Parallel Multiscale Linear Solver for Highly Detailed Reservoir Models

A. M. Manea, Stanford University; **J. Sewall,** Intel Corporation; and **H. A. Tchelepi,** Stanford University

Summary

To realize the potential of the latest high-performance computing (HPC) architectures for reservoir simulation, scalable linear solvers are necessary. We describe a parallel algebraic multiscale solver (AMS) for the pressure equation of heterogeneous reservoir models. AMS is a two-level algorithm that uses domain decomposition with a localization assumption. In AMS, basis functions, which are local (subdomain) solutions computed during the setup phase, are used to construct the coarse-scale system and grid-transfer operators between the fine and coarse levels. The solution phase is composed of two stages: global and local. The global stage involves solving the coarse-scale system and interpolating the solution to the fine grid. The local stage involves application of a smoother on the fine-scale approximation.

The design and implementation of a scalable AMS on multicore and many-core architectures, including the decomposition, memory allocation, data flow, and compute kernels, are described in detail. These adaptations are necessary to obtain good scalability on state-of-the-art HPC systems. The specific methods and parameters, such as the coarsening ratio (C_r), basis-function solver, and relaxation scheme, have significant effects on the asymptotic convergence rate and parallel computational efficiency.

The balance between convergence rate and parallel efficiency as a function of C_r and the local stage parameters is analyzed in detail. The performance of AMS is demonstrated using heterogeneous 3D reservoir models, including geostatistically generated fields and models derived from SPE10 (Christie and Blunt 2001). The problems range in size from several million to 128 million cells. AMS shows excellent behavior for handling fixed-size problems as a function of the number of cores (so-called strong scaling). Specifically, for a 128-million-cell problem, a ninefold speedup is obtained on a single-node 12-core shared-memory architecture (dual-socket multicore Intel Xeon E5-2620-v2), and more than 12-fold on a single-node 20-core shared-memory architecture (dual-socket multicore Intel Xeon E5-2690-v2). These are encouraging results given the limited memory bandwidth that cores can share within a single node, which tends to be the major bottleneck for truly scalable solvers. We also compare the robustness and performance of our method with the parallel system algebraic mutligrid (SAMG) solver (Stüben 2012) from Fraunhofer SCAI.

Introduction

In most reservoir simulation models, reservoir formation properties—mainly permeability and porosity—have the largest influence on the subsurface fluid-flow behavior. Those properties typically vary locally by many orders of magnitude and possess complex multiscale spatial correlation structures. Therefore, highly detailed geological models, in the range of $O(10^8)$ to $O(10^9)$ grid cells, are often needed to resolve such large multiscale heterogeneities. Although the accuracy of the simulation results relies substantially on the detailed geological description of the underlying reservoir, the computational cost of simulating such high-resolution models is prohibitively high for existing reservoir simulators. Thus, the resolution of the original geological model is often reduced systematically by use of upscaling techniques (Durlofsky 2005) to allow for reasonable simulation times. The simulation of upscaled models generally yields good average results; however, they cannot resolve the fine-scale details of the subsurface fluid flow, which can be of great importance in several situations.

Multiscale methods (Hou and Wu 1997; Efendiev et al. 2000; Aarnes and Hou 2002; Arbogast and Bryant 2002; Chen and Hou 2003; Jenny et al. 2003; Aarnes 2004; Aarnes et al. 2005; Efendiev and Hou 2009) were motivated by the need to reduce the computational costs of simulating large reservoir models without abandoning their fine-scale details. In their general framework, multiscale methods are based on the idea of nonoverlapping domain decomposition, where the global fine-scale computational domain is decomposed into subdomains, on each of which the original problem is solved independently using boundary conditions derived by a localization assumption. These local solutions, called basis functions, are then used to construct a coarse-scale global problem that is significantly less expensive to solve. In addition, the basis functions are also used to reconstruct the fine-scale details from the solution to the coarse-scale problem.

Several multiscale approaches have been developed in the literature, which can be generally divided into three main categories: the multiscale finite element (MSFE) methods (Hou and Wu 1997; Efendiev and Hou 2009); the multiscale mixed finite-element (MSMFE) methods (Arbogast 2002; Arbogast and Bryant 2002; Chen and Hou 2003; Aarnes 2004; Aarnes et al. 2005), and the multiscale finite-volume (MSFV) methods (Jenny et al. 2003). These methods vary mainly in the way they construct the global coarse-scale system from the local basis functions. This, in turn, distinguishes their behavior with respect to mass conservation, which is a crucial property if mass transport (i.e., evolution of the saturation/concentration distribution) is to be addressed together with the flow. In particular, the MSMFE and MSFV methods have a systematic way of constructing a conservative velocity field for mass transport (Wang et al. 2014).

Although the original (single-pass) multiscale methods generally provide fine-scale solutions that are in good agreement with reference solutions for a wide range of heterogeneous problems, the accuracy of some multiscale methods can drop substantially for some challenging problems, such as fields with extreme permeability contrasts, or long-scale coherent structures of high permeability (channels) or low permeability (shales) (Lunati and Jenny 2007). This degradation of the accuracy of the method is directly related to the localization assumption used to decouple the local problems used to construct the basis functions. To overcome this limitation, the multiscale method was extended as an iterative solver (Hajibeygi et al. 2008; Hajibeygi and Jenny 2011) capable of converging to the fine-scale reference solution. Algebraically, the iterative multiscale method was formulated as a two-stage solver, called two-stage algebraic multiscale solver (TAMS) (Zhou and Tchelepi 2012), and later as AMS (Wang et al. 2014). In TAMS and AMS, the original single-pass multiscale solver is applied in the first stage (global stage), and then a local preconditioner is applied in a second stage (local stage). As the names imply, spectral analysis (Hajibeygi et al. 2008; Zhou and Tchelepi 2012) shows that the global stage acts on damping low-frequency error modes associated with long-range coupling of the variables, and leaves the high-frequency error modes associated with short-range coupling of the variables to be handled at the local stage. The global stage of the solver can be any variant

This paper (SPE 173259) was accepted for presentation at the SPE Reservoir Simulation Symposium, Houston, 23–25 February 2015, and revised for publication. Original manuscript received for review 17 March 2015. Revised manuscript received for review 2 December 2015. Paper peer approved 22 February 2016.

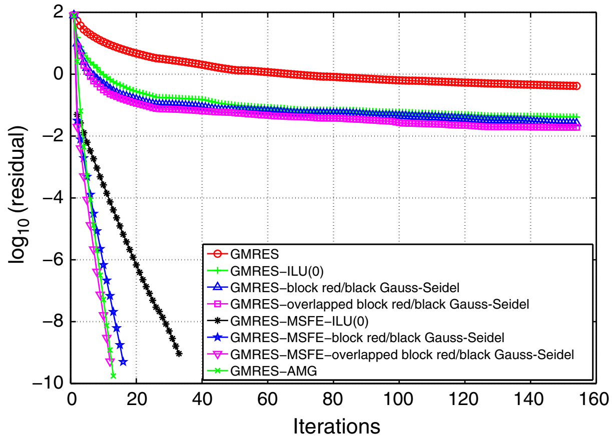

Fig. 1—Convergence histories when solving Eq. 1 for a 1.2-million-cell problem derived from the SPE10 (Christie and Blunt 2001) top layer by piecewise-constant grid refinement, with two sides as fixed-pressure boundary conditions and the other two sides as no-flow boundary conditions, by use of (a) GMRES alone; (b) GMRES preconditioned by simple single-level preconditioners: ILU(0) and block red/black Gauss Seidel smoothing, with and without overlapping blocks; (c) GMRES preconditioned by robust multilevel preconditioners: MSFE with ILU(0), MSFE with block red/black Gauss Seidel smoothing, with and without overlapping blocks, and AMG.

of the original multiscale methods. It has been shown, however (Zhou and Tchelepi 2012; Wang et al. 2014), that the most efficient global-stage choice, in terms of both numerical and computational performance, is the MSFE method. In addition, the iterative solver can still be concluded with an MSFV step in the last iteration to yield mass-conservative results.

In addition to its specific application to reservoir simulation, AMS has been shown to be a computationally efficient elliptic solver. In fact, the MSFE-based AMS solver was shown to have a performance comparable to that of the state-of-the-art algebraic multigrid (AMG) solver (Stüben 2012) for multimillion-cell elliptic problems in a serial computational environment (Zhou and Tchelepi 2012; Wang et al. 2014). Moreover, the AMS method has, by construction, a large inherent degree of parallelism (Jenny et al. 2003), which, if exploited carefully, can result in highly scalable performance on emerging massively parallel architectures.

Our work focuses on the linear solver that is the core of the Newton-Krylov solution method (Cao et al. 2005) commonly used in reservoir simulators; the system that must be solved for each Newton iteration has a strong near-elliptic component (i.e., pressure) that is poorly conditioned because of the heterogeneity, and this requires specialized methods to achieve acceptable convergence rates.

The constrained-pressure-residual formulation (Wallis 1983; Wallis et al. 1985; Cao et al. 2005) is frequently used to address the particular spectra contributed by the pressure equations, thereby accelerating the convergence of the Krylov subspace method (Saad 2003) at the core of the linear solver.

The development of preconditioners is challenging; we seek to speed up a solver by taking fewer iterations, but must spend time setting up the preconditioner and then repeatedly applying it in the solver. Preconditioning is thus a balance between "strength" (fewer iterations) and "cost" (time it takes to set up and apply the preconditioner). These two qualities are usually in direct opposition. Furthermore, in parallel computing environments, the Basic Linear Algebra Subprograms (BLAS) Levels 1 and 2 operations that constitute the core of the Krylov subspace method (Saad 2003) are known to scale. To be ultimately beneficial in real-world applications, preconditioners must have scalable performance as well.

However, the large elliptic problems that arise in reservoir simulations are strongly dependent on communication and memory bandwidth for performance: A Krylov step (Saad 2003) requires that the entire system (unknowns, matrix, right-hand side, plus several residual and auxiliary vectors) be read. The ratio of floating-point operations to bytes of main memory transferred, also known as arithmetic intensity (Asanovic et al. 2009), of these kernels is low, and the problems of interest never fit in cache, nor can they be blocked across steps. The most promising method of mitigating this particular problem is to simply reduce the number of Krylov steps through efficient preconditioning.

Multigrid methods have long been the method of choice for preconditioning elliptic problems, and AMG techniques suitable for reservoir-simulation problems have been developed and demonstrated to be very effective at accelerating convergences with modest overhead (Stüben et al. 2003, 2007; Cao et al. 2005; Clees and Ganzer 2007; Klie et al. 2007).

In this paper, we demonstrate that the multiscale solver is a scalable and efficient preconditioner for heterogeneous elliptic problems. In particular, we focus on the TAMS formulation (Zhou and Tchelepi 2012; Wang et al. 2014), and demonstrate superior performance and scalability on up to 12 cores on a shared-memory system. We further demonstrate the scalability of the parallel AMS on 20 cores of a recent Intel Xeon-based system.

The contribution of this work is twofold. First, to the best of our knowledge, there is no work in the literature that formally analyzes and experimentally demonstrates the performance of the AMS algorithm on parallel platforms. Second, this work serves to highlight the potential of AMS as a highly scalable elliptic linear solver that is well-suited for massively parallel architectures. The rest of this paper is organized as follows. First, some basic background information regarding the general AMS algorithm is highlighted, followed by a detailed analysis of the main computational kernels of AMS. A basic model for analyzing the performance of the algorithm is given next. Following that, the results of several numerical experiments highlighting the robustness and scalability of our parallel AMS implementation on different multicore and many-core architectures are presented. The paper closes with concluding remarks and possible extensions.

Background

In reservoir simulation, a considerable portion of the simulation time is spent solving a variant of the following elliptic partial-differential equation (PDE), which governs the pressure behavior of an incompressible single-phase fluid in a porous medium on some computational domain, Ω:

$$\nabla \cdot (\boldsymbol{\lambda} \cdot \nabla p) = q, \quad \text{(1)}$$

where p is the pressure; $\boldsymbol{\lambda}$ is the positive-definite mobility tensor, which is defined by the ratio of the porous-medium permeability to the fluid viscosity; and q is a source or sink term.

Eq. 1 is a Poisson-like equation; however, it involves highly discontinuous coefficients (i.e., $\boldsymbol{\lambda}$), the values of which can vary by several orders of magnitude. Thus, the system resulting from discretizing this equation is often badly conditioned, so it cannot be solved efficiently using common preconditioned Krylov subspace methods (Saad 2003), such as incomplete-lower-upper (ILU) (0)-preconditioned generalized minimal residual (GMRES) method **(Fig. 1).** Rather, such a system requires more-complex preconditioners that take into account the severe discontinuities in the underlying PDE coefficients.

Assume that Eq. 1 is discretized on a fine-grid, Ω^f, using the standard two-point flux approximation, resulting in the following algebraic system of equations:

$$A^f p^f = q^f, \quad \text{(2)}$$

where p^f refers to the discretized-pressure unknowns on the given fine-grid computational domain, Ω^f.

