Mo A05

Realizing the Potential of GPUs for Reservoir Simulation

K. Esler* (Stone Ridge Technology), K. Mukundakrishnan (Stone Ridge Technology), V. Natoli (Stone Ridge Technology), J. Shumway (Stone Ridge Technology), Y. Zhang (Stone Ridge Technology) & J. Gilman (iReservoir)

SUMMARY

Higher stakes from deep-ocean drilling, increasing complexity from unconventional reservoirs, and an overarching desire for a higher-fidelity subsurface description have led to a demand for reservoir simulators capable of modelling many millions of cells in minutes. Recent advances in heterogeneous computing hardware offer the promise of faster simulation times, particularly through the use of GPUs. Thus far, efforts to take advantage of hardware accelerators have been primarily focused on linear solvers and, in particular, simple preconditioners which often sacrifice rapid convergence for the sake of easy parallelism. This relatively weak convergence, the remaining unaccelerated code paths, and communication bottlenecks have prevented dramatic reductions in run time. A comprehensive approach, however, built from the ground up for accelerators, can deliver on the hardware's promise to meet industry demand for fast, scalable reservoir simulation.

We present the results of our efforts to fully accelerate reservoir simulations on multiple GPUs in an extended black-oil formulation discretized using a fully-implicit finite volume method. We implement all major computational aspects, including property evaluation, Jacobian construction, and robust solvers/ preconditioners on GPUs. The CPR-AMG preconditioner we employ allows low iteration count and near-optimal order(N) scaling of computational effort with system size. This combination of algorithms and hardware enables the simulation of fine-scale models with many millions of cells in minutes on a single workstation without any upscaling of the original problem. We discuss the algorithms and methods we employ, give performance and accuracy results on a range of benchmark problems and real assets, and discuss the strong and weak scaling behavior of performance with model size and GPU count. This work was supported by the Marathon Oil Corporation.

Introduction

The changing landscape of the oil and gas industry presents new challenges for reservoir simulation. Higher fidelity simulations involving huge, seismic-scale models that account for multiple levels of heterogeneity lead to better recovery prediction and favorable economic impact (Dogru, 2011). Very high resolution models with many millions of cells are also needed for simulating unconventionals such as shale and tight gas/oil reservoirs to capture interactions between multiple fractures, between the reservoir and fractures, and between wells (Du et al., 2013). Efficient history matching and optimization workflows are aided by fast turnaround times. In addition, the desire to mitigate risk through uncertainty quantification requires a large ensemble of simulation runs (Alpak et al., 2013). These industry trends create an urgent need for high-performance reservoir simulations scalable to large models.

The means of satisfying these demands for performance in reservoir simulation have evolved, of necessity, with changes in the underlying hardware architectures. The first simulators were designed for scalar processors and benefited from the continuous exponential growth in clock speeds available in the early decades of high-performance computing. This growth allowed simulations to run faster each year just by running on newer hardware. Nonetheless, performance demands outpaced increases in clock speeds, leading to the development of parallel simulators based on distributed memory. Due to constraints imposed by communication, the parallelism employed was necessarily coarse-grained, usually with domain decomposition. This change from serial to parallel execution required the development of new algorithms and a significant change in the architecture of existing codes.

In the last decade, processor clock speeds have plateaued due to power constraints so that further progress in hardware performance now comes almost exclusively through parallelism. As a result, processors have seen a rapid growth in on-chip concurrency due to growth in core count, widening SIMD vector units, and simultaneous multithreading. There is growing consensus that such large-scale on-chip concurrency requires a fundamentally different, *fine-grained* approach to parallelism to complement the coarse-grained approach taken between processors. This need for fine-grained parallelism has been most apparent in the context of GPUs, which provide extremely high memory bandwidth and arithmetic throughput at the cost of exposing thousands of simultaneous threads of execution per GPU.

To take advantage of this performance in reservoir simulation, we have developed a massively parallel, fully-implicit, black-oil simulator built from the outset for multiple GPUs. It combines both a coarse-grained decomposition of data between GPUs and a fine-grained parallel implementation of algorithms executed on each GPU, with careful attention to data layout and a minimalist approach to storage to conserve memory. Robust solver algorithms with strong convergence properties are employed, including the widely-used CPR-AMG preconditioner. In contrast to previous published efforts, all major computational operations—including property evaluation, Jacobian construction and assembly, and solution of the linearized equations—are accelerated on GPUs, avoiding bottlenecks from unaccelerated code paths and data transfer between CPU and GPU. The resulting many-fold increase in simulation speed, even in the presence of complex features, shows that this approach can help meet the evolving performance demands of the industry.

Background

GPU technology has been widely embraced in the oil and gas industry for applications in seismic processing. In reverse time migration (RTM) applications, for example, the stencil-based forward time propagation maps very naturally onto the massively parallel architecture of GPUs (Micikevicius, 2009). Hence, a basic GPU implementation for such imaging workflows is possible with only modest effort. On the other hand, modern reservoir engineering workflows involve a wide range of nonlinear physical processes and require a rich set of application features. While this complexity has limited the scope

of previous attempts to accelerate reservoir simulations, there continues to be great interest in utilizing GPUs to reduce reservoir simulation run times.

The vast majority of published results in reservoir simulation have focused on the development of GPU-accelerated linear solvers, since solver time typically comprises the majority of the computation time. Bayat and Killough (2013) compared GPU and CPU implementations of a Jacobi-preconditioned conjugate gradient linear solver and reported a 45x speedup of an open-source reservoir simulator. This was achieved through an efficient use of shared memory and registers, reduction of bank conflicts, warp synchronization, memory coalescence, and development of a mixed-precision preconditioner. Fung et al. (2013) used a heterogeneous approach in which their simulator ran on CPUs while the linear solver ran optionally on a CPU or GPU. An overall speedup factor of 9x was achieved using a GPU over execution on a single CPU core and 1.57x over parallel execution on CPUs. Tchelepi and Zhou (2013) developed a massively parallel nested factorization preconditioner in combination with a BiCGStab Krylov solver for their reservoir simulator. On the industry standard SPE10 problem (Christie and Blunt, 2001), their GPU solution showed speedup factors of 26x for single-precision and 19x for double-precision computations compared to a single core run on a CPU. Yu et al. (2012) developed GPU-based parallel ILU preconditioners coupled with an in-house black-oil simulator and showed a 6x speedup on the SPE 10 problem relative to their CPU implementation. Chen et al. (2013) later discussed an algebraic multigrid implementation which accelerated only the iteration stage of the preconditioner and demonstrated a performance advantage on homogeneous Poisson problems. Klie et al. (2011) developed a GPU-based GMRES solver with multiple preconditioners and achieved an overall computational speedup of 2x compared to conventional CPU multicore simulations. The only negative results are reported by Dzyuba et al. (2012), who observe significantly slower performance on a GPU compared to a dual-socket multicore CPU simulation for many of their test cases. Citing the complexity of a fundamental rewrite, their GPU code consisted of a semi-manual port of their CPU-optimized block ILU(0)-based preconditioner. On the whole, however, GPUs have been shown to provide a significant performance boost.

