SPE-173259-MS

Parallel Multiscale Linear Solver for Highly Detailed Reservoir Models

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

This paper was prepared for presentation at the SPE Reservoir Simulation Symposium held in Houston, Texas, USA, 23–25 February 2015.

Abstract

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 employs 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 multi- 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 impact on the asymptotic convergence rate and parallel computational efficiency.

The balance between convergence rate and parallel efficiency as a function of the coarsening ratio (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. The problems range in size from several-million to 128-million cells. AMS shows excellent behavior for handling a fixed-size problem as a function of the number of cores (so-called strong scaling). Specifically, for a 128-million cell problem, a speed-up of nine-fold is obtained on a single-node 12-core shared-memory architecture (dual-socket multi-core Intel® Xeon® E5-2620-v2), and more than twelve-fold on a single-node 20-core shared-memory architecture (dual-socket multi-core Intel® Xeon® E5-2690 v2). These are encouraging results given the limited memory-bandwidth that cores can share in 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 SAMG solver 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,

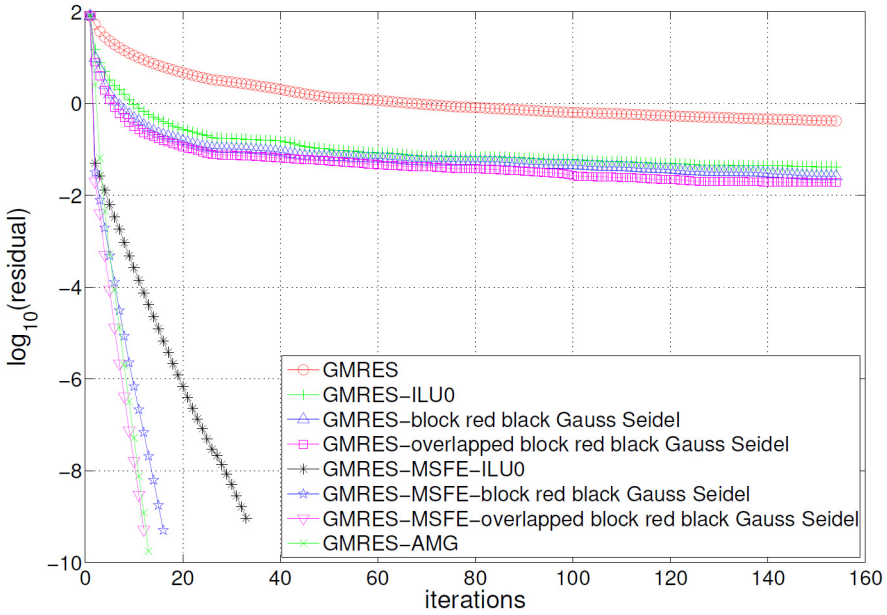

Figure 1—Convergence histories when solving equation 1 for a 1.2 million-cell problem derived from SPE10 top-layer by piece-wise constant grid refinement, with two sides as fixed-pressure boundary conditions and the other two sides as no-flow boundary conditions, using (a) GMRES alone; (b) GMRES preconditioned by simple single-level preconditioners: ILU(O) and block red-black Gauss Seidel smoothing—with and without overlapping blocks; (c) GMRES preconditioned by robust multi-level preconditioners: MSFE with ILU(O), MSFE with block red-black Gauss Seidel smoothing—with and without overlapping blocks—and AMG.

highly detailed geological models, in the range of $0(10^8)$ to $0(10^9)$ grid cells, are often needed to resolve such large multi-scale heterogeneities. While 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 using 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 (MS) methods (Hou and Wu 1997; Efendiev et al. 2000; Aames and Hou 2002; Arbogast et al. 2002; Jenny et al. 2003; Chen and Hou 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 giving up on their fine-scale details. In their general framework, MS methods are based on the idea of non-overlapping domain decomposition, where the global fine-scale computational domain is decomposed into sub-domains, on each of which the original problem is solved independently using boundary conditions derived by a localization assumption. Those local solutions, called *basis functions*, are then used to construct a coarse-scale global problem that is much cheaper to solve. In addition, the basis functions are also used to rebuild the fine-scale details from the solution to the coarse-scale problem.

Several MS approaches has been developed in the literature, which can be generally divided into three main categories: the MS finite-element (MSFE) methods (Hou and Wu (1997); Efendiev and Hou (2009)), the MS mixed finite-element (MSMFE) methods (Arbogast et al. (2002); Chen and Hou (2003); Aarnes (2004); Aarnes et al. (2005); Arbogast (2002)), and the MS finite-volume (MSFV) methods (Jenny et al. (2003)). Those 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 along with the flow. In particular, the MSMFE and MSFV methods has a systematic way of constructing a conservative velocity field for mass transport (Wang et al. (2014)).

While the original (single-pass) MS 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 MS 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 involved in constructing the basis functions. To overcome this limitation, the MS method was extended as an iterative solver (Hajibeygi and Jenny (2011); Hajibeygi et al. (2008)), capable of converging to the fine-scale reference solution. Algebraically, the iterative MS method was formulated as a two-stage solver, called Two Stage Algebraic MS (TAMS) (Zhou et al. (2012)), and later as the Algebraic Multiscale Solver (AMS) (Wang et al. (2014)). In TAMS and AMS, the original single-pass MS 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 in (Hajibeygi et al. (2008); Zhou et al. (2012)) shows that the the global stage acts on damping low frequency error modes associated with long-range coupling of the variables, while leaving 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 of the original MS methods. It has been shown, however, in (Zhou et al. (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.

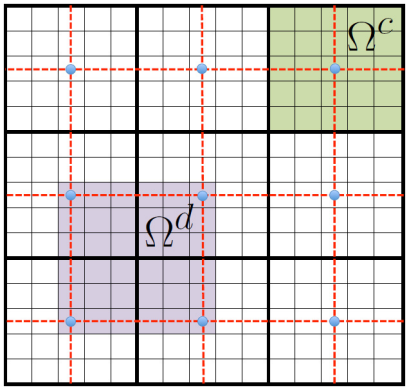

Figure 2—Multiscale grids in 2D. The solid thick black lines denote the primal coarse grid Ω^c, while the dashed red lines denote the dual coarse grid Ω^d. The blue solid circles represent the coarse points, called vertices. Finally, the solid thin lines denote the original fine grid Ω^f.

Besides its specific application to reservoir simulation, AMS has been shown to be a computationally efficient elliptic solver. In fact, MSFE-based AMS solver was shown to have a comparable performance to the state-of-the-art Algebraic Multigrid Solver (AMG) (Stüben (2012)) for multi-million-cell elliptic problems in a serial computational environment (Zhou et al. (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 a 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 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 due to the heterogeneity, and this requires specialized methods to achieve acceptable convergence rates.

The Constrained Pressure Residual (CPR) formulation (Wallis et al. 1983, 1985; Cao et al. 2005) is frequently used to address the particular spectra contributed by the pressure equations so as to accelerate the convergence of the Krylov subspace method 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, 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 rough qualities are usually in direct opposition. Furthermore, in parallel computing environments, the BLAS level-1 and level-2 operations that constitute the core of the Krylov subspace method 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 requires that the entire system — unknowns, matrix, right-hand side, plus several residual and auxiliary vectors — to 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 Algebraic Multigrid (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 et al. 2007; Klie et al. 2007).

In this paper, we highlight the importance of MS as a scalable, efficient preconditioner for heterogeneous elliptic problems. In particular, we focus on the Two-stage Algebraic Multiscale formulation (Zhou et al. 2012; Wang et al. 2014) (AMS), and demonstrate superior performance and scalability on up to 12 cores on a shared-memory system. We further demonstrate the scalability of parallel AMS alone 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 not work in the literature that formally analyzes and experimentally demonstrates the performance of the AMS algorithm on parallel environments. 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: Section 2 highlights some basic background information about the general AMS algorithm. The main computational kernels of AMS are studied and analyzed in Section 3. A basic model for analyzing the performance of the algorithm is given in Section 4. Then, Section 5 presents the results of several numerical experiments highlighting the robustness and scalability of our parallel AMS implementation on different multi- and many-core architectures. Finally, Section 6 highlights some concluding remarks and possible extensions to this work.

Background

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

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

where p is the pressure, λ is the positive-definite mobility tensor, which is defined by the the ratio of the porous medium permeability to the fluid viscosity, and q is a source or sink term.

Equation 1 is a Poisson-like equation; however, it involves highly discontinuous coefficients (i.e., λ), whose values can vary by several orders of magnitude. Thus, the system resulting from discretizing this equation is often badly conditioned, which cannot be solved efficiently using common preconditioned Krylov subspace methods, such as ILU(0)-preconditioned GMRES, (Figure 1). Rather, such a system requires more complex preconditioners that take into account the severe discontinuities in the underlying PDE coefficients.

