SPE 163661

Developement of Algebraic Multigrid Solvers Using GPUs

Hui Liu, Song Yu, and Zhangxin Chen, University of Calgary

Abstract

In reservoir simulation, algebraic multigrid (AMG) solvers are the most effective solvers for linear systems arising from the discretization of pressure equations. The AMG solvers are also important components for many multi-stage preconditioners. In this paper, we introduce our work on developing these solvers on NVIDIA Tesla GPUs. A classical algebraic multigrid solver is developed. Numerical experiments are performed to test the efficiency of this GPU-based parallel AMG solver. The numerical experiments show that the GPU-based algebraic solver is sped up to 12.5 times faster than the corresponding CPU-based algebraic multigrid solver.

Introduction

In a typical reservoir simulator, both pressure equations and saturation equations are required to solve [1]. The pressure solution phase may control the whole solution time of linear systems arising from the entire reservoir equations, and this is the foundation behind the development of many multi-stage preconditioners [1, 2]. When solving the linear systems arising from reservoir simulation, commonly used methods - Krylov subspace solvers with Incomplete LU factorization (ILU) preconditioners [18, 19, 20, 25, 26] tend to slow down considerably as these linear systems grow larger. Developing fast and accurate solvers for the solution of pressure equations is important to large-scale reservoir simulators.

Pressure equations are elliptic or parabolic partial differential equations (PDE) depending on the consideration of the underlying reservoir (stationary or transient), and the linear systems arising from the discretization of these equations are often positive definite. If general Krylov subspace solvers and ILU preconditioners are applied, this combination works well when the size of these systems is moderate. However, it may significantly slow down when their size is big, such as for a reservoir system with the order of 100,000 grid blocks. According to our knowledge, the solution error consists of a high frequency error and a low frequency error. The high frequency part converges well while the low frequency part is hard to converge, which makes the Krylov subspace solvers deteriorate. Instead of using one grid (matrix), algebraic multigrid (AMG) solvers use several grids (matrices). The AMG solvers project a low frequency error to a coarser grid, and convert it to a high frequency error. In this case, the resulting high frequency error is easy to converge again. The convergence rate of these AMG solvers is optimal [3, 4, 5, 6] in terms of the number of iterations. Ruge and Stüben developed a classical AMG solver [3, 4, 5, 6, 9] and the RS coarsening strategy, which were the foundation behind the development of many other AMG solvers in the early age. Luby, Jones and Plassmann designed a parallel coarsening strategy CLJP for parallel computers [7, 10]. Henson and Yang proposed parallel coarsening strategies PMIS and HMIS [8, 12]. Besides, Yang and her collaborators developed a famous parallel AMG solver BoomerAMG [7, 8, 11, 12], which is the most famous parallel AMG solver/preconditioner for parallel computers. Haase et al. developed a parallel AMG solvers using a GPU cluster [13]. Bell et al. from NVIDIA investigated fine-grained parallelism of AMG solvers using a single GPU [14]. Researchers also developed efficient solvers and preconditioners for reservoir simulation using AMG, such as in the CPR and combinative preconditioners [1].

In this paper, our work on developing a parallel classical AMG solver using NVIDIA GPU is presented. Two coarsening strategies, two interpolation operators and several smoothers are implemented. Numerical experiments are also performed to show the efficiency of this solve.

GPU Computing

GPUs (Graphics Precessing Units) are special devices designed for manipulating large amounts of graphical data. Since each pixel data is independent of each other, they are processed in parallel. The architectures of GPUs are parallel [27, 28]. The NVIDIA Fermi GPU, as shown in Fig. 1 as an example, has 16 SMs (Streaming Multi-processors), and each SM has 32 SPs (Streaming Processors). This GPU has 512 streaming processors shown as green blocks while a normal CPU has only 2, 4, 6 or 8 cores. The NVIDIA Tesla C2070 has 448 streaming processors, and has a peak performance of 1030G FLOPS in single precision and a peak performance of 515G

