Available online at www.prace-ri.eu

Partnership for Advanced Computing in Europe

Performance Analysis of BLAS Libraries in SuperLU_DIST for SuperLU_MCDT (Multi Core Distributed) Development

M. Serdar Celebi[a,c], Ahmet Duran[b,a]*, Mehmet Tuncel[a,c], Bora Akaydın[a,c], Figen Oztoprak[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 11, 2013

Abstract

SuperLU_DIST is a distributed memory parallel solver for sparse linear systems. The solver makes several calls to BLAS library routines in its numerical factorization phase. The performance of the BLAS library can significantly affect the overall performance of the solver as the required BLAS operations are typically computationally dense. In this regard, we examine how the overall performance of the SuperLU_DIST solver can be improved by employing optimized BLAS libraries. In particular, we try using Intel Math Kernel Library (MKL) and Parallel Linear Algebra Subroutines for Multicore Architecture (PLASMA) libraries. Using MKL can provide an approximate performance improvement of 50 %, and using PLASMA can improve the performance by around 10 % for the best tile size. Based on our findings, we have improved SuperLU_MCDT solver.

1. Introduction

SuperLU_DIST is an MPI (Message Passing Interface) based parallel LU factorization framework and sparse linear system solver for AX=B. The solver uses BLAS library routines (see [7]), i.e. the third level routines for matrix-matrix multiplication, during numerical factorization [1, 2]. A careful performance profiling reveals that these BLAS calls are among the most time consuming parts in a SuperLU_DIST run. Therefore, we believe that using an optimized BLAS library can improve the overall performance of the solver significantly.

In this work, two different BLAS implementations are examined in order to accelerate the SuperLU_DIST solver: Intel MKL (see [8]) and PLASMA. Intel MKL (Math Kernel Library) is a highly optimized version of BLAS for Intel CPUs that does out of order computing and pipelining, whereas PLASMA (Parallel Linear Algebra Subroutines for Multicore Architecture) is a BLAS library developed for multicore systems [4]. Using a multithreaded library within SuperLU_DIST results in a hybrid implementation which can utilize the multiple cores in each MPI compute node while executing the BLAS operations. Based on our results, we have developed SuperLU_MCDT solver, in addition to our other improvements (see Duran et al. [5] and Celebi et al. [6]). The SuperLU_MCDT solver has new properties of parallel data input and distribution of related parts of matrices;

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

integration of OpenMP directives for threads based hybrid programming in the code; exception handling of challenging matrices and using tuned parameters; and linking of PLASMA library with optimized parameters such as tile size. The developing process of the SuperLU_MCDT solver is in progress.

In the next section, we further introduce the BLAS libraries Intel MKL and PLASMA. The performance results for SuperLU_DIST with these libraries are discussed in Section 3. We used SuperLU_DIST Version 3.2 in all our analysis and tests; therefore in the rest of the paper 'SuperLU_DIST' refers to SuperLU_DIST Version 3.2.

2. BLAS libraries

In this study, the SuperLU_DIST library is compiled and linked with three different versions of BLAS. The first one is the legacy BLAS library which can be obtained from the Netlib repository. We compiled the legacy BLAS library with the default options using the Intel Fortran compiler and the SuperLU_DIST source code using the Intel MPI C compiler.

Intel MKL library is included in the Intel Composer toolkit. We used BLAS95 and LAPACK95 static MKL libraries. Since the names of the routines included in BLAS and MKL libraries do match, no change is needed in the source code of SuperLU_DIST. Only the linking parameters are changed.

The third library is PLASMA, which is developed by the University of Tennessee. It contains multithreaded equivalents of the BLAS routines in either synchronous or asynchronous modes, as well as the Quark (Queuing And Runtime for Kernels) scheduler [4]. PLASMA uses a different memory layout than LAPACK. It divides matrices into small contiguous tiles such that these small tiles can fit into the L1 or L2 cache memories of the CPU. The primary goal of PLASMA is to utilize the cache memory efficiently by using small matrix tiles [3].

PLASMA depends on core BLAS routines and the Quark library for scheduling [3, 4]. These core BLAS routines are contained in LAPACK and CBLAS [4]. The detailed dependency graph is illustrated in Figure 1. All libraries are built using their default build options except that SuperLU_DIST is built using –mt_mpi option in order to allow hybrid processes. Since the structure of the SuperLU_DIST source code is not changed for MPI+PLASMA hybrid execution, all the PLASMA tests are done with a single PLASMA thread, i.e. PLASMA library is initialized every time for a single thread only.

