Available online at www.prace-ri.eu

**Partnership for Advanced Computing in Europe**

# Design and Implementation of New Hybrid Algorithm and Solver on CPU For Large Sparse Linear Systems

Ahmet Duran[a,b]*, M. Serdar Celebi[a,c], 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**

It is important to have a fast, robust and scalable algorithm to solve a sparse linear system *AX=B*. Many multiscale modelling applications in science and engineering would like to capture more details of the system that results in more general matrices. In this work, we consider scalable direct solvers and, in particular, we examine the effectiveness of the SuperLU_DIST 3.0 for distributed memory and SuperLU_MT 2.0 for shared memory parallel machines. We use test matrices containing randomly populated sparse matrices in addition to patterned matrices. For randomly populated large sparse matrices, we find that numerical factorization, symbolic factorization, and consequently wall clock time spike up around the sparsity level of 7. We propose a new hybrid algorithm utilizing the MPI+OpenMP hybrid programming approach among other modifications to solve large sparse linear systems so that we can avoid extra communication overhead with MPI within node and we could have a better scalability than both pure MPI and OpenMP. It combines the advantages of SuperLU_DIST and SuperLU_MT and diminishes some of their limitations.

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

## 1. Introduction

It is important to have a fast, robust and scalable algorithm to solve a sparse linear system *AX=B* in many science and engineering applications. We design and implement new hybrid algorithm and solver for large sparse linear systems. In this work, we consider scalable direct solvers for various reasons. First, 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 Li et al. [1], Amestoy et al. [2], Schenk and Gartner [3, 4], Duran and Saunders [5], Duran et al. [6] and references contained therein).

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

SuperLU_MT (see Demmel et al. [7]) has three major steps including a) sparsity ordering, b) factorization that arranges partial pivoting, symbolic factorization and numerical factorization steps to perform in an alternating fashion, and c) triangular solution. While SuperLU_DIST uses BLAS 3 for factorization, SuperLU_MT has only BLAS 2.5 with multiple matrix vector multiplication. Therefore, SuperLU_DIST outperforms SuperLU_MT (see [8] and [15]) for various sparse matrices.

SuperLU_DIST (see Li and Demmel [9]) uses static pivoting [10] 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 due to the static pivoting. On the other hand, the backward error of a matrix cannot be decreased to machine precision and SuperLU_DIST may be considered for a certain types of matrices only. Therefore, it is important to determine and classify those matrices where SuperLU_DIST works well. The maximum matching algorithm (see Duff and Koster [11]) is utilized to maximize the product of the magnitudes of the diagonal entries for a matrix. SuperLU_DIST can use ParMeTiS [12] or MeTiS [13] ordering on the structure of $A+A^T$ in addition to the multiple minimum degree ordering on the structure of $A+A^T$ or $A^TA$ for fill-in reducing preordering. Unlike sequential SuperLU, SuperLU_DIST does not have a COLAMD option that works well for many unsymmetric sparse matrices to reduce fill-ins.

In this work, we discuss advantages and limitations of the SuperLU solvers. Although the existing versions of SuperLU are scalable and tuned for many matrices, they are sensitive to tuning and need further customization for various large sparse matrices. Therefore, we designed and generated a collection of large patterned and random sparse matrices which are larger than most of those real matrices from the University of Florida sparse matrix collection [14]. For example, we did sensitivity analysis to several parameters including total number of nonzeros and degree of sparsity for randomly populated sparse matrices.

We modify the SuperLU solvers in order to improve their scalability via several ways. We propose a new hybrid algorithm utilizing the MPI+OpenMP hybrid programming approach that combines the advantages of SuperLU_DIST and SuperLU_MT and diminishes some of their limitations. The remainder of this work is organized as follows. In Section 2, the test matrices including randomly populated matrices and patterned matrices are described. Later, the scalability of SuperLU-DIST and SuperLU_MT are discussed and several illustrative examples are given. Section 3 concludes this work.

