Society of Petroleum Engineers

SPE-193880-MS

A Massively Parallel Algebraic Multiscale Solver for Reservoir Simulation on the GPU Architecture

A. M. Manea and T. Almani, Saudi Aramco

This paper was prepared for presentation at the SPE Reservoir Simulation Conference held in Galveston, Texas, USA, 10–11 April 2019.

Abstract

In this work, the scalability of the Algebraic Multiscale Solver (AMS) (Wang et al. 2014) for the pressure equation arising from incompressible flow in heterogeneous porous media is investigated on the GPU massively parallel architecture. The solvers robustness and scalability is compared against its carefully optimized implementation on the shared-memory multi-core architecture (Manea et al. 2016), which this work is directly extending. Although several components in the AMS algorithm are directly parallelizable, its scalability on GPU's depends heavily on the underlying algorithmic details and data-structures design of each step, where one needs to ensure favorable control- and data-flow on the GPU, while extracting enough parallel work for a massively parallel environment. In addition, the type of the algorithm chosen for each step greatly influences the overall robustness of the solver. Taking all these constraints into account, we have developed a GPU-based AMS that exploits the parallelism in every module of the solver, including both the setup phase and the solution phase. The performance of AMS—with our carefully optimized algorithmic choices on the GPU for both the setup phase and the solution phase, is demonstrated using highly heterogeneous 3D problems derived from the SPE10 Benchmark (Christie et al. 2001). Those problems range in size from millions to tens of millions of cells. The GPU implementation is benchmarked on a massively parallel architecture consisting of NVIDIA Kepler K80 GPU's, where its performance is compared to the multi-core CPU architecture using an optimized multi-core AMS implementation (Manea et al. 2016) running on a shared memory multi-core architecture consisting of two packages of Intel's Haswell-EP Xeon(R) CPU E5-2667. While the GPU-based AMS parallel implementation shows good scalability for the solution stage, its setup stage is less efficient compared to the CPU, mainly due to the dependence on a QR-based basis functions solver.

Introduction

Currently, with modern reservoir characterization techniques, highly detailed reservoir models can be constructed, in which the permeabilities of the underlying geological formations vary significantly due to the complex multiscale heterogeneities involved. In order to construct such models, one needs to integrate data and information from different sources at various scales including data obtained from core analysis, well logs, seismic images, and dynamic production monitoring tools. The resulting reservoir models can have

up to $O(10^7)$ to $O(10^8)$ grid cells, justifying the need to come up with efficient upscaling algorithms that can be used to convert these so-called geocellular models into reservoir simulation models (Zhou et al. 2012b).

Several multiscale methods have been developed and analyzed in the past (Hou and Wu 1997; Efendiev et al. 2000; Aarnes and Hou 2002; Arbogast et al. 2002; Chen and Hou 2003; Jenny et al. 2003; Aarnes 2004; Aarnes et al. 2005; Efendiev and Hou 2009; Manea et al. 2016). In these methods, the global fine-scale problem is divided into several local problems which are used to construct the underlying basis functions. These basis functions, which are numerical solutions of the local problems, are used to construct accurate coarse-scale quantities. In addition, they are also used to map the solution of the coarse-scale system to the fine-scale system, once the coarse scale system is solved. The ultimate goal of these methods is to resolve the fine-scale information without solving the global fine-scale problem explicitly.

As stated above, The history of such methods goes back to the work of (Hou and Wu 1997) in which a Multiscale Finite-Element Method (MSFE) was proposed. In this method, despite the fact that the fine-scale information are captured through the construction of special basis functions, the fine-scale velocity field is not mass conservative. To overcome this problem, The Multiscale Mixed Finite-Element Method (MSMFE) was introduced (Hou and Wu 1997; Arbogast et al. 2002; Chen and Hou 2003; Aarnes 2004) where the mass is locally conserved. Another proposed multiscale method which is locally mass-conservative is the Multiscale Finite Volume (MSFV) method (Jenny et al. 2003). In this method, the coarse-scale system is constructed using locally computed basis functions in a finite-volume framework, and the fine-scale conservative velocity field is obtained by solving additional local Neumann problems over the primal coarse control volumes (Wang et al. 2014). This method was extended and applied to a wide range of problems including models incorporating gravity and capillary pressure (Lunati and Jenny 2008), the effects of compressibility (Hajibeygi and Jenny 2009; Zhou and Tchelepi 2008), faults and fractures (Hajibeygi et al. 2011a,b), three-phase flow and compositional displacements Tchelepi et al. (2008); Hajibeygi and Tchelepi (2014), and complex wells Wolfsteiner et al. (2006); Jenny and Lunati (2009). Moreover, adaptive computations of the basis functions were introduced in the work of Jenny et al. (2004, 2006); Zhou et al. (2012a); Hajibeygi and Jenny (2011) to enhance the efficiency of the method for problems involving time-dependent multiphase models.

It should be noted that for a wide range of heterogeneous porous media problems, the existing multiscale methods result in solutions with accuracies comparable to the accuracies of the reference fine-scale solutions. However, for problems involving channelized permeability fields or highly anisotropic properties, the accuracy of the traditional so-called (single pass) multiscale methods degrades, as a result of the localization assumption used to construct the basis functions (Zhou et al. 2012b). To overcome these difficulties, the iterative multiscale finite-volume (i-MSFV) method was introduced in the work (Hajibeygi et al. 2008) in which locally computed Correction Functions (CF) are used to reduce the induced MSFV errors in a systematic way. However, this method suffers from weak convergence issues for anisotropic and highly heterogeneous problems (Lunati et al. 2011). To overcome this issue, the MSFE operator was introduced (Bonfigli and Jenny 2009; Zhou et al. 2012b) resulting in faster convergence rate, and mass-conservative solutions.