In the broader context of code parallelization, Amdahl's law strictly limits the performance gains that are achievable through the acceleration of the linear solver alone. For example, if an efficient linear solver comprises 70% of total execution time, the overall simulator speedup will only be about 2.7x even if a 10x speedup of the solver is achieved. Because of this limit, there have been some attempts to accelerate other parts of the simulator in addition to the linear solver. Appleyard et al. (2011) accelerated both the Jacobian assembly and a parallel nested-factorization based solver in a fully implicit black-oil formulation. They observed a considerable performance improvement on a variety of problems over a serial CPU implementation of very similar algorithms. However, these authors performed the reservoir property calculations on the CPU owing to a huge volume of code that would otherwise require porting and optimization.

The present authors' earlier work in reservoir simulation focused on the acceleration of an oil company's in-house implicit pressure/explicit saturation (IMPES) simulator. This work demonstrated significant speedups in the overall simulation time relative to the original CPU code by performing the numerous explicit saturation updates used in the dual time-stepping method on GPU (Esler et al., 2011). Further speedup was obtained by developing a GPU-based algebraic multigrid (AMG) solver for the implicit pressure equation (Esler et al., 2012). This solver, which utilizes a fine-grained parallel approach to accelerate both the setup and solve phases of the preconditioner, was a significant outcome of the original effort, and it has been carried over to the present work. The IMPES code was very successful for two-phase waterflood problems, but robustness and performance were limited by the usual difficulties experienced by an IMPES formulation in the presence of a mobile gas phase. In addition, further acceleration was hampered by the complexities of efficiently managing data transfers from CPU data structures that were not appropriate for optimal performance on GPUs.

The above results, taken together, show the promise of GPUs for enhancing throughput and productivity in reservoir simulation. This story is far from complete. While most of the published results are impressive, many of the GPU speedups reported are for the acceleration of simple preconditioners alone. State-of-the art CPU-based reservoir simulators often utilize more complex multilevel and multistage algorithms that offer stronger convergence properties, particularly as models are scaled to larger sizes. It is not clear how well the published GPU implementations of simple preconditioners will compete with more advanced CPU algorithms. Larger performance gains may be realized by directly confronting the difficult task of implementing complex algorithms and the rich feature sets of reservoir simulations in GPU accelerated code.

Overview

This paper presents our contribution to realizing the full potential of GPUs by combining recent advances in solver and simulator technology with a ground-up implementation of a fully-implicit black-oil formulation in which all major computational tasks are accelerated. By storing only the most relevant data in GPU memory and keeping property evaluations, Jacobian construction, and all linear and nonlinear solver operations on GPUs, fine-grained parallelism and large memory bandwidth are exploited throughout the simulation. Further, we have implemented a growing set of simulator features to apply to real-world black oil models, which already includes rock compaction, hysteresis, primary and secondary well controls, vertical flow performance (VFP) tables, aquifers, and dual porosity / dual permeability. We discuss the challenges we have encountered, the approaches we have taken to meet them, and present some representative results on real-world assets and synthetic benchmark models.

The paper is organized as follows. The algorithms and methods that we employ in our simulator are discussed first. Performance and accuracy results on a range of benchmark problems and real assets are provided next. This is followed by discussions on the strong and weak scaling behavior of performance with model size and GPU count. We conclude by discussing how the performance results justify the effort needed to exploit fine-grain parallelism and what this means for the future outlook for reservoir simulation.

Methodology

Formulation

The governing equations for a black-oil formulation with three fluid components and phases, namely, oil, water, and gas (denoted by subscripts o, w, and g) are,

$$\frac{\partial}{\partial t}\left(\frac{\phi S_o}{B_o}\right) = \nabla \cdot \left[\frac{K k_{ro}}{B_o \mu_o}\left(\nabla P_o - \rho_o g \nabla D\right)\right] + q_o, \tag{1}$$

$$\frac{\partial}{\partial t}\left(\frac{\phi S_w}{B_w}\right) = \nabla \cdot \left[\frac{K k_{rw}}{B_w \mu_w}\left(\nabla \left(P_o + P_{cow}\right) - \rho_w g \nabla D\right)\right] + q_w, \tag{2}$$

and

$$\frac{\partial}{\partial t}\left[\phi\left(\frac{S_g}{B_g} + \frac{R_s S_o}{B_o}\right)\right] = \nabla \cdot \left[\frac{K k_{rg}}{B_g \mu_g}\left(\nabla \left(P_o - P_{cog}\right) - \rho_g g \nabla D\right) + \frac{R_s K k_{ro}}{B_o \mu_o}\left(\nabla P_o - \rho_o g \nabla D\right)\right] + q_g. \tag{3}$$

Here, 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. Also, ϕ is the porosity, K is the static permeability tensor, g is the acceleration due to gravity, D is the depth, and R_s is the solution gas-oil ratio. The flow terms q_α denote the net rate of well injection or production of phase α. For the primary solution variables for each cell, we use P_o, S_w, and either S_g or R_s in the saturated and undersaturated states, respectively. P_{cow} and P_{cog} are the capillary pressures for the oil-water and oil-gas systems.

Algorithm 1 Overview of the important operations of the present simulator. All the operations are performed entirely on the GPUs.

