Advances in Applied Mathematics and Mechanics
Adv. Appl. Math. Mech., Vol. **8**, No. 6, pp. 971-991

DOI: 10.4208/aamm.2015.m1138
December 2016

Toward Cost-Effective Reservoir Simulation Solvers on GPUs

Zheng Li[1,*], Shuhong Wu[2], Jinchao Xu[3] and Chensong Zhang[4]

[1] *Kunming University of Science and Technology, Kunming 650093, China*
[2] *Research Institute of Petroleum Exploration and Development, CNPC, Beijing 100083, China*
[3] *Department of Mathematics, Penn State University, University Park, PA 16802, USA*
[4] *LSEC & NCMIS, Academy of Mathematics and Systems Science, Beijing 100190, China*

Received 5 June 2015; Accepted (in revised version) 13 October 2015

Abstract. In this paper, we focus on graphical processing unit (GPU) and discuss how its architecture affects the choice of algorithm and implementation of fully-implicit petroleum reservoir simulation. In order to obtain satisfactory performance on new many-core architectures such as GPUs, the simulator developers must know a great deal on the specific hardware and spend a lot of time on fine tuning the code. Porting a large petroleum reservoir simulator to emerging hardware architectures is expensive and risky. We analyze major components of an in-house reservoir simulator and investigate how to port them to GPUs in a cost-effective way. Preliminary numerical experiments show that our GPU-based simulator is robust and effective. More importantly, these numerical results clearly identify the main bottlenecks to obtain ideal speedup on GPUs and possibly other many-core architectures.

AMS subject classifications: 65M10, 78A48

Key words: GPUs, reservoir simulation, fully-implicit method.

1 Introduction

Nowadays, computers are equipped with multicore CPUs and coprocessors which have tens or even thousands of light cores, such as Intel Many Integrated Core (MIC) coprocessors and graphic processing units (GPUs). This kind of heterogeneous architecture provides opportunities for development of more powerful and more energy efficient supercomputers. In the Top500 (spring 2015) list of supercomputers, there are 17 clusters

*Corresponding author.
Email: lizhxtu@126.com (Z. Li), wush@petrochina.com.cn (S. Wu), xu@math.psu.edu (J. Xu), zhangcs@lsec.cc.ac.cn (C. Zhang)

accelerated by Nvidia or AMD GPUs and 11 by Intel Xeon Phi (MICs) chips in the top 100 systems. Although, the GPUs still dominates over the MIC coprocessors in terms of number of systems in the Top500 list for the moment, it is still too early to tell which coprocessor(s) will eventually win the race to the exascale era and beyond.

Reservoir simulation plays an important role in designing efficient development processes and improving oil recovery ratio. High-resolution reservoir models can better characterize the reservoir heterogeneity and describe complex water encroachment [19, 38]. The demand for more accurate computer simulations has led to larger and, in turn, more heterogeneous, reservoir models. Such models entail larger and more difficult linear systems. For fully implicit reservoir simulation, the solution of Jacobian systems often accounts for the majority of simulation time. Hence simulation can be greatly accelerated by faster linear solvers. Reservoir property calculation and matrix assembling, which although accounts for the majority part in source code, is cheap in terms of computational cost and is embarrassingly parallelizable. Since the parallel scalability depends strongly on what algorithm is used in application, speedup itself should not be over emphasized. State-of-the-art sequential solvers often utilize multilevel and multistage algorithms that offer fast convergence, especially for large models. It is challenging to port these algorithms to many-core architectures without compromising the rate of convergence or robustness. Many simple preconditioning methods, such as weighted Jacobi, polynomial preconditioning and multi-color versions of the incomplete factorization methods, are naturally parallel and exhibit strong scalability. Exposing fine-grained parallelism often involves either a careful redesign of existing algorithms or developing novel algorithms in which sequential dependencies on data are reduced or eliminated [21]. In both cases, reusing the legacy code is very difficult if not completely impossible.

A lot of research effort has been devoted to creating fine-scale reservoir models and efficient reservoir simulators on high-performance computers; see [26] and references therein for details. Many researchers have investigated reservoir simulation on many-core architectures; most of them are on GPUs. Klie et al. [29] developed a GPU-based GMRES solver with multiple preconditioners and achieved an overall computational speedup of 2x compared to conventional CPU multicore simulations. Yu et al. [53] developed GPU-based block ILU preconditioners and showed a 6x speedup compared with their CPU implementation for the industry standard SPE10 benchmark [15]. Appleyard et al. [1] developed a massively parallel nested factorization for two-stage preconditioner in combination with a GMRES solver and also assembled matrix on GPU. An overall speedup factor of 10x was achieved using a GPU over execution on a single CPU core. Tchelepi and Zhou [44], based on the Appleyard's work, developed a massively parallel nested factorization preconditioner on multiple GPUs and achieved 19x speedup for double-precision computations compared to sequential CPU code for the SPE10 problem. Fung et al. [24] used a heterogeneous approach in which their simulator ran on CPUs while the linear solver ran automatically chosen device (CPU or GPU). Esler et al. [21] developed a GPU based reservoir simulator and they claim that it costs less than two minutes on one Nvidia K40 card to solve the full SPE10 benchmark.

GPUs have great computing power and large band-width, which has attracted many computation scientists and engineers. However, exploiting the increasing number of processing units without stretching the time needed to develop algorithms and software is very challenging. Difficulties in parallel programming, like load balancing, sequential dependencies and synchronization, now challenge application scientists, algorithm developers and software engineers. Moreover, hardware's life span is typically five years or less and an application or software is usually much more long-lived. This is especially the case for petroleum reservoir simulation, where a vast set of legacy code presents and a lot of effort has been devoted to make the existing code robust. It seems too expensive, or at least not cost-effective, to port the whole serial reservoir simulators to multi-core platforms.

In this paper, we investigate potential speed gains by porting a legacy serial simulator to a CPU-GPU heterogenous computer. We perform numerical experiments with the in-house simulator HiSim [52], which is an fully implicit reservoir simulator developed by the Research Institute of Petroleum Exploration and Development, CNPC. In the sequential version of HiSim, there are 150,000 lines of source code for the main simulator (not including pre/post processing or GUI) and more than 50,000 source lines for several linear algebraic solvers (from purely algebraic methods like ILU to several multi-stage ones like CPR). We plan to use this simulator as an example to investigate how much effort is needed to take full advantage of the computing power provided by new CPU-GPU architectures.

We plan to find a cost-effective road map toward efficient reservoir simulation on many-core architectures. Our numerical study is valuable for petroleum engineers before making implementation decisions. More specifically:

- We employ the widely-used True-IMPES decoupling and a CPR-AMG preconditioned GMRES solver for solving the Jacobian systems arising in the fully implicit method (FIM). We compare several matrix storage data structures for the coupled Jacobian systems and present a new hybrid storage format.
- We compare the performance of classical AMG and aggregation-type AMG methods on GPUs for solving pressure equations. Numerical results suggest that aggregation-type AMG methods, compared with the classical AMG, are not only cheaper in terms of AMG setup but also more efficient for typical pressure equations from FIM.
- We investigate numerically the convergence and scalability of CPR-AMG on GPUs. As numerical experiments indicates, we can easily obtain about 3.0x speedup for the linear solver part and 6.5x speedup for the SOLVE phase alone.

The rest of this paper is organized as follows: In Section 2, we will briefly review the mathematical model of the black-oil model, its fully implicit discretization, decoupling method and the CPR-AMG preconditioner. We then compare and analyze the scaling performance of standard Krylov iterative methods and ILU preconditioner in Section 3.