## 2. Methods and results

Many multiscale modelling applications in science and engineering would like to capture more details of the system without ignoring any important conservation laws as much as possible, resulting in more general matrices. Therefore we consider a portfolio of test matrices containing randomly populated sparse matrices in addition to patterned matrices. We generate 30 different randomly populated matrices RAND_30K_3, ..., RAND_30K_100 for each. Each experiment is done at least four times. We describe the matrices in Table 1 and Table 2, respectively.

### *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 x $10^6$ | UHeM |
| RAND_30K_5 | 30000 | 150000 | 5 | 4.22 x $10^6$ | UHeM |
| RAND_30K_7 | 30000 | 210000 | 7 | 1.76 x $10^6$ | UHeM |

| Name | Order | NNZ | NNZ/N | Condition number | Origin |
|---|---|---|---|---|---|
| RAND_30K_9 | 30000 | 270000 | 9 | $2.51 \times 10^{6}$ | UHeM |
| RAND_30K_11 | 30000 | 330000 | 11 | $8.82 \times 10^{5}$ | UHeM |
| RAND_30K_30 | 30000 | 900000 | 30 | $1.13 \times 10^{6}$ | UHeM |
| RAND_30K_50 | 30000 | 1500000 | 50 | $7.03 \times 10^{5}$ | UHeM |
| RAND_30K_75 | 30000 | 2250000 | 75 | $1.16 \times 10^{6}$ | UHeM |
| RAND_30K_100 | 30000 | 3000000 | 100 | $3.39 \times 10^{6}$ | UHeM |
| RAND_10K_3 | 10000 | 30000 | 3 | $7.10 \times 10^{5}$ | UHeM |
| RAND_20K_3 | 20000 | 60000 | 3 | $3.19 \times 10^{5}$ | UHeM |
| RAND_30K_3 | 30000 | 90000 | 3 | $1.20 \times 10^{6}$ | UHeM |
| RAND_40K_3 | 40000 | 120000 | 3 | $3.90 \times 10^{6}$ | UHeM |
| RAND_50K_3 | 50000 | 150000 | 3 | $1.20 \times 10^{6}$ | UHeM |
| RAND_60K_3 | 60000 | 180000 | 3 | $2.14 \times 10^{6}$ | UHeM |

Table 2. Description of patterned matrices

| Patterned matrices | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Name | Order | NNZ | NNZ/N | Nonzero pattern symmetry | Numeric value symmetry | Condition number | Origin | Kind of problem |
| 7DIAG_1M_545 | 1000000 | 5450000 | 5.45 | 0% | 0% | $3.47 \times 10^{5}$ | UHeM | |
| BBMAT | 38744 | 1771722 | 45.73 | 53% | 0% | $2.09 \times 10^{9}$ | UFMM | Computational fluid dynamics (CFD) |
| ECL32 | 51993 | 380415 | 7.32 | 92% | 60% | $9.41 \times 10^{15}$ | UFMM | Semiconductor device |
| EMILIA_923 | 923136 | 40373538 | 43.74 | 100% | 100% | | UFMM | Geomechanical structural |
| G7JAC200SC | 59310 | 717620 | 12.10 | 3% | 0% | $1.43 \times 10^{14}$ | UFMM | Economic |
| HELM2D03LOWER_20K | 392257 | 1939353 | 4.94 | 0% | 0% | | UHeM | |
| INVEXTR1_NEW | 30412 | 1793881 | 58.99 | 97% | 72% | $2.77 \times 10^{18}$ | UFMM | CFD |
| LHR71C | 70304 | 1528092 | 21.74 | 0% | 0% | $1.56 \times 10^{17}$ | UFMM | Light hydrocarbon recovery |
| MARK3JAC140SC | 64089 | 376395 | 5.87 | 7% | 1% | $5.83 \times 10^{13}$ | UFMM | Economic |
| MIXTANK_NEW | 29957 | 1990919 | 66.46 | 100% | 99% | $4.40 \times 10^{11}$ | UFMM | CFD |
| PRE2 | 659033 | 5834044 | 8.85 | 33% | 7% | $3.11 \times 10^{23}$ | UFMM | Frequency-domain circuit simulation |
| STOMACH | 213360 | 3021648 | 14.16 | 85% | 0% | $8.01 \times 10^{1}$ | UFMM | 3D electro-physical model |

