho\|_\infty = 1E5$. For higher $\|\rho\|_\infty$, we need tighter GMRES residuals due to the deterioration of conditioning and the number of iterations jumps to 100s of iterations. Although the timings are not bad (just 13 minutes on a single core for the most expensive run), for large variations of ρ preconditioning should be used.

ϵ_{tree}	$\|\rho\|_\infty$	N_{oct}	ϵ_{gmres}	N_{iter}	L^∞	L^2	T
1E-1	1E+5	64	1E-3	5	8.3E-3	1.5E-2	3.91
1E-2	1E+5	120	1E-4	6	5.7E-4	7.0E-4	12.4
1E-3	1E+5	176	1E-7	14	4.6E-7	8.4E-7	45.4
1E-6	1E+5	736	1E-8	17	6.8E-8	1.9E-7	229
1E-1	1E+7	64	1E-4	12	2.2E-2	5.0E-2	9.45
1E-2	1E+7	120	1E-7	61	2.3E-4	3.4E-4	127
1E-3	1E+7	176	1E-9	103	7.6E-7	1.9E-6	337
1E-1	1E+9	64	1E-4	12	2.3E-2	5.5E-2	9.44
1E-2	1E+9	120	1E-7	82	6.2E-4	2.4E-3	171
1E-3	1E+9	176	1E-9	246	1.9E-5	1.6E-4	818

Table 5 *Convergence for a* ***variable-coefficient*** *Stokes flow, with reducing tree refinement tolerance ϵ_{tree} and GMRES tolerance ϵ_{gmres} for different values of ρ. Chebyshev degree $q = 14$, multipole order $m = 10$; T is the overall time in seconds on a single x86 core.*

In the **third test case**, we consider a problem with discontinuous ρ. We consider Stokes flow around a sphere for which we have an analytic solution. The solution in the exterior of the sphere is smooth but if we extend the velocity by zero inside the sphere, its derivatives are discontinuous at the boundary of the sphere. We approximate the solution of this problem using the penalty formulation and solving in the whole domain. For convergence, we must increase η as we refine and the problem becomes increasingly ill conditioned. Notice that we do *not* advocate using our method for this particular problem. Boundary integrals should be used instead. We just use it to illustrate the convergence rate of our scheme in this setting. The L^2 and L^∞ errors are measured on the exterior of the sphere only.

$\|\rho\|_\infty$	L	N_{oct}	ϵ_{gmres}	N_{iter}	L^∞	L^2	T
2.5E+5	1	1	4.0E-6	4	2.6E-1	2.4E-1	0.1
5.0E+5	2	8	2.0E-6	29	1.7E-1	8.9E-2	1.2
1.0E+6	3	64	1.0E-6	36	1.2E-1	4.8E-2	4.9
2.0E+6	4	120	5.0E-7	43	4.6E-2	6.8E-3	17
4.0E+6	5	512	2.5E-7	44	2.0E-2	8.8E-4	43

Table 6 *Convergence for Stokes flow around a sphere of radius=0.15 (****variable and discontinuous coefficients****) in $\Omega = [0,1]^3$; the Chebyshev degree $q = 14$, multipole order $m = 10$ with decreasing GMRES residual ϵ_{gmres} and increasing penalty $\|\rho\|_\infty$ and tree refinement level L. Again T is the total time in seconds on 16 x86 cores.*

q	m	N_{oct}	N_{iter}	L^∞	T
14	10	1,856	45	1.4E-2	140
14	6	1,856	45	1.4E-2	89
6	6	5,328	45	1.0E-2	38
4	6	5,328	53	1.4E-2	35
2	6	20,952	46	1.2E-2	103

Table 7 *Single node performance results for flow around a sphere (test case three, non-smooth solution) with fixed accuracy and different Chebyshev degrees q, multipole orders m. Here we examine the effect of using a high order discretization for a non-smooth problem. The timings are on a single node using 16 threads.*

Also in Table 7, we examine the cost for different element orders. We keep the target accuracy tolerance fixed (1% error) and we vary the polynomial order q and the FMM far-field accuracy m. In all of these examples, we set the penalty parameter to $\eta = 1E{+}7$. As we expected, higher-order elements will not help with the convergence rate but they can help with the constants by allowing faster convergence away from the discontinuity. From this table we observe that using high order approximation (high q and high m) increases the cost significantly and should not be used. The cases of $q = 6$ and $q = 4$ give similar timings. Next we give more details on performance optimizations and on our strategy of integrating our scheme with co-processor acceleration.