We compare the classical and the pair-wise aggregation AMG methods on GPU in Section 4. In Section 5, we design several numerical tests and show potential speed gains by using GPUs for reservoir simulation. And we draw some conclusions in Section 6.

2 Mathematical models and fully implicit methods

2.1 The black-oil model

In the black oil model, we assume that the water phase does not exchange mass with the other phases and the liquid and gaseous phases exchange mass between them. Let P_α, S_α, B_α, $k_{r\alpha}$, ρ_α and μ_α denote the pressure, saturation, formation volume factor, relative permeability, density and viscosity of phase α, respectively. ϕ is the porosity, κ is the static permeability tensor, g is the acceleration due to gravity, Z is the depth and R_s is the solution gas-oil ratio. Under isothermal condition, the black oil model are composed of three mass conservation equations for oil, water and gas components [13]. For brevity, we denote them by subscripts o, w and g, respectively.

$$\frac{\partial}{\partial t}\Big(\frac{\phi S_o}{B_o}\Big)=\nabla\cdot\Big(\frac{1}{B_o}u_o\Big)+q_o, \tag{2.1a}$$

$$\frac{\partial}{\partial t}\Big(\frac{\phi S_w}{B_w}\Big)=\nabla\cdot\Big(\frac{1}{B_w}u_w\Big)+q_w, \tag{2.1b}$$

and

$$\frac{\partial}{\partial t}\Big[\phi\Big(\frac{S_g}{B_g}+\frac{R_sS_o}{B_o}\Big)\Big]=\nabla\cdot\Big(\frac{1}{B_g}u_g+\frac{R_s}{B_o}u_o\Big)+q_g. \tag{2.2}$$

The Darcy velocity of phase α in the porous media can be written as

$$u_\alpha=\frac{\kappa\kappa_{r\alpha}}{\mu_\alpha}(\nabla p_\alpha-\rho_\alpha g\nabla Z),\quad \alpha=o,w,g,$$

and phase saturations satisfy:

$$S_o+S_w+S_g=1.$$

The source/sink terms q_α denote the net rate of well production injection of phase α. We use the well-known Peaceman model [39] for the volumetric well rates at the standard condition.

2.2 Fully implicit methods

Among many possible discretization methods for the above model, we only consider the standard fully implicit method (FIM) [20]. In FIM, Newton linearization is combined with first-order upstream-weighting finite difference spatial discretization [13]. This method is robust and stable. But it has a major disadvantage, i.e., the computational cost associated with solving the Jacobian systems arising from Newton's linearization is high. Very often, solving such linear systems takes more than 80% of the computation time in reservoir simulation.

We choose P_o and S_w as the primary variables for each cell in two-phase (water-oil) problems and we use P_o, S_w and S_g (or R_s) in the saturated (undersaturated) states as the primary variables for three-phase problems. When FIM is combined with cell-center finite volume or finite difference methods, a fully coupled linear algebraic system

$$Ax=b, \quad \text{i.e.,} \begin{pmatrix} A_{RR} & A_{RW} \\ A_{WR} & A_{WW} \end{pmatrix} \begin{pmatrix} x_R \\ x_W \end{pmatrix} = \begin{pmatrix} b_R \\ b_W \end{pmatrix}, \tag{2.3}$$

needs to be solved in each Newton step. Where the subscripts "R" and "W" stand for the reservoir and implicit well parts, respectively, of the main solution variables. A_{RR} is the derivatives of the reservoir equations with respect to the reservoir variables, A_{RW} is the derivatives of the reservoir equations with respective to the well variables, A_{WR} is the derivatives of the well equations with respect to the reservoir variables and A_{WW} is the derivative of the well equations with respect to the well variables, x_R refer to primary variables corresponding to reservoir and x_W is well variable (well bottom pressure p_{bh}). Let m be the number of unknowns in each grid cell. For example, in FIM for the black oil model, m is equal to 3; and m is equal to 2 for the dead oil case (no gas phase). Assume that there are N active grid cells and M implicit wells. Then the size of the Jacobian matrix is $mN+M$.

Remark 2.1. The coupling between the reservoir equations and the well constraints is usually strong. Based on this observation, we pad all the implicit well variables, by introducing artificial auxiliary saturation variables to each implicit well block, such that they have the same dimension as the reservoir blocks. At the same time, we pad the right-hand side with zeros at the corresponding positions of these artificial variables. Then the size of the expanded coefficient matrix A (with abuse of notation) is $m(N+M)$ and can be stored in a uniform block compressed sparse row (BSR) format. The details of this technique can be found in [52].

2.3 Linear solution methods

The pressure and saturation unknowns are strongly coupled in the Jacobian system (expanded Jacobian system). Many methods [3, 16, 30] have been proposed to weaken the coupling between pressure and saturations. The different decoupling methods will cause different effect to Jacobian systems. For the Alternate Block Factorization (ABF) method [3], the CPR method can not work well for the decoupled Jacobian system anymore. The preconditioner depends strongly on decoupled method used. We will address this else where. The most commonly used decoupling method in the CPR method is the so-called True-IMPES method [16], which operates Jacobian matrix to form an approximate pressure matrix. In the rest of this paper, only the decoupled Jacobian system will be considered.

Different parts of A have different algebraic properties due to the analytic nature of their corresponding continuous differential equations [45]. The equation corresponding to the pressure unknowns is elliptic and the equation corresponding to the saturation unknowns is hyperbolic. This makes the resulting Jacobian systems very challenging to solve. Moreover, the complicated geometric and physical properties of the reservoir, such as the anisotropy and heterogeneity of real reservoir media, make the Jacobian systems more difficult to solve. Based on this mixed nature of continuous partial differential equations, Wallis et al. [49, 50] have proposed a two-stage constrained pressure residual (CPR) preconditioner, which can be described in the following general form

$$M = \tilde{A}^{-1}[I - A\Pi^T \tilde{A}_p^{-1}\Pi] + \Pi^T \tilde{A}_p^{-1}\Pi.$$

Here $\tilde{A}_p^{-1}$ and $\tilde{A}^{-1}$ are approximate inverses of the pressure matrix A_p and the Jacobian matrix A, respectively. I is the identity matrix, $\Pi = (I_{N\times N}, 0)^T \in \mathbb{R}^{(mN+M)\times N}$ is the restriction matrix and Π^T denotes the prolongation.

In the first-stage, the pressure equations can be efficiently solved (approximately) by algebraic multigrid methods (AMG). In the second-stage, the block incomplete LU factorization (BILU) methods and the line successive over relaxation (LSOR) method are quite effective at resolving such system in practice [12]. In this paper, AMG and BILU(0) methods are applied to the first and second stage, respectively. This is the so-called CPR-AMG method, which can be naturally divided into two phases, SETUP and SOLVE. The SETUP phase is the part where AMG prepares the multilevel hierarchy and ILU decomposes the coefficient matrix into triangular matrices. SETUP is needed only once for each Newton iteration and sometimes multiple Newton iterations can even share a single SETUP step. The SOLVE phase is the actual preconditioned iterative method and it is performed in each linear iteration.

According to our numerical experiments (see Fig. 4 in Section 5.3), the CPR preconditioning accounts for as high as 70% in the SOLVE phase of linear solver. This means that, for an effective and relatively robust preconditioner like CPR-AMG, we should pay more attention to the implementation of preconditioning instead of just the iterative methods themselves (which accounts for less than 30% in our tests). In the next two sections, we shall discuss the implementation of both the iteration procedure and the preconditioner on GPUs.

