EAGE

ECMOR XV

Mo Efe 10

GPU Acceleration of Equation of State Calculations in Compositional Reservoir Simulation

R. Gandham* (Stone Ridge Technology), K. Esler (Stone Ridge Technology), K. Mukundakrishnan (Stone Ridge Technology), Y.P. Zhang (Stone Ridge Technology), C. Fang (The University of Tulsa) & V. Natoli (Stone Ridge Technology)

SUMMARY

Equation-of-state (EOS) based compositional simulations accurately capture the dynamics of reservoirs with strong compositional effects. One of the major computational bottlenecks in such simulations is the need to enforce the phase equilibrium constraint for the hydrocarbon system for every grid block in the model. These constraints must be enforced at every time step and possibly, at every nonlinear iteration level within each time step for implicit methods. Hence, detailed simulations of models with many millions of cells and a large number of hydrocarbon components are prohibitively time-consuming. However, the high computational intensity and parallelism exhibited by these calculations make them ideal for significant acceleration using high throughput devices such as Graphics Processing Units (GPUs).

In this study, we propose new techniques for accelerating the EOS-based phase equilibrium calculations on the GPUs. First, we make full use of the large number of fast registers and floating point units available on GPUs for the double-precision arithmetic , thereby significantly accelerating the equilibrium calculations. Second, we exploit the fast hardware intrinsics available for single precision to further increase the performance. By iteratively combining the single and double-precision calculations, we not only achieve the full accuracy of double-precision but also gain an order-of-magnitude speedup over using double-precision arithmetic alone.

Accuracy and performance results from several benchmark problems available in the literature will be provided to demonstrate the speedup achieved using our proposed techniques. The performance results will then be compared with the recently published timings generated using highly optimized code on the CPUs. We will discuss the implications of such performance gains on the selection of implicit algorithms for the full compositional flow simulation.

Introduction

Oil and gas reservoirs for which the phase behavior of the fluids are far from the critical region of the hydrocarbon phase diagram, can be adequately described by a black-oil model. However, reservoirs with near-critical temperature or those with production enhanced via miscible injection can exhibit complex phase behavior which cannot be adequately described by a two-component model such as black-oil. For these models, a compositional description can provide greater predictive accuracy by directly accounting for the variation of compositions within the hydrocarbon phases, and the compositional dependence of thermodynamic and fluid properties.

In a compositional description, hydrocarbon components of different molecular weight and characteristics are individually tracked as they move through the subsurface. Depending on pressure and temperature, the hydrocarbon mixtures may segregate into two or more phases, in which the components are transported. The distribution of the component into phases is governed by the thermodynamic equilibrium of components in each phase. There are two common ways to describe this equilibrium: an equation-of-state (EOS) approach or a so-called K-value approach. In this paper, we focus on the more physically rigorous EOS-based approach.

This approach requires determining the stable phases and solving for the equilibrium composition of each such phase in every grid block for each time step in the simulation. This requirement introduces additional computational cost that can result in much longer run times than black-oil simulations. Hence, detailed simulations of models with many millions of cells and a large number of hydrocarbon components can be prohibitively time-consuming. A major goal in the industry has been to improve the technology to make large-scale compositional simulations more feasible in practice.

Fortunately, the equilibrium calculations for each cell are entirely independent of all other cells, which makes the computation very amenable to parallelization. This inherent parallelism, coupled with high computational intensity, makes these calculations ideally suited for acceleration using high throughput devices such as Graphics Processing Units (GPUs). We detail the methods we use to adapt known algorithms to the computational characteristics of GPUs. Validation and performance results from standard benchmark problems will be provided to demonstrate the speedup achieved using our proposed techniques. Finally, we discuss the implications this speedup has on the selection of the nonlinear solution methods for the overall compositional simulation, i.e. the coupled transport/phase equilibrium problem.

Background

The standard formulation for computing the phase equilibrium consists of a phase stability analysis step based on the minimization of the tangent plane distance (Michelsen (1982a)), followed by a flash procedure that uses the equality of fugacity for each component in different phases (Michelsen (1982b)). Strategies for reducing the computational time of the phase equilibrium calculations have traditionally focused on modifying the algorithmic aspects underlying these two steps (Michelsen et al. (2008)). Recently, a shadow region method has been proposed by Belkadi et al. (2013) that involves skipping the stability calculations in single phase regions and speeding up phase split calculations in two phase regions using information from the previous time steps . Other algorithmic approaches include the tie line-based Compositional Space Tabulation (CST) in which a table of converged calculations (tie-lines) is built to parameterize the compositional space (Voskov and Tchelepi (2008)). A simple table-lookup approach is subsequently used to replace the compute intense phase stability analysis. A more optimized modification of CST algorithm known as the Compositional Space Adaptive Tabulation (CSAT) method has been proposed by Voskov and Tchelepi (2009), where the table of tie-lines is adaptively updated during the course of simulation. Other class of algorithms to improve the computational efficiency of phase equilibrium calculations include the reduced variables method outlined in Hendriks (1988); Hendriks and van Bergen (1992); Firoozabadi and Pan (2002); Pan and Firoozabadi (2003); Nichita (2006). In these approaches, the calculations are carried out in a reduced basis obtained by a dominant eigenvalue decomposition of the binary interaction parameter matrix employing Lagrange multipliers.

The performance analysis of such methods have been studied in detail by (Haugen and Beckner (2011); Michelsen et al. (2013)).