In practice, the iterative and single pass multiscale methods are formulated algebraically in order to reduce the complexity of implementation, and allow for easy and efficient integration of the method into existing legacy reservoir simulators (Wang et al. 2014). Specifically, a two-stage formulation of the iterative multiscale (MS) method was assumed, known as the Two Stage Algebraic MS (TAMS) solver (Zhou et al. 2012b) or the Algebraic Multiscale (AMS) solver (Wang et al. 2014). In both (TAMS, and AMS), the first stage involves a single-pass of the original MS solver, and the second stage involves the application of a local preconditioner. The first stage is called the *global-stage* and based on the spectral analysis shown in Hajibeygi et al. (2008); Zhou et al. (2012b), it works on damping the low-frequency error modes related to the long-range coupling of the variables. On the other hand, the second stage is called the *local-stage*, and

works on damping the high frequency error modes related to the short-range coupling of the variables. We note here that any variant of the original MS method can be used as the underlying method for the global stage. However, it was shown in Zhou et al. (2012b); Wang et al. (2014) that the Multiscale Finite-Element Method (MSFE) is the most efficient choice of the global stage numerically and computationally. The MSFV method can be also used in the last iteration of the iterative solver to maintain a mass-conservative velocity field of the final result. Finally, AMS has been demonstrated as an efficient and highly scalable solver on shared-memory multi-core environments (Manea et al. 2016, 2017)

In this work, we investigate the efficiency and parallel scalability of the Algebraic Multiscale Solver (AMS) (Wang et al. 2014) for solving the pressure equation of an incompressible flow system in highly heterogeneous porous media on the GPU massively parallel architecture. The scalability of the GPU-based solver is compared against its carefully optimized implementation on the shared-memory multi-core architecture, using different highly heterogeneous models obtained from the SPE10 Benchmark (Christie et al. 2001). Despite the fact that several components of the AMS method are embarrassingly parallel, the underlying algorithmic details and data-structures play a significant role on leveraging the scalability of the method on the GPU architecture. Moreover, the underlying considered algorithm for each step has a significant impact on the overall robustness of the solver. Considering all of the above factors, this paper tries to investigate the optimal algorithmic design of the AMS method on GPU's. To the best of our knowledge, this is the first time AMS is demonstrated on the GPU architecture.

The paper is organized as follows: Section (2) provides a brief background on multiscale methods and their applications in solving flow in porous media equations. Our AMS implementation approach on the GPU architecture is presented in Section (3). Numerical results are shown in Section (4), followed by the recommendations and conclusions in Section (5).

Background

In reservoir simulation, a considerable amount of time and computational efforts are spent on solving a variation of the following elliptic partial differential equation representing the pressure behavior in an incompressible singe phase flow model:

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

In this equation, p represents the pressure over a computational domain Ω, λ is a positive-definite mobility tensor, defined as the ratio of the porous medium permeability to the fluid viscosity, and q is a source or sink term. Despite its simplicity, this equation involves highly discontinuous coefficients in heterogeneous media (such as the permeability tensor which includes values ranging by several orders of magnitudes). This results in a badly conditioned discretized system, requiring clever preconditioners which can resolve the different discontinuities encountered in the underlying PDE coefficients. Now, using the standard two-point flux approximation scheme to discretize equation 1 on the fine grid, denoted by Ω_f, the following algebraic system of equation is obtained:

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

Here, p^f denotes the discretized pressure unknowns on the fine-grid domain, Ω^f. The Multiscale Method solves the fine-scale system as follows: first the whole fine-grid domain is subdivided into N_c subdomains Ω_k^c, known as the *primal* coarse-grid blocks, where $1 \leq k \leq N_c$. The centers of the primal grid blocks are joined together to form the *dual* coarse grid, Ω_m^d, consisting of N_d grid blocks where $1 \leq m \leq N_d$. Both the primal and dual coarse grids are shown in Figure 1. An important related measure is called the *coarsening ratio*, C_r, and is defined as $C_r = \frac{N_f}{N_c}$. Once the primal and dual grids are constructed, Equation 1 is solved locally on each coarse grid block Ω_m^d with reduced boundary conditions to obtain the basis functions. In this process, fluxes normal to the dual grid boundary are ignored (localization assumption), resulting in (*n*

- 1)*D*-problem on each dual coarse-grid block edge or face, where *n* denotes the topological dimensions of the original problem. From an algebraic point of view, the basis functions are obtained by solving the following system (Wang et al. 2014):

$$\begin{aligned} \nabla \cdot (\lambda \cdot \nabla \phi_j^i) &= 0, && \in \Omega_j^d \\ \nabla_{\parallel} \cdot (\lambda \cdot \nabla \phi_j^i)_{\parallel} &= 0, && \in \partial\Omega_j^d \\ \phi_j^i(x_k) &= \delta_{ik}, && \forall x_k \in \{1, ..., N_c\} \end{aligned} \tag{3}$$

Here, each ϕ_j^i represents the basis function corresponding to coarse node *i* in the dual coarse-grid block Ω_j^d. In addition, the notation $\parallel$ refers to the tangential projection of the operator or vector on the boundary of the dual coarse-grid block $\partial\Omega_j^d$. This projection operation ignores the contributions of the normal fluxes, and considered the primary source of errors in multiscale methods. Using the principle of superposition, we can write:

$$\tilde{p}^f = \sum_{j=1}^{N_d} \sum_{i=1}^{N_c} \phi_j^i p_i^c = \mathcal{P} p^c, \tag{4}$$

where $\tilde{p}^f$ denotes the fine-scale solution (approximate fine-scale pressure solution), p^c denotes the coarse-scale pressure solution, and $\mathcal{P} \in \mathbb{R}^{N_f \times N_c}$ denotes the prolongation operator (mapping coarse-scale solutions to fine-scale solutions). It should be noted here that the restriction operation denoted by $\mathcal{R} \in \mathbb{R}^{N_c \times N_f}$ is used to map the fine-scale solution to the coarse scale one.

