A15

Adaptive Control for Solver Performance Optimization in Reservoir Simulation

I.D. Mishev* (ExxonMobil Upstream Research Company), N. Fedorova (NeurOK TechSoft), S. Terekhov (NeurOK TechSoft), B.L. Beckner (ExxonMobil Upstream Research Company), A.K. Usadi (ExxonMobil Corporate Strategic Research), M.B. Ray (ExxonMobil Corporate Strategic Research) & O. Diyankov (NeurOK TechSoft)

SUMMARY

Run-time performance of a reservoir simulator is significantly impacted by the selection of the linear solver preconditioner, iterative method and their adjustable parameters. The choice of the best solver algorithm and its optimal parameters is a difficult problem that even the experienced simulator users cannot adequately solve by themselves. The typical user action is to use the default solver settings or a small perturbation of them that are frequently far from optimal and consequently the performance may deteriorate.
There has been extensive research to develop automatic performance tuning and self-adaptive solver selection systems. For example Self-Adapting Large-scale Solver Architecture (SALSA) developed at the University of Tennessee requires running of large number of problems to initialize the system before using it. In contrast we propose an adaptive control on-line system to optimize the simulator performance by dynamically adjusting the solver parameters during the simulation. We start with a large set of parameters and quickly choose the best combinations that are continuously adapted during the simulation using the solver runtime performance measurements (e.g. solver CPU time) to guide the search.
This software system, called the Intelligent Performance Assistant (IPA), has been successfully integrated into ExxonMobil s proprietary reservoir simulator and deployed with it worldwide. The system can handle a large number of combinations of solver parameters, currently in the order of 108, and consistently improves run time performance of real simulation models, frequently by 30% or more, compared to the performance with the default solver settings. Moreover, IPA includes a persistent memory of solver performance statistics. The runtime statistics from these individual runs can be gathered, processed using data mining techniques and integrated in the IPA system, thus allowing its continuous improvement.

Introduction

Current state-of-the-art reservoir simulators can model different types of flows in the porous media depending on the types of fluids (black oil, compositional, thermal, etc.), displacement mechanisms, and formulations (implicit pressure and explicit saturation, sequential implicit, or fully implicit). They are very complicated modeling tools and their efficient use requires engineers and geologists with deep knowledge in a broad area of geology, physics and engineering. Most competent users have only basic knowledge about the linear solvers and in general it is not reasonable to expect more. They have neither the time, nor the necessary background to acquire deeper understanding and knowledge about the iterative methods, preconditioners, and their parameters. Not choosing a good solver can significantly impact the overall performance of the simulator. Even the most efficient solvers can take between 10% and 80% of the total simulation time.

The typical action of most users is to run the simulations with the default parameters for the solver. This is a good choice for some models as shown for Model 2 in our numerical experiments, but for other models such an approach could lead to significant slow down of the simulation, for example more than 60% for Model 1, or even worse as shown for Model 3, where even finishing the simulation in a timely manner becomes problematic. The reason for the disappointing performance of the solver is that either the convergence is slower with the default parameters (Model 1), or there is frequent divergence that leads to many time step cuts (Model 3). It is not possible to choose default parameters that work well for all cases.

Automatic performance tuning and implementation of a self-adaptive solver selection systems are one possible solution to the problem described above. There has been significant research on developing such systems for long-running time-stepping simulations. One approach is to investigate the properties of a few sample matrices and relate the solver performance to the matrix properties. The Self-Adapting Large-scale Solver Architecture (SALSA) developed at the University of Tennessee is a representative of this approach [1]. A disadvantage of the method is that the cheap matrix properties like min / max entry are not very informative, the informative properties like condition number, eigenvalues and their distribution, etc., are very expensive to compute and therefore this can be done only off-line. Generating an accurate description of the relationship between matrix properties and solver performance requires running a large number of simulations on different computer platforms/architectures. The resulting predictive system is static and must be updated/retrained with more recent examples/matrices in order to stay relevant and to incorporate new data (for new matrices, cases, models, for new computer architectures, etc.), as well as algorithmic improvements and implementation enhancements to the linear solver.

