Reservoir Simulation on NVIDIA Tesla GPUs

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

ABSTRACT. In this paper, we introduce our work on accelerating a black oil simulator using GPU-based parallel iterative linear solvers. We develop iterative linear solvers and several commonly used preconditioners on NVIDIA Tesla GPUs. These solvers and preconditioners are coupled with our in-house reservoir simulator. Numerical experiments show that our GPU-based black oil simulator is sped up around six times faster than a pure CPU-based simulator.

1. Introduction

For large scale reservoir simulation, especially when the number of grid blocks is over millions, the running time of reservoir simulators can be very long. To our experience, solving a linear system arising from reservoir simulation is the most time-consuming part. For SPE 10 [**CMFPM**], for example, over 95% running time is spent on the solution of linear systems. It is clear that if the linear systems are solved efficiently, the whole simulation can be sped up.

GPUs are now much more powerful in float point calculation than conventional C-PUs [**NVCUDAPG, CUDABPG**]. They have become very popular nowadays and have been used in many scientific applications [**ESMV, ISMV, ECMCP, GPILS**]. In this paper, we introduce our work on accelerating a black oil simulator on GPUs. We develop a new matrix vector multiplication kernel [**ESMV, ISMV**] for NVIDIA GPUs and other related BLAS 1/2 subroutines. Based on these subroutines, seven Krylov subspace solvers [**TLS, IMSLS, GPILS, ECMCP**] are developed. Several commonly used preconditioners, such as polynomial, block ILU(k), ILU(k), block ILUT, ILUT [**TLS, IMSLS**] and domain decomposition preconditioners [**RAS**], are also developed. Our solvers and preconditioners are applied to our in-house black oil simulator. The SPE 10 problem is chosen as a benchmark. The number of grid blocks in SPE 10 is over 1.1 million and the number of all unknowns is over 2.2 millions. Numerical experiments show that we can speed the whole simulation up around six times faster than our pure CPU-based simulator.

The layout is as follows. In §2, our new matrix format and sparse matrix-vector multiplication kernel is proposed first, then linear solvers and preconditioners are introduced. In §3, numerical experiments are employed to test the efficiency of the GPU-based linear solvers and preconditioners.

1991 *Mathematics Subject Classification.* Primary 568W10, 68N30; Secondary 65F10, 65F50.

Key words and phrases. Reservoir simulation, linear solver, GPU.

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.

2. Iterative Linear Solvers and Preconditioners

In this section, we introduce our sparse matrix-vector multiplication kernel first, and then the GPU-based linear solvers and preconditioners.

2.1. Sparse Matrix-Vector Multiplication Kernel. The matrix format used for G-PUs is HEC (hybrid ELL and CSR), which is developed in [**PPRS**] and is demonstrated by Figure 1. From this figure, we can see that an HEC matrix contains two submatrices, an ELL matrix and a CSR matrix. The ELL matrix has two matrices, one for the column indices and the other one for non-zeros. The length of each row in these two matrices is the same. A CSR matrix contains three arrays, the first one for the offset of each row, the second one for the column indices and the last one for non-zeros.

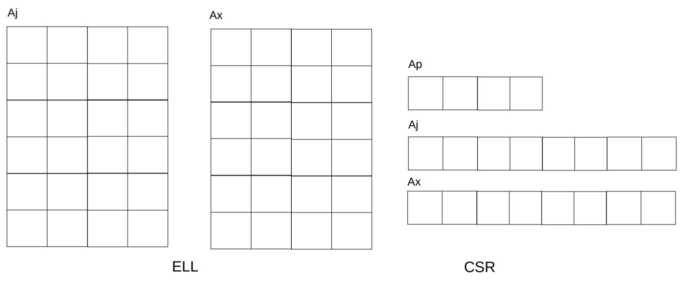

FIGURE 1. HEC matrix format.

The ELL matrix is in column-major order and is aligned in GPUs, which ensure that the data access pattern of global memory for NVIDIA Tesla GPUs is well coalesced [**ESMV, ISMV, NVCUDAPG, CUDABPG**]. In this case the data access speed for the ELL matrix is high. A disadvantage of the ELL format is that even if only one row has too many elements, the length of all rows must be the same. Hence it is a waste of memory. Then a CSR matrix is applied to overcome this problem.

For a HEC format matrix, the corresponding sparse matrix-vector multiplication kernel is described in Algorithm 1. This is a two-step algorithm. In the first step, the ELL part is calculated, where each CUDA thread [**NVCUDAPG, CUDABPG, ESMV, ISMV**] is responsible for one row. Then the CSR part is calculated. Other BLAS 2 subroutines are developed similarly. One BLAS 1 subroutine is described in Algorithm 2. For this algorithm, each CUDA thread calculates only one element. Other BLAS 1 subroutines are similar.

