Parallelization of a Commercial Streamline Simulator and Performance on Practical Models

R.P. Batycky, SPE, Streamsim Technologies; **M. Förster,** Fraunhofer's Scientific Computing Institute; **M.R. Thiele,** SPE, Streamsim Technologies/Stanford University; and **K. Stüben,** Fraunhofer's Scientific Computing Institute

Summary

We present the parallelization of a commercial streamline simulator to multicore architectures based on the OpenMP programming model and its performance on various field examples. This work is a continuation of recent work by Gerritsen et al. (2009) in which a research streamline simulator was extended to parallel execution.

We identified that the streamline-transport step represents approximately 40–80% of the total run time. It is exactly this step that is straightforward to parallelize owing to the independent solution of each streamline that is at the heart of streamline simulation. Because we are working with an existing large serial code, we used specialty software to quickly and easily identify variables that required particular handling for implementing the parallel extension. Minimal rewrite to existing code was required to extend the streamline-transport step to OpenMP. As part of this work, we also parallelized additional run-time code, including the gravity-line solver and some simple routines required for constructing the pressure matrix. Overall, the run-time fraction of code parallelized ranged from 0.50 to 0.83, depending on the transport physics being considered.

We tested our parallel simulator on a variety of large models including SPE 10, Forties–a UK oil/water model, Judy Creek–a Canadian waterflood/water-alternating-gas (WAG) model, and a South American black-oil model. We noted overall speedup factors from 1.8 to 3.3x for eight threads. In terms of real time, this implies that large-scale streamline simulation models as tested here can be simulated in less than 4 hours. We found speedup results to be reasonable when compared with Amdahl's ideal scaling law. Beyond eight threads, we observed minimal speedups because of memory bandwidth limits on our test machine.

Introduction

Reservoir simulation models are routinely limited in size because of run-time and memory constraints. Yet modern reservoir engineering workflows for history matching, uncertainty assessment, and optimizing forecast scenarios rely on the ability to run large, finely gridded models multiple times. For some of these workflows, such as ensemble history matching or ranking/screening multiple realizations, it is acceptable to run multiple models in parallel with a serial simulator on a computer cluster. However, for the majority of practicing engineers, the reality is that desktop hardware is their only compute platform, they have a limited number of models to work with (usually one or two), and they are doing well-level history-matching or forecast scenarios one run at a time.

Recent advances in desktop hardware have made multicore-CPU shared-memory workstations the preferred computing environment for today's reservoir engineers, with workstations having four to 16 cores. Simulators that are parallelized by means of OpenMP directives can take advantage of these shared-memory systems, speeding up any serial workflow. All major commercial finite-difference (FD) simulators are available with a parallel compute option, either OpenMP and/or MPI (Collins et al. 2003; DeBaun et al. 2005; Shiralkar et al. 2005), to take advantage of this hardware.

Commercial streamline-based simulators (SLSs) have traditionally been less mature than FD methods because they have been introduced only within the last 10 years. SLSs are not readily available for parallel computing, meaning today's engineers cannot take advantage of the latest multicore hardware developments. However, because SLS is ideally suited for simulating large, geologically heterogeneous models, quantifying upscaling processes, and/or history matching, a parallel extension is highly desirable. Fortunately, the SLS-architecture is inherently parallelizable. Recent work on a research streamline simulator presented promising results (Gerritsen et al. 2009). Their work focused on parallelization of the transport step only in which they quote speedups of approximately 6x for eight threads (AMD™ chip) and 13.5x for 16 threads (SUN™ chip) for a large, synthetic reservoir model. Gerritsen et al. (2009) addressed issues such as load balancing of the streamlines and testing performance on different hardware architectures.

The main purpose of this paper is to build on the results of Gerritsen et al. (2009) and show the extension of an existing serial commercial streamline simulator (Streamsim 2008) to a parallel environment. The challenge we faced was how to migrate the existing serial infrastructure to OpenMP while minimizing code-rewrite, and to see how this adapted code would perform on real reservoir models. One critical difference between this work that of Gerristen at al. is that it is not possible to predetermine the work per streamline, meaning that there is no effective way to load balance the streamlines. Rather, we let the compiler determine the load balancing.

Overview of Streamline Simulation

Here, we briefly review the key steps in streamline simulation and show why the method is inherently suitable to parallelization. For a detailed discussion on the mathematics of the streamline method, see Batycky et al. (1997), Thiele (2005), or Datta-Gupta and King (2007). For simplicity, we have assumed incompressible flow in this brief review because fluid compressibility does not change which regions of the simulator to parallelize. Fluids compressibility does, however, have an impact on parallel performance, as we will show later.

In SLSs at each timestep, pressure is solved for implicitly while saturations (or compositions) are solved for explicitly. Streamline simulation can be viewed as a dual-grid approach because the implicit pressure solve is computed on a static cell-based (Eulerian) grid while saturations are computed on a dynamically changing flow-based (Lagrangian) grid defined by streamlines. Thus, there are two mapping steps involved: from the cell-based grid to the streamlines at the beginning of the timestep and then from the streamlines back to the cell-based grid at the end of the timestep **(Fig. 1).**

The implicit pressure solver is similar to that used in conventional FD methods. An implicit pressure matrix is constructed on the basis of a multiphase-flow pressure equation and the volume-balance method outlined by Ács et al. (1985) and Watts (1986). For incompressible systems with no capillary pressure, the volume-balance equation summed over all phases (n_p) reduces to

Copyright © 2010 Society of Petroleum Engineers

This paper (SPE 118684) was accepted for presentation at the SPE Reservoir Simulation Symposium, The Woodlands, Texas, USA, 2–4 February 2009 and revised for publication. Original manuscript received for review 3 November 2008. Revised manuscript received for review 19 June. Paper peer approved 29 July 2009.