To solve this system, the multiscale method proceeds by subdividing the total fine-grid domain, Ω^f, into subdomains, Ω_k^c, called primal-coarse gridblocks, where k ranges from unity to the number of primal-coarse gridblocks, N_c. Another coarse grid, called the dual-coarse grid, Ω_m^d, is introduced by joining the centers of the primal-coarse gridblocks, Ω_k^c, where m ranges from

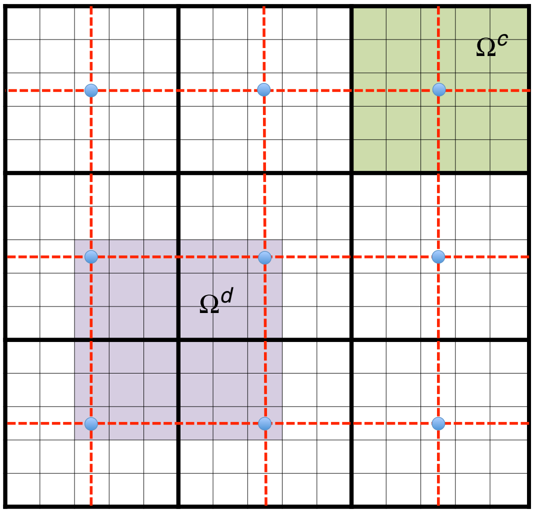

Fig. 2—MS grids in two dimensions. The solid thick black lines denote the primal coarse grid, Ω^c, whereas the dashed red lines denote the dual coarse grid, Ω^d. The blue solid circles represent the coarse points, called vertices. Finally, the solid thin lines denote the original fine grid, Ω^f.

unity to the number of dual-coarse gridblocks, N_d, as shown in **Fig. 2.** The ratio of the number of fine-scale gridblocks, N_f, to the number of coarse-scale primal-coarse gridblocks, N_c, is denoted as the coarsening ratio, C_r:

$$C_r = \frac{N_f}{N_c}. \quad (3)$$

Each dual-coarse gridblock, Ω_m^d, defines a local subdomain on which Eq. 1 is solved locally to obtain the basis functions using reduced boundary conditions. These are determined by a localization assumption, in which fluxes normal to the dual-grid boundary are ignored. This yields an $(n-1)D$ problem on each face/edge of the dual-coarse gridblock, where n is the number of dimensions of the original problem. Algebraically, the basis functions are obtained by solving (Wang et al. 2014) the following:

$$\begin{aligned} &\nabla \cdot (\boldsymbol{\lambda} \cdot \nabla \phi_j^i) = 0, \ \in \Omega_j^d \\ &\nabla_{||} \cdot (\boldsymbol{\lambda} \cdot \nabla \phi_j^i)_{||} = 0, \ \in \partial\Omega_j^d \\ &\phi_j^i(x_k) = \delta_{ik}, \ \forall x_k \in \{1, ..., N_c\} \end{aligned} , \quad (4)$$

where ϕ_j^i is the basis function associated with the coarse-node i in the dual-coarse gridblock Ω_j^d, and the subscript $||$ denotes the projection of the vector or operator along the tangential direction of the dual-coarse-gridblock boundary, $\partial\Omega_j^d$. Note that such a projection ignores the effect of normal fluxes between dual-coarse gridblocks, which is the main source of error in multiscale methods.

The basis functions are used to obtain an approximate fine-scale pressure solution, $\tilde{p}^f$, from the coarse-scale pressure solution, p^c, using the principle of superposition:

$$\tilde{p}^f = \sum_{j=1}^{N_d} \sum_{i=1}^{N_c} \phi_j^i p_i^c = \mathcal{P} p^c, \quad (5)$$

where $\mathcal{P}$ is the $N_f \times N_c$ prolongation operator constructed from the basis functions, and $\tilde{p}^f$ is the MS fine-scale solution. Another operator is needed to map the solution from the fine space to the coarse space, or the $N_c \times N_f$ restriction, $\mathcal{R}$, operator.

Multiscale methods vary in the way they define $\mathcal{R}$, which distinguishes their respective $N_c \times N_c$ coarse-scale operators, A^c. In MSFE, which is the main focus of this paper, the Galerkin definition (Smith et al. 1998; Toselli and Widlund 2005; Tchelepi and Wallis 2010) is used for $\mathcal{R}$:

$$\mathcal{R} = \mathcal{P}^{\mathcal{T}}. \quad (6)$$

The global coarse-scale system can be written as

$$\mathcal{R} A^f \mathcal{P} p^c = \mathcal{R} q^f, \quad (7)$$

where the $N_c \times N_c$ coarse-scale operator, A^c, is defined as

$$A^c = \mathcal{R} A^f \mathcal{P}. \quad (8)$$

The main computational advantage of multiscale formulations is that the coarse-scale system is much smaller; thus, it is less expensive to solve than the original fine-scale system. The single-pass MSFE solution is expressed as

$$\tilde{p}^f = \mathcal{P}(A^c)^{-1} q^c = \mathcal{P}(\mathcal{R} A^f \mathcal{P})^{-1} \mathcal{R} q^f, \quad (9)$$

from which we can write the global-stage MSFE preconditioning operator, M_g^{-1}, as

$$M_g^{-1} = \mathcal{P}(\mathcal{R} A^f \mathcal{P})^{-1} \mathcal{R}. \quad (10)$$

The steps involved in constructing the single-pass MSFE solution, $\tilde{p}^f$, can be described by linear operations on the fine-scale linear operator, A^f, as follows. First, the operator is permuted into a "wirebasket" ordering (Smith et al. 1998; Tchelepi and Wallis 2010), in which the dual-coarse grid is used to subdivide the underlying fine-grid cells into three classes: interior cells, p_I; edge cells, p_E; and vertex cells, p_V. For 3D problems, cells falling on the faces of the dual domain can be combined with the edge cells into one class. Thus, we have the following permuted linear system:

$$\begin{bmatrix} A_{II} & A_{IE} & 0 \\ A_{EI} & A_{EE} & A_{EV} \\ 0 & A_{VE} & A_{VV} \end{bmatrix} \begin{bmatrix} p_I \\ p_E \\ p_V \end{bmatrix} = \begin{bmatrix} q_I \\ q_E \\ q_V \end{bmatrix}. \quad (11)$$

The reduced-boundary-condition assumption ignores the influence of the interior cells on the edge cells, A_{EI}; the influence of edge cells on vertex cells, A_{VE}; and the right-hand-side terms of the interior and edge cells, q_I and q_E (Zhou and Tchelepi 2012). In addition, we replace A_{VV}, which is defined on the centers of the primal-coarse gridblocks, with our MSFE coarse-operator, A^c. We also replace the right-hand side term q_V, with $q^c = \mathcal{R} q^f$ and the unknowns p_V with p^c. This results in the following system:

$$\begin{bmatrix} A_{II} & A_{IE} & 0 \\ 0 & \tilde{A}_{EE} & A_{EV} \\ 0 & 0 & A^c \end{bmatrix} \begin{bmatrix} p_I \\ p_E \\ p^c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ q^c \end{bmatrix}. \quad (12)$$

This system is block upper-triangular with the global coarse-scale system appearing in the last block. Note that $\tilde{A}_{EE}$ is derived from A_{EE} by adding the contribution of A_{EI} into A_{EE}. Given the solution of the coarse system, p^c, the p_E and p_I are found by backward substitution as follows:

$$p_E = -\tilde{A}_{EE}^{-1} A_{EV} p^c, \quad (13)$$

$$p_I = -A_{II}^{-1} A_{IE} p_E = A_{II}^{-1} A_{IE} \tilde{A}_{EE}^{-1} A_{EV} p^c. \quad (14)$$

The prolongation operator, $\mathcal{P}$, can be defined as

$$p^f = W \begin{bmatrix} p_I \\ p_E \\ p^c \end{bmatrix} = W \begin{bmatrix} A_{II}^{-1} A_{IE} \tilde{A}_{EE}^{-1} A_{EV} \\ -\tilde{A}_{EE}^{-1} A_{EV} \\ I_{VV} \end{bmatrix} p^c = \mathcal{P} p^c, \quad (15)$$

where W is the "wirebasket" permutation matrix and I_{VV} is the identity matrix of size $N_c \times N_c$. It can be seen that the computation of A_{II}^{-1} and $\tilde{A}_{EE}^{-1}$, which amounts to finding the basis function, is purely local and therefore naturally amenable to parallelism.

Zhou and Tchelepi (2012) showed that the global MSFE preconditioner, M_g^{-1}, is rank-deficient by $N_f - N_c$ and does not yield a convergent scheme if used alone (e.g., in a Richardson iteration). The main deficiency in M_g^{-1} is in its inability to eliminate high-

frequency error modes; so, it must be accompanied by a local preconditioner, M_l^{-1}, that complements the global nature of M_g^{-1} by reducing high-frequency error modes in the residual. The overall preconditioning scheme, called TAMS, was proposed by Zhou and Tchelepi (2012) and has been further studied and generalized by Wang et al. (2014). The TAMS operator can be written as

$$M_{TAMS}^{-1} = M_g^{-1} + M_l^{-1} - M_l^{-1} A^f M_g^{-1}. \quad \ldots\ldots\ldots\ldots \quad (16)$$

The choice of the local-stage preconditioner, M_l^{-1}, is a design choice that has a strong effect on the robustness and scalability of the method, which we explore in this paper.

Summary of AMS Steps. AMS involves two main phases: setup and solution. The setup phase involves computing the basis functions, assembling the prolongation/restriction operators, constructing and factoring the coarse-scale system, and setting up the local-stage preconditioner. The solution phase involves restricting the fine-scale right-hand side, solving the coarse-scale system, interpolating the solution back to the fine space, and applying the local-stage preconditioner.

To summarize, the basic MSFE-based AMS preconditioner has the following steps.

Step 1: Setup Phase.

- Select the MSFE grids (primal- and dual-coarse grids).
- Compute the basis functions, ϕ_j^i, by solving the system defined by Eq. 4.
- Assemble the prolongation operator, $\mathcal{P}$, and restriction operator, $\mathcal{R}$, from the basis functions, ϕ_j^i.
- Construct the coarse-scale system: $A^c = \mathcal{R}A^f\mathcal{P}$.
- Factor the coarse-scale system: $A^c = L^c U^c$.
- Set up the local-stage preconditioner, M_l^{-1}.

Step 2: Solution Phase.

- Global stage:
  - Restrict the fine-scale right-hand side: $q^c = \mathcal{R}q^f$.
  - Solve the coarse-scale system: $p^c = (A^c)^{-1}q^c$.
  - Prolong the coarse solution to the fine space: $p^f = \mathcal{P}p^c$.
- Local stage:
  - Apply the local-stage preconditioner, M_l^{-1}.

AMS Computational Kernels

The overall robustness and parallel scalability of AMS depend on the specific choices of the involved kernels. In this section, we take a closer look at these kernels with special focus on the two most critical ones: the basis functions (setup phase) and the local-stage preconditioner (solution phase).

Setup Phase. The way in which AMS grids are selected has a substantial effect on both the robustness and scalability of the overall algorithm. In particular, the value of the coarsening-ratio parameter, C_r, defined in Eq. 3, controls the balance between the global and local stages.

The support of each basis function, assuming uniform coarsening in all cardinal directions, is equal to $(C_r^{1/3}+1)^3 \approx C_r$. Assuming that we compute the basis functions using a direct methods, which is the most-efficient choice (as we establish later), we have an asymptotic computational cost of $O(C_r^3)$ per basis-function linear system. The total number of basis-function linear systems to be factored is equal to $N_d \approx N_c$. Thus, the total cost of computing the basis functions can be estimated as

$$\text{Total basis function cost} \approx N_c \times C_r^3 = \frac{N_f}{C_r} \times C_r^3 = N_f \times C_r^2. \quad \ldots\ldots\ldots\ldots\ldots\ldots (17)$$

As C_r increases (i.e., coarsening becomes more aggressive), the total cost of computing the basis functions increases quadratically. This is expected because the support of the basis functions increases with C_r.

The parallel scalability of the basis-functions-computation kernel is also influenced by the value of C_r. Because the number of basis-function linear systems $\approx N_c = N_f/C_r$, it is clear that as C_r decreases, the number of basis-function linear systems and the amount of independent work increase.

Although both the computational complexity and the scalability of the basis-functions-computation kernel favor smaller values of C_r, very-small C_r values result in very-large coarse-scale systems that are too expensive to factor. As $C_r \to N_f$ (i.e., when there is only one large coarse cell), the total cost becomes N_f^3, which is equivalent to factoring the fine-scale system.

On the other hand, the size of the coarse-scale system, N_c, is also determined by C_r because $N_c = N_f/C_r$. If we only consider the coarse-scale-system size, we would prefer to have large C_r to reduce the cost of applying M_g^{-1}. However, increasing C_r exacerbates the rank deficiency of the solver and leaves a wider spectrum of error modes intact. The convergence rate of the encapsulating scheme (Richardson or Krylov) will suffer in such a case unless a very-robust—and more computationally expensive—technique is used for M_l^{-1} at the local stage.

In addition to the selection of the AMS grids, the basis-functions-computation kernel makes up the bulk of the setup phase, even though all the basis functions can be computed independently. The concurrent computation of the basis functions can involve substantial memory-bandwidth usage, and care must be taken to scale properly, particularly on shared-memory architectures.

We are free to choose the order in which the basis functions are computed because they are all independent of each other. We can formulate the basis subproblems in terms of their complete (compact) regions of support, or further decompose each one into quadrants/octants by dual blocks, as shown in **Fig. 3.**

Although the first approach is more natural and needs no extra work in mapping the basis functions to construct the prolongation operator, it is inefficient from a memory/cache usage point of view: Its active working set per thread/core is much larger (4X in two dimensions and 8X in three dimensions) than the second approach, and there is less reuse of the system coefficients.

In the dual-domain-centered approach, we have 4 or 8 systems to solve for each dual domain; the relatively small size of the dual domains makes direct factorization attractive. Note that a single factorization can be reused for all incident basis functions. Moreover, the poor conditioning of the local elliptic systems is challenging to Krylov subspace methods (Saad 2003) without further preconditioning; the benchmark results in **Fig. 4** provide an analysis for different solvers.

Because there are many independent operators to solve (one for each dual domain; i.e., a total of N_d systems), we should not be concerned with extracting large amounts of parallelism from any single LU factorization. Moreover, considerable computation savings can also be made by reusing the fill-in-reducing permutation among dual-domain systems with a similar nonzero pattern, which is the case for all interior dual domains.

Construction of the Coarse-Scale System. The coarse-scale operator, A^c, is given by Eq. 8; the triple product of sparse matrices with dimensions $N_c \times N_f$, $N_f \times N_f$, and $N_f \times N_c$ represents the relationship between each pair of coarse nodes. Although the matrix-product formulation is convenient for illustrative and validation purposes, the resulting operator, A^c, is itself a structured sparse matrix, and it is highly wasteful to construct using standard sparse BLAS operations, particularly in light of Eq. 6. An intermediate product (either $\mathcal{R}A^f$ or $A^f\mathcal{P}$) must be computed and stored, and it is difficult to take advantage of the potential storage reduction suggested by the prolongation-restriction relationship and still be computationally efficient.