Initialize reservoir and well states $X(t=0)$
Compute properties, Jacobian matrix $J(X)$, and residual $R(X)$
Begin **Time Loop:**
while $t < t_{\text{end}}$ **do**
  Select Δt adaptively
  $t \leftarrow t + \Delta t$
  Begin **Newton-Raphson Loop:**
  $i_{\text{Newton}} = 0$
  while $R(X)$ not converged and $i_{\text{Newton}} \leq$ max Newtons **do**
    Solve linear system $J\,\Delta X = -R(X)$ for state increment ΔX to specified tolerance
    Update state $X \leftarrow X + \Delta X$
    Compute new properties, Jacobian matrix $J(X)$, and residual $R(X)$
    $i_{\text{Newton}} \leftarrow i_{\text{Newton}} + 1$
  end while
  if not converged **then**
    Choose a smaller Δt
    Repeat time loop
  end if
end while

Numerical Method and Solver

We use a standard finite-volume discretization of the black-oil model given by equations (1), (2), and (3). The coupled, nonlinear system of equations are solved in a fully-implicit manner using a Newton-Raphson iterative method to achieve quadratic convergence. Within each iteration, the flow and well control equations are linearized to form a Jacobian system, which is solved using GMRES and a selection of preconditioners.

Algorithm 1 outlines the principal set of operations of the simulation and comprises almost all the run time. All these operations are performed on the GPUs in order to minimize data transfer between CPUs and GPUs and to provide maximum speedup of the overall simulation. The CPUs are primarily used for input, output, model initialization, coordination of the GPUs, and for other operations with negligible processing requirements.

The relative computational times required for major elements of our GPU-accelerated reservoir simulator are shown in Figure 1. Even after acceleration, it is clear that the most effort is spent on the linear equation solver used for obtaining the solution of the Jacobian system. Initialization time, which was negligible before full GPU acceleration, now often comprises 5-10% of a typical run. This initialization, which includes the parsing of ASCII input data, is currently implemented as a serial CPU operation. Future work will address this through the use of compact binary inputs and parallelization, which will be important for scalability to larger models.

For all but the smallest problems, we use a preconditioned GMRES solver to obtain the solution of the linear equations. We find that selecting a robust preconditioner is very critical for achieving optimal performance. This involves balancing several considerations such as algorithmic optimality, parallel scalability, and hardware constraints.

When selecting a preconditioner, it is important to consider the character of the underlying differential equations. The pressure and saturation subsystems exhibit contrasting behaviors: pressure is elliptic in

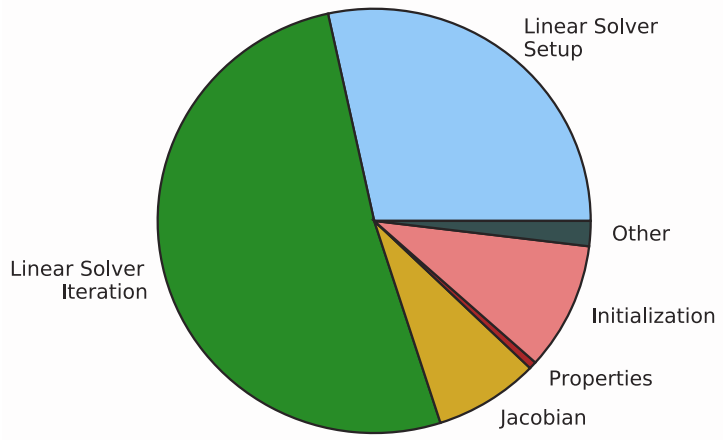

Figure 1: Breakdown of the computational time required for major operations of the GPU-accelerated reservoir simulator. The chart is generated by profiling the SPE 10 benchmark with 1.1 million active cells for 2000 simulation days on one GPU (see Table 1). Total run time is 103 seconds and includes unaccelerated input, output, and model initialization stages.

nature with long-range interactions, while saturations are mostly hyperbolic, exhibiting sharp fronts with short-range interactions. Preconditioners that are highly local in their influence, such as Jacobi and some incomplete factorizations, are therefore well-suited for solving the saturation subsystem. For the elliptic pressure subsystem, it is well known that convergence slows down for such simple preconditioners with increasing system size. Fortunately, methods with optimal convergence rates (i.e. independent of system size) have been developed. Notable among these are the multigrid methods, which include both the geometric and purely algebraic variants. Algebraic multigrid (AMG) has become a popular choice for the pressure subsystem, due to its versatility in handling unstructured grids and its ability to robustly manage models with very strong inhomogeneity in the cell-to-cell transmissibilities. For all the above reasons, a two-stage, constrained-pressure-residual (CPR)-AMG preconditioner has been employed for our simulations. A considerable effort has been spent in optimizing the AMG solver for GPU acceleration (Esler et al., 2012). Furthermore, the CPR reduction method (Wallis et al., 1985; Wallis, 1983) is implemented in such a way that the elliptic nature of the pressure subsystem is strongly preserved. This avoids difficulties with AMG convergence that we have encountered with published CPR reduction schemes. A multicolored block-ILU(0) preconditioner is employed for the CPR second stage.

Parallel scalability also plays an important role in the selection of preconditioners. Many simple preconditioning methods—such as weighted Jacobi, polynomial preconditioning, and multicolored versions of the ILU incomplete factorization method—are naturally parallel and exhibit strong-scaling properties. It can be difficult or impossible to attain the same level of strong-parallel-scalability for AMG, especially on small models. However, for large models, the performance gains achieved by using AMG far outweigh the aforementioned disadvantage in the strong scaling behavior. This has been confirmed from our results, which are shown later.

The choice of preconditioner can also vary depending on the size and complexity of the model and the hardware resources available. For smaller problems, a more traditional incomplete factorization method, such as ILU(0) or nested factorization, may be sufficient to achieve fast convergence. This avoids the overhead involved in setting up complex precondtioners such as CPR-AMG. Since such small models may already be simulated in a very short wall time, we do not focus our efforts on optimizing them but rather the larger and more challenging cases.

Challenges for GPU acceleration

Significant technological advances have been made in GPU architectures, and GPU programming models such as NVIDIA's Compute Unified Device Architecture (CUDA) now support nearly all the abstractions of modern C++. However, utilizing their full potential is not always straightforward. The major challenges involved in a GPU implementation of the simulator are: (i) exposing sufficient fine-grained parallelism, (ii) efficiently using high memory bandwidth, and (iii) working within the limited amount of memory available on the GPU hardware.