3 Iterative methods and ILU on GPU

Thanks to their applicability and limited demands on memory, iterative methods are generally preferred over direct solvers for solving large-scale. The Krylov subspace methods (KSMs) [40], such as ORTHMIN, GMRES and BiCGstab, are efficient iterative methods for solving large sparse linear systems. The performance of KSMs depends on the efficiency of sparse matrix-vector multiplication (SpMV) implementation. Conventional implementations of SpMV have historically performed poorly, running at 10% or less of system peak performance on many uniprocessors, for two major reasons: (1) indirect and irregular memory accesses generally result in little spatial or temporal locality and (2) the speed of accessing index information in the data structure is limited by memory band-width [11]. Hence, SpMV is generally memory band-width-bounded instead of computation-bounded. How to make SpMV as efficient as possible for a specific architecture is still a challenging problem in computer science.

3.1 Block hybrid storage format

Exposing fine-grained parallelism and regularity of execution paths (as well as memory accesses) are vital to SpMV on GPUs. Both of them depend strongly on the sparse matrix storage format. Bell and Garland [5] implemented several traditional sparse storage formats on GPU, such as diagonal, COO, CSR and ELL formats and presented a efficient GPUs-suited Hybrid (ELL/COO) storage format. This format assures that data can be coalesced access by threads and achieve maximal band-width. Double-precision SpMV performance is more than 10x greater than that of a quad-core Intel Clovertown system. Based on the ELL format and block CSR format, Choi et al. [14] presented a block ELL format (BELL), this format not only can be coalesced, but exploits block structure to reduce row and column index and release burden on band-width. Experiments showed that SpMV based on BELL gains 1.8x and 1.5x for single and double-precision speedup over unblocked state-of-the-art implementations on GPUs. In addition, there are many other important aspects should be aware of in SpMV, like multithreads per row to expose sufficient parallelism to hide latency; reorder matrix to make load balance for each thread and etc. Interested readers are referred to [5, 18, 31].

Better performance of SpMV should be gained from the right combination of data structures and corresponding implementation that best exploit the GPU architecture. In fully implicit simulation of the black-oil model, the coefficient matrices are naturally in a block structure. Due to the presence of wells and faults, the band-width of Jacobian matrix will be affected by number of perforations and the number of non-neighboring connections (NNCs). Motivated by [5, 14], we present a block hybrid (BHYB) format and store the regular part of matrix in BELL format and irregular part (corresponding to well and fracture) in BCOO format in this paper. The BHYB format preserves the coalesced memory access pattern, avoids much padding introduced by ELL format and also reduces index set needed to store matrix. In our implementation, different levels of cache memory are used. Frequently reused parameters are cached to on-chip shared memory and vectors are binded to texture memory. And also, bitwise operations are used to index non-zero value for block data structure.

We compare the performance of SpMV based on the BHYB format with the cuSPARSE's HYB format in Fig. 1. The experiments show the speedup of the HYB and BHYB data structures on GPU. The comparison is based on the BSR performance on a single CPU core, where the BSR format is used in our reservoir simulator HiSim. The size of the test problems are Jacobian matrix arising in the fully implicit method for the

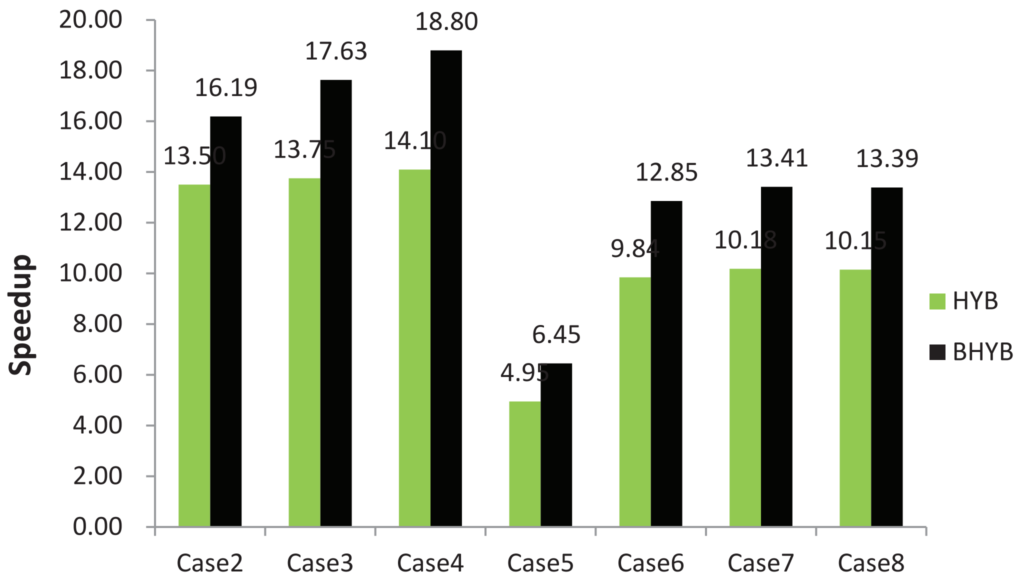

Figure 1: Performance of SpMV using BHYB and HYB on GPU compared with BSR on CPU.

black-oil test problems listed in Table 5. The problem size ranges from 0.2 million to 2.7 million.

Fig. 1 suggests that (1) for all test matrices, the BHYB format out-perform the HYB format in cuSPARSE (shows about 25% performance improvement); (2) speedup for two-phase problems (Case 2–4) are usually higher than those three-phase problems (Case 5–8); (3) for the same type of problems, the speedup for BHYB is higher when the problem size is larger (in contrast, the speedup in the HYB format is almost the same for different sizes). GPUs have great computing power and high memory band-width. Larger problem size means more active thread blocks and enough transactions in flight saturate the memory bus to hide latency.

Remark 3.1. The block data structures require less indexing than non-block formats. For memory-bounded operations, less indexing means less indirect memory accesses in the SpMV operation. This is why BHYB is faster than HYB. GPUs are throughput-oriented processors and are good at handling large data set. The speedup of BHYB matrices is also related to the block-size because larger block-size will introduce more paddings, which makes it less efficient.

Remark 3.2. So far it seems optimistic because we can get very good performance of SpMV on GPUs. Unfortunately, we must point out that the most expensive part in modern KSMs is usually not the iteration procedure itself. In fact, preconditioning takes much more time than iteration. We will see that the CPR-AMG preconditioner takes a large portion of the computation time and it is the main bottleneck when we port the simulator to many-core architectures; see Section 5.3 for details.

3.2 ILU on GPU

In the second stage of CPR preconditioner, ILU(0) is usually applied to full Jacobian systems in order to resolve high frequency of error. It is an important component of CPR. However, ILU is a sequential algorithm and has weak parallel scalability on GPUs. Therefore, many variant version of ILU, such as block ILU and multi-color ILU, are often used in order to scale well on many-core platform in reservoir simulation [29, 42, 53]. Also, Li and Saad [31] provided a few strategies on GPU implementation of ILUs. The main strategy is to form a disjoint partition of rows via algebraic coloring algorithm or METIS, reorder coefficient matrices to form blocks and then do ILU decomposition to each diagonal block. Actually, the BILU becomes more scalable as the number of blocks increasing. As reported in [31], large number of iterations are still needed and improved parallelism may be outweighed by increased number of iterations. As mentioned earlier, CPR preconditioner is the most time-consuming part in the linear solver and plays an important role in the convergence of GMRES. In this sense, it is justified to keep balance between scalability and robustness of the preconditioner. We employ the level scheduling strategy [40] to implement BILU(0) on GPUs in this paper.