In contrast to the large number of works that have focused on algorithmic modifications, much less attention has been paid to the computational implementation and optimization, which can have a huge impact on performance. Hardware characteristics and performance have significantly evolved in the last decade with the advent of multi- and many-core architectures. Haugen and Beckner (2013) achieve a significant speedup for conventional stability and flash calculations by making explicit use of the vector units available in Intel CPUs by using single-instruction-multiple-data (SIMD) intrinisics which map directly to the vector instructions. They further increase performance by utilizing mixed precision, hand-optimized transcendental functions, and improved data locality through tiled matrix operations. These modifications combine to provide a speedup of over $5\times$ relative to a reference scalar implementation.

Understanding algorithmic performance begins with identifying the bottlenecks which limit throughput. Generally, most computational methods are limited by either the maximum floating point throughput (*compute bound*) or by the rate of data transfer between the processor and memory (*memory-bandwidth bound*). For example, in a black-oil reservoir simulation, the computation is dominated by the solution of large, sparse linear systems. Since very few arithmetic operations are required per fetch from memory, performance is almost always bandwidth bound. On the other hand, the stability analysis and phase-split flash calculations in compositional simulations are dominated by dense linear algebra calculations and are compute bound.

In this context, current generation of GPUs provide a major performance advantage over CPUs in both floating point throughput and memory bandwidth. The speedups achieved with GPUs on bandwidth-bound black-oil simulations have been reported in Appleyard et al. (2011); Bayat et al. (2013) which focused only on accelerating the solvers and preconditioners. Using a fully-accelerated reservoir simulator built from the ground-up for GPUs, Esler et al. (2014); Mukundakrishnan et al. (2015) reported higher speedups on complex real-field assets over CPU-based simulators. In this study, we explore the acceleration that GPUs can provide for compute-bound equation-of-state phase stability and flash calculations.

Overview

In the sections below, we describe the approaches we have taken to extract maximal performance from GPUs for phase equilibrium calculations. We take full advantage of the fine-grain parallelism, fast register memory, constant memory, and in-hardware transcendental function evaluation offered by modern GPUs. We compare the computational performance of our algorithms with published results from a highly-optimized CPU algorithm (Haugen and Beckner (2013)) for compositions from selected SPE benchmark models.

The paper is organized as follows. First, we describe the mathematical formulation of the phase equilibrium problem. We next discuss our choice of algorithms, GPU parallelization, and hardware-specific optimizations. We then present performance timing and scaling results for SPE benchmark problems. Finally, we discuss the implications that extremely fast phase equilibrium calculations have on the selection of the overall nonlinear solution procedure for compositional reservoir simulation and its performance in general.

Methodology

In a compositional reservoir simulation, the partial differential equations governing the transport of components are coupled to the nonlinear algebraic constraints imposing thermodynamic equilibrium of components partitioned among different phases. There are many approaches to handling the coupling between the transport and equilibrium constraints, which can roughly be divided into two groups: 1) methods which decouple the constraints from the the transport equations in an explicit or a sequential implicit manner; 2) methods in which the transport and constraint equations are coupled in a monolithic

fashion and solved simultaneously. In this study, we have adopted the former approach and hence, at each time or Newton level, a well-posed set of thermodynamic constraints is provided for each cell to be solved.

Specifically, given the pressure, temperature, and overall component molar *feed* fractions, z_i, for the cell, we must first determine if the system exists as a single phase or multiple phases. Physically, this is determined by calculating the overall Gibbs free energies of a single phase system and that of a system partitioned into two or more phases but containing the same feed fraction z_i. A reduction in Gibbs free energy from a single to a multiple phase system implies an unstable single phase system while an increase implies a stable system. We refer to this stage of computation as the *stability check*. While a single phase system of hydrocarbon mixtures can yield more than two stable phases under appropriate conditions, we assume that at most a single liquid and a single vapor phase are present in our study.

Next, for cells with two stable phases (unstable single phase), we must determine the partitioning of the component moles or the overall mole fraction, z_i, between the liquid and vapor phases that yields the minimum Gibbs free energy. Equivalently, this can be considered as imposing the equality of the component *fugacities* in each phase.

$$f_i^l = f_i^v. \tag{1}$$

Here, f is the fugacity and the superscripts l and v, refer to the liquid and the vapor phases, respectively. The expressions for the liquid and vapor fugacities are given as follows

$$f_i^l = \varphi_i^l \, P \, x_i \tag{2}$$
$$f_i^v = \varphi_i^v \, P \, y_i, \tag{3}$$

where, P is the overall system pressure, φ_i^l and φ_i^v are the fugacity coefficients of liquid and vapor. Also, x_i and y_i are the phase mole fractions of component i in the liquid and vapor phases such that

$$z_i = \beta y_i + (1-\beta)\, x_i, \tag{4}$$

where, β denotes the vapor phase fraction. Given z_i, the molar partitioning among the phases which involves the determination of x_i, y_i, and β is referred to as the *flash calculation*, due to its connection with flash liberation experiments.