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

$$A^f p^f = q^f, \tag{2}$$

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-grid blocks, where k ranges from 1 to to the number of primal

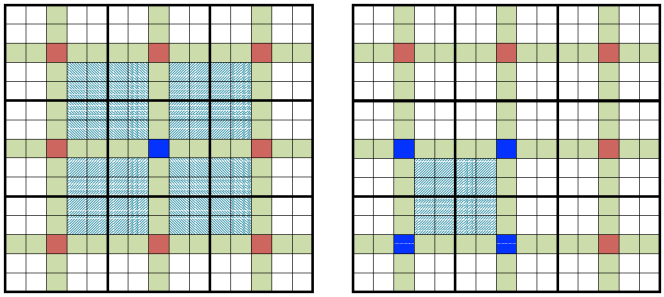

Figure 3—Computing the basis function in 2D on a global support (*left*) and on a quadrant of the support per dual coarse-grid block (*right*).

coarse-grid blocks, N_c. Another coarse grid, called the *dual* coarse grid, Ω_m^d , is introduced by joining the centers of the primal coarse-grid blocks, Ω_k^c , where m ranges from 1 to the number of dual coarse-grid blocks, N_d, as shown in Figure 2. The ratio of the number of fine-scale grid blocks, N_f, to the number of coarse-scale primal coarse-grid blocks, N_c, is denoted as the *coarsening ratio*, C_r. That is:

$$C_r = \frac{N_f}{N_c}. \tag{3}$$

Each dual coarse-grid block Ω_m^d defines a local subdomain on which Equation 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-grid block, where n is the number of the dimensions of the original problem. Algebraically, the basis functions are obtained by solving (Wang et al. 2014):

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

where ϕ_j^i is the basis function associated with the coarse node i in the dual coarse-grid block Ω_j^d , and the subscript $\|$ denotes the projection of the vector or operator along the tangential direction of the dual coarse-grid block boundary, $\partial\Omega_j^d$. Note that such a projection ignores the effect of normal fluxes between dual coarse-grid blocks, 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, namely:

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

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, namely, 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 $\mathcal{R}$:

$$\mathcal{R} = \mathcal{P}^T. \tag{6}$$

The global coarse-scale system can be written as:

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

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

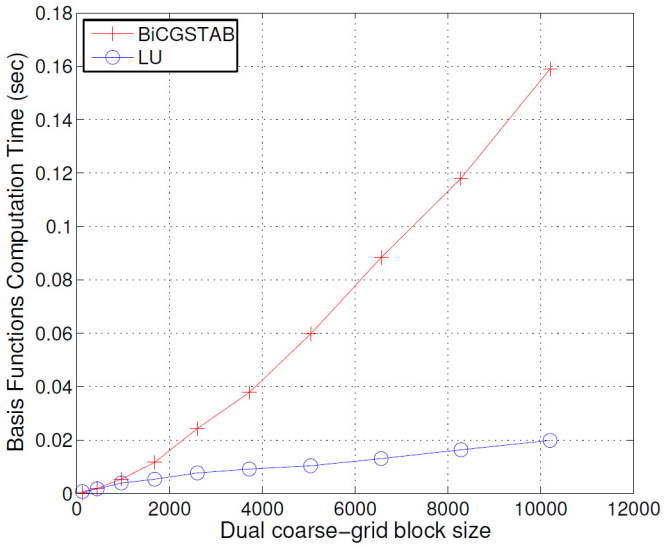

Figure 4—A benchmark comparing the time it takes a direct solvers (LU) (Intel MKL 2013), against an iterative solver (BiCGSTAB) (Guennebaud et al. (2010)), configured with a tight tolerance of 10^{-8} (absolute), for various 2D dual coarse-grid block sizes, from $11^2 \approx 100$ cells to $101^2 \approx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

$$A^c = \mathcal{R} A^f \mathcal{P}. \tag{8}$$

The main computational advantage of multiscale formulations is that the coarse-scale system is much smaller — and so less expensive to solve — than the original fine-scale system. The single-pass MSFE solution is expressed as:

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

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

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

The steps above 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:

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

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 et al. 2012). In addition, we replace A_{vv}, which is defined on the centers of the primal coarse-grid blocks, with our MSFE coarse-operator, A^c, along with replacing the right-hand side term, q_V, with $q^c = Rq^f$ and the unknowns p_V with p^c. This results in the following system:

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

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, pE and pI are found by backward substitution as follows:

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

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

From this, the prolongation operator, $\mathcal{P}$, can be defined as:

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

where W is the wirebasket permutation matrix, and I_{vv} is the identity matrix of size $N_c \times N_c$. It can be easily 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.

It has been shown in Zhou et al. (2012) 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_g^{-1} , that complements the global nature of M_g^{-1} by reducing high-frequency error modes in the residual. The overall preconditioning scheme — called the two-stage algebraic multiscale solver (TAMS) — was proposed by Zhou et al. (2012) and has been further studied and generalized in Wang et al. (2014). The TAMS operator can be written as:

$$M_{TAMS}^{-1} = M_g^{-1} + M_l^{-1} - M_l^{-1} A^f M_g^{-1}. \tag{16}$$

The choice of the local-stage preconditioner, M_g^{-1} , is a design choice that has a strong impact 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:

1. SETUP PHASE:
   a. Select the MSFE Grids (primal and dual coarse grids).
   b. Compute the basis functions, ϕ_j^i , by solving the system defined by Eq. 4.
   c. Assemble the prolongation operator, $\mathcal{P}$, and restriction operator, $\mathcal{R}$, from the basis functions, ϕ_j^i.
   d. Construct the coarse-scale system: $A^c = \mathcal{R}\ A^f\ \mathcal{P}$.
   e. Factor the coarse-scale system: $A^c = L^c U^c$.
   f. Setup the local stage preconditioner, M_g^{-1}.
2. SOLUTION PHASE:
   a. *Global Stage:*
      i. Restrict the fine-scale right hand side: $q^c = \mathcal{R}\ q^f$.
      ii. Solve the coarse-scale system: $p^c = (A^c)^{-1}\ q^c$.
      iii. prolong the coarse-solution to the fine space: $p^f = \mathcal{P}\ p^c$.

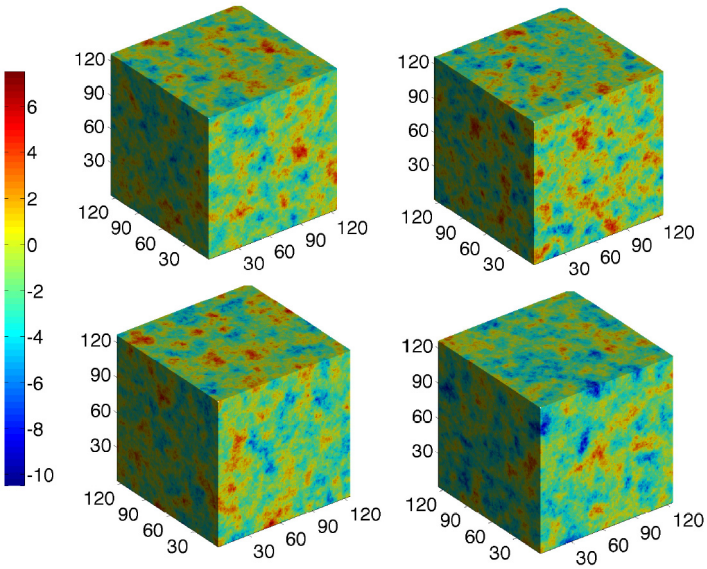

Figure 5—Natural logarithm of four realizations of the first set (126^3) of permeability fields.

b. *Local Stage:*
   i. Apply the local stage preconditioner, M_g^{-1}.

Table 1—Parameter Settings for AMS

Parameter		Value
C_r		9×9×9
M_g^{-1}		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

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, namely, 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 impact on both the robustness and scalability of the overall algorithm. In particular, the value of the coarsening ratio parameter, C_r, defined in Equation 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 $\left(C_r^{1/3}+1\right)^3 \approx C_r$. Assuming we compute the basis functions using 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 computational cost involved in computing the basis functions can be estimated as:

$$\text{Total basis function cost} \approx N_c \times C_r^3 = \frac{N_f}{C_r} \times C_r^3 = N_f \times C_r^2 \tag{17}$$

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

The parallel scalability of the basis functions kernel is also influenced by the value of C_r. Since 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 — increases.