Exposing sufficient fine-grained parallelism often involves either a careful redesign of existing algorithms or the development of novel ones in which sequential dependencies on data are reduced or eliminated. In some cases, such as the evaluation of cell-based properties, the obvious parallelism provided by millions of independent cells is sufficient. In the case of the linear solver, exposing sufficient parallelism without compromising the rate of convergence is extremely challenging. It is often necessary to significantly modify or replace the existing algorithms. As mentioned earlier, many simple preconditioners with strong convergence properties are inherently sequential in nature. In a coarse-grained approach to parallelism, the unknowns are divided evenly between processors and a sequential algorithm is often applied to the interior points of each domain independently. The points adjacent to the boundaries between domains, or *halos*, require data exchange between processors, and are typically treated separately. The convergence rate and efficiency of such solvers generally decline as the number of processors grows due to the increasing ratio of boundary points to interior points. In the case of GPUs, many thousands independent threads are required to maximize hardware utilization. In this extreme, nearly all points would be on domain boundaries and the domain decomposition approach would break down. A fundamentally different, more fine-grained approach to solver parallelism is required. In such an approach, threads operate cooperatively in a much more tightly coupled manner, often performing the same instruction on adjacent data, in a manner reminiscent of large-scale vector computing. Matrix reordering techniques, such as multicolored variants of incomplete factorizations, can often be effective. Other methods for exposing this type of parallelism for irregular workloads with branched execution are discussed further in (Bell et al., 2011) and (Esler et al., 2012) in the context of algebraic multigrid.

The second challenge is the maximum utilization of memory bandwidth, which typically limits performance in implicit simulations. As such, efficient data layout which maximizes bandwidth utilization is needed. This is important for CPU codes, but is absolutely critical for GPU codes, which typically have smaller cache sizes and stronger requirement for vectorized data access. For example, strided memory access arising from an array-of-structures (AoS) data layout can result in very poor GPU performance when compared to structure-of-arrays (SoA) layout. On CPUs, it is well-known that such a layout can inhibit vectorization, but the penalty is typically smaller. This can be particularly challenging with large legacy codebases originally written for CPUs, in which the AoS layout is quite common. In these cases, large swaths of code may require a rewrite to obtain good performance. Avoiding this problem was one of the reasons we elected to build a code from the ground-up with no legacy heritage.

The third important challenge with developing a GPU simulator is the limited total memory on the hardware. While GPU device memory has a very high bandwidth, the total device memory is constrained by pin count and power usage. Current cards are limited to about 12 GB per GPU, which is far less than can be attached to a single CPU socket. To allow large models to be simulated with as few GPUs as possible, we elect to store only that data which is essential for computational kernel execution. At the same time, we attempt to minimize large transfers to and from the GPU memory. Hence, frequently-used data—including reservoir state vectors, property tables, the Jacobian matrix, and the residual vectors—are always kept GPU resident. When transfers between CPU and GPU are required, they can often be overlapped with simultaneous computation, which mitigates or eliminates the cost.

During our initial development, we coded features first for CPUs and later ported them to CUDA for GPU execution. In our recent development, however, this ordering has been reversed. With a sufficient infrastructure, it is not much more complicated to add model features with GPU kernels than with more traditional CPU code. Development efficiency is actually enhanced by this approach, because the testing/debugging cycles are made much shorter by the fast GPU execution times. In this way, developer productivity benefits in the same way as the end-user.

Code design and architecture

The facilities for abstraction provided by object-oriented languages can greatly assist in managing complexity and maintaining flexibility. We elected to use C++ for its ability to provide these features without compromising performance. Our GPU code is written in C for CUDA, which is essentially C++ with some extensions for configuring kernel execution and for distinguishing on- and off-chip memory.

At the high level, we have divided the code into two conceptual pieces: a front-end *driver* which manages input parsing, output, and gross execution logic; and a group of back-end *engines*, which execute the primary timestepping and nonlinear solution of the simulator. At present, the driver is executed entirely on CPU, in part due to the requirement for disk IO, which is not currently possible from GPU. The abstract *Engine* class is specialized for execution on GPU or CPU. At present, the primary focus is on the GPU engine, but CPU execution is possible, and the design permits even a heterogenous cooperation between different engine types. The reservoir is decomposed into roughly contiguous domains with an effort to minimize domain boundaries. One engine object is instantiated for each domain, and each engine is assigned to a GPU or CPU. Communication between engines is handled through an abstract communicator object, which implemenents a message-passing scheme. This scheme can be implemented on top of a message passing layer (e.g. MPI) or it can be implemented on lower-level primitives, such as peer-to-peer memory copies between GPUs.

Results and discussion

Test hardware configuration

All GPU results presented below have been executed on a single 2U server (shown in Figure 2), containing two Intel Xeon E5-2620 v2 processors clocked at 2.1 GHz, with 256 GB of RAM and eight NVIDIA Tesla K40 GPUs at 875 MHz, each with 12GB of graphics memory. Most tests were run with only a subset of the GPUs. The CPU results, which were provided by the client oil company, were executed on Intel Xeon X5677 processors each with 8 cores clocked at 3.47 GHz.

Validation on industry benchmark models

The simulator presented in this work has been validated on a range of test models. Figure 3 shows a comparison of simulated production and injection rates computed in the present work with those from a widely-used commercial simulator for the SPE9 (Killough, 1995) industry benchmark model. This is a very small, three-phase black-oil model with only 9,000 active cells, one water injector and twenty-five production wells. Similarly, Figure 4 compares water cuts from present work against the same commercial simulator for the much larger SPE10 test case (Christie and Blunt, 2001). This dead-oil waterflood model contains 1.1 million active cells with a highly heterogeneous permeability field, one water injector, and four production wells. On both models, the results are in excellent agreement, and any small discrepancies are likely the result of differences in adaptive time step selection.

Performance benchmarks

The SPE10 test case (Christie and Blunt, 2001) is perhaps most frequently cited in the literature to examine performance in reservoir simulation. While the feature complexity is low, the extremely high heterogeneity of the permeability field is challenging for many linear solvers. Islam and Sepehrnoori

Figure 2: 2U GPU server that holds 8 Telsa K40 GPU cards. This compute node, used for all the calculations reported here, can simulate models with over 50 million cells.

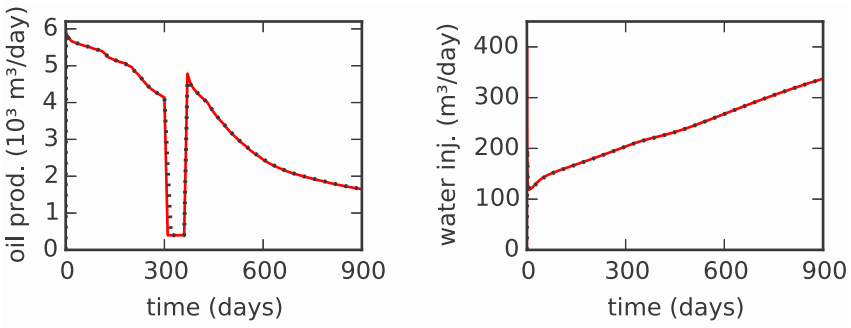

Figure 3: SPE9 benchmark, demonstrating accuracy of GPU accelerated simulator (solid red line) relative to an industry-standard simulator (dotted black line), for (a) field oil production rate and (b) water injection rate.

