Available online at www.prace-ri.eu

**Partnership for Advanced Computing in Europe**

# Analysis of SuperLU Solvers on Intel® MIC Architecture

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

25 December 2013

**Abstract**

Intel Xeon Phi is a coprocessor with sixty-one cores in a single chip. The chip has a more powerful FPU that contains 512-bit SIMD registers. Intel Xeon Phi chip can benefit from the algorithms that operate with the large vectors. In this work, sequential, multithreaded and distributed versions of SuperLU solvers are tested on the Intel Xeon Phi using offload programming model and they work well. There are several offload programming alternatives depending on where to place pragma directives. We find that the sequential SuperLU benefited up to 45% performance improvement from the offload programming depending on the sparse matrix type and the size of transferred and processed data. On the other hand, the partitioning method of SuperLU_DIST and SuperLU_MT generates very small sized submatrices. Therefore, we observe that the matrix partitioning method and several other tradeoffs influence their performance via the Xeon Phi architecture.

## 1. Introduction

In many science and engineering applications, problems may result in solving a sparse linear system AX=B. It is important to test the status and potential improvements of state-of-the-art solvers on new technologies. Intel Xeon Phi (referred as MIC in the rest of the paper) is a new kind of processor on a PCI-Express card that cannot operate alone without a CPU. It can be more effective than CPU for big vector operations. MIC has some similarities and also differences with GPGPU's [1]. The MIC has 61 cores operating at 1.053 GHz but 60 of them are accessible to the programmer. The FPU on MIC has 512-bit vector registers and the cache memory on the MIC chip is fully coherent. Its size is 30 or 60 MB on chip (512 KB or 1MB per core) depending on the model. The coprocessor has 8 GB GDDR5 memory on chip. 8 GB can be insufficient for some problems.

Despite MIC coprocessor's speed and throughput on floating point operations, it uses PCI-E bus. This can be a bottleneck for overall performance because excessive numbers of very small floating point operations cannot be

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

handled efficiently in MIC because of its bus access and slow clock rate compared to CPU. MICs can efficiently operate with huge vectors and/or matrices that can scale also well in CPU threads.

SuperLU library [2] has two different parallel versions: Message Passing Interface (MPI) based SuperLU_DIST and thread based SuperLU_MT. SuperLU_MT has some limitations for scaling-up over 32 processors [3]. In this work, offload programming model is applied to sequential SuperLU, SuperLU_MT 2.1 and SuperLU_DIST 3.3. They are tested for some randomly located sparse matrices and patterned matrices. Moreover, native programming model is applied to SuperLU_DIST.

It is challenging to decide where to place offload pragmas and there are many potential places to consider. We examine the places which are among the top time consuming code blocks by using Intel® loop and function profile viewer which is a part of Intel Composer XE Suite. Most of the trial places for offload pragmas are in the factorisation parts, because the factorisation part is the dominant time consuming part. According to our measurements on Hydra cluster in RZG (Rechenzentrum Garching), SuperLU_DIST performs almost 40% efficiency until 16 processors using PAPI library. This indicates that there is potential for improvement.

The remainder of this work is organised as follows: In Section 2, MIC programming models and test environment are presented. In Section 3, test results are discussed. Section 4 concludes this work.

## 2. Programming models and test environment

There may be three different programming models such as native, offload and symmetric models to work with Intel MIC. In native model, the code is compiled only for MIC and runs on MIC only. The offload programming model is similar to the GPGPU kernels. Most of the code is compiled for the CPU but certain parts that are more appropriate to run on MIC are compiled for the MIC coprocessor. In this paper, we focus on native and offload programming models.

The MIC containing nodes of EURORA test cluster (see [5]) is used for all tests. The MIC has 61 physical cores. Every physical core has also four logical cores. In the rest of this paper physical cores will be referred as “core”. Logical cores will be referred explicitly.

## 3. Test results

### *3.1. Description of matrices*

We use two randomly located matrices [3] and eight patterned matrices from University Florida Matrix Market (UFMM) [10].