Stability checks

For the phase stability test, we use modified tangent plane distance criteria proposed by Michelsen et al. (2008). In this approach, the overall cell molar fractions, $\mathbf{z} = \{z_i\},\ i = 1, 2, .., N_c$, is assumed to exist in a single phase. Here, N_c is the number of components in the mixture. Then a trial phase that is opposite to the phase assumed is introduced with a molar composition $\mathbf{W} = \{W_i\}$. The modified tangent plane distance, tm, is then computed as follows.

$$tm(\mathbf{W}) = 1 + \sum_i W_i \left[\ln W_i + \ln \varphi_i(P, T, \mathbf{W}) - \ln z_i - \ln \varphi_i(P, T, \mathbf{z}) - 1\right], \tag{5}$$

The fugacity coefficient, φ, can be evaluated directly from the specified equation of state as a function of pressure, temperature, and the trial composition $\mathbf{W}$. While our implementation supports several standard forms of the EOS parameterization such as Peng-Robinson, Redlich-Kwong, and Soave-Redlich-Kwong, we focus on the Peng-Robinson model in this study for illustration purposes. For the Peng-Robinson model, given the component mole fractions, w_i, the fugacity coefficient φ_i is evaluated as follows.

$$\ln \varphi_i = \frac{B_i}{B}(Z-1) - \ln(Z-B) + \frac{A}{2\sqrt{2}B}\left(\frac{B_i}{B} - \frac{2}{A}\sum_{j=1}^{N_c} w_j A_{ij}\right) \ln\left[\frac{Z + (1+\sqrt{2})\,B}{Z - (1-\sqrt{2})\,B}\right]. \tag{6}$$

Here, Z is the compressibility factor of the assumed phase, which is defined in terms of the equation-of-state for the phase,

$$Pv = ZRT, \tag{7}$$

where P is the pressure, T the temperature, v the specific molar volume, and R the universal gas constant. Z is obtained by solving the following cubic equation of state based on the Peng-Robinson model,

$$Z^3 + c_2 Z^2 + c_1 Z + c_0 = 0, \tag{8}$$

where,

$$\begin{aligned} c_0 &= -AB + B^2 + B^3 \\ c_1 &= A - 3B^2 - 2B \\ c_2 &= B - 1. \end{aligned} \tag{9}$$

In general, there will be either one or three real roots for Z in Eq. 8, which can be expressed in closed form. In the case of three real roots, the largest is selected for the vapor phase, while the smallest is selected for the liquid phase.

The terms A and B appearing in the coefficients $c_0 - c_3$ of Eq. 9 are functions of w_i and are expressed as

$$\begin{aligned} A &= \sum_{i=1}^{N_c} \sum_{k=1}^{N_c} w_j w_k A_{jk} \\ B &= \sum_{i=1}^{N_c} w_j B_j \end{aligned} \tag{10}$$

Furthermore, the A_{ij} coefficients can be written in terms of the EOS model parameters as:

$$A_{ij} = (1 - \delta_{jk}) \sqrt{A_i A_j}, \tag{11}$$

where δ_{jk} is the specified binary-interaction coefficients (BIC) between components j and k. In addition, coefficients A_i and B_i in Eqns. 10 - 11 are given as

$$A_i = \Omega_a \frac{P_{ri}}{T_{ri}^2} \left[1 + m_i \left(1 - \sqrt{T_{ri}} \right) \right]^2 \tag{12}$$

$$B_i = \Omega_b \frac{P_{ri}}{T_{ri}}. \tag{13}$$

Here $P_{ri} = P/P_i^{\text{crit}}$ is the *reduced pressure* and $T_{ri} = T/T_i^{\text{crit}}$ is the *reduced temperature*. Ω_a and Ω_b are constants and the factor m_i is given by

$$m_i = 0.37464 + 1.54226\omega_i - 0.26992\omega_i^2, \omega_i \leq 0.49, \tag{14}$$

$$= 0.379642 + 1.48503\omega_i - 0.164423\omega_i^2 + 0.016666\omega_i^3, \omega_i > 0.49. \tag{15}$$

In Eq. 15, ω_i is the specified "acentric" factor of the component i.

A second phase will be stable if there exists a trial composition $\mathbf{W}$ for which the tangent plane distance is negative. In principle, an exhaustive search of the valid composition space is required to rigorously establish single-phase stability. In practice, we perform a search for a minimum of tm starting from the Wilson estimate of the equilibrium ratio, K_i. The test is performed first assuming that the feed phase is a liquid and the trial phase is a vapor. The test is then repeated with opposite phase assignments. If either of these tests indicates a stable second phase, then both a liquid and a vapor phase will be present at the given conditions.

Flash calculations

For the two-phase flash calculations, we solve for the equality of phase fugacities given by Eq. 1, which is expressed in terms of fugacity coefficients through Eqns. 2, 3 as,

$$\ln K_i + \ln \varphi_i^v - \ln \varphi_i^l = 0, \qquad i = 1, 2, .., N_c. \tag{16}$$

Here, K_i is the equilibrium ratio given by

$$K_i = \frac{x_i}{y_i}. \tag{17}$$

In Eq. 16, $\ln \varphi_i^v$ and $\ln \varphi_i^l$ are evaluated using Eqns. 6 - 15. For known values of K_i and component feed fractions z_i, the vapor phase fraction β is solved using a modified form of Eq. 4 and is given by,

$$\sum_{i=1}^{N_c} \frac{z_i(K_i - 1)}{1 + \beta(K_i - 1)} = 0. \tag{18}$$

Eq. 18 is also known as the Rachford-Rice equation.

Eqns. 5, 16 and 18 constitute a set of highly nonlinear and coupled set of equations that needs to be solved efficiently for each cell during every timestep and possibly for every nonlinear iteration. Furthermore, the stability check and phase equilibrium flash calculations given by Eqns. 6 - 18, require the evaluation of several transcendental functions, including `log` (logarithmic), `sqrt` (square root), and `cbrt` (cube root) together with other arithmetic operations such as double summations over components (Eq. 10), for every component in each cell. These calculations are compute-intense and hence, are *compute bound*.