4 Algebraic multigrid methods on GPU

The multigrid (MG) method is one of the most important advance in the area of numerical solution of discrete PDEs in the 20th century. Motivated by the observation that reasonable operator-dependent interpolation can often be derived directly from the underlying matrices, without any reference to the underlying grids, researchers found an automatic coarsening process in the early 1980s [7–9]. This is the algebraic multigrid (AMG) method, which can be considered as a "black-box" solver and has a wide application in practical. Currently, the most popular AMG methods for solving scalar elliptic PDEs are the so-called classical AMG method [22, 41] and the aggregation-based AMG methods [6, 28, 35, 37, 47, 48].

4.1 Classical and aggregation-based AMG methods

A typical Classical AMG method forms the coarse-level variables by a C/F-splitting algorithm, which partitions the fine-level variables Ω into C-variables and F-variables measured by strong or weak connection between nodes, that is, $\Omega = C \cup F$ and $\phi = C \cap F$. The interpolation operator P is defined via a weighted sum of the coarse grid nodes and usually has form of

$$e_i^h = (Pe^H)_i = \begin{cases} e_i^H, & \text{if } i \in C, \\ \sum\limits_{k \in C_i^h} P_{ik} e_k^H, & \text{if } i \in F, \end{cases} \tag{4.1}$$

where $C_i^h \subset C$ is interpolator set of node i. The restriction operator usually is transport of the interpolation operator. The coarse matrix is built by the Galerkin principle $A^H = P^T A^h P$. One thing should be notice is that interpolator P constructed by (4.1) will make A^H dense. The classical AMG converges quickly and also is very robust for many applications. However, interpolation P in (4.1) leads to high complexity of coarse grid matrices (i.e., more nonzero entries in each row) and requires large amount of memory during SETUP.

In aggregation AMG methods, on the other hand, the set of fine nodes are decomposed into small mutually disjoint subsets $\Omega = C_1 \cup C_2 \cdots C_{m-1} \cup C_m$ and each subset C_i corresponds to a coarse grid node on the coarse level. The interpolator P is defined as

$$P_{ij} = \begin{cases} 1, & \text{if } j \in C_i, \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

Note that P has exactly one nonzero entry per row, which makes coarse grid matrix sparse. However, this "simple" interpolator P will deteriorate the convergence of AMG and do not provide grid independent convergence.

In order to overcome this drawback, two alternative approaches are possible. One strategy is to damp P by a relaxation step (for example $P = (I - wD^{-1}A)P$). This approach, usually referred to as the smoothed aggregation (SA) AMG, is proposed by [48]. It is successfully applied to linear elasticity. But smoothed interpolation typically leads to a hierarchy with very dense coarse level matrices, which will cause expensive matrix-vector product computations and excessive requirement for memory. Another approach is to adopted some enhanced multilevel cycles in the solve phase rather than modifying interpolator P, like algebraic multi-level iteration cycle (AMLI-cycle) and Krylov-based cycles (K-cycle). Notay [35, 37] employed an unsmoothed aggregation (UA) along with K-cycle [27, 36] as preconditioning for iterative Krylov methods. Compared with the classical AMG and SA AMG, unsmoothed aggregation AMG is cheaper in its setup and low grid and operator complexities; see Table 1 (Grid Comp. and Op. Comp.).

We consider to solve pressure systems obtained from Jacobian systems using GMRES preconditioned by the classical AMG and unsmoothed aggregation AMG methods. The stopping criteria is to reduce relative residual in Euclidean norm to 10^{-6}. We employ the

Table 1: Grid and operator complexities of classical and unsmoothed aggregation AMGs.

Case	Size	Classical AMG		UA AMG		AmgX	
		Grid Comp.	Op. Comp.	Grid Comp.	Op. Comp.	Grid Comp.	Op. Comp.
1	2,097,152	1.61	3.51	1.33	1.33	2.47	2.93
2	556,402	2.01	5.46	1.47	1.78	1.82	3.18
3	1,094,422	2.07	4.11	1.29	1.40	2.21	10.39
4	2,188,843	1.68	3.48	1.21	1.28	3.15	4.69
5	200,002	2.03	4.38	1.32	1.30	1.53	2.03
6	512,002	1.92	3.94	1.33	1.32	1.58	2.28
7	900,002	1.81	5.56	1.42	1.56	1.88	2.92
8	900,002	1.81	5.56	1.42	1.56	1.88	2.92

Table 2: Performance of classical and unsmoothed aggregation AMGs (time: seconds).

Case	Size	Classical AMG (V-cycle)				UA AMG (K-cycle)				AmgX	
		#Its	SETUP	SOLVE	Total	#Its	SETUP	SOLVE	Total	#Its	Total
1	2,097,152	6	10.10	3.61	13.82	10	1.46	5.72	7.26	36	8.46
2	556,402	7	2.48	2.26	4.76	15	0.87	3.30	4.21	78	8.20
3	1,094,422	7	3.57	3.36	6.98	14	1.05	5.75	6.86	36	6.16
4	2,188,843	7	6.48	5.11	11.70	12	1.68	6.95	8.74	25	7.46
5	200,002	4	0.41	0.37	0.79	9	0.19	0.45	0.66	14	0.52
6	512,002	5	1.53	1.20	2.75	9	0.39	1.19	1.59	13	0.92
7	900,002	8	4.42	3.46	7.92	11	1.01	3.45	4.51	36	4.40
8	900,002	13	4.41	5.64	10.09	8	1.03	2.53	3.60	100	6.18

open-source linear solver package FASP (`http://fasp.sourceforge.net`) for both AMG methods. We can see, from Table 2, that UA AMG has a slower convergence rate than the classical AMG method, but cost much less CPU time than the classical AMG for all tested problems. Here, "#Its" denotes the number of iteration needed to converge for GMRES method, "SETUP" denotes the wall time of setup phase of AMG, "SOLVE" denotes the wall time of GMRES method and "Total" is total time, including AMG setup and GMRES. This is because of the setup time of classical AMG is more than 50% of total solution time, even up to 75% (for Case 1), while the ratio is less than 30% for the UA AMG method. The results show that UA AMG is more efficient than classical AMG method for the pressure equations in reservoir simulation.

4.2 AMG implementation on GPU

The SETUP phase of AMG is inherently serial, it is not easy to be implemented on multi-core processors, especially for fine-grained parallelism GPUs. However, many attempts have been tried. Bell et al. [4] presented an SA AMG code on GPUs, which is included into the state-of-the-art algebraic multigrid GPU solver package AmgX developed by Nvidia (`https://developer.nvidia.com/amgx`). However, this parallel AMG method is not robust enough for our test problems and the convergence performance is much worse than our sequential UA AMG; see the last column in Table 2. Brannick et al. [10] developed a parallel UA methods based on parallel maximal independent set algorithm; the numerical therein showed the proposed UA AMG method converges uniformly for the Laplacian problem. Liu et al. [32] implemented several classical AMG methods on GPUs but the SETUP phase is serial and executes on CPU.

Based on above investigations, it is very difficult to obtain an efficient parallel AMG method for highly nonsymmetric pressure matrix on GPUs. Hence we perform the AMG SETUP phase using FASP on CPU and only do SOLVE phase on GPUs. When compared with classical AMG, the SETUP phase of UA AMG accounts for a small portion in the solution time and also UA AMG has lower operator complexity (see Table 1) and will save time for transferring data from host to device. Moreover the convergence rate of UA on GPUs is almost same to as CPU because only the SOLVE phase is done on GPUs.