2.3 Performance optimizations

Accelerations on the FMM: In the following section we overview optimizations that we have made in our algorithm. These result in significantly improved performance and scalability.

2.3.1 Interactions and Octree Traversal

We define a source and a target octant to be well separated (or far) if they are at the same depth in the octree and are not adjacent. To compute far interactions we use two building blocks: multipole expansions and local expansions. The multipole expansion approximates the potential of an octant far away from it. The local expansion approximates the potential within an octant due to sources far away from it. The interactions are then approximated by computing a multipole expansion (source-to-multipole, multipole-to-multipole) for the source octant, multipole-to-local (V-list) translation and then evaluating the local expansion (local-to-local, local-to-target) at the target octant. We use the kernel-independent variant of FMM in our implementation. The form of the multipole and local expansions and the V-list translation operator for this variant are discussed in detail in [27].

Optimizations for near interactions: For particle FMM codes, it is typical to compute the near interactions by looping over every leaf octant, and then collecting its neighbors and computing the interactions. This calculation is compute bound. However, we found that this approach can be further improved by taking into account the structure of $\mathcal{G}$. Instead of looping over leaf octants, we compute all the near interactions as follows. We loop over interaction directions (north, south, west, east, and the diagonals for a total of 27) and for each interaction direction, we collect all source vectors for each source-target interaction pair in that direction. We then compute all interactions in that direction at once through a single DGEMM. For uniform trees the size of DGEMM is $(N_{\text{cell}} \times N_{\text{cell}}) \times (N_{\text{cell}} \times N_{\text{oct}})$, thus achieving excellent performance. Furthermore, several interactions are related through a spatial symmetry transform (rotation of coordinates) and this allows us to perform interactions along several directions at once. This is crucial for execution on accelerators, since they require the matrices to be large to achieve good performance. However, to do this we need to perform permutation operations on the source and target vectors; for this, we implemented hand vectorized and highly efficient kernels for CPU and co-processors (Phi, GPU).

Optimizations for far interactions: Similarly, as with the near interactions, we have re-organized the far interactions, also known as V-list. This is an expensive calculation, which, in its standard form is memory-bound. For every target octant (leaf or non-leaf), we have to visit 189 octants (its V-list) and perform a Hadamard vector product. We modify the algorithm and introduce blocking in which, instead of performing pairwise interactions, we perform interactions between blocks of eight octants at a time. This allows us to use vector-level parallelism by computing 8×8 matrix-vector products. We also block V-list interactions to use the spatial locality of V-list interactions i.e. the fact that neighboring octants share the same V-list octants. By doing so we make better utilization of the L1 and L2 cache. This blocking increases the performance by an order of magnitude. Details can be found in [22].

2.3.2 Parallel Fast Multipole Method

Here we summarize the important features of the intra-node parallelism (accelerators, multithreading and vectorization) and distributed memory parallelism. A detailed discussion of these optimizations is beyond the scope of this paper. We refer the interested readers to [22] for a detailed discussion of these concepts.

Asynchronous Execution on Co-processor: We compute U,W,X-list interactions on the co-processor and the remaining interactions (V-list, L2L and L2T) on the CPU. All computations and memory transfers between the co-processor and the host are asynchronous and overlapped with computation on CPU.

U,W,X-List Optimizations: We group similar interactions, those with the same interaction matrix or related by a spatial symmetry relation into a single matrix-matrix product, evaluated efficiently through DGEMM.

V-List Optimizations: The V-list interactions involve computation of the Hadamard products, which has low computational intensity and is therefore bandwidth bound. We rearrange data and use spatial locality of V-list interactions to optimize cache utilization. This along with use of AVX and SSE vector intrinsics and OpenMP allowed us to achieve over 50% of peak performance for this operation on the Intel Sandy Bridge architecture.

Distributed 2:1 Balance Refinement: We developed a new distributed memory algorithm for 2:1 balance refinement, which is more robust for highly non-uniform octrees than our earlier implementation.

Distributed Memory Parallelism: We use Morton ordering to partition octants across processors during the tree construction. In the FMM evaluation, after the upward pass, we need to construct the local essential tree through a reduce-broadcast communication operation. For this, we use the hypercube communication scheme of [21].