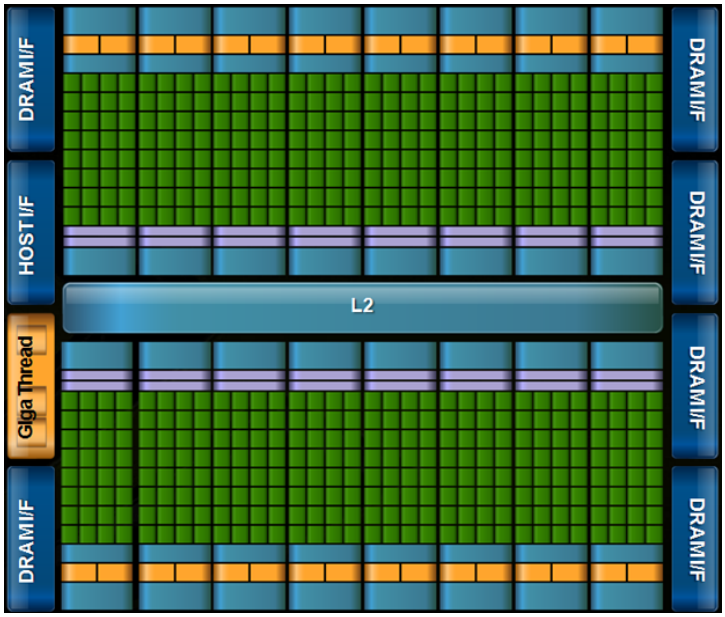

Figure 1: NVIDIA Fermi Architecture.

FLOPS in double precision, which are around 10 times faster than that of CPUs.

Each SM has its own L1 cache, shared memory and register. They share L2 cache, constant memory, texture memory and global memory. The global memory stores most of data, and is used to communicate with CPUs. The NVIDIA Tesla C2070 has 6 GB memory. Its memory speed is around 144 GB/s while the memory speed of CPUs is around 15 GB/s. The GPU memory is also about 10 times faster than the CPU memory.

GPU computing is becoming more and more popular due to its powerful float point performance. However, the architectures of GPUs are different from those of CPUs. Special tools and algorithms should be developed to utilize GPUs. NVIDIA provides CUDA Toolkit [27, 28] to help users develop high performance programs easily. Bell et al. from NVIDIA proposed a HYB matrix format to accelerate sparse matrix-vector multiplication and CUSP linear solver package [23, 24]. Saad et al. designed a sparse matrix-vector multiplication algorithm for the JAD matrix format and GPU-based GMRES solver [25, 26]. Chen et al. developed a HEC matrix format for GPU and the corresponding sparse matrix-vector multiplication algorithm [22], linear solvers and preconditioners [20, 21]. Naumov [15] and Chen et al. [20] studied parallel triangular solvers for GPUs. Haase et al. developed a parallel AMG solver for a GPU cluster [13], and researchers from NVIDIA investigated an AMG solver using a single GPU [14].

Algebraic Multigrid Solver

The linear system we solve is written as follows:

$$Ax = b, \tag{1}$$

where A is a sparse positive definite matrix ($A \in R^{n \times n}$), b is the right-hand vector ($b \in R^n$), and x is the unknown to be calculated ($x \in R^n$). This linear system may be derived from the discretization of a pressure equation by using the finite element method, the finite difference method or the finite volume method.

The structure of the standard V-cycle AMG solver is shown in Fig. 2. The solver has $L+1$ levels. The original linear system $Ax = b$ stands for the finest level ($l = 0$) and level L is the coarsest level.

An AMG solver consists of two phases, a setup phase and a solution phase. In the setup phase, the hierarchical system shown in Fig. 2 is assembled. On each level l except level L, a coarse grid is chosen. The coarse grid block size should be much larger than the current grid block size such that the linear system on the coarse grid is easy to solve, and the error on it should approximate the error on the current fine grid. Then a restriction operator R and an interpolation (prolongation) operator P are determined. The restriction operator projects an error from a finer grid onto a coarser grid and converts a low frequency error to a high frequency error while the interpolation operator transfers a solution on a coarser grid to that on a finer grid. In general, the restriction operator R is the transpose of the interpolation (prolongation) operator P; i.e., $R = P^T$. The setup phase on each level l ($0 \leq l < L$) is formulated in Algorithm 1.