| Name | Order | NNZ | NNZ/N | Nonzero pattern symmetry | Numeric value symmetry | Condition number | Origin | Kind of problem |
|---|---|---|---|---|---|---|---|---|
| TWOTONE | 120750 | 1206265 | 9.99 | 24% | 11% | $4.46 \times 10^9$ | UFMM | Frequency-domain circuit simulation |
| WANG4 | 26068 | 177196 | 6.80 | 100% | 5% | $4.91 \times 10^4$ | UFMM | Semiconductor device |

*2.2. Scalability of SuperLU_DIST*

The code of SuperLU_DIST has been tested in order to measure the performance scalability of various randomly populated sparse matrices and patterned sparse matrices up to 512 cores (depending on number of nonzeros and sparsity level) on the Linux Nehalem Cluster (see [16]) available at the National Center for High Performance Computing (UHeM).

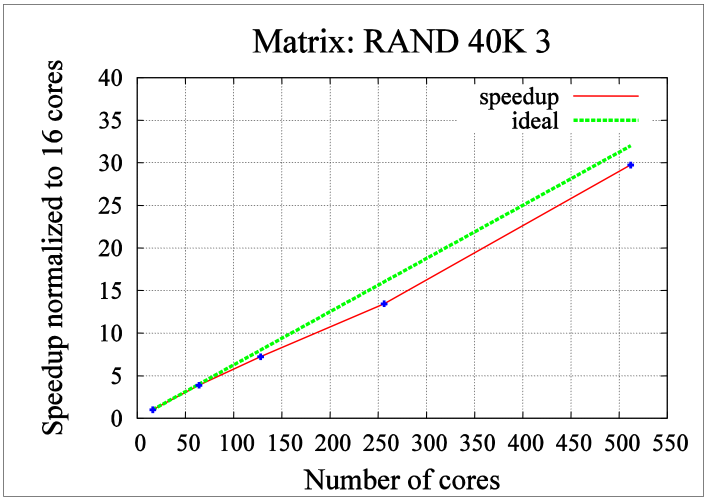


Fig. 1. Speed up for matrix RAND_40K_3

Table 3. Wall clock time and normalized speed-up for RAND_40K_3

| # cores (meshes) | Wall clock time (s) | Speed-up |
|---|---|---|
| 16 (4x4) | 849.69 | 1.00 |
| 64 (8x8) | 218.49 | 3.89 |
| 128 (8x16) | 117.55 | 7.23 |
| 256 (16x16) | 63.21 | 13.44 |
| 512 (16x32) | 28.58 | 29.73 |

The rich pattern spectrum of matrices and the NP-complete problem of best reordering for minimum fill-in are important challenges. For example, the code has shown scalable speed-up up to 512 cores for RAND_40K_3 in our tests as illustrated in Figure 1 and Table 3. While the speed-up for the symmetric matrix EMILIA_923 is close to ideal up to 256 cores, we observe divergence at 512 cores in Figure 2 and Table 4.