Table 1. Description of patterned and randomly located matrices

| Matrix Name | Order | NNZ | NNZ/N | Nonzero pattern symmetry | Numeric value symmetry | Condition number | Origin | Kind of problem |
|---|---|---|---|---|---|---|---|---|
| ADD32 | 4960 | 19848 | 4.00 | 100% | 31% | 136.677 | UFMM | Circuit simulation problem |
| CAGE8 | 1015 | 11003 | 10.84 | 100% | 14% | 11.4135 | UFMM | DNA electrophoresis |
| CAGE10 | 11397 | 150645 | 13.22 | 100% | 17% | 11.0175 | UFMM | DNA electrophoresis |
| ECL32 | 51993 | 380415 | 7.32 | 92% | 60% | $9.41 \times 10^{15}$ | UFMM | Semiconductor device |
| MARK3JAC140SC | 64089 | 376395 | 5.87 | 7% | 1% | $5.83 \times 10^{13}$ | UFMM | Economic |

Table 1 (continued). Description of patterned and randomly located matrices

| Matrix Name | Order | NNZ | NNZ/N | Nonzero pattern symmetry | Numeric value symmetry | Condition number | Origin | Kind of problem |
|---|---|---|---|---|---|---|---|---|
| MIXTANK_NEW | 29957 | 1990919 | 66.46 | 100% | 99% | 4.40 x $10^{11}$ | UFMM | Computational fluid dynamics |
| PRE2 | 659033 | 5834044 | 8.85 | 33% | 7% | 3.11 x $10^{23}$ | UFMM | Frequency-domain circuit simulation |
| Rand_10K_3 | 10000 | 29997 | 3 | Asymmetric | Asymmetric | $\approx 7.1068 \cdot 10^5$ | UHeM | |
| Rand_30K_3 | 30000 | 89997 | 3 | Asymmetric | Asymmetric | $\approx 1.2466 \cdot 10^6$ | UHeM | |
| STOMACH | 213360 | 3021648 | 14.16 | 85% | 0% | 8.01 x $10^1$ | UFMM | 3D electro-physical model |

### 3.2. Sequential SuperLU

SuperLU [2] uses a compression approach to store the sparse matrices like CRS (Compressed Row Storage) and CCS (Compressed Column Storage). These formats, which are used for the memory efficiency, do not store the matrices in a single contiguous array. Therefore, this is a tradeoff between exceeding the memory limitation with the data storage strategies and being able to use the vector operations which is major advantage of MIC over CPUs. SuperLU partitions the matrices into the chunks according to the row and column ordering and distribution of the matrix data on the rectangular mesh. The chunk size is not proportional to the matrix size. Big matrices can be partitioned either in small or large chunks but it is not expected to obtain big chunks in small matrices.

Table 2. Benchmark for sequential and offload programming approach

| | Sequential Time (s) | | Offload Programming Time (s) | |
|---|---|---|---|---|
| | Factorisation | Solving | Factorisation | Solving |
| RAND_30K_3 | 144,85 | 0,13 | 142,71 | 0,13 |
| CAGE10 | 21,77 | 0,06 | 21,98 | 0,1 |
| ECL32 | 50,17 | 0,18 | 50,49 | 0,18 |
| MARK3JAC140SC | 43,81 | 0,14 | 44,5 | 0,14 |
| MIXTANK_NEW | 77,1 | 0,19 | 77,73 | 0,2 |
| PRE2 | 723,35 | 1,4 | 497,34 | 0,99 |
| STOMACH | 91,68 | 0,49 | 92,14 | 0,49 |

We apply the offload programming approach for the sequential SuperLU by using 120 MIC threads and MIC affinity is set to ‘balanced’. This affinity preference is slightly better than other settings like ‘compact’ and ‘scatter’. As seen in the Table 2, we obtain up to 45% performance improvement for the matrix *PRE2* because this is an illustrative example to have big chunks of data. The transferred and processed data size is around 60 MB for *PRE2*. However, we couldn't observe significant difference for the other matrices due to the small sizes around 10 MB.