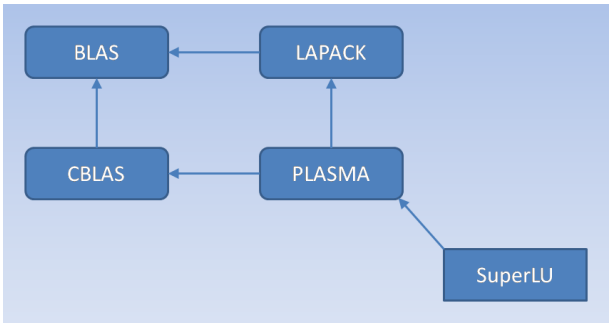

Fig. 1. Dependency graph of SuperLU_DIST using PLASMA

In order to use the PLASMA library efficiently, we first need to determine the optimal tile size. The library comes with the default setting of 128 tiles. We tested different tile sizes varying between 8 and 1024. Figure 2 shows the change in the runtime depending on the tile size. The figure indicates that the tile sizes between 8 and 48 are certainly not optimal. The results for tile sizes 8 and 12 are omitted in Figure 2 because the runtime values for these sizes are too large, and deteriorate the readability of the figure.

The minimal runtime is obtained for the tile size of 64 and the runtime increases slightly as tile size gets larger.

Therefore, two different tile sizes are selected for further experiments: 64 and 128.

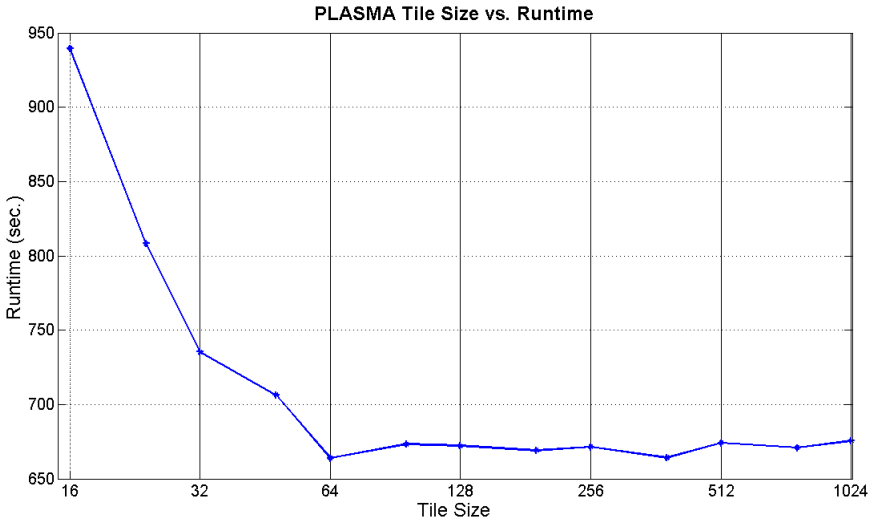

Fig. 2. The runtime of SuperLU_DIST with PLASMA for varying PLASMA tile size.

3. Results

The numerical experiments are conducted for three different sparse matrices. All matrices have different characteristics. The first matrix is an unsymmetrical 900K x 900K matrix with 13 362 067 nonzeros, and it has a relatively good condition number. The second matrix is an 2.3M x 2.3M matrix with 13 752 839 nonzeros. It is also unsymmetrical, but worse conditioned as compared to the first matrix. The third problem is a geomechanical structural problem from Florida Matrix Market Collection and it is a symmetrical 923K x 923K matrix with 40M nonzeros. Detailed descriptions of the problems are provided in Table 1.

Table 1. Detailed description of the problems

Matrix Properties	First Problem M_uhem1	Second Problem M_uhem2	Third Problem Emilia_923
Size specifications			
Order	900 000	2 302 320	923 136
Number of nonzeros (NNZ)	13 362 067	13 752 839	40 373 538
NNZ per row	14.85	5.97	43.74
Pattern specifications			
NNZ on diagonal	900 000	2 302 320	
NNZ below diagonal	7 101 903	6 197 947	
NNZ above diagonal	5 360 164	5 252 572	
Max/min NNZ per row	21 / 2	21 / 2	
Max/min NNZ per column	19 / 5	21 / 1	

Table 1 (continued). Detailed description of the problems

Matrix Properties	First Problem M_uhem1	Second Problem M_uhem2	Third Problem Emilia_923
Symmetry			
Pattern Symmetry	60.28%	59.79%	100%
Numerical Symmetry	32.12%	41.9%	100%
Spectral specifications			N/A
Largest Magnitude Eigenvalues	600.75	52 492.33	
	560.09	50 029.09	
	502.22	45 848.17	
	427.33	45 605.1	
Eigenvalues with smallest real part	0.003644	0.003398	
	0.006038+0.001321i	0.092618	
	0.006038-0.001321i		
Eigenvalues closest to zero	0.003639	0.00412	
	0.006069+0.001357i	0.037204	
	0.006069-0.001357i	0.054307	

All tests are repeated at least eight times, and the average runtime is recorded. The specifications of the computational system, which is provided by ITU National High Performance Computing Center (UHeM), are in Table 2.