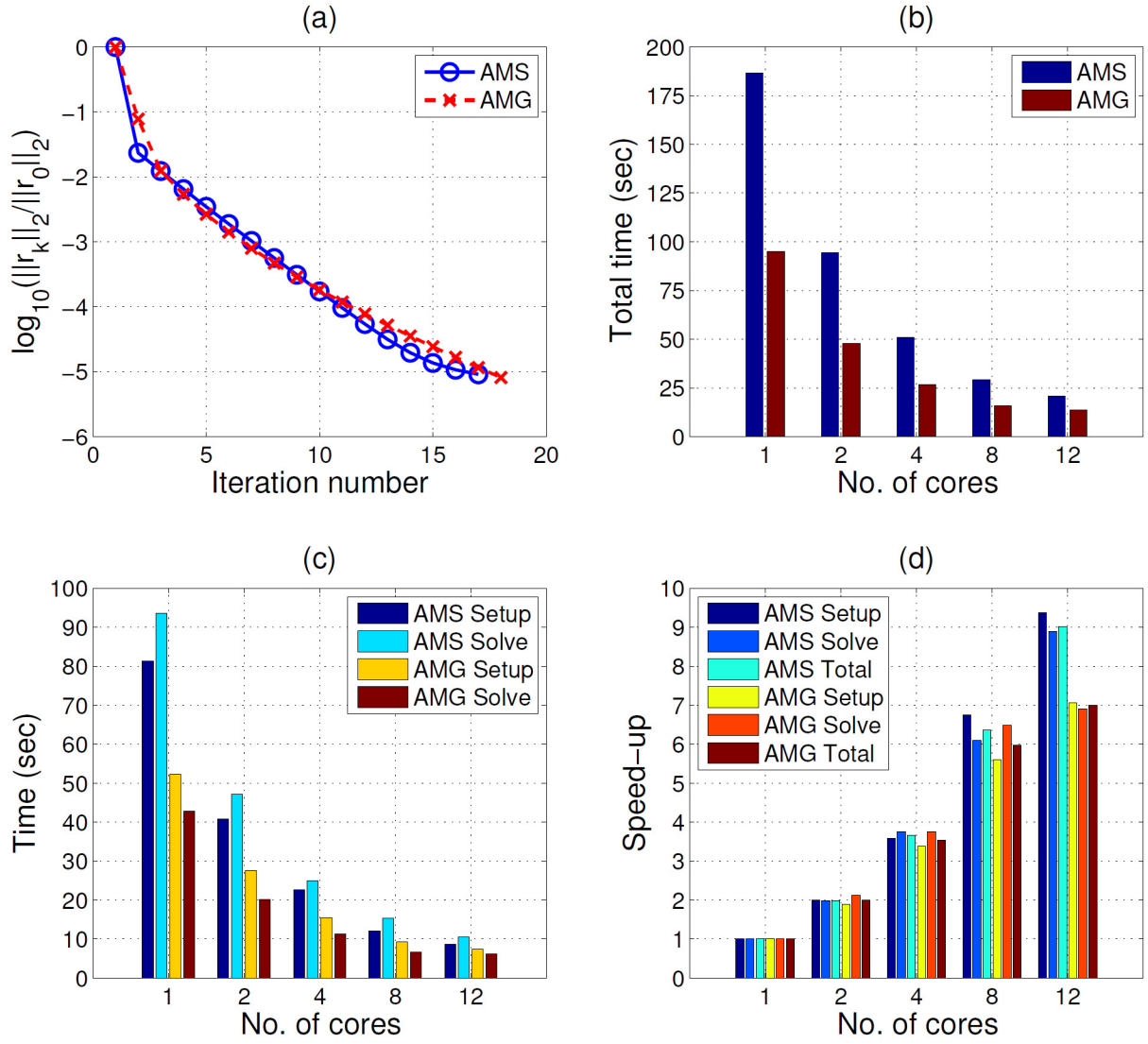

Figure 6—Convergence rate and scalability of AMS and AMG using the SPE10-Derived Model.

Although both the computational complexity and the scalability of the basis functions 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 $C_r \rightarrow N_f$ (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, as $N_c = N_f / C_r$. 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_g^{-1} at the local stage.

Besides the selection of AMS grids, the basis functions computation kernel is 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.

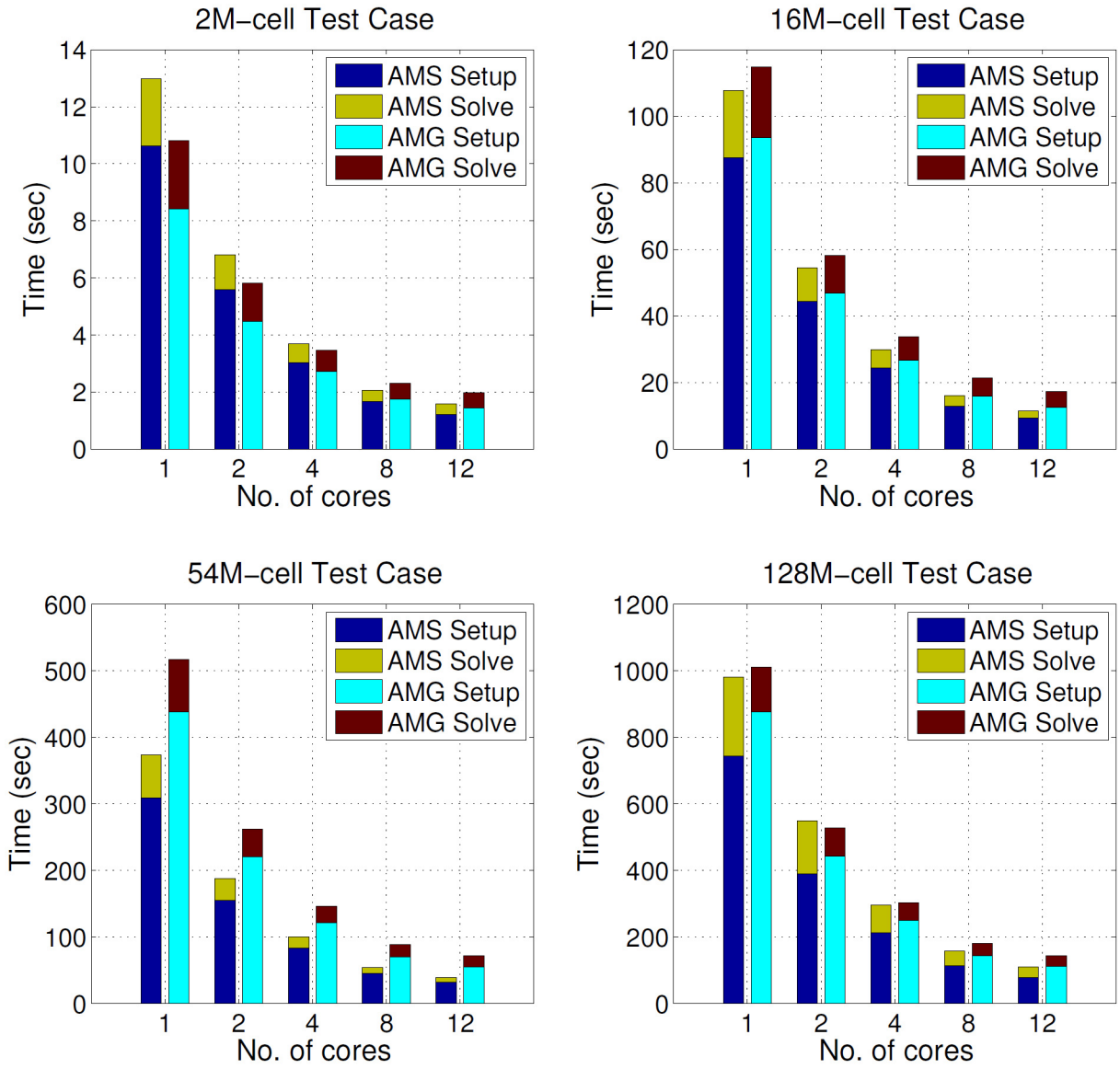

Figure 7—Performance of AMS and AMG for the second test case.

Given their independence, we are free to choose the order the basis functions are computed in. We may formulate the basis subproblems in terms of their complete (compact) regions of support, or further decomposed into quadrants/octants by dual blocks, as shown in Figure 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 utilization point of view: its active working set per thread/core is much larger (4× in 2D and 8 × in 3D) than the second approach, and there is less reuse of the system coefficients.

Using the dual domain-centered approach, we have 4/8 systems to solve for each dual domain; this, plus the relatively small size of the dual domains, make direct factorization attractive. This is because a single factorization can be reused for all of the incident basis functions. The poor conditioning of the local elliptic systems are challenging to Krylov subspace methods without further preconditioning — see the benchmark results in Figure 4 for an analysis using 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

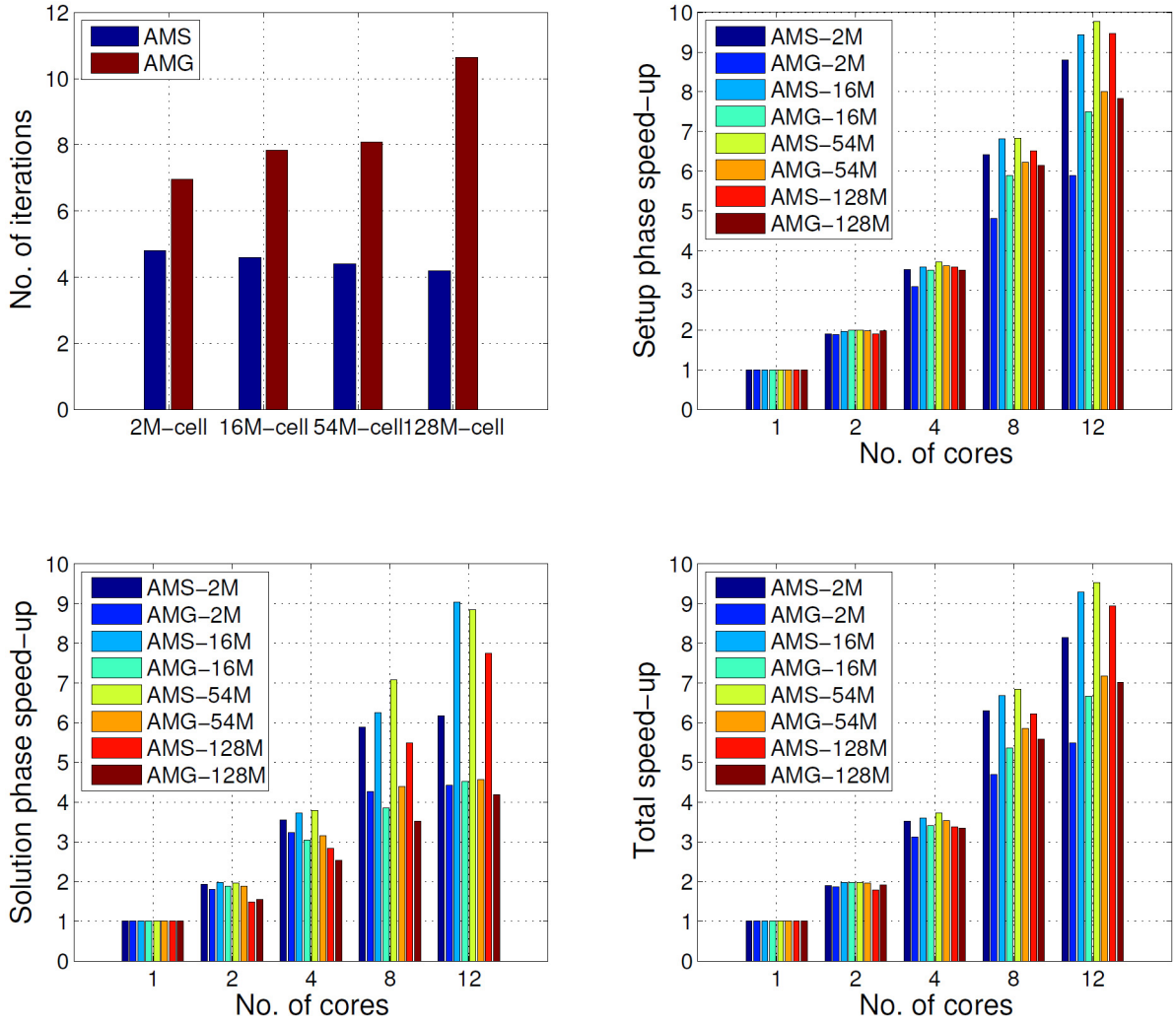

Figure 8—Convergence rate and scalability of AMS and AMG for the second test case.