The coarsening algorithms we use are the RS algorithm introduced by Ruge and Stüben [3, 4] and the CLJP algorithm introduced by Cleary, Luby, Jones and Plassmann. The prolongation operators are the classical one proposed by Ruge and Stüben and the one introduced in Haase's paper [13]. The second one is simpler and has lower complexity compared to the classical prolongation operator.

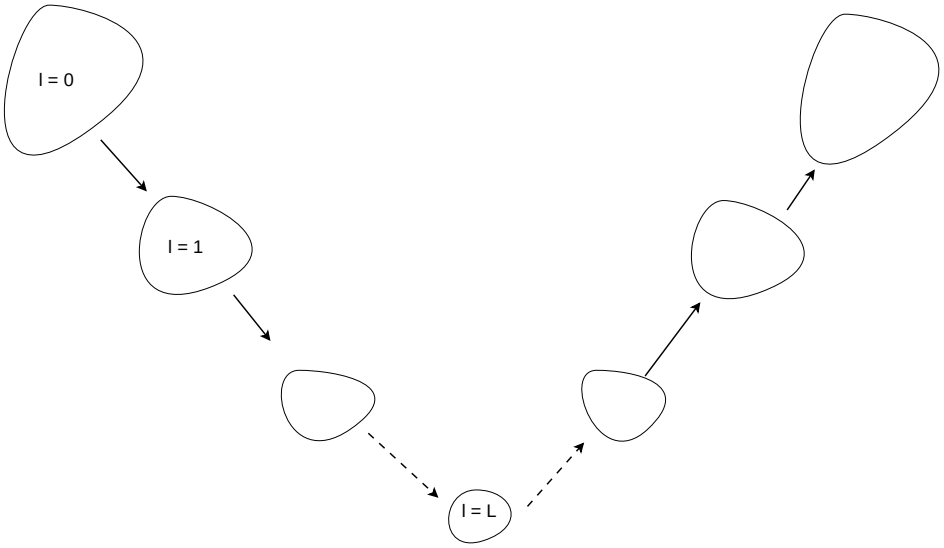

Figure 2: Structure of AMG solver.

Algorithm 1 AMG setup Algorithm

1: Calculate strength matrix S.
2: Choose coarsening nodes set ω_{l+1} according to strength matrix S, such that $\omega_{l+1} \subset \omega_l$.
3: Calculate prolongation operator P_l.
4: Derive restriction operator $R_l = P_l^T$.
5: Calculate coarse matrix A_{l+1}: $A_{l+1} = R_l \times A_l \times P_l$.
6: Choose pre-smoother S_l and post-smoother T_l.

Let P be the second prolongation operator to be calculated; we have

$$P_{i,j} = \begin{cases} 1 & i \text{ is a coarening node,} \\ 0 & i \text{ is a fine node and is weakly connected with j,} \\ 1/n_i & i \text{ is a fine node and is strongly connected with j,} \end{cases} \tag{2}$$

where $P_{i,j}$ is the (i, j)-th element of operator P and n_i is the number of coarsening nodes connected with fine node i.

For a given matrix A, the right-hand side b and the initial solution x, the pre-smoother S_l and the post-smoother T_l have the same form so that

$$x^{m+1} = x^m + \alpha M^{-1}(b - Ax^m) = x^m + \alpha M^{-1} r. \tag{3}$$

The difference is how to choose matrix M and coefficient α, where α is positive and often equals to 1. If M is chosen as the diagonal matrix of A, we have a Jacobi smoother, and if α is less than 1, we have a damped Jacobi. If M is the block diagonal part of A, we have a block Jacobi smoother. If M is the lower part of matrix A, we have the so-called Gauss-Seidel smoother. In our solver package, we have implemented the Jacobi, damped Jacobi, block Jacobi, weighted Jacobi, Gauss-Seidel, block Gauss-Seidel, approximate inverse, domain decomposition, polynomial and hybrid smoothers. For the block Jacobi, Gauss-Seidel, block Gauss-Seidel and domain decomposition smoothers [16, 29], triangular systems are required to solve. Parallel GPU-based triangular solvers [20] are applied here.