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

```
1: for i = 1: n do                      ▷ ELL, Use one GPU kernel to deal with this loop
2:     the ith thread calculates the ith row of ELL matrix;          ▷ Use one thread
3: end for
4:
5: for i = 1: n do                      ▷ CSR, Use one GPU kernel to deal with this loop
6:     the ith thread calculates the ith row of CSR matrix;          ▷ Use one thread
7: end for
```

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

```
1: for i = 1: n do                          ▷ Use one GPU kernel to deal with this loop
2:     $y[i] = \alpha x[i] + \beta y[i]$;                        ▷ Use one thread
3: end for
```

2.2. Iterative Linear Solvers. We consider the following linear system:

$$Ax = b, \tag{2.1}$$

where A is a nonsingular $n \times n$ matrix, b is the right-hand side and x is the solution to be solved for. Several Krylov subspace linear solvers are listed in [**TLS, IMSLS**]. From the descriptions of these solvers, we can see that these solvers share the following common operations:

$$y = \alpha Ax + \beta y, \quad \alpha, \beta \in R, \tag{2.2}$$

$$z = \alpha Ax + \beta y, \quad \alpha, \beta \in R, \tag{2.3}$$

$$y = \alpha x + \beta y, \quad \alpha, \beta \in R, \tag{2.4}$$

$$z = \alpha x + \beta y, \quad \alpha, \beta \in R, \tag{2.5}$$

$$\alpha = \langle x, y \rangle, \tag{2.6}$$

where A is a matrix, x, y and z are vectors, α and β are real numbers, and $\langle \cdot, \cdot \rangle$ is the scalar product.

These subroutines are simple variants of Algorithm 1 and Algorithm 2. With these BLAS 1/2 operations, the linear solvers can be developed in a straightforward manner. Seven GPU-based Krylov subspace solvers are developed, including GMRES, CG, BICGSTAB, GCR, CGS, ORTHOMIN and ORTHODIR [**IMSLS, TLS**]. The CPU-based versions are also developed.

2.3. Preconditioners. In practice, an equivalent linear system of equations (2.1) is solved:

$$M^{-1}Ax = M^{-1}b, \tag{2.7}$$

where M is called a preconditioner or left-preconditioner. When choosing preconditioner M, a general principle is that M is an approximation of A and in this case, it means that the product of M^{-1} and A approximates the unit matrix I. The condition number of $M^{-1}A$ is smaller than that of A and the linear system (2.7) is much easier to solve compared to the original equation (2.1). Meanwhile, M should be easy to construct and be easy to solve.

When the spectrum of $N = I - A$ is less than 1, we have the Neumann expansion [**TLS**]

$$A^{-1} = I + N + N^2 + N^3 + N^4 + \cdots. \tag{2.8}$$

For any positive integer s, a Neumann polynomial preconditioner is defined as follows:

$$M^{-1} = I + N + N^2 + N^3 + \cdots + N^s. \tag{2.9}$$

When we solve the preconditioned system, only the matrix-vector multiplication is involved.

A simple idea of constructing a preconditioner is to apply LU factorization. However, for a given sparse matrix A, the accurate L and U are usually much denser than the lower and upper parts of A, respectively. Alternatively, incomplete-LU is applied. The ILU factorization computes a sparse lower triangular matrix L and a sparse upper triangular matrix U for a given matrix A. If the non-zero pattern of L and U is the same as that of the lower and upper parts of A, respectively, we obtain the so-called ILU(0) preconditioner and higher order ILU(k) is obtained similarly [**IMSLS**]. Another method is ILUT, which drops entries based on the numerical values of the fill-in elements [**IMSLS, GPILS**], where L and U are controlled by the drop tolerance and the maximal number of fill-ins in each row.

The solution procedure for L and U is sequential. In this paper, block ILU(0) and block ILUT are implemented. If the number of blocks increases, both preconditioners have better parallel performance. The matrix A is partitioned by METIS [**METIS**]. The lower and upper triangular problems are solved by a modified level schedule method [**PPRS**].

Cai et al. developed a restricted additive Schwarz preconditioner (RAS) for solving general sparse matrices [**RAS**]. The basic idea is to partition the original problem to some smaller problems and then to solve these smaller problems simultaneously. In this paper, the matrix is also partitioned by METIS [**METIS**]. The submatrices are extended according to the topology of the original matrix. Each smaller problem is solved by ILU(0) or ILUT.