The adaptive solver described in [2] also finds problems with similar features in the persistent database of previously solved matrices to generate near-optimal settings for the next time step/solve. The solver adjusts the ILUT parameter lfil (level of fill) using the pre-programmed heuristic formula/rules that were derived through off-line statistical analysis of solver runtime performance descriptors gathered online during previous simulation runs.

α-SAMG described in [3] does not use a persistent database of previously solved cases. Instead it implements an automatic adaptive strategy based solely on online solver performance statistics gathered for the current simulation run. In α-SAMG, matrices solved during a simulation are divided into several classes ("D-classes") depending on their size/dimension. During a simulation run solver convergence, timings and memory requirement history is continuously stored. Within a given D-class, comparisons between different solvers (currently between one-level and multi-level variant of the basic solver) and different solver settings (currently between different combinations of lfil and drop tolerance parameters of ILUT preconditioner) are made from time to time (e.g. for every 50-th matrix from this D-class). For each D-class, the solver is switched and its parameters are changed according to results of these online tests using heuristics/rules that utilize the explicit knowledge of SAMG algorithm.

Our software system, called the Intelligent Performance Assistant (IPA), developed in 2004, implements a self-guided, self-acting adaptive control strategy based on general optimization techniques, genetic search and Response Surface Methodology (RSM) [4].

Although the IPA system includes a persistent database of solver performance runtime statistics that is gathered during previous simulation runs and can be used to improve IPA performance, this persistent database is not a requirement for the IPA adaptive control agent to work properly. Furthermore, the IPA algorithm does not rely on the results of preliminary experimenting with matrices, and does not require training/updating a classifier like that in SALSA, or deriving and programming heuristics/rules like those used in the adaptive solver [2] or in α-SAMG. Indeed, the IPA system does not require explicit knowledge of the solver algorithm, and therefore can be applied to optimize any linear solver.

System overview

Conceptually, IPA is an adaptive control agent that is used to control linear solves performed for different matrices generated in the course of a reservoir simulation.

In reinforcement learning and adaptive control theory, the learner and decision-maker (*agent*) and its *environment* interact over a sequence of discrete time steps, the agent selecting *actions* and the environment *responding* to those actions and presenting new situations to the agent. The environment also gives rise to *rewards*, a special performance response whose values the agent tries to maximize over time. The agent's goal is to maximize the total amount of reward it receives over the long run [5].

The goal of the IPA agent is to minimize total computational costs (e.g. total time spent in linear solver) accumulated over all the time steps of reservoir simulation. Since matrices solved during reservoir simulation change from one time step to the next, IPA agent’s environment is non-stationary, and IPA agent’s optimal policy is a moving target.

In recent years, researchers have become increasingly interested in the use of Genetic Algorithms (GAs) as a means to control time-varying noisy systems because of their ability to effectively search among a large number of parameter combinations performing either single-objective or multi-objective optimization and because of their in-built mechanism for adaptation [6].

A genetic algorithm implies a sequence of steps that can be briefly described as follows.

1. Encode the problem into chromosomes of genes.

2. Generate the initial population of chromosomes (i.e. the group of possible solutions) either at random or heuristically.

3. Evaluate fitness value for each chromosome (it should depend on the distance to the optimum).

4. Select the chromosomes that will "mate" according to their rating in the population global fitness.

5. Perform crossover, mutations, etc. for the selected chromosomes.

6. Repeat from point 3.

With genetic algorithm we always obtain a whole population of optimal and near-optimal solutions (and not just one optimal solution). Changes in the environment exert constant selective pressure in favor of the solutions that are most fit for the current environment. As long as the environment changes slowly with respect to the time required for evaluating a population of solutions, the population is able to track a changing fitness landscape.