We can exploit the structure in Eq. 8 by noting that each column of $\mathcal{P}$ (and each row of $\mathcal{R}$) corresponds to a coarse degree of freedom with the rows (columns of $\mathcal{R}$) representing the influence that the coarse point has on each fine degree of freedom. The compact support of the global basis functions associated with each coarse degree of freedom dictates the sparsity pattern of the operator. A^f is our discretized differential operator, a seven-point

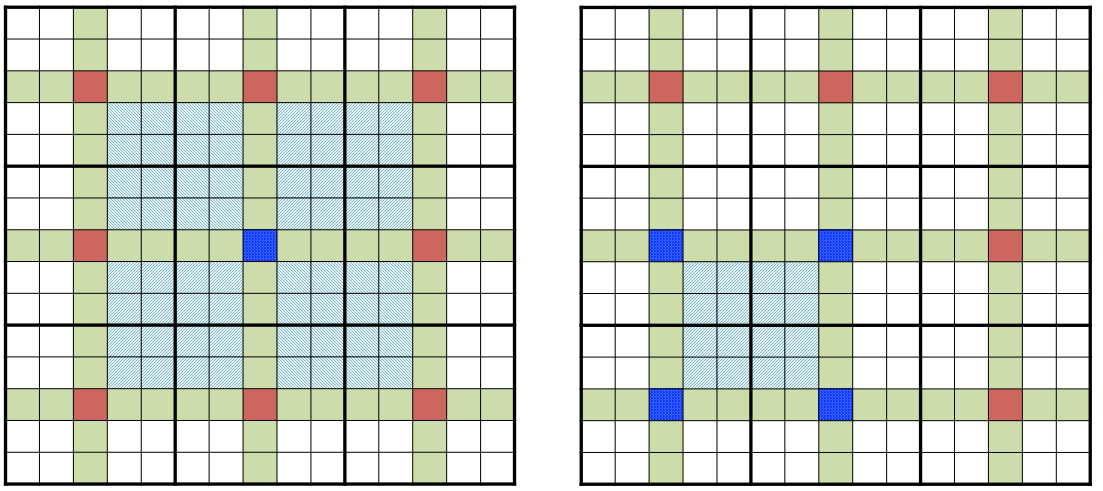

Fig. 3—Computing the basis function in two dimensions on a global support (left) and on a quadrant of the support per dual-coarse gridblock (right).

stencil with spatially varying coefficients, and admits an efficient banded-matrix representation.

A geometric interpretation of the construction shows that the subterm $A^f\mathcal{P}$ alters the nonzero pattern of $\mathcal{P}$ by growing the region of support for each coarse degree of freedom's basis function by one fine grid cell in each cardinal direction, up to the domain boundaries. Thus, we can construct A^c by iterating over each coarse degree of freedom and computing the product of its basis function with each "fattened" basis (i.e., from portions of $A^f\mathcal{P}$) with which it has an intersection. An illustration of these steps is shown in **Fig. 5.**

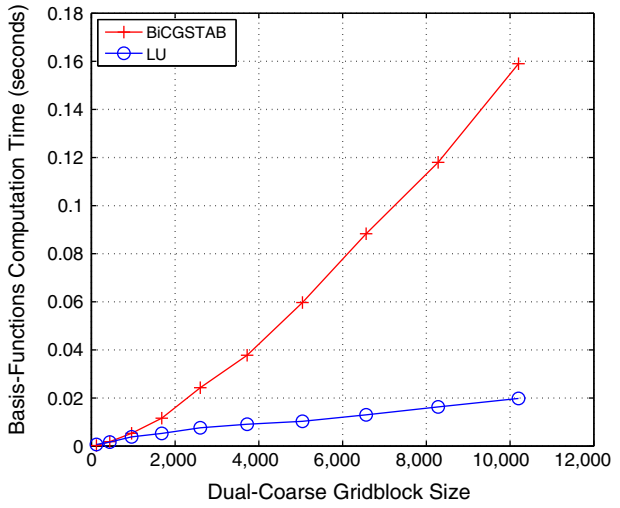

Fig. 4—A benchmark comparing the time it takes a direct solver (LU) (Intel 2013), against an iterative solver (BiCGSTAB) (Guennebaud and Jacob 2010), configured with a tight tolerance of 10^{-8} (absolute), for various 2D dual-coarse-gridblock sizes, from $11^2 \approx 100$ cells to $101^2 \approx 10{,}000$ cells.

Although the description of the AMS algorithm lists the assembly of the prolongation operator, $\mathcal{P}$, and restriction operator, $\mathcal{R}$, as a step of the setup phase, it is inefficient to explicitly assemble $\mathcal{P}$ and $\mathcal{R}$ from the basis functions as they are computed. With the aforementioned "matrix-free" algorithm to construct the coarse system, these operators need not be explicitly computed; the dense individual basis functions can be used to restrict the right-hand sides and prolong coarse solutions.

Finally, the cost of factoring the coarse-scale system is directly dependent on the value of the coarsening ratio, C_r. However, unless the value of C_r is extremely small, the cost of factoring the coarse-scale system has negligible contribution to the total cost of the setup phase.

Setup of the Local-Stage Preconditioner, M_l^{-1}. The local-stage preconditioner, M_l^{-1}, is a critical part of AMS because its robustness and scalability can be greatly influenced by the robustness and scalability of M_l^{-1}.

Main Requirements for M_l^{-1}. There are two critical constraints that must be honored when choosing an algorithm for M_l^{-1}. First, M_l^{-1} should focus on resolving error components missed by the global-stage preconditioner, M_g^{-1}. Otherwise, M_l^{-1} would be performing a wasteful computation by repeating effort carried out by

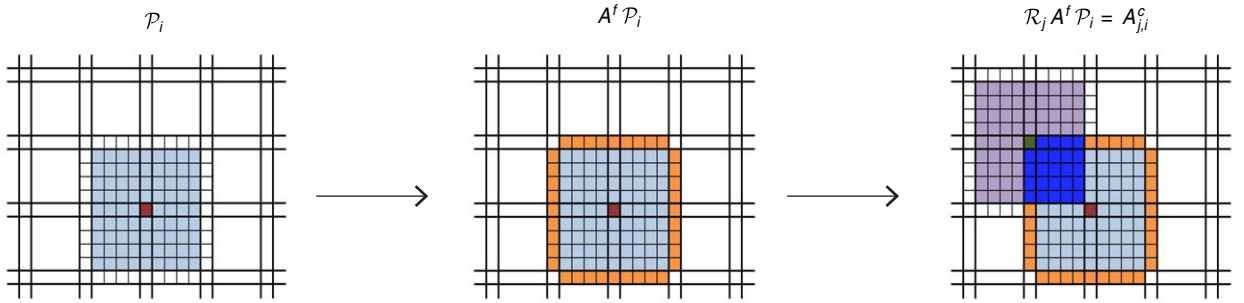

Fig. 5—A geometric illustration of the steps involved in computing the entry (j, i) of the coarse-scale operator, A^c, denoted by $A^c_{j,i}$, in 2D settings. $\mathcal{P}_i$ denotes the ith column of the prolongation operator, $\mathcal{P}$, whereas $A^f\mathcal{P}_i$ denotes the same column after left multiplication by the fine-scale operator, A^f. $\mathcal{R}_j$ denotes the jth row of the restriction operator, $\mathcal{R}$. The shaded areas represent the support of each column/row (light-blue for $\mathcal{P}_i$, with the location of unity in red; similar color-coding for $A^f\mathcal{P}_i$, with additional fill-in in orange; and purple for $\mathcal{R}_j$, with the location of unity in dark green). The dark-blue shaded area in the rightmost subfigure represents locations where the supports of $A^f\mathcal{P}_i$ and $\mathcal{R}_j$ overlap, which results in $A^c_{j,i}$ being structurally nonzero.

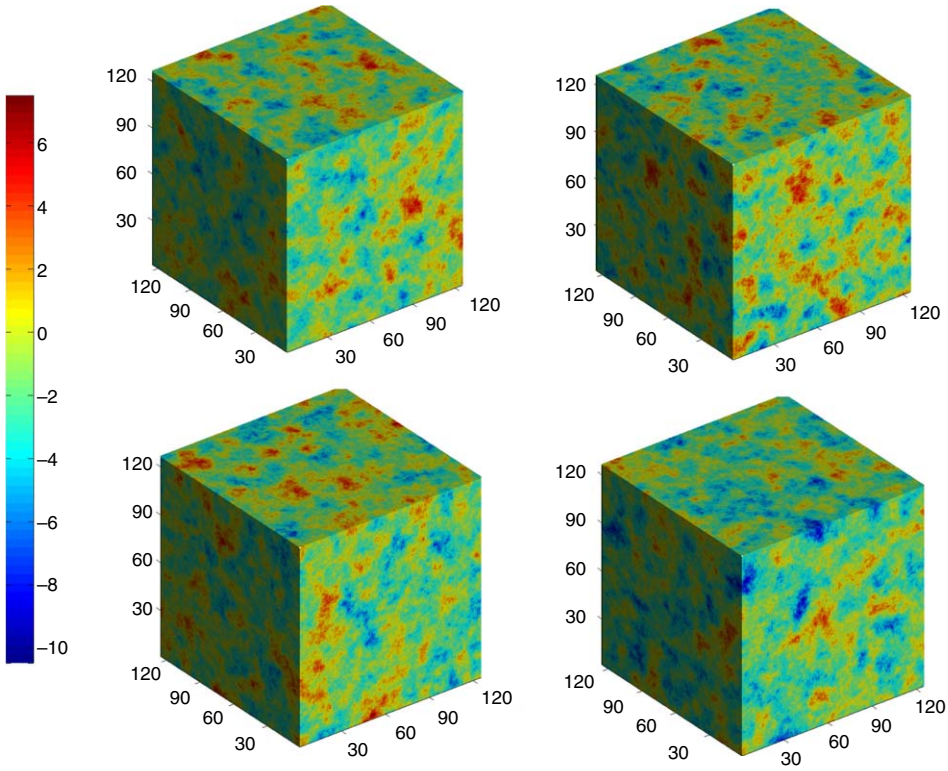

Fig. 6—Natural logarithm of four realizations of the first set (126^3) of permeability fields.

M_g^{-1}. Second, M_l^{-1} must have scalable setup and solution kernels to aid overall AMS scalability.

Many robust algorithms for M_l^{-1} (Zhou and Tchelepi 2012; Wang et al. 2014) have an expensive setup cost and are not parallelizable. These are poor candidates for building a scalable AMS. Examples include standard ILU factorization and block ILU factorization. On the other hand, there are other robust and parallelizable algorithms for M_l^{-1}, such as block-smoothing and additive Shwarz techniques (Toselli and Widlund 2005). Simple algorithms like point smoothers do not have any setup cost, but this comes at the expense of convergence. The choice between the latter two classes of algorithms for M_l^{-1} is a design decision that is very closely related to the value of the coarsening ratio, C_r, and the architecture on which AMS is to be executed. A performance comparison between a block-smoother (block Jacobi) and a point smoother (damped red/black Gauss-Seidel) is presented in Appendix A.

Solution Phase. The solution phase consists of two stages: global and local. The global stage involves three basic kernels. First, the restriction of the right-hand side to the coarse scale is a sparse matrix-vector multiplication (SpMV) kernel. Second, the solution of the coarse-scale system is a standard forward/backward-solution kernel. Finally, the prolongation of the coarse-scale solution to the fine scale is another SpMV kernel. These basic kernels have minimal effect on the total solution-phase cost unless the value of C_r is extremely small.

The local stage, where we apply M_l^{-1} to the current fine-scale-solution estimate, is the most important part in the solution phase of AMS. It dominates the total cost of the solution phase for practical values of C_r [e.g., $O(10)$ in each cardinal direction]. For this reason, we seek a local-stage preconditioner with a scalable solution kernel.

Performance Characteristics

In this section, we analyze the performance and scalability of key portions of the AMS solver. A more-detailed storage-complexity analysis is given in Appendix B.

For the sake of simplicity, we restrict our analysis to cubic domains. The primary parameters of the problem are the fine-scale-grid dimensions $N_{f_x} = N_{f_y} = N_{f_z} = N_f^{1/3}$, and the coarsening ratio, C_r. With Eq. 3, we have the coarse-system dimensions $N_{c_x} = N_{c_y} = N_{c_z} = N_c^{1/3}$.

There are N_f degrees of freedom in the fine system and N_c degrees of freedom in the coarse system; note that the number of actual degrees of freedom in the system is generally reduced by some constant factor $\ll N_f$, depending on the boundary conditions.

The fine-scale operator, A^f, is an $N_f \times N_f$ septadiagonal banded matrix, with each off-diagonal element representing the coefficient of transmissability between two adjacent fine cells. A^f is structurally symmetric, but does not generally exhibit a useful value-based symmetry.

The fine system thus has N_f double-precision quantities to store the unknown vector p^f, N_f doubles for the right-hand side q^f, and $N_f \times 7$ doubles to represent A^f in a banded form. Thus, the total fine-scale base working set is: $9 \times N_f \times 8 = 72\ N_f$ bytes, assuming an 8-byte word is used for representing double-precision variables.

The coarse-scale system, A^c, is a banded $N_c \times N_c$ matrix with 27 bands; it is generated by the overlapping-basis-function technique described previously and factored into $L^c U^c$ during the setup stage. We can conservatively assume the occupancy of the factorization to be a filled-in band for each row. With a matrix bandwidth of $2(N_c^{2/3} + N_c^{1/3} + 1)$, the $L^c U^c$ factorization has $2N_c(N_c^{2/3} + N_c^{1/3} + 1) = O(N_c^{5/3})$ nonzeros. The key two steps constraining the performance during the construction of the coarse-scale system are the basis-functions computation and the Galerkin product, $A^c = \mathcal{R}A^f\mathcal{P}$.