SPE_REE_118684_100030.indd 1 5/15/10 10:14:37 AM

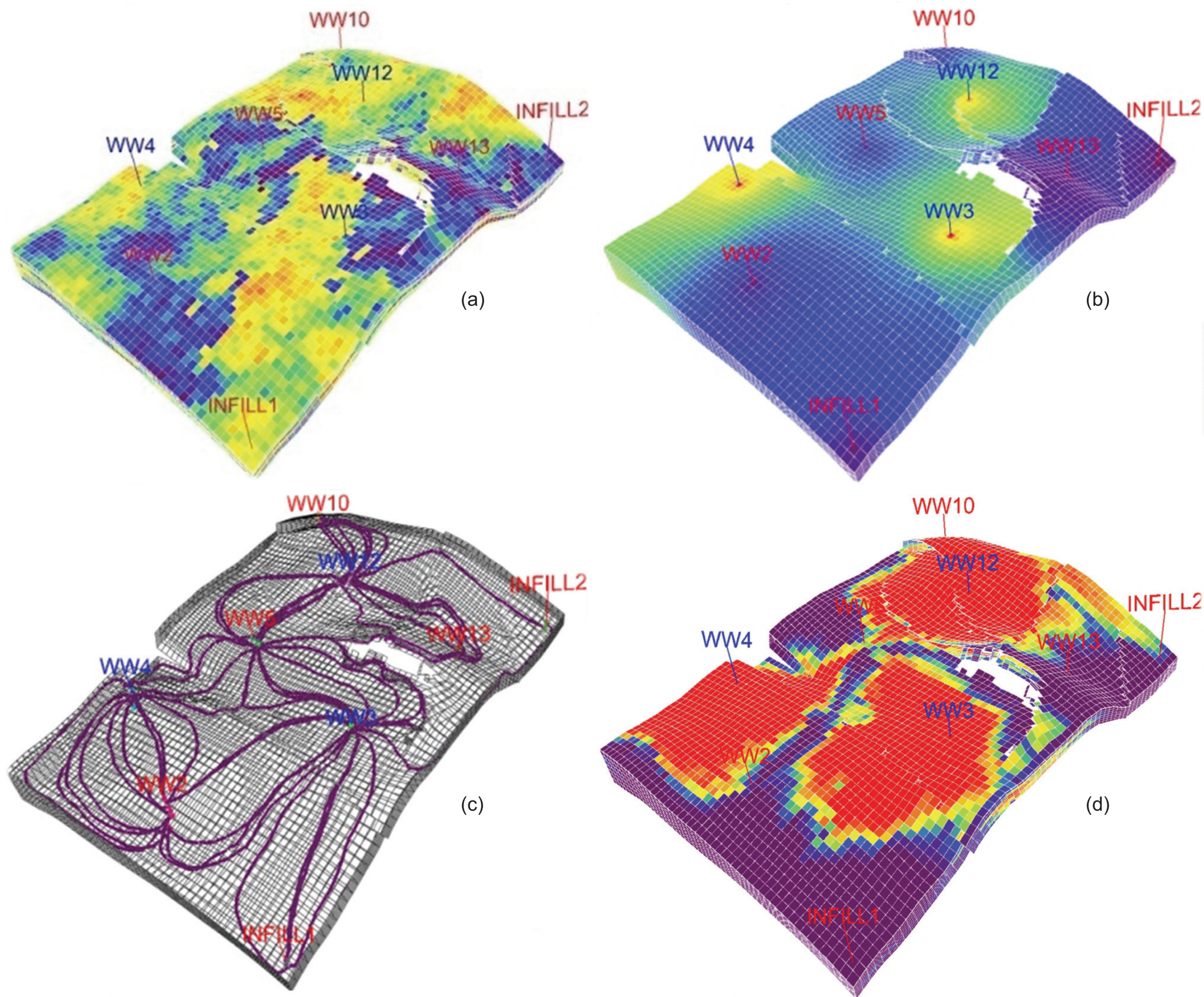


Fig. 1—Static properties, well locations, and initial conditions (a) are used to solve the pressure field implicitly (b). The velocity field is then computed from the pressures, streamlines are traced, the 1D transport equations are solved along each streamline (c), and the results are mapped back to the static grid (d).

$$\sum_{j=1}^{n_p} \nabla \cdot \overrightarrow{\overrightarrow{K}} \cdot (\lambda_j \nabla P + g\rho_j \lambda_j \nabla D) = 0, \quad \text{.......... (1)}$$

where the phase j mobility is given by λ_j, phase density ρ_j, permeability tensor K, depth D, gravitational acceleration g, and P is the unknown pressure. The result is one unknown pressure per gridblock, one unknown wellbore pressure for each well on rate control, and a diagonal sparse matrix with a seven-point stencil for 3D grids. We solve the sparse matrix by means of an efficient multigrid method (Stüben 2001). Once pressures are known on the static Eulerian grid, grid-cell interface velocities are computed on the basis of Darcy's law. From the velocity field, streamlines are traced to construct the flow-based grid.

The mass-balance equation for phase saturations (S_j) on the underlying 3D grid that is coupled to the preceding pressure equation is given by

$$\phi \frac{\partial S_j}{\partial t} + \vec{u}_t \cdot \nabla f_j + \nabla \cdot \overrightarrow{G_j} = 0, \quad \text{.......... (2)}$$

where the phase fractional flow is given by,

$$f_j = \frac{\lambda_j}{\lambda_T}; \quad \lambda_j = \frac{k_{rj}}{\mu_j}, \quad \text{.......... (3)}$$

with phase relative permeabilities k_{rj} and phase viscosities μ_j. If we assume that the phase velocity caused by buoyancy (G_j) is aligned along the z-axis, then G_j is given by

$$\overrightarrow{G_j} = K_z \cdot g f_j \nabla D \sum_{i=1}^{n_p} \lambda_i (\rho_i - \rho_j), \quad \text{.......... (4)}$$

where K_z is the vertical permeability.

In streamline simulation, the 3D mass-balance equation can be recast into a set of 1D equations, each solved along a streamline using the time-of-flight variable τ. This is the key coordinate of interest along a streamline and is a reflection of the velocity field derived from the pressure solve. The gravity term, which is not aligned along streamlines, is accounted for in a separate step by means of operator splitting. See Thiele (2005) for details on the entire transport step.