In the current IPA/GA implementation, the solver parameters are divided into several groups representing elements of solve procedure. For each such group, the choice of linear solver settings consists of the choice of solve method, and the choice of parameters of this method.

In IPA, all solver parameters, both discrete (e.g. type of preconditioner) and continuous (e.g. drop tolerances for FILU preconditioner [7]), are represented with finite non-ordered sets of discrete categorical values that are encoded with integer-valued genes. Each combination of solver settings is represented with a vector ('chromosome') of such genes. We discretize each continuous parameter, limiting it to a few discrete values that we believe to be most practically useful. For example, for the 2nd drop tolerance parameter of FILU preconditioner [7] we only consider the choices 0.1, 1, 10. Similar 'binning' technique for continuous parameters is used in SALSA [1].

When asked to generate solver settings for the next matrix, IPA converts its internal chromosome-based representation to a command line string that can be then parsed and translated into solver settings either by the solver itself, or by some intermediate software layer.

The IPA agent starts from some initial list of solver parameter combinations and the vector of *a priori* ratings (selection probabilities) assigned to all these combinations/chromosomes. In the simplest case the IPA agent can be given the list of all possible parameter combinations available in the solver, with equal relative ratings/probabilities. Based on these *a priori* ratings IPA generates some initial population of chromosomes.

The IPA implements a feedback adaptive control mechanism. The IPA agent generates a combination of solver parameters at each time step (one combination per time step). These settings determine uniquely the linear solver used at this time step. Solver performance descriptors (total solve CPU time, number of solver iterations, memory usage, etc.) are passed back to IPA agent. These statistics are further used to update the relative ratings of parameter combinations/chromosomes. Combinations with good performance (with small CPU times, etc.) get better ratings. Non- or slow- convergent combinations get very low or zero ratings.

Solver settings for the next time steps are chosen with respect to their relative ratings/probabilities. Combinations with high ratings are chosen to "mate" and produce offsprings more often than those with small ratings. This mechanism enables the IPA to explore the most promising combinations, and finally converge to exploiting a set of several near-optimal combinations with the largest ratings/selection probabilities.

If for some combination of parameters the solver does not converge, simulator performs a second run for the same matrix, with solver settings that are currently considered optimal.

Since matrix properties can change drastically from one time step to the next, to make comparisons of the linear solver performance on different matrices more accurate it is desirable (although not mandatory) to perform sorting of solver performance measurements taking into account actual matrix complexity that should be expressed with some numerical measure (e.g. with the number of variables as in α-SAMG). There are no pre-programmed rules in IPA. Instead, generic RSM models are built using experimental data gathered online during the simulation.

Runtime solver performance statistics and ratings/probabilities of all parameter combinations are stored in a database of solved cases. These data can be used to estimate *a posteriori* probabilities of all combinations for a new case ('given a case that is similar to previously solved case(s), what is the probability of all presets?'). These probabilities can be further used to generate good starting lists of parameter combinations for the new simulation runs of similar cases. These lists may be used instead of the full list of all possible combinations to speed up the online search for the optimal combinations.

Numerical results

First we apply IPA to optimize 3 parameters of SparSol solver [7]. IPA algorithm selects among 12 scalings, 3 reorderings, and 20 combinations of 2 parameters of FILU preconditioner, an ILU with threshold type of a preconditioner. We present IPA results for 3 models. IPA-assisted simulation runs are compared with simulation runs with default solver settings.

Model 1 is a black oil, three phase flow, sequential formulation model with 93K cells and 31 wells that ran for 5000 days. The linear system is for the pressure equation, one unknown per cell. The IPA performed very well. The overall simulation runtime using IPA was 39% faster in the total CPU time and the solver with parameters provided by IPA was 2.06 times faster than the solver with the default parameters (see Fig. 1). The discrepancy is due to this model's non-solver cost and non IPA overhead.

