GPU-based Parallel Reservoir Simulators

Zhangxin Chen[1], Hui Liu[1], Song Yu[1], Ben Hsieh[1] and Lei Shao[1]

Key words: GPU computing, reservoir simulation, linear solver, parallel

1 Introduction

Nowadays reservoir simulators are indispensable tools to reservoir engineers. They are widely used in the optimization and prediction of oil and gas production. However, for large-scale reservoir simulation, computational time is usually too long. A case with over one million grid blocks may run weeks or even months. High performance processors and well-designed software are demanded. Though today's CPUs (Central Processing Unit) are much more powerful than before, performance of single CPU tends to slow down due to material and energy consumption and heat dissipation issues. Processor vendors have begun to move to multiple processing units, which form two major directions: multi-core CPUs and many-core GPUs [11].

In reservoir simulation, numerical methods like the finite difference and finite volume methods [7] are often used to discretize the mathematical models. Linear and nonlinear systems arising from the discretized models by those methods are sparse, which are usually time-consuming and difficult to solve. Krylov subspace solvers [18, 1] are general methods to solve these linear systems, and for large-scale reservoir simulation with over one million grid blocks, a reservoir simulator may take 90% or even more time on the solution of the linear systems. Fast and accurate linear and nonlinear solvers are essential to reservoir simulators. Saad et al. developed the GMRES solver for general unsymmetric linear systems [1, 18] and Vinsome designed the ORTHOMIN solver, which was originally developed for reservoir simulators [19]. PCG, BICGSTAB, algebraic multigrid and direct linear solvers were also proposed. Commonly used preconditioners were also developed, such as Incomplete LU (ILU) factorization, domain decomposition, algebraic multigrid, and multi-stage preconditioners [1, 18]. GPUs (Graphics Processing Unit) are usually used for display. Since each pixel can be processed simultaneously, GPUs are designed in such a way that they can manipulate data in parallel. Their float point performance and memory speed are very high [16, 15]. In general, GPUs are ten times faster than general CPUs [16, 15], which makes them powerful devices

[1]Department of Chemical and Petroleum Engineering, University of Calgary, 2500 University Drive NW, Calgary, AB, Canada, T2N 1N4, e-mail: {zhachen}{hui.j.liu}{soyu}{bhsieh}{lshao}@ucalgary.ca

for parallel computing. Since GPUs are designed for graphics processing and not for general tasks, their architectures are different from those of CPUs. Hence new algorithms for GPUs should be developed to utilize GPUs' performance. NVIDIA developed a hybrid matrix format, the corresponding sparse matrix-vector multiplication kernel and a GPU-based linear solver package CUSP [4, 5, 2]. Bell et al. from NVIDIA also investigated fine-grained parallelism of AMG solvers using a single GPU [3]. Saad et al. developed a sparse matrix-vector multiplication kernel for JAD matrix format and the GMRES solver [11]. Chen et al. designed a new matrix format, HEC, a new matrix-vector multiplication kernel, Krylov subspace solvers, algebraic multigrid solvers and several preconditioners [20, 13, 14, 12]. Haase et al. developed a parallel AMG solver for a GPU cluster [10]. In this paper, we will introduce our work on developing a GPU-based parallel iterative linear solver package and applying it to reservoir simulation.

The framework is as follows: In §2.1, GPU computing, our parallel linear solvers and GPU-based reservoir simulators are introduced. In §3, numerical experiments are presented.

2 Parallel Reservoir Simulator

In this section, we will propose GPU-based parallel linear solvers and preconditioners, and apply these solvers to reservoir simulation.

2.1 GPU Computing

The NVIDIA Fermi GPU, Tesla C2070, has 14 SMs (Streaming Multi-processors), and each SM has 32 SPs (Streaming Processors). That's 448 streaming processors in total while a normal CPU has only 2, 4, 6 or 8 cores. The GPU architectures are being developed rapidly. Each SM of the new Tesla Kepler GPUs has 192 SPs, much more than Fermi GPUs. The Tesla Kepler K20X has 2688 processors in total. At this moment we are using Fermi GPUs. The NVIDIA Fermi GPU, Tesla C2070, has a peak performance of 1030G FLOPS in single precision and a peak performance of 515G FLOPS in double precision, which are around 10 times faster than that of CPUs. The Tesla Kepler K20X GPU has a peak performance of 1310G FLOPS in double precision and a peak performance of 3950G FLOPS in single precision [17].

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. The Tesla Kepler K20X GPU has a memory speed of 250GB/s [17].

NVIDIA provides CUDA Toolkit [16, 15] to help users develop high performance programs.