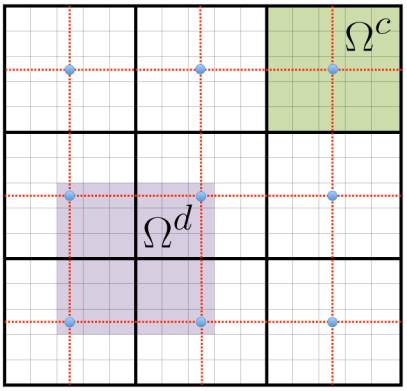

Figure 1—2D Multiscale Grid. The original fine grid is shown in solid thin lines. The primal coarse grid Ω^c is shown in thick solid black lines, and the dual coarse grid Ω^d is shown in dashed red lines. The coarse points (vertices) are shown in blue circles. (Image Courtesy of (Manea et al. 2016))

Multiscale methods differ in the way the restriction operator $\Re$ is defined, which in turn, determines the corresponding coarse scale operator $A^c \in \mathbb{R}^{N_c \times N_c}$. In the MSFE method, the Galerkin definition (Smith et al. 1998; Toselli and Widlund 2005; Tchelepi and Wallis 2010) is used as follows:

$$\mathcal{R} = \mathcal{P}^{\mathcal{T}}. \tag{5}$$

Based on that, the global coarse-scale system can be written as:

$$\mathcal{R} A^f \mathcal{P} p^c = \mathcal{R} q^f, \tag{6}$$

and the coarse-scale operator, A^c, is defined as:

$$A^c = \mathcal{R} A^f \mathcal{P}. \tag{7}$$

We note here that the coarse-scale system is smaller, and less expensive to solve compared to the original fine-scale system, which is the main computational advantage of the multiscale method. The single-pass fine-scale solution and the global-stage preconditioning operator are given by:

$$\tilde{p}^f = \mathcal{P}\left(A^c\right)^{-1} q^c = \mathcal{P}\left(\mathcal{R}A^f\mathcal{P}\right)^{-1}\mathcal{R}q^f, \tag{8}$$

$$M_g^{-1} = \mathcal{P}\left(\mathcal{R}A^f\mathcal{P}\right)^{-1}\mathcal{R}. \tag{9}$$

respectively. In linear operations, the single-pass solution *tildep*^{f} is constructed as follows. First, the dual coarse grid is used to subdivide the fine-grid into three classes: interior cells p_I, edge cells p_E, and vertex cells p_V, using a "wirebasket" permutation operator (Smith et al. 1998; Tchelepi and Wallis 2010) acting on A^f (the fine-scale linear operator). The resulting permuted system is shown below:

$$\begin{bmatrix} A_{II} & A_{IE} & 0 \\ A_{EI} & A_{EE} & A_{EV} \\ 0 & A_{VE} & A_{VV} \end{bmatrix} \begin{bmatrix} p_I \\ p_E \\ p_V \end{bmatrix} = \begin{bmatrix} q_I \\ q_E \\ q_V \end{bmatrix}. \tag{10}$$

The reduced boundary condition assumption allows us to eliminate the blocks A_{EI} (the influence of interior cells on edge cells), and A_{VE} (the influence of edge cells on vertex cells), and the corresponding right hand side vectors q_I and q_E (Zhou et al. 2012b). In addition, A_{VV} is replaced with the MSFE coarse-operator A^c, q^V, with $q^c = \Re q^f$, and p^V with p^c. This results in a block upper- triangular system given as follows:

$$\begin{bmatrix} A_{II} & A_{IE} & 0 \\ 0 & \tilde{A}_{EE} & A_{EV} \\ 0 & 0 & A^c \end{bmatrix} \begin{bmatrix} p_I \\ p_E \\ p^c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ q^c \end{bmatrix}. \tag{11}$$

The coarse-scale system appears in the last block in the system above, and $\tilde{A}_{EE}$ is obtained by adding the contribution of A_{EI} into A_{EE}. When the coarse system is solved, p_E and p_I are obtained as follows:

$$p_E = -\tilde{A}_{EE}^{-1} A_{EV} p^c, \tag{12}$$

$$p_I = -A_{II}^{-1} A_{IE} p_E = A_{II}^{-1} A_{IE} \tilde{A}_{EE}^{-1} A_{EV} p^c. \tag{13}$$

Therefore, the prolongation operator, P can be defined as follows:

$$p^f = W \begin{bmatrix} p_I \\ p_E \\ p^c \end{bmatrix} = W \begin{bmatrix} A_{II}^{-1} A_{IE} \tilde{A}_{EE}^{-1} A_{EV} \\ -\tilde{A}_{EE}^{-1} A_{EV} \\ I_{VV} \end{bmatrix} p^c = \mathcal{P} p^c. \tag{14}$$

In the above, $I_{VV} \in \mathbb{R}^{N_c \times N_c}$ is the identity matrix, and W is the wirebasket permutation matrix. Finally, the two-stage algebraic multiscale solver (TAMS preconditioning operator (Zhou et al. 2012b; Wang et al. 2014) is given by:

$$M_{TAMS}^{-1} = M_g^{-1} + M_l^{-1} - M_l^{-1} A^f M_g^{-1}. \tag{15}$$

We note that the selection of local-stage preconditioner M_l^{-1} is a design choice, and has an effect on the scalability and robustness of the method.

GPU-Based AMS

As stated above, the AMS method consists of two main steps: the *setup* phase and the *solution* phase. These steps are summarized as follows:

1. SETUP PHASE:
    a. Set up the primal and dual MSFE Grids.

b. **Basis Functions Computation:** Compute basis functions, ϕ_j^i.
c. Assemble the restriction ($\mathfrak{R}$) and prolongation (P) operators.
d. **Coarse-Scale System Construction:** Form the coarse-scale system: $A^c = \mathfrak{R}A^f P$.
e. **Coarse-Scale System Factorization:** Factor the coarse-scale system: $A^c = L^c U^c$.
f. Setup the local stage preconditioner, M_l^{-1}.