Numerical algorithms

Both the stability and flash calculations can be cast as a minimization procedure or as a set of nonlinear algebraic equations. Three methods are commonly employed to solve these equations. The simplest of these is successive substitution (SS), which is an unconditionally stable iteration procedure. For fluids far from the critical point it is often rapidly convergent, but becomes increasingly ineffective as the critical point is approached. SS involves the following procedure: for a given k^{th} iteration-level solution, $k+1^{th}$ iteration-level solution for the minimization of Eq. 5 is obtained as follows:

$$\ln(W_i^{k+1}) = \ln z_i + \ln \varphi_i(P, T, \mathbf{z}) - \ln \varphi_i(P, T, W^k), \qquad i = 1, 2, .., N_c. \tag{19}$$

The generalized dominant eigenvalue method (GDEM) is a modification to the successive substitution approach. It uses previous SS iterations to estimate the lowest eigenvalue(s) for the iteration. This estimate is then used to approximately extrapolate the iterative solution to infinite iteration count.

$$\ln(W_i^{\infty}) = \ln(W_i^k) + \frac{1}{1-\lambda}\Delta_i^{k+1}, \tag{20}$$

where,

$$\Delta_i^{k+1} = \ln(W_i^{k+1}) - \ln(W_i^k) \tag{21}$$

$$\lambda = \frac{\left(\Delta^k\right)^T \left(\Delta^{k+1}\right)}{\left(\Delta^k\right)^T \left(\Delta^k\right)} \tag{22}$$

This extrapolation generally accelerates convergence, but also invalidates the unconditional stability of successive substitution, requiring additional checks on updates. Furthermore, since the eigenvector estimates involve subtracting successive iterates (Eq. 21) which may differ by only a very small amount, the eigenvalue estimates can be extremely sensitive to floating point truncation error.

Finally, a Newton-Raphson (NR) iteration procedure can be employed to solve the nonlinear system given by Eqns. 16 and 18. In our study, we have used a modified version of Eqns. 16 and 18 as suggested by Michelsen et al. (2008). The details of the choice of the modified system of variables, the resulting residual vector, and the linearized Jacobian matrix are all provided in Michelsen et al. (2008) and are not provided here for brevity. Since the resulting $N_c \times N_c$ Jacobian matrix is usually small, the linear system can be solved efficiently with a dense solution method, e.g. Gaussian elimination.

GPU parallelization

On modern processors, fetching data from DRAM involves latencies on the order of hundreds of clock cycles. For frequently reused data, this latency can be avoided through on-chip caching, which is present in both CPUs and GPUs, but the cache sizes are generally much larger on CPUs. This caching is implemented in hardware and outside direct software control. In addition to the hardware-controlled cache, GPUs also provide a programmer-controlled on-chip *shared memory* storage and a much larger *register file* than CPUs. Since all data must reside in registers before arithmetic operations can be applied, keeping data present in registers results in the highest possible performance. Although the register file is relatively large on GPUs, it is still limited, and on current GPU hardware, a single thread can access a maximum of 255 32-bit registers per execution kernel. Thus, a critical focus of our implementation is to determine how to make best use of this resource.

Work assignment

In parallelizing any algorithm, one must decide how to apportion work to available computing resources. In the context of EOS calculations, there are two natural approaches to this assignment. Since the work for each cell in independent of all others, it is natural to assign a GPU thread of execution to each cell. This allows writing essentially sequential code for each cell and operating in parallel. This approach avoids any potential overhead from communicating data between threads. However, as the number of components employed in the simulation increases, the working data set for each cell increases in size, and it becomes impossible to simultaneously store all required variables in registers. Thus, data may have to be stored and retrieved repeatedly to/from DRAM, decreasing performance.

In this case, the use of a finer-grained parallelism can help reduce the per-thread register requirements. For a fluid with N_c components, N_c threads can be assigned to work collaboratively on each cell, so that exceeding the maximum number of registers can be avoided. However, exchange of data between threads then becomes necessary. Furthermore, there is some inherently sequential computation required for each cell which cannot be parallelized among the N_c threads, decreasing parallel efficiency. Thus, neither the one-thread-per-cell nor the N_c-threads-per-cell approach is ideal in all circumstances.

We have experimented with both the thread-per-cell and thread-per-component approaches on the EOS algorithms. In general, we find that the per-cell thread assignment is faster than the per-component approach with SS and GDEM. This is likely because a large part of these calculations are inherently sequential. In addition, if carefully programmed, the working data set size can be held in registers for a reasonable number of components. In contrast, the Newton-Raphson approach has a register requirement that grows as N_c^2. Thus, the required register count for the Jacobian storage quickly overwhelms the maximum. Thus, we find the per-component thread assignment preferable for NR, particularly for large N_c.

Mixed precision

Traditionally, most scientific and engineering applications have utilized double-precision floating point arithmetic exclusively to avoid any possibility of truncation error corrupting results. However, modern CPUs and GPUs generally provide higher performance in single-precision. For example, NVIDIA K80 has 2.9 TFLOPS peak double precision performance, while the peak single precision performance is approximately $3\times$ higher at 8.74 TFLOPS. This performance difference has motivated the development of *mixed-precision* algorithms which, when converged, can attain the same final accuracy of fully double-precision calculations, but at a reduced computational cost.