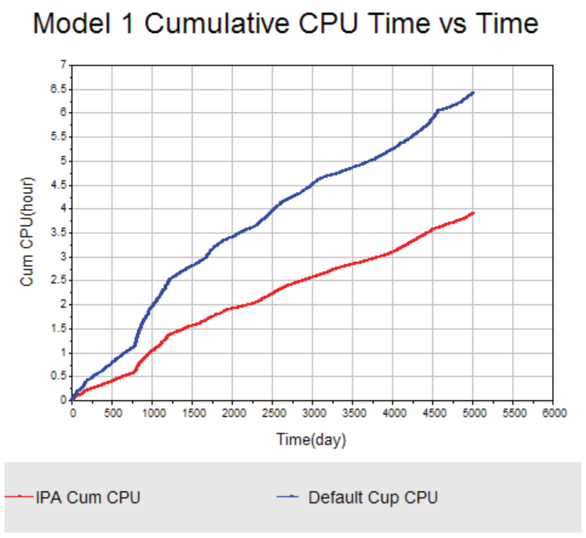

Figure 1. Results for Model 1.

Model 2 is also a black oil, three phase flow model, but the formulation is fully implicit. It has 103K cells with 79 wells. Simulation ran for 5000 days. There are four unknowns per cell, i.e., the size of the linear system is 412K. The IPA solver was about 20% faster than the default solver (Fig. 2). The default parameters were relatively good and the solver performance was less sensitive to the parameters than the solver performance for Model 1. In the beginning of the simulation the performance of the solver with the default parameters was even slightly better. This is due to the exploration for best parameters that adds some extra cost in the early stage of the simulation. The default parameters were not so close to the optimal ones in the later stage of the simulation and the IPA provided a performance improvement.

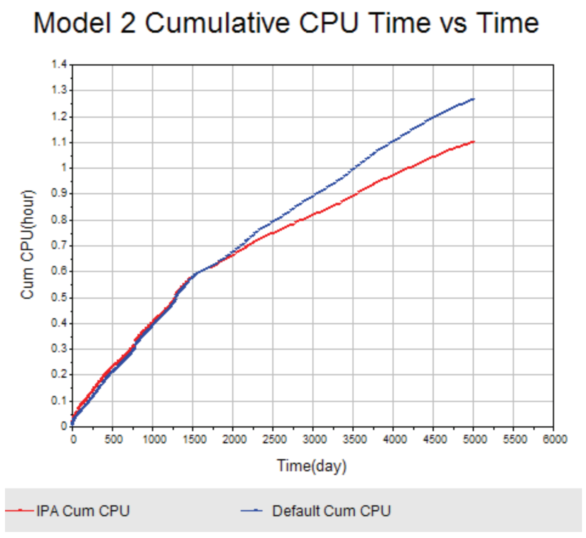

Figure 2. Results for Model 2.

Model 3 is a fully implicit, thermal model with 14K cells (five unknowns per cell). The simulation ran for 2909 days. Such models are particularly difficult for the solver because the energy equation is scaled very differently from the other equations. The default parameters that worked well for Model 2 were not good at all for this model. There were multiple divergences (no convergence for 250 iterations, see Fig. 4) of the solver that lead to time step cuts. The solver converged well with the parameters provided by IPA, the number of iterations was smaller than 50 with only one exception (see Fig. 5). As a result the total CPU simulation time with the default parameters was 4.5 times slower than the one of the IPA run. We usually recommend using a scaling before the preconditioner is built for thermal cases. That improved the performance considerably. The CPU simulation time with scaling is only 4% slower than the CPU simulation time for the IPA run. This models demonstrates how new or inexperienced user can achieve results comparable to the results obtain by experienced user by using IPA.