2.2 Parallel Linear Solvers

GPUs have different architectures from general purpose CPUs. The NVIDIA GPUs access global memory in a coalesced way, which means that 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 and codes are efficient. GPUs are emerging parallel devices. However, algorithms that work well on CPUs may not work effectively on GPUs [12]. We develop a new matrix format and the corresponding sparse matrix-vector multiplication kernel (SPMV) to accelerate iterative linear solvers. The new matrix format, HEC (Hybrid of ELL and CSR format), is shown in Fig. 1. A HEC matrix has two submatrices, ELL matrix and CSR matrix. The ELL submatrix stores the regular part of a given matrix and the CSR submatrix stores the irregular part of the given matrix. The ELL submatrix is stored in column-major manner. The main advantage of HEC is that it is friendly to ILU-related preconditioners. When we store a lower triangular matrix, it's clear that the last element in ELL or CSR part is a diagonal element and elements before the diagonal one are easy to recognize. If we use HYB, which is efficient for SPMV, the irregular part is stored in a COO matrix, we don't know which element is the diagonal one or elements before it. The second advantage is that SPMV algorithm for HEC is simple and implementation is straightforward. The psudo-codes for SPMV operation is listed in Alg. 1. Alg. 1 is for calculating $y = Ax$. Other related BLAS 2 operations are similar. We also develop BLAS 1 operations. A typical operation $y = \alpha x + \beta y$ is shown in Alg. 2.

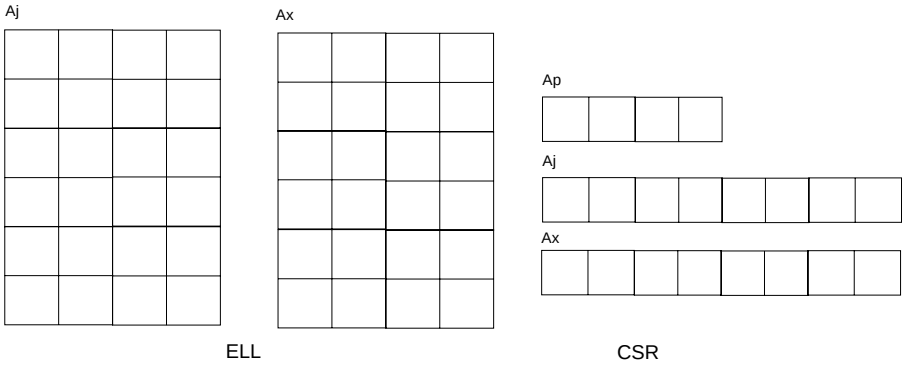

Fig. 1 HEC matrix format

We consider the following linear system:

Algorithm 1 Sparse Matrix-Vector Multiplication, $y = Ax$

for i = 1: n **do**
  the i-th thread calculate the i-th row of ELL matrix; {ELL part, use one thread for each row}
end for

for i = 1: n **do**
  the i-th thread calculate the i-th row of CSR matrix; {CSR part, use one thread for each row}
end for

Algorithm 2 BLAS 1 subroutine, $y = \alpha x + \beta y$

for i = 1: n **do**
  $y[i] = \alpha x[i] + \beta y[i]$; {Use one GPU kernel to deal with this loop}
end for

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

where A is a nonsingular $n \times n$ matrix, b is the right-hand side and x is the solution to be solved for. Krylov subspace solvers are general purpose methods for the solution of linear systems. Based on BLAS 1 and BLAS 2 subroutines we have implemented several Krylov subspace solvers and algebraic multigrid (AMG) solvers, including GMRES, CG, BICGSTAB, ORTHOMIN, classical AMG and smoothed aggregation AMG solvers.

In practice, an equivalent linear system of equation (1) is solved:

$$M^{-1}Ax = M^{-1}b, \tag{2}$$

where M is a nonsingular $n \times n$ matrix, called a preconditioner or left-preconditioner. When we choose a preconditioner M, a general principle is that M is an good approximation of A and in this case, it means that the product of M^{-1} and A approximates the unit matrix I well. The condition number of $M^{-1}A$ is much smaller than that of A and the preconditioned linear system (2) is much easier to solve compared to the original equation (1). Meanwhile, M should also be easy to construct and be easy to solve. We have implemented ILU(k)[18], block ILU(k) [18], ILUT(tol, p)[18], block ILUT(tol, p)[18], domain decomposition, approximate inverse, polynomial and algebraic multigrid preconditioners. For many preconditioners, an upper triangular linear system and a lower triangular linear system are required to solve. GPU-based parallel triangular solvers are developed to speed the solving of triangular linear systems. Details can be read in [13].

2.3 Reservoir Simulator

The reservoir simulator generates a Jacobian matrix in each Newton iteration. As mentioned above, the solution of linear systems in each Newton iteration may dom-

inate the whole simulation time. For a large-scale black oil simulator, the linear solvers take over 90% of the running time. It is necessary for us to apply high performance linear solvers. We replace CPU-based linear solvers with GPU-based parallel linear solvers. The linearized systems are transferred to GPUs, and then GPUs solve the linear system using hundreds of microprocessors in parallel and transfer the solution back to the simulator. By applying GPU-based linear solvers, reservoir simulators run much faster and it is possible for personal computers to run larger cases.