The performance benefits of mixed-precision on GPUs is further enhanced by the presence of dedicated *special function units*, which evaluate some transcendental function very quickly in hardware. However, these instructions are available only in single-precision. As a result, the throughput for these functions in single precision can be more than an order of magnitude higher than in double precision. Fig 1, illustrates the performance gap between single and double precision for several transcendental functions utilized in

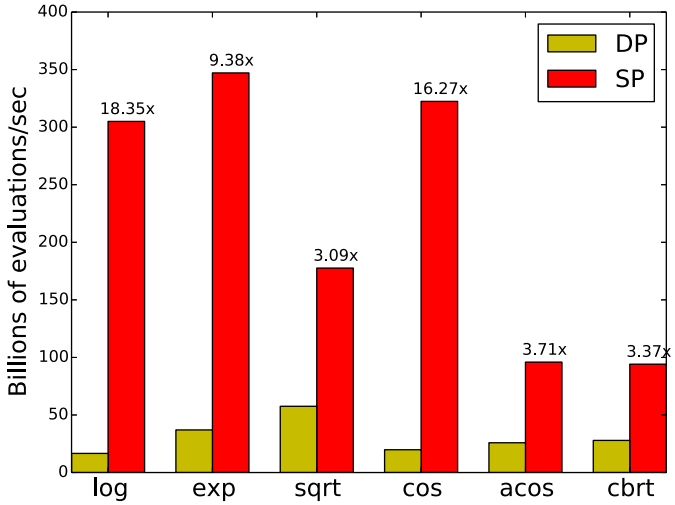


Figure 1 *Micro benchmark, performance of single precision vs double precision transcedentals on NVIDIA K80 GPU. The labels indicate the speedup achieved with single precision computations compared to double precision computations.*

the present work. For those functions which have hardware implementations for single precision, including `log` (logarithmic), `exp` (exponential), and `cos` (cosine), the gap is an order-of-magnitude or more. For other functions `sqrt` (square root), `acos` (inverse cosine), and `cbrt` (cube root) the performance gap is about $3\times$, which can be directly attributed to the ratio of peak floating-point performance.

In addition to the throughput, the single precision computations require half the on-chip resources (registers and shared memory) per variable as in double-precision. This allows a compositions with larger number of components to fit entirely in the on-chip storage without spilling over into DRAM. Such *register spilling* can result in a significant performance penalty. Table 1 gives the number of successive substitution iterations performed per second in single and double-precision, and demonstrates the marked gap in performance between the two. To take advantage of the superior single precision perfor-

	Double precision (M iteration/s)	Single precision (M iteration/s)	SP / DP ratio
Stability checks	398	2500	6.3
Flash calculations	121	837	6.9

Table 1: *Performance of successive substitution iterations on one NVIDIA K80 GPU.*

mance while retaining full accuracy, we devise a mixed-precision approach. The key idea is to perform a majority of computations in single precision and use double precision arithmetic only to refine the solution to the desired accuracy.

In one-sided stability checks, the stability is decided based on the sign of tangent plane distance at the converged solution. Because the result of the check depends only on the sign of this distance, in most cases the stability can be inferred directly from the single-precision calculations. However, in cases in which the magnitude of the computed tangent plane distance is of the same order as single-precision truncation error, we must proceed to converge the distance in double-precision in that cell to ensure we have the correct sign. We use a conservative value of 10^{-5} for the relative single-precision truncation error. For the test problems we have explored, double-precision checks are required for only 1-2% of the cells. Tolerances for residuals in the logarithm of the fugacity are set to 10^{-4} and 10^{-10} for single and double precision respectively.

In flash calculations, we estimate the component fractions using a few iterations of successive substitution in single precision. These estimates are then used as an initial guess and a few GDEM iterations are performed in double precision. For the cells that are still not converged, the solution procedure is completed using Newton's method with the solution obtained from GDEM as the initial guess. Newton's method significantly improves the convergence near critical regions.

For the mentioned problem, the number of double precision iterations is about a third of the number of single precision iterations. It is important to note that double precision iteration(s) are required for all two phase cells here, unlike the stability checks. This is because the convergence tolerance is set to 10^{-10}. We observe a speedup of $3\times$ using this mixed precision computations compared to using full double precision computation for this problem.

Though the convergence properties of Newton's method are superior to GDEM, the overall performance is better when we use a few GDEM to give a good initial estimate for the NR iteration. This is because one NR iteration is an order of magnitude slower than one GDEM iteration on a K80 GPU. On CPUs, however, an NR iteration is only about twice the cost of a GDEM iteration. For this reason, Michelsen et al. (2008) suggest using NR iteration alone on CPUs.

To optimize the overall performance of the equilibrium calculations, we experimented with the number of iterations chosen for each method. We present the overall algorithm 1, that resulted in optimal performance on GPU for various problems.