Each coarse point supports a $(2C_r^{1/3} + 1)^3 = 8C_r + 12C_r^{2/3} + 6C_r^{1/3} + 1 = O(C_r)$-cell basis function centered at the coarse point, which we represent as eight $(C_r^{1/3} + 1)^3$-cell partially overlapping subbases, one for each dual domain incident on the point. These are computed by extracting the basis-function operator for each dual domain Ω_k^d from A^f, factoring it, and using the factored operator to solve (by means of forward/backward substitution) the subsystems that arise from each of the eight basis functions incidents on the Ω_k^d.

Each dual domain corresponds to a $(C_r^{1/3} + 1)^3 \times (C_r^{1/3} + 1)^3$ septadiagonal banded independent suboperator of A^f; this is

Parameter		Value
C_r	All cases	9×9×9
M_l^{-1}	All cases	Damped point red/black Gauss-Seidel
n_s	(Test Case 1)	16
	(Test Cases 2, 3)	8
ω	(Test Case 1)	1.96
	(Test Cases 2, 3)	1.8

Table 1—Parameter settings for AMS.

factored into LU with $(C_r^{1/3}+1)^3 \times 2 \times (C_r^{1/3}+1)^2 = 2(C_r^{1/3}+1)^5$ nonzeros (following the same conservative estimates outlined previously).

A key factor in the design of the AMS algorithm is that there are $(N_c^{1/3}+1)^3$ dual domains with completely independent small systems to factor/backsolve; for the $C_r^{1/3}=9$ we have used in this paper, we are left with no more than $2\cdot(9+1)^5 = 200,000$ nonzeros in each suboperator, which can be efficiently handled by the hierarchical memory systems of modern central processing units. Even the smallest problem size we have considered in this work ($N_f^{1/3}=126$) yields large numbers of independent systems ($N_c^{1/3}=126/9=14$; $15^3=3,375$) for ensuring core occupancy.

Galerkin Product, $\mathcal{R}A^f\mathcal{P}$. The basis functions are used to prolong the coarse-scale solution to the fine scale and to restrict the fine-scale right-hand side of the system to the coarse scale, and to construct the coarse-scale system.

The computation of the (implicit) $A^f\mathcal{P}$ terms is simply a stencil operation applied to each nonzero of the basis functions [i.e., $(2\times C_r^{1/3}+1)^3$ elements for each of the N_c coarse points]; once the product $A^f\mathcal{P}$ for a coarse point has been computed, we iterate over the 27-point neighborhood of adjacent coarse points. The corresponding reductions that fill A^c contain $2^{(3-o)}\times(C_r^{1/3}+1)^3$ multiplication and addition operations each, where o is the number of indices (i.e., i, j, k) that differ from the centerpoint of the basis function located at the corresponding coarse point.

As in the basis-function computation, this kernel allows reuse of small intermediate regions of computation to leverage the memory hierarchy and allow high core occupancy.

Numerical Experiments

In this section, the robustness and scalability of AMS are experimentally demonstrated for various multicore shared-memory platforms using different test cases that vary in size and heterogeneity. In addition, we compare the performance of AMS with that of algebraic multigrid (AMG)—using the Fraunhofer state-of-the-art SAMG software (Stüben 2012)—on the same platform. Our experiments show that AMS is competitive with AMG in robustness and scalability, and they highlight the potential of AMS on emerging many-core architectures.

This section is organized as follows. First, the design of the test cases is described. Then, the specific parameter settings used for testing AMS and AMG are highlighted. The performance of AMS and AMG are then presented for a variety of test cases. Finally, the scalability of AMS is demonstrated on a large multicore shared-memory architecture.

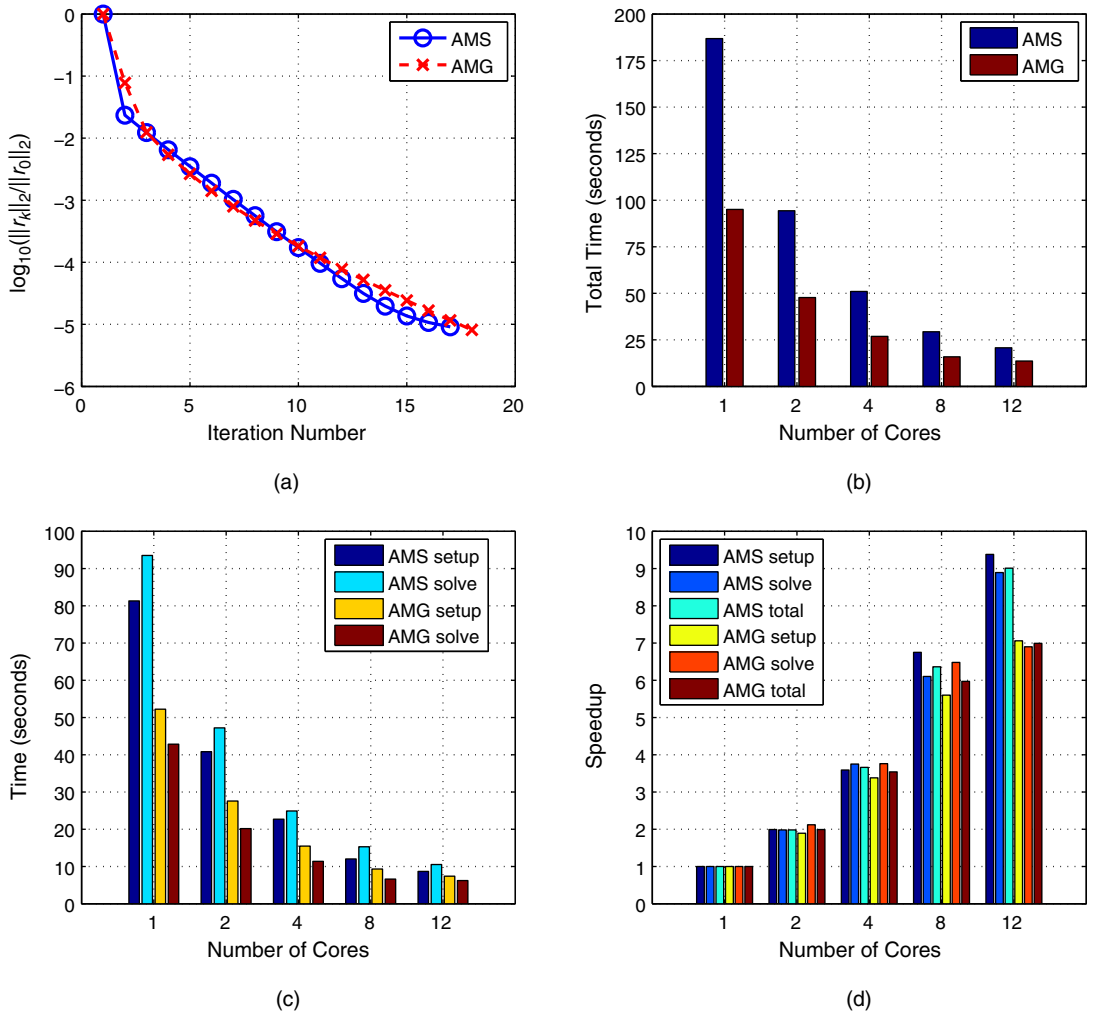

Fig. 7—Convergence rate and scalability of AMS and AMG for the model derived from SPE10 (Christie and Blunt 2001).

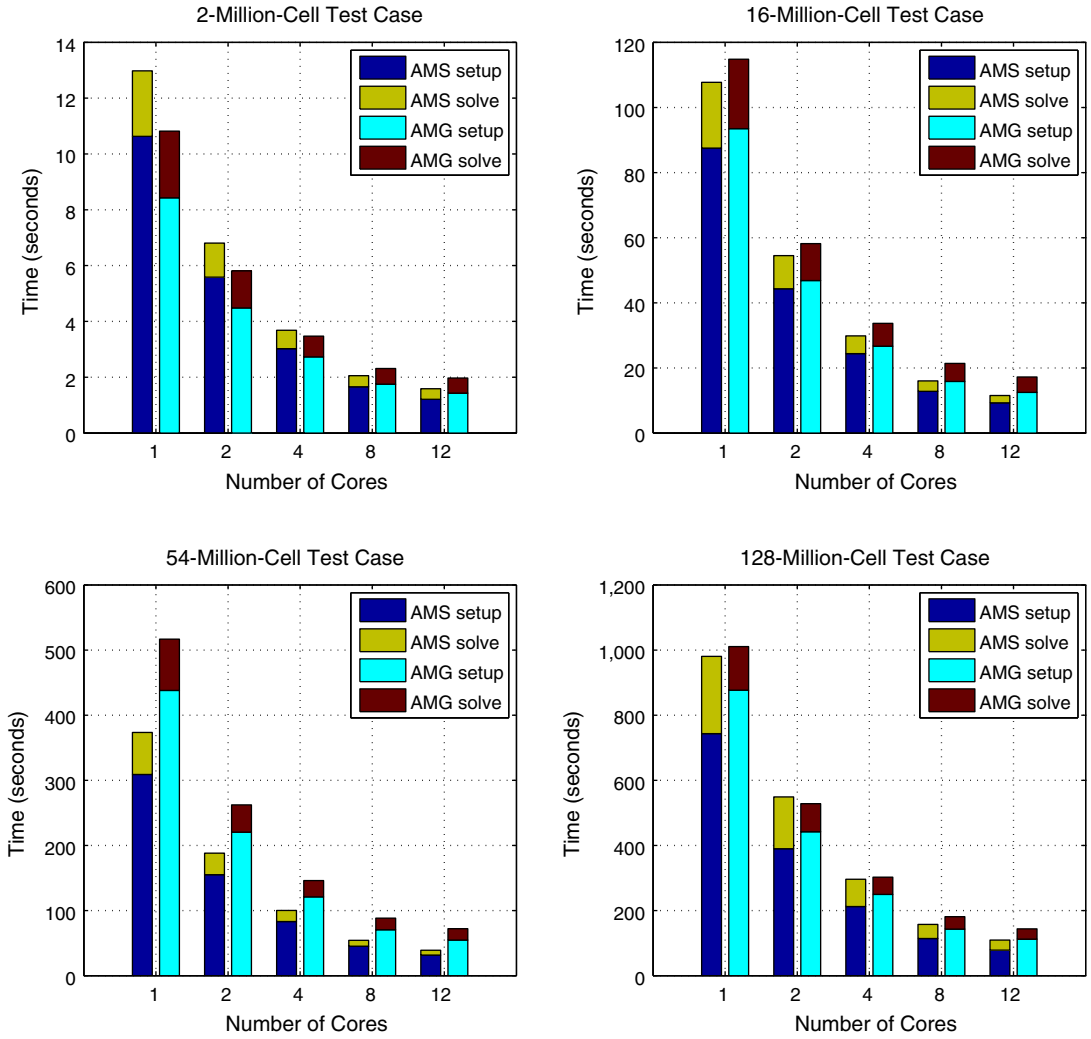

Fig. 8—Performance of AMS and AMG for the second test case.

Note: Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured by use of specific computer systems, components, software, operations, and functions. Any change to any of these factors may cause the results to vary. The authors encourage readers to consult other information and performance tests for assistance in fully evaluating any contemplated purchases, including the performance of that product when combined with other products.

Test-Cases Design. Three test cases are used. The first test case is derived from the SPE10 problem (Christie and Blunt 2001) to examine the robustness of AMS and compare it with AMG. The second test case uses an ensemble of geostatistically generated permeability fields and is designed to compare the average performance of AMS and AMG in terms of convergence-rate and scalability. The third test case is identical to the second, but is focused only on the parallel performance of AMS. The third test case has been performed on a machine with a large number of cores.

Test Case 1. The first test case uses a permeability field derived from the SPE10 problem (Christie and Blunt 2001), which is modified in two ways: First, the original size of the SPE10 problem (1.1 million cells) is increased to 16 million by tiling the entire permeability field in each direction to reach the desired size, which is 252^3, or approximately 16 million cells. This avoids homogenizing the problem, which happens with piecewise-constant grid refinement. Second, the permeability field is made isotropic by using the permeability values in the x-direction to populate the permeabilities in the y- and the z-direction.

Test Cases 2 and 3. The second and third test cases use four sets of permeability fields that are log-normally distributed with spherical variorgrams. The fields were generated using sequential Gaussian simulation (Remy et al. 2009). These sets vary in size, starting with 126^3 or 2 million cells for the smallest test case, and then extending the dimensions of the problem by steps of 126 for each next size, leading to the following sizes: 252^3 or approximately 16 million cells, 378^3 or approximately 54 million cells, and finally 504^3 or approximately 128 million cells. For all cases, the variance of log-permeability is four, which represents an extremely heterogeneous case. The correlation length of the permeability in each direction is one-tenth of the dimension of the domain. Each set has five equiprobable realizations **(Fig. 6);** thus, all the presented results for these cases have been averaged over the five realizations.

Common Settings. The iterative procedure used for all test cases is the simple Richardson scheme to reveal the real convergence behavior of the preconditioner with no stabilization help by Krylov subspace methods (Saad 2003). The boundary conditions for all the problems are set to Dirichlet-type boundary conditions on the left and right faces of the domain, where the pressure is fixed to the dimensionless values of unity and zero, respectively. The other faces are set to Neumann-type no-flow boundary conditions. All the test cases are performed until the l_2 norm of the initial residual is reduced by five orders of magnitude (i.e., $\frac{\|r_k\|_2}{\|r_0\|_2} \leq 10^{-5}$).