The resulting 3D saturation equation in a streamline simulator is then given by

$$\sum_{\text{streamlines}}^{\text{all}} \frac{\partial S_j}{\partial t} + \frac{\partial f_j}{\partial \tau} = 0 \quad \text{.......... (5a)}$$

$$\sum_{\text{gravity lines}}^{\text{all}} \frac{\partial S_j}{\partial t} + \frac{1}{\phi} \frac{\partial G_j}{\partial z} = 0. \quad \text{.......... (5b)}$$

SPE_REE_118684_100030.indd 2 5/15/10 10:14:38 AM

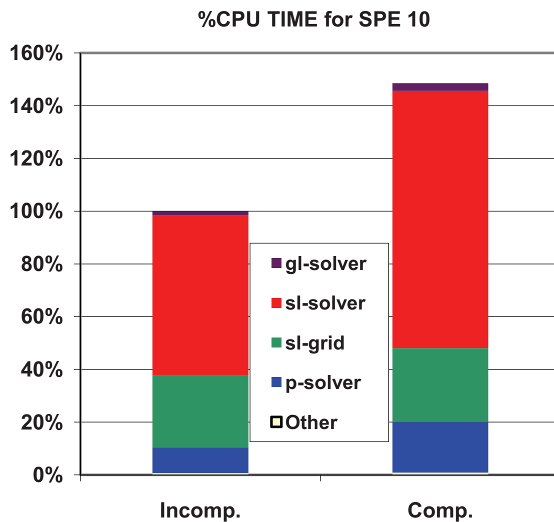


Fig. 2—Percentage of run time for major code sections for SPE10 (60×220×85), incompressible vs. compressible. It is assumed that the entire CPU time for the incompressible run represents 100% CPU time.

The solution to Eq. 5 is to first update saturation along all streamlines by means of a 1D convective solver, map each 1D result to the 3D grid, and then update saturation along each gravity line directly on the 3D grid by means of a 1D gravity solver (operator splitting). Within a global timestep, each streamline is solved for independently of other streamlines, and then each gravity line is solved for independently of other gravity lines. Thus, the original 3D mass-balance equation is reduced to a series of independent 1D equations, which is why SLS lends itself to parallelization. There is no reformulation of the underlying mass-balance equations to extend the method to parallel computation.

Computational Efficiency of a Streamline Simulator. Before discussing parallelization, we explain the computational efficiency of the streamline method in serial mode. For each timestep, Eq. 1 is solved first, Eq. 5a is solved second, and Eq. 5b is solved last. The total run time for a streamline simulator solving Eq. 1 and Eq 5a-5b can be approximated as,

$$T \approx \sum_{\text{timesteps}}^{\text{all}} \left(\begin{array}{l} t^{\text{psolve}} + t^{\text{sl-grid}} \\ + \sum\limits_{\text{streamlines}}^{\text{all}} t^{\text{sl-solver}} + \sum\limits_{\text{gravity lines}}^{\text{all}} t^{\text{gl-solver}} \end{array} \right). \quad (6)$$

At each timestep, t^{psolve} is the time required to set up and solve Eq. 1 as well as compute the velocity field. Once the velocity field is known, $t^{\text{sl-grid}}$ is the time required to construct the streamline-based grid, $t^{\text{sl-solver}}$ is the time to solve a 1D transport equation along a streamline (Eq. 5a), and $t^{\text{gl-solver}}$ is the time to solve a 1D transport equation along a gravity line (Eq. 5b). Note that $t^{\text{sl-grid}}$ occurs because of the dual-grid nature of streamline simulation, where for each timestep the result of the pressure solution is used to construct the flow-based grid. During this construction phase, checks such as proper streamline coverage and adequate well connections with streamlines are made.

To illustrate the proportion of CPU time required for each component of Eq. 6, we simulated the SPE 10 fine-grid model (60×220×85) containing approximately 1.1 million active grid cells to 2,000 days of water injection (Christie and Blunt 2001) for both compressible and incompressible pressure/volume/temperature (PVT). The incompressible model required 25 global timesteps, while the compressible model required 27 global timesteps. **Fig. 2** shows the contribution of each term in Eq. 6 for the two runs, scaled to the total runtime of the incompressible result. First, note that for both cases the solution to the multiple 1D equations along streamlines represents the largest percentage of CPU used, followed by sl-grid setup, followed by the pressure solve. Also note that the compressible run is approximately 50% slower than the equivalent incompressible run, because of increased iterations required for the pressure solve, a more complex PVT flash calculation, but mainly the more complex 1D transport equation to solve along each streamline. In general for streamline simulation, as the complexity of the 1D transport equation along a streamline increases, $t^{\text{sl-solver}}$ increases. Thus, extensions such as dual-porosity (DiDonato et al. 2003), compositional (Thiele et al. 1997), or polymer displacements (Thiele et al. 2010) primarily have an impact on $t^{\text{sl-solver}}$. Last, the gravity step represents very little of the overall work per timestep. This is particularly true for SPE 10 in which gravity effects are minor, with the 1D gravity solver having to do few timesteps per gravity line.

```
!$OMP PARALLEL PRIVATE (sl_path,sl_sats)
!$OMP DO SCHEDULE(dynamic,1)
DO i=1,nsl
     CALL tracestreamline(i,sl_path,sl_sats)
     CALL one_d_slsolver(sl_path,sl_sats)
!$OMP CRITICAL
     CALL mapsl2grid(sl_path,sl_sats)
     CALL mapsl2wells(sl_path,sl_sats)
!$OMP END CRITICAL
END DO
```

Fig. 3—Pseudocode with OpenMP directives.

Development of a Parallel Streamline Simulator

To parallelize the simulator, our goal was to include OpenMP directives within the existing code in the areas identified on the basis of Fig. 2 and similar timing tests. Key to this implementation was the easy identification of variables that could be shared between parallel threads and variables that were deemed critical (accessed by only one thread at a time). Because it was difficult to identify critical and shared variables by hand, we used the ADAPTOR Fortran compilation tool system developed by Brandes (2003). This tool analyzes the source code to identify all critical accesses to variables inside desired parallel regions. ADAPTOR also assisted in deciding how to proceed with each variable by listing all other read/write accesses throughout the parallel and serial parts of the code, particularly at the interprocedural level. With this information gathered, we could easily decide whether a variable needed special treatment or not.

We also used the profiling capabilities of the ADAPTOR system with access to the hardware performance counters to show that in a desired parallel region, memory performance was not a bottleneck for serial runs. This confirmed that parallelization to eliminate CPU bottlenecks was worthwhile.

Brief Description of OpenMP. With OpenMP directives, it is straightforward to included parallel processing of loops within existing Fortran code. One must first identify the loop to parallelize and use the SCHEDULE directive to control how the iterations of the loop are to be divided among the threads. Since all variables are assumed to be shared—that is, each thread has access to all variables in the program at all times—one must also define which variables are PRIVATE. Variables defined as PRIVATE are replicated for each thread and can be accessed only by the thread they are assigned to. Last, CRITICAL regions may be defined within the loop and are regions that can be accessed only by a single thread at a time.