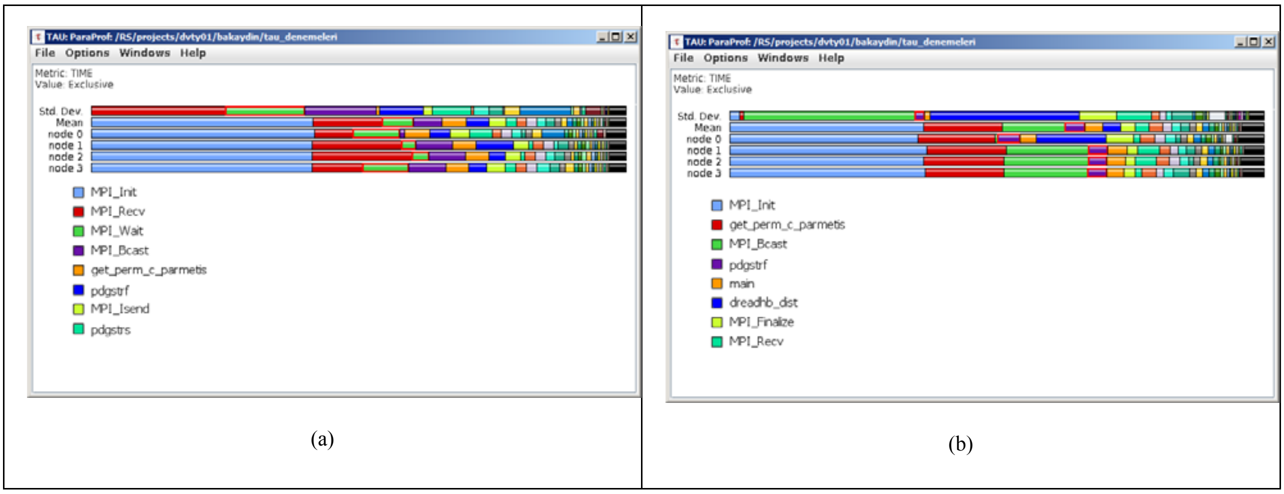
### 3.3. Multithreaded SuperLU (SuperLU_MT)

SuperLU_MT is tested for offload programming model. Matrix-vector operations are very important in solvers like SuperLU_MT. It is preferred to use SuperLU’s own backsolve and matrix-vector multiplication (dmatvec function) code instead of BLAS library [11] because these routines are more convenient to compile and link to the SuperLU_MT code compared with the BLAS. The function 'dmatvec' receives relatively big data as an argument and it is considered that 'dmatvec' function is a good candidate for being offloaded onto the MIC. The matrix and vector data is scattered between MIC cores and the resulting vector is gathered using a wrapper function for offloading. With this change, offloaded SuperLU_MT is tested with different matrices but very low performance is obtained. The performance obtained is 100 to 300 times slower than that of pure CPU run. The reasons for this deceleration will be discussed in next chapter but the most distinctive reason is the consecutive 'dmatvec' calls on MIC. The communication occurs between MIC and RAM numerous times and also too small chunk of data. This communication passes through PCI-E bus which is the bottleneck for MIC coprocessors. A threshold value on the matrix size is defined, so that only the matrices larger than the threshold value are sent to MIC. The low performance issue does not change. This problem occurs due to matrix decomposition strategy of the SuperLU library.

As an example, Rand_10K_3 is a relatively small problem for SuperLU (solved in 2.77 seconds by four threads (SuperLU_MT) and in 14.95 seconds by four processes (SuperLU_DIST)). For this test matrix (for specifications see table 1), 'dmatvec' function is called 234 665 times where the largest input matrix has 32 406 double elements. 214 548 of them are larger than 1500 elements and 195 475 of them are larger than 4000 elements. Mean input matrix size is 6688 elements and this size is very small for MIC offloading. Rand_30K_3 can be considered as an average problem for four processors or threads but it is even small for more processors. Solving rand_30K_3 requires 1 794 512 'dmatvec' calls which causes a huge communication cost between MIC and RAM. Maximal input matrix size is 83 620 elements. Thus, offloading these parts is inefficient.

### 3.4. Distributed SuperLU (SuperLU_DIST)

For SuperLU_DIST, we tried automatic offloading. The Intel compiler detects BLAS calls when linked to MKL [8], and decides to offload or to run in CPU according to the size of data for specific BLAS functions (Xgemm, Xsymm, Xtrmm and Xtrsm [9]). For Intel C compilers the threshold is 2048 x 2048 matrix with double elements and for Fortran compilers threshold is 1280 x 1280 matrix with double elements. The decision of offloading is made in runtime.