The solution phase of AMG is recursive and is formulated in Algorithm 2, which shows one iteration of AMG. On each level l, the algorithm requires A_l, R_l, P_l, S_l and T_l. A_l, R_l and P_l are matrices. The pre-smoother S_l and post-smoother T_l are typically damped Jacobi, Gauss-Seidel, block Gauss-Seidel and domain decomposition smoothers.

Remark: The setup phase is difficult to parallelize, which has been proven by many reseachers [14]. We use CPU to assemble the AMG system and use GPU to solve the AMG system. We will attempt to parallelize the setup phase in the future.

Sparse Matrix-vector Multiplication

The NVIDIA GPUs have different architectures from general purpose CPUs [27, 28]. They access GPU memory in a coalesced way, which means if the memory access is arranged well, threads in a grid block can fetch data in one or a few rounds. In this case, the memory access speed is the highest. Since GPUs are emerging parallel devices, algorithms that work well on CPUs may not work effectively. New data structures and algorithms for GPUs are required to design. To speed sparse matrix-vector multiplication operation, we propose a hybrid matrix format HEC shown in Fig. 3. An HEC matrix consists of two parts: ELL part and CSR part. The ELL

Algorithm 2 AMG Solution Algorithm: mg_solve(l)

Require: b_l, x_l, A_l, R_l, P_l, S_l, T_l, $0 \leq l < L$

$b_0 = b$
if $(l < L)$ **then**
  $x_l = S_l(x_l, A_l, b_l)$ ▷ Pre-smoothing
  $r = b_l - A_l x_l$
  $b_{r+1} = R_l r$ ▷ Restriction
  mg_solve$(l+1)$ ▷ Next level
  $x_l = x_l + P_l x_{l+1}$ ▷ Prolongation
  $x_l = T_l(x_l, A_l, b_l)$ ▷ Post-smoothing
else
  $x_l = A_l^{-1} b_l$
end if
x = x_0

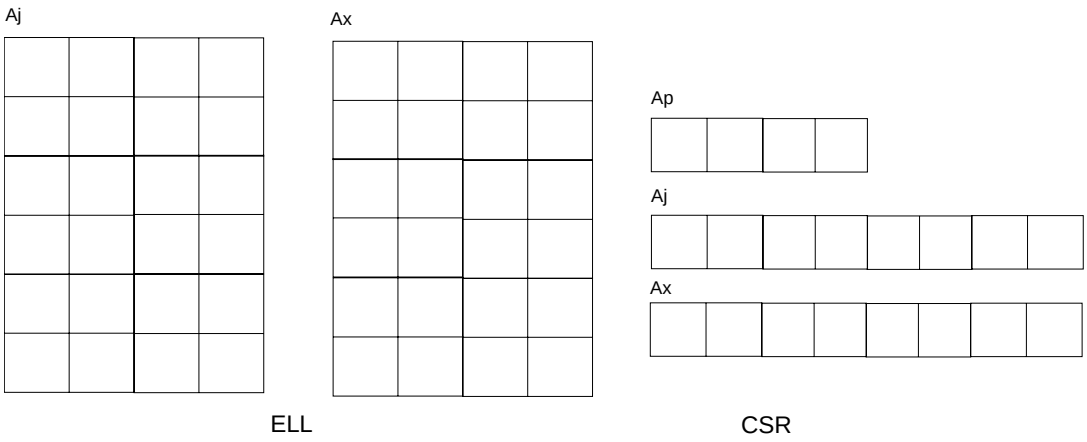

Figure 3: HEC matrix format

Algorithm 3 Sparse Matrix-Vector Multiplication, $y = Ax$

for i = 1: n **do** ▷ ELL, Use one GPU kernel to deal with this loop
  the i-th thread calculate the i-th row of ELL matrix; ▷ Use one thread
end for

for i = 1: n **do** ▷ CSR, Use one GPU kernel to deal with this loop
  the i-th thread calculate the i-th row of CSR matrix; ▷ Use one thread
end for

matrix is regular and each row has the same length. When being stored in GPU, the matrix is in column-major order. The CSR matrix stores the irregular part. The HEC matrix is friendly to ILU-like preconditioners [20].

