

COMPARISON BETWEEN ALGEBRAIC AND MATRIX-FREE GEOMETRIC MULTIGRID FOR A STOKES PROBLEM ON ADAPTIVE MESHES WITH VARIABLE VISCOSITY

THOMAS C. CLEVINGER

tcleven@clemson.edu, Clemson University

TIMO HEISTER

heister@clemson.edu, Clemson University

ABSTRACT. Problems arising in Earth’s mantle convection involve finding the solution to Stokes systems with large viscosity contrasts. These systems contain localized features which, even with adaptive mesh refinement, result in linear systems that can be on the order of 10^9 or more unknowns. One common approach for preconditioning to the velocity block of these systems is to apply an Algebraic Multigrid (AMG) v-cycle (as is done in the ASPECT software, for example), however, with AMG, robustness can be difficult with respect to problem size and number of parallel processes. Additionally, we see an increase in iteration counts with adaptive refinement when using AMG. In contrast, the Geometric Multigrid (GMG) method, by using information about the geometry of the problem, should offer a more robust option.

Here we present a matrix-free GMG v-cycle which works on adaptively refined, distributed meshes, and we will compare it against the current AMG preconditioner (Trilinos ML) used in the ASPECT[3] software. We will demonstrate the robustness of GMG with respect to problem size and show scaling up to 114688 cores and 217 billion unknowns. All computations are run using the open source, finite element library `deal.II`. [1]

1. INTRODUCTION

The major bottleneck of computations of processes in Earth’s mantle convection is the solution of Stokes systems to solve for velocity and pressure. These systems often have large variation in their coefficients, as well as highly localized features requiring adaptive mesh refinement to yield high enough resolution while still being computationally feasible to solve. Even with adaptive refinement, there is a desire to solve on meshes containing well over a billion unknowns, which many current open-source codes are not equipped to handle.

When open-source codes like the mantle convection code ASPECT[3] were first developed, the state-of-the-art solvers often relied on algebraic multigrid (AMG) methods when preconditioning on the velocity space of the Stokes finite element discretizations (see, e.g., the works of Geenen et al.[11] and Kronbichler et al.[17]). While these algebraic methods can be very powerful for smaller problems, their performance tends to deteriorate with highly adaptive meshes and when distributing the problem over a large number of cores.

Key words and phrases. geometric multigrid, matrix-free finite elements, parallel computing, adaptive mesh refinement, Stokes equations.

In addition, these methods require the storing of matrices which greatly limits the size of the problem which can be solved, and can provide a major computational bottleneck as the access of matrix entries from main memory starts to become slower than the actual computation with those entries.[19]

In response to these limitations, much of the research into solving the Stokes systems is now focused on using either geometric multigrid (GMG) or a hybrid multigrid (HMG) (where AMG is used on coarse meshes of the GMG method) for preconditioning on the velocity space. These methods have the advantage that, by using geometric information about the problem to construct the multigrid level hierarchy, there should be less deterioration with highly adaptive meshes. Also, using these geometric methods allow for the implementation of matrix-free operator evaluation which not only reduces the memory required for an application (allowing one to solve larger problems), but also can lead to faster matrix-vector products, particularly for higher order finite elements.[18] These methods have lead to solutions of systems on the order of 10^{13} unknowns[13] (low order, globally refined tetrahedral mesh) and 10^{11} unknowns[21] (higher order, adaptively refined hexahedral mesh).

Here we will present a comparison of a Stokes application using the matrix-free GMG method presented in the work of Clevenger et al.[8] and implemented by the authors into ASPECT with the current AMG-based method described in the work of Kronbichler et al.[17] and Heister et al.[14] (with Trilinos ML AMG) widely used by the ASPECT community. We will demonstrate the advantages of the geometric method over the algebraic method, as well as show weak and strong scalability of the geometric algorithm on the sinker benchmark used in the work of May et al.[20] and Rudi et al.[22]

2. LINEAR SOLVER FOR THE STOKES EQUATIONS

We will consider the Stokes equations in the form

$$(1) \quad \begin{aligned} -\nabla \cdot (2\mu \varepsilon(\mathbf{u})) + \nabla p &= \mathbf{f} & \text{in } \Omega \\ \nabla \cdot \mathbf{u} &= 0 & \text{in } \Omega \\ \mathbf{u} &= 0 & \text{in } \partial\Omega, \end{aligned}$$

where \mathbf{u} denotes the fluid's velocity, p it's pressure, and $\mu(\mathbf{x})$ it's viscosity. Here we use the strain-rate tensor $\varepsilon(\mathbf{u}) = \frac{1}{2}(\nabla \mathbf{u} + (\nabla \mathbf{u})^T)$ as we will be considering non-constant viscosity. The right hand side \mathbf{f} is a forcing term that, inside ASPECT, typically comes from temperature variation, but for our purposes here, we will only be considering Stokes with velocity and pressure variables and a manufactured right hand side. Finally we are considering incompressible flow with homogeneous Dirichlet boundary conditions, but the developed algorithms also apply to the more general case of $\nabla \cdot \mathbf{u} = g$, as well as for no-normal-flux boundary constraints $\nabla \mathbf{u} \cdot \mathbf{n} = 0$.

2.1. Discretization. Following traditional finite element methods, and using the notation in the work of Kronbichler et al.,[17] we seek coefficients \mathbf{u}_j and p_j where, for finite element shape functions φ_j^u and φ_j^p ,

$$(2) \quad \begin{aligned} \mathbf{u}_h &= \sum_{j=1}^{N_u} \mathbf{u}_j \varphi_j^u \\ p_h &= \sum_{j=1}^{N_p} p_j \varphi_j^p \end{aligned}$$

such that the weak formulation

$$(3) \quad \begin{aligned} a(\varphi_i^u, \mathbf{u}_h) + b(\varphi_i^u, p_h) &= f(\varphi_i^u) \\ b(\mathbf{u}_h, \varphi_i^p) &= 0, \end{aligned}$$

defined by

$$a(\varphi_i^u, \mathbf{u}_h) = \int_{\Omega} 2\mu \varepsilon(\varphi_i^u) : \varepsilon(\mathbf{u}_h) \, dx \quad b(\varphi_i^u, p_h) = - \int_{\Omega} (\nabla \cdot \varphi_i^u) p_h \, dx \quad f(\varphi_i^u) = \int_{\Omega} \varphi_i^u \cdot \mathbf{f} \, dx,$$

holds for each $1 \leq i \leq N_u$ and $1 \leq l \leq N_p$ where N_u and N_p are the total number of degrees of freedom for velocity and pressure respectively. We will choose the shape functions from the Taylor-Hood elements $[\mathbb{Q}_{k+1}]^{\text{dim}} \times \mathbb{Q}_k$, $k \geq 1$, which are known to be stable for the system considered here.[10, 6]