2.3.3 Complexity

The cost of FMM evaluation is given by the number of interactions between the octree nodes weighted by the cost of each interaction. The cost of each interaction depends on the multipole order m and the degree of Chebyshev polynomials q. Let N_{oct} be the local octree nodes, N_{leaf} the number of local leaf nodes. Also, let N_U, N_V, N_W, $N_X (= N_W)$ denote the number of interactions of each type U,V,W and X-list respectively. The overall cost is summarized in Table 8.

Interaction Type	Computational Cost
S2M, L2T	$\mathcal{O}(N_{leaf} \times q^3 \times m^2)$
M2M, L2L	$\mathcal{O}(N_{oct} \times m^2 \times m^2)$
W-list, X-list	$\mathcal{O}(N_W \times m^2 \times q^3)$
U-list	$\mathcal{O}(N_U \times q^3 \times q^3)$
V-list	$\mathcal{O}(N_V \times m^3 + N_{oct} \times m^3 \log m)$

Table 8 *Computational cost for each interaction type.*

The communication cost for the hypercube communication scheme is discussed in detail in [21]. For an uncongested network, that work provides a worst case complexity which scales as $\mathcal{O}(N_s(q^3 + m^2)\sqrt{p})$, where N_s is the maximum number of shared octants per processor. However, assuming that the messages are evenly distributed across processors in every stage of the hypercube communication process, we get a cost of $\mathcal{O}(N_s(q^3 + m^2) \log p)$. In our experiments with uniform octrees, the observed complexity agrees with this

estimate. Also, since shared octants are near the boundary of the processor domains, we have $N_s \sim (N_{oct}/p)^{2/3}$, where N_{oct} is total number of octants.

In the uniform octree case, there are $N_{oct} = N_{\text{dof}}/q^3$ total octants, $N_U = 27N_{oct}$ and $N_V = 189N_{oct}$. Due to the large constant factors, the cost of U-list and V-list interactions dominate over other interactions and the overall cost is:

$$T_{\text{FMM}} = \mathcal{O}\left(q^3 \frac{N_{\text{dof}}}{p}\right) + \mathcal{O}\left(\frac{m^3}{q^3}\frac{N_{\text{dof}}}{p}\right) + \mathcal{O}\left(\left(\frac{N_{\text{dof}}}{p}\right)^{2/3} q \log p\right).$$

For tree construction, is $\mathcal{O}(q^4)$ per octants and $\mathcal{O}(qN_{\text{dof}}/p)$ overall (excluding cost of redistributing octants which is data dependent). For the 2:1 balance refinement the cost is $\mathcal{O}\left(qN_{\text{dof}}/p + (N_{\text{dof}} \log N_{\text{dof}})/(pq^3)\right)$, assuming a hypercube interconnect. These are setup costs.

3. PERFORMANCE ANALYSIS

In this section, we analyze the performance of our implementation. Below, we briefly describe the experimental setup used in this work.

Hardware. All experiments were performed on the Stampede (TACC) supercomputer, a Linux cluster consisting of 6,400 compute nodes connected by 56GB/s FDR Mellanox InfiniBand network in a fat tree configuration. Each compute node has dual eight-core Intel Xeon E5-2680 CPUs running at 2.7 GHz and 32GB of memory. In addition, most nodes have an Intel Xeon Phi SE10P co-processor, while a few have an NVIDIA K20 GPU co-processor. In §3.1, we give a comparison of the single-node performance of our solver for the three configurations: CPU only, CPU+Phi and CPU+GPU. The system has a theoretical peak performance of 1.42TFLOPS per node (345.5GFLOPS for CPU and 1.07TFLOPS for Phi) and about 9PFLOPS for the entire system. Of this, Stampede achieved 5.2PFLOPS with 6,006 compute nodes on the Linpack benchmark.

Software. We use the Intel compiler version 13.1.0 along with Intel MPI Library 4.1 for Linux to compile our code. We use PETSc-3.4.3, FFTW3/3.3.2 and Intel MKL-11.0.1 libraries for the BLAS operations and NVIDIA CudaBLAS for the GPU version. Our code was compiled with -O2 optimization level.