A sparse matrix-vector multiplication kernel [22] is developed as shown in Algorithm 3. One thread calculates one row. The ELL part is calculated first and the CSR part is then processed. More details can be read in reference [22].

Benchmarks

Four numerical experiments are performed on our workstation with Intel Xeon X5570 CPUs and NVIDIA Tesla C2050/C2070 GPUs. The operating system is CentOS 6.3 X86_64 with CUDA Toolkit 5.1 and GCC 4.4. All CPU codes are compiled with -O3 option. The type of float point number is double.

Example 1. The matrix tested is from a two-dimensional elliptic partial differential equation. Its grid size is 1,000x1,000. The dimension of the matrix is 1,000,000 and the number of non-zeros is 4,996,000. The RS coarsening strategy and damped Jacobi smoother are applied. The AMG solver has eight levels. The termination criterium is 1e-6.

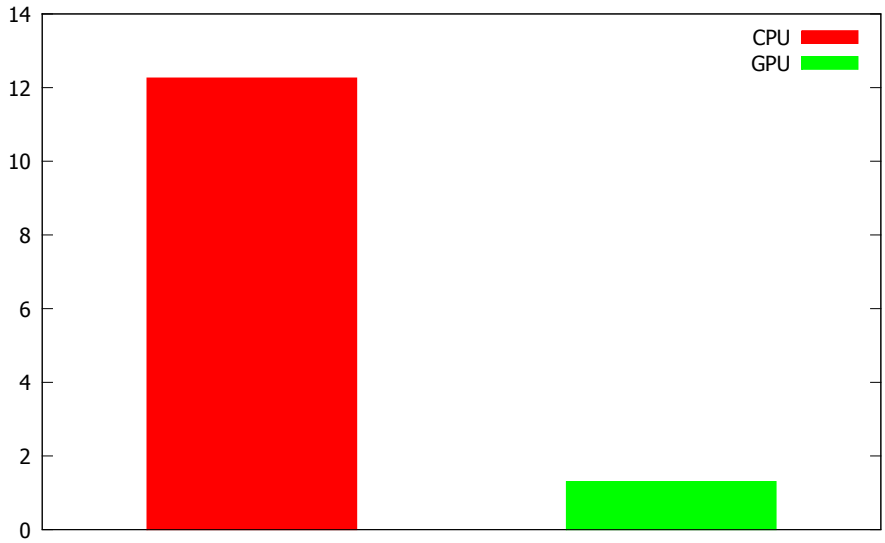

Figure 4: Running time of Example 1.

The GPU-based AMG solver terminates in 36 iterations and so does the CPU-based AMG solver. The performance data is shown in Fig. 4. The GPU-based AMG solver uses 1.305s of the solution time shown in green color while the CPU-based solver takes 12.261s shown in red color. The speedup of the parallel AMG solver is 9.33. Our parallel AMG solver is efficient for large-scale problems.

Example 2. The matrix tested is also from a two-dimensional elliptic partial differential equation. Its grid size is 1,500x1,500. The dimension of the matrix is 2,250,000 and the number of non-zeros is 11,244,000. The RS coarsening strategy and weighted Jacobi smoother are applied. The AMG solver has 8 levels. The termination criterium is 1e-6.

The matrix in the second example is much larger than the first one. Both the GPU-based AMG solver and the CPU-based solver use 31 iterations. The performance data is shown in Fig. 5. Our solvers are effective for this problem. The GPU-based solver uses 2.584s shown in green color and the CPU-based solver takes 24.501s shown in red color. The speedup of the GPU-based AMG solver is 9.42.

Example 3. The matrix tested is from a three-dimensional elliptic partial differential equation. Its grid size is 100x100x100. The dimension of the matrix is 1,000,000 and the number of non-zeros is 6,940,000. The RS coarsening strategy and damped Jacobi smoother are applied. The AMG solver has 8 levels. The termination criterium is 1e-6.