Algorithm 1 *Overall phase stability/equilibrium procedure*

Stability checks:
for all cells in parallel **do**
  Perform stability checks in single precision using 100 successive substitution iterations.
  Identify whether each cell is two phase, one phase, or the tests are inconclusive.
end for
Create a list of cells for which the tests are inconclusive.
for all inconclusive cells in parallel **do**
  Perform stability checks in double precision using GDEM.
end for

Flash calcuations:
Create a list of two phase cells.
for all two phase cells in parallel **do**
  Perform 50 iterations of successive substitution in single precision (Eq. 19).
end for
for all two phase cells in parallel **do**
  Perform 6 iterations of GDEM with one acceleration step and check for convergence (Eqns. 20 - 22).
end for
for all unconverged cells in parallel **do**
  Perform Newton iterations until convergence.
end for

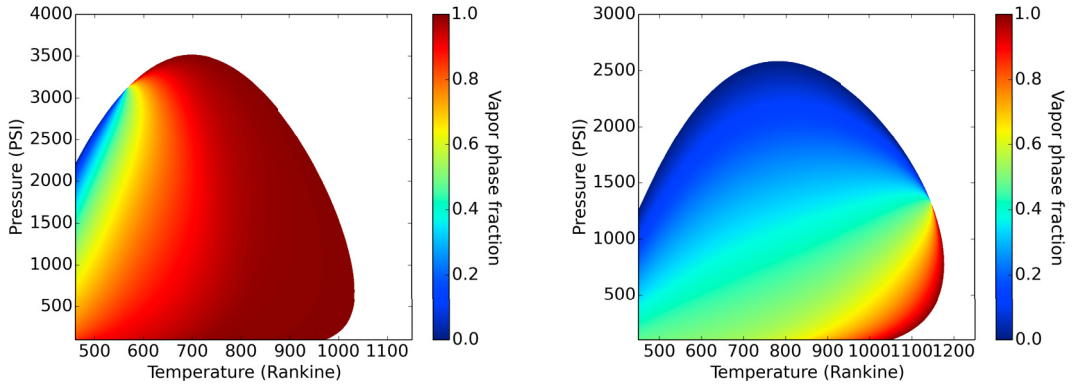
Results

We consider two representative test problems with fluid compositions taken from the Third and Fifth SPE Comparative Solution Projects to validate and evaluate the performance of our algorithm implementations. In each test, we consider a hydrocarbon mixture with fixed composition and evaluate the phase stability and equilibrium composition for a wide range of temperatures and pressure, i.e. generate Pressure-Temperature (PT) diagram of phase behavior. This is done via creating a Cartesian grid of

1000×1000 blocks, each grid block assuming a unique combination of uniformly distributed pressure and temperature in a chosen range.

The ranges are chosen to encompass the majority of the phase envelope in order to include points in both single and two-phase regions, near and far from the critical point. This allows us to test not only the robustness of the algorithms, but also the dependence of performance on the region of the PT diagram. These same problems have been considered by Haugen et al. (2013), which provide a reference for both the correctness and performance of the present work.

For SPE3 benchmark, the mixture composition is taken from Kenyon (1987). In particular, we use the 9-component parametrization of the fluid provided by Arco, so as to be directly comparable to the results provided by Haugen et al. (2013). For SPE5 benchmark, the fixed composition of 6-component mixture is taken from Killough et al. (1987). In both cases, we use Peng-Robinson (PR) cubic equation of state (Peng and Robinson (1976)). Fig 2a and Fig 2b show the variation of vapor phase fractions of SPE3 and SPE5 mixtures over a range of pressure and temperature, obtained by our implementations. These are in excellent agreement with published results.



(a) *SPE3 9-component mixture* (b) *SPE5 6-component mixture*

Figure 2 *Vapor phase fraction vs temperature (on horizontal axis) and pressure (on vertical axis).*

We study the performance scaling with respect to the number of components by generating additional components without affecting the phase behavior. In particular, the mole fraction of the heaviest pseudo-component is divided into n equal components with identical properties (accentric factor, binary interaction coefficients, critical temperature and pressure). By cloning components in this way, the resulting composition has identical phase behavior as the original mixture. This approach, adopted from Haugen et al. (2013), allows us to directly measure the scaling of the algorithms with component number.

To make the CPU/GPU comparison as fair as possible, we also provide results from our own implementation of the same algorithms on CPU. To avoid a straw-man comparison, we make full use of the SIMD vector capabilities of modern CPUs through the use of the C++ Vector Class Library (VCL) (Fog (2012-2016)). This library provides primitive SIMD vector types and associated arithmetic operators that allow one to write explicitly vectorized functions without resorting to assembly or low-level intrinsics. In addition, the library provides optimized SIMD implementations for common transcendental functions.

The CPU tests given below were run on a system with two Intel Haswell architecture CPUs, Xeon E5-2630 v3, each with eight cores clocked at 2.4 GHz. Each of these cores contains two 256-bit vector units supporting the *AVX2* instruction set. These instructions allow the same arithmetic operation to be simultaneously applied to four double-precision or eight single-precision values. The CPUs are also backwards compatible with the older 128-bit SSE4.2 instruction set, which has half the vector width and thus roughly half the peak throughput. The GPU results were generated on an NVIDIA Tesla K80 board,

which contains two GK210 GPUs. For simplicity, only one CPU socket or one GPU were utilized in these tests.

We first compare our CPU performance results with the timings published in Haugen et al. (2013) in Table 2 on the SPE3 benchmark problem. For a direct comparision, the results are obtained using a single CPU core. The first column shows the number of components in the mixture, while the second and third column show the overall compute time when AVX2 and SSE4.2 vector instructions, respectively, are used. The last column gives the published timings, which employed SSE4.2 instructions on the same problem.

N_c	Present AVX2 (s)	Present SSE4.2 (s)	Haugen et al. SSE4.2 (s)
10	9.1	12.1	9.2
15	14.7	20.0	14.7
20	21.2	29.0	22.5
25	28.7	39.0	34.6
30	37.4	50.0	49.0

Table 2 *CPU performance: Overall compute time of equilibrium calculations for SPE3 benchmark on a single Intel CPU core.*

The performance of our CPU implementation is slightly lower but comparable to the published results. This is due to the differences in choice of algorithms, implementation, and possibly hardware – details for the processor used in the previous study were not provided. We observe a speedup of 30% with AVX2 instructions when compared to SSE4.2 instruction set. In principle, we might expect up to a $2\times$ speed advantage, since AVX2 provides twice the vector width, but this is not realized in practice.