(a)

(b)

Fig. 1. Profiling result of SuperLU_DIST in four processors (without MIC) for the matrices ADD32 (a) and CAGE8 (b)

Intel states that MIC coprocessor can benefit from large vector operations [6, 7]. Although SuperLU_DIST works well with large matrices, very small submatrices are obtained by the decomposition of the input matrix into the supernodes. SuperLU_DIST uses Metis or ParMetis for partitioning the input matrix into smaller subblocks and utilizes supernodes. Supernodes are atomic structures. In graph representation of the matrix, supernodes correspond to the simple graphs with no branching [2]. The maximum size of a supernode is determined by a parameter in sp_ienv function. This parameter is an upper bound for the supernode but a lower bound for the supernode size is not provided. Although the maximum size is set to 60 by default (i.e. 60 x 60 matrix); in practice, a 60 x 60 sized supernode was never encountered with our test set. Also this size is even too small for the offloading threshold. Thus, automatic offload approach seems to be not feasible for SuperLU_DIST.

'dgemm' (double general matrix-matrix multiplication) portion of SuperLU_DIST is forced for running on MIC using pragma directives. It was mentioned in previous sections that MIC coprocessor card has 8GB of RAM which can be sometimes limiting factor for running large matrices on MIC. Moreover, we limit the runtime with 30 minutes and the experiment is terminated after 30 minutes because it is not appropriate to run when a matrix is solved less than 5 minutes on pure-CPU and cannot be solved in 30 minutes on MIC (i.e. rand_30K_3).

Table 3. Wallclock time (in seconds) of SuperLU_DIST with MIC offloaded 'dgemm' function

| | | Pure CPU Run | | | Offloaded to MIC | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | # of MPI Procs | 4 | 8 | 16 | 4 | 8 | 16 | | |
| Input Matrix | CAGE8<br>(1000x1000) | 0.24 | 0.07 | 0.29 | 290.29 | 259.85 | 231.82 | 10 | # of MIC Threads |
| | | | | | N/A | N/A | N/A | 30 | |
| | | | | | 241.18 | 212.48 | 217.92 | 60 | |
| | ADD32<br>(5000x5000) | 0.03 | 0.06 | 0.24 | 73.12 | 66.2 | 83.63 | 10 | |
| | | | | | 79.68 | 69.91 | 83.78 | 30 | |
| | | | | | 146.31 | 69.67 | 83.59 | 60 | |

Pure CPU runs do not scale-up because the size of the matrix is too small for the SuperLU_DIST and the scale-up breaks down due to overhead of MPI. The computation to communication ratio is small. After partitioning the input matrix, submatrices are very small so that these pieces cannot saturate the CPU and idle waiting occurs especially in MPI_Init phase (see Figure 1a and 1b). Therefore, the wallclock times in Table 3 are given only to provide a base time for comparison. When the code parts are offloaded to MIC, scale-up cannot be obtained. The main reason is the cumulative latency originated by the communication between host and the MIC card. The communication surpasses all other functions and therefore scale-up cannot be observed.

The memory limitation of the MIC card and wallclock time limitations do not allow us to test SuperLU_DIST for matrix *PRE2* and other large matrices which worked well for sequential SuperLU.

Native programming approach is also tested with SuperLU_DIST. As offload programming model is running the program partly on MIC, native approach can be considered as offloading the whole program. Factorisation time is often the dominant part of the wall clock time but when the library consumes considerable time outside of the factorisation (i.e. permutation, symbolic factorisation, back solve etc.) then the algorithm shows poor scale-up because of the serial parts of the algorithm. Running serial code blocks on MIC is inefficient due to slow clock rate compared to CPU. Therefore, unsatisfactory results are obtained with the native approach.

### *3.4.1. Analysis on BLAS calls*

The arguments of BLAS calls in SuperLU_DIST are inspected and profiled. The general matrix multiplication (GEMM) routine of the BLAS library is decisive in overall runtime. In this section, call arguments of one GEMM call will be presented for both of the randomly located matrices. A detailed analysis and comparison on SuperLU_DIST using BLAS and Intel Math Kernel Library (MKL) on CPU can be found in [4].