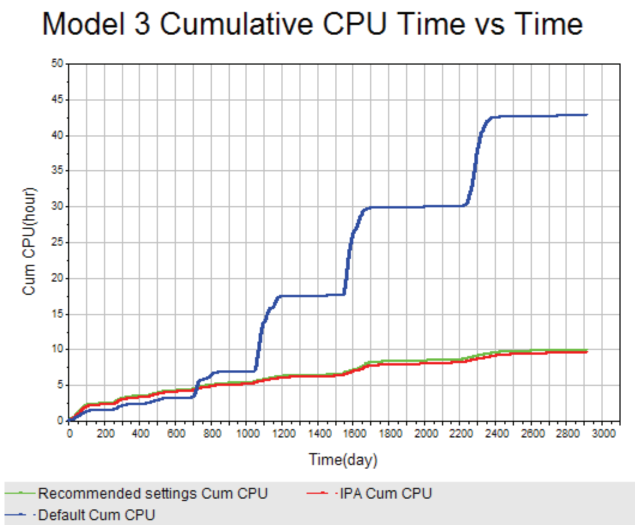

Figure 3. Results for Model 3.

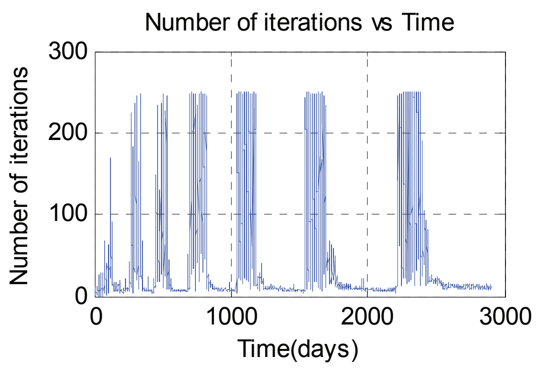

Figure 4. Number of iterations vs. Time for Model 3. Default parameters.

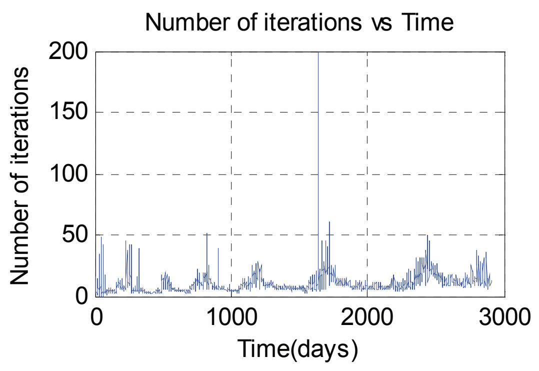

Figure 5. Number of iterations vs. Time for Model 3. IPA parameters.

We also used IPA for online tuning of the SAMG [8] parameters. We consider SAMG, an advance Algebraic MultiGrid solver, as a “black box”, relying only on the user manual and several recommendations by Klaus Stuben [9], one of the authors of the original algorithm and major developer of the software. We tried two of his recommendations, denoted here “good” and “bad”. The “good” recommendation (BiCGStab, GS smoother, standard coarsening, wells excluded) works very well on difficult problems. The “bad” combination (GMRES, MILU smoother, aggressive coarsening, wells are not excluded) performs very well on simpler problems.

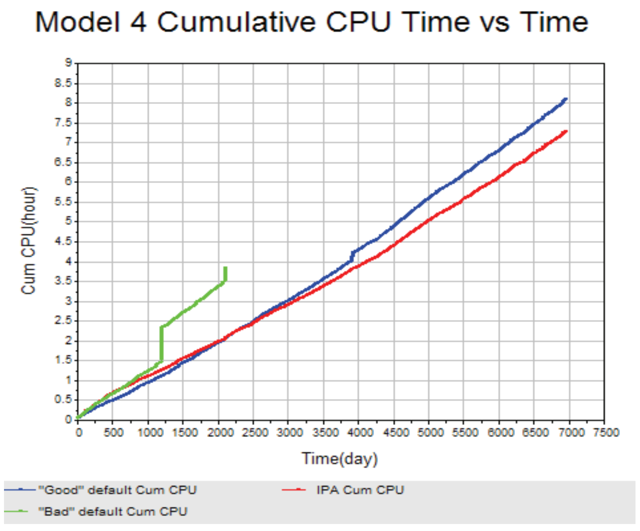

Figure 6. Results for Model 4.