3.1 Single Node Performance

In this section we present runtime and performance (in GFLOPS) on a single node of Stampede. We use the smooth variable coefficient test case with $\rho(x) = 10^6 \exp(-500x^2)$. In Table 9, we show results for both uniform and adaptive meshes. We vary the discretization order q for a fixed solution accuracy of about 1E-6. With higher order approximation, we require significantly fewer octree nodes and consequently solve the problem faster. We achieve a speedup of 2.7× and 3.1× on CPU for the uniform and adaptive cases respectively by increasing q from 8 to 16. Furthermore, using adaptive mesh requires an order of magnitude fewer unknowns and is about 10× faster for the same q. We can also use either an Intel Phi or NVIDIA GPU to accelerate the near interactions. For low-order discretization, since there is not enough work in near interactions, we do not see a significant advantage in using co-processors. However, for $q = 16$, we see 2.5× and 2.7 − 4.0× speedups for CPU+Phi and CPU+GPU cases respectively, with GPUs giving significantly better performance even for small problems with just 232 octants. Overall, we have over 100× speedup going from low-order CPU case to high-order, adaptive CPU+GPU case and achieve 579GFLOPS or about 40% of the theoretical peak performance of the compute node.

3.2 Weak Scalability

We now present some isogranular or weak scalability results on Stampede. In Figure 4 and Table 10, we present results for a low-order case with discretization order $q = 6$ and multipole order $m = 6$. We solve for flow around a distribution of 250 spheres each of radius 5E-2 in a unit cube. The configuration is similar to that visualized in Figure 3. We use the penalty method with η = 1E+9 inside the spheres and zero outside. We adaptively refine our mesh on the boundary of the spheres and the problem size is determined by the maximum refinement level L. Here, we vary L from 5 to 10 and increase the number of compute nodes while keeping the number of unknowns per processor (N_{dof}/p) fixed at roughly one million. In Figure 4, we present a breakdown of the time spent in each stage of the solver. Of the total solve time, GMRES corresponds to the time spent internally in the PETSc's Krylov subspace iterative solver. In each iteration of GMRES, a matrix multiplication operation (the LHS in equation 5), labeled as MatMul, is performed. In this operation, the convolution with the Green's function is implemented using our volume FMM and we show the time spent in computation (FMMcomp) and communication (FMMcomm). We show results for execution on CPU and CPU+Phi as we increase the number of compute nodes from 1 to 2048. We observe that the FMM accounts for over 60% of the total solve time. As expected, the FMM computation stage is about 15% faster with the co-processor than without it. The communication cost of FMM increases gradually as we increase the number of MPI processes. This trend appears to be consistent with the expected $\mathcal{O}(\log p)$ complexity estimate. For very large process counts, the scalability of GMRES appears to suffer and there is a significant increase in the time spent in this stage for 2048 compute nodes. In Table 10, we show performance results for CPU+Phi case. Overall the code scales well and on 2048 compute nodes we achieve 150TFLOPS with 61% parallel efficiency η. We lose some performance due to communication overhead and increase in time spent in the GMRES stage.

In Figure 5 and Table 11 we solve for Stokes flow using the geometry visualized in Figure 1. Since we examined the behavior of the solver in the case of low-order approximations above, here we consider the case with of high-order approximations. Here we have used a high-order discretization with $q = 14$ and multipole order $m = 10$. In this test case, we see significant speedup ($\sim 2\times$) using the Phi co-processor since we have enough work in the near interactions. Consequently, we also get significantly higher FLOP rates, achieving 286TFLOPS on 1024 compute nodes and a parallel efficiency $\eta = 0.79$. Again the number of GMRES iterations is significant due the large $\|\rho\|_\infty$.

3.3 Strong Scalability

We report strong scalability results where we fix the problem size and increase the number of compute nodes. In Figure 6 and Table 12, we simulate flow around a random distribution of 250 spheres using high-order discretization ($q = 14$, $m = 10$) and with an adaptive mesh refined to 7 levels corresponding to 232 million unknowns. We solved GMRES to a residual tolerance ϵ_{gmres} = 1E-8 and the solve required 101 iterations. As we increase the number of com-

						CPU		CPU+Phi		CPU+GPU	
q	N_{oct}	L	N_{iter}	L^∞	L^2	T_{solve}	GFLOPS	T_{solve}	GFLOPS	T_{solve}	GFLOPS
8	32,768	6	48	3.1E-6	1.0E-6	1026.4	148	940.0	162	942.9	161
10	32,768	6	46	1.0E-7	3.0E-7	1157.9	161	963.3	194	952.1	196
12	4,096	5	49	3.2E-6	1.4E-6	183.5	184	129.3	262	123.9	273
14	4,096	5	47	1.7E-7	3.1E-7	259.3	210	138.0	394	128.5	424
16	4,096	5	46	8.3E-8	3.1E-7	384.3	242	159.5	582	140.2	662
6	29,240	9	41	4.8E-7	4.0E-7	722.6	148	681.0	157	669.1	159
8	4,656	8	41	6.6E-7	6.7E-7	118.8	160	101.0	188	97.4	195
10	2,024	7	41	2.5E-7	3.5E-7	66.9	166	44.3	251	42.4	262
12	1,240	6	44	1.1E-7	3.1E-7	63.4	179	31.4	360	30.3	374
14	736	6	42	9.2E-8	3.0E-7	56.4	192	23.2	467	19.1	565
16	232	5	41	1.5E-7	3.2E-7	38.8	147	15.6	366	9.8	579