factorization. Moreover, a considerable computation saving can also be made by re-using the fill-in reducing permutation among dual-domain systems with similar non-zero pattern, which is the case for all interior dual domains.

Construction of the Coarse-Seale System The coarse-scale operator A^c is given by Eq. (8): a triple product of sparse matrices with dimensions $N_c \times N_f$, $N_f \times N_f$, and $N_f \times N_c$ that represents the relationship between each pair of coarse nodes. While 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.

The structure of A^c suggests that we can exploit structure in Eq. (8). Recall 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 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 7-point stencil with spatially-varying coefficients — and admits an efficient banded matrix representation.

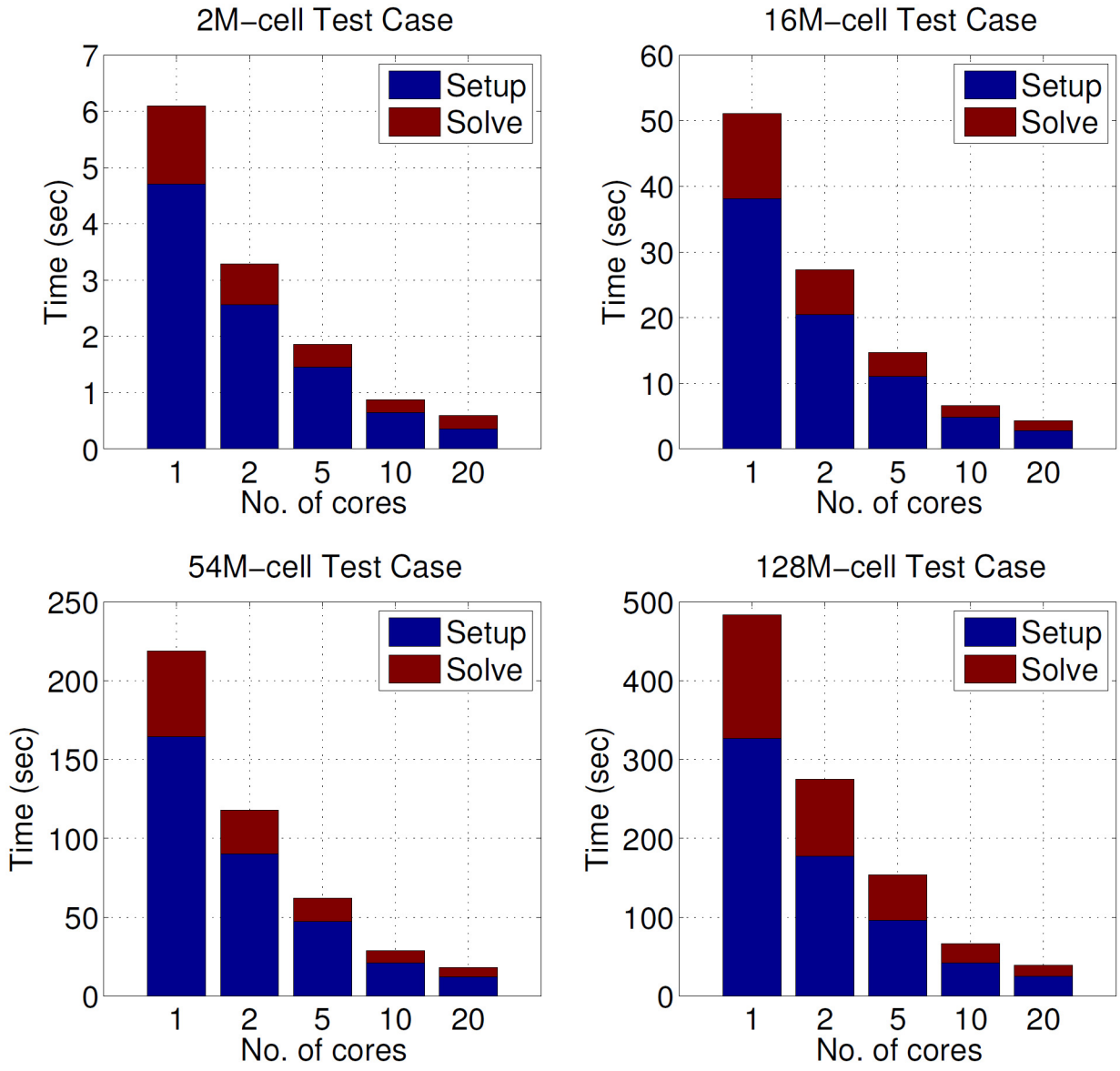

Figure 9—Performance of setup and solve kernels of AMS for several problem sizes running on 2 × Intel® Xeon® Processor E5-2690 v2.

A geometric interpretation of the construction shows that the sub-term $A^f \mathcal{P}$ 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^fP) with which it has an intersection.

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 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.

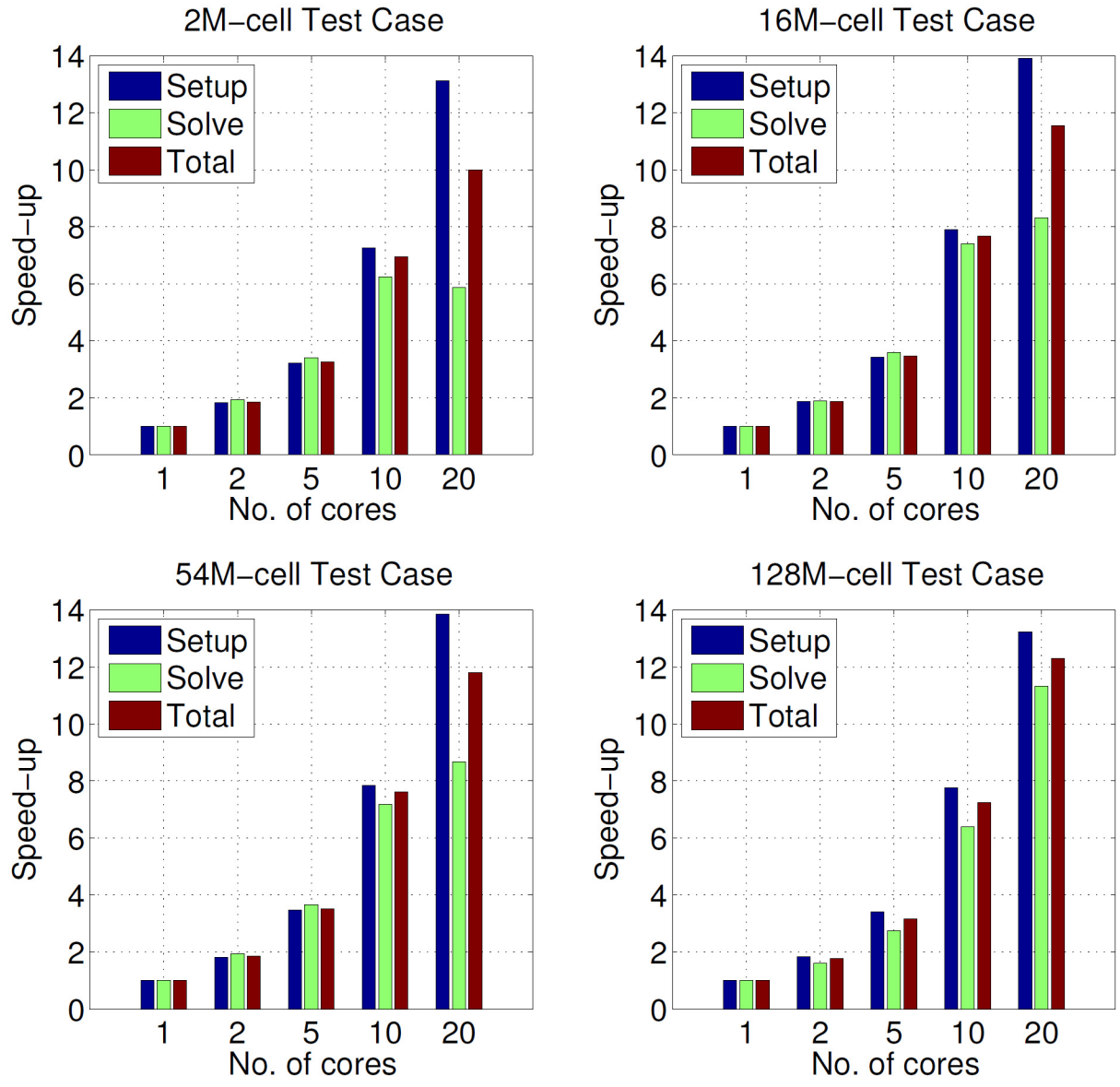

Figure 10—Speed-up of setup and solve kernels of AMS for several problem sizes running on 2× Intel® Xeon® Processor E5-2690 v2.