Solving this system for coefficients $U = \{u_i\}$ and $P = \{p_j\}$ is then equivalent to solving the block linear system

$$(4) \quad \begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} U \\ P \end{pmatrix} = \begin{pmatrix} F \\ 0 \end{pmatrix}$$

where

$$A_{ij} = \int_{\Omega} 2\mu \varepsilon(\varphi_i^u) : \varepsilon(\varphi_j^u) dx \quad B_{ij} = - \int_{\Omega} (\nabla \cdot \varphi_j^u) \varphi_i^p dx \quad F_j = \int_{\Omega} \varphi_j^u \cdot \mathbf{f} dx.$$

2.2. Linear Solver. There are three common approaches used in the literature for solving (4) on large scales: (i) a pressure corrected, Schur complement CG scheme, using multigrid as an approximation to the velocity block,[13], (ii) a block-preconditioned Krylov method, also using multigrid on the velocity block, and (iii) an all-at-once multigrid performed on the entire Stokes system, using Uzawa-type smoothers.[13, 4] For method (ii), there are two main types:

(a) GMRES[20, 21] (or any Krylov method not requiring symmetry) with block-triangular preconditioner

$$(5) \quad P = \begin{pmatrix} A & B^T \\ 0 & -S \end{pmatrix},$$

or

(b) MINRES[13] with block-diagonal preconditioner

$$(6) \quad P_D = \begin{pmatrix} A & 0 \\ 0 & -S \end{pmatrix}$$

where $S = BA^{-1}B^T$ is the *Schur complement*. [9]

In the work of Gmeiner et al.,[13] a comparison is given for each method (using MINRES for (ii)), and the all-at-once multigrid method (iii) was shown to give the fastest and most consistent convergence, as well as the lowest memory overhead. However, there is no comparison that we are aware of for method (ii) using the preconditioner P defined in (5), and both method have been used separately to demonstrate impressive scaling results (see the work of Bauer et al.[4] for method (iii) and Rudi et al.[21] for method (ii) with GMRES).

For the computations in this paper, we will choose method (ii), as this is the framework used in ASPECT. For preconditioning, we choose P defined in (5), since, as illustrated by Table 1, this leads to roughly half the solve time for a GMRES solve as compared to using P_D . This is due to the fact that the block-triangular preconditioner is roughly the same cost to apply as the block-diagonal preconditioner (only containing an extra B^T matrix-vector product), but leads to roughly half the total GMRES iterations.¹

Now, for the preconditioner P , consider the following preconditioned system

$$(7) \quad \begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} A & B^T \\ 0 & -S \end{pmatrix}^{-1} = \begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} A^{-1} & A^{-1}B^TS^{-1} \\ 0 & -S^{-1} \end{pmatrix} = \begin{pmatrix} I & 0 \\ BA^{-1} & I \end{pmatrix}.$$

This system has only 1 distinct eigenvalue $\lambda = 1$ and will converge in at most 2 GMRES iterations assuming an exact preconditioner. Computing exact representations for A^{-1} and S^{-1} is highly impractical, therefore we seek approximate \hat{A}^{-1} and \hat{S}^{-1} , respectively.

¹It should be noted that the majority of the time P_D is used with MINRES instead of GMRES. However, we do not expect the MINRES to significantly outperform GMRES in runtimes.

Preconditioner:	P			P_D		
	iters	Solve time	Time/iter	iters	Solve time	Time/iter
860K DoFs	26	0.671	0.026	54	1.294	0.020
1.8M DoFs	27	1.829	0.068	56	4.799	0.086
3.8M DoFs	28	5.320	0.190	55	10.457	0.190
7.7M DoFs	28	11.416	0.408	57	21.142	0.371

TABLE 1. Iterations required to reduce the residual by 10^6 , time to solution (in seconds), and average time per iteration of a GMRES solve for the Sinker benchmark defined in Section 3.2 (4 sinkers and a viscosity ratio of 10^4), on a series of adaptively refined meshes, and using 32 cores on a workstation. \hat{A} and \hat{S} approximations are defined in Section 2.2. Here we test the block-triangular preconditioner P against the block-diagonal preconditioner P_D . Using P results in roughly half the iterations and solve time compared to P_D .

2.2.1. *Choosing \hat{A}^{-1} .* Since A comes from a vector Laplacian equation, multigrid would appear to be a logical choice given that these methods are widely known to have convergence independent of mesh size h for elliptic boundary value problems.[5, 28]

Currently in ASPECT, \hat{A}^{-1} is approximated by 1 AMG V-cycle for each Krylov subspace iteration, however the AMG method is not based on the matrix A in (4), but instead we consider the matrix \hat{A} defined as

$$(8) \quad \hat{A}_{ij} = \sum_{d=1}^{\dim} \int_{\Omega} 2\mu [\varepsilon(\varphi_i^u)]_{d,d} : [\varepsilon(\varphi_j^u)]_{d,d}$$

where $[\varepsilon(\varphi_i^u)]_{d,d} = [\nabla \varphi_i^u]_{d,d}$ is the d th diagonal entry of the gradient operator. The reasons for using this partial coupling of velocity components are the following:

- (i) AMG methods, which depend on the sparsity structure of the underlying matrix, tend to deteriorate when coupling vector components in higher order computations,[11] and
- (ii) the resulting matrix \hat{A} will have far fewer entries (1/3 the entries in 3D, see, e.g., Figure 1), and therefore less storage requirements, faster AMG setup and faster application.