The performance data is shown in Fig. 6. Both solvers terminate in 21 iterations. The GPU takes 1.127s to solve the linear system shown in green color while the CPU solver takes 10.8s to solve it shown in red color. The GPU-based parallel AMG solver is 9.58 times faster than the CPU-based AMG solver.

Example 4. The matrix tested is also from a three-dimensional elliptic partial differential equation. Its grid size is 130x130x130. The dimension of the matrix is 2,197,000 and the number of non-zeros is 15,277,600. The RS coarsening strategy and weighted Jacobi smoother are applied. The AMG solver has 8 levels. The termination criterium is 1e-6.

The performance data of this example is shown in Fig. 7. The two AMG solvers terminate in 25 iterations. The GPU-based AMG solver takes 3.02s shown in green color while the CPU-based AMG solver takes 37.86s. The speedup of the GPU-based AMG solver for this matrix is around 12.47. This parallel solver is much more efficient than the CPU-based solver.

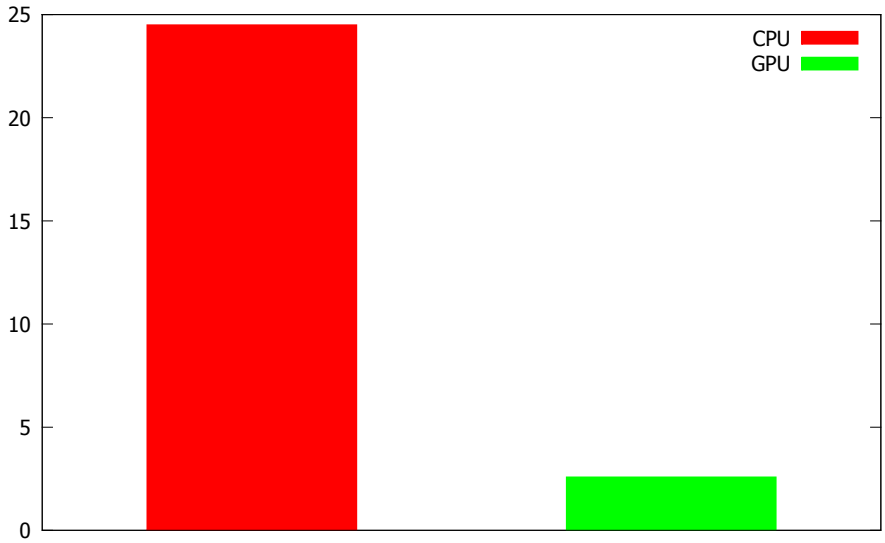

Figure 5: Running time of Example 2.

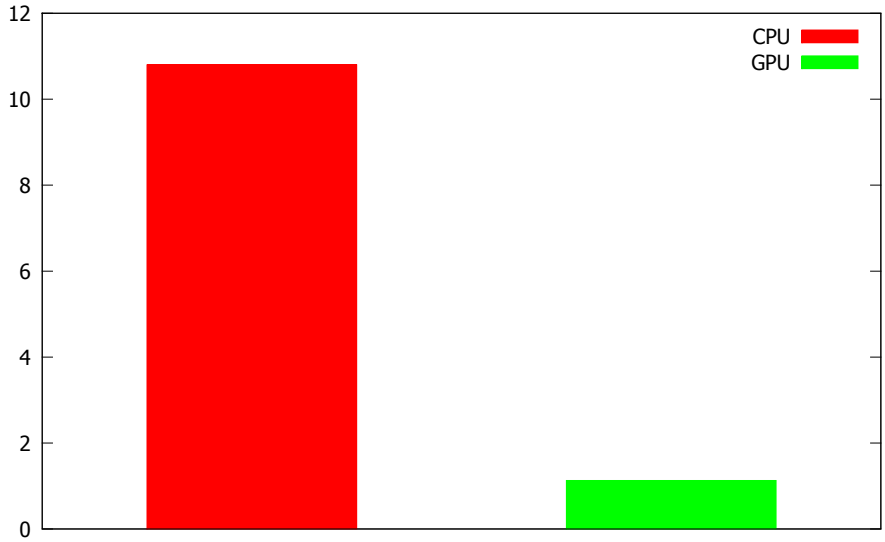

Figure 6: Running time of Example 3.