We compare the GPU and CPU performances using both the test cases in Table 3. In this table, the first column shows the number of components in the mixture, the second column shows the overall compute time using GPU, the third column shows the overall compute time using one CPU socket with AVX2 vector instructions, and the last column shows the speedup achieved with our GPU implementations over our CPU implementations.

Test case	N_c	GPU (s)	AVX2 (s)	Speedup
SPE3	10	0.12	1.52	12.7
	15	0.23	2.46	10.7
	20	0.49	3.55	7.2
	25	0.77	4.81	6.2
	30	1.16	6.26	5.4
SPE5	6	0.05	0.85	17.0
	10	0.11	1.42	12.9
	15	0.21	2.26	10.8
	20	0.38	3.18	8.4
	25	0.63	4.27	6.8
	30	0.97	5.60	5.8

Table 3 *GPU vs CPU performance: overall compute time for equilibrium calculations on an NVIDIA K80 and an 8-core Intel CPU.*

The overall speedup ranges from 5 (for 30 components) to 17 (for 6 components). Whereas the ratio of peak double precision performance is roughly 5 (K80 peak performance is 1.45 TFLOPS, CPU peak performance is about 300 GFLOPS). As the number of components increases, the number of registers required per thread increases. When this exceeds the maximum number per thread supported by the hardware (currently 255), the compiler is forced to resort to GPU DRAM for storage, which reduces performance. In contrast, the 32 KB L1 data cache per CPU core is large enough to handle the full working data sets for the problems considered. Hence, the CPU's scaling ($\mathcal{O}(N_c^{1.3})$) is better than the

GPU's ($\mathcal{O}(N_c^2)$) in our implementations. Nonetheless, the absolute performance is still significantly higher on GPU for all the numbers of components considered in this study.

The presented performance results are obtained assuming that there is no *a priori* knowledge of the phase composition. While running a real simulation, however, we can utilize the information available from previous time steps. In regions in which the time evolution of the pressure and composition in each cell is slow, the composition from previous time step is very close to the phase composition at current time. Utilizing this information as initial guess can often dramatically reduce the number iterations required to converge the stability checks and flash calculations. Other algorithmic acceleration methods, such as the shadow region and tie-line based tabulation methods mentioned earlier, can also be combined with the computational methods described in this paper, potentially with multiplicative gains in performance.

Conclusions

We presented an extremely fast GPU-accelerated method for phase equilibrium calculations of multi component systems in Reservoir simulations. The speedups range from $6\times$ to $17\times$ compared to optimized multicore CPU implementations. For fluids with 15 or fewer components, the GPU implementation achieves more than an order of magnitude speedup. These significant speedups are due to high throughput of GPUs, carefully designed implementations for fine-grain parallelism which take full advantage of the fast registers, fast constant memory, and hardware intrinsics available for single precision transcendentals.

The performance advantages provided by modern GPUs are continuously increasing. The recently introduced NVIDIA Tesla P100 GPU (code-named Pascal) promises to provide an 80% improvement in double precision compute performance and a 50% increment in memory bandwidth relative to the Tesla K80 used in this study, (NVIDIA Corporation (2016)). Conservatively, we expect these hardware advances to provide at least a 60% performance improvement over the GPU results presented in this paper.

Implications for the choice of the nonlinear solution scheme

The dominant algorithms for the implicit solution of the EOS-based compositional flow equations were first developed during a period in which floating-point throughput was almost always the main performance limiter. Solution schemes were thus developed to converge nonlinear equations with a minimum of floating point operations. For the hardware platforms of the time, algorithms such as the one described in (Coats (1980)) often provided the most rapid convergence by combining the phase equilibrium constraint with the component transport equations into a monolithic linear system. Such monolithic schemes avoided the separate phase equilibrium calculations and often minimized the total number of Newton iterations.

On modern computing platforms constrained by power considerations, floating point operations are comparatively cheap relative to movement of data from DRAM. Monolithic linearization requires the solution of a very large sparse linear system with storage and bandwidth requirements that grow as $\mathcal{O}(N_c^2)$. In contrast, nonlinear solution schemes with phase stability checks and flash calculations that are decoupled from the component transport may avoid a large part of the storage and bandwidth requirement if formulated properly. As the ratio of FLOPS to bandwidth provided by modern processors continues to grow, such methods may merit investigation.

Future work

We believe that there is room for further improving the performance of the algorithms presented here. Algorithmically, we will investigate the use of Newton's method for stability checks to improve the rate of convergence near critical regions and to provide a better estimate for the equilibrium ratios used to initiate subsequent flash calculations. We will also explore a mixed-precision Newton iteration scheme in which the Jacobian is computed in single precision while the residual is retained in double precision.

In future work, the fast phase equilibrium calculations described in this paper will be combined with GPU-accelerated algorithms for the transport equations, leveraging methods already developed for a fully-accelerated black-oil simulator (Esler et al. (2014); Mukundakrishnan et al. (2015)). By minimizing memory bandwidth requirements, we believe that a significant performance gain can be attained relative to existing implicit compositional simulation methods, beyond that provided by the GPU hardware alone. The combined speedup provided by this advance in algorithms and GPU hardware could make the compositional simulation of detailed reservoir models with many millions of cells not just feasible but practical for everyday industry application.