We will test this implementation of \hat{A}^{-1} using an AMG v-cycle against one using the GMG v-cycle developed in the work of Clevenger et al.[8] For this geometric method, we consider a matrix-free v-cycle based on a degree 4 Chebyshev smoother with one smoothing step per level. In contrast with the AMG v-cycle, the level matrices of the GMG v-cycle are based on the fully coupled system A , not the partially coupled system \hat{A} since, for the geometric method, there should be no deterioration when using a fully coupled system (contrast to item (i) above), and, as we are using matrix-free operators where each matrix entry is computed on-the-fly at the quadrature level, using the strain rate tensor only consists of adding the off diagonal term in the correct place and dividing by 2; essentially, it is for free (contrast to item (ii) above). From item (iii) above, this could give the GMG-based method an advantage over an AMG-based method in terms of the effectiveness of the preconditioner, while not requiring any extra storage or computational time.

2.2.2. *Choosing \hat{S}^{-1} .* A common choice for approximating $S = BA^{-1}B^T$ is a weighted pressure mass matrix M_p , where $[M_p]_{i,j} = \int_{\Omega} \mu^{-1} \varphi_i^p, \varphi_j^p dx$. [24, 9, 17] The reasons for this is that S and M_p are spectrally equivalent for constant viscosity,[9] making M_p^{-1} a good approximation to S^{-1} , while M_p^{-1} is far easier to compute. The application of \hat{S}^{-1} then is a simple CG solve with an ILU preconditioner (for matrix-based AMG method) or a Chebyshev iteration (for matrix-free GMG method), converging in between 1-5 iterations.

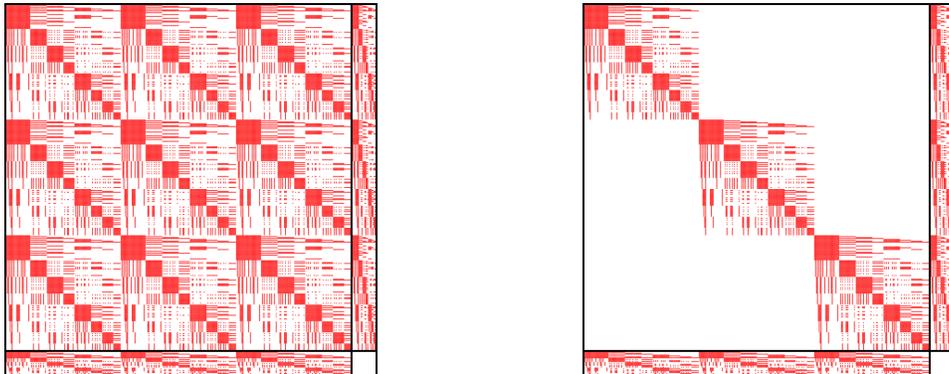


FIGURE 1. Sparsity pattern for the matrix (4) for full coupling of vector components (left) and partial coupling (right). 3D Stokes with 402 degrees of freedom (375 velocity, 27 pressure), ordered component-wise. A red pixel represents a non-zero matrix entry. We see that for partial coupling we have approximately 3x fewer entries as with full coupling.

Compared with \hat{A}^{-1} , it is not computationally significant.[17] Since the application of P now requires a CG solve whose iteration count may change between applications, a flexible Krylov subspace method must be used for the outer iteration. We will use the flexible variant of GMRES referred to as FGMRES. This solver has the same guaranteed convergence property as GMRES[23], however it will require the storage of 2 temporary vectors per iteration (up to the restart length).

It should be noted that this Schur complement approximation begins to break down for large $\text{DR}(\mu)$ (see the work of Rudi et al.[22]) and a more sophisticated approach may be required in those cases.

2.3. Viscosity Averaging. The viscosity $\mu(\mathbf{x})$ is evaluated at each quadrature point on the finest cells and then averaged using harmonic averaging to obtain a piece-wise constant representation per cell. Since in many applications the viscosity will not come from a functional representation (and therefore cannot be easily evaluated on coarser grids), we must have a way to transfer this cell-wise viscosity on the active mesh to a viscosity on the meshes throughout the level hierarchy. We accomplish this by transferring these coefficients using the same grid restriction operator as discussed in the work of Clevenger et al.,[8] the work of Brenner and Scott,[6] and the work of Janssen and Kanschat,[16] based on a $\mathbb{Q}_0^{\text{disc}}$ element (one constant per cell). Essentially, this involves averaging the active cells viscosity, and then recursively setting the the viscosity of a parent cell in the hierarchy to the average of the viscosity of each of its children.

It should be noted that, for problems where the viscosity is represented by a smooth function, this harmonic averaging of the viscosity on the active mesh can result in the loss of convergence order of the solution. However from the work of Heister et al.,[14], for problems with highly discontinuous viscosity, averaging viscosity can be beneficial for controlling under- and over-shoots in the numerical approximations of the solution gradient. We do not consider alternatives to harmonic averaging here.

3. RESULTS

3.1. Software. For all computations we will be using the open-source library `deal.II`,[1] which offers scalable parallel algorithms for finite element computations. The `deal.II` library uses functionality from other libraries such as Trilinos[15] (for linear algebra, including Trilinos ML AMG preconditioner) and `p4est`[7] (for mesh partitioning).

As mentioned above, we will be comparing our solver to the one used by ASPECT. ASPECT is an open-source library written on top of deal.II for the specific purpose of solving problems related to the earth’s mantle. The method presented here is available in the current development version of ASPECT.

3.2. Benchmark Problem. We give results demonstrating the scalability of the GMG-based method developed here, as well as a comparison with the AMG-based method used in ASPECT. The test problem used is the “Sinkers” benchmark described in the work of May et al.[20] and the work of Rudi et al.[22] It consists of solving the Stokes problem (1) on the unit cube domain, with n randomly positioned “sinkers” of higher viscosity throughout the domain. By specifying $\text{DR}(\mu)$, we define a smooth viscosity by $\mu(\mathbf{x}) \in [\mu_{\min}, \mu_{\max}]$ where for $X(\mathbf{x}) \in [0, 1]$

$$X(\mathbf{x}) = \prod_{i=1}^n \left[1 - \exp \left(-\delta \max \left[0, |\mathbf{c}_i - \mathbf{x}| - \frac{\omega}{2} \right]^2 \right) \right],$$