Table 3: Percentage of wall-time spent on each level in UA AMG (K-cycle).

Level	#Rows	#Nonzeros	NNZ/Row	CPU (%)	GPU (%)
0	1094426	7478974	6.83	39.29	24.16
1	331645	3175097	9.57	21.56	14.58
2	110788	1313952	11.86	16.07	12.70
3	40167	546211	13.60	7.91	7.03
4	16142	235160	14.57	5.99	4.04
5	7250	106112	14.64	2.81	8.87
6	3579	50663	14.16	2.55	4.99
7	1941	25593	13.19	1.02	4.81
8	1094	13430	12.28	0.38	5.05

Hence we apply the UA method in the first stage of CPR-AMG preconditioner on GPUs.

AMG is a multi-level method and the number of unknowns reduces exponentially from the finest to the coarsest level. Gandham et al. [25] present some statistics for classical multigrid method based on V-cycle and they find a interesting phenomena that time distribution for solving phase is very different on CPU and GPU. On CPU, the time used for each level is almost proportion to number of unknowns on each level, but this does not hold on GPU. The percentage of time used on fine levels in SOLVE phase decreases but that on coarse levels increases, when compared with time distribution on CPU.

There are two facts that contribute to this phenomena–one is the cache structure difference between CPU and GPU, another is the architectural difference. UA AMG method based on K-cycle involves more coarse spaces than V-cycle, thus, it's very important to investigate the performance of the method on GPUs. We tested pressure matrix and calculate the time of each level for both on CPU and GPU. In Table 3 the ratio of each level's time over the total AMG time is presented. We exclude the coarsest level, which is solved on CPU by a direct solver. We can see, from Table 3, the ratio on the finest 3 levels (Level 0 $\sim$ Level 2) decrease significantly but increase on those coarser levels (Level 5 $\sim$ Level 8). On CPU, the coarse levels (with few unknowns) are more efficiently handled due to large cache size, while GPUs are throughput-oriented and are good for large data set. We should terminated the coarsening process with proper gird size.

4.3 Smoothers for multigrid methods

Smoother is a critical component of multigrid methods and aims to making the underlying error smooth so that it can be approximated accurately and efficiently on a coarse grid. Very often, some simple iterative methods, like Richardson, Jacobi and Gauss-Seidel (GS), are used as the smoother for multigrid methods. But in the context of multi-core or many-core processors, the typically applied smoothers, such as lexicographical GS, usually have poor scalability. Baker et al. [2] investigated the properties of smoothers in the context of algebraic multigrid running on parallel computers with potentially millions of processors and compare C-F GS smoother, Polynomial smoothers, Chebyshev

Table 4: Performance comparison between smoothers on GPU.

Case	Weighted Jacobi			Multi-Color GS			Hybrid Smoother		
	#Nits	#Lits	Time	#Nits	#Lits	Time	#Nits	#Lits	Time
2	246	3369	571.88	246	3288	589.17	246	3357	558.12
3	239	3746	1152.12	239	3600	1100.68	240	3674	1049.68
4	343	5790	3153.77	344	5720	3200.96	339	5769	3063.96
5	1697	12711	1410.16	1550	12242	1345.16	1651	12547	1400.76
6	2017	18005	4534.27	2061	17527	4534.76	2108	17807	4516.44
7	224	999	770.10	224	960	763.32	224	981	740.70
8	1107	9594	4177.77	1000	9121	4212.45	1079	9330	3947.52

smoothers and proposed a Hybrid GS smoother, that is, using GS independently on each processor and updating in a Jacobi-like manner on processor boundaries. The popular Hybrid GS smoother has multigrid smoothing properties which are independent of the number of processors in many applications, provided that the problem size per processor is large enough. However, this Hybrid GS smoother can not be well worked on GPUs due to GPUs is fine-grained parallelism, which varies the coarse-grained parallelism of MPI. Li and Saad [31] discussed smoothers (preconditioners) on GPUs, including ILU, multi-color SSOR and least square polynomial.

In this paper, the lexicographical ordering GS smoother is used on CPU and a multi-color GS smoother [23] is used on GPU. In multi-color GS smoother, nodes of graph corresponding to matrix are first divided into groups via some algebraic graph coloring strategies; see for example [40, Chapter 3]. After the coloring step, nodes of same color can be concurrently smoothed, which is like the Jacobi method and different color groups are smoothed sequentially, which is like the GS method. In AMG methods, the coarse matrices usually become more and more dense as coarsening continues; see Table 3 (even more so in the Classical AMG method). This observation suggestions that each node will has more connections to adjacent nodes, which will result in more color groups and hence smaller group sizes (less parallelism). In our linear solver, we use unsmoothed aggregation AMG with the K-cycle, which involves more times of smoothing sweeps than the V-cycle. Moreover, each color group corresponds to a kernel function, the overhead of function launches can not be ignored in this case. Therefor, we propose a hybrid smoother–The multi-color GS is used in fine levels (finest 2 levels) and the weighted Jacobi method is used on the remaining levels, to avoid small data set caused by coloring strategy on coarse space. The weight of weighted Jacobi method can be obtained by the Arnoldi procedure [40, Chapter 6], which can be fully parallelized on GPUs.

We compare the weighted Jacobi, multi-color GS and hybrid smoothers in our linear solver for the Jacobian systems from FIM and the results are reported in Table 4. "#Nits" and "#Lits" denote the number of Newton iterations and linear iterations, respectively. "Time" denotes time of linear solver in simulation. From this table, we find that hybrid smoother cost less time than multi-color GS smoother and damped Jacobi smoother, while iteration number of hybrid smoother is a little more than multi-color GS. From the time aspect, exposing sufficient parallelism with compromising the rate of convergence is tradeoff here.

4.4 Coarsest-level solvers

In AMG, direct solvers are often applied to the coarsest linear systems due to the small size of coarsest space. There are many good sparse direct solvers on CPU, such as MUMPS [34], UMPACK [46], SuperLU [43]. However there are not many efficient sparse direct solver packages on GPUs. MAGMA [33] is a direct solver for dense matrices designed for heterogeneous/hybrid architectures, starting with current "Multicore + GPU" systems. However, it is a linear algebraic library oriented dense matrix. The sparse LU solver in cuSOLVER [17] developed by Nvidia is not flexible due to the fact that the LU decomposition and back substitution are packed in one function. This, obviously, is not suitable for our situation when the decomposition is done once and back substitution is needed in each iteration. In this paper, we solve the coarsest linear systems on GPUs by simple iterative methods, such as the Jacobi method. By using the Jacobi iteration is used as the coarsest space solver on GPUs, we get about 8% performance gain compared with MUMPS on CPUs in numerical tests.

5 Numerical experiments

In this section, we construct several field-scale reservoir models and test the performance of each of part of HiSim carefully.

5.1 Test environment and test problems

All experiments are run on a work station with 2 Intel(R) Xeon(R) CPU X5675, 39GB of RAM and an NVIDIA Tesla C2075 coprocessor with 6GB of memory. The CPU clock is 3.07GHz and the sizes of L1, L2 and L3 cache are 64KB (32KB double and 32KB int), 256KB and 12288KB, respectively. The coprocessor have 448 cores and the cache size is 96KB. The operator system of the workstation is Ubuntu 12.04.1LTS. Host compiler is GCC 4.9.2 and device compiler is NVCC 7.0.