2. SOLUTION PHASE:

a. *Global Stage:*

i. **Restriction Operation:** Restrict the fine-scale RHS $q^c = \mathfrak{R}q^f$.
ii. **Coarse-Scale System Solution:** Solve the coarse-scale system $p^c = (A^c)^{-1}\, q^c$.
iii. **Prolongation Operation:** Compute the fine-scale solution $p^f = Pp^c$.

b. *Local Stage:*

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

The parallel scalability and robustness of the MS method on GPU-architectures depend on the specific choices of the underlying key computational kernels (shown in boldface above). Inthis section, we describe the design approach taken to handle each of these kernels in our GPU implementation.

Basis Functions Computation

Computing the basis functions represents the bulk of the setup phase of AMS, as it involves factorizing a large number of small independent systems with multiple righthand sides. Thus, a careful design of this kernel is very important to the overall AMS scalability on GPU's. While the factorization step can be performed using either direct or iterative algorithms, it was shown in Manea et al. (2016) that direct methods are computationally more efficient for such a task. Thus, designing a scalable basis function computation kernel on the GPU relies on using a scalable direct solver on the GPU. While the amount of parallel work in a small LU can be limited, the large number of independent linear systems for the basis functions provides another opportunity for extracting parallelism. However, as the GPU is a single-instruction-multiple-data (SIMD) architecture, where a regular control flow is very important for achieving parallel efficiency, these systems need to be pre-processed for the GPU to handle them efficiently.

In our approach, dual-domains sharing the same size are grouped and factored together (in batches, as shown in Figures 2 and 3). Ideally, each batch should be factored once, then solved several times with different right hand side vectors. Since developing a GPU-based direct solver is outside the scope ofthis work, we made use of the batched QR functionality (cusolverSpDcsrqrsvBatched) provided in NVIDIA's cuSOLVER Library NVIDIA Corporation (2018), as it was the best available choice at hand when writing this paper. However, it is worth to note that this choice for the computing basis functions has two scalability limitations. First, QR factorization is a relatively expensive solution method compared to regular LU algorithms, as it requires twice the number of numerical operations. Second, the interface provided for the batched QR operations does not separate the numerical factorization phase from the solution phase, which results in repeating the numerical factorization for every new right hand side.