Conclusions

We have developed a GPU-based classical AMG solver. When solving a positive definite linear system, this parallel solver is much more efficient than a CPU-based sequential AMG solver. From the numerical experiments above, we can see that this parallel solver is up to 12.5 times faster than the sequential solver. However, the setup phase is still difficult to parallelize and efforts should be made to overcome this problem in the future.

Acknowledgements

The support of Department of Chemical and Petroleum Engineering, University of Calgary and Reservoir Simulation Group is gratefully acknowledged. The research is partly supported by NSERC/AIEE/Foundation CMG and AITF Chairs.

References

[1] Z. Chen, G. Huan, and Y. Ma, *Computational Methods for Multiphase Flows in Porous Media*, in the Computational Science and Engineering Series, Vol. 2 (SIAM, Philadelphia, 2006).

[2] Xiaozhe Hu, Wei Liu, Guan Qin, Jinchao Xu, Yan Yan, Chensong Zhang, Development of A Fast Auxiliary Subspace Preconditioner for Numerical Reservoir Simulators, SPE Reservoir Characterisation and Simulation Conference and Exhibition, 9C11

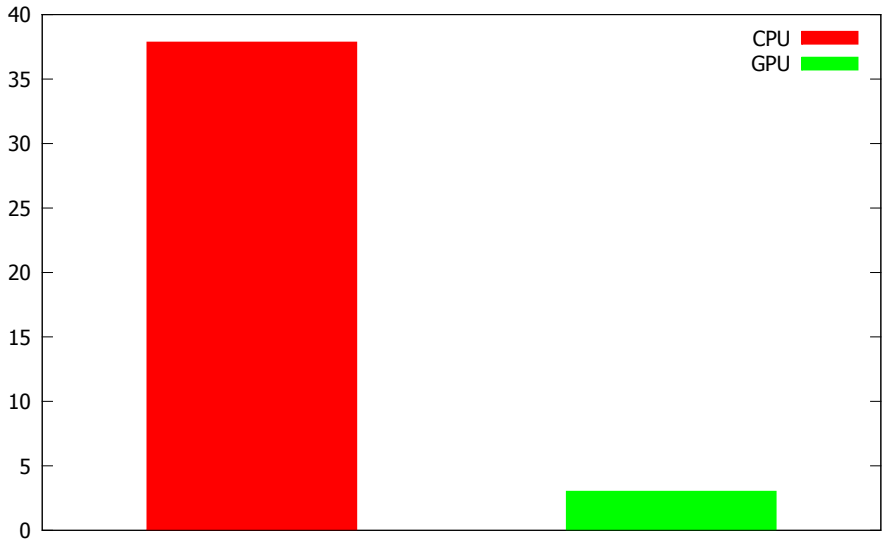

Figure 7: Running time of Example 4.

October 2011, Abu Dhabi, UAE, SPE-148388-MS.

[3] K. Stüben, A review of algebraic multigrid, Journal of Computational and Applied Mathematics Volume 128, Issues 1-2, 2001, 281–309.

[4] J.W. Ruge and K. Stüben, Algebraic multigrid (AMG), in: S.F. McCormick (Ed.), Multigrid Methods, Frontiers in Applied Mathematics, Vol. 5, SIAM, Philadelphia, 1986.

[5] A. Brandt, S.F. McCormick and J. Ruge, Algebraic multigrid (AMG) for sparse matrix equations D.J. Evans (Ed.), Sparsity and its Applications, Cambridge University Press, Cambridge, 1984, 257–284.

[6] C. Wagner, Introduction to Algebraic Multigrid, Course notes of an algebraic multigrid course at the University of Heidelberg in the Wintersemester, 1999.

[7] Robert Falgout, Andy Cleary, Jim Jones, Edmond Chow, Van Henson, Chuck Baldwin, Peter Brown, Panayot Vassilevski, and Ulrike Meier Yang, Hypre home page, 2011. http://acts.nersc.gov/hypre

[8] Van Emden Henson and Ulrike Meier Yang, BoomerAMG: a Parallel Algebraic Multigrid Solver and Preconditioner, Applied Numerical Mathematics, 41, 2000, 155–177.

