Scalable and Improved SuperLU on GPU for Heterogeneous Systems

M. Serdar Celebi[a,c], Ahmet Duran[b,a]*, Mehmet Tuncel[a,c], Bora Akaydın[a,c]

[a]*Istanbul Technical University, National Center for High Performance Computing of Turkey (UHeM), Istanbul 34469, Turkey*
[b]*Istanbul Technical University,Department of Mathematics, Istanbul 34469, Turkey*
[c]*Istanbul Technical University, Informatics Institute, Istanbul 34469, Turkey*

July 13, 2012

Abstract

We consider a fast, robust and scalable solver using graphic processing units (GPU) as accelerators for a sparse linear system $AX=B$. In this project, a new parallel hybrid direct solver is designed and implemented on GPU for large sparse linear systems. In particular, we work on GPU programming using directive based Open ACC in order to obtain a scalable and improved SuperLU on CPU+GPU heterogeneous systems.

Project ID: FP7-INFRASTRUCTURES-2011-2, PRACE-2IP —PRACE - Second Implementation Phase Project

1. Introduction

It is important to use graphic processing units (GPU) as accelerators when we consider a fast, robust and scalable solver for a sparse linear system $AX=B$ in many science and engineering applications. In this project, a new parallel hybrid direct solver is designed and implemented on GPU for large sparse linear systems. In particular, we work on GPU programming using directive based Open ACC in order to obtain a scalable and improved SuperLU on CPU+GPU heterogeneous systems.

Section 2 describes how to implement SuperLU on CPU+GPU heterogeneous systems. Later, the scalability of SuperLU_MT is discussed for randomly populated matrices in terms of speed up for a given matrix and dependency on sparsity level. Section 3 concludes this work.

2. Methods and results

As a first step, we examine the effectiveness of the SuperLU_DIST 3.0 for distributed memory and SuperLU_MT 2.0 for shared memory parallel machines among several sparse direct solvers (see [1, 2, 3, 4, 5, 6]) on CPU and (see [7] for small matrices) on CPU-GPU. SuperLU_MT (see [8]) has three major steps including sparsity ordering, factorization that arranges partial pivoting, symbolic factorization and numerical factorization steps to perform in an alternating fashion, and triangular solution. While SuperLU_DIST uses BLAS 3 for factorization, SuperLU_MT has only BLAS 2.5 with multiple matrix vector multiplication. SuperLU_DIST (see [10]) uses static pivoting [11] instead of partial pivoting because the implementation of numerical pivoting is complicated on distributed memory architecture. It is advantageous that symbolic and numerical factorization steps can be separated

* Corresponding author. *E-mail address*: aduran@itu.edu.tr.

due to the static pivoting. Therefore, SuperLU_DIST outperforms SuperLU_MT (see [9] and [12]) for many sparse matrices.

Second, SuperLU is a complex algorithm and it is important to choose the right combination for better intra-node communications and inter-node communications within CPU+GPU heterogeneous systems, given current technology limitations and developments. While SuperLU_MT is a good starting reference for intra-node communications, SuperLU_DIST is more appealing for GPU clusters having inter-node communications using infiniband (IB) network among its several advantages. In this project, we take advantage of SuperLU_DIST such as the usage of the extract parallelism reducing communication by avoiding and defining dependencies of data in addition to the usage of static pivoting and BLAS 3 for factorization. Moreover, we benefit from multicores approach inside node analogous to SuperLU_MT. The first goal is to complete intra-node multi-GPU programming. Next step would be inter-node multi-GPU programming.

Many multiscale modelling applications in science and engineering result in more general matrices in order to capture more details in the system. Therefore we consider a portfolio of test matrices containing randomly populated sparse matrices. We generate 30 different randomly populated matrices RAND_30K_3, ..., RAND_30K_30 for each. Each experiment is done at least four times. We describe the matrices in Table 1.

2.1. Description of matrices

Table 1. Description of randomly populated matrices

Randomly populated matrices Name	Order	NNZ	NNZ/N	Condition number	Origin
RAND_30K_3	30000	90000	3	1.20×10^6	UHeM
RAND_30K_5	30000	150000	5	4.22×10^6	UHeM
RAND_30K_7	30000	210000	7	1.76×10^6	UHeM
RAND_30K_9	30000	270000	9	2.51×10^6	UHeM
RAND_30K_11	30000	330000	11	8.82×10^5	UHeM
RAND_30K_15	30000	450000	15	3.20×10^7	UHeM
RAND_30K_20	30000	600000	20	9.51×10^6	UHeM
RAND_30K_25	30000	750000	25	5.10×10^6	UHeM
RAND_30K_30	30000	900000	30	1.13×10^6	UHeM
RAND_40K_3	40000	120000	3	3.90×10^6	UHeM

2.2. Scalability of SuperLU_MT

The code of SuperLU_MT has been tested up to 64 threads for a list of patterned sparse matrices on HP Integrity Superdome SD32B [see 13], a computing server with shared memory architecture at UHeM. A performance scalability between 4 (for an unsymmetric matrix with low sparsity) and 32 (for a small almost symmetric matrix with low sparsity) is achieved depending on the number of nonzeros per row, total number of nonzeros and structural symmetry (see [12]). These results with different machine are in the line of Demmel et al. [8].