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

Main requirements for M_g^{-1}

There are two critical constraints that must be honored when choosing an algorithm for M_g^{-1}. First, M_g^{-1} should focus on resolving error components missed by the global stage preconditioner, M_g^{-1} Otherwise, M_g^{-1} would be performing a wasteful computation by repeating effort carried out by M_g^{-1}. Second, M_g^{-1} must have scalable setup and solve kernels to aid overall AMS scalability.

Many robust algorithms for M_g^{-1} (Zhou et al. 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 Incomplete LU factorization (ILU) and Block Incomplete LU factorization (BILU). On the other hand, there are other robust and parallelizable algorithms for M_g^{-1}, such as block-smoothing and Additive-Shwarz techniques. Simple algorithms like point smoothers do not have any setup cost, but this comes at

the expense of convergence. The choice between the later two classes of algorithms for M_g^{-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 the appendix.

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 handside 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. Those basic kernels have minimal impact on the total solution phase cost unless the value of C_r is extremely small.

The local stage, where we apply M_g^{-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, following the steps described in Section 2. A more detailed storage complexity analysis is given in the Appendix.

For simplicity's sake, we will restrict our analysis to cubic domains. The primary parameters to the problem are the fine-scale grid dimensions $N_x = N_y = N_z = N_f^{1/3}$, and coarsening ratio C_r. With Eq. (3), we have the coarse system dimensions $C_x = C_y = C_z = N_c^{1/3}$.

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-wise symmetry.

The fine system thus has N_f double-precision quantities to store for the unknown vector p, N_f doubles for the right-hand side q, and $N_f \times 7$ doubles to represent A^f in banded form, for a total fine-scale base working set of $9 \times N_f \times 8 = 72N_f$ 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 using the overlapping-basis function technique described in Section 3, and factored into L^cU^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 $2\left(N_c^{2/3} + N_c^{1/3} + 1\right)$, the L^cU^c factorization has $2N_c\left(N_c^{2/3} + N_c^{1/3} + 1\right) = O(N_c^{5/3})$ 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, $A^c = \mathcal{R}\ A^f\ P$.

Each coarse point supports a $\left(2C_r^{1/3} + 1\right)^3 = 8C_r + 12C_r^{2/3} + 6C_r^{1/3} + 1 = O\left(C_r\right)$-cell basis function centered at the coarse point, which we represent as eight $\left(C_r^{1/3} + 1\right)^3$-cell partially-overlapping sub-bases — 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 (via forward/backward substitution) the sub-systems that arise from each of the 8 basis functions incident on the Ω_k^d.

Each dual domain corresponds to a $\left(C_r^{1/3}+1\right)^3 \times \left(C_r^{1/3}+1\right)^3$ septadiagonal banded independent sub-operator of A^f; this is factored into LU with $\left(C_r^{1/3}+1\right)^3 \times 2 \times \left(C_r^{1/3}+1\right)^2 = 2\left(C_r^{1/3}+1\right)^5$ nonzeros (following the same conservative estimates outlined above).

A key factor in the design of AMS algorithm is that there are $\left(N_c^{1/3}+1\right)^3$ 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 sub-operator, which can be efficiently handled by the hierarchical memory systems of modern CPUs. Even the smallest problem size we have considered in this work $\left(N_f^{1/3}=126\right)$ yields plentiful independent systems $\left(N_c^{1/3}=126/9=14; 15^3=3375\right)$ for ensuring core occupancy.

Galerkin product, $\mathcal{R}$ $\mathbf{A^fP}$

The basis functions are used to prolong the coarse system 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 as described in Section 3.

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. $\left(2\times C_r^{1/3}+1\right)^3$ 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 $2^{(3-o)}\times\left(C_r^{1/3}+1\right)^3$ multiply-adds each, where o is the number of indices (i.e. i, j, k) that differ from the center point 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 on 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 Fraunhofer's 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 multi-core shared-memory architecture.[1]

Test Cases Design

Four test cases were used. The first test case is derived from the SPE10 Problem (Christie et al. 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.

[1] 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 using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Test Case 1

The first test case uses a permeability field derived from the SPE10 Problem (Christie et al. 2001), which is modified in two ways: First, the original size of SPE10 problem (1.1M, where M denotes 'million') is increased to 16M, by tiling the whole permeability field in each direction to reach the desired size, which is 252^3 ~16M 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- directions.

Test Cases 2,3

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 ~ 2 million (2M) 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 ~ 16M cells, 378^3 ~ 54M cells, and finally 504^3 ~ 128M cells. For all cases, the variance of log-permeability is 4, which represents an extremely heterogenous case. The correlation length of the permeability in each direction is one tenth of the dimension of the domain. Each set has 5 equiprobable realizations, see Figure 5; thus, all the presented results for these cases have been averaged over the 5 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. 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 1 and 0, 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. $\frac{\|r_k\|_2}{\|r_0\|_2} \leq 10^{-5}$).

Computer platforms

The first two test cases were run on the same platform, namely, a multicore architecture consisting of 12 cores of Intel® Xeon® E5-2620 v2 @ 2.10GHz. Since the third test case is used to show AMS scaling, we have used a machine with a larger configuration, namely, a dual-socket configuration of Intel® Xeon® Processor E5-2690 v2, which is a modern multi-core server chip formerly codenamed 'Ivytown'. Each socket features 25MB of shared L3 cache and 10 cores running at 3GHz, each with 256KB of L2 cache and 32 KB each of L1 data and instruction cache. The cores are capable of 2-way SMT and feature superscalar execution and the AVX SIMD instruction set.

Parameter Settings

AMS parameter settings

As discussed earlier, the performance of the AMS algorithm depends on several key parameters, which 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_g^{-1} . A set of experiments were made to test several options for C_r and M_g^{-1} , in order to come up with the most efficient settings for both. Based on these experiments, we have chosen C_r as $9 \times 9 \times 9$ and M_g^{-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_g^{-1} , namely, the number of smoothing steps, n_s, and the damping factor, ω (see Table 1). The details of the numerical experiments used to obtain the AMS parameter settings are in the Appendix.

AMG parameter settings

We used AMG from Fraunhofer Institute SCAI, release 27a1 of November 2013 (Stüben 2012). AMG was configured to use a V-cycle with one pre-smoothing and one post-smoothing steps 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 128M cells, where we used A2 aggressive coarsening (Stüben (2001)) to reduce memory usage.

Test Case 1: AMS Performance Compared to AMG Using the SPE10-Derived Model

Using the AMS parameter settings outlined in Table 1, the 16M-cell permeability field derived from the SPE10 Problem was used to test both AMS and AMG on the 12-core multicore architecture, as shown in Figure 6. 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 Figure 6-(a). This highlights the robustness of AMS as a linear solver for a challenging permeability field derived from the SPE10 Problem. On the other hand, although AMG takes almost half the time AMS takes to solve the problem (Figure 6-(b)), both solvers show good 'strong scalability' from 1 to 12 cores (Figure 6-(c)). In particular, AMS exhibits an excellent scalability, for both the setup and solution phases, achieving more than 9x speed-up 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 to 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 run on the 12-core multicore architecture. The results, which are averaged over the 5 realizations of each permeability set, are shown in Figure 7 and Figure 8. In 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, Figure 8-(a). Moreover, 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 128M-cell problem, where 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, namely the 16M-, 54M-, and 128M-cell problems, Figure 7. The setup phase of both solvers show good scalability up to 12 cores, Figure 8-(b). In particular, the setup phase of AMS achieves more than 9x speed-up on 12 cores for all the problem sizes, except the smallest 2M 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 2M case, showing 8x to 9x speed-up on 12 cores, Figure 8-(c). On the other hand, AMG's solution phase shows good scalability up to 8 cores, but then the scalability is limited afterwords. The total speed-up achieved by AMS for the 16M, 54M, and 128M cases on 12 cores is at least 9x, Figure 8-(d), which is an excellent scalability behavior, given the limited both 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 Multi-core 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. Figure

9 shows the performance of AMS, broken into setup and solution phases, while Figure 10 shows the setup, solve, and overall speed-up. AMS scales up to 20 cores on all problem sizes, thanks to scaling on both the setup kernel and the solution kernel. In particular, the setup phase has an excellent scalability, reaching up to 13x speed-up 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. Nonetheless, for the 128M-cell case, AMS has around 12.3x overall speed-up on 20 cores (with more than 13x for the setup phase and more than 11x for the solution phase), where the 128M-cell problem is setup 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_g^{-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 up to 20-core shared-memory architecture. Moreover, the solver robustness and scalability were compared against the state-of-the-art 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_g^{-1} We are also working on optimizing the implementation and scalability of AMS on the Intel® Xeon® architecture. Another direction we are interested in is studying the scalability of AMS on other emerging highly parallel architectures, such as Intel® Xeon Phi™ and GPUs.

Acknowledgements

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

References

J. Aarnes and T. Hou. Multiscale Domain Decomposition Methods for Elliptic Problems with High Aspect Ratios. *Acta Mathematicae Applicatae Sinica,* **18**(1):63–76, 2002. ISSN 0168-9673. doi: 10.1007/s102550200004. URL http://dx.doi.org/10.1007/s102550200004.