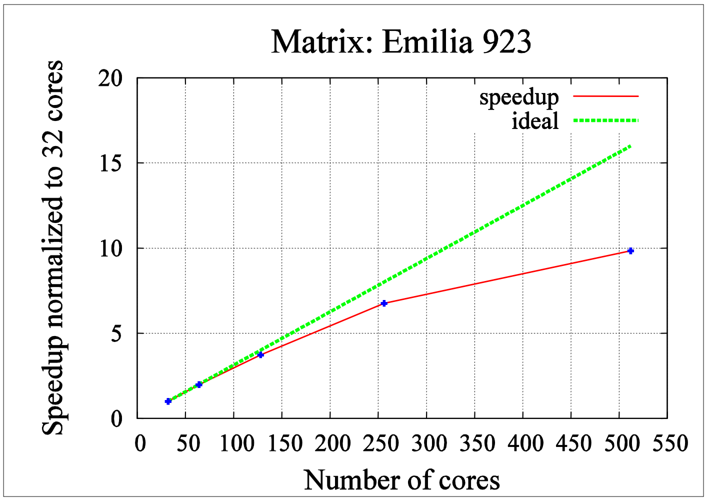


Fig. 2. Speed up for matrix EMILIA_923

Table 4. Wall clock time and normalized speed-up for EMILIA_923

| # cores (meshes) | Wall clock time (s) | Speed-up |
|---|---|---|
| 16 (4x4) | 1472.02 | 1.00 |
| 64 (8x8) | 743.29 | 1.98 |
| 128 (8x16) | 394.78 | 3.73 |
| 256 (16x16) | 217.85 | 6.76 |
| 512 (16x32) | 149.63 | 9.84 |

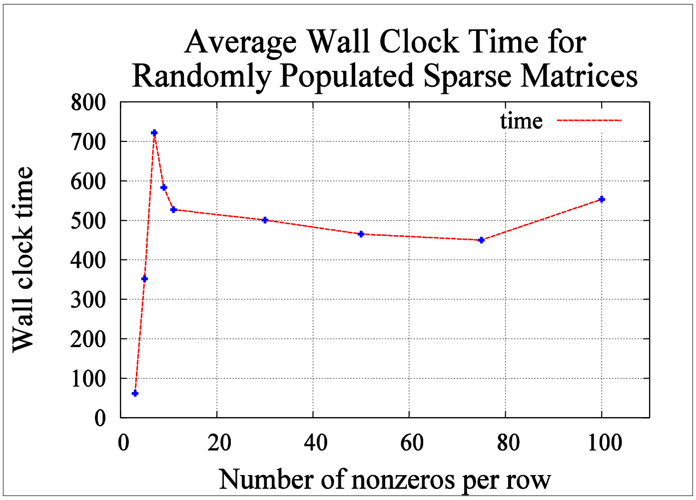


Fig. 3. Average wall clock time as a function of various sparsity levels for randomly populated sparse matrices.

Table 5. Wall clock time for randomly populated sparse matrices RAND_30K_3, ..., RAND_30K_100 as the sparsity level decreases using 64 core (8x8)

| NNZ per row | 3 | 5 | 7 | 9 | 11 | 30 | 50 | 75 | 100 |
|---|---|---|---|---|---|---|---|---|---|
| Wall clock time | 61.87 | 352.10 | 721.95 | 583.15 | 527.20 | 500.66 | 465.00 | 450.08 | 553.23 |

For randomly populated large sparse matrices, we find a peak of numerical factorization, symbolic factorization, and consequently wall clock time for a value of seven nonzeros per row in Figure 3 and Table 5. This may be related to availability of supernodes. After 7, they decrease gradually as sparsity decreases to 75 with a slow rise at 100 nonzeros per row.

Table 6. Distribution of wall clock time for randomly populated sparse matrices RAND_10K_3, ..., RAND_60K_3 as the number of nonzeros increases using 64 core (8x8)

| Order | 10000 | 20000 | 30000 | 40000 | 50000 | 60000 |
|---|---|---|---|---|---|---|
| NNZ | 30000 | 60000 | 90000 | 120000 | 150000 | 180000 |
| Equil time | 0.00 | 0.01 | 0.01 | 0.01 | 0.02 | 0.02 |
| RowPerm time | 0.01 | 0.02 | 0.04 | 0.06 | 0.12 | 0.11 |
| ColPerm time | 0.82 | 1.20 | 1.48 | 2.05 | 1.65 | 2.04 |
| SymFact time | 0.06 | 0.38 | 1.08 | 2.11 | 3.54 | 5.42 |
| Distribute time | 0.06 | 0.07 | 0.20 | 0.20 | 0.30 | 0.45 |
| Factor time | 0.98 | 14.65 | 74.95 | 212.43 | 334.01 | 857.66 |
| Solve time | 0.02 | 0.05 | 0.11 | 0.18 | 0.22 | 0.33 |
| Refinement time | 0.08 | 0.15 | 0.26 | 0.47 | 0.48 | 0.70 |
| Total | 2.03 | 16.53 | 78.13 | 217.51 | 340.34 | 866.73 |