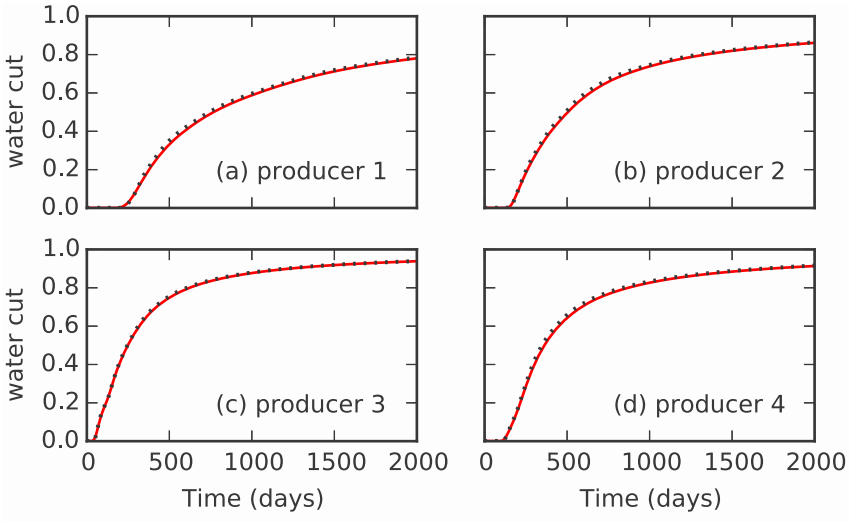

Figure 4: Water cut for SPE10, demonstrating accuracy of GPU accelerated simulator (solid red line) relative to an industry-standard simulator (dotted black line).

	SPE10 (1 GPU)	SPE10 (2 GPUs)
Time steps	68	68
Newton iterations	418	417
Linear iterations	1678	1617
Linears/nonlinear	4.0	3.9
Solver setup time	31 s	20 s
Solver solve time	53 s	40 s
Jacobian time	8 s	4 s
Initialization time	7 s	7 s
Total wall time	103 s	75 s

Table 1: Performance metrics for SPE10 on one and two GPUs. Using two GPUs gives a speedup of about 1.3x over one GPU. The thousands of cores in GPU hardware work optimally for larger systems, so we do not see strong scaling for this model with only 1.1 million cells.

(2013) includes a summary of published timings for this model. The best time reported for a parallel, highly optimized commercial streamline simulator was reported by Natvig et al. (2011) at 170 seconds, while the best fully-implicit finite-volume results are reported by Gratien et al. (2007) at 620 seconds on 64 CPUs using a CPR-AMG preconditioner. However, Fung and Dogru (2008) report a slightly lower total solution time of 490 seconds on 64 processors using a CPR-LSPS preconditioner. Note, however, that since details of tolerance and time stepping are not given, small differences in timing should not be given tremendous weight.

The first column of Table 1 summarizes the present results on this benchmark. The total run time of 103 seconds, executing on a single GPU, is quite fast compared to previously reported results. Adding a second GPU contributes a further speedup of about 1.3x, resulting in a total wall time of 75 s. We do not typically observe strong scaling for models with only 1.1 million cells, because the massive parallelism of the GPU architecture requires larger systems to be optimally subscribed. This can be seen in the absolute timing of the solver setup and solve times in the table. Further, the serialized model initialization time (about 7 seconds) also limits strong scaling. As noted earlier, this is an example of Amdahl's law: the 7 second initialization time was a trivial fraction of the run time before we accelerated other parts of the simulator, but now it becomes a potential serial bottleneck, which will be eliminated in future work.

To examine the scalability of the simulator with system size, we generate horizontal tilings of the original model. We mirror adjacent copies of the original model across tile boundaries. An example 5×10 tiling with 54.7M active cells is shown in Figure 5. Although this tiling results in a fully-connected reservoir, the symmetry of the mirroring implies that the production curves for each well in the tiled model should be the same as the corresponding well in the original model. In this way, we can verify the accuracy for the large models without having to attempt simulation with a commercial product. Note that we do not impose this symmetry on the solution in any way, but it appears naturally in the converged solution.

We show the results of weak scaling on the tiled SPE10 model in Figure 6. We have generated eight different tiled models, to be run on one to eight GPUs, respectively. Each model has six copies of SPE10, connected together as reflected tiles, much like Figure 5. We see a small increase in total run time and the total number of linear iterations as we increase from one to eight GPUs. This indicates good weak scaling, showing that models with tens of millions of cells are practical in a single compute server (Figure 2) with multiple GPUs. This is an important test of both the scalability of the CPR-AMG approach as well as the scalability of our simulator implementation across multiple GPUs. Extrapolating the result would suggest that similar models with hundreds of millions of cells should be possible to simulate in less than an hour on a cluster architecture with dozens of GPUs.

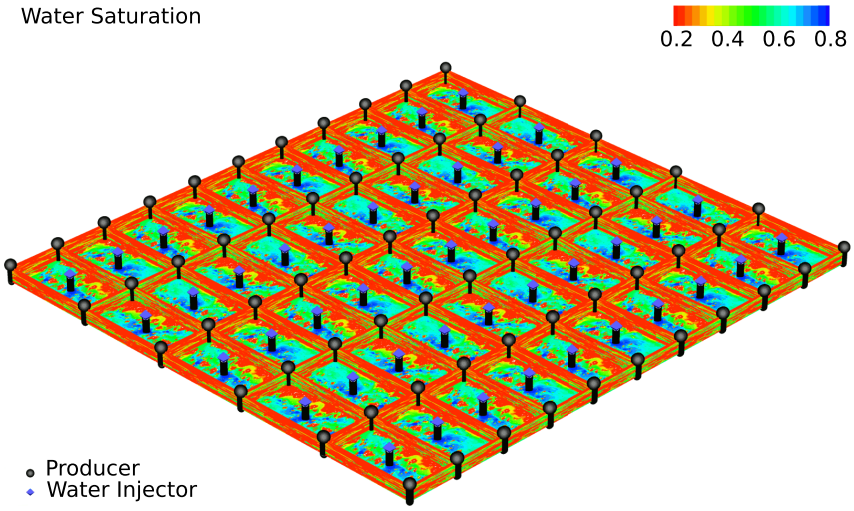

Figure 5: Final water saturation in a 10×5 horizontal tiling of the SPE10 benchmark model. This model contains about 55 million active cells and 250 wells.