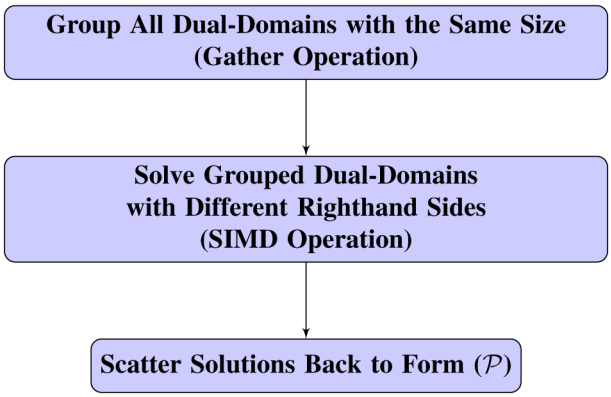

Figure 2—Computing Basis Functions on the GPU Architecture

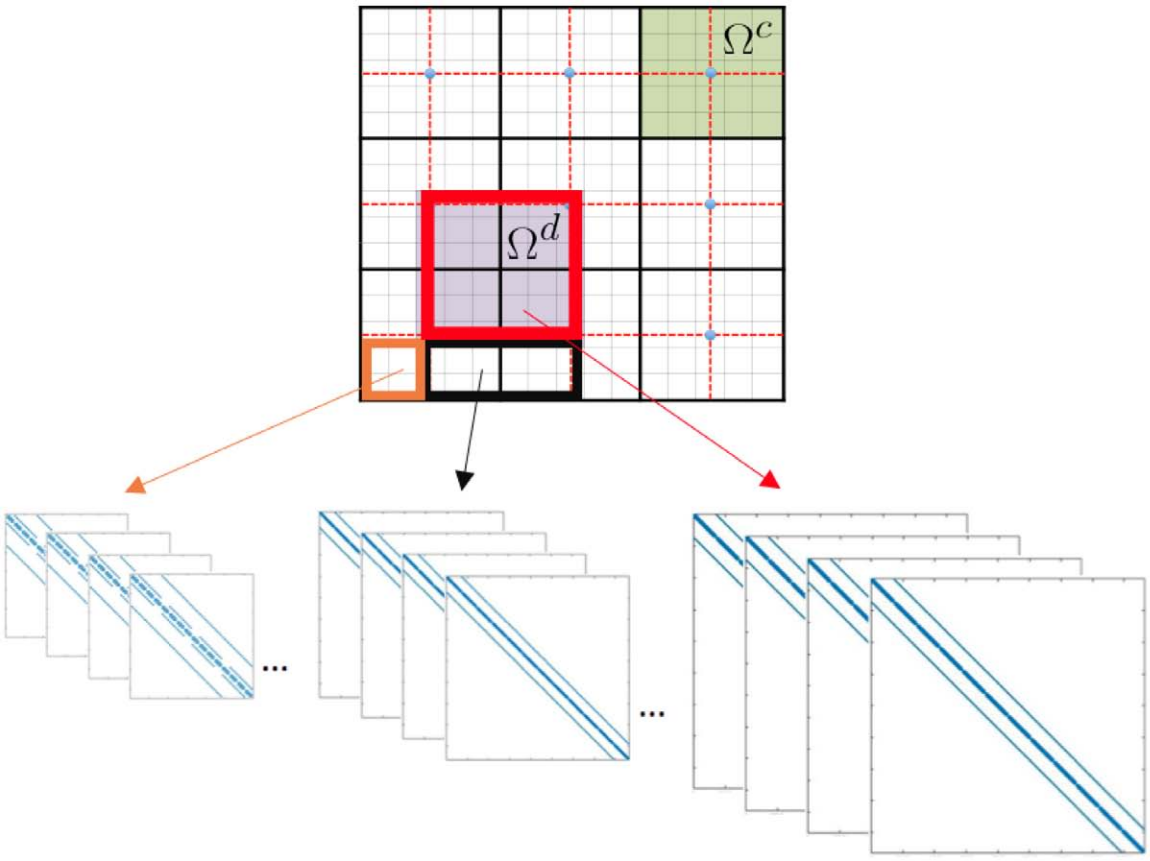

Figure 3—Grouping Dual Domains Sharing The Same Size into Batches

Coarse-Scale System Construction

The construction of the coarse scale system represent a triplet sparse matrix-matrix operation. We follow a similar approach to the approach taken in Manea et al. (2016), where the coarse-scale system is formed by a direct application of A^f onto P, then applying $\Re$ on the $A^f P$ product. However, for this computation to be split between GPU threads, we padded the resultant A^c with physical values of zero entries in all trivial zero entries falling on A^c's 27 bands. This is needed to generate a Compressed-Sparse-Row (CSR) matrix representation of A^c in parallel on the GPU without the need to synchronize between GPU threads.

Coarse-Scale System Factorization

The choice of the algorithm used to factorize A^c is important to the scalability of AMS, as it affects both the setup stage and the solution stage. As it is hard to extract enough parallel work from a direct sparse solvers on the GPU (in both the factorization phase during the setup stage and the forward-backward solution phase during the solver stage), we have chosen to use a GPU-based Algebraic Multigrid Solver to handle the coarse-scale system, namely AmgX package from NVIDIA NVIDIA Corporation (2014).

Solution Phase

All the solution phase kernels, i.e, the **restriction**, **prolongation**, **local stage smoothing** and **coarse-scale system solution** (when using AMG as a coarse-scale solver), mainly involve sparse matrix-vector multiplications (a memory-bound operations), where the design should choose data-structures that would allow for taking full advantage of the high memory bandwidth on the GPU. Thus, we used the diagonal-matrix format (DIA) to store the system matrix, and dense vectors to store **restriction** and **prolongation**, where the data-access patterns coalesce memory accesses on the GPU when performing these operations.

Numerical Experiments

In this section, the performance and scalability of our GPU-based AMS implementation is demonstrated and compared against a carefully optimized multicore shared-memory AMS implementation (Manea et al. 2016), using various heterogenous cases derived from the SPE10 Benchmark (Christie et al. 2001). The details of our numerical experiments setup is described first. Following that, the results are presented and analyzed.

Test Cases Design

To demonstrate the performance of AMS on the GPU architecture, four 3D test cases were used. All the test cases are derived from the SPE10 Problem (Christie et al. 2001) by piecewise constant grid refinement. These test cases are varied in sizes, from a relatively small sizes to the largest size that could be fitted on a single GPU, as follows: 100^3 (1M Case), 200^2 (8M Case), 300^3 (27M Case), and 3453 (41M Case).

Problem settings. All test cases impose a Dirichlet-type boundary conditions on two opposite faces of the domain, while other faces are set to no-flow Neumann boundary condition. Richardson iteration was used in all test cases to examine the behavior of AMS without any aid from Krylov stabilization. The problems are solved until five orders of magnitudes are reduced from the l_2 norm of the initial residual (i.e. $\frac{\|r_k\|_2}{\|r_0\|_2} \leq 10^{-5}$). AMS is setup with MSFE as the multiscale (global stage) preconditioner, and the damped point red-black Gauss-Seidel (Golub and Van Loan 2012) is used as the local-stage smoothing preconditioner. The AMS parameters are listed in Table 1. For the coarse solver, AmgX was used configured with a V(2,2) cycle, a Parallel Maximal Independent Set (PMIS) coarsening, a distance-2 (D2) interpolator and a Jacobi smoother.

Table 1—Parameter Settings for AMS

Parameter	Value
Local-Stage Preconditioner (M_l^{-1})	point red-black Gauss-Seidel
Number of smoothing sweeps (n_s)	20
C_r	5 × 5 × 5
ω	1.8

Computer platforms. For testing AMS on the GPU platform, an NVIDIA Kepler K80 GPU is used. The shared memory AMS version is tested on a multi-core architecture consisting of two packages of Intel® Haswell-EP Xeon® CPU E5-2667.

Results and Analysis

The results for benchmarking the GPU-based AMS against the optimized multi-core shared-memory version, using the problem settings outlined above, are shown in Figure 4–7. Figure 4 shows the total time each version of the solver spent for each problem size. The time is broken into both setup stage and solution stage. As apparent from the trend in the figure, the relative GPU to CPU performance improves as the problem size gets larger, as the GPU version shows better weak scalability since more computational work is available to 'saturate' the GPU resources. However, this trend does not hold when broken down into setup and solution time, as the GPU version setup stage is evidently less scalable than the CPU version of the solver. And on the contrary is the solution stage, where the GPU-based AMS is almost two times faster than its CPU implementation.