In order to get better ideas on the efficiency and robustness of the simulator, we design a set of eight test problems, whose main characteristics are listed in Table 5. In the table, "#ActCells" is number of active cells, "#Wells" is number of wells, "Time" is simulation period in days and "Characteristics" are some other key characteristics of the test problems.

5.2 Comparison with commercial software

In order to establish some performance benchmark, we first compare the CPU version of our simulator with a main-stream commercial simulator. All test problems are solved

Table 5: Summary of test problems.

Case	#ActCells	#Wells	Time (days)	Characteristics
1	2,097,152	0	–	Single-phase, steady state, isotropic
2	561,000	5	2000	Two-phase (water-oil), anisotropic, heterogeneous
3	1,122,000	5	2000	Two-phase (water-oil), anisotropic, heterogeneous
4	2,244,000	5	2000	Two-phase (water-oil), anisotropic, heterogeneous
5	200,000	2	3656	Three-phase, isotropic, gas injection
6	512,000	2	3656	Three-phase, isotropic, gas injection
7	900,000	2	900	Three-phase, anisotropic, heterogeneous
8	900,000	26	900	Three-phase, anisotropic, heterogeneous

Table 6: Performance comparison between simulators (time: seconds).

Simulator	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8
Commercial	32766.72	>3 days	>3 days	6443.02	36375.00	3865.40	46149.47
HiSim-CPU	2018.21	3782.34	10900.23	4423.21	15739.69	1962.23	11380.30
Speedup	16.24	–	–	1.45	2.31	1.96	4.05

using HiSim-CPU, HiSim-GPU and the commercial simulator.

We now compare our numerical results with that produced by a commercial simulator (2012 version). For brevity, we only report the results of Case 2 and 8 in Figs. 2 and 3, respectively. From the figures, we find that the curves of oil production rate and water-cut fraction of three simulators are very close to each other. This shows, at least for the models we constructed, that HiSim is robust and gives reasonable simulation results. Moreover the run time by the commercial simulator are listed in the Table 6. Compared with this commercial simulator, HiSim-CPU shows great advantages in terms of CPU wall time, especially for large-scale models, i.e., Case 2, 3, 4 and 8.

5.3 Speedup

In our numerical experiments, the run time distribution of linear solver in HiSimCPU for each problem is exhibited in Fig. 4. The stopping criteria for the CPR-AMG precondition-

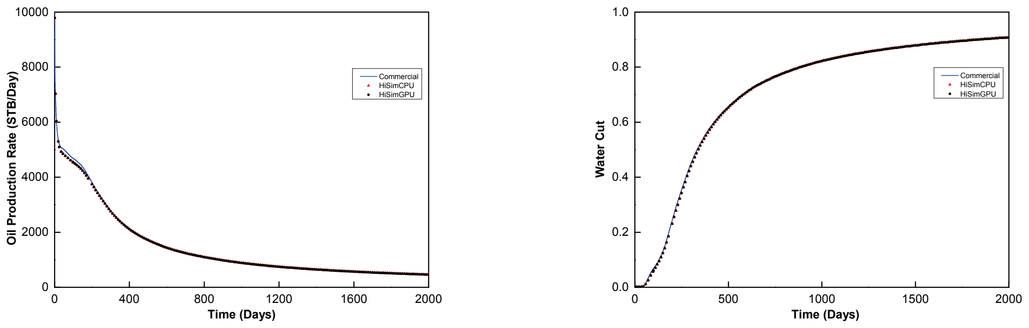

Figure 2: Comparison of field oil production rate (Left) and field water-cut (Right): Case 2.

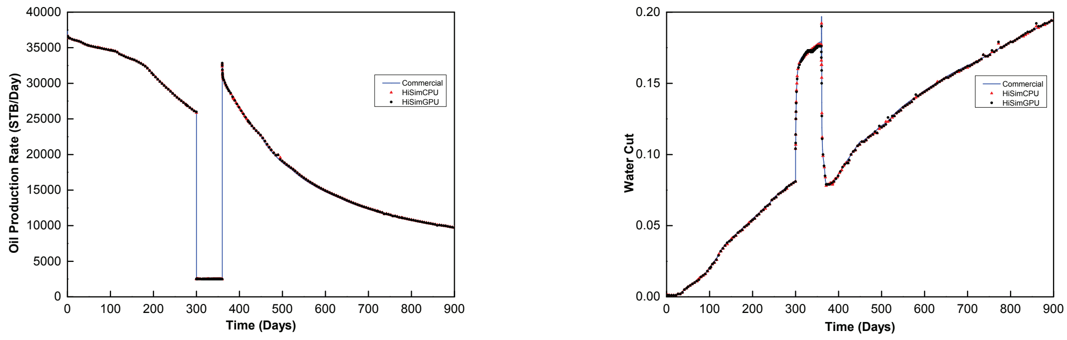

Figure 3: Comparison of field oil production rate (Left) and field water-cut (Right): Case 8.

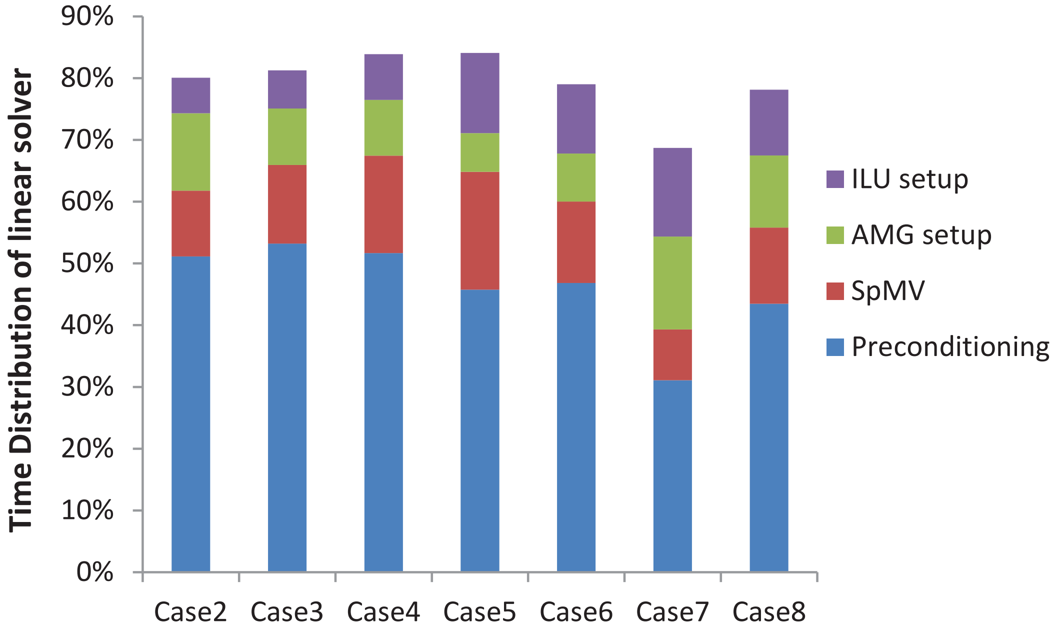

Figure 4: Time distribution for each part of linear solver.

ed GMRES method is for the relative residual in the Euclidian norm to be less than 10^{-3}. We can see from the figure that linear solver is the major computational part (more than 75% for most cases) of the simulations and the preconditioning (CPR-AMG) takes a large fraction of the linear solver (up to 60%). In this sense, we have two possibilities to improve performance of our simulator on the CPU-GPU architecture: One is to fine-tune the implementation of CPR-AMG; The other is to construct a better preconditioner which takes full advantage of the new architecture.