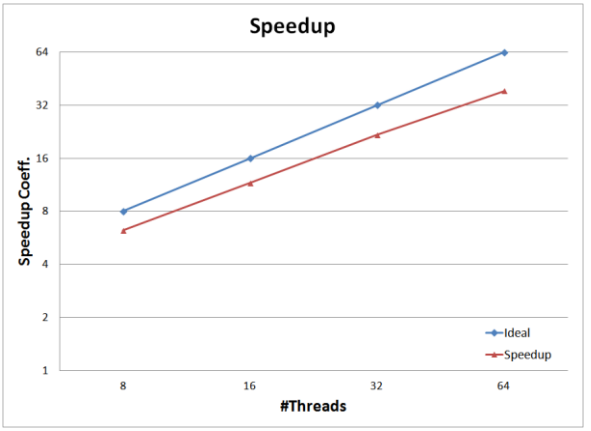

Fig. 1. Speed up for matrix RAND_40K_3.

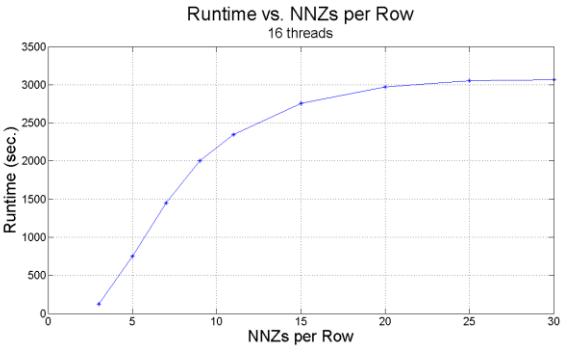

Fig. 2. Average wall clock time as a function of various sparsity levels for randomly populated sparse matrices.

Table 1. Wall clock time using SuperLU_MT for randomly populated sparse matrices as sparsity level decreases with 16 cores

NNZ per row	3	5	7	9	11	15	20	25	30
Wall clock time	124.46	750.10	1448.22	2003.09	2346.67	2759.27	2969.65	3054.11	3062.57

In this project, the code of SuperLU_MT has been tested up to 64 threads for randomly populated sparse matrices on HP Integrity Superdome SD32B (see [13]) computing server. Almost linear speedup is achieved (see for example Figure 1 for RAND_40K_3). Moreover, we tested the scalability of SuperLU_MT depending on the sparsity level in terms of NNZ per row. Figure 2 and Table 1 show that average wall clock time increases slowly as sparsity levels decreases and almost levels off after 15 number of nonzeros per row for randomly populated sparse matrices of order 30000.

3. Conclusions

In sum, after obtaining a robust version of scalable SuperLU we design a new hybrid algorithm for CPU+GPU heterogeneous systems by taking SuperLU_DIST as a starting reference in this project. We are implementing directive based parallelization approach using OpenACC for CPU+GPU heterogeneous systems. Later, we will be testing its performance on the heterogeneous systems for various large sparse matrices.

Acknowledgements

This work was financially supported by the PRACE project funded in part by the EUs 7th Framework Programme (FP7/2011-2013) under grant agreement no. 283493. Computing resources used in this work were provided by the National Center for High Performance Computing of Turkey (UHeM) (http://www.uybhm.itu.edu.tr/eng) under grant number 1001682012.

References

1. X. S. Li, J. W. Demmel, J. R. Gilbert, L. Grigori, M. Shao, and I. Yamazaki, SuperLU Users' Guide, Tech. Report UCB, Computer Science Division, University of California, Berkeley, CA, 1999, update: 2011.

2. P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 15–41.

3. O. Schenk and K. Gartner, Solving unsymmetric sparse systems of linear equations with PARDISO, Future Generation Computer Systems, 20 (2004), pp. 475-487.

4. O. Schenk and K. Gartner, *On fast factorization pivoting methods for sparse symmetric indefinite systems*, Electronic Transactions on Numerical Analysis, 23 (2006), pp. 158 – 179.

5. A. Duran and B.D. Saunders, Gen_SuperLU package (version 1.0, August 2002), a part of LinBox package, containing a set of subroutines to solve a sparse linear system A*X=B over any field.

6. A. Duran, B. D. Saunders and Z. Wan, Hybrid algorithms for rank of sparse matrices, *Proceedings of the SIAM International Conference on Applied Linear Algebra (SIAM-LA)*, Williamsburg, VA, July 15-19, 2003.

7. L. Li, L. Li and Y. Guangwen, A highly efficient GPU-CPU hybrid parallel implementation of sparse LU factorization, Chinese J. of Electronics, 21:7-12, 2012.

8. J.W. Demmel, J.R. Gilbert, and X.S. Li. An asynchronous parallel supernodal algorithm for sparse gaussian elimination. SIAM J. Matrix Analysis and Applications, 20(4):915-952, 1999.

9. X.S. Li. Evaluation of sparse LU factorization and triangular solution on multicore platforms. VECPAR 2008, Springer.

10. X. S. Li and J. W. Demmel, *Superlu-dist: A scalable distributed-memory sparse direct solver for unsymmetric linear systems*, ACM Trans. Math. Softw., 29 (2003), pp. 110–140.

11. L. Grigori, J.W. Demmel, and X.S. Li. Parallel symbolic factorization for sparse LU with static pivoting. SIAM J. Scientific Computing, 29(3):1289-1314, 2007.

12. M.S. Celebi, A. Duran, M.Tuncel, B. Akaydin, Scalability of SuperLU solvers for large scale complex reservoir simulations, SPE and SIAM Conference on Mathematical Methods in Fluid Dynamics and Simulation of Giant Oil and Gas Reservoirs, Istanbul, Turkey, September 3-5, 2012.

13. nPartition Administrator's Guide, HP part number: 5991-1247B, 1st Edition, February 2007, Hewlett-Packard Development Company.