Table 2 Description of used hardware metrics of CPU.

System name	Karadeniz
Processor	Intel Xeon 5550 [Quad Core] (Nehalem) @ 2,67 GHz
L1 cache	4 X 32 KB
L2 cache	4 X 256 KB
L3 cache	8 MB
Number of compute nodes	64
Number of compute cores	512
Memory architecture	Distributed
Per core memory amount	3 GB
Disk Space per node	292 GB
High performance network	InfiniBand 20 Gbps

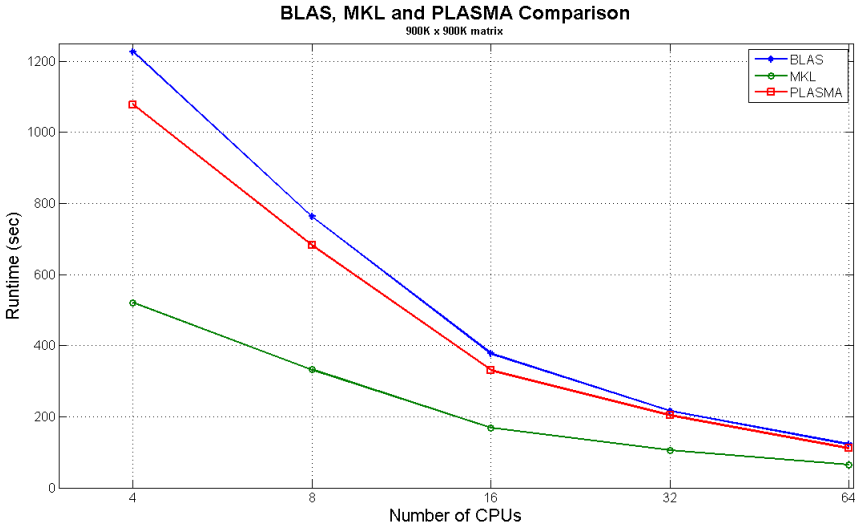

Fig. 3. Comparison of the runtimes for SuperLU_DIST using different BLAS implementations for the first matrix, M_uhem1.

Figure 3 shows the runtimes of SuperLU_DIST with BLAS, MKL and PLASMA libraries for the first problem (M_uhem1). Only the tile size of 64 is considered on the plot because the results for the tile sizes 64 and 128 are almost the same. As the figure clearly reveals, PLASMA provides approximately 10 % performance improvement over the legacy BLAS for this problem, whereas the runtime obtained by employing MKL is almost half of the runtime with legacy BLAS. We should note that MKL is strictly optimized for Intel CPUs, so this graph can look somehow different on other platforms with different processors. SuperLU_DIST loses its stability for this problem for more than 64 cores, but this situation is not relevant to the BLAS implementation used.

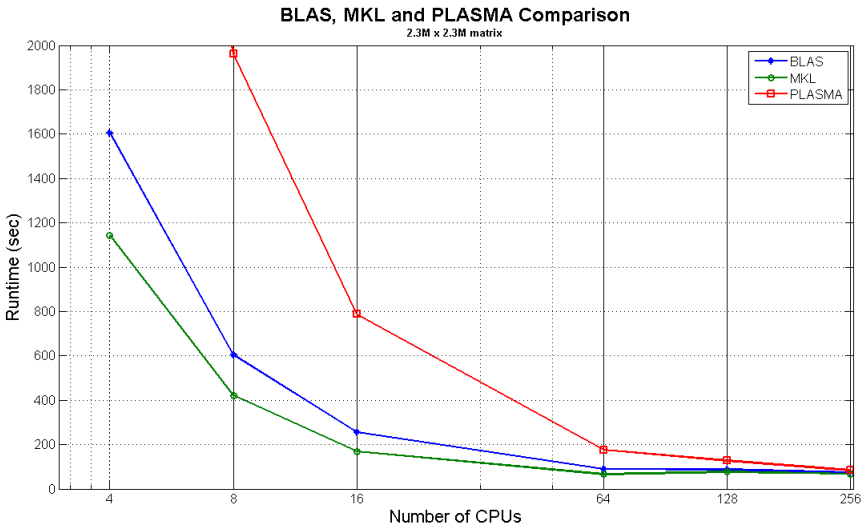

Fig. 4. Comparison of the runtimes for SuperLU_DIST using different BLAS implementations for the second matrix, M_uhem2

The set of runtimes for the second problem (M_uhem2) is given in Figure 4. Again, only the tile size of 64 is considered on the plot. We have mentioned above that this problem has different characteristics as compared to the first one, and we observe from Figure 4 that this difference in characteristics results in a big difference in the performance results. The figure shows that PLASMA performs much worse than the legacy BLAS for this problem. Intel MKL still provides a superior performance even if the improvement here is not as large as the improvement achieved for the first problem. SuperLU_DIST successfully completed the factorization of M_uhem2 for different number of cores varying between 4 and 256 without any error except for the run on 32 CPUs. This time, we excluded the runtime for 4 CPUs in Figure 4 to improve its readability (the runtime of SuperLU_DIST with PLASMA on four CPUs is 5313.77 seconds).