$$\mu(\mathbf{x}) = X(\mathbf{x})\mu_{\min} + (1 - X(\mathbf{x}))\mu_{\max}.$$

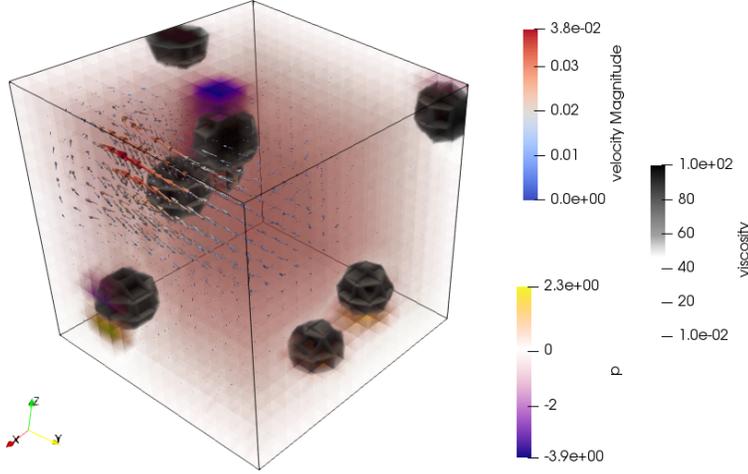
Here $\mu_{\min} = \text{DR}(\mu)^{-1/2}$, $\mu_{\max} = \text{DR}(\mu)^{1/2}$, \mathbf{c}_i are the center of each sinker, $\delta = 200$ controls the exponential decay of the viscosity, and $\omega = 0.1$ is the diameter of the sinkers. The right hand side is given by $\mathbf{f}(\mathbf{x}) = (0, 0, \beta(X(\mathbf{x}) - 1))$ with $\beta = 10$ and we use homogeneous Dirichlet boundary conditions for the velocity. Physically, this represents gravity pulling down the high viscosity sinkers. Figure 2 gives a representation of both the velocity and the pressure solution of this benchmark.

The problem difficulty can be increased by increasing n or $\text{DR}(\mu)$. Table 2 gives the iterations required to reduce the residual of the outer FGMRES solve by 10^6 for different values of these parameters. We see that both AMG and GMG deteriorate as both n and $\text{DR}(\mu)$ increase, with GMG being slightly more robust. This problem can likely be addressed using methods derived in the work of Rudi et al.[22], where it was shown that this deterioration is due to the approximation loss in the Schur complement solve, and a more sophisticated Schur complement approximation was proposed based on least squares communicators. For the remaining results, we will only consider $n = 4$ and $\text{DR}(\mu) = 1e4$, as this is within the range of problems where our Schur complement approach is sufficient.

All timings in this section were from computations run on either Frontera or Stampede2, both at The University of Texas at Austin’s Texas Advanced Computing Center. For the Frontera runs, we will be using the Intel Xeon Platinum 8280 (Cascade Lake) nodes which have 56 cores and 192GB per node, and for the Stampede2 runs, we will be using the Intel Xeon Platinum 8160 (Skylake) nodes which have 48 cores and 192GB per node. Both support AVX-512 instructions allowing for vectorization over 8 doubles. The deal.II version used is 9.1.0-pre, and we compile using gcc 7.1.0, intel-mpi 17.0.3. The p4est version is 2.0.0, the Trilinos version is 12.10.1 and the ASPECT version is 2.1.0-pre.

Each line in the timing plots connect the median time of 5 distinct runs, with all 5 runs shown as points. We will test strong scaling (problem size stays constant, number of cores increase) and weak scaling (problem size per core stays constant) for key parts of the computation. Tests run on adaptively refined meshes will include the model for partition imbalance described in the work of Clevenger et al.[8] This model should represent the ideal scaling we can expect to see for the GMG method given the imbalance of cells in the level hierarchy.

3.3. AMG/GMG Comparison. For a comparison between the AMG and GMG preconditioners, we will consider an adaptively refined mesh, where for each refinement, the number of cells are roughly doubled. We start with a mesh of 4 global refinements and create each new mesh using a gradient based estimator to refine roughly 1/7 of the cells from the previous refinement cycle, doubling the number of cells in our mesh. Trilinos ML AMG is used with a Chebyshev smoother containing two sweeps and an aggregation threshold



DR(μ)	1e2	1e4	1e6
AMG			
4 sinkers	46	48	52
8 sinkers	81	196	229
12 sinkers	76	171	237
GMG			
4 sinkers	20	24	26
8 sinkers	37	72	128
12 sinkers	39	80	141

FIGURE 2. $n = 8$, $DR(\mu) = 1e4$

TABLE 2. FGMRES iterations required to reduce the residual by 10^6 for the sinker benchmark with increasing n and $DR(\mu)$. Run on a 3D mesh with 860K degrees of freedom ($[Q_2]^{\text{dim}} \times Q_1$ element), distributed over 32 cores.

	AMG	GMG	factor
Setup	12.6s	10.3s	1.2x
Assemble	32.5s	2.9s	11.2x
Solve	38.6s	14.8s	2.6x
Total	83.7s	28.0s	3.0x

TABLE 3. Timing comparison between AMG and GMG for an adaptively refined, 3D mesh, with 18.5×10^6 DoFs ($[Q_2]^{\text{dim}} \times Q_1$ element) on one node (48 cores). Timings are from the Stampede2 machine.

of 0.001. These values are chosen here as they are the default values in ASPECT and represent the values used by the majority of the users of the code.

Table 3 gives the runtimes for such a mesh with 5 levels of adaptive refinement on 48 cores (18.5×10^6 degrees of freedom). The “Setup” time includes the distribution of the degrees of freedom, setting up of any sparsity patterns necessary, as well as the setup of the data structures required for the matrix-free GMG transfer. Here, our GMG method requires roughly 2x the work for distributing the degrees of freedom, but does not need to build any sparsity patterns. This results in roughly equivalent setup times between AMG and GMG, with GMG being slightly faster. In theory, we should easily be able to lower the requirement of 2x the work in degree of freedom distribution. The “Assemble” timing includes all matrix assembly (system matrix, preconditioner matrix, AMG setup) as well as assembling the right hand side of the linear system and vectors/tables related to the matrix-free operators. Here is where we see the largest advantage for the GMG method as it has no matrices to assemble, resulting in more than a 10x faster assembly. Combining setup and assembly with the linear solve, we have that the GMG method is around 3x faster for this problem.