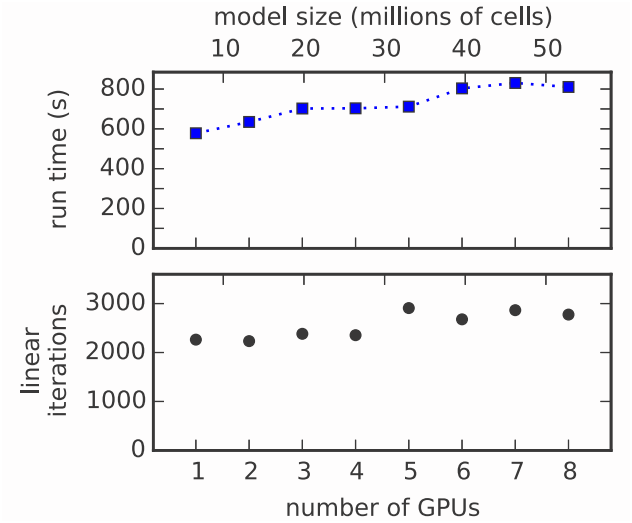

Figure 6: Weak scaling study of tiled SPE10 model. There are six copies of SPE10 for each GPU; for example, the 4 GPU simulation is a 6×4 tiling with approximately 26 million cells. The reported run times in this weak-scaling test do not include parsing of the ASCII input decks and initialization of the model, since we have not yet parallelized these operations.

Model	Important features
Conventional A	Aquifers, Killough's Hysteresis model, Stone I model for oil relative permeability, horizontal wells with history-match controls, artificial lift tables
Conventional B	Multiple aquifers, multiple equilibrium regions, Stone I model for oil relative permeability, gas cap and mobile gas
Unconventional	Tartan grid, horizontal wells, multistage completions, irreversible rock compaction, mobile gas evolution

Table 2: List of important features used in production assets simulations. All models are specified in corner-point geometry.

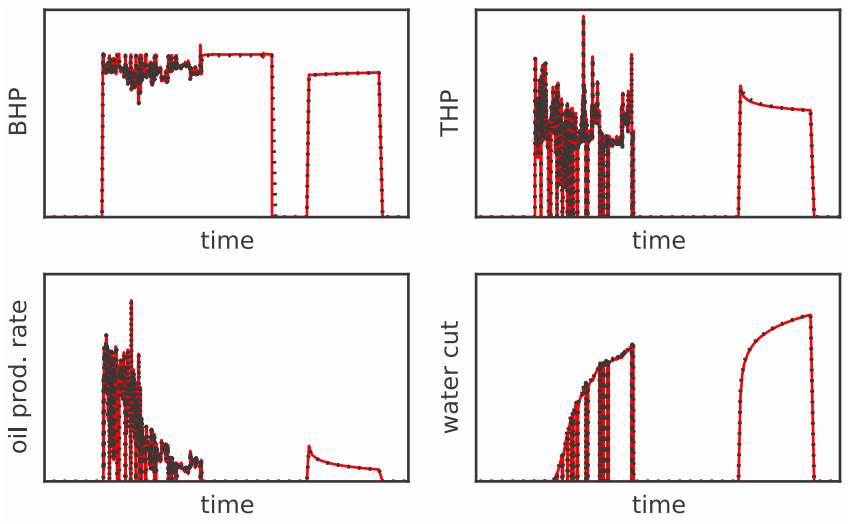

Figure 7: Benchmark results for a representative production well from model "Conventional A" with 179K active cells. Comparison of GPU accelerated simulator (solid red line) relative to a parallel commercial simulator (dotted black line). Both a history-match period and a forecast period with target fluid rate are shown.

Production assets

We now present representative results from the simulations of real production assets provided by a client oil company. These models contain many features that present a number of numerical and implementation challenges. Accuracy and performance metrics are used to validate the current parallel GPU simulator by comparing the results with those generated by our client company using a commercial parallel simulator. The linear solver and nonlinear convergence tolerances are taken to be the same.

The important features of various assets simulated in the present study are listed in Table 2. Due to the proprietary nature of these assets, detailed descriptions and numerical quantification of the results are not provided.

Accuracy evaluation for production asset models

Development of models for production assets are a result of detailed reservoir characterization processes to incorporate essential data and interpretations. The characterization process is iteratively refined through history matching, where historical field production and pressure data are compared against numerically predicted values. Producing gas-oil ratio and water-cut (or water-oil ratio) data are also matched during this period. The history match period is followed by numerous forecast scenarios where future predictions of oil, water and gas production are generated for alternative development assumptions.

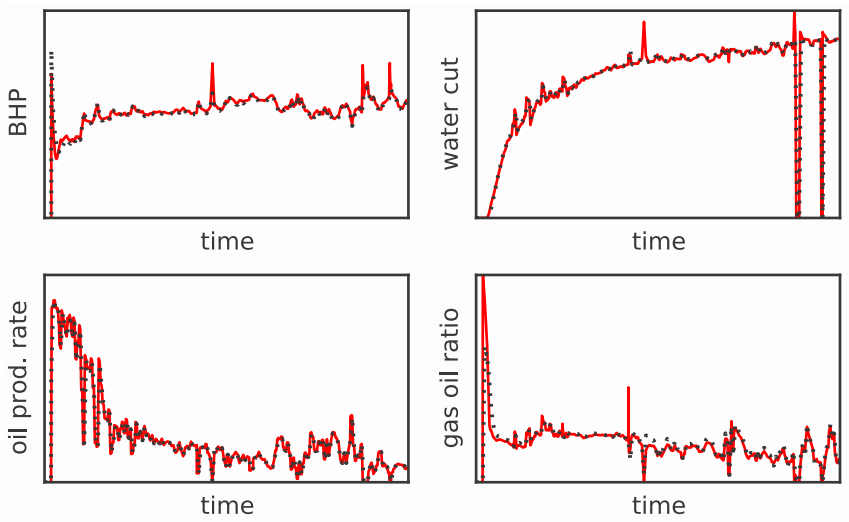

Figure 8: Benchmark results for a reprentative production well from model "Conventional B" with 1.4M active cells. Comparison of GPU accelerated simulator (solid red line) relative to a commercial simulator (dotted black line).