Table 9 *Single node performance results for a variable coefficient problem ($\|\rho\|_\infty$ = 1E+6) for uniform and non-uniform meshes with increasing Chebyshev degree q and fixed multipole order $m = 10$, GMRES tolerance ϵ_{gmres} = 1E-9. This example demonstrates two orders of magnitude speed-up from a 9th-order, uniform grid approximation, to a 17th-order GPU-accelarated, adaptive scheme reducing 1026 seconds to 9.8 seconds.*

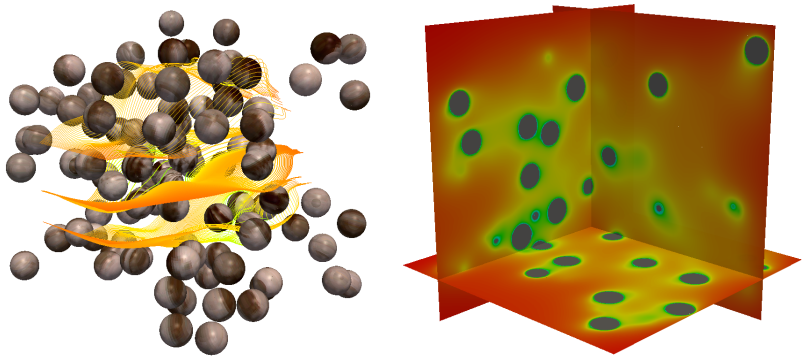


Figure 3 *Here, we solve Stokes flow around a random distribution of spheres. On the left we visualize the streamlines around the spheres. The figure on the right shows cross sections of the magnitude of the velocity field with velocity field increasing from green to red.*

pute nodes from 32 to 1024, we get 9× and 7× speedup for the CPU and CPU+Phi respectively. Overall, we achieve 94.8TFLOPS and a parallel efficiency of 22%.

Next, in Figure 7 and Table 13, we use the porous medium geometry as a test case and use low order discretization for the solver. The octree is refined to 8 levels and we have 169 million unknowns. Due to the low-order discretization, we gain little from using the Phi accelerator and get about 24TFLOPS for both CPU and CPU+Phi on 1024 compute nodes. Scaling from 32 compute nodes to 1024 compute nodes, we get similar parallel efficiency as before.

4. CONCLUSIONS

We have presented a novel scheme for solving the Stokes equations with variable coefficients. We demonstrated the convergence of our scheme and its efficiency for constant and variable coefficients and showed scalability on hybrid architectures. For smooth problems, we demonstrated that the combination of high-order accuracy, adaptivity, integration with accelerators, algorithmic optimality, and distributed memory parallelism can result in many orders of magnitude speedups.

Here we have only scratched the surface of the capabilities and challenges of the proposed methodologies. There remain challenges in terms of performance, preconditioners, general geometries, and of course careful verification for porous media flows.

For more general geometries and boundary conditions, one can use block solvers as we discussed before. The transformation Jacobian from an arbitrary hexahedral domain to the unit cube, can be easily handled by our variable coefficient solver. As long the number of hexahedral elements is reasonably small and the elements are well shaped (for example in the absence of large anisotropy), we can handle non-cubic geometries by using ideas similar to macromesh construction codes [8], overset grids and other similar ideas.

Another issue that we do not discussed here is the need for preconditioning. We saw an increase in the number of iterations as ρ_∞ increases significantly. Many algorithms for preconditioning integral equations exist but none has been scaled to such complexity. Promising schemes include multilevel solvers and hierarchical inexact factorizations. These aspects are currently being investigated in our group.

There are several other issues regarding performance. When the average ρ_0 is not zero, the Green's function is not scale invariant. This means that the precomputed matrices for near interactions will depend on the level of the source leaf octant. This will increase the storage requirements and reduce performance. Variability on the viscosity does not cause any problems.

Finally, although we get excellent performance for high values of q, this is not the case with reducing q. The performance we observe is still quite good compared to FEM codes but more work is needed to ensure a more robust performance envelope.