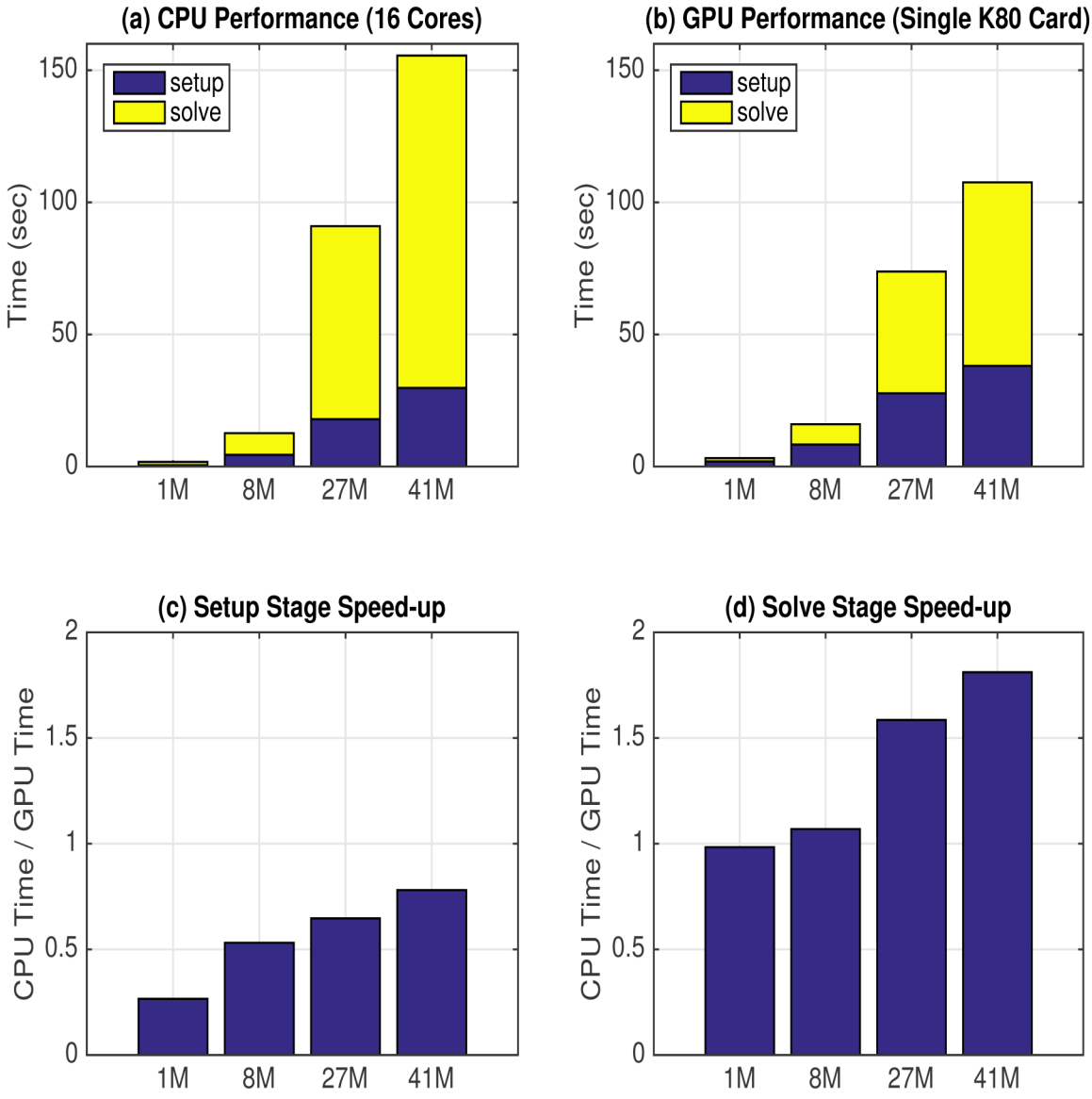

Figure 4—The total time for solving all cases on (a) the CPU and (b) the GPU, along with (c) setup stage GPU speed up and (d) solve stage GPU speed up.

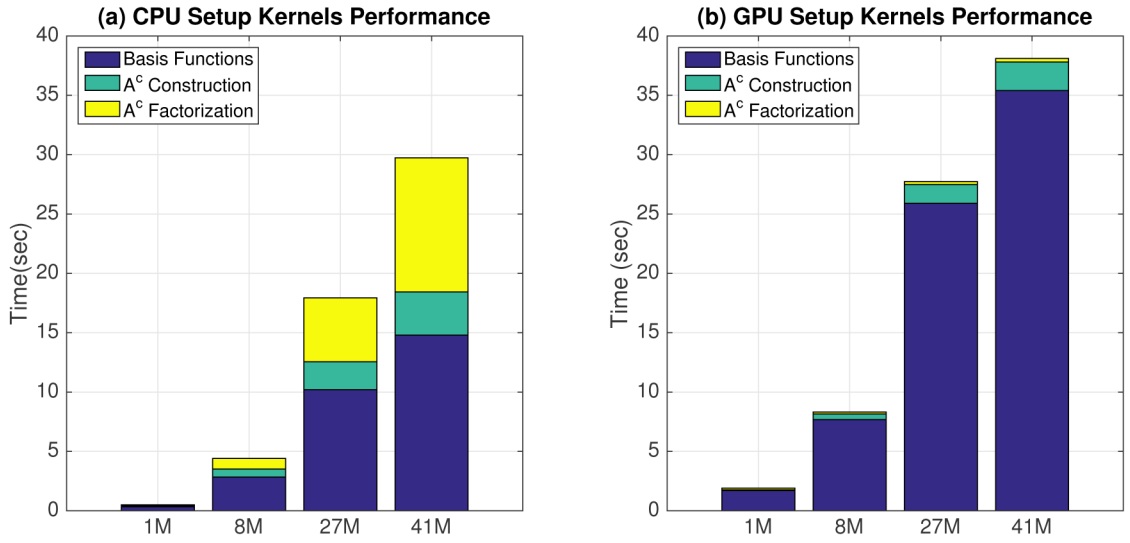

Figure 5—The setup stage time for all cases on (a) the CPU and (b) the GPU.