2.4. Package Structure. Figure 2 is the basic structure of our linear solver package. This package has a multi-level structure. The bottom is the infrastructure, where memory management, communication, input, output, and preprocessing modules are developed. These modules serve the whole package. The middle level includes the matrix and vector operations. The top level includes our solvers and preconditioners. These solvers and preconditioners are designed in such a way that each solver or preconditioner is independent of each other. In this case, this package is friendly to the user, who can choose the proper solver and preconditioner depending on the individual application, and if one solver or preconditioner has bugs, these bugs do not affect other solvers or preconditioners.

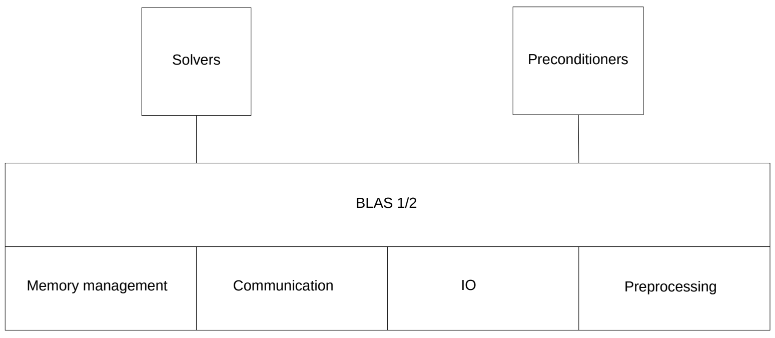

FIGURE 2. Structure of our package.

3. Numerical Results

In this section, numerical experiments are performed on our workstation with Intel Xeon X5570 CPUs and NVIDIA Tesla C2050/C2070 GPUs. The operating system is

Fedora 13 X86_64 with CUDA Toolkit 4.0 and GCC 4.4. All CPU codes are compiled with *-O3* option. The type of float point number is double.

EXAMPLE 3.1. In this example, the matrix is from SPE 10 [**CMFPM**]. The dimension of this matrix is 2,188,851 and the number of non-zeros is 29,915,573. Three solvers are tested without using any preconditioner, and the number of iteration is fixed at 20. Performance data is collected in Table 1.

TABLE 1. Performance of solvers without preconditioner

Solver	CPU (s)	GPU (s)	Speedup
BICGSTAB	3.27	0.31	9.95
ORTHOMIN(20)	5.95	0.52	10.61
ORTHOMIN(40)	5.71	0.53	9.92
GMRES(20)	60.39	5.61	10.72
GMRES(40)	178.08	17.01	10.45
GMRES(60)	361.34	34.32	10.52

This example is designed to test the framework of our package. From Table 1, we can see that when no preconditioner is applied, the average speedup for each solver is around 10.4. We have a maximal speedup of 10.72 when GMRES(40) solver is employed. The table also indicates that the BLAS 1/2 subroutines are efficient, and the whole framework of our package works well.

EXAMPLE 3.2. The matrix used in this example is the same as that in Example 3.1. Here the Neumann polynomial preconditioner is applied, and the order, s, of the polynomial preconditioner is 8. The number of iterations is also 20. Performance data is collected in Table 2.

TABLE 2. Performance of solvers with Neumann polynomial preconditioner

Solver	CPU (s)	GPU (s)	Speedup
BICGSTAB	20.64	2.06	9.90
ORTHOMIN(40)	22.83	2.33	9.62
GMRES(20)	251.97	24.06	10.46
GMRES(40)	619.94	53.07	11.67

This example is employed to test the performance of the developed sparse matrix-vector multiplication kernel, which is fundamental to a linear solver package. From Table 2, we can conclude that the performance of our sparse matrix-vector multiplication kernel is high, and for this example, a maximal speedup of 11.67 is achieved. The average speedup is around 10.5.

EXAMPLE 3.3. Here only the solver GMRES(20) is employed. The preconditioner is block ILU(0) with a different number of blocks. The matrix used here is the same as that in Example 3.1. The terminating criteria is $2e-2$. Performance data is collected in Table 3.

TABLE 3. Performance of GMRES(20) with block ILU(0)

Blks	CPU (s)	GPU (s)	Speedup	IT
1	122.33	14.99	8.14	21
4	124.33	15.00	8.27	21
8	126.40	15.31	8.23	23
16	180.06	19.03	9.44	29

The combination of GMRES and ILU(0) is the most commonly used method for sequential reservoir simulation. Since the solution of ILU(0) is sequential in nature, it is hard to parallelize. This example is to test the parallel performance of our GPU-based block ILU(0) preconditioner. When the number of blocks is one, then the block ILU(0) is the so-called ILU(0). Though ILU(0) is sequential, we can still speed up this preconditioner around 8.14 times faster than the CPU-based ILU(0). When we increase the number of blocks, the speedup increases. It means that the block ILU(0) has better parallel performance. However, the number of iteration increases, too. For this matrix, we have an average speedup of 8.3. When the number of blocks is 16, a maximal speedup of 9.44 is achieved.