We tested SAMG on Model, a black oil, three phase flow, sequential formulation model with 241K cells and 258 wells that ran for 6957 days. The IPA chose between two iterative methods (BiCGstab and GMRES), 3 smoothers (GS, ILU, MILU), six coarsening algorithms, and wells settings (exclude or keep the wells in the linear system). It performed very well, the total CPU time was 10% less that the "good" recommendation (the solver was 17% faster), and essentially the optimal combination was very close to the "good" recommendation with slightly more aggressive coarsening. The "bad" recommendation did not performed well. There were multiple divergences, and the run was stopped.

Conclusions and Outlook

We described an adaptive control on-line system to optimize the simulator performance by dynamically adjusting the solver parameters during the simulation. We start with a large set of parameters based on available solver options and quickly choose the best combinations that are continuously adapted during the simulation using the solver runtime performance measurements (e.g. solve CPU time) to guide the search. We use a genetic algorithm and Response Surface Methodology (RSM) to perform this search effectively.

Online experimentation with solver parameters is combined with actual solves performed on the successive time steps of a long-running simulation. At each time step, the matrix is solved just once. No additional solves are performed to assess/compare the performance of different solver settings for that matrix. Despite the fact that in the beginning of simulation some of the matrices are solved with solver settings that are not optimal, overall performance of reservoir simulator is improved.

This software system, called the Intelligent Performance Assistant (IPA), has been successfully integrated into ExxonMobil's proprietary reservoir simulator and deployed with it worldwide. The system can handle a large number of combinations of solver parameter combinations, currently in the order of 10^8, and consistently improves run time performance of real simulation models, frequently by 30% or more, compared to the performance with the default solver settings.

Similar adaptive control techniques can be applied to optimize other parts of reservoir simulator.

References

[1] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc, R. ClintWhaley, K. Yelick, "Self Adapting Linear Algebra Algorithms and Software", *IEEE Proceedings*, Vol 93, Issue 2, February 2005 , pp. 293-312.

[2] B. Norris, I. Veljkovic. Performance monitoring and analysis components in adaptive PDE-based simulations. Technical Report ANL/MCS-P1221-0105, Argonne National Laboratory, January 2005.

[3] T. Clees, L. Ganzer, "An efficient algebraic multigrid solver strategy for adaptive implicit methods in oil reservoir simulation", SPE Reservoir Simulation Symposium 2007 CD-ROM, Houston, Texas, USA, February 26-28, 2007.

[4] N. Fedorova, S. Terekhov, O. Diyankov, A. Usadi, B. Beckner, M. Ray, I. Mishev, "Simulation system and method", US2006/043286 (2006-11-08).

[5] R. Sutton, A. Barto, *Reinforcement Learning: An Introduction*, MIT Press, 1998.

[6] D.E. Moriarty, A.C. Schultz, J.J. Grefenstette, “Evolutionary Algorithms for Reinforcement Learning”, *Journal of Artificial Intelligence Research*, 11, 1999, pp. 241-276.

[7] O. Diyankov, S. Koshelev, S. Kotegov, I. Krasnogorov, N. Kuznetsova, V. Pravilnikov, B. Beckner, S. Maliassov, I. Mishev, A. Usadi, Sparsol sparse linear systems solver, *Russian Journal of Numerical Analysis and Mathematical Modelling*, 2007, Vol 22, Number 4, pp. 325-340.

[8] A. Krechel, K. Stuben, SAMG User‘s Manual, Release 21z, Fraunhofer Institute SCAI, Germany, 2005.

[**9**] K. Stuben, Private communications, 2006