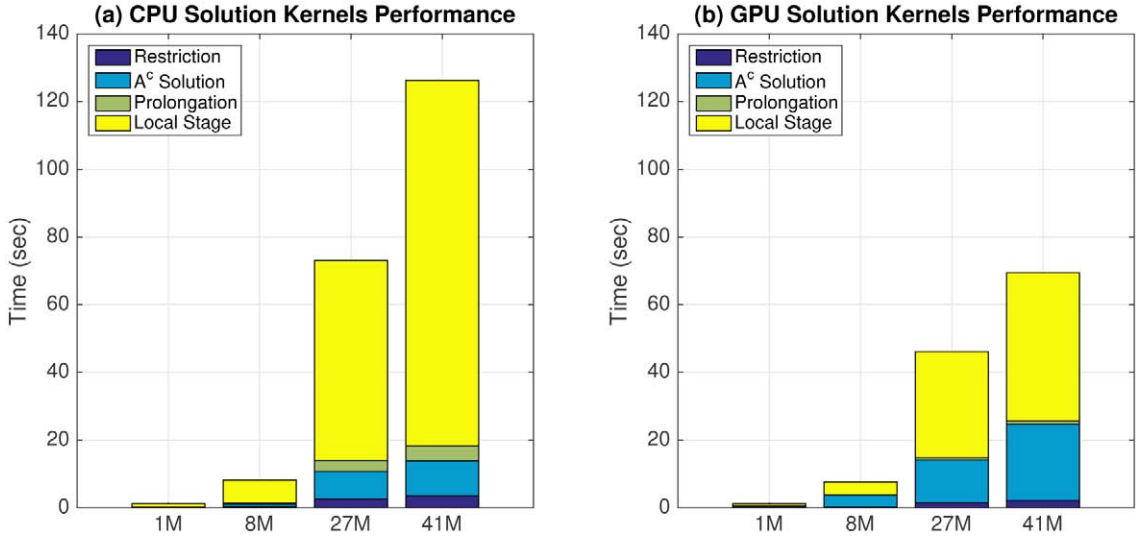

Figure 6—The solution stage time for all cases on (a) the CPU and (b) the GPU.

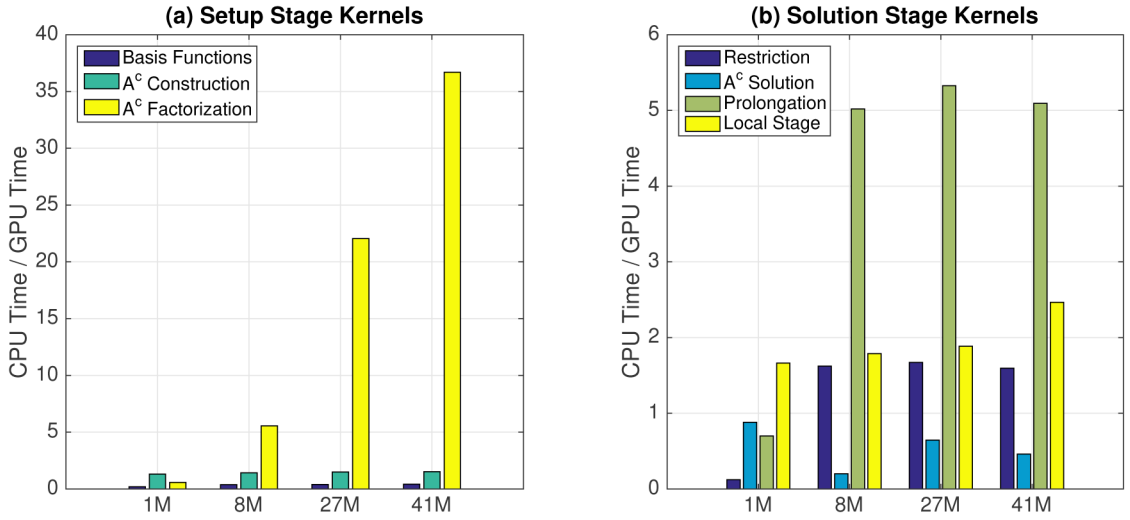

Figure 7—The GPU-based AMS speedup over the 16-core parallel implementation on the CPU for (a) the setup kernels and (b) the solution kernels.

To understand this behavior between the two versions of the solver better, Figures 5 and 6 shows the relative performance between the two solvers broken into individual setup and solution kernels, respectively. As is evident from the results shown in Figures 5 (b), the basis functions computation kernel is the major scalability limitation in our GPU-based AMS implementation. This is expected given the fact that we compute the basis functions using a QR factorization algorithm, which requires at least twice the number of operations compared to LU factorization used by the CPU implementation. Moreover, the QR numerical factorization is repeated for all the 8 different right hand sides of each dual domain system, resulting in doing 7 times more work than is essentially needed. When these factors are put together, the AMS GPU version is indeed doing at least 14 times more work that its CPU version. Yet, the GPU basis functions kernel is only 2 times slower than the CPU optimized kernel.

Aside from the basis functions computation kernel, the coarse-scale factorization is more efficient on the GPU than the CPU. This can be, in a major part, attributed to using an AMG algorithm with a scalable setup phase to handle the coarse-scale system on the GPU. AMG theoratically needs only $O(N)$ operations for both the setup and solve stages Stüben (2001), compared to $O(N^3)$ for LU factorization. In addition, the construction of the coarse-scale system is also more efficient on the GPU.

For the solution stage kernels, the GPU-based AMS is clearly more efficient that the CPU-based version. The main gain in the GPU performance is evidently from the memory-bound local stage, where our choice of the diagonal matrix format have clearly given a good utilization of the high-bandwidth GPU memory. This relative GPU-CPU performance gain gets more pronounced as we add more contention on the system memory by increasing the problem size. Moreover, the memory-bound prolongation and restriction kernels also show superior performance on the GPU compared to their CPU versions. Finally, the coarse-scale system solution on the GPU is evidently less optimal that the CPU version, and again this is majorly due to the fact that the coarse-scale system on our GPU implementation is handled using AMG, where multiple multigrid cycles are needed per one coarse-scale solution step compared to a single forward-backward sweep in the CPU version.