In Table 6, the numerical factorization time dominates in the distribution of total wall clock time as expected for the randomly populated sparse matrices with 3 nonzeros per row. We observe that the wall clock time and consequently total time increases as matrix order and number of nonzeros increase, given fixed sparsity.

We find that the memory overhead coming from ParMeTiS [12] becomes one of the dominating factors in the distribution of wall clock time on n-diagonal sparse matrices for certain large numbers of cores. For example, we generated 7DIAG_1M_545 as a seven diagonal unsymmetric matrix with distances +50000, +100000, +400000, -200000, -300000 and -500000 from main diagonal having random 5450000 real numbers between 0.5 and 1. The column permutation time takes 41% of the wall clock time for 7DIAG_1M_545 when 64 cores are used. We find similar results for this kind of n-diagonal unsymmetric/symmetric sparse matrices while using a number of cores such as 64. This affects the scalability of SuperLU_DIST negatively. In Table 7, the total time increased from 9.96 s. (16 cores) to 17.38 s. (64 cores).

Table 7. Distribution of wall clock time (sec.) for 7DIAG_1M_545 using ParMeTiS and MeTiS respectively for column permutation

| | ParMeTiS | | | MeTiS | | |
|---|---|---|---|---|---|---|
| # of cores (mesh) | 4 (2x2) | 16 (4x4) | 64 (8x8) | 4 (2x2) | 16 (4x4) | 64 (8x8) |
| Equil time | 0.09 | 0.17 | 0.21 | 0.09 | 0.17 | 0.21 |
| RowPerm time | 0.83 | 0.85 | 0.88 | 0.80 | 0.85 | 0.88 |
| ColPerm time | 3.41 | 2.30 | **7.11** | 10.06 | 10.29 | 10.55 |
| SymFact time | 0.34 | 0.17 | 0.20 | 0.24 | 0.25 | 0.25 |
| Distribute time | 1.17 | 0.64 | 0.54 | 0.59 | 0.41 | 0.13 |
| Factor time | 2.00 | 2.62 | 6.07 | 0.53 | 0.43 | 0.55 |
| Solve time | 0.92 | 0.75 | 0.56 | 0.25 | 0.15 | 0.08 |
| Refinement time | 3.09 | 2.46 | 1.81 | 1.04 | 0.66 | 0.37 |
| Total | 11.85 | 9.96 | 17.38 | 13.60 | 13.21 | 13.02 |

### *2.3. Scalability of SuperLU_MT*

The code of SuperLU_MT has been tested up to 64 threads for all sparse matrices in the list on a HP Integrity Superdome SD32B (see [17]), a computing server with shared memory architecture at UHeM. A performance scalability between 4 (LHR71C, an unsymmetric matrix with low sparsity) and 32 (MIXTANK_NEW, a small almost symmetric matrix with low sparsity) is achieved depending on the number of nonzeros per row, total number of nonzero and structural symmetry. For example, the speed up graphs of MIXTANK_NEW and TWOTONE are represented in Figure 4 and Figure 5, respectively. MIXTANK_NEW has more number of nonzeros than that of TWOTONE and has a better scalability. These results with different machine are in the line of Demmel et al. [7].

While SuperLU_DIST works well for a symmetric sparse matrix EMILIA_923 with less sparsity, we observe that SuperLU_MT gives segmentation fault for it and similar large matrices related to memory usage. The measurements of SuperLU_MT for the patterned matrices from Table 2 are listed in Table 8.