On the basis of the preceding description, we show in **Fig. 3** using Fortran pseudocode, how to extend the streamline solver to parallel threads with OpenMP.

In the example in Fig. 3, the loop is parallelized over all streamlines (nsl). Each free thread is given a unique value of *i*. Within the call to tracestreamline( ) the value *i* points to a set of starting

SPE_REE_118684_100030.indd 3 5/15/10 10:14:42 AM

coordinates to trace the *i*th streamline from. The streamline is then traced and stored in sl_path along with the fluid saturations (sl_sats) from the underlying grid. These PRIVATE variables are then passed to the streamline solver where sl_sats are updated to the new time level by solving Eq. 5a. Once the transport step is completed, a thread will wait until it has sole access to the CRITICAL section that maps the new saturations back to the underlying grid and wells. To achieve ideal speedups for a large number of threads, it is important that the run time of the CRITICAL section of the loop be small compared to the work in the parallel section. Finally, because of the SCHEDULE(dynamic,1) construct, once a thread is free it will be assigned the next value of *i.* Thus, the work per thread is balanced dynamically, with each thread getting a new streamline once it has completed its current streamline.

Gravity-Line Solver. Although the gravity-line solver represented a small fraction of the total run time, this was the first section that was parallelized because it required no rewriting of the existing code. Furthermore, this region required parallelization of PVT routines, which would also be used for the more-complicated streamline-solver parallel region. Parallelization was at the level of a single gravity line per thread. Most importantly, the gravity-line solver had no critical variables to be concerned with because there is only one gravity line passing through any cell on the 3D grid.

Streamline Solver. As discussed, the solution of the transport equations along streamlines is intrinsically parallelizable—each streamline transport step is independent of all other streamlines—and represents the majority of CPU time during a simulation. Some code modifications were required to break loop order dependence because the code was originally written for serial implementation only. Once completed, we parallelized the loop that cycles over all the existing streamlines to be passed to the 1D solver at the level of a single streamline per thread. OpenMP critical directives were added to the routine that maps the results of each streamline solution back to the Eulerian grid and to the routine that maps the 1D injection/production information to the associated wells. Although these two routines cause each thread to wait before starting another streamline solution, the CPU requirements here are a tiny fraction compared with those of the 1D streamline solver (SL-solver) requirements, meaning that we do not expect these critical sections to cause significant performance bottlenecks.

Pressure Solver. The pressure solver was parallelized in three distinct places. First, we parallelized the flash of all gridblocks to update all flow properties on the basis of the last pressure solution. Second, the construction of the sparse matrix was parallelized. And third, the calculation of the velocity field on the basis of the new pressure solution was parallelized. All three of these components were parallelized and represent approximately 2% of the total CPU time of a run.

The SAMG multigrid solver, which represents the largest fraction of t^{psolve}, was provided as both a serial and an OpenMP version. However, because SAMG itself has both parallel and serial regions, it is difficult to separate timing of each region. To simplify our analysis and show how the SLS scaled, we used the serial version of SAMG for all our results.

Results

The hardware used for all simulations was an AMD™ Opteron™ 8-CPU 8384 quad-core 2.7-Ghz system with 128 GB of memory, running Linux-64 RH4™. Although this specific hardware is more representative of that found in high-end computer laboratories, quad-core and dual quad-core workstations are readily available to today's engineers.

We used the Intel 10.1 64-bit compiler. All loops were load balanced using the dynamic scheduling directive of the compiler, meaning that the compiler gives each thread a new task from the list of remaining tasks, once a thread's current task is finished. Depending on which loop was parallelized, a single task represented a single streamline, a gravity line, or a grid cell. This differes from the work of Gerritsen et al. (2009), in which they were able to predetermine how to optimally load balance each thread since they were solving linear transport problems along each streamline. Here, the vast majority of transport problems we solve are nonlinear, meaning we cannot predetermine how much work each streamline will require and thus use a compiler load-balancing option.

For all timing results, we had dedicated access to the hardware and performed each multithread run one at a time. To time performance, we used OMP_WALL_CLOCK timing directives around select parallel regions as well as the entire code.

We compared our speedup (*SU*) results with ideal speedup scaling, as defined by Amdahl's law (Amdahl 1967) as

$$SU = 1/(s + p/N), \qquad (7)$$

where the number of threads is N, the fraction of parallel code is p, and the serial portion is $s=1-p$. For comparison with Amdahl's law, we first computed the fraction p of run time code that was parallelized by timing performance of the parallel regions vs. total run time for a one-thread run for each simulation model. This value of p was then used in Amdahl's law to compare performance of multithread runs. As we show below, Amdahl's law implies that the maximum possible value of SU is limited by the value of s, regardless of the number of threads used. In other words, as the number of threads increases, the run time is eventually limited by the serial portion of code.

Repeatability of a Parallel-Run Result. In reservoir simulation, there are variables whose values are the result of running sums computed within a loop. In streamline simulation specifically, the gridblock saturations at the new time level are computed as running sums of all the new streamline saturations associated with the gridblock. Because of the fixed precision of floating-point calculations, the final value of the sum is dependent on the order in which the individual streamline saturations are added to the sum.

For a serial run, the order of the streamline tracing and, thus, the order of the sum is always the same, meaning simulation results are repeatable. However, in a multithread environment the summation of a gridblock's saturation resides inside a parallel loop, with the order of the running sum now a function of the number of threads used and processor load balances. Thus, the overall simulation results change slightly if a given data set is simulated using two threads vs. four threads, for example. Furthermore, because of variations in processor load balances between separate runs, repeating a four-thread flow simulation, for example, will not give exactly the same results either.

For parallel simulation runs, the dependency of results on the number of threads means that the exact same problem is not being computed as a model is tested on various numbers of threads. Thus, for all results, we confirmed that the overall field production profiles for each multithread run were within engineering accuracy to the one-thread run of each case studied. Nonuniqueness also means that the value of s (and p) computed for Amdahl's law for a one-thread run is an estimate that is subject to small variability.

SPE 10

The fine-scale 60×220×85 SPE 10 model was simulated on 1 to 16 threads for both incompressible and compressible PVT systems. As expected from Fig. 2, which showed the sl-solver representing a larger portion of run time code for the compressible run vs. the incompressible run, we computed p=0.63 for the incompressible run and p=0.70 for the compressible run. **Fig. 4** shows that run-time speedups compare favorably with ideal scaling.