The first model, "Conventional A" features aquifers, significant hysteresis for the non-wetting phases, artificial lift tables for tubing head pressure (THP) controls of the production wells, and the Stone I model for the three-phase oil relative permeability (see Table 2). In Figure 7, the production and forecast data for the bottom hole pressure (BHP), THP, water cut, and oil rate for a representative production well are provided and compared against the results from a commercial parallel simulator. Excellent agreement within 1% is noted for all the quantities and for all production wells in the reservoir.

Model "Conventional B" (see Table 2) is characterized by the presence of a gas cap, multiple equilibrium regions, and aquifers. The presence of mobile gas poses a significant challenge for solvers because of its high mobility, particularly for simulations with large time steps. From Figure 8, the results agree very well with that of the commericial simulator. No convergence difficulties were observed with the solver even with larger time steps. This validates the robustness of our CPR-AMG solver algorithm even in the presence of mobile gas.

Some minor disagreement is noted in the gas-oil ratio between the current results and those of the commercial simulator. The sensitivity of the numerical simulations with high degrees of nonlinearities to the solver parameters coupled with a mismatch in time steps taken by the two simulators are assumed to be the cause of this disagreement. However, excellent agreement is achieved for all the other quantities as can be seen from the figure.

The results from the model of "Unconventional" (see Table 2) are provided in Figure 9. The unconventional nature of the reservoir is characterized by the presence of very high permeability hydraulic fractures interspersed in a low permeability matrix. Furthermore, the model also features an irreversible rock compaction, saturated to undersaturated state transitions for the oil and vice versa, and horizontal wells. The fractures are resolved using a fine tartan grid (grid cells with extreme length scale ratios). The interaction of all these features make this problem highly nonlinear and challenging. As can be seen from Fig 9, the current GPU simulator accurately reproduces the results of the commercial simulator.

Performance evaluation for production asset models

Table 3 provides the GPU speedups obtained on the various productions assets compared to both single and multicore CPU runs of the commercial simulator. The timings for the commercial simulator were provided by the client and reflect a practical real-world estimate, but may not represent optimally tuned

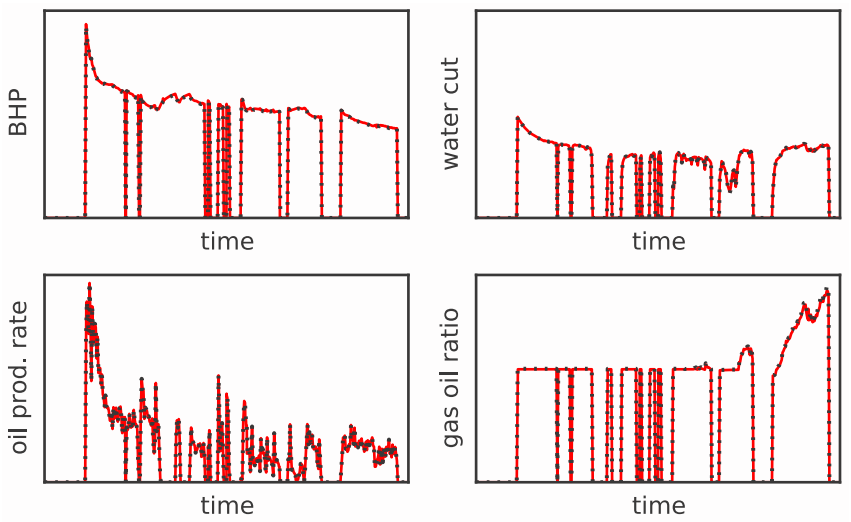

Figure 9: Benchmark results for a representative production well from model "Unconventional" with with rock compaction. Comparison of GPU accelerated simulator (solid red line) relative to a commercial parallel simulator (dotted black line).

Model	Active cells	Commercial run hardware	Present work hardware	Reported speedup
Conventional A	179 000	1 CPU core	1 GPU	27
		16 CPU cores	1 GPU	6
Conventional B	1 360 000	1 CPU core	2 GPUS	430
		32 CPU cores	2 GPUS	45
Unconventional	370 000	1 CPU core	1 GPU	125
		32 CPU cores	1 GPU	43

Table 3: Speedup factors on various real field production assets as reported by our client oil company using the current GPU simulator against a commercial parallel simulator. Note that these speedups reflect the choice of solver algorithm in addition to the hardware performance advantage of the GPU.

performance. As such, the relative speedups we report should be considered as rough estimates. These comparisons demonstrate that even for complex models, a fully-accelerated GPU simulator is capable of achieving significant speedup without loss of accuracy.

On the "Conventional A" model with only 179K cells, the magnitude of the GPU speedup is the smallest compared to the other cases. This is because, in general, fine-grained parallelism using GPU threads is not optimal for accelerating such small-sized problems because of poor thread occupancy and the overhead associated with setting up the threads for launching the kernels. Therefore, as expected, the highest speedup factors are noted for the largest size model "Conventional B" with 1.36M elements. Along the same line of reasoning, the speedup for model "Unconventional" falls in between those of models "Conventional A" and "Conventional B".

In some of the production assets, the maximum time step during the history match period was restricted to smaller values (typically, 1-day steps) in order to match the available production data. While for such small time steps significant speedups are still obtainable, the full potential of our multistage and multiscale solver algorithms are best realized for larger time steps. This is because these algorithms permit larger time steps without loss of performance and accuracy in spite of significant nonlinearities. In such cases, simpler preconditioners usually experience convergence difficulties and may be forced to cut time steps resulting in longer simulation times. In particular, the greatest performance benefits are

observed during the forecast period in which time steps are unconstrained by the historical data report times.

Conclusions

High performance computing (HPC) has entered the era of many-core processors. This new architecture enables significant growth in the speed and density of computation by exposing thousands of on die computation cores and offering high memory bandwidth to them. GPUs have been at the vanguard of this transition and, in our experience, currently offer the best balance of price, performance and usability, but they are not unique participants. Products from competing vendors, including HPC-specific CPUs and accelerators, specialized vector processors, and even commodity CPUs, are increasing in core count and SIMD vector width. Though the hardware architectures differ, there is growing consensus among vendors and researchers that a fine-grained parallel approach will be required to extract optimal performance from these massively parallel processors. The multiplicity in product options has sometimes made companies hesitant to develop software for one particular technology. However, the effort to develop algorithms amenable to fine-grained parallelism is transferable among current and projected many-core architectures and constitutes the fundamental challenge to significant performance gains.