J. E. Aarnes. On the use of a mixed multiscale finite element method for greater flexibility and increased speed or improved accuracy in reservoir simulation. *Multiscale Modeling & Simulation,* **2**(3):421–439, 2004.

J. E. Aarnes, V. Kippe, and K.-A. Lie. Mixed multiscale finite elements and streamline methods for reservoir simulation of large geomodels. *Advances in Water Resources,* **28**(3):257–271, 2005.

T. Arbogast. Implementation of a locally conservative numerical subgrid upscaling scheme for two-phase Darcy flow. *Computational Geosciences,* **6**(3-4):453–481, 2002.

T. Arbogast, S. L. Bryant, et al A two-scale numerical subgrid technique for waterflood simulations. *SPE Journal,* **7**(04):446–457, 2002.

K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick. A view of the parallel computing landscape. *Commun. ACM,* **52**(10):56–67, Oct. 2009. ISSN 0001-0782. doi: 10.1145/1562764.1562783. URL http://doi.acm.org/10.1145/1562764.1562783.

H. Cao, H. A. Tchelepi, J. R. Wallis, H. E. Yardumian, et al Parallel scalable unstructured CPR-type linear solver for reservoir simulation. In SPE Annual Technical Conference and Exhibition. Society of Petroleum Engineers, 2005.

Z. Chen and T. Y. Hou. A Mixed Multiscale Finite Element Method for Elliptic Problems with Oscillating Coefficients. *Math. Comput.,* **72**(242):541–576, Apr. 2003. ISSN 0025-5718. doi: 10.1090/S0025-5718-02-01441-2. URL http://dx.doi.org/10.1090/S0025-5718-02-01441-2.

M. Christie, M. Blunt, et al Tenth SPE comparative solution project: A comparison of upscaling techniques. *SPE Reservoir Evaluation & Engineering,* **4**(04):308–317, 2001.

T. Clees, L. Ganzer, et al An efficient algebraic multigrid solver strategy for adaptive implicit methods in oil reservoir simulation. *SPE Journal,* **15**:670–681, 2007.

L. J. Durlofsky. Upscaling and gridding of fine scale geological models for flow simulation. In 8th International Forum on Reservoir Simulation Iles Borromees, Stresa, Italy, pages 20–24, 2005.

Y. Efendiev and T. Y. Hou. *Multiscale finite element methods: theory and applications,* volume **4**. Springer, 2009.

Y. R. Efendiev, T. Y. Hou, and X.-H. Wu. Convergence of a Nonconforming Multiscale Finite Element Method. *SIAM J. Numer. Anal.,* **37**(3):888–910, Feb. 2000. ISSN 0036-1429. doi: 10.1137/S0036142997330329. URL http://dx.doi.org/10.1137/S0036142997330329.

G. Guennebaud, B. Jacob, et al *Eigen v3*. http://eigen.tuxfamily.org, 2010.

H. Hajibeygi and P. Jenny. Adaptive iterative multiscale finite volume method. *Journal of Computational Physics,* **230**(3):628–643, 2011.

H. Hajibeygi, G. Bonfigli, M. A. Hesse, and P. Jenny. Iterative multiscale finite-volume method. *Journal of Computational Physics,* **227**(19):8604–8621, 2008.

T. Y. Hou and X.-H. Wu. A Multiscale Finite Element Method for Elliptic Problems in Composite Materials and Porous Media. *J. Comput. Phys.,* **134**(1):169–189, June 1997. ISSN 0021-9991. doi: 10.1006/jcph.1997.5682. URL http://dx.doi.org/10.1006/jcph.1997.5682.

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

P. Jenny, S. H. Lee, and H. A. Tchelepi. Multi-scale Finite-volume Method for Elliptic Problems in Subsurface Flow Simulation. *J. Comput. Phys.,* **187**(1):47–67, May 2003. ISSN 0021-9991. doi: 10.1016/S0021-9991(03)00075-5. URL http://dx.doi.org/10.1016/S0021-9991(03)00075-5.

H. Klie, M. F. Wheeler, K. Stüben, T. Clees, et al Deflation amg solvers for highly ill-conditioned reservoir simulation problems. In SPE Reservoir Simulation Symposium. Society of Petroleum Engineers, 2007.

I. Lunati and P. Jenny. Treating highly anisotropic subsurface flow with the multiscale finite-volume method. *Multiscale Modeling & Simulation,* **6**(1):308–318, 2007.

N. Remy, A. Boucher, and J. Wu. *Applied geostatistics with SGeMS: A user's guide*. Cambridge University Press, 2009.

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

K. Stüben. A review of algebraic multigrid. *Journal of Computational and Applied Mathematics,* **128**(1):281–309, 2001.

K. Stüben. *Samg user's manual*. Fraunhofer Institute SCAI, 2012.

K. Stüben, P. Delaney, and S. Chmakov. Algebraic multigrid (amg) for ground water flow and oil reservoir simulation. In Proceedings of the Conference MODFLOW and More 2003: Understanding through Modeling, pages 17–19, 2003.

K. Stüben, T. Clees, H. Klie, B. Lu, M. F. Wheeler, et al Algebraic multigrid methods (amg) for the efficient solution of fully implicit formulations in reservoir simulation. In SPE Reservoir Simulation Symposium. Society of Petroleum Engineers, 2007.

H. A. Tchelepi and J. Wallis. *Apparatus, method and system for improved reservoir simulation using an algebraic cascading class linear solver,* Mar. 23 2010. US Patent 7,684,967.

A. Toselli and O. Widlund. *Domain decomposition methods: algorithms and theory,* volume **3**. Springer, 2005.

J. Wallis, R. Kendall, T. Little, et al Constrained residual acceleration of conjugate residual methods. In SPE Reservoir Simulation Symposium. Society of Petroleum Engineers, 1985.

J. Wallis et al Incomplete Gaussian elimination as a preconditioning for generalized conjugate gradient acceleration. In SPE Reservoir Simulation Symposium. Society of Petroleum Engineers, 1983.

Y. Wang, H. Hajibeygi, and H. A. Tchelepi. Algebraic Multiscale Solver for Flow in Heterogeneous Porous Media. *J. Comput. Phys.,* **259**:284–303, Feb. 2014. ISSN 0021-9991. doi: 10.1016/j.jcp.2013.11.024. URL http://dx.doi.org/10.1016/j.jcp.2013.11.024.

H. Zhou, H. A. Tchelepi, et al Two-stage algebraic multiscale linear solver for highly heterogeneous reservoir models. *SPE Journal,* **17**(02):523–539, 2012.

Appendix A

Optimizing AMS Parameter Settings

In this section, 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, namely, the value of the coarsening ratio, C_r, and the choice of the local stage preconditioner, M_g^{-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, namely, 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 precondtioner is presented. The first set of permeability fields from the the second case (i.e., the 2M-cell cases) were 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, namely $\{ 7 \times 7 \times 7, 9 \times 9 \times 9, 21 \times 21 \times 21, 63 \times 63 \times 63 \}$. Note that in all the runs, the local stage preconditioner was red-black point Gauss-Seidel smoother. The results, averaged over 5 realizations, are shown in Figure 11. 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, Figure 11-(a). This highlights the strong dependence of the robustness of AMS on the value of C_r. Figure 11-(b) shows how the total runtime of the solver changes when changing the value of C_r. The total runtime of the solver grows exponentially with the value of C_r. Those results suggest that the optimal values of C_r, when using red-black point Gauss-Seidel smoother as the local stage preconditioner, are around 10 in each dimension. Thus, for all our tests, we have chosen C_r to be $9 \times 9 \times 9$.

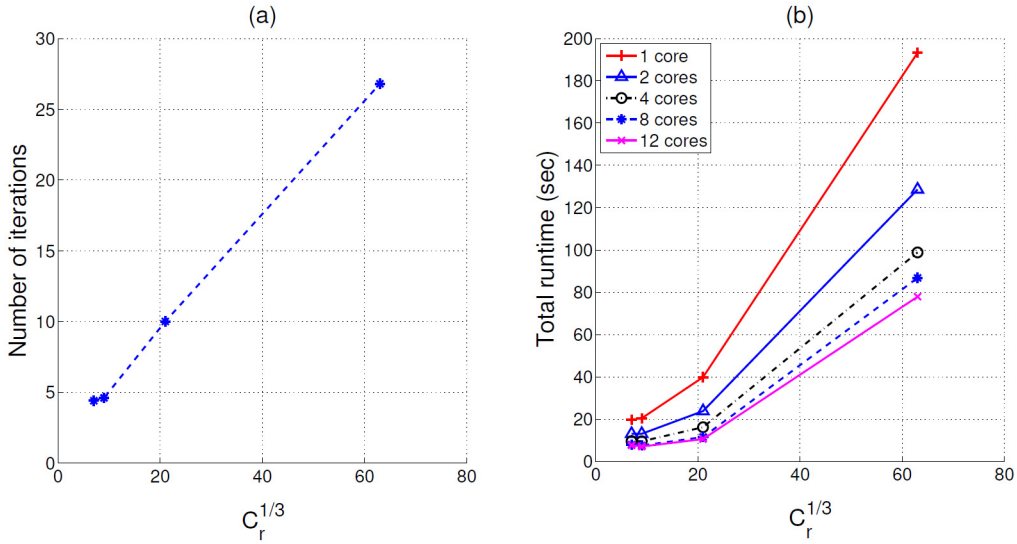

Figure 11—Average total number of iterations (a) and average total runtime (b) to solve the 2M-cell problem for varying values of C_r.