Procs	DoFs	AMG	GMG
48	18M	53	27
96	36M	56	27
192	72M	62	28
384	141M	62	28
768	278M	68	28
1536	551M	75	28
3072	1.1B	80	28
6144	2.2B	83	28

TABLE 4. FGMRES iterations required to reduce the residual by 10^6 on problems depicted in Figure 3(a).

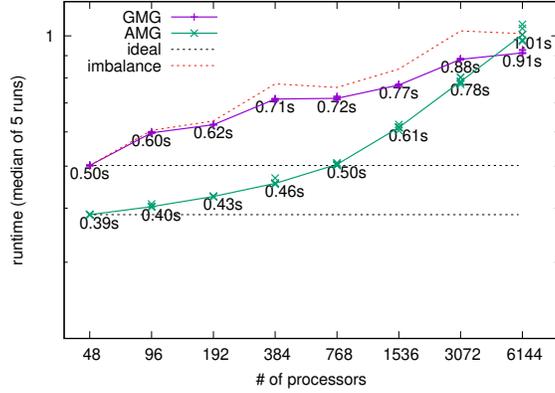
However, for time dependent applications, many time steps will typically be solved without further refining the mesh, in which case, we no longer need to call the “Setup” functionality. Then the program time will be dominated by assembly and solve, in which case GMG will be about 4x faster here.

Expanding on this, we look at the weak scaling of each component up to 6,144 cores and mesh size of 2.2×10^9 degrees of freedom. Starting with the linear solve, as with the scaling plots for global refinement above, Figure 3(a) gives the timing for the preconditioner application, and Table 4 gives the number of FGMRES iteration required in the solve. Here we see that, while the AMG preconditioner is cheaper to apply for all but the last data point, the iteration counts for GMG are much lower and stay constant while the AMG iteration counts increase by over 50%. Figure 3(b) shows the solve time and Figure 3(c) shows the speedup. The red dashed line in this plot represents the ideal weak scaling (black dashed line) multiplied by a factor representing the imbalance of the parallel mesh partition that occurs on computations using adaptive mesh refinement, explain in the work of Clevenger et al..[8] This factor represents the increase in runtime one can expect with the current mesh partition (as opposed to a fair distribution), and is both dependent on the mesh refinement scheme and the number of cores used. Here we see that, even with the imbalance of the partition, the scaling for GMG is more efficient than AMG, and there is near perfect efficiency when taking into account the imbalance of the mesh partition, which, according to the work of Clevenger et al.,[8] should remain bounded.

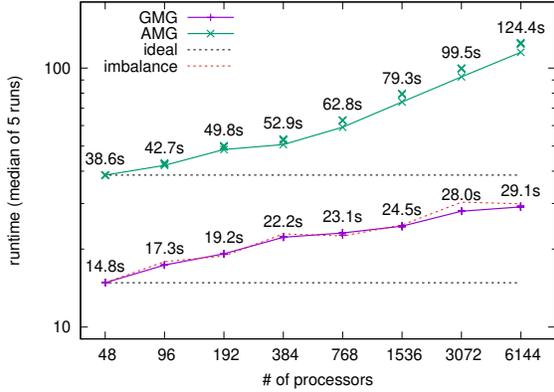
The weak scaling of the setup is shown in Figure 4(a), and again we see that AMG and GMG are roughly equivalent, with GMG being slightly faster. The setup of the linear system should be optimized in the future since we are roughly on the same order of magnitude as the solve time, and efforts should be made to improve the scaling. The weak scaling of the assembly is shown in Figure 4(b). Unsurprisingly, the GMG assembly (consisting mostly of viscosity evaluation) is much cheaper than AMG as there are no matrices to assemble.

3.4. Memory Consumption. The storage requirements for FGMRES, required by the preconditioner in ASPECT due to the non-constant nature of the CG solve for the mass matrix (discussed in Section 2.2.2), is quite restrictive as one will need to store two additional vectors for each Krylov subspace iteration up to the restart length. With a restart length of 50, as is the default inside ASPECT, one will have to allocate the storage of up to 101 vectors inside the Krylov solve alone. This can greatly affect the overall storage of the GMG-based method, as seen in Table 5.

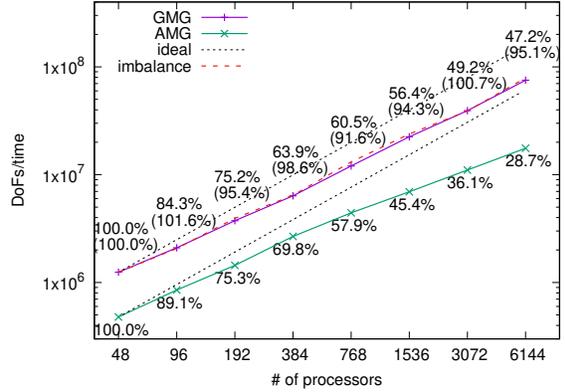
Table 5 gives an estimation of the memory consumptions (in MB) of the largest global objects required by each method on a globally refined mesh with 113K degrees of freedom run on a single core. While, for the AMG-based method the temporary vectors make up around 35% of the total storage, these vectors represent around 95% of the storage required by the GMG-based method. Therefore, we only see around 2.7x less memory used for the GMG-based method, even though there are no matrices that need to be stored.



(a) One application of the Stokes block preconditioner.



(b) FGMRES solve.



(c) FGMRES solver speedup (DoFs/time). Labels represent weak scaling efficiency, parentheses is efficiency to the partition imbalance.

FIGURE 3. Weak scaling comparison, adaptive refinement, 3D mesh, $[Q_2]^{\text{dim}} \times Q_1$ element. Timings are from the Stampede2 machine.

The first step to remove this requirement for temporary vectors is to find a preconditioner which does not require a flexible Krylov method. This can easily be done by considering a further approximation of the Schur complement S by taking $\hat{S} = \text{diag}(M_p)$, as is done in the work of May et al.[20] Another option is to approximate S as one multigrid v-cycle of the mass matrix M_p , which we will do for the remainder of the results. As with the CG solve of the mass matrix (described in Section 2.2.2), the application of the v-cycle will not be computationally significant as compared to the application of \hat{A}^{-1} . With this new Schur complement approximation, one can use GMRES instead of FGMRES, which will reduce the number of temporary vectors by half. Then, for the test in Table 5, we now have a total of 94.1MB of storage for vectors out of a total of 104.4MB for the GMG-based method (90% of the total), and we see around 4.9x less memory as opposed to the current AMG-based method in ASPECT.