Like the transition from serial to parallel computing, the changes necessary to realize the benefits of many-core require a significant development effort. We have shown that if this investment is made, the benefits are substantial. On a single GPU, the full SPE10 benchmark can be simulated in under two minutes as shown in Table 1, which exceeds the state-of-the-art performance obtainable on a modest cluster just a few years ago (Gratien et al., 2007; Fung and Dogru, 2008). Real-world production assets can be simulated with speedups that range from 6x to nearly 50x faster than multi-core, multiple-CPU simulations, as shown in Table 3, with the greater gains achieved for the larger models. Furthermore, scaling tests on a tiled-SPE10 models show that a single server or workstation with several GPUs can rapidly turn-around reservoir models with tens of millions of cells. As we continue our development, we intend to explore how a cluster of such machines can simulate many instances of even larger models. If weak scalability continues, billion cell models of giant reservoirs should be addressable with a single rack of GPU-accelerated servers. Similarly, many realizations of unconventional models, each with tens of millions of cells and complex features, may be rapidly simulated on the same hardware configuration.

Technology roadmaps give every indication that many-core performance is expected to grow rapidly. Memory bandwidth, arguably the most important metric for performance in implicit simulation codes, is slated to roughly quadruple for GPUs over the next two years through the integration of 3D stacked memory. Once a code has been restructured for fine-grained parallelism, the benefits of these hardware advances become automatic, much as serial codes used to benefit from exponential growth in clock speeds. Current industry pressures are increasing demands on reservoir simulation to eliminate compromises in accuracy imposed by upscaling, to accelerate time-consuming workflows such as history matching, and to mitigate risk through better uncertainty quantification. Harnessing the growing performance of massively parallel hardware by adapting, refining, and optimizing simulation and solver algorithms provides a key advance to meet these challenges.

Acknowledgements

The authors would like to thank Marathon Oil Corporation for funding this work, for providing models for comparison, and for permission to publish.

References

Alpak, F.O., Vink, J.C., Gao, G. and Mo, W. [2013] Techniques for effective simulation, optimization, and uncertainty quantification of the in-situ upgrading process. *Journal of Unconventional Oil and Gas Resources*, **3**, 1–14.

Appleyard, J., Appleyard, J., Wakefield, M. and Desitter, A. [2011] Accelerating reservoir simulators using GPU technology. *SPE Reservoir Simulation Symposium*, SPE-141402-MS.

Bayat, M. and Killough, J.E. [2013] An experimental study of GPU acceleration for reservoir simulation. *SPE Reservoir Simulation Symposium*, SPE-163628-MS.

Bell, N., Dalton, S. and Olson, L. [2011] Exposing fine-grained parallelism in algebraic multigrid methods. NVIDIA Technical Report NVR-2011-002, NVIDIA Corporation.

Chen, Z.J., Liu, H. and Yu, S. [2013] Development of algebraic multigrid solvers using GPUs. *SPE Reservoir Simulation Symposium*, SPE-163661-MS.

Christie, M.A. and Blunt, M.J. [2001] Tenth SPE comparative solution project: A comparison of upscaling techniques. SPE-66599-MS.

Dogru, A.H. [2011] Giga-cell simulation. *Saudi Aramco Journal of Technology*, (Spring 2011), 2–8.

Du, S. et al. [2013] Multiscale grid design and upscaling for a full field tight gas reservoir model on a next generation simulator. *SPE Unconventional Gas Conference and Exhibition*.

Dzyuba, V.I., Bogachev, K.Y., Bogaty, A.S., Lyapin, A.R., Mirgasimov, A.R. and Semenko, A.E. [2012] Advances in modeling of giant reservoirs. *Mathematical Methods in Fluid Dynamics and Simulation of Giant Oil and Gas Reservoirs*, SPE-163090-MS.

Esler, K., Atan, S., Ramirez, B. and Natoli, V. [2011] Accelerating reservoir simulation with GPUs. *73rd EAGE Conference & Exhibition*.

Esler, K., Natoli, V. and Samardžić, A. [2012] GAMPACK (GPU Accelerated Algebraic Multigrid PACKage). *ECMOR XIII - 13th European Conference on the Mathematics of Oil Recovery*.

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

Fung, L.S. and Dogru, A.H. [2008] Parallel unstructured-solver methods for simulation of complex giant reservoirs. *SPE Journal*, **13**(04), 440–446.

Gratien, J.M., Guignon, T., Magras, J.F., Quandalle, P., Ricois, O.M. et al. [2007] Scalability and load balancing problems in parallel reservoir simulation. *SPE Reservoir Simulation Symposium*.

Islam, A. and Sepehrnoori, K. [2013] A review on SPE's comparitive solution projects (CSPs). *Journal of Petroleum Science Research*, **2**(4), 167–180.

Killough, J. [1995] Ninth SPE comparative solution project: a reexamination of black-oil simulation. *Symposium on reservoir simulation*, 135–147.

Klie, H.M., Sudan, H.H., Li, R. and Saad, Y. [2011] Exploiting capabilities of many core platforms in reservoir simulation. *SPE Reservoir Simulation Symposium*, SPE-141265-MS.

Micikevicius, P. [2009] 3d finite difference computation on gpus using cuda. *Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, 79–84.

Natvig, J.R. et al. [2011] Multiscale mimetic solvers for efficient streamline simulation of fractured reservoirs. *SPE Journal*, **16**(04), 880–888.

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

Wallis, J.R. [1983] Incomplete gaussian elimination as a preconditioning for generalized conjugate gradient acceleration. *SPE Reservoir Simulation Symposium*, SPE-12265-MS.

Wallis, J.R., Kendall, R.P. and Little, T.E. [1985] Constrained residual acceleration of conjugate residual methods. *SPE Reservoir Simulation Symposium*, SPE-13536-MS.

Yu, S., Liu, H., Chen, Z.J., Hsieh, B. and Shao, L. [2012] GPU-based parallel reservoir simulation for large-scale simulation problems. *SPE Europe/EAGE Annual Conference*, SPE-152271-MS.