AMS Performance Dependence on the Local Stage Preconditioner, M_g^{-1}

Two parallel local stage preconditioners, namely, 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. Point red-black Gauss-Seidel smoother is computationally cheap, accounting for a couple of SpMV's 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× 9× 9, to cover one dual domain support. On the other hand, since red-black Gauss-Seidel has no notion of blocks, 9 iterations of the smoother per one M_g^{-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 Figure 12. AMS with point red-black Gauss-Seidel is faster than AMS with block Jacobi,

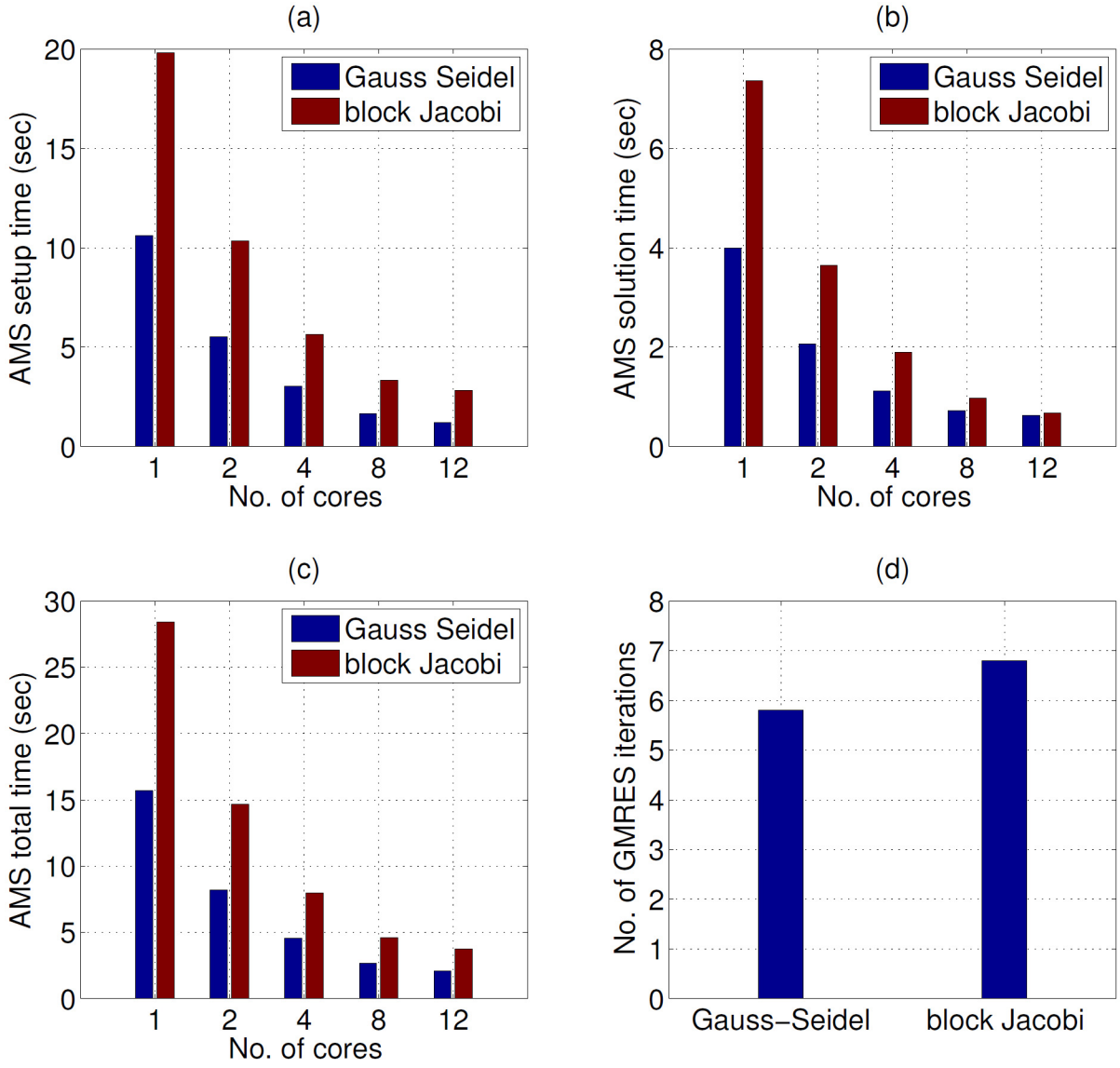

Figure 12—Performance of AMS with different local stage preconditioners, M_g^{-1} : (1) point red-black Gauss-Seidel with 9 smoothing steps (2) Block Jacobi with 9×9×9 blocks.

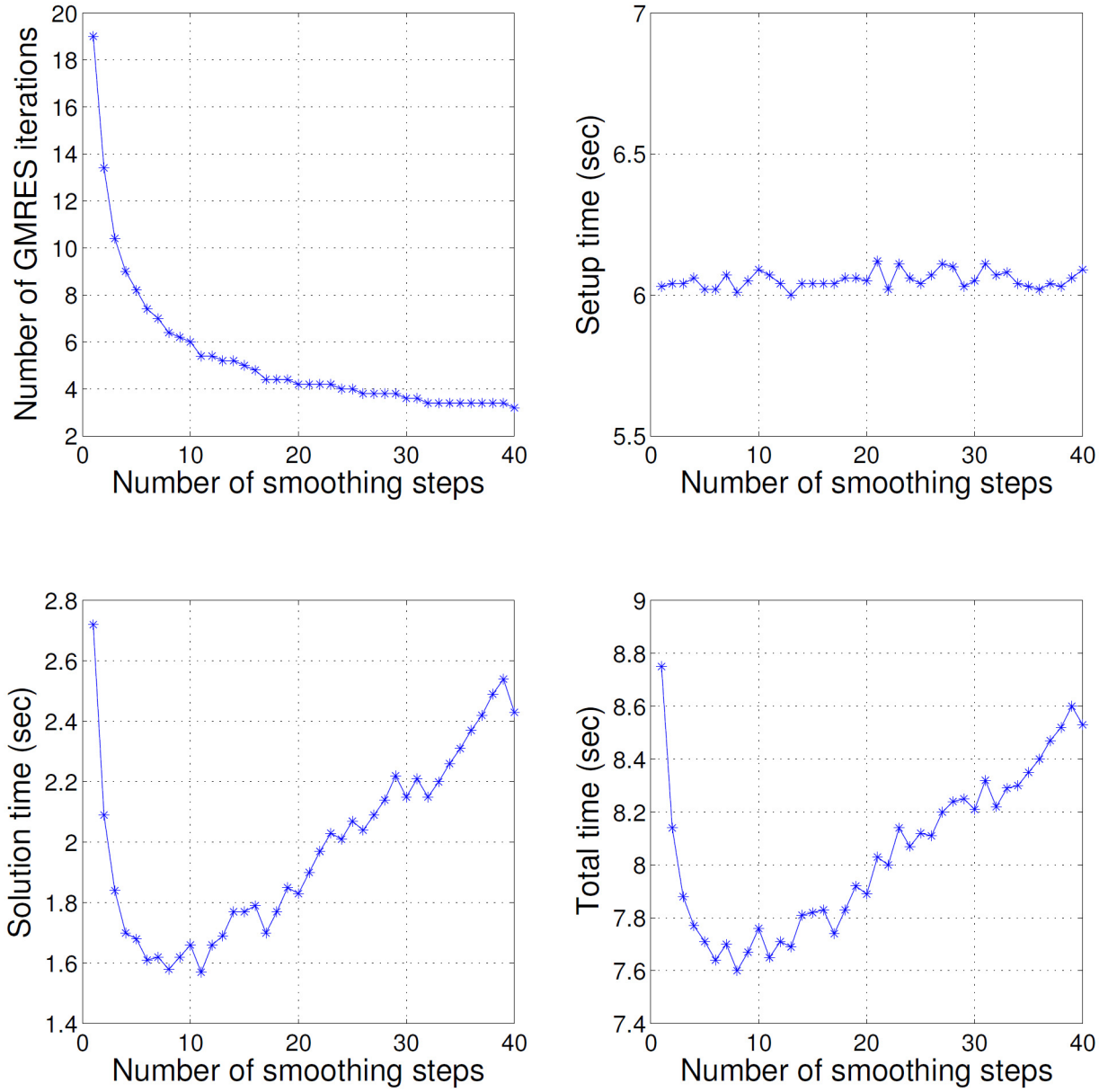

Figure 13—Performance of AMS with point red-black Gauss-Seidel as the local stage preconditioner, M_g^{-1} , with a varying number of smoothing steps n_s.

for all the parallel runs (Figures 12-(a),(b) and (c)). In addition, point red-black Gauss-Seidel, with 9 iterations per one M_g^{-1} application, is actually more robust than block Jacobi with 9×9×9 blocks, Figure 12-(d). 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 from 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 using a parallel AMS solver running on the 12-cores platform. The results are shown in Figure 13 and Figure 14. Figure 13 shows that the number of smoothing steps significantly improves the overall performance of AMS when it is initially increased from 1 to 10, and then the performance degrades afterwards — suggesting that the best value of n_s is around 10. Figure 14 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