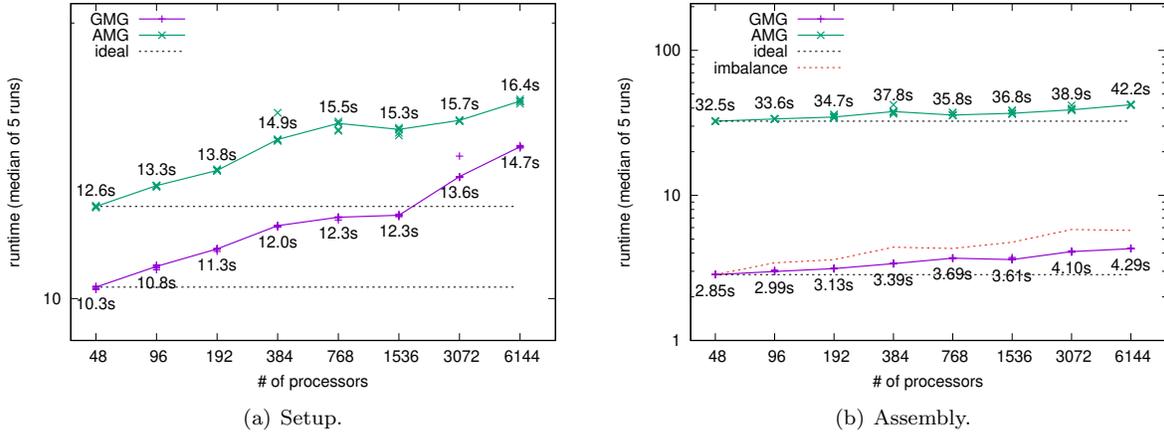


FIGURE 4. Setup/Assembly of the linear system. Timings are from the Stampede2 machine.

Memory (MB)	AMG	GMG
Triangulation	1.9	1.9
DoFHandlers	2.8	5.7
Constraints	1.0	2.7
A	174.2	-
B and B^T	31.4	-
\hat{A}	58.5	-
\hat{S}	1.4	-
Vectors(105)	179.6	179.6
AMG matrices	59.8	-
Total	510.6	189.9

TABLE 5. Memory consumption required for major components of AMG- and GMG-based methods for globally refined, 3D mesh, with 113K DoFs ($[\mathbb{Q}_2]^{\text{dim}} \times \mathbb{Q}_1$ element) on 1 core. The total number of vectors required is set at 105. This estimates takes into account the temporary vectors in the FGMRES solve as well as others required throughout the code (right-hand side vector, solution vector, and two previous timesteps are common).

While this is a step in the right direction, we would like a method whose storage was constant with respect to the number of iterations. We will seek further improvements using the IDR(s) method from the work of Gijzen and Sonneveld.[25] These Krylov methods have a short-term recurrence, requiring the storage of $5 + 3s$ vectors for the solve, and there are $s + 1$ matrix-vector products and preconditioner applications per iteration. Assuming the correct choice of parameters, IDR(1) is equivalent to BiCGStab, and, as s is increased, the convergence rate has been demonstrated to approach the convergence rate of GMRES.[25] The implementation in `deal.II` is based on the preconditioned IDR(s) from the work of Gijzen and Sonneveld[12]. We consider IDR(2) for the computations in this section.

Table 6 gives a comparison between the strong scaling and storage requirements of GMRES and IDR(2) for the Sinkers benchmark run on a globally refined mesh with 27×10^9 unknowns. Since the preconditioner

Cores	GMRES				IDR(2)			
	iters	solve time	# of P^{-1}	vectors	iters	solve time	# of P^{-1}	Vectors
14,336	30	40.5	30	31	11	42.1	33	11
28,672	30	23.1	30	31	11	21.8	33	11
57,344	30	13.0	30	31	13	13.5	39	11
114,688	30	7.74	30	31	13	7.90	39	11

TABLE 6. Iterations required to reduce the residual by 10^6 , time to solution (in seconds), number of preconditioner applications, and vectors stored for a solve of the Sinkers benchmark defined in Section 3.2 (4 sinkers and a viscosity ratio of 10^4), on a globally refined mesh with 27×10^9 DoFs. Timings are from the Frontera machine.

Memory (MB)	FGMRES(50)		GMRES(50)	IDR(2)
	AMG	GMG	GMG	GMG
Triangulation	1.9	1.9	1.9	1.9
DoFHandlers	2.8	5.7	5.7	5.7
Constraints	1.0	2.7	2.7	2.7
A	174.2	-	-	-
B and B^T	31.4	-	-	-
\hat{A}	58.5	-	-	-
\hat{S}	1.4	-	-	-
Vectors	179.6	179.6	94.1	25.7
AMG matrices	59.8	-	-	-
Total	510.6	189.9	104.4	36.0

TABLE 7. Memory consumption required for major components of AMG- and GMG-based methods for globally refined, 3D mesh, with 113K DoFs ($[\mathbb{Q}_2]^{\text{dim}} \times \mathbb{Q}_1$ element) on 1 core. FGMRES(50) (105 total vectors) estimates are given for both AMG and GMG, and both GMRES(50) (55 total vectors) and IDR(2) (15 total vectors) estimates are given for GMG.

application dominates the overall runtime of the solve, it is no surprise that the IDR(2) method is slightly more expensive. However, the times are comparable and both methods see similar scaling. The IDR(2) method, on the other hand, requires roughly 1/3 of the total vector storage.

Returning then to the test in Table 5, consider Table 7 which gives the storage requirement for all FGMRES(50), GMRES(50), and IDR(2). We see that, for the GMG-based method, switching from GMRES to FGMRES leads to a 1.8x memory increase, and switching from IDR(2) to FGMRES leads to a 5.3x memory increase. If we compare the increase in memory from IDR(2) with the GMG-based method to FGMRES with the AMG-based method, we can expect a roughly 14x memory increase. This implies that the capabilities of solving large scale problems inside ASPECT would be drastically increased by using the GMG-based Stokes solve with IDR(2), allowing for runs with an order of magnitude more unknowns.