SuperLU_DIST does 1 897 974 GEMM calls in an execution with Rand_10K_3 matrix. In these calls, matrix sizes vary from 1x1 to 18x18. The output is a 1x1 matrix (a scalar), in 55.6% of these calls. This is a significant amount of multiplications. The output of GEMM call is a vector in 92.6% of all operations (1x1 matrices are included). Furthermore, in %80.3 of all operations whose outputs are 1x1 matrices, the input matrices (factors) are also 1x1 matrices. The remaining operations are inner products of vectors with sizes varying 2 to 18. Thus, 44.7% of all GEMM calls are actually product of scalars.

For the matrix “Rand_30K_3”, these statistics are: 16 473 153 total GEMM calls. Matrix sizes vary from 1x1 to 19x19. In %53.9 of these calls, the output is a scalar. In 92.0% of these calls, the output is a vector (1x1 matrices are included). In %80.4 of the operations whose outputs are 1x1 matrices, the factors are also scalar and 43.3% of all GEMM calls are actually product of scalars. Additional tests are also done with various input matrices but the results are similar. This also explains why BLAS on MIC performs faster than MKL on MIC.

## 4. Conclusions

In this work, sequential, multithreaded and distributed versions of SuperLU solvers are examined on Intel Xeon Phi coprocessors. This new architecture can benefit from high parallelism and large vectors. In order to get a benefit from offloading, input data needs to be at least around 60MB or more (it corresponds to a double matrix with approximately size of 2800 x 2800) based on the findings. The existence of the automatic offloading threshold validates this finding. For example, when we apply the offload programming approach for the sequential SuperLU by using 120 MIC threads, we have up to 45% performance improvement for the matrix *PRE2* because of its big chunks of data.

In experiments SuperLU_MT and SuperLU_DIST show low performance on MIC. There are several reasons for this. The most computationally intensive parts of these libraries are the BLAS library calls and therefore these calls are strong candidates for running on MIC or GPGPU but also these parts are called in excessive number of times with different data. Therefore, these parts are not appropriate for MIC. In every call, the input data is driven to the PCI-E bus which is slower than the bus between RAM and cache memory, also cache memory and CPU. The PCI-E bus has latency and when this cost is multiplied by the number of BLAS calls, it takes very long time and offloading these parts do not show any benefit from small matrix blocks in terms of CPU and wallclock time.

## Acknowledgements

This work was financially supported by the PRACE-1IP project funded in part by the EUs 7th Framework Programme (FP7/2007-2013) under grant agreement no. RI-261557. The work is achieved using the PRACE Research Infrastructure resources EURORA cluster of CINECA and UHeM cluster of Istanbul Technical University.

## References

[1] F. Affinito. Introduction to Xeon Phi Programming Model, Presentation, https://hpc-forge.cineca.it/files/CoursesDev/public/2013/Introduction%20to%20GPGPU%20and%20CUDA%20programming/Bologna/MIC.pdf

[2] 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.

[3] 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

[4] M. S. Celebi, A. Duran, M. Tuncel, B. Akaydın, and F. Oztoprak. Performance Analysis of BLAS Libraries in SuperLU_DIST for SuperLU_MCDT (Multi Core Distributed) Development, PRACE PN:283493, PRACE-2IP white paper, Libraries, WP 83, August 20, 2013. http://www.prace-project.eu/IMG/pdf/wp83.pdf

[5] http://www.cineca.it/en/content/eurora, Accessed in 24.12.2013

[6] http://ark.intel.com/products/61428, Accessed in 11.10.2013

[7] http://ark.intel.com/products/75801

[8] http://software.intel.com/en-us/intel-mkl

[9] Using Intel® MKL Automatic Offload on Intel® Xeon Phi™ Coprocessors, Intel White Paper, http://software.intel.com/sites/default/files/11MIC42_How_to_Use_MKL_Automatic_Offload_0.pdf

[10] T.A. Davis, University of Florida sparse matrix collection. http://www.cise.ufl.edu/research/sparse/matrices/.

[11] http://www.netlib.org/blas