[9] Panayot S. Vassilevski, Lecture Notes on Multigrid Methods, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, 2010.

[10] Andrew J. Cleary, Robert D. Falgout, Van Emden Henson, Jim E. Jones, Thomas A. Manteuffel, Stephen F. McCormick, Gerald N. Miranda, and John W. Ruge, Robustness and Scalability of Algebraic Multigrid, SIAM J. Sci. Comput., 21, 2000, 1886–1908.

[11] R.D. Falgout, An Introduction to Algebraic Multigrid, Computing in Science and Engineering, Special Issue on Multigrid Computing, 8, 2006, 24–33.

[12] U.M. Yang, Parallel Algebraic Multigrid Methods - High Performance Preconditioners, chapter in Numerical Solution of Partial Differential Equations on Parallel Computers, A.M. Bruaset and A. Tveito, eds., Springer-Verlag, 51, 2006, 209–236.

[13] G. Haase, M. Liebmann, C. C. Douglas and G. Plank, A Parallel Algebraic Multigrid Solver on Graphics Processing Units, HIGH PERFORMANCE COMPUTING AND APPLICATIONS, 2010, 38-47.

[14] Nathan Bell, Steven Dalton and Luke Olson, Exposing Fine-Grained Parallelism in Algebraic Multigrid Methods, NVIDIA Technical Report NVR-2011-002, June 2011

[15] M. Naumov, Parallel solution of sparse triangular linear systems in the preconditioned iterative methods on the GPU, NVIDIA Technical Report, June 2011.

[16] X.-C. Cai and M. Sarkis, A restricted additive Schwarz preconditioner for general sparse linear systems, SIAM J. Sci. Comput., 21, 1999, 792-797.

[17] R. Grimes, D. Kincaid, and D. Young, ITPACK 2.0 User's Guide, Technical Report CNA-150, Center for Numerical Analysis, University of Texas, August 1979.

[18] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. Van der Vorst , Templates for the solution of linear systems: building blocks for iterative methods, 2nd Edition, SIAM, 1994.

[19] Y. Saad, Iterative methods for sparse linear systems (2nd edition), SIAM, 2003.

[20] H. Liu, S. Yu, Z. Chen, B. Hsieh and L. Shao, Parallel Preconditioners for Reservoir Simulation on GPU, SPE Latin American and Caribbean Petroleum Engineering Conference held in Mexico City, Mexico, 16-18 April 2012, SPE 152811-PP.

[21] Song Yu, Hui Liu, Zhangxin Chen, Ben Hsieh, and Lei Shao, *GPU-based Parallel Reservoir Simulation for Large-scale Simulation Problems*, *SPE Europec/EAGE Annual Conference* (Copenhagen, Denmark, 2012).

[22] Hui Liu, Song Yu, Zhangxin Chen, Ben Hsieh and Lei Shao, *Sparse Matrix-Vector Multiplication on NVIDIA GPU*, International Journal of Numerical Analysis and Modeling, Series B, Volume 3, No. 2 (2012).

[23] N. Bell and M. Garland, Efficient sparse matrix-vector multiplication on CUDA, NVIDIA Technical Report, NVR-2008-004, NVIDIA Corporation, 2008.

[24] N. Bell and M. Garland, Implementing sparse matrix-vector multiplication on throughput-oriented processors, Proc. Supercomputing, November 2009, 1-11.

[25] H. Klie, H. Sudan, R. Li, and Y. Saad, Exploiting capabilities of many core platforms in reservoir simulation, SPE RSS Reservoir Simulation Symposium, 21-23 February 2011

[26] R. Li and Y. Saad, GPU-accelerated preconditioned iterative linear solvers, Technical Report umsi-2010-112, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2010.

[27] NVIDIA Corporation, Nvidia CUDA Programming Guide (version 3.2), 2010.

[28] NVIDIA Corporation, CUDA C Best Practices Guide (version 3.2), 2010.

[29] G. Karypis and V. Kumar, A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs, SIAM Journal on Scientific Computing, 20(1), 1999, 359-392.