3.5. Large Scale Computations. Figure 5(a) gives the strong scaling for an application of the Stokes preconditioner from a globally refined mesh with between 6-11 refinement levels, and Table 8 gives the corresponding IDR(2) iteration counts to reduce the residual by 10^6 . The dashed scaling lines here are all computed based on the data point at 56 cores/ 6.7×10^6 DoFs, and therefore the plot represents both strong and weak scaling. We see scaling to around 30-60K DoFs/core for the preconditioner and roughly constant

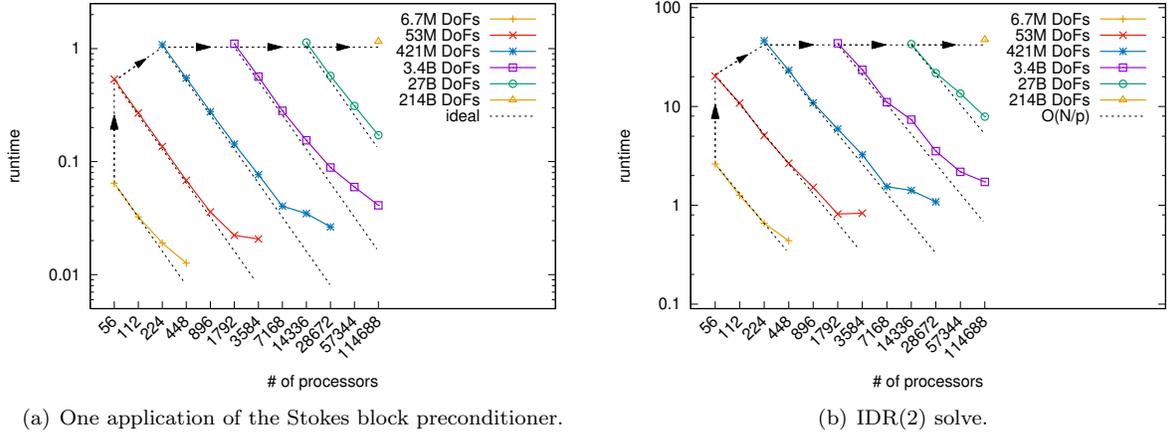


FIGURE 5. Strong and weak scaling for GMG-based method for globally refined, 3D mesh with $[\mathbb{Q}_2]^{\text{dim}} \times \mathbb{Q}_1$ element. Timings are from the Frontera machine.

Procs\DoFs	6.7M	53M	412M	3.4B	27B	217B
56	12	11				
112	12	12				
224	11	11	13			
448	11	12	13			
896		13	12			
1792		11	12	12		
3584		11	12	12		
7168			12	12		
14336			12	14	11	
28672			13	12	11	
57344				12	13	
114688				12	13	12

TABLE 8. IDR(2) iterations required to reduce the residual by 10^6 for the GMG-based preconditioner on problems depicted in Figure 5(a) (Globally refined mesh).

iteration counts. Figure 5(b) gives the timings for the IDR(2) solve, and unsurprisingly, since the iteration counts are almost constant, we see the same scaling.

Lastly, we will compare the performance of the GMG v-cycle on the velocity block (representing approximately 70% of the preconditioner runtime) to the peak performance possible on the Frontera machine. Using the tool LIKWID,[27] we find that we must perform approximately 1,575 Flops/DoF for a globally refined mesh. If we consider that the v-cycle application on the velocity space for the problem containing 217×10^9 Stokes DoFs (206×10^9 velocity DoFs) takes 0.827s of compute time, we have an arithmetic throughput of 0.40 Petaflops/s. The Intel Xeon Platinum 8280 (Cascade Lake) nodes on Frontera can execute 16 double precision floating point operations per cycle, with a clock speed of 2.7 GHz, and so on 114,688 cores we have a peak performance of 5.0 Petaflops/s. This means that we are operating at 8% of peak performance.

Similar performance for GMG v-cycles using `deal.II` was seen in the work of Arndt et al.,[2] where a GMG preconditioner based on a scalar Laplacian discretized with a $\mathbb{Q}_4^{\text{disc}}$ elements was calculated to be around 13% peak performance. There, they found the computations were memory bound,[2] which is also expected for the computations here.

4. CONCLUSION

In this article we presented a geometric multigrid method for preconditioning of the velocity block in a Stokes solve with variable viscosity. The presented method was designed using a matrix-free framework and has capabilities to be run on adaptively refined meshes and in parallel. The parallel scalability of this method was shown for both strong and weak scaling, for both global and adaptive refinement, with up to 114,688 cores and up to 217 billion degrees of freedom. We performed a comparison of the developed method with the AMG-based preconditioner currently used in the `ASPECT` code and found that the GMG-based preconditioner was both more robust (lower iteration counts and constant convergence with mesh refinement) and exhibits better weak scaling than the AMG method, resulting in at least 3x faster computations than the AMG-based method. The GMG-based method was also estimated to require 14x less memory. All-in-all, the GMG-based Stokes solver in this paper (and implemented in `ASPECT`) has allowed for the solution of problems 2 orders of magnitude larger than the largest problem ever solved by the AMG-based method in `ASPECT` (these largest runs are given by the 2.2 billion degrees of freedom run in Section 3.3). Finally, the GMG-based method, using `IDR(2)` as a solver, was shown to scaling to around 30K DoFs/core, and the GMG v-cycle for the velocity block, which represents around 70% of the total preconditioner application, was shown to have a throughput of 8% of peak performance on 114,688 cores.

ACKNOWLEDGMENTS

The authors were partially supported by NSF Award EAR-1925575, OAC-2015848, DMS-2028346, and by the Computational Infrastructure in Geodynamics initiative (CIG), through the NSF under Award EAR-0949446 and EAR-1550901 and The University of California – Davis.

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number ACI-1548562.[26] The authors acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing access to both the Stampede2 and Frontera machines that have contributed to the research results reported within this paper. Clemson University is acknowledged for generous allotment of compute time on Palmetto cluster. Lastly, we would like to thank Martin Kronbichler of the Technical University of Munich (TUM) for his extensive help in the understanding and implementation of matrix-free algorithms.