In Table 7, we summarize the results of HiSim-CPU and HiSim-GPU for all case problems. Columns 3, 4 and 5 are number of time steps, number of Newton iterations and

Table 7: Summary of simulation results (time: seconds).

Case	Arch	Time Steps	Newton Iteration	Linear Iteration	SOLVE Time	LinSolver Time	Total Time	LinSolver Percentage
	CPU	202	246	3270	1246.94	1615.70	2018.21	80.06%
2	GPU	202	246	3357	193.10	558.12	960.63	58.10%
	speedup	–	–	–	6.5x	2.9x	2.1x	–
	CPU	165	242	3618	2493.77	3074.28	3782.34	81.28%
3	GPU	164	240	3674	410.04	1049.68	1757.74	59.71%
	speedup	–	–	–	6.1x	2.9x	2.2x	–
	CPU	245	340	5632	7354.12	9144.50	10900.23	83.89%
4	GPU	248	339	5769	1166.52	3063.96	4819.69	63.01%
	speedup	–	–	–	6.3x	3.0x	2.3x	–
	CPU	174	1443	11861	2867.07	3719.48	4423.21	84.09%
5	GPU	202	1651	12547	566.30	1400.76	2303.75	60.80%
	speedup	–	–	–	5.1x	2.7x	1.9x	–
	CPU	255	2131	17021	9446.04	12435.17	15739.69	79.01%
6	GPU	263	2108	17807	1836.86	4516.44	7820.96	57.75%
	speedup	–	–	–	5.1x	2.8x	2.0x	–
7	CPU	194	224	976	771.59	1348.08	1962.23	68.70%
	GPU	194	224	981	146.97	740.70	1354.85	54.67%
	speedup	–	–	–	5.2x	1.8x	1.5x	–
8	CPU	229	1067	8011	6352.42	8890.68	11380.30	78.12%
	GPU	224	1079	9330	1161.12	3947.52	6437.14	61.32%
	speedup	–	–	–	5.5x	2.3x	1.8x	–

number of linear iterations, respectively. In most Cases, the numbers of time steps and Newton iterations of HiSim-GPU changed slightly when compared with HiSim-CPU. There is a modest increase in the total number of linear iterations, which is caused by hybrid smoother used in AMG in HiSim-GPU. These facts show that GPU based linear solver is robust.

Note that time of the linear solver accounts for 78% $\sim$ 84% of total simulation time. According to the Amdahl's Law

$$S=\frac{1}{q+(1-q)/p},$$

where q represents the fraction of serial code and p is the number of processors used to accelerate the complementary parallel fraction of the code. The maximal speedup could be around 5.0x for simulator for these test problems if the whole linear solver can be scalable and implemented on GPUs.

The third row of Table 7 gives the speedup of SOLVE phase, linear solver and the whole simulator, respectively. For all test cases, the SOLVE phase of HiSim-GPU is of the order 4.0x to 6.5x faster than that of HiSim-CPU. This indicates that GPU still has a great potential for iterative methods. Due to the fact that AMG setup and ILU decomposition (SETUP phase) are sequential and done on CPU, the speedup of linear solver is less than 3.0x.

6 Conclusions

We investigate a GPU-based linear solver for our in-house reservoir simulator HiSim. A hybrid block ELL storage format for fully-implicit reservoir simulation is presented and SpMV based on this data structure gains a satisfactory speedup. We implement the SOLVE phase of ILU(0) and develop a hybrid smoother for AMG method on GPU. Numerical results show that our simulator on GPU is robust and effective. The numerical experiments indicates that the linear solver, especially the preconditioner part, on GPU can and must be further improved. To achieve desirable performance on many-core architectures like GPU, we need to redesign both algorithm and implementation:

- The serial SETUP phase of the linear solver severely deteriorates the parallel scalability of our implementation. The original algorithm (like AMG setup) is difficult to be ported to fine-grained parallel computing platforms efficiently. We need to redesign the AMG algorithm. For example, the auxiliary grid AMG technique proposed in [51] can be exploited to speedup AMG setup on GPU.
- The software eco system for numerical reservoir simulation on GPUs is still far less developed than on CPUs. We are in great need for efficient and robust implementation of AMG, ILU and sparse direct solvers. Our numerical study shows that, with these fundamental modules available, we can obtain a cost-effective simulator on CPU-GPU architecture rather quickly.

Acknowledgments

The authors would like to thank Professor Chunsheng Feng for his assistance in numerical tests and many helpful discussions. The paper is finished during Li's visit to the State Key Laboratory of Scientific and Engineering Computing (LSEC). Li is thankful to the kind support from LSEC. The authors would like to thank RIPED, PetroChina, for providing data for the numerical tests and support through PetroChina New-generation Reservoir Simulation Software (No. 2011A-1010), the Program of Research on Continental Sedimentary Oil Reservoir Simulation (No. z121100004912001) founded by Beijing Municipal Science & Technology Commission and PetroChina Joint Research Funding No. 12HT1050002654.

References

[1] J. R. Appleyard, J. D. Appleyard, M. A. Wakefield and A. L. Desitter, *Accelerating reservoir simulators using GPU technology*, SPE Reservoir Simulation Symposium, 2011.

[2] A. H. Baker, R. D. Falgout, T. V. Kolev and U. M. Yang, *Multigrid smoothers for ultra-parallel computing,* SIAM J. Sci. Comput., 33(5) (2011), pp. 2864–2887.

[3] R. E. Bank, T. F. Chan, W. M. Coughran Jr and R. K. Smith, *The Alternate-Block-Factorization procedure for systems of partial differential equations,* BIT Numer. Math., 29(4) (1989), pp. 938–954.

[4] N. Bell, S. Dalton and L. N. Olson, *Exposing fine-grained parallelism in algebraic multigrid methods*, SIAM J. Sci. Comput., 34(4) (2012), pp. C123–C152.

[5] N. Bell and M. Garland, *Efficient sparse matrix-vector multiplication on CUDA,* Technical report, Nvidia Technical Report NVR-2008-004, Nvidia Corporation, 2008.

[6] D. Braess, *Towards algebraic multigrid for elliptic problems of second order*, Computing, 55(4) (1995), pp. 379–393.

[7] A. Brandt, *Algebraic multigrid theory: the symmetric case,* Appl. Math. Comput., 19(1) (1986), pp. 23–56.

[8] A. Brandt, S. McCormick and J. Ruge, *Algebraic multigrid (amg) for automatic multigrid solutions with application to geodetic computations,* Report, Inst. for Computational Studies, Fort Collins, Colo, 1982.

[9] A. Brandt, S. McCoruick and J. Ruge, *Algebraic multigrid (amg) for sparse matrix equations,* Sparsity Appl., (1985), pp. 257–284.

[10] J. Brannick, Y. Chen, X. Hu and L. Zikatanov, *Parallel unsmoothed aggregation algebraic multigrid algorithms on gpus,* Numerical Solution of Partial Differential Equations: Theory, Algorithms and Their Applications, pages 81–102, Springer, 2013.

[11] J.-H. Byun, R. Lin, K. A. Yelick and J. Demmel, *Autotuning sparse matrix-vector multiplication for multicore,* Technical Report UCB/EECS-2012-215, EECS Department, University of California, Berkeley, November 2012.