3 Numerical Experiments

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 and in this paper only one CPU core is employed. The type of float point number is double and blocks mean the number of sub-domains in this section.

Example 1. Several different SPMV algorithms [2, 4] are compared using matrices from the University of Florida sparse matrix collection [9]. Performance data [14] is collected in Tab 1 and numbers in the table mean speedup of GPU-based algorithms.

Table 1 Example 1: Performance of SPMV

Matrix	CSR	ELL	HYB	HEC
msc23052	0.71	1.55	2.29	2.50
cfd2	1.41	8.06	6.86	11.61
ESOC	1.06	11.56	11.56	11.61
cage13	1.25	7.56	9.42	11.04
af_shell8	1.17	8.66	9.63	11.12
parabolic_fem	4.36	9.97	7.56	10.00
Emilia_923	0.97	6.64	8.36	8.65
atmosmodd	2.94	14.54	14.50	14.57
Serena	0.98	1.79	7.26	7.29
SPE10	1.13	1.24	11.15	10.27

In Tab 1, the first column stands for matrix and others mean speedup using different SPMV algorithms. We can see that algorithms using HYB and HEC matrix formats are always efficient and the performance of our HEC matrix format is better than that of HYB.

Example 2. The matrix used here is from SPE10 [7, 8]. The SPE10 problem is a standard benchmark for the black oil simulator. The problem is highly heterogenous and it is designed to solve hard. The grid size for SPE10 is 60x220x85. The number of unknowns is 2,188,851 and the number of non-zeros is 29,915,573. The linear

solver employed is GMRES(20), and block ILU(k), block ILUT(tol, p) and domain decomposition preconditioners are applied as preconditioners. Here RAS (Restricted Additive Schwarz) a domain decomposition method developed by Cai[6]. Performance data is collected in Table 2. For this example, loading time is always less than 1s.

Table 2 Example 2: Performance of SPE10

Preconditioner	Parameters	Setup (s)	CPU (s)	GPU (s)	Speedup
BILU(0)	(1, 0)	4.23	76.65	12.42	6.16
BILU(0)	(16, 0)	2.04	92.80	12.78	7.25
BILU(0)	(128, 0)	1.76	86.22	12.05	7.14
BILU(0)	(512, 0)	1.63	92.82	12.87	7.20
BILUT	(1, 0)	4.76	23.50	9.03	2.60
BILUT	(16, 0)	2.49	32.00	7.17	4.46
BILUT	(128, 0)	1.94	42.51	7.82	5.42
BILUT	(512, 0)	1.81	47.44	8.80	5.37
RAS	(256, 1)	9.28	106.61	14.36	7.41
RAS	(1024, 1)	11.91	110.36	16.36	6.73
RAS	(256, 2)	10.99	107.89	17.28	6.23
RAS	(1024, 2)	15.28	138.60	20.93	6.61

In Table 2, the first column stands for preconditioners. The second column stands for parameters used for each preconditioner and they are the number of sub-domains and overlap respectively. The others are for setup time, running time and speedup, respectively. From the table, we can speed ILU(0) 6.2 times faster. The speedup can be higher if we increase the number of blocks. The average speedup of BILU(0) is about 7. In this example, BILUT is the most effective preconditioner. It always takes the least running time. However, due to its irregular non-zero pattern, its speedup is lower than that of the other two precondtioners. Since there is not enough memory on GPU, the maximum number of blocks for RAS in this case is 1024. The average speedup of RAS is 6.5.

Example 3. The matrix used here is also from SPE10, which has the same size with the one we use in Example 2. Its pressure part has a dimension of 1,094,421 and has 7,478,141 non-zeros. Here we solve pressure matrix using algebraic multigrid solver and the entire matrix is solved by GMRES (40) with CPR-AMG (Constrained Pressure Residual) preconditioner [7]. To compare, the entire matrix is also solved using GMRES(40) with ILU(0) preconditioner. Classical AMG solver is applied here. The standard interpolator and damped Jacobi smoother are applied. V-cycle is used for solving phase and the coarsest level is solved using GMRES. The AMG solver has 8 levels. The termination criterium is 1e-6. Performance data is shown in Tab 3.