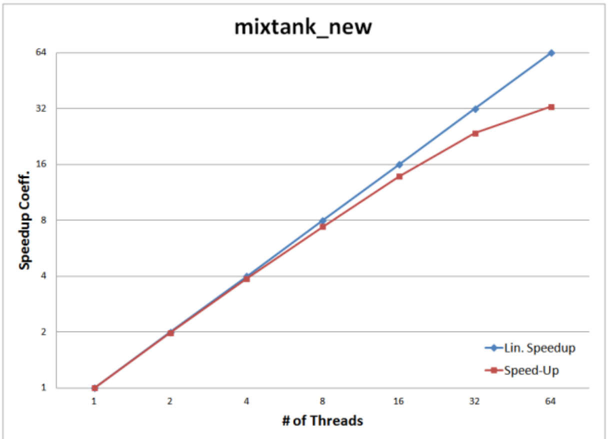


Fig. 4. Speed up graph of MIXTANK_NEW for SuperLU_MT

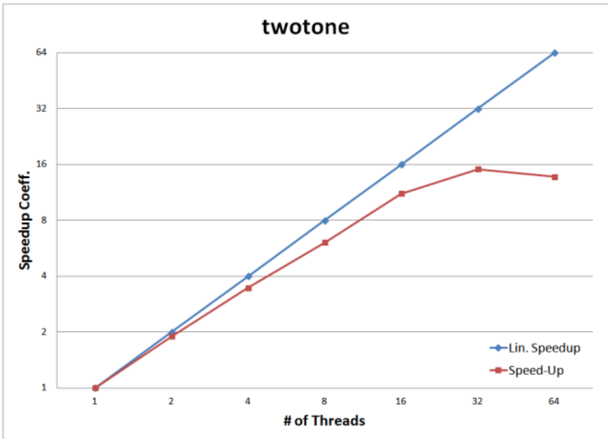


Fig. 5. Speed up graph of TWOTONE for SuperLU_MT

Table 8. Wall clock time for patterned matrices in seconds

| Patterned matrices # of threads | | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|---|---|
| BBMAT | Minimum | 548.88 | 295.77 | 155.43 | 90.33 | 55.88 | 37.99 | 27.44 |
| | Average | 549.71 | 296.15 | 156.24 | 90.66 | 56.53 | 38.27 | 27.66 |
| | Maximum | 550.58 | 296.32 | 157.48 | 90.82 | 57.06 | 38.61 | 27.86 |
| | Speed up | 1.00 | 1.86 | 3.52 | 6.06 | 9.72 | 14.36 | 19.88 |
| ECL32 | Minimum | 1219.38 | 638.21 | 362.33 | 225.55 | 128.27 | 86.41 | 63.50 |
| | Average | 1221.05 | 639.31 | 362.92 | 226.02 | 128.51 | 86.92 | 63.97 |
| | Maximum | 1221.91 | 639.81 | 363.93 | 226.59 | 128.70 | 87.31 | 64.69 |
| | Speed up | 1.00 | 1.91 | 3.36 | 5.40 | 9.50 | 14.05 | 19.09 |
| G7JAC200SC | Minimum | 352.52 | 176.76 | 91.68 | 48.27 | 26.51 | 15.53 | 11.27 |
| | Average | 352.65 | 177.05 | 91.99 | 48.40 | 26.62 | 15.57 | 11.55 |