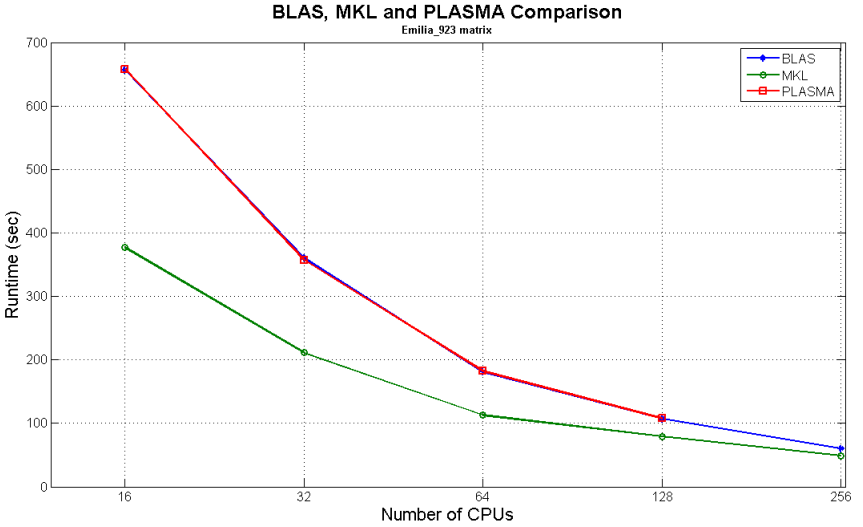

Fig. 5. Comparison of the runtimes of SuperLU_DIST using different BLAS implementations for the Emilia_923 matrix

Figure 5 shows the runtime result of SuperLU_DIST using the same libraries. For the PLASMA line, only the tile size of 64 is considered. Because the number of nonzeros is very large for this problem, the experiments cannot be done using 4 or 8 processors. Emilia_923 is a denser matrix compared with the two previous examples but it is less challenging than M_uhem2. Therefore, PLASMA gives almost the same performance as BLAS, but MKL gives again better performance than the two other libraries do. The performance differences are 70 % for the matrix Emilia_923 with 16 processors and 23 % with 256 processors. It can be concluded that performance of PLASMA is strictly dependent on the characteristics of the matrix. The performance difference between MKL and BLAS also depends on the matrix characteristics. While MKL is considerably better for well-conditioned problem, the performance difference between BLAS and MKL drops for the ill-conditioned problem.

Conclusion

We observed that the Intel MKL library can provide a very good performance for SuperLU_DIST as compared to the legacy BLAS. However, the results for PLASMA are not as expected. We believe that SuperLU_DIST can benefit from the PLASMA library by making necessary modifications on the SuperLU_DIST source code and on the communication scheme of the PLASMA library. Specifically, PLASMA threads can work more efficiently if the

processes on different compute nodes are the masters for PLASMA threads. Based on our findings, we have been developing the SuperLU_MCDT solver.

Acknowledgements

This work was financially supported by the PRACE-2IP project funded in part by the EUs 7th Framework Programme (FP7/2011-2013) under grant agreement no. RI-283493. Computing resources used in this work were provided by the National Center for High Performance Computing of Turkey (UHeM) (see [9]) 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. X.S. Li. Evaluation of sparse LU factorization and triangular solution on multicore platforms. VECPAR 2008, Springer.
3. J. Kurzak, J. Dongarra. PLASMA Usage, Presentation, Supercomputing 2012, Salt Lake City, UT, 2012.
4. J. Dongarra, J. Kurzak, et. al. PLASMA Users' Guide, Tech. Report, University of Tennessee, TN, version 2.3, update: 04 Sept 2010
5. A. Duran, M.S. Celebi, M. Tuncel and B. Akaydın, Design and implementation of new hybrid algorithm and solver on CPU for large sparse linear systems, PRACE-2IP white paper, Libraries, WP 43, July 13, 2012, http://www.prace-ri.eu/IMG/pdf/wp43-newhybridalgorithmfo_lsls.pdf
6. M.S. Celebi, A. Duran, M. Tuncel and B. Akaydın, Scalable and improved SuperLU on GPU for heterogeneous systems, PRACE-2IP white paper, Libraries, WP 44, July 13, 2012, http://www.prace-ri.eu/IMG/pdf/scalablesuperluongpu.pdf
7. http://www.netlib.org/blas
8. http://software.intel.com/en-us/intel-mkl
9. http://www.uybhm.itu.edu.tr