The overall speedup results can be broken down further into speedup factors for each parallelized region. Recall that the streamline solver (red line) represents the greatest portion of parallelized code, meaning that how this region scales dictates the overall speedup shown in Fig. 4. As shown in **Fig. 5,** results for the streamline solver compressible run, which represents approximately 66% of run time for a one-thread run, had a speedup factor of 11.4x

SPE_REE_118684_100030.indd 4 5/15/10 10:14:43 AM

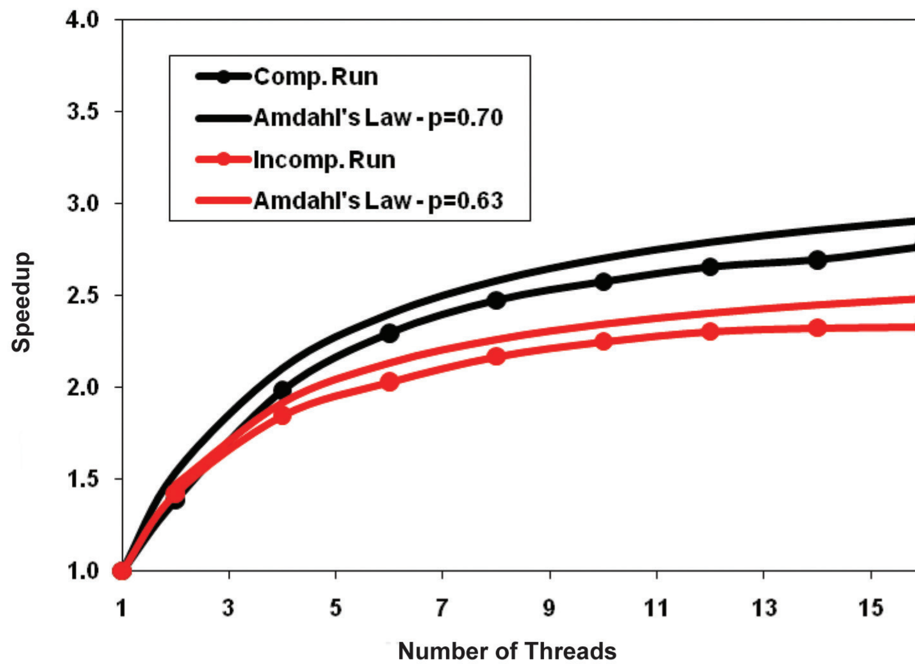


Fig. 4—Speedup factor vs. number of threads for SPE 10 incompressible model (black) and compressible model (red). Amdahl's law, which represents ideal scaling, is shown as the dashed lines.

on 16 threads. This would explain why overall speedups do not follow ideal scaling. Also note in Fig. 5 the poor scaling of the grid flash, construction of the pressure matrix, and the velocity-field calculation. Fortunately these represent only a small fraction of the run-time code. Recall that these routines are parallelized at the gridblock level, meaning that the work per thread is low and data transfer is high, compared with the work per thread of the streamline solver. In the Discussion section, we will comment further on speedup scaling performance.

Field Case 1—Forties, North Sea. The Forties simulation model shown here was provide by Apache Corporation. The flow model contains almost 1.5 million active cells, 235 wells, and more than 40 years of history over 76 timesteps. A one-thread run required approximately 6 hours to simulate, of which the parallel code fraction was p=0.52 of the total run time.

Overall speedup factors for 1 to 16 threads, in two threads increments, are shown in **Fig. 6a,** where again the drop in speedup for the 12- and 16-thread runs were caused by increased run time in the serial sl-grid setup routines. The speedup factor of each parallel section is shown in Fig. 6b, where we observed scaling of the sl-solver (12.2x at 16 threads) similar to that for SPE 10. In terms of actual run time, this model required approximately 3.5 hours or less of CPU time when eight or more threads were used,

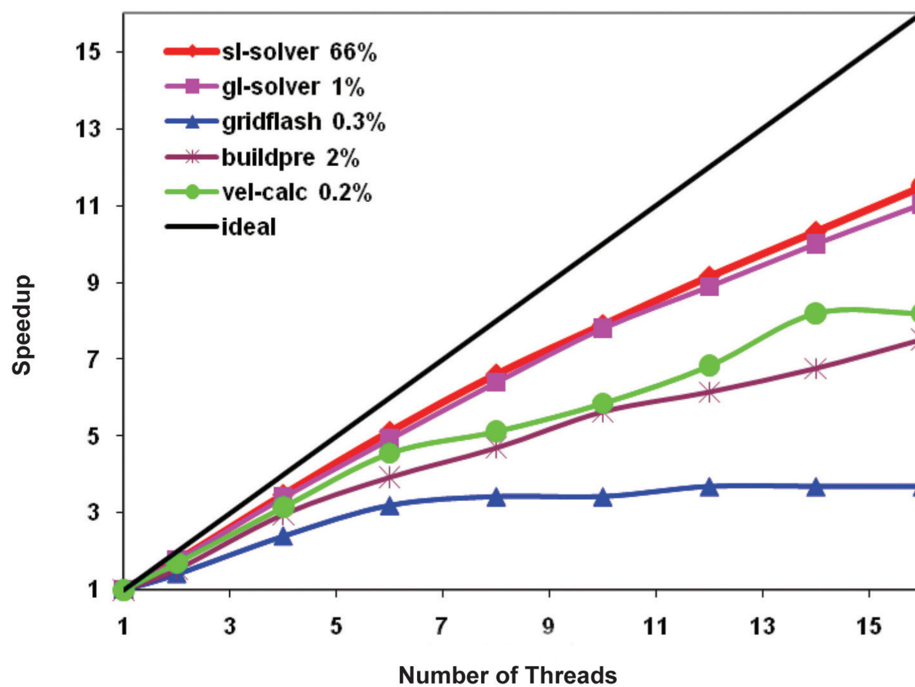
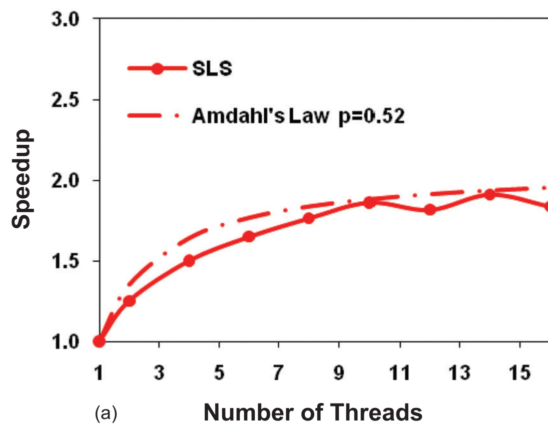


Fig. 5—Speedup factors for each parallel region in simulator for SPE 10 compressible model. The percentage numbers in the legend represent the fraction of run-time code that the parallelized section required for the one-thread run.