Overall, our scheme provides an excellent starting point that resolves several outstanding issues: scalability with respect high-order elements, more freedom in using discretization spaces, and excellent utilization of massively parallel hybrid architectures.

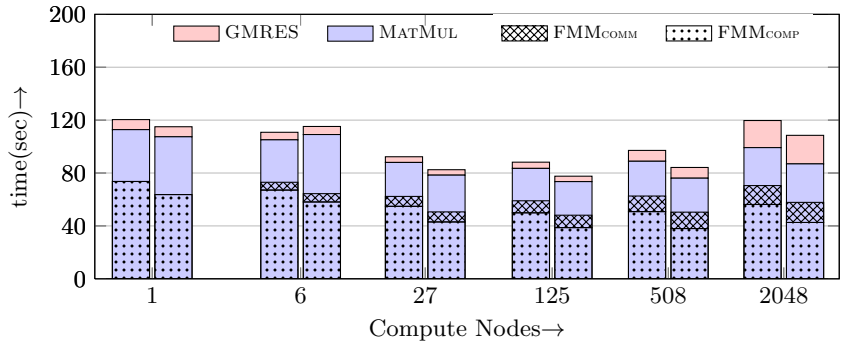


Figure 4 *Weak scalability results for low-order discretization ($q = 6$, $m = 6$) showing a breakdown of time in seconds for different stages of the solver for CPU and CPU+Phi runs respectively. We solve for flow around a random distribution of 250 spheres.*

p	N_{dof}/p	N_{iter}	T_{solve}	TFLOPS	η
1	9.8E+5	200	115.0	0.12	1.00
6	9.5E+5	163	115.2	0.62	0.86
27	1.0E+6	116	82.5	3.12	0.96
125	1.0E+6	99	77.6	13.0	0.87
508	1.1E+6	92	84.2	47.9	0.78
2048	1.1E+6	90	108.5	150	0.61

Table 10 *Solve time, total* FLOP *rate and parallel efficiency of the solver for flow around 250 spheres, for weak scaling up to 2K compute nodes on Stampede using Phi co-processor.*

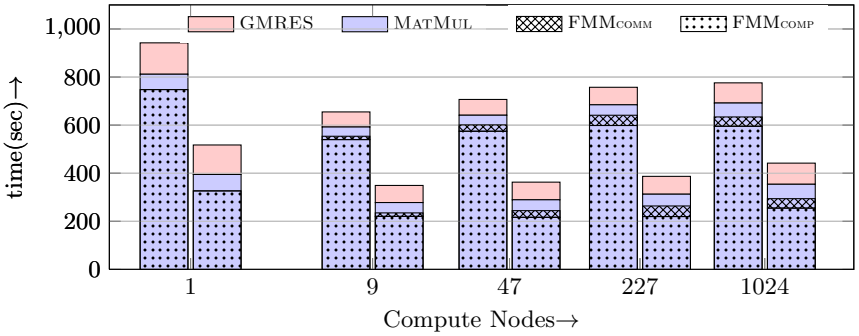


Figure 5 *Weak scalability results for high-order discretization ($q = 14$, $m = 10$) for flow through a porous media. We show breakdown of time in seconds for different stages of the solver for CPU and CPU+Phi runs respectively.*

p	N_{dof}/p	N_{iter}	T_{solve}	TFLOPS	η
1	8.4E+6	158	518	0.35	1.00
9	6.0E+6	128	349	3.12	0.98
47	6.1E+6	124	363	16.6	1.00
227	6.1E+6	115	387	73.4	0.91
1024	6.1E+6	111	442	286	0.79

Table 11 *Solve time, total* FLOP *rate and parallel efficiency of the solver for flow through a porous structure, for weak scaling up to 1K compute nodes on Stampede using Phi co-processor.*

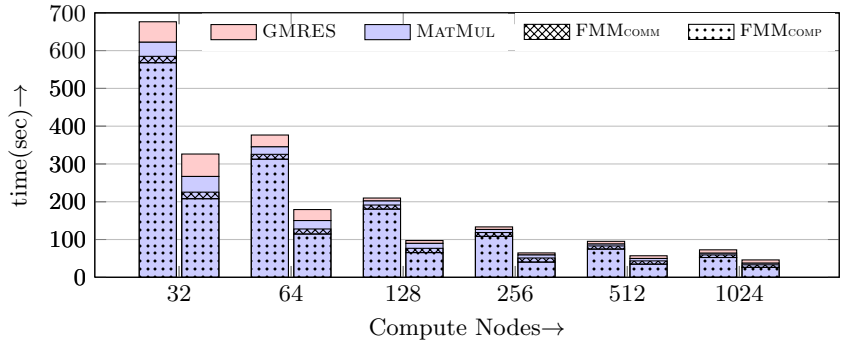


Figure 6 *Strong scalability up to 1024 compute nodes using high-order discretization ($q = 14$, $m = 10$) for flow around spheres for 232 million unknowns. We report breakdown of time in seconds for different stages of the solver for CPU and CPU+Phi runs respectively.*