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advance Scientific Computing Research, under Award Number DE-SC0015214. Additional funding has also been provided by the Marathon Oil Corporation.

Notation

EOS	Equation of State
SS	Successive Substitution
GDEM	Generalized Dominant Eigenvalue Method
NR	Newton Rapson
GPU	Graphics Processing Unit
CPU	Central Processing Unit
SIMD	Single Instruction Multiple Data
SSE	Streaming SIMD Extensions
AVX	Advanced Vector Extensions
SP	Single Precision
DP	Double Precision
VCL	Vector Class Library
GFLOPS	Giga FLoating point Operations Per Second
TFLOPS	Tera FLoating point Operations Per Second
DRAM	Dynamic Random Access Memory
N_c	Number of components
SPE	Society for Petroleum Engineers

References

Appleyard, J., Appleyard, J., Wakefield, M., Desitter, A. et al. [2011] Accelerating reservoir simulators using GPU technology. In: *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers.

Bayat, M., Killough, J. et al. [2013] An experimental study of GPU acceleration for reservoir simulation. In: *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers.

Belkadi, A., Yan, W., Moggia, E., Michelsen, M.L., Stenby, E.H., Aavatsmark, I., Vignati, E., Cominelli, A. et al. [2013] Speeding up compositional reservoir simulation through an efficient implementation of phase equilibrium calculation. In: *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers.

Coats, K.H. [1980] An Equation of State Compositional Model. *Society of Petroleum Engineers Journal*, **20**(05), 363–376.

Esler, K., Mukundakrishnan, K., Natoli, V., Shumway, J., Zhang, Y. and Gilman, J. [2014] Realizing the Potential of GPUs for Reservoir Simulation. In: *ECMOR XIV-14th European conference on the mathematics of oil recovery*.

Firoozabadi, A. and Pan, H. [2002] Fast and Robust Algorithm for Compositional Modeling: Part I - Stability Analysis Testing. *SPE Journal*, **7**(01), 78–89.

Fog, A. [2012-2016] C++ vector class library. `http://www.agner.org/optimize/#vectorclass`.

Haugen, K.B. and Beckner, B.L. [2011] Are Reduced Methods For EOS Calculations Worth The Effort? Society of Petroleum Engineers.

Haugen, K.B. and Beckner, B.L. [2013] Highly Optimized Phase Equilibrium Calculations. Society of Petroleum Engineers.

Haugen, K.B., Beckner, B.L. et al. [2013] Highly optimized phase equilibrium calculations. In: *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers.

Hendriks, E. and van Bergen, A. [1992] Application of a reduction method to phase equilibria calculations. *Fluid Phase Equilibria*, **74**, 17–34.

Hendriks, E.M. [1988] Reduction theorem for phase equilibrium problems. *Industrial & Engineering Chemistry Research*, **27**(9), 1728–1732.

Kenyon, D. [1987] Third SPE Comparative Solution Project: Gas Cycling of Retrograde Condensate Reservoirs. *Journal of Petroleum Technology*, **39**(08), 981–997.

Killough, J., Kossack, C. et al. [1987] Fifth comparative solution project: evaluation of miscible flood simulators. In: *SPE Symposium on Reservoir Simulation*. Society of Petroleum Engineers.

Michelsen, M., Yan, W. and Stenby, E.H. [2013] A Comparative Study of Reduced-Variables-Based Flash and Conventional Flash. *SPE Journal*, **18**(05), 952–959.

Michelsen, M.L. [1982a] The isothermal flash problem. Part I. Stability. *Fluid Phase Equilibria*, **9**(1), 1–19.

Michelsen, M.L. [1982b] The isothermal flash problem. Part II. Phase-split calculation. *Fluid Phase Equilibria*, **9**(1), 21–40.

Michelsen, M.L., Mollerup, J. and Breil, M.P. [2008] Thermodynamic Models: Fundamental & Computational Aspects. In: *Thermodynamic Models: Fundamental & Computational Aspects*, Tie-Line Publications.

Mukundakrishnan, K., Esler, K., Dembeck, D., Natoli, V., Shumway, J., Zhang, Y., Gilman, J., Meng, H. et al. [2015] Accelerating Tight Reservoir Workflows With GPUs. In: *SPE Reservoir Simulation Symposium*. Society of Petroleum Engineers.

Nichita, D.V. [2006] A reduction method for phase equilibrium calculations with cubic equations of state. *Brazilian Journal of Chemical Engineering*, **23**(3), 427–434.

NVIDIA Corporation [2016] NVIDIA Tesla P100: The Most Advanced Datacenter Accelerator Ever Built. Tech. Rep. WP-08019-001_v01.1, NVIDIA Corporation.

Pan, H. and Firoozabadi, A. [2003] Fast and Robust Algorithm for Compositional Modeling: Part II - Two-Phase Flash Computations. *SPE Journal*, **8**(04), 380–391.

Peng, D.Y. and Robinson, D.B. [1976] A New Two-Constant Equation of State. *Industrial & Engineering Chemistry Fundamentals*, **15**(1), 59–64.

Voskov, D. and Tchelepi, H.A. [2008] Compositional space parametrization for miscible displacement simulation. *Transport in Porous Media*, **75**(1), 111–128.

Voskov, D.V. and Tchelepi, H.A. [2009] Compositional space parameterization: theory and application for immiscible displacements. *SPE Journal*, **14**(03), 431–440.