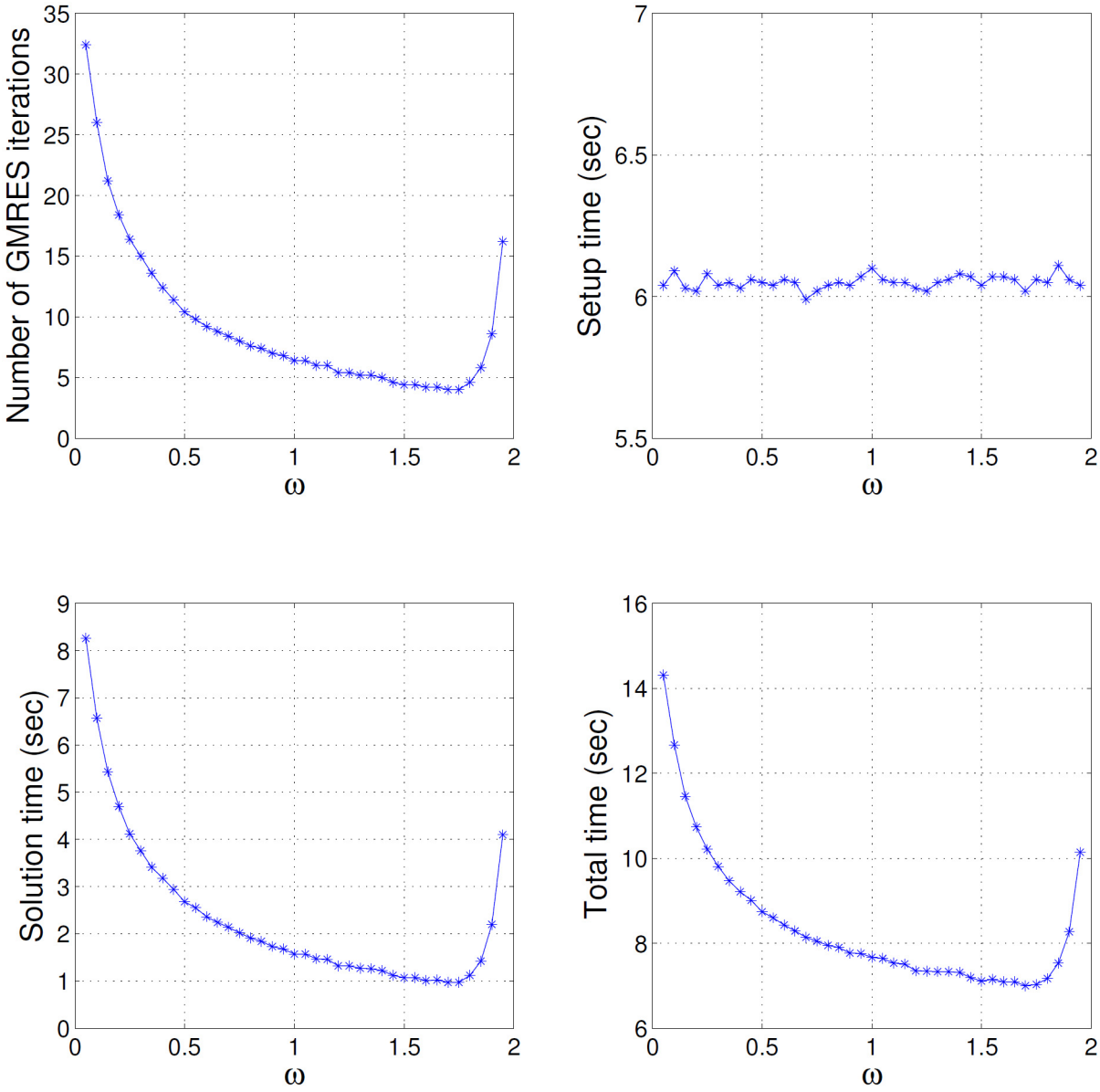

Figure 14—Performance of AMS with point red-black Gauss-Seidel as the local stage preconditioner, M_g^{-1} , with a varying value of the damping factor (ω).

efficiency of the local stage preconditioner, and, in effect, the efficiency of AMS. The best ω value happens between 1.7 and 1.8.

Appendix B

Performance Model for AMS Storage

In this section, 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 is 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_{l_f}, where $l \in \{x, y, z\}$. The number of *fine* degrees of freedom, N_f, is then defined as:

$$N_f = n_{x_f} \times n_{y_f} \times n_{z_f}. \tag{18}$$

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), is then defined as:

$$N_c = n_{x_c} \times n_{y_c} \times n_{z_c}. \tag{19}$$

Using 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:

$$C_{r_l} = \frac{n_{l_f}}{n_{l_c}}, l \in \{x, y, z\}. \tag{20}$$

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 assumption are made. First, we assume a uniform coarsening ratio in all directions, i.e.:

$$C_{r_x} = C_{r_y} = C_{r_z} = C_r^{1/3}. \tag{21}$$

Thus, the coarsening ratio, C_r, is then given as:

$$C_r = \frac{N_f}{N_c} \Leftrightarrow N_c = \frac{N_f}{C_r}. \tag{22}$$

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, dual-domain structures, . . .etc. is not accounted for. Finally, the *local stage* and the outer solver (e.g., GMRES or Richardson) storage requirements are not accounted for.

Storage Requirements per AMS Kernel

Based on the above assumptions, the storage requirements for the basic AMS kernels is calculated as follows.

Basic Structures

The *fine* operator, A^f, is an $N_f \times N_f$-banded matrix, with 7 bands; thus, requiring a storage $\approx 7 \times N_f$. 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 3D) of a basis function defined on a global support has a local elliptic operator of size $\left(C_r^{1/3}+1\right)^3 \times \left(C_r^{1/3}+1\right)^3$. The operator bandwidth is $2\left(C_r^{1/3}+1\right)^2$. An upper limit on the storage needed to hold the *LU* for each octant operator of a basis function is:

$$\left(C_r^{1/3}+1\right)^3 \times 2\left(C_r^{1/3}+1\right)^2 = 2\left(C_r^{1/3}+1\right)^5. \tag{23}$$

The number of different octants of basis functions (i.e., the number of dual domains) $\approx N_c = N_f/C_r$ (note that this is *not* an under-estimation, 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 size in the interior of the computational domain). In the worst case, where all the *LU*'s of the octant operators of bases are stored, the total storage required for holding the factored bases operators is:

$$\frac{N_f}{C_r} \times 2\left(C_r^{1/3}+1\right)^5 = \frac{2N_f\left(C_r^{1/3}+1\right)^5}{C_r}. \tag{24}$$

However, considering a practical parallel implementation, at any point in time, only N_{th} *LU*'s are stored. Thus, a more realistic measure for the total storage required for holding the factored basis operators is:

$$2N_{th}\left(C_r^{1/3}+1\right)^5. \tag{25}$$

Solution of Basis Functions (i.e., $\mathcal{P}$ or $\mathcal{R}$)

For each dual domain, we have 8 right-hand sides to solve for, yielding 8 solutions, each of size $\left(C_r^{1/3}+1\right)^3$. Taking the number of local basis operators $\approx N_c = N_f/C_r$, the required storage is:

$$8 \times \frac{N_f}{C_r} \times \left(C_r^{1/3}+1\right)^3 = \frac{8N_f\left(C_r^{1/3}+1\right)^3}{C_r}. \tag{26}$$

Coarse System LU

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

$$N_c \times 2\left(N_c^{2/3}+N_c^{1/3}+1\right) = 2\left(N_c^{5/3}+N_c^{4/3}+1\right) = 2\left(\left(\frac{N_f}{C_r}\right)^{5/3}+\left(\frac{N_f}{C_r}\right)^{4/3}+1\right). \tag{27}$$

Total Storage Requirements

Based on the above analysis, the total storage requirements can be approximated as:

$$9N_f + 2N_{th}\left(C_r^{1/3}+1\right)^5 + \frac{8N_f\left(C_r^{1/3}+1\right)^3}{C_r} + 2\left(\left(\frac{N_f}{C_r}\right)^{5/3}+\left(\frac{N_f}{C_r}\right)^{4/3}+1\right). \tag{28}$$

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

$$9N_f + 2\left[\left(C_r^{1/3}+1\right)^3\left(\frac{4N_f}{C_r}+N_{th}\left(C_r^{1/3}+1\right)^2\right)+\left(\left(\frac{N_f}{C_r}\right)^{5/3}+\left(\frac{N_f}{C_r}\right)^{4/3}+1\right)\right]. \tag{29}$$

We can further simplify equation 29 by assuming that: (1) The size of the octant of a basis function can be approximated as $C_r \times C_r$, instead of $\left(C_r^{1/3}+1\right)^3 \times \left(C_r^{1/3}+1\right)^3$. Note that, even this seems as an under-estimation, it can still serve as an upper limit, since 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 1/2 from the worst-case fully-filled-band assumed earlier in this model. (2) The size of the solutions of basis functions, i.e. equation 26, can be approximated as $8N_f$, again by assuming that $\left(C_r^{1/3}+1\right)^3 \approx C_r$. This is also a reasonable assumption, given that any point int the *fine*-scale computational domain would have at most 8 *coarse*-scale points to interpolate from. With this, equation 29 simplifies to:

$$17N_f + 2\left(N_{th}C_r^{5/3}+\left(\frac{N_f}{C_r}\right)^{5/3}+\left(\frac{N_f}{C_r}\right)^{4/3}+1\right). \tag{30}$$

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):

- 2M-cell ($N_f = 126^3$): ~ 280 MB
- 16M-cell ($N_f = 252^3$): ~ 2.4 GB
- 54M-cell ($N_f = 378^3$): ~ 9.1 GB
- 128M-cell ($N_f = 504^3$): ~ 25.1 GB