[12] H. Cao, H. Tchelepi, J. Wallis and H. Yardumian, *Parallel scalable unstructured CPR-type linear solver for reservoir simulation,* Paper SPE 96809 presented at the SPE Annual Technical Conference and Exhibition, Dallas, Texas, 9-12 October, 2005.

[13] Z. Chen, G. Huan and Y. Ma, Computational Methods for Multiphase Flows in Porous Media, Volume 2, SIAM, 2006.

[14] J. W. Choi, A. Singh and R. W. Vuduc, *Model-driven autotuning of sparse matrix-vector multiply on GPUs*, ACM SIGPLAN Notices, 45(5) (2010), pp. 115.

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

[16] K. H. Coats et al., *A note on IMPES and some IMPES-based simulation models,* SPE J., 5(03) (2000), pp. 245–251.

[17] CuSOLVER, `https://developer.nvidia.com/cusolver.`

[18] H. V. Dang and B. Schmidt, *CUDA-enabled sparse matrix-vector multiplication on GPUs using atomic operations*, Parallel Comput., 39(11) (2013), pp. 737–750.

[19] A. H. Dogru, L. S. Fung and U. Middya et al., *A next-generation parallel reservoir simulator for giant reservoirs,* SPE/EAGE Reservoir Characterization & Simulation Conference, 2009.

[20] J. Douglas Jr, D. Peaceman and H. Rachford Jr et al., *A method for calculating multi-dimensional immiscible displacement,* Trans. Amer. Inst. Min. Metallurgical Petroleum Eng., pages 297–306, 1959.

[21] K. Esler, K. Mukundakrishnan, V. Natoli, J. Shumway, Y. Zhang and J. Gilman, *Realizing the potential of GPUs for reservoir simulation,* ECMOR XIV-14th European Conference on the Mathematics of Oil Recovery, 2014.

[22] R. Falgout, *An introduction to algebraic multigrid computing*, Comput. Sci. Eng., 8(6) (2006).

[23] C. Feng, Multilevel Iterative Methods and Solvers for Reservoir Simulation on CPU-GPU

Heterogenous Computers, PhD thesis, Xiangtan University, 2014.

[24] L. S. Fung, M. O. Sindi and A. H. Dogru et al., *Multi-paradigm parallel acceleration for reservoir simulation,* SPE Reservoir Simulation Symposium, 2013.

[25] R. Gandham, K. Esler and Y. Zhang, *A GPU accelerated aggregation algebraic multigrid method,* Comput. Math. Appl., 68(10) (2014), pp. 1151–1160.

[26] M. E. Hayder and M. Baddourah et al., *Challenges in high performance computing for reservoir simulation,* Paper SPE, 152414 (2012), pp. 4–7.

[27] X. Hu, P. S. Vassilevski and J. Xu, *Comparative convergence analysis of nonlinear AMLI-cycle multigrid,* SIAM J. Numer. Anal., 51(2) (2013), pp. 1349–1369.

[28] H. Kim, J. Xu and L. Zikatanov, *A multigrid method based on graph matching for convection–diffusion equations,* Numer. Linear Algebra Appl., 10(1-2) (2003), pp. 181–195.

[29] H. M. Klie, H. H. Sudan, R. Li and Y. Saad et al., *Exploiting capabilities of many core platforms in reservoir simulation,* SPE Reservoir Simulation Symposium, Society of Petroleum Engineers, 2011.

[30] S. Lacroix, Y. V. Vassilevski and M. F. Wheeler, *Decoupling preconditioners in the implicit parallel accurate reservoir simulator (IPARS)*, Numerical Linear Algebra Appl., 8(8) (2001), pp. 537–549.

[31] R. Li and Y. Saad, *GPU-accelerated preconditioned iterative linear solvers*, J. Supercomput., 63(2) (2013), pp. 443–466.

[32] H. Liu, B. Yang and Z. Chen, *Accelerating algebraic multigrid solvers on NVIDIA GPUs*, Comput. Math. Appl., 70(5) (2015), pp. 1162–1181.

[33] MAGMA, `http://icl.cs.utk.edu/magma/`.

[34] MUMPS, `http://mumps-solver.org`.

[35] A. Napov and Y. Notay, *An algebraic multigrid method with guaranteed convergence rate,* SIAM J. Sci. Comput., 34(2) (2012), pp. A1079–A1109.

[36] Y. Notay, *Flexible conjugate gradients*, SIAM J. Sci. Comput., 22(4) (2000), pp. 1444–1460.

[37] Y. Notay, *Aggregation-based algebraic multigrid for convection-diffusion equations*, SIAM J. Sci. Comput., 2012.

[38] E. J. Pavlas Jr et al., *Fine-scale simulation of complex water encroachment in a large carbonate reservoir in saudi arabia,* SPE Reservoir Evaluation & Engineering, 5(05) (2002), pp. 346–354.

[39] D. W. Peaceman, *Presentation of a horizontal well in numerical reservoir simulation*, The 11th SPE Symposium on Reservoir Simulation, 1991.

[40] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, 2003.

[41] K. Stüben, Algebraic Multigrid (AMG): an Introduction with Applications, GMD Forschungszentrum Informationstechnik, 1999.

[42] H. Sudan, H. Klie, R. Li and Y. Saad, *High performance manycore solvers for reservoir simulation,* 12th European Conference on the Mathematics of Oil Recovery, 2010.

[43] SuperLU, `http://crd-legacy.lbl.gov/∼xiaoye/SuperLU/`.

[44] H. Tchelepi and Y. Zhou et al., *Multi-GPU parallelization of nested factorization for solving large linear systems,* SPE Reservoir Simulation Symposium, Society of Petroleum Engineers, 2013.

[45] J. A. Trangenstein and J. B. Bell, *Mathematical structure of the black-oil model for petroleum reservoir simulation,* SIAM J. Appl. Math., 49(3) (1989), pp. 749–783.

[46] UMFPACK, `http://faculty.cse.tamu.edu/davis/suitesparse.html`.

[47] P. Vaněk, M. Brezina and J. Mandel et al., *Convergence of algebraic multigrid based on smoothed aggregation,* Numer. Math., 88(3) (2001), pp. 559–579.

[48] P. Vaněk, J. Mandel and M. Brezina, *Algebraic multigrid by smoothed aggregation for sec-*

ond and fourth order elliptic problems, Computing, 196 (1996), pp. 179–196.

[49] J. WALLIS, *Incomplete Gaussian elimination as a preconditioning for generalized conjugate gradient acceleration,* Paper SPE 12265 presented at the SPE Reservoir Simulation Symposium, San Francisco, California, 15-18 November, 1983.

[50] J. WALLIS, R. KENDALL, T. LITTLE AND J. NOLEN, *Constrained residual acceleration of conjugate residual methods,* SPE, 13536 (1985), pp. 10–13.

[51] L. WANG, X. HU, J. COHEN AND J. XU, *A parallel auxiliary grid algebraic multigrid method for graphic processing units,* SIAM J. Sci. Comput., 35(3) (2013), pp. C263–C283.

[52] S. WU, C. FENG, C.-S. ZHANG, Q. LI AND E. AL, *A multilevel preconditioner and its shared memory implementation for new generation reservoir simulator*, Petroleum Science, (2014), pp. 1–18.

[53] S. YU, H. LIU, Z. J. CHEN, B. HSIEH AND L. SHAO ET AL., *GPU-based parallel reservoir simulation for large-scale simulation problems,* SPE Europec/EAGE Annual Conference, Society of Petroleum Engineers, 2012.