REFERENCES

- [1] D. Arndt, W. Bangerth, T. C. Clevenger, D. Davydov, M. Fehling, D. Garcia-Sanchez, G. Harper, T. Heister, L. Heltai, M. Kronbichler, R. M. Kynch, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells. The `deal.II` library, version 9.1. *Journal of Numerical Mathematics*, 2019. accepted.
- [2] D. Arndt, W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells. The `deal.II` finite element library: design, features, and insights, 2019.
- [3] W. Bangerth, J. Dannberg, R. Gassmoeller, T. Heister, et al. `ASPECT` v2.1.0 [software], April 2019.
- [4] S. Bauer, M. Huber, S. Ghelichkhan, M. Mohr, U. Rde, and B. Wohlmuth. Large-scale simulation of mantle convection based on a new matrix-free approach. *Journal of Computational Science*, 31:60 – 76, 2019.
- [5] D. Braess and W. Hackbusch. A New Convergence Proof for the Multigrid Method Including the V-cycle. *SIAM J. Sci. Comput.*, 20(5):967–975, 1983.
- [6] S.C. Brenner and L.R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer, 2nd edition edition, 2002.
- [7] C. Burstedde, L.C. Wilcox, and O. Ghattas. `p4est`: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees. *SIAM J. Sci. Comput.*, 33(3):1103–1133, 2011.

- [8] T.C. Clevenger, T. Heister, G. Kanschat, and M. Kronbichler. A Flexible, Parallel, Adaptive Geometric Multigrid method for FEM. submitted.
- [9] H. Elman, D.J. Silvester, and A.J. Wathen. *Finite Elements and Fast Iterative Solvers : with Applications in Incompressible Fluid Dynamics*. Oxford University Press, 2005.
- [10] A. Ern and J-L. Guermond. *Theory and Practice of Finite Elements*. Springer, 2010.
- [11] T. Geenen, M. ur Rehman, S.P. MacLachlan, G. Segal, C. Vuik, A.P. van den Berg, and W. Spakman. Scalable Robust Solvers for Unstructured FE Geodynamic Modeling Applications: Solving the Stokes Equation for Models with Large Localized Viscosity Contrasts. *Geochemistry, Geophysics, Geosystems*, 10(9).
- [12] M.B. Gijzen and P. Sonneveld. Algorithm 913: An elegant IDR(s) variant that efficiently exploits biorthogonality properties. *ACM Transactions on Mathematical Software (TOMS)*, 38, 11 2011.
- [13] B. Gmeiner, M. Huber, L. John, U. Rude, and B.I. Wohlmuth. A Quantitative Performance Study for Stokes Solvers at the Extreme Scale. *J. Comput. Science*, 17:509–521, 2015.
- [14] T. Heister, J. Dannberg, R. Gassmoeller, and W. Bangerth. High Accuracy Mantle Convection Simulation through Modern Numerical Methods II: Realistic Models and Problems. *Geophysical Journal International*, 210(2):833–851, 05 2017.
- [15] M.A. Heroux, R.A. Bartlett, V.E. Howle, R.J. Hoekstra, J.J. Hu, T.G. Kolda, R.B. Lehoucq, K.R. Long, R.P. Pawlowski, E.T. Phipps, A.G. Salinger, H.K. Thornquist, R.S. Tuminaro, J.M. Willenbring, A. Williams, and K.S. Stanley. An Overview of the Trilinos Project. *ACM Trans. Math. Softw.*, 31(3):397–423, 2005.
- [16] B. Janssen and G. Kanschat. Adaptive Multilevel Methods with Local Smoothing for H^1 - and H^{curl} -Conforming High Order Finite Element Methods. *SIAM J. Sci. Comput.*, 33(4):2095–2114, 2011.
- [17] M. Kronbichler, T. Heister, and W. Bangerth. High Accuracy Mantle Convection Simulation through Modern Numerical Methods. *Geophysical Journal International*, 191(1):12–29.
- [18] M. Kronbichler and K. Kormann. A Generic Interface for Parallel Cell-based Finite Element Operator Application. *Computers & Fluids*, 63:135–147, 2012.
- [19] M. Kronbichler and W.A. Wall. A Performance Comparison of Continuous and Discontinuous Galerkin Methods with Fast Multigrid Solvers. *SIAM J. Scientific Computing*, 40:A3423–A3448, 2018.
- [20] D.A. May, J. Brown, and L. Le Pourhiet. A Scalable, Matrix-free Multigrid Preconditioner for Finite Element Discretizations of Heterogeneous Stokes Flow. *Computer Methods in Applied Mechanics and Engineering*, 290:496 – 523, 2015.
- [21] J. Rudi, A.C.I. Malossi, T. Isaac, G. Stadler, M. Gurnis, P.W.J. Staar, Y. Ineichen, C. Bekas, A. Curioni, and O. Ghattas. An Extreme-scale Implicit Solver for Complex PDEs: Highly Heterogeneous Flow in Earth’s Mantle. In *SC ’15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12, Nov 2015.
- [22] J. Rudi, G. Stadler, and O. Ghattas. Weighted BFBT Preconditioner for Stokes Flow Problems with Highly Heterogeneous Viscosity. *SIAM Journal on Scientific Computing*, 39(5), 2017.
- [23] Y. Saad. A Flexible Inner-outer Preconditioned GMRES Algorithm. *SIAM J. Sci. Comput.*, 14(2):461–469, March 1993.
- [24] D. Silvester and A. Wathen. Fast Iterative Solution of Stabilised Stokes Systems Part II: Using General Block Preconditioners. *SIAM Journal on Numerical Analysis*, 31(5):1352–1367, 1994.
- [25] P. Sonneveld and M.B. Gijzen. IDR(s): A Family of Simple and Fast Algorithms for Solving Large Nonsymmetric Systems of Linear Equations. *SIAM J. Scientific Computing*, 31:1035–1062, 01 2008.
- [26] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G.D. Peterson, R. Roskies, J. Scott, and N. Wilkins-Diehr. XSEDE: Accelerating Scientific Discovery. *Computing in Science & Engineering*, 16(05):62–74, sep 2014.
- [27] J. Treibig, G. Hager, and G. Wellein. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *Proceedings of PSTI2010, the First International Workshop on Parallel Software Tools and Tool Infrastructures*, San Diego CA, 2010.
- [28] U. Trottenberg, C.W. Oosterlee, and Anton Schuller. *Multigrid*. Academic Press, 2001.