Conclusions

In this work, we have investigated the scalability of AMS on the massively parallel the GPU architecture. While several kernels in the AMS algorithm are amenable to parallelization, mapping these kernels on the emerging massively parallel GPU architecture requires adapting the data- and control-flow of the algorithm to minimize control divergence and increase memory bandwidth utilization. Taking all these constraints into account, we have developed a GPU-based AMS that exploits the parallelism in every module of the solver, including both the setup phase and the solution phase. The performance of AMS on GPU is demonstrated using highly heterogeneous 3D problems derived from the SPE10 Benchmark (Christie et al. 2001). Those problems range in size from millions to tens of millions of cells. The GPU implementation is benchmarked on a massively parallel architecture consisting of NVIDIA Kepler K80 GPU's, where its performance is compared to the multi-core CPU architecture using an optimized multi-core AMS implementation (Manea et al. 2016) running on a shared memory multi-core architecture consisting of two packages of Intel's Haswell-EP Xeon(R) CPU E5-2667. While the GPU-based AMS parallel implementation shows good scalability for the solution stage, its setup stage is less efficient compared to the CPU, mainly due to the dependence on a QR-based basis functions solver. These results suggests that the most scalable AMS setup would be hybrid CPU-GPU design, where the setup phase can be carried out on the CPU, while the solution phase is executed out on the GPU. We are currently working on a detailed analysis and benchmarking of such a hybrid implementation. Another future research direction we are interested in is developing a specialized efficient basis functions solver on the GPU architecture.

Acknowledgements

The authors would like to thank Saudi Aramco management for giving them permissions to publish this work.

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

G. Bonfigli and P. Jenny. Recent Developments in the Multi-Scale-Finite-Volume Procedure. In *Large-Scale Scientific Computing*, pages 124–131, 2009.

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. ISSN0025-5718. 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.

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 ofa 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. H. Golub and C. F. Van Loan. *Matrix computations*, volume **3**. JHU Press, 2012.

H. Hajibeygi and P. Jenny. Multiscale finite-volume method for parabolic problems arising from compressible multiphase flow in porous media. *Journalof Computational Physics*, **228**(14):5129–5147, 2009. ISSN0021-9991. DOI: https://doi.org/10.1016/j.jcp.2009.04.017. URL http://www.sciencedirect.com/science/article/pii/S0021999109001946.

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

H. Hajibeygi and H. A. Tchelepi. Compositional multiscale finite-volume formulation. *SPE Journal*, **19**(2), 2014.

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

H. Hajibeygi, R. Deb, and P. Jenny. Multiscale finite volume method for non-conformal coarse grids arising from faulted porous media. In *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers, 2011a.

H. Hajibeygi, D. Karvounis, and P. Jenny. A hierarchical fracture model for the iterative multiscale finite volume method. *Journal of Computational Physics*, **230**(24):8729–8743, 2011b. ISSN 0021-9991. DOI: https://doi.Org/10.1016/j.jcp.2011.08.021. URL http://www.sciencedirect.com/science/article/pii/S0021999111005080.

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.

P. Jenny and I. Lunati. Modeling complex wells with the multi-scale finite-volume method. *Journal of Computational Physics*, **228**(3): 687 – 702, 2009. ISSN 0021-9991.

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. ISSN0021-9991. DOI: 10.1016/S0021-9991(03)00075-5. URL http://dx.doi.org/10.1016/S0021-9991(03)00075-5.

P. Jenny, S. H. Lee, and H. A. Tchelepi. Adaptive Multiscale Finite-Volume Method for Multiphase Flow and Transport in Porous Media. *Multiscale Modeling & Simulation*, **3**(1):50–64, 2004.

P. Jenny, S. Lee, and H. Tchelepi. Adaptive fully implicit multi-scale finite-volume method for multi-phase flow and transport in heterogeneous porous media. *Journal of Computational Physics*, **217**(2):627–641, 2006. ISSN 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2006.01.028. URL http://www.sciencedirect.com/science/article/pii/S002199910600026X.

I. Lunati and P. Jenny. Multiscale finite-volume method for density-driven flow in porous media. *Computational Geosciences*, **12**(13): 337–350, 2008.

I. Lunati, M. Tyagi, and S. H. Lee. An iterative multiscale finite volume algorithm converging to the exact solution. *Journal of Computational Physics*, **230**(5):1849–1864, 2011.

A. Manea, J. Sewall, and Tchelepi. Parallel Multiscale Linear Solver for Highly Detailed Reservoir Models. *SPE Journal*, **21**, 2016.

A. Manea, H. Hajibeygi, P. Vassilevski, H. Tchelepi, et al. Parallel enriched algebraic multiscale solver. In *SPE Reservoir Simulation Conference*. Society of Petroleum Engineers, 2017.

NVIDIA Corporation. Amgx: Multi-grid accelerated linear solvers for industrial applications. https://developer.nvidia.com/amgx, 2014.

NVIDIA Corporation. CUSOLVER Library, Version 10.0. https://docs.nvidia.com/pdf/CUSOLVER_Library.pdf, October 2018.

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

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.

H. A. Tchelepi, C. Wolfsteiner, and S. H. Lee. Multiscale finite-volume formulation for multiphase flow in porous media: black oil formulation of compressible, three-phase flow with gravity. *Computational Geosciences*, **12**(13):351–366, 2008.

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

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.

C. Wolfsteiner, S. H. Lee, and H. A. Tchelepi. Well Modeling in the Multiscale Finite Volume Method for Subsurface Flow Simulationt. *Multiscale Modeling & Simulation*, **5**(3):900–917, 2006. ISSN 0036-1429.

H. Zhou and H. A. Tchelepi. Operator-Based Multiscale Method for Compressible Flow. *SPE Journal*, **13**(02), 2008.

H. Zhou, S. H. Lee, and H. A. Tchelepi. Multiscale finite-volume formulation for saturation equations. *SPE Journal*, **17**(1), 2012a.

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