SPE_REE_118684_100030.indd 5 5/15/10 10:14:43 AM



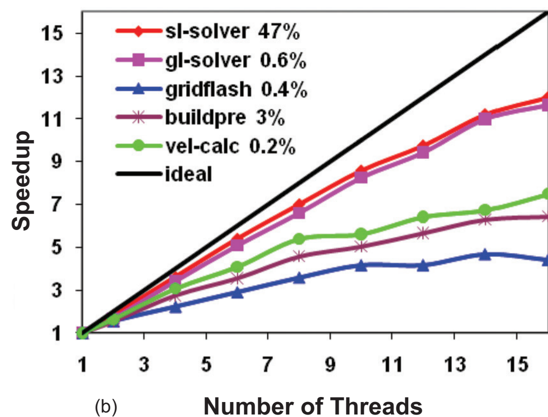
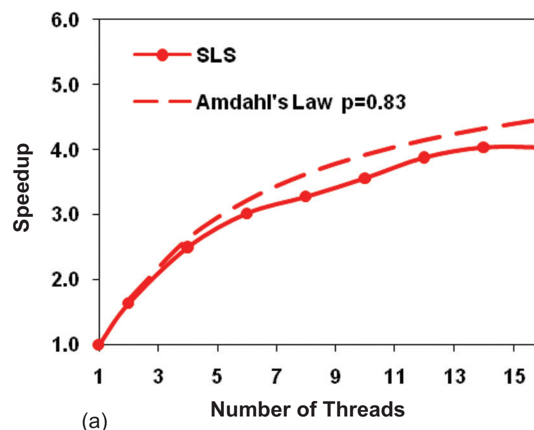


Fig. 6—Overall speedup factor for Field Case 1 vs. number of threads (a). Speedup factor vs. number of threads for each parallel region (b), with the overall run-time percentage that each region required for the one-thread run listed in the legend.



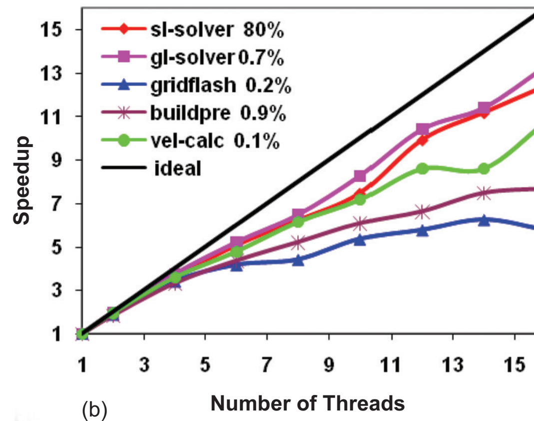


Fig. 7—Overall speedup factor for Field Case 2 vs. number of threads (a). Speedup factor vs. number of threads for each parallel region (b), with the overall run-time percentage each region required for the one-thread run listed in the legend.

but given the low value of *p,* there is little improvement beyond eight threads.

Field Case 2—Judy Creek, Alberta, Canada. The Judy Creek simulation model shown here is the same as the final history-matched model discussed by Batycky et al. (2007). The flow model contains approximately 623,000 active cells, 300 wells, and 46 years of history lumped into 1-year timestep intervals. Water injection started soon after initial production. For the latest 20-year period, there has been both water injection and miscible gas injection. The PVT assumed for this field was a three-phase oil/water/solvent incompressible system with solvent/oil miscibility modeled using a Todd-Longstaff-type formlation. A one-thread run required approximately 5.2 hours, of which the parallel code fraction was p=0.83 of the total run time. This large percentage of parallel code was expected because of the more complex miscible model being solved along each streamline, in comparison to the Forties or SPE 10 compressible models.

Overall speedup factors for 1 to 16 threads, in increments of two threads, are shown in **Fig. 7a,** while the speedup factor of each parallel section is shown in Fig. 7b. The overall speedup results agree well with ideal scaling, with speedups of approximately 3.3x or better for eight or more threads, representing an actual run time of approximately 1.6 hours or less. The streamline solver parallel region, the largest portion of runtime code, showed a speedup of 12.5x for 16 threads.

Field Case 3—Aquifer-Drive Black-Oil Model, South America. In the preceding two field examples, the SLS method was taking between 6-month and 1-year timesteps. Those models are ideal for streamline simulation because the results are weak functions of pressure and were considered "converged" solutions even at these large timesteps. However, shorter timesteps may be required because of well constraints or nonlinearities such as gravity or compressibility. This means that more execution time is spent in nonparallelized portions of the code such as the pressure solver and streamline-grid setup routines, leading to overall lower speedups.

To illustrate the impact of timestep size on performance, we tested a 970,000-grid-cell black-oil model with more than 25 producers, bottom aquifer support, and 24 years of production history. A single-thread 1-year timestep size model required approximately 2.1 hours and giving p=0.57. We then forced the simulator to take 1-month timesteps, requiring 12.6 hours with a resulting p=0.50. As expected, p is now lower since a larger portion of time is spent in the SAMG pressure solver and the streamline-grid setup routines. In other words, this model represents a worst-case scenario in terms of fraction of run time code that is not parallelized.

Speedup factors were computed up to 16 threads and are shown in **Fig. 8a,** with the maximum speedup limited to approximately 1.8x, given the low value of p. The speedup factor of each parallel section is shown in Fig. 8b. Note that the maximum speedup scaling of the streamline solver is the lowest of all examples presented, 9.6x on 16 threads.