EXAMPLE 3.4. Here the block ILUT is applied. All other settings are the same as those in Example 3.3. Performance data is collected in Table 4.

TABLE 4. Performance of GMRES(20) with block ILUT

Blks	CPU (s)	GPU (s)	Speedup	IT
1	34.19	11.70	2.92	5
4	45.52	10.34	4.39	7
8	45.78	9.57	4.76	7
16	63.12	12.42	5.07	10

The ILUT preconditioner is computed by dropping small elements of lower and upper triangular matrices. The non-zero pattern of L and U is less regular than that of ILU(0), which means that their data dependency is more complicated than that in ILU(0). This is also reflected from Table 4. The speedup of block ILUT is lower compared to that of block ILU(0). An average speedup of 4.2 is achieved. However, comparing the data in Table 3 and Table 4, we find that block ILUT is better than block ILU(0) in terms of total running time and the number of iterations. The block ILUT is also sensitive to the number of blocks. The number of iterations increases when the number of blocks increases.

EXAMPLE 3.5. The RAS preconditioner is tested. The solver is GMRES(20) and the matrix is also the same as above. For the RAS preconditioner, the smaller problems are solved by ILU(0) and ILUT here. Data is collected in Tables 5 and 6.

From Tables 5 and 6, we find that ILU(0) has better speedup than ILUT. But in terms of the solution time and the number of iterations, ILUT is still better. Since the subdomain is enlarged, the data from Tables 5 and 6 shows that the number of iterations does not change largely when we increase the number of blocks. It means that the RAS preconditioner is not as sensitive as block ILU(0) and block ILUT. In addition, we can increase the number of blocks to have better performance. For ILU(0), we have an average speedup of 8, and meanwhile, we have an average speedup of 4.5 for ILUT.

TABLE 5. Performance of RAS using ILU(0)

Blks	overlap	CPU (s)	GPU (s)	Speedup	IT
4	1	101.64	15.25	6.65	21
8	1	134.96	15.18	8.87	21
16	1	142.18	16.14	8.78	22

TABLE 6. Performance of RAS using ILUT

Blks	overlap	CPU (s)	GPU (s)	Speedup	IT
4	1	36.28	8.56	4.22	5
8	1	36.88	8.11	4.53	5
16	1	45.21	9.13	4.93	5

EXAMPLE 3.6. The SPE 10 problem is tested. SPE 10 is a standard benchmark for the black oil simulator [**CMFPM**]. The problem is highly heterogenous and it has been designed to be difficult to solve. The grid size for SPE 10 is 60x220x85. The number of unknowns is 2,188,851 and the number of non-zeros is 29,915,573. The time period is 100 days. The solver is GMRES(20). Performance data is collected in Table 7.

TABLE 7. Performance of the SPE10

Preconditioner	Blks	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
RAS + ILU(0)	8	47855.24	8451.55	5.66
RAS + ILU(0)	16	49315.97	8812.98	6.00
RAS + ILUT	8	18553.33	7730.54	2.40
RAS + ILUT	16	19541.72	7419.27	2.63

From Table 7, we can see that when the block ILU(0) and RAS with ILU(0) are applied, the average speedup is around 6. This means that we can speed up the black oil simulator 6 times faster. When the block ILUT and RAS with ILUT are applied, the average speedup is lower, which is only about 2.5. We can still speed up the simulator 2.5 times faster than the pure CPU simulator. The parallel performance of these preconditioners is similar, but for sequential performance, the ILUT-related preconditioners are much better than ILU(0)-related preconditioners.

4. Conclusion

We have presented our work on accelerating a black oil simulator using GPU-based linear solvers and preconditioners. The numerical experiments show that these solvers and preconditioners are efficient. The simulator can be sped up around 6 times faster with these solvers and preconditioners.

References

[CMFPM] 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.

[IUG] 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.

[TLS] 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.

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

[PPRS] 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.

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

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

[ECMCP] 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

[GPILS] 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.

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

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

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

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

Current address: Department of Chemical and Petroleum Engineering, University of Calgary, Alberta, Canada

E-mail address: `zhachen@ucalgary.ca`

Current address: Department of Chemical and Petroleum Engineering, University of Calgary, Alberta, Canada

E-mail address: `hui.j.liu@ucalgary.ca`

Current address: Department of Chemical and Petroleum Engineering, University of Calgary, Alberta, Canada

E-mail address: `yusong0926@gmail.com`

Current address: Department of Chemical and Petroleum Engineering, University of Calgary, Alberta, Canada

E-mail address: `bhsieh@ucalgary.ca`

Current address: Department of Chemical and Petroleum Engineering, University of Calgary, Alberta, Canada

E-mail address: `jedyfun@gmail.com`