Implementation Details. The parallel implementation of AMS is designed to exploit the parallelism of modern symmetric multiprocessing shared-memory architectures using the open multiprocessing application programming interface. The actual

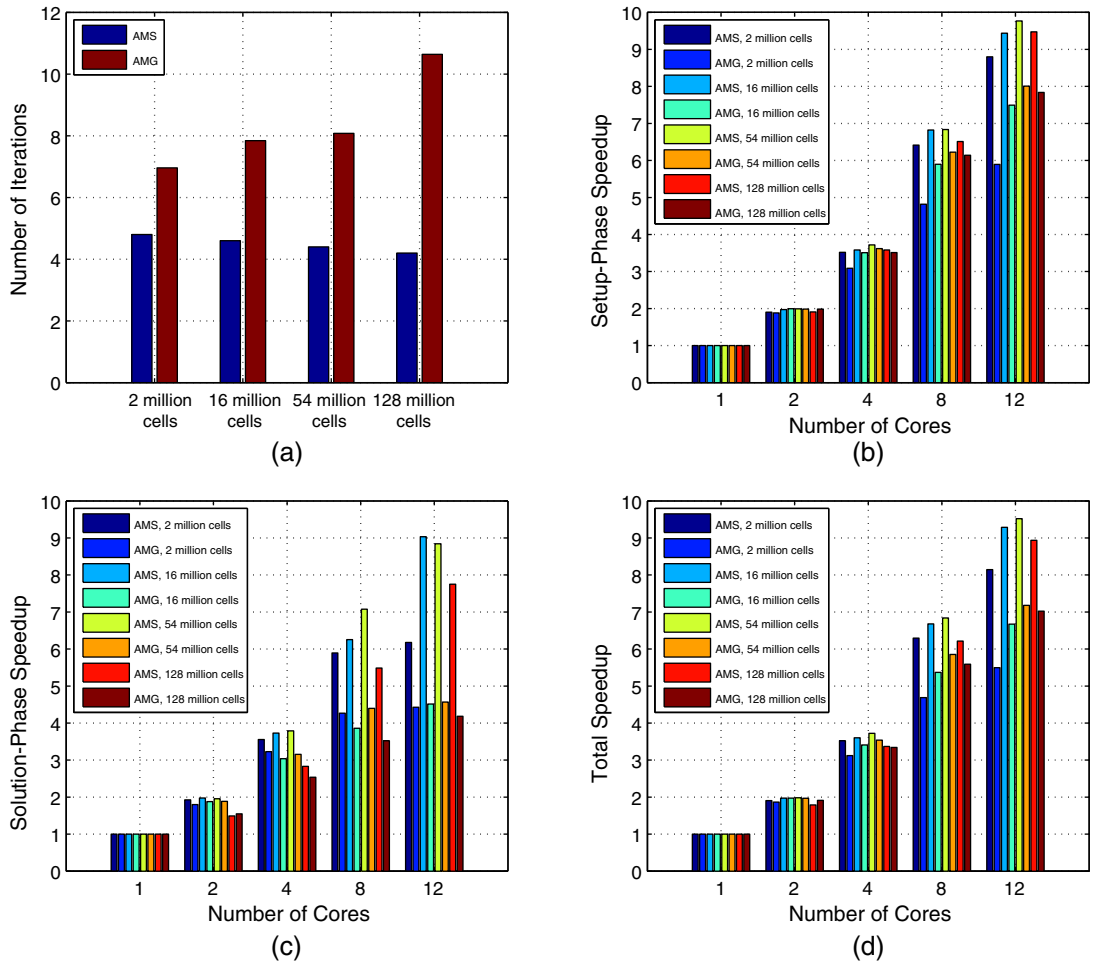

Fig. 9—Convergence rate and scalability of AMS and AMG for the second test case.

implementation is realized using the C++11 programming language.

Computer Platforms. The first two test cases were run on the same platform, a multicore architecture consisting of 12 cores of Intel Xeon E5-2620 v2 at 2.10 GHz. Because the third test case is used to show AMS scaling, we have used a machine with a larger configuration, a dual-socket configuration of Intel Xeon Processor E5-2690 v2, which is a modern multicore server chip formerly codenamed "Ivytown." Each socket features 25 MB of shared L3 cache and 10 cores running at 3GHz, each with 256 KB of L2 cache and 32 KB of L1 data and instruction caches. The cores are capable of two-way simultaneous multithreading and feature superscalar execution and Advance Vector Extensions Single-Instruction-Multiple-Data instruction set.

Parameter Settings. ***AMS Parameter Settings.*** As discussed earlier, the performance of the AMS algorithm depends on several key parameters that must be chosen carefully to have a highly scalable solver. In particular, the most important parameters to consider here are the coarsening ratio, C_r, and the choice of the local-stage preconditioner, M_l^{-1}. A set of experiments were made to test several options for C_r and M_l^{-1} to determine the most-efficient settings for both. On the basis of these experiments, we have chosen C_r as $9\times9\times9$ and M_l^{-1} as the damped point red/black Gauss-Seidel. In addition, several tests were made to come up with the best parameter setting for M_l^{-1}—namely, the number of smoothing steps, n_s, and the damping factor, ω **(Table 1).** The details of the numerical experiments used to obtain the AMS parameter settings are discussed in Appendix A.

AMG Parameter Settings. We used AMG from Fraunhofer, Release No. 27a1 (Stüben 2012). AMG was configured to use a V-cycle with one pre-smoothing step and one post-smoothing step of coarse/fine Gauss-Seidel on each level. A dense solver is used at the coarsest level. Standard coarsening was used for all problem sizes, except the largest problem size of 128 million cells, where we used A2 aggressive coarsening (Stüben 2001) to reduce memory usage.

Test Case 1: AMS Performance Compared with AMG Using the SPE10-Derived Model. Using the AMS parameter settings outlined in Table 1, the 16-million-cell permeability field derived from the SPE10 problem (Christie and Blunt 2001) was used to test both AMS and AMG on the 12-core multicore architecture, as shown in **Fig. 7.** In general, the two solvers have comparable performance in terms of robustness and scalability. In particular, the convergence behavior of both AMS and AMG are very similar, as evident from the convergence history shown in Fig. 7a. This highlights the robustness of AMS as a linear solver for a challenging permeability field derived from the SPE10 problem (Christie and Blunt 2001). On the other hand, although AMG takes almost half the time AMS takes to solve the problem (Fig. 7b), both solvers show good "strong scalability" from one to 12 cores (Fig. 7c). In particular, AMS exhibits an excellent scalability for both the setup and solution phases, achieving a speedup of more than ninefold on 12 cores compared with its serial time on a single core. This excellent scalability of AMS reduces the gap between the two solvers on 12 cores from 1:2 to 2:3, as AMS continues to scale well up to 12 cores.

Test Case 2: AMS Performance Compared with AMG Using the Geostatistically Generated Models. In this test case, the performance of AMS configured with the settings outlined in Table 1 was compared against AMG using the four sets of geostatistically generated models, running on the 12-core multicore architecture. The results, which are averaged over the five realizations of each permeability set, are shown in **Figs. 8 and 9.** In

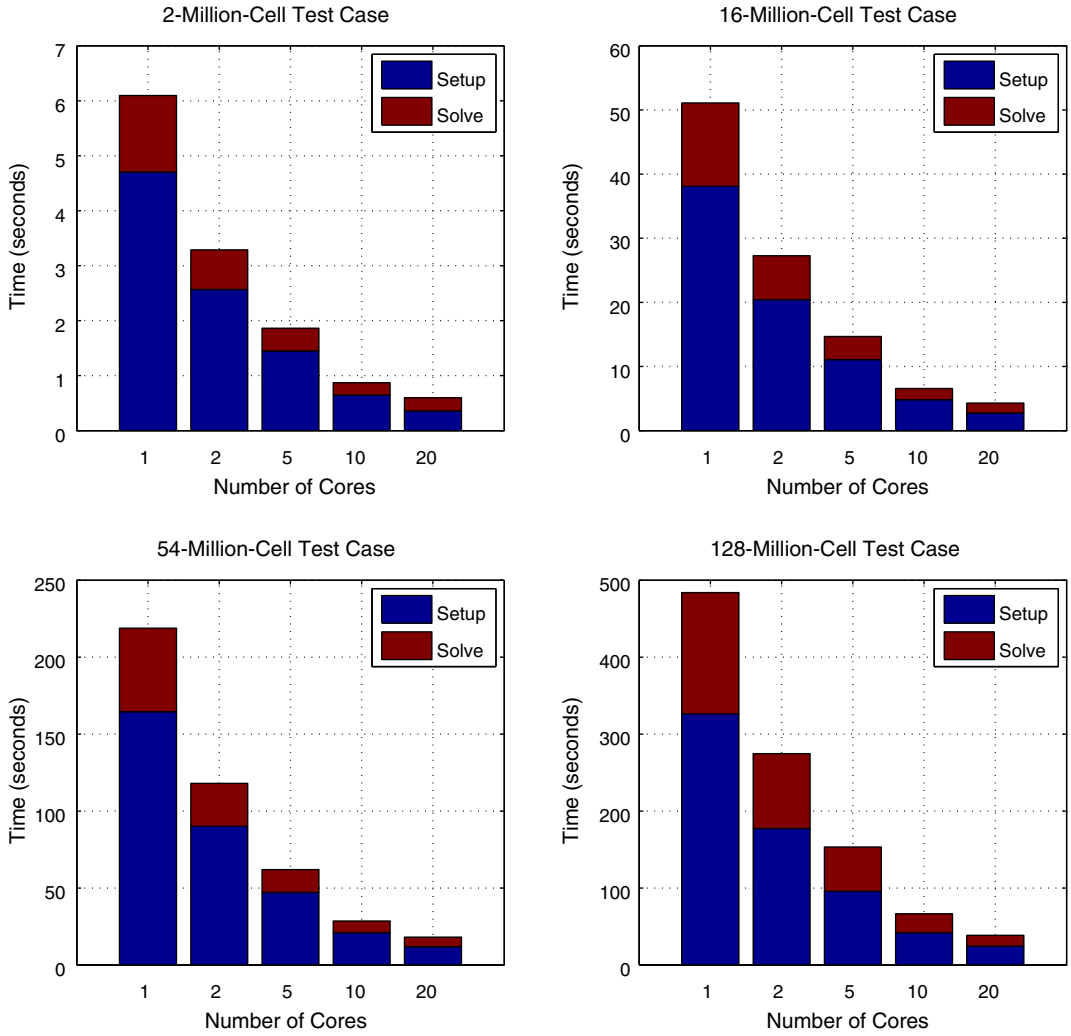

Fig. 10—Performance of setup and solve kernels of AMS for several problem sizes running on 2 × Intel Xeon Processor E5-2690 v2.

general, both solvers show robust convergence behavior and good strong and weak scalability. In particular, the convergence rate of AMS outperforms that of AMG for all the test cases, as shown in Fig. 9a. Moreover, the AMS convergence rate does not deteriorate as the problem size increases, which highlights the good algorithmic scalability of the solver. This is also the case with AMG except for the 128-million-cell problem, in which aggressive coarsening was used to save memory, and that typically affects the convergence behavior of AMG (Stüben 2001). With respect to the total run time, AMS and AMG have similar scalability behavior, with AMS being a little faster for the larger problems, specifically, the problems with 16, 54, and 128 million cells (Fig. 8). The setup phase of both solvers show good scalability up to 12 cores, as shown in Fig. 9b. In particular, the setup phase of AMS achieves a speedup of more than ninefold on 12 cores for all the problem sizes, except the smallest 2-million-cell case, for which the amount of available parallel work, and hence the core occupancy, eventually becomes a limiting factor upon adding more cores. In addition, the solution phase of AMS also has excellent scalability for all the problems except the 2-million-cell case, showing eightfold to ninefold speedups on 12 cores, as seen in Fig. 9c. On the other hand, the AMG solution phase shows good scalability up to eight cores, but then the scalability is limited afterward. The total speedup achieved by AMS for the cases with 16, 54, and 128 million cells on 12 cores is at least ninefold, as seen in Fig. 9d, which is an excellent scalability behavior given the limited hardware resources and amount of parallel work typically encountered in shared-memory strong-scaling experiments.

Test Case 3: The Scalability of AMS on a Large Multicore Architecture. In this test case, the scalability and absolute performance of AMS were tested on a dual-socket configuration of Intel Xeon Processor E5-2690 v2 using the geostatistically generated models. **Fig. 10** shows the performance of AMS broken into setup and solution phases, whereas **Fig. 11** shows the setup, solution, and overall speedup. AMS scales up to 20 cores for all problem size, thanks to scaling on both the setup kernel and the solution kernel. In particular, the setup phase has an excellent scalability, reaching up to a 13-fold speedup on 20 cores for all the problem sizes. On the other hand, the solution phase has good scalability, but not as high as the setup phase, as the memory bandwidth eventually becomes a bottleneck. Nevertheless, for the 128-million-cell case, AMS has a speedup of approximately 12.3-fold on 20 cores (with more than 13-fold for the setup phase and more than 11-fold for the solution phase), in which the 128-million-cell problem is set up and completely solved in less than 41 seconds.