Discussion

For the preceding field cases, the fraction of run-time code parallelized ranged from 0.50 to 0.83, with the 1D streamline solver representing the largest portion of parallelized run-time code in all cases. As expected the streamline solver represented a larger portion of parallel code the more complex the 1D-transport equations became. For all cases, the scaling of the streamline solver drifted from ideal scaling as the number of threads increased, particularly above approximately four threads. To determine why the scaling showed this drop, we removed the critical regions in the solver

SPE_REE_118684_100030.indd 6 5/15/10 10:14:44 AM

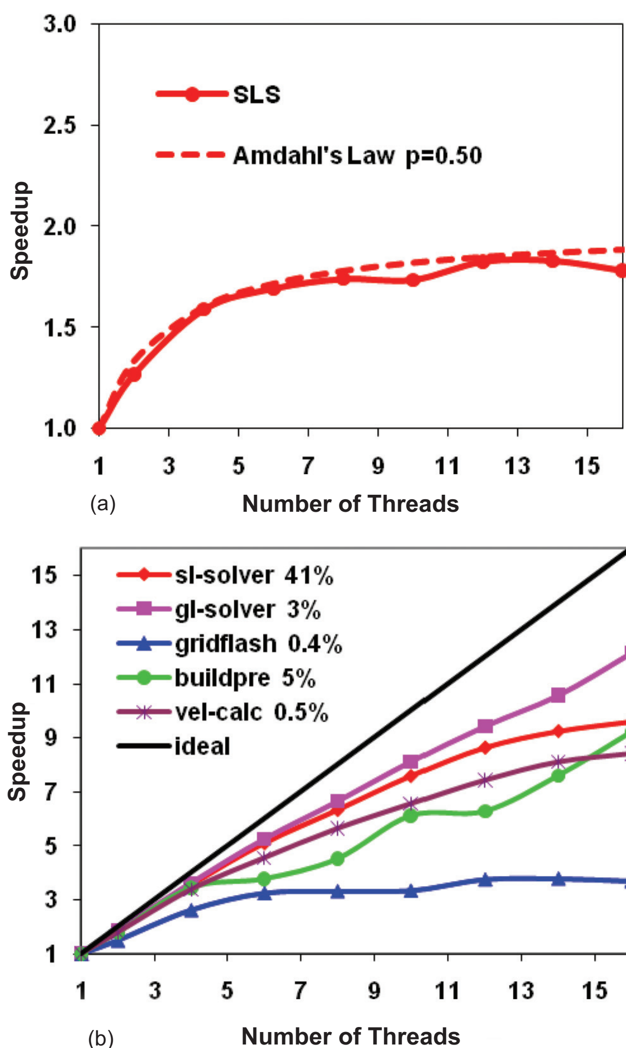


Fig. 8—Overall speedup factor for Field Case 3 vs. number of threads (a). Speedup factor vs. number of threads for each parallel region (b), with the overall run-time percentage each region required for the one-thread run listed in the legend.

region, yet noted minimal improvement in scaling performance. We concluded from this experiment that threads waiting for access to the critical regions were not the cause in the scaling drift. Such behavior was expected since the routines containing the critical regions are simple and efficient relative to the overall 1D-solver requirements for each streamline. As a second test, we checked scaling performance of the Forties model on similar hardware but with slower processors (1.8 Ghz). We noted that all parallel regions scaled better, with the streamline solver showing close to ideal scaling (approximately 16x on 16 threads). These improved scaling results suggest that the observed drift in speedup is most likely related to memory-bandwidth limitations. In other words, when using a slower processor, each thread requires more time to complete its task, meaning access to shared memory occurs at a reduced frequency. In turn, the memory bus is better able to handle the data flow, threads are not waiting as much for new work. This would also explain the poor scaling of the sl-solver for the final example, in which the global timesteps are small, the work per streamline is reduced, and the communication load is high. This observation also holds when looking at the scaling of the flash-grid and velocity-field calculation routines. These routines require little work per thread, yet require information from memory at a much higher frequency than the sl-solver region does, with the memory bus speed being the limiting factor.

Recall from Fig. 2 that the pressure solve requires approximately 20% run time, and that the SAMG solver, which is part of t^{psolve}, represents the largest portion of the pressure solve, particularly for compressible models. For all results shown here, we used the serial version of SAMG. Although not all of SAMG is parallelized, the OpenMP version can still give a significant time savings. Tests at eight threads showed a further savings in CPU time of 1 hour for Forties (increase in p to 0.74) and 1 hour for Field Case 3 (increase in p to 0.62), when using the OpenMP version of SAMG.

Conclusions and Future Efforts

We have successfully parallelized an existing commercial SLSs and tested it on practical reservoir models. The level of modifications required to parallelize the code was minimal, owing to the intrinsically parallel nature of the streamline transport step. For eight threads, a number that any of today's reasonably priced workstations would have, we found wall clock speedup factors in the range of 1.8 to 3.3*x*. This means that large streamline simulation models, as we tested here, are within reach for reservoir engineers to run in less than 4 hours.

The overall scaling performance was principally determined by how well the streamline solver parallel region scaled. We observed speedup factors for this region ranging from approximately 9.6 to 12.5x for our various 16-thread runs. We attribute nonideal speedup to a memory-bandwidth bottleneck on our test machine. A slower CPU machine with similar memory-bus speed showed near-optimal scaling of the streamline solver.

Amdahl's law shows that the greatest incremental speedups are at small values of N-threads. As N increases, overall speedup factors are eventually limited by the fixed computing cost of the serial regions. Furthermore, we observed that beyond approximately eight threads, memory bottlenecks limited scaling performance of the parallel regions. Both observations imply that overall speedups are limited as N increases, and that it is worthwhile focusing on performance for reasonable values of N ($2<N<8$).

Nomenclature

D = depth, L
f_j = fractional flow of phase j, dimensionless
g = gravitational acceleration, L/t^2
G_j = phase j velocity because of density differences, L/t
k_{rj} = relative permeability of phase j, dimensionless
K = permeability, L^2
n_p = number of phases
N = number of threads
p = fraction of parallel run time code, dimensionless
P = pressure, m/Lt^2
s = fraction of serial run time code, dimensionless
S_j = saturation of phase j, dimensionless
SU = speedup factor, dimensionless
t = time
u_t = total velocity of all phases, L/t
z = vertical grid coordinate, L
λ_j = mobility of phase j, Lt/m
ϕ = porosity, dimensionless
ρ_j = density of phase j, m/L^3
μ_j = viscosity of phase j, m/Lt
τ = time of flight, t

Acknowledgments

We would like to thank Thomas Brandes of Fraunhofer's Scientific Computing Institute for the use and support of his Adaptor compilation tool, Richard Jones of Apache Corporation for allowing us to use Forties, and Andrew Seto of Pengrowth Corporation for allowing us to use Judy Creek.