	CPU			CPU+Phi		
p	T_{solve}	TFLOPS	η	T_{solve}	TFLOPS	η
32	676	6.11	1.00	326	12.7	1.00
64	377	11.0	0.90	179	23.1	0.91
128	210	19.8	0.81	97	42.7	0.84
256	133	31.4	0.63	65	64.9	0.63
512	95	44.6	0.44	57	74.6	0.36
1024	73	60.0	0.29	46	94.8	0.22

Table 12 *Solve time, total* FLOP *rate and parallel efficiency of the solver for flow over 250 spheres, for strong scaling up to 1K compute nodes on Stampede.*

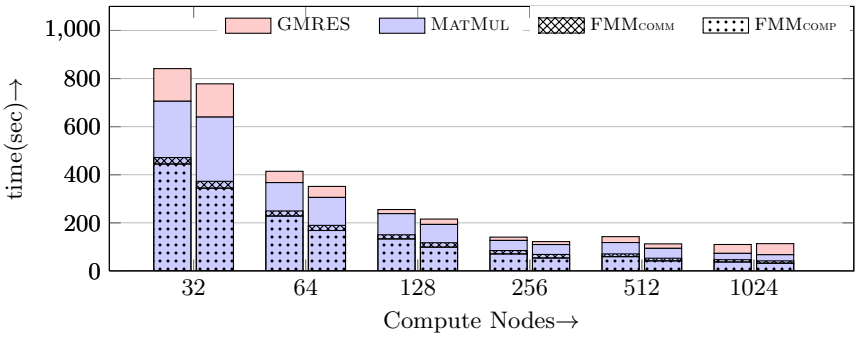


Figure 7 *Strong scalability using low-order discretization ($q = 6$, $m = 6$) for flow through a porous medium using 169 million unknowns. We report breakdown of time in seconds for different stages of the solver for CPU and CPU+Phi runs respectively.*

	CPU			CPU+Phi		
p	T_{solve}	TFLOPS	η	T_{solve}	TFLOPS	η
32	841	3.0	1.00	778	3.2	1.00
64	414	6.1	1.02	351	7.2	1.11
128	255	9.9	0.83	216	11.7	0.91
256	140	18.1	0.76	122	20.9	0.81
512	142	18.1	0.38	112	22.9	0.44
1024	109	24.0	0.25	113	23.1	0.22

Table 13 *Solve time, total* FLOP *rate and parallel efficiency of the solver for flow through a porous media, for strong scaling up to 1K compute nodes on Stampede.*

Acknowledgment

We would like to thank Wolfgang Bangerth and Timo Heister for their help with deal.II. This material is based upon work supported by AFOSR grants FA9550-12-10484 and FA9550-11-10339; and NSF grants CCF-1337393, OCI-1029022, and OCI-1047980; and by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Numbers DE-SC0010518, DE-SC0009286, and DE-FG02-08ER2585. Computing time on the Texas Advanced Computing Centers Stampede system was provided by an allocation from TACC and the NSF.

5. REFERENCES

[1] P. Angot, C.-H. Bruneau, and P. Fabrie, *A penalization method to take into account obstacles in incompressible viscous flows*, Numerische Mathematik, 81 (1999), pp. 497–520.

[2] S. Balay, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B. F. Smith, and H. Zhang, *PETSc Web page*. http://www.mcs.anl.gov/petsc, 2014.

[3] W. Bangerth, R. Hartmann, and G. Kanschat, *deal.II – a general purpose object oriented finite element library*, ACM Transactions in Mathematical Software, 33 (2007), pp. 24/1–24/27.

[4] W. Bangerth, T. Heister, L. Heltai, G. Kanschat, M. Kronbichler, M. Maier, B. Turcksin, and T. D. Young, *The* `deal.ii` *library, version 8.1*, arXiv preprint http://arxiv.org/abs/1312.2266v4, (2013).