Conclusions

The AMS algorithm involves several key kernels and parameters. These include the local-stage preconditioner, M_l^{-1}, and the coarsening ratio, C_r, which must be carefully designed to achieve high scalability on emerging HPC architectures. We have experimentally demonstrated algorithmic and parameter choices for AMS for which the solver shows good strong and weak scalability on a 20-core shared-memory architecture. Moreover, the solver robustness and scalability were compared against the state-of-the-art

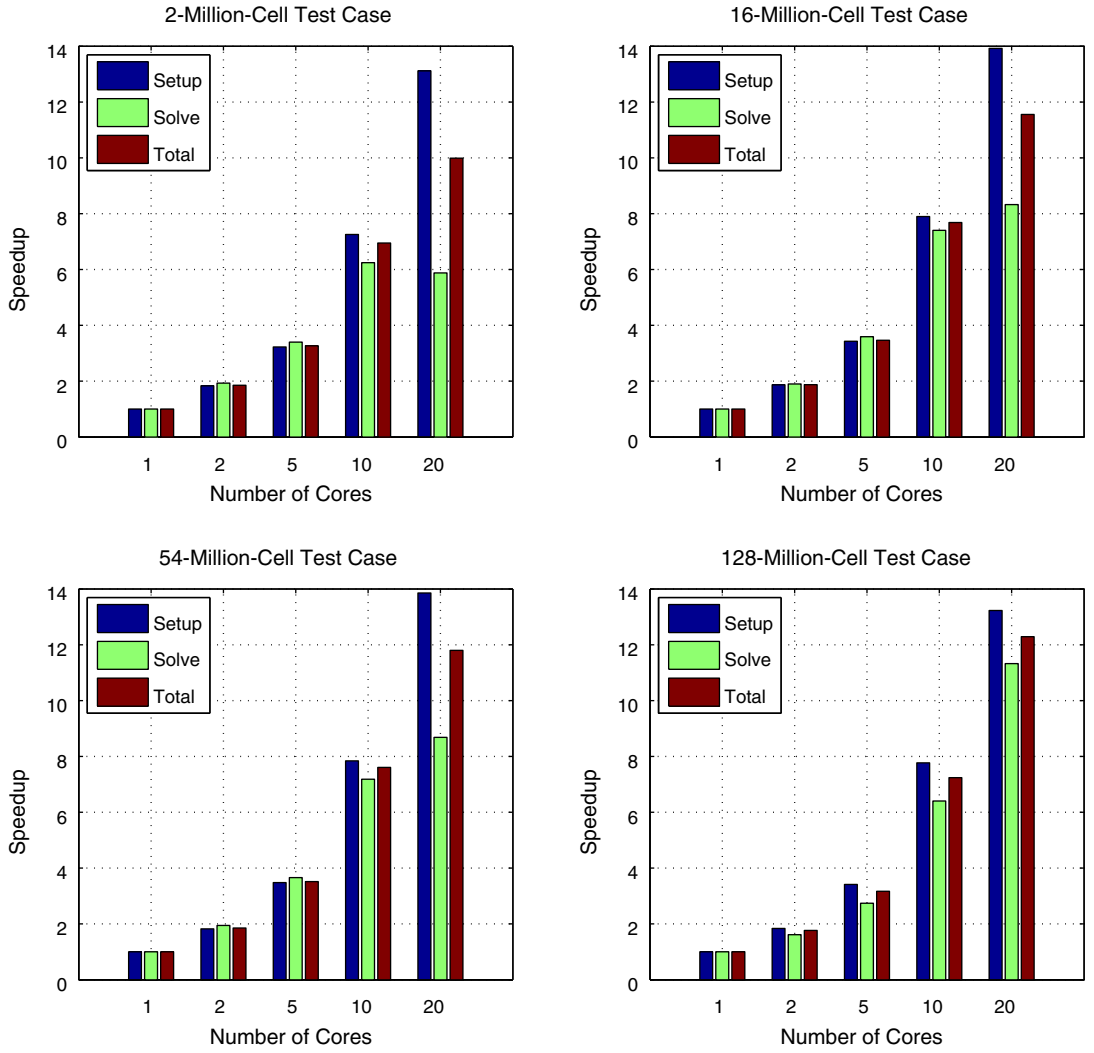

Fig. 11—Speedup of setup and solve kernels of AMS for several problem sizes running on two sockets of Intel Xeon Processor E5-2690-v2.

AMG solver, SAMG (Stüben 2012). Although the setup phase of AMS shows good scalability, the scalability of the solution phase can be limited by the memory bandwidth on shared-memory architectures with large numbers of cores. We are working on improving the scalability of the solution phase, particularly the local-stage preconditioner, M_l^{-1}. We are also working on optimizing the implementation and scalability of AMS on the Intel Xeon architecture. In addition, we are working on extending the parallel implementation of the solver to run on distributed-memory multinode systems. Another direction we are interested in is studying the scalability of AMS on other emerging massively parallel architectures, such as Intel Xeon Phi and graphics processing units.

Nomenclature

A^c = $N_c \times N_c$ coarse-scale operator
A^f = $N_f \times N_f$ fine-scale operator
A_{EE} = connections between edge cells in A^f, according to the wirebasket definition
$\tilde{A}_{EE}$ = modified A_{EE} by adding the contribution of A_{EI} into A_{EE}
A_{EI} = connections from edge cells to interior cells in A^f, according to the wirebasket definition
A_{EV} = connections from edge cells to vertex cells in A^f, according to the wirebasket definition
A_{IE} = connections from interior cells to edge cells in A^f, according to the wirebasket definition
A_{II} = connections between interior cells in A^f, according to the wirebasket definition
A_{IV} = connections from interior cells to vertex cells in A^f, according to the wirebasket definition
A_{VE} = connections from vertex cells to edge cells in A^f, according to the wirebasket definition
A_{VI} = connections from vertex cells to interior cells in A^f, according to the wirebasket definition
A_{VV} = connections between vertex cells in A^f, according to the wirebasket definition
C_r = coarsening ratio
C_{r_x} = coarsening ratio in the x direction
C_{r_y} = coarsening ratio in the y direction
C_{r_z} = coarsening ratio in the z direction
I_{VV} = identity matrix of size $N_c \times N_c$
L^c = lower operator of the LU decomposition of A^c
M_g^{-1} = global-stage MSFE preconditioning operator
M_l^{-1} = local-stage MSFE preconditioning operator
n = number of dimensions of the domain
n_s = number of smoothing steps
N_c = number of primal-coarse gridblocks
N_{c_x} = number of primal-coarse gridblocks in the x direction
N_{c_y} = number of primal-coarse gridblocks in the y direction
N_{c_z} = number of primal-coarse gridblocks in the z direction
N_d = number of dual-coarse gridblocks
N_f = number of fine-scale gridblocks
N_{f_x} = number of fine-scale gridblocks in the x direction
N_{f_y} = number of fine-scale gridblocks in the y direction
N_{f_z} = number of fine-scale gridblocks in the z direction
N_{th} = number of threads

p = pressure term
p^c = coarse-scale pressure solution
p^f = discretized pressure unknowns on Ω_f
$\tilde{p}^f$ = multiscale fine-scale pressure solution
p_E = pressure in edge cells, according to the wirebasket definition
p_I = pressure in interior cells, according to the wirebasket definition
p_V = pressure in vertex cells, according to the wirebasket definition
$\mathcal{P} = N_f \times N_c$ prolongation operator
q = source or sink term
q^f = discretized source or sink terms on Ω_f
q_E = source or sink terms in edge cells, according to the wirebasket definition
q_I = source or sink terms in interior cells, according to the wirebasket definition
q_V = the source or sink terms in vertex cells, according to the wirebasket definition
$\mathcal{R} = N_c \times N_f$ restriction operator
U^c = upper operator of the LU decomposition of A^c
W = wirebasket permutation matrix
δ = Kronecker delta
$\boldsymbol{\lambda}$ = positive-definite mobility tensor
ϕ_j^i = basis function associated with the coarse node i in the dual-coarse gridblock $\partial\Omega_j^d$
ω = damping factor
Ω^c = primal-coarse grid
Ω^d = dual-coarse grid
Ω_f = fine grid
$\partial\Omega_j^c$ = boundary of the jth dual-coarse gridblock
$||$ = projection of the vector or operator along the tangential direction of the jth dual-coarse gridblock boundary $\partial\Omega_j^d$
$||\cdot||_2 = l_2$ norm

Acknowledgments

Financial support from Saudi Aramco, Intel Corporation, the industrial affiliates of the Stanford University Petroleum Research Institute for Reservoir Simulation, and the Stanford Earth Sciences Algorithms and Architectures Initiative made this work possible and is gratefully acknowledged.

References

Aarnes, J. E. 2004. On the Use of a Mixed Multiscale Finite Element Method for Greater Flexibility and Increased Speed or Improved Accuracy in Reservoir Simulation. *Multiscale Model. Simul.* **2** (3): 421–439. http://dx.doi.org/10.1137/030600655.

Aarnes, J. and Hou, T. 2002. Multiscale Domain Decomposition Methods for Elliptic Problems with High Aspect Ratios. *Acta Math. Appl. Sin.* **18** (1): 63–76. http://dx.doi.org/10.1007/s102550200004.

Aarnes, J. E., Kippe, V. and Lie, K.-A. 2005. Mixed Multiscale Finite Elements and Streamline Methods for Reservoir Simulation of Large Geomodels. *Adv. Water Resour.* **28** (3): 257–271. http://dx.doi.org/10.1016/j.advwatres.2004.10.007.

Arbogast, T. 2002. Implementation of a Locally Conservative Numerical Subgrid Upscaling Scheme for Two-Phase Darcy Flow. *Computat. Geosci.* **6** (3–4): 453–481. http://dx.doi.org/10.1023/A:1021295215383.

Arbogast, T. and Bryant, S. L. 2002. A Two-Scale Numerical Subgrid Technique for Waterflood Simulations. *SPE J.* **7** (4): 446–457. SPE-81909-PA. http://dx.doi.org/10.2118/81909-PA.

Asanovic, K., Bodik, R., Demmel, J. et al. 2009. A View of the Parallel Computing Landscape. *Commun. ACM* **52** (10): 56–67. http://dx.doi.org/10.1145/1562764.1562783.

Cao, H., Tchelepi, H. A., Wallis, J. R. et al. 2005. Parallel Scalable Unstructured CPR-Type Linear Solver for Reservoir Simulation. Presented at the SPE Annual Technical Conference and Exhibition, Dallas, 9–12 October. SPE-96809-MS. http://dx.doi.org/10.2118/96809-MS.

Chen, Z. and Hou, T. Y. 2003. A Mixed Multiscale Finite Element Method for Elliptic Problems with Oscillating Coefficients. *Math. Comput.* **72** (242): 541–576. http://dx.doi.org/10.1090/S0025-5718-02-01441-2.

Christie, M. A. and Blunt, M. J. 2001. Tenth SPE Comparative Solution Project: A Comparison of Upscaling Techniques. *SPE Res Eval & Eng* **4** (4): 308–317. SPE-72469-PA. http://dx.doi.org/10.2118/72469-PA.

Clees, T. and Ganzer, L. 2007. An Efficient Algebraic Multigrid Solver Strategy for Adaptive Implicit Methods in Oil-Reservoir Simulation. *SPE J.* **15** (3): 670–681. SPE-105789-PA. http://dx.doi.org/10.2118/105789-PA.

Durlofsky, L. J. 2005. Upscaling and Gridding of Fine Scale Geological Models for Flow Simulation. Oral presentation given at the 8th International Forum on Reservoir Simulation, Stresa, Italy, 20–24 June.

Efendiev, Y. and Hou, T. Y. 2009. *Multiscale Finite Element Methods: Theory and Applications*, Vol. 4. New York City: Springer.

Efendiev, Y. R., Hou, T. Y. and Wu, X.-H. 2000. Convergence of a Nonconforming Multiscale Finite Element Method. *SIAM J. Numer. Anal.* **37** (3): 888–910. http://dx.doi.org/10.1137/S0036142997330329.

Guennebaud, G. and Jacob, B. 2010. Eigen v3, http://eigen.tuxfamily.org.

Hajibeygi, H. and Jenny, P. 2011. Adaptive Iterative Multiscale Finite Volume Method. *J. Comput. Phys.* **230** (3): 628–643. http://dx.doi.org/10.1016/j.jcp.2010.10.009.

Hajibeygi, H., Bonfigli, G., Hesse, M. A. et al. 2008. Iterative Multiscale Finite-Volume Method. *J. Comput. Phys.* **227** (19): 8604–8621. http://dx.doi.org/10.1016/j.jcp.2008.06.013.

Hou, T. Y. and Wu, X.-H. 1997. A Multiscale Finite Element Method for Elliptic Problems in Composite Materials and Porous Media. *J. Comput. Phys.* **134** (1): 169–189. http://dx.doi.org/10.1006/jcph.1997.5682.

Intel. 2013. Intel Math Kernel Library, https://software.intel.com/en-us/articles/intel-mkl/.

Jenny, P., Lee, S. H. and Tchelepi, H. A. 2003. Multi-Scale Finite-Volume Method for Elliptic Problems in Subsurface Flow Simulation. *J. Comput. Phys.* **187** (1): 47–67. http://dx.doi.org/10.1016/S0021-9991(03)00075-5.

Klie, H., Wheeler, M. F., Stüben, K. et al. 2007. Deflation AMG Solvers for Highly Ill-Conditioned Reservoir Simulation Problems. Presented at the SPE Reservoir Simulation Symposium, Houston, 26–28 February. SPE-105820-MS. http://dx.doi.org/10.2118/105820-MS.

Lunati, I. and Jenny, P. 2007. Treating Highly Anisotropic Subsurface Flow With the Multiscale Finite-Volume Method. *Multiscale Model. Simul.* **6** (1): 308–318. http://dx.doi.org/10.1137/050638928.

Remy, N., Boucher, A. and Wu, J. 2009. *Applied Geostatistics with SGeMS: A User's Guide*. Cambridge: Cambridge University Press.

Saad, Y. 2003. *Iterative Methods for Sparse Linear Systems*, second edition. Philadelphia, Pennsylvania: SIAM.

Smith, B. F., Bjorstad, P. E., Gropp, W. D. et al. 1998. Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations. *SIAM Rev.* **40** (1): 169–170.

Stüben, K. 2001. A Review of Algebraic Multigrid. *J. Comput. Appl. Math.* **128** (1–2): 281–309. http://dx.doi.org/10.1016/S0377-0427(00)00516-1.

Stüben, K. 2012. SAMG User's Manual. Sankt Augustin, Germany: Fraunhofer Institute for Algorithms and Scientific Computing (SCAI).

Stüben, K., Delaney, P. and Chmakov, S. 2003. Algebraic Multigrid (AMG) for Ground Water Flow and Oil Reservoir Simulation. Oral presentation given at MODFLOW and More 2003: Understanding through Modeling Conference, Golden, Colorado, 16–19 September.

Stüben, K., Clees, T., Klie, H. et al. 2007. Algebraic Multigrid Methods (AMG) for the Efficient Solution of Fully Implicit Formulations in Reservoir Simulation. Presented at the SPE Reservoir Simulation Symposium, Houston, 26–28 February. SPE-105832-MS. http://dx.doi.org/10.2118/105832-MS.

Tchelepi, H. A. and Wallis, J. 2010. Apparatus, Method and System for Improved Reservoir Simulation Using an Algebraic Cascading Class Linear Solver. US Patent No. 7,684,967.

Toselli, A. and Widlund, O. 2005. *Domain Decomposition Methods: Algorithms and Theory*, Vol. 3. Berlin: Springer.