References

3DSL User Manual v4.0. 2009. Calgary: Streamsim Technologies.
Ács, G., Doleschall, S., and Farkas, E. 1985. General Purpose Compositional Model. *SPE J.* **25** (4): 543–553. SPE-10515-PA. doi: 10.2118/10515-PA.

SPE_REE_118684_100030.indd 7 5/15/10 10:14:45 AM

Amdahl, G. 1967. Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities. In *AFIPS Conference Proceedings: 1967 Spring Joint Computing Conference*, Vol. 30, 483–485.

Batycky, R.P, Seto, A.C., and Fenwick, D.H. 2007. Assisted History Matching of a 1.4-Million-Cell Simulation Model for Judy Creek 'A' Pool Waterflood/HCMF Using a Streamline-Based Workflow. Paper SPE 108701 presented at the SPE Annual Technical Conference and Exhibition, Anaheim, California, USA, 11–14 November. doi: 10.2118/108701-MS.

Batycky, R.P., Blunt, M.J., and Thiele, M.R. 1997. A 3D Field-Scale Streamline-Based Reservoir Simulator. *SPE Res Eng* **12** (4): 246–254. SPE-36726-PA. doi: 10.2118/36726-PA.

Brandes, T. 2003. Adaptor: Parallel Fortran Compilation System. Fraunhofer SCAI, http://www.scai.fraunhofer.de/EP-CACHE/adaptor/www/adaptor_home.html.

Christie, M.A. and Blunt, M.J. 2001. Tenth SPE Comparative Solution Project: A Comparison of Upscaling Techniques. *SPE Res Eval & Eng* **4** (4): 308–317. SPE-72469-PA. doi: 10.2118/72469-PA.

Collins, D.A, Grabenstetter, J.E., and Sammon, P.H. 2003. A Shared-Memory Parallel Black-Oil Simulator with a Parallel ILU Linear Solver. Paper SPE 79713 presented at the SPE Reservoir Simulation Symposium, Houston, 3–5 February. doi: 10.2118/79713-MS.

Datta-Gupta, A. and King, M.J. 2007. *Streamline Simulation: Theory and Practice*. Textbook Series, SPE, Richardson, Texas, USA **11**.

DeBaun, D., Byer, T., Childs, P., Chen, J., Saaf, F., Wells, M., Liu, J. et al. 2005. An Extensible Architecture for Next Generation Scalable Parallel Reservoir Simulation. Paper SPE 93274 presented at the SPE Reservoir Simulation Symposium, Houston, 31 January–2 February. doi: 10.2118/93274-MS.

Di Donato, G., Huang, W., and Blunt, M. 2003. Streamline-Based Dual Porosity Simulation of Fractured Reservoirs. Paper SPE 84036 presented at the SPE Annual Technical Conference and Exhibition, Denver, 5–8 October. doi: 10.2118/84036-MS.

Gerritsen, M.G., Löf, H., and Thiele, M.R. 2009. Parallel implementations of streamline simulators. *Computational Geosciences* **13** (1): 135–149. doi: 10.1007/s10596-008-9113-y.

Shiralkar, G.S., Fleming, G.C., Watts, J.W., Wong, T.W., Coats, B.K., Mossbarger, R., Robbana, E., and Batten, A.H. 2005. Development and Field Application of a High Performance, Unstructured Simulator with Parallel Capability. Paper SPE 93080 presented at the SPE Reservoir Simulation Symposium, The Woodlands, Texas, USA, 31 January–2 February. doi: 10.2118/93080-MS.

Stüben, K. 2001. An Introduction to Algebraic Multigrid. In Multigrid, ed. U. Trottenberg, C.W. Ooosterlee, and A. Schüller, Appendix, 413–532. London: Academic Press.

Thiele, M.R. 2005. Streamline Simulation. Keynote address presented at the 9th International Forum on Reservoir Simulation, Stresa, Italy, 20–24 June.

Thiele, M.R., Batycky, R.P., and Blunt, M.J. 1997. A Streamline-Based 3D Field-Scale Compositional Reservoir Simulator. Paper SPE 38889 presented at the SPE Annual Technical Conference and Exhibition, San Antonio, Texas, USA, 5–8 October. doi: 10.2118/38889-MS.

Thiele, M.R., Batycky, R.P., Pollitzer, S., and Clemens, T. 2010. Polymer-Flood Modeling Using Streamlines. *SPE Res Eval & Eng* **13** (2): 313-322. SPE-115545-PA. doi: 10.2118/115545-PA.

Watts, J.W. 1986. A Compositional Formulation of the Pressure and Saturation Equations. *SPE Res Eng* **1** (3): 243–252; *Trans.,* AIME, **281**. SPE-12244-PA. doi: 10.2118/12244-PA.

Rod Batycky is cofounder of Streamsim Technologies. He holds MSc and PhD degrees in petroleum engineering from Stanford U.—during his studies, he was awarded the SPE Cedrick K. Ferguson Medal. Previously, Batycky worked as a reservoir engineer with Shell Canada, and initially obtained a BSc degree in chemical engineering from the U. of Calgary. He is a technical editor for *SPE Res Eval & Eng* and *J. Cdn. Pet. Tech.* **Malte Förster** is a PhD student at the numerical software department of Fraunhofer's Scientific Computing Institute (FhG-SCAI), where he is a member of the numerical solver group. He focuses on the development of efficient parallel software, both MPI- and OpenMP-based. Förster holds an MSc degree in mathematics from the U. of Cologne. **Marco Thiele** is cofounder of Streamsim Technologies and a consulting professor at Stanford U. He holds BS and MSc degrees from the U. of Texas at Austin and a PhD from Stanford U., all in petroleum engineering. Thiele is a recipient of the 1996 SPE Cedrick K. Ferguson Medal, serves on the SPE Books Committee and is an Associate Editor for *SPE Res Eval & Eng*. **Klaus Stüben** is deputy head of the numerical software department of FhG-SCAI, and is responsible for the scientific development of modern fast numerical solvers. He is one of the inventors of the algebraic multigrid (AMG) technique and is heading the development of the advanced software library, SAMG, which has become a well-established tool in many industrial simulation systems. Stüben is currently focusing on the development of efficient numerical approaches for solving various types of partial differential equations, including such diverse areas as fluid flow, structural mechanics, oil reservoir and ground water simulation, casting and molding, process and device simulation in solid state physics, electro-chemistry, and circuit simulation. He holds MSc and PhD degrees in mathematics from the U. of Cologne.

SPE_REE_118684_100030.indd 8 5/15/10 10:14:46 AM