The speedup of AMG is 6.49 when solving pressure matrix. When CPR-AMG is used as a preconditioner, the GMRES(40) converges quickly. CPR-AMG is much

Table 3 Example 3: Performance of AMG

Solver	Preconditioner	Setup (s)	CPU (s)	GPU (s)	Speedup	Residual	Iterations
AMG		3.96	6.86	1.04	6.49	4.88e-7	11
GMRES(40)	CPR-AMG	8.43	185.42	27.51	6.74	3.91e-7	9
GMRES(40)	ILU(0)	4.3	1037.26	195.96	5.29	9.42e-4	100

more efficient than ILU(0). And a speedup of 6.74 is obtained. From Tab 3, we can also see that the setup phase takes too much time, which should be optimized in future.

Example 4. This example is to test the speedup of our GPU solver in the whole black oil simulator. The case is SPE10 simulation in 100 days. The grid size for SPE10 is 60x220x85. The solver is GMRES(20). Performance data is shown in Tab 4.

Table 4 Example 4: Performance of black oil simulator

Preconditioner	Blocks	CPU (s)	GPU (s)	Speedup
BILU(0)	1	49610.28	7721.09	6.43
BILU(0)	4	53350.63	8524.31	6.26
BILU(0)	8	54286.07	8720.25	6.23
BILUT	1	19533.45	9008.22	2.17
BILUT	4	23187.85	8670.53	2.67
BILUT	8	21718.45	7908.42	2.75

As shown in Tab 4, the block ILU(0) preconditioner achieves a speedup of 6.2 while the speedup of block ILUT is much lower. The reason is that the non-zero pattern of ILUT is irregular and the ILU factorization takes too much time.

4 Conclusion

We have developed a GPU-based parallel linear solver package. When solving matrices from reservoir simulation, the parallel solvers are much more efficient than CPU-based linear solvers. However, efforts should be made to improve the setup phase of domain decomposition, the factorization of ILUT and parallelism of block ILUT preconditioner.

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/AIEES/Foundation CMG and AITF Chairs.

References

1. Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., C., R., der Vorst H., V.: Templates for the solution of linear systems: building blocks for iterative methods, 2nd Edition. SIAM (1994)
2. Bell N.and Garland, M.: Cusp: Generic parallel algorithms for sparse matrix and graph computations. (2012). URL `http://cusp-library.googlecode.com`. Version 0.3.0
3. Bell, N., Dalton, S., Olson, L.: Exposing fine-grained parallelism in algebraic multigrid methods. (2011)
4. Bell, N., Garland, M.: Efficient sparse matrix-vector multiplication on CUDA. (2008)
5. Bell, N., Garland, M.: Implementing sparse matrix-vector multiplication on throughput-oriented processors. In: Proc. Supercomputing (2009)
6. Cai X.-C.and Sarkis, M.: A restricted additive schwarz preconditioner for general sparse linear systems. SIAM J. Sci. Comput. pp. 792–797
7. Chen, Z., Huan, G., Ma, Y.: Computational Methods for Multiphase Flows in Porous Media. SIAM (2006)
8. Christie, M., Blunt, M.: Tenth spe comparative solution project: A comparison of upscaling techniques. SPE Reservoir Engineering and Evaluation. pp. 308–317
9. Davis, T.A.: University of florida sparse matrix collection, na digest. (1994)
10. Haase, G., Liebmann, M., Douglas, C.C., Plank, G.: A parallel algebraic multigrid solver on graphics processing units. HIGH PERFORMANCE COMPUTING AND APPLICATIONS. pp. 38–47 (2010)
11. Klie, H., Sudan, H., Li, R., Saad, Y.: Exploiting capabilities of many core platforms in reservoir simulation. In: SPE RSS Reservoir Simulation Symposium (2011)
12. Liu, H., Yu, S., Chen, Z.: Development of algebraic multigrid solvers using gpus. In: SPE RSS Reservoir Simulation Symposium. (2012)
13. Liu, H., Yu, S., Chen, Z., Hsieh, B., Shao, L.: Parallel preconditioners for reservoir simulation on gpu. In: SPE Latin America and Caribbean Petroleum Engineering Conference. (2012)
14. Liu, H., Yu, S., Chen, Z., Hsieh, B., Shao, L.: Sparse matrix-vector multiplication on nvidia gpu. International Journal of Numerical Analysis and Modeling. (2012)
15. NVIDIA: Cuda c best practices guide (version 3.2). (2010)
16. NVIDIA: Nvidia cuda programming guide (version 3.2). (2010)
17. NVIDIA: Nvidia tesla gpu products. (2012). URL `http://www.nvidia.com/object/tesla-servers.html`
18. Saad, Y.: Iterative methods for sparse linear systems (2nd edition). SIAM (2003)
19. Vinsome, P.: an iterative method for solving sparse sets of simultaneous linear equations. In: SPE Symposium on Numerical Simulation of Reservoir Performance (1976)
20. Yu, S., Liu, H., Chen, Z., Hsieh, B., Shao, L.: Gpu-based parallel reservoir simulation for large-scale simulation problems. In: SPE EAGE Annual Conference & Exhibition, SPE-152271 (2012)