Wallis, J. 1983. Incomplete Gaussian Elimination as a Preconditioning for Generalized Conjugate Gradient Acceleration. Presented at the SPE Reservoir Simulation Symposium, San Francisco, 15–18 November. SPE-12265-MS. http://dx.doi.org/10.2118/12265-MS.

Wallis, J., Kendall, R. and Little, T. E. 1985. Constrained Residual Acceleration of Conjugate Residual Methods. Presented at the SPE Reservoir Simulation Symposium, Dallas, 10–13 February. SPE-13536-MS. http://dx.doi.org/10.2118/13536-MS.

Wang, Y., Hajibeygi, H. and Tchelepi, H. A. 2014. Algebraic Multiscale Solver for Flow in Heterogeneous Porous Media. *J. Comput. Phys.*

259 (15 February): 284–303. http://dx.doi.org/10.1016/j.jcp.2013.11.024.

Zhou, H. and Tchelepi, H. A. 2012. Two-Stage Algebraic Multiscale Linear Solver for Highly Heterogeneous Reservoir Models. *SPE J.* **17** (2): 523–539. SPE-141473-PA. http://dx.doi.org/10.2118/141473-PA.

Appendix A—Optimizing AMS Parameter Settings

Here, we discuss the details of the numerical tests performed to find the most scalable settings for our AMS implementation. In particular, we focus on investigating the most efficient settings for the two key options in the algorithm: the value of the coarsening ratio, C_r, and the choice of the local-stage preconditioner, M_l^{-1}. First, the performance of AMS as a function of the coarsening ratio, C_r, is investigated. Then, the performance of AMS with two different local-stage preconditioners—damped point red/black Gauss-Seidel and block Jacobi—is investigated. Finally, the performance of AMS as a function of specific parameters for the damped point red/black Gauss-Seidel local-stage preconditioner is presented. The first set of permeability fields from the second case (i.e., the 2-million-cell cases) was used in all these numerical experiments.

AMS Performance Dependence on the Coarsening Ratio, C_r. To study how the robustness and performance of AMS is affected by varying the value of C_r, AMS was tested with four distinct values of C_r: $7 \times 7 \times 7$, $9 \times 9 \times 9$, $21 \times 21 \times 21$, $63 \times 63 \times 63$. Note that in all the runs, the local-stage preconditioner was the red/black Gauss-Seidel smoother. The results, averaged over five realizations, are shown in **Fig. A-1.** Increasing the value of C_r affects the robustness of the solver, as is evident from the linear relationship between the number of iterations and the value of C_r (Fig. A-1a). This highlights the strong dependence of the robustness of AMS on the value of C_r. Fig. A-1b shows how the total run time of the solver changes when changing the value of C_r. Clearly, the run time grows super-linearly with the value of C_r. Those results suggest that the optimal values of C_r, when using red/black Gauss-Seidel smoother as the local-stage preconditioner, are approximately 10 in each dimension. Thus, for all our tests, we have chosen C_r to be $9 \times 9 \times 9$.

AMS Performance Dependence on the Local-Stage Preconditioner, M_l^{-1}. Two parallel local-stage preconditioners, the point red/black Gauss-Seidel smoother and block Jacobi smoother, were used to study the dependence of AMS performance on the choice of the local-stage preconditioner. The point red/black Gauss-Seidel smoother is computationally inexpensive, accounting for a couple of SpMVs per one color pass, and does not require any setup. However, it is less robust than block smoothing methods when compared on a single-iteration basis. On the other hand, block Jacobi is more robust than point red/black Gauss-Seidel, but is more expensive—in terms of both computational and storage requirements—and requires a setup stage, where all the blocks are factored. In this test, the blocks of the block Jacobi smoother were set to $9 \times 9 \times 9$ to cover one dual-domain support. On the other hand, because red/black Gauss-Seidel has no notion of blocks, nine iterations of the smoother per one M_l^{-1} application were used, to be "fair" with both preconditioners in regard to data access. We used AMS inside a GMRES outer solver. The performance of AMS with those two different local-stage preconditioners is shown in **Fig. A-2.** AMS with point red/black Gauss-Seidel is faster than AMS with block Jacobi, for all the parallel runs (Figs. A-2a through A-2c). In addition, point red/black Gauss-Seidel, with nine iterations per one M_l^{-1} application, is actually more robust than block Jacobi with $9 \times 9 \times 9$ blocks (Fig. A-2d). Those results suggest that point red/black Gauss-Seidel is a better option as the local-stage preconditioner of AMS from both robustness and scalability points of view.

AMS With Point Red/Black Gauss-Seidel: Performance Dependence on ω and n_s. When using AMS with point red/black Gauss-Seidel smoother as the local-stage preconditioner, the number of smoothing steps performed at every AMS iteration, n_s, and the damping factor, ω, can influence the robustness and scalability of the solver. In this experiment, we study this influence numerically by changing AMS configuration to use varying values of n_s and ω, from 1 to 20 and 0.05 to 1.95, respectively. All the runs were performed by a parallel AMS solver running on the 12-core platform. The results are shown in **Figs. A-3 and A-4.** Fig. A-3 shows that when the number of smoothing steps is initially increased from 1 to 10, the overall performance of AMS is improved significantly. However, the performance degrades when increasing the number of smoothing steps further, which indicates that the best value of n_s is approximately 10. Fig. A-4 shows how the value of ω affects the behavior of AMS. Setting $\omega = 1.0$ represents the situation when no damping is used. Clearly, damping does help improve the efficiency of the local-stage preconditioner, and, in effect, the efficiency of AMS. The best ω value happens between 1.7 and 1.8.

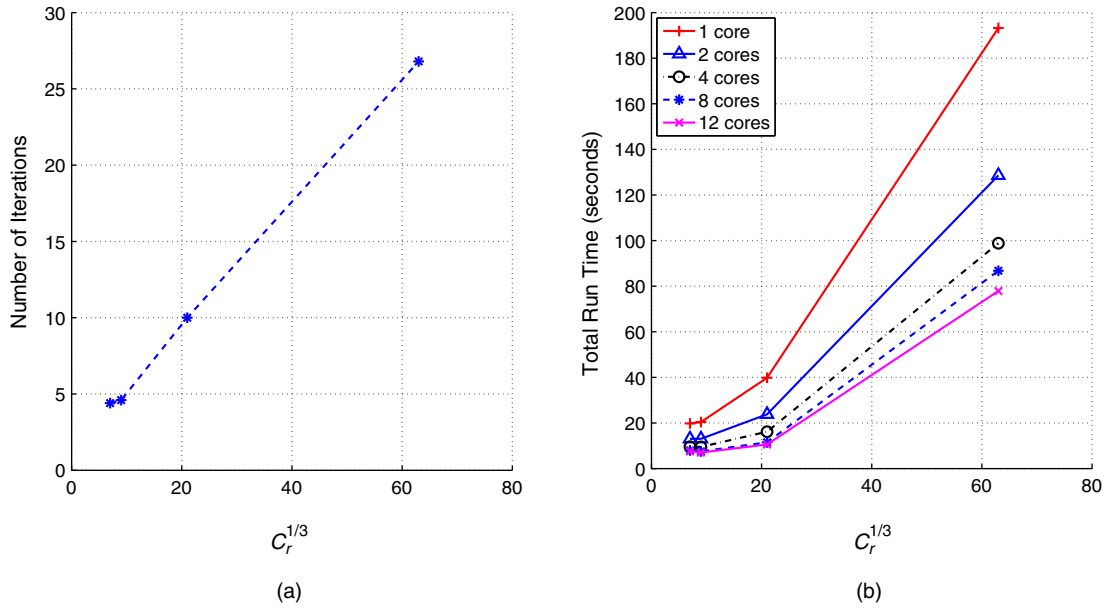

Fig. A-1—Average total number of iterations (a) and average total run time (b) to solve the 2-million-cell problem for varying values of C_r.

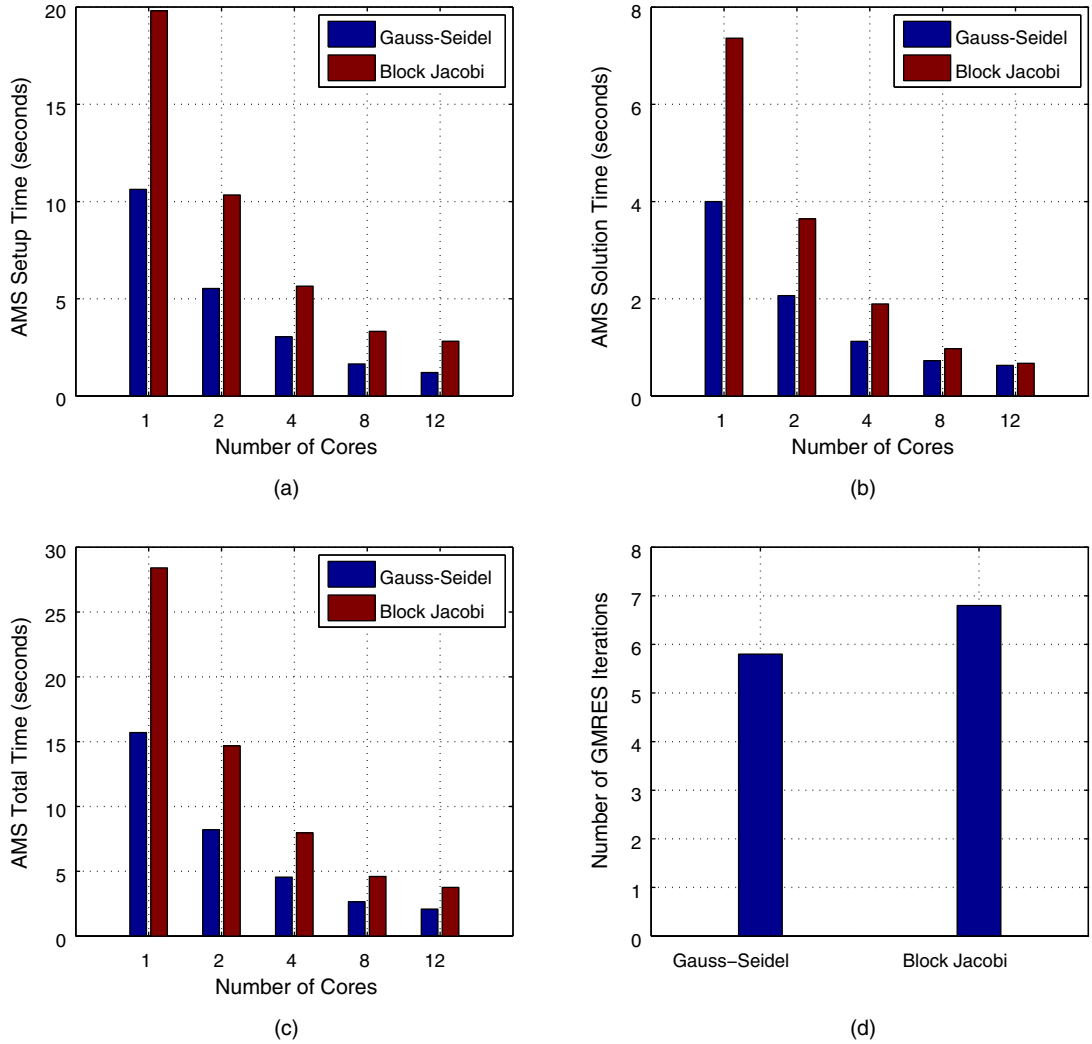

Fig. A-2—Performance of AMS with different local-stage preconditioners, M_l^{-1}: (1) point red/black Gauss-Seidel with nine smoothing steps; (2) block Jacobi with 9 × 9×9 blocks.

Appendix B—Performance Model for AMS Storage

Here, we present a detailed analysis of the parallel AMS storage requirements. We start by defining some basic quantities. Then, we give a set of assumptions made in this model. The storage requirements for the various AMS kernels are then analyzed. Finally, we use the model to estimate the storage requirements for the second set of test cases used in this paper.

Definitions. Define the number of fine-grid cells along the l-axis to be N_{f_l}, where $l \in \{x, y, z\}$. The number of fine degrees of freedom, N_f, is then defined as

$$N_f = N_{f_x} \times N_{f_y} \times N_{f_z}. \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (B\text{-}1)$$

Similarly, define the number of coarse-grid cells along the l-axis to be N_{l_c}, where $l \in \{x, y, z\}$. The number of coarse degrees of freedom (N_{c_l}) is then defined as

$$N_c = N_{c_x} \times N_{c_y} \times N_{c_z}. \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (B\text{-}2)$$

By use of similar notation, define the coarsening ratio along the l-axis to be C_{r_l}, where $l \in \{x, y, z\}$, and is computed as

$$C_{r_l} = \frac{N_{f_l}}{N_{c_l}}, l \in \{x, y, z\}. \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (B\text{-}3)$$

Finally, the fine elliptic operator is denoted as A^f, the coarse operator as A^c, the right-hand side vector as b, the solution vector as x, and the number of threads as N_{th}.

Assumptions. In this storage-requirements analysis, the following assumptions are made. First, we assume a uniform coarsening ratio in all directions:

$$C_{r_x} = C_{r_y} = C_{r_z} = C_r^{1/3}. \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (B\text{-}4)$$

Thus, the coarsening ratio C_r is given as

$$C_r = \frac{N_f}{N_c} \Longleftrightarrow N_c = \frac{N_f}{C_r}. \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (B\text{-}5)$$

In addition, we assume that each thread handles exactly one LU at a time, and continue with that LU until the factorization and forward/backward solution of all the specific dual-domain right-hand-side vectors are completed. Moreover, any extra storage for matrix permutations or dual-domain structures is not accounted for. Finally, the local stage and the outer solver (e.g., GMRES or Richardson) storage requirements are also not accounted for.

Storage Requirements per AMS Kernel. On the basis of the previously discussed assumptions, the storage requirements for the basic AMS kernels are calculated as follows.