[5] M. Benzi, G. H. Golub, and J. Liesen, *Numerical solution of saddle point problems*, Acta numerica, 14 (2005), pp. 1–137.

[6] J. Brown, *Efficient nonlinear solvers for nodal high-order finite elements in 3D*, Journal of Scientific Computing, 45 (2010), pp. 48–63.

[7] C. Burstedde, O. Ghattas, M. Gurnis, T. Isaac, G. Stadler, T. Warburton, and L. Wilcox, *Extreme-scale amr*, in Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, IEEE Computer Society, 2010, pp. 1–12.

[8] C. Burstedde, O. Ghattas, G. Stadler, T. Tu, and L. C. Wilcox, *Parallel scalable adjoint-based adaptive solution of variable-viscosity Stokes flow problems*, Computer Methods in Applied Mechanics and Engineering, 198 (2009), pp. 1691–1700.

[9] C. Burstedde, G. Stadler, L. Alisic, L. C. Wilcox, E. Tan, M. Gurnis, and O. Ghattas, *Large-scale adaptive mantle convection simulation*, Geophysical Journal International, 192 (2013), pp. 889–906.

[10] X.-C. Cai and O. B. Widlund, *Domain decomposition algorithms for indefinite elliptic problems*, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 243–258.

[11] J. Carrier, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm for particle simulations*, SIAM Journal on Scientific and Statistical Computing, 9 (1988), pp. 669–686.

[12] D. D. Chandar, J. Sitaraman, and D. J. Mavriplis, *A GPU-based incompressible Navier–Stokes solver on moving overset grids*, International Journal of Computational Fluid Dynamics, 27 (2013), pp. 268–282.

[13] D. Colton and R. Kress, *Inverse Acoustic and Electromagnetic Scattering Theory, 2nd Edition*, Applied Mathematical Sciences, Springer, 1998.

[14] F. Ethridge and L. Greengard, *A new fast-multipole accelerated poisson solver in two dimensions*, SIAM Journal on Scientific Computing, 23 (2001), pp. 741–760.

[15] M. Gee, C. Siefert, J. Hu, R. Tuminaro, and M. Sala, *ML 5.0 smoothed aggregation user's guide*, Tech. Rep. SAND2006-2649, Sandia National Laboratories, 2006.

[16] L. Greengard and M. C. Kropinski, *An integral equation approach to the incompressible Navier–Stokes equations in two dimensions*, SIAM Journal on Scientific Computing, 20 (1998), pp. 318–336.

[17] M. D. Gunzburger, *Finite Element for Viscous Incompressible Flows*, Academic Press, 1989.

[18] R. Kress, *Newton's method for inverse obstacle scattering meets the method of least squares*, Inverse Problems, 19 (2003).

[19] M. Kronbichler, T. Heister, and W. Bangerth, *High accuracy mantle convection simulation through modern numerical methods*, Geophysical Journal International, 191 (2012), pp. 12–29.

[20] H. Langston, L. Greengard, and D. Zorin, *A free-space adaptive FMM-based PDE solver in three dimensions*, Communications in Applied Mathematics and Computational Science, 6 (2011), pp. 79–122.

[21] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros, *A massively parallel adaptive fast multipole method on heterogeneous architectures*, Communications of the ACM, 55 (2012), pp. 101–109.

[22] D. Malhotra and G. Biros, `pvfmm`*: A distributed memory fast multipole method for volume potentials*, 2014. submitted.

[23] C. Pozrikidis, *Boundary Integral and Singularity Methods for Linearized Viscous Flow*, Cambridge University Press, 1992.

[24] A. Rahimian, I. Lashuk, S. Veerapaneni, A. Chandramowlishwaran, D. Malhotra, L. Moon, R. Sampath, A. Shringarpure, J. Vetter, R. Vuduc, D. Zorin, and G. Biros, *Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures*, in SC '10: Proceedings of the 2010 ACM/IEEE conference on Supercomputing, IEEE Press, 2010, pp. 1–12.

[25] P. K. Smolarkiewicz and C. L. Winter, *Pores resolving simulation of darcy flows*, Journal of Computational Physics, 229 (2010), pp. 3121 – 3133.

[26] L. Trefethen, *Spectral methods in MATLAB*, Society for Industrial Mathematics, 2000.

[27] L. Ying, G. Biros, and D. Zorin, *A kernel-independent adaptive fast multipole method in two and three dimensions*, Journal of Computational Physics, 196 (2004), pp. 591–626.