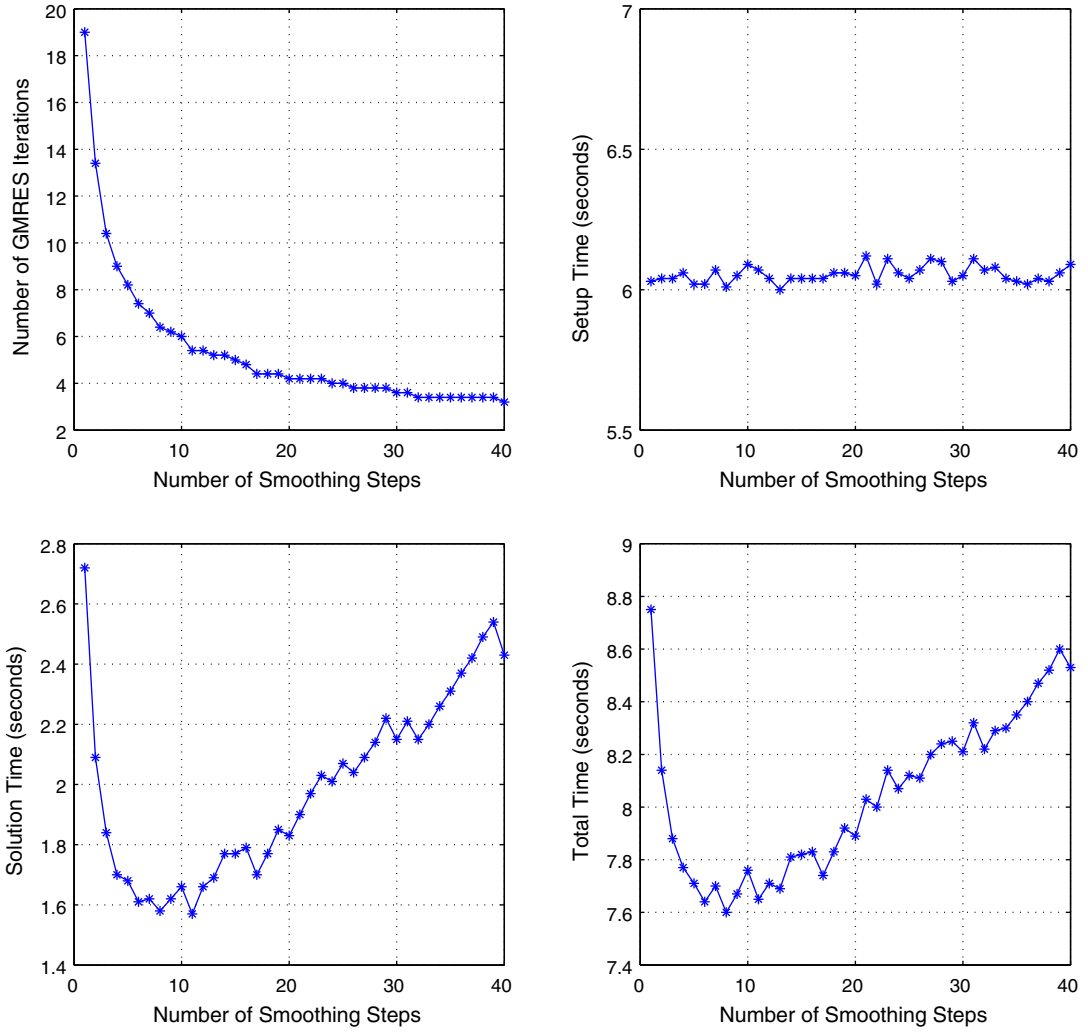

Fig. A-3—Performance of AMS with point red/black Gauss-Seidel as the local-stage preconditioner, M_l^{-1}, with a varying number of smoothing steps, n_s.

Basic Structures. The fine operator, A^f, is an $N_f \times N_f$ banded matrix, with seven bands, which requires storage of $\sim 7 \times N_f$. The right-hand side, b, and the solution vector, x, are of size N_f each. The total storage required for those basic structures is $9N_f$.

Basis-Functions LU Factors. Each octant (i.e., in three dimensions) of a basis function defined on a global support has a local elliptic operator of size $(C_r^{1/3}+1)^3 \times (C_r^{1/3}+1)^3$. The operator bandwidth is $2(C_r^{1/3}+1)^2$. An upper limit on the storage needed to hold the LU for each octant operator of a basis function is

$$(C_r^{1/3}+1)^3 \times 2(C_r^{1/3}+1)^2 = 2(C_r^{1/3}+1)^5. \quad \ldots\ldots \quad \text{(B-6)}$$

The number of different octants of basis functions (i.e., the number of dual domains) is $\approx N_c = N_f/C_r$ (note that this is not an underestimation, given that the size of the dual domains, and hence, the octants of bases on the computational-domain boundaries are halves/quadrants/octants of their general sizes in the interior of the computational domain). In the worst case, where all the LUs of the octant operators of bases are stored, the total storage required for holding the factored bases operators is

$$\frac{N_f}{C_r} \times 2(C_r^{1/3}+1)^5 = \frac{2N_f(C_r^{1/3}+1)^5}{C_r}. \quad \ldots\ldots\ldots\ldots \quad \text{(B-7)}$$

However, considering a practical parallel implementation, at any point in time, only N_{th} LUs are stored. Thus, a more-realistic measure for the total storage required for holding the factored bases operators is

$$2N_{th}(C_r^{1/3}+1)^5. \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \text{(B-8)}$$

Solution of Basis Functions ($\mathcal{P}$ or $\mathcal{R}$). For each dual domain, we have eight right-hand sides to solve for, yielding eight solutions, each of size $(C_r^{1/3}+1)^3$. Taking the number of local basis operators as $\sim N_c = N_f/C_r$, the required storage is

$$8 \times \frac{N_f}{C_r} \times (C_r^{1/3}+1)^3 = \frac{8N_f(C_r^{1/3}+1)^3}{C_r}. \quad \ldots\ldots\ldots \quad \text{(B-9)}$$

Coarse System LU. The bandwidth of the coarse-scale operator, A^c, is $2(N_c^{2/3}+N_c^{1/3}+1)$. This gives the following upper limit on the storage required for the L and U factors of the coarse-scale system:

$$\begin{aligned} N_c \times 2(N_c^{2/3}+N_c^{1/3}+1) &= 2(N_c^{5/3}+N_c^{4/3}+1) \\ &= 2\left[\left(\frac{N_f}{C_r}\right)^{5/3}+\left(\frac{N_f}{C_r}\right)^{4/3}+1\right]. \quad \ldots\ldots\ldots \quad \text{(B-10)} \end{aligned}$$

Total Storage Requirements. On the basis of the previously discussed analysis, the total storage requirements can be approximated as

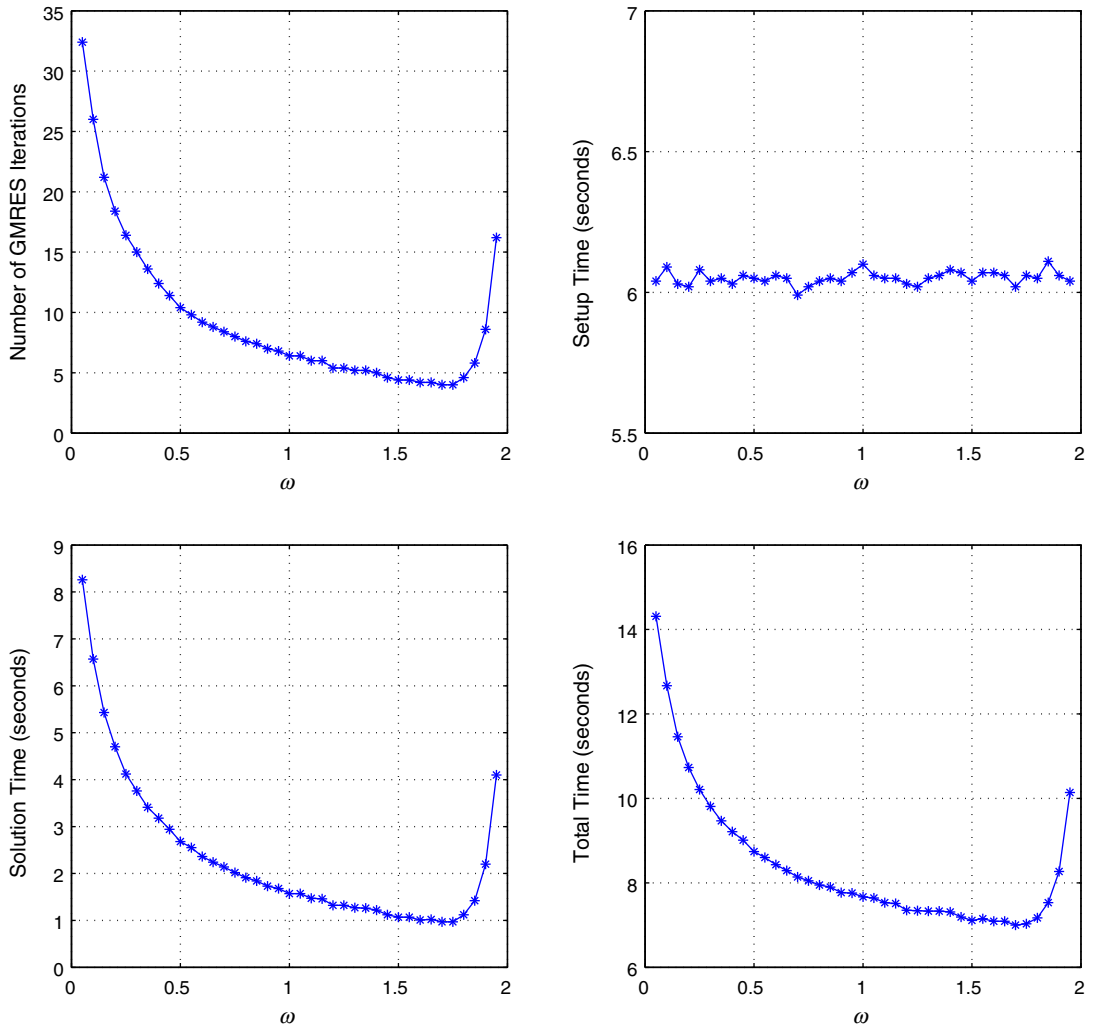

Fig. A-4—Performance of AMS with point red/black Gauss-Seidel as the local-stage preconditioner, M_l^{-1}, with a varying value of the damping factor, ω.

$$9N_f + 2N_{th}(C_r^{1/3}+1)^5 + \frac{8N_f(C_r^{1/3}+1)^3}{C_r} + 2\left[\left(\frac{N_f}{C_r}\right)^{5/3} + \left(\frac{N_f}{C_r}\right)^{4/3} + 1\right]. \quad \text{(B-11)}$$

After some basic algebraic simplifications, we can arrive at this formula:

$$9N_f + 2\left\{(C_r^{1/3}+1)^3\left[\frac{4N_f}{C_r} + N_{th}(C_r^{1/3}+1)^2\right] + \left[\left(\frac{N_f}{C_r}\right)^{5/3} + \left(\frac{N_f}{C_r}\right)^{4/3} + 1\right]\right\}. \quad \text{(B-12)}$$

We can further simplify Eq. B-12 by assuming that the size of the octant of a basis function can be approximated as $C_r \times C_r$, instead of $(C_r^{1/3}+1)^3 \times (C_r^{1/3}+1)^3$. Note that even this seems as an underestimation, it can still serve as an upper limit because the basis-function linear systems would typically be solved with an algorithm that uses a fill-in-reducing permutation, which should reduce the fill-in by at least a factor of one-half from the worst-case fully filled band assumed earlier in this model. Next, we assume that the size of the solutions of basis functions (i.e., Eq. B-9), can be approximated as $8N_f$, again by assuming that $(C_r^{1/3}+1)^3 \approx C_r$. This is also a reasonable assumption given that any point in the fine-scale computational domain would have at most eight coarse-scale points to interpolate from. With this, Eq. B-12 simplifies to

$$17N_f + 2\left[N_{th}C_r^{5/3} + \left(\frac{N_f}{C_r}\right)^{5/3} + \left(\frac{N_f}{C_r}\right)^{4/3} + 1\right]. \quad \text{(B-13)}$$

Examples. Taking the parameters of the second test case, with $C_r = 9 \times 9 \times 9$ and assuming we use 12 threads, we can arrive at these approximate storage requirements (note that the double-precision word is assumed to occupy 8 bytes):

- 2 million cells ($N_f = 126^3$): approximately 280 MB
- 16 million cells ($N_f = 252^3$): approximately 2.4 GB
- 54 million cells ($N_f = 378^3$): approximately 9.1 GB
- 128 million cells ($N_f = 504^3$): approximately 25.1 GB

Abdulrahman M. Manea is a computational scientist at the Upstream Advanced Research Center in Saudi Aramco. His current research interests include parallel linear and nonlinear solvers, large-scale parallel simulation, high-performance scientific computing, and unstructured gridding and discretization. Manea is a member of SPE. He holds a PhD degree in energy resources engineering with a minor in computational mathematics, from Stanford University, and master's and

bachelor's degrees, both in computer science, from King Fahd University of Petroleum and Minerals, Saudi Arabia.

Jason Sewall is an application engineer at Intel, working on parallel numerical algorithms and ecosystem enabling. His research interests include physical simulation, numerical analysis, parallel programming models, and algorithms. Sewall has coauthored dozens of research papers across many conferences and journals in computer science and other disciplines. He holds PhD and master's degrees, both in computer science, from the University of North Carolina, and holds bachelor's degrees in mathematics and in computer science from the University of Maine.

Hamdi A. Tchelepi is a professor at Stanford University and codirector of the university's Center for Computational Earth and Environmental Sciences. His research covers various aspects of numerical modeling of flow and transport in natural porous media. Specific interests include analysis of unstable miscible and immiscible flows in heterogeneous formations; scalable (efficient for large problems) linear and nonlinear solution algorithms of multiphase flow in highly heterogeneous systems; and stochastic-moment equation methods for quantifying the uncertainty associated with predictions of low performance in the presence of limited reservoir-characterization data. Tchelepi is a member of SPE.