| # of threads | | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
|---|---|---|---|---|---|---|---|---|
| G7JAC200SC | Maximum | 352.79 | 177.18 | 92.34 | 48.49 | 26.76 | 15.61 | 12.02 |
| | Speed up | 1.00 | 1.99 | 3.83 | 7.29 | 13.25 | 22.64 | 30.52 |
| INVEXTR1_NEW | Minimum | 211.18 | 108.22 | 57.05 | 31.66 | 18.62 | 12.69 | 11.63 |
| | Average | 211.25 | 108.30 | 57.24 | 31.75 | 18.68 | 12.84 | 11.84 |
| | Maximum | 211.33 | 108.52 | 57.42 | 31.86 | 18.77 | 12.90 | 12.15 |
| | Speed up | 1.00 | 1.95 | 3.69 | 6.65 | 11.31 | 16.45 | 17.85 |
| LHR71C | Minimum | 29.21 | 15.59 | 9.33 | 5.78 | 3.81 | 2.84 | 2.72 |
| | Average | 29.28 | 15.61 | 9.37 | 5.82 | 3.83 | 2.86 | 2.80 |
| | Maximum | 29.40 | 15.68 | 9.41 | 5.85 | 3.85 | 2.88 | 3.00 |
| | Speed up | 1.00 | 1.88 | 3.13 | 5.03 | 7.65 | 10.24 | 10.45 |
| MARK3JAC140SC | Minimum | 604.50 | 342.85 | 214.20 | 122.16 | 88.26 | 71.06 | 56.66 |
| | Average | 605.04 | 343.29 | 214.74 | 122.50 | 88.42 | 71.23 | 57.29 |
| | Maximum | 605.65 | 343.95 | 215.45 | 122.95 | 88.70 | 71.39 | 57.69 |
| | Speed up | 1.00 | 1.76 | 2.82 | 4.94 | 6.84 | 8.49 | 10.56 |
| MIXTANK_NEW | Minimum | 806.82 | 407.13 | 206.88 | 108.75 | 58.27 | 34.11 | 24.38 |
| | Average | 806.90 | 407.24 | 207.68 | 109.04 | 58.47 | 34.20 | 24.61 |
| | Maximum | 807.01 | 407.49 | 208.47 | 109.21 | 58.74 | 34.37 | 24.93 |
| | Speed up | 1.00 | 1.98 | 3.89 | 7.40 | 13.80 | 23.59 | 32.79 |
| PRE2 | Minimum | 15633.23 | 8256.28 | 4685.85 | 2655.72 | 1597.11 | 1149.65 | 845.54 |
| | Average | 15651.66 | 8265.67 | 4691.67 | 2657.74 | 1606.55 | 1167.28 | 872.38 |
| | Maximum | 15667.32 | 8272.05 | 4696.84 | 2660.19 | 1614.53 | 1176.26 | 891.79 |
| | Speed up | 1.00 | 1.89 | 3.34 | 5.89 | 9.74 | 13.41 | 17.94 |
| STOMACH | Minimum | 764.92 | 385.73 | 205.55 | 112.11 | 65.85 | 47.74 | 45.00 |
| | Average | 765.21 | 392.20 | 205.88 | 112.44 | 66.15 | 47.89 | 45.09 |
| | Maximum | 765.90 | 395.59 | 206.25 | 112.73 | 66.35 | 48.00 | 45.30 |
| | Speed up | 1.00 | 1.95 | 3.72 | 6.81 | 11.57 | 15.98 | 16.97 |
| TORSO1 | Minimum | 283.62 | 141.97 | 78.03 | 42.29 | 24.39 | 17.12 | 14.17 |
| | Average | 284.02 | 142.37 | 78.27 | 42.46 | 24.48 | 17.14 | 15.16 |
| | Maximum | 284.51 | 142.68 | 78.57 | 42.55 | 24.58 | 17.17 | 16.15 |
| | Speed up | 1.00 | 1.99 | 3.63 | 6.69 | 11.60 | 16.57 | 18.74 |
| TWOTONE | Minimum | 46.17 | 24.33 | 13.22 | 7.60 | 3.79 | 2.96 | 3.30 |
| | Average | 46.22 | 24.46 | 13.38 | 7.61 | 4.15 | 3.07 | 3.37 |
| | Maximum | 46.31 | 24.85 | 13.51 | 7.64 | 4.25 | 3.10 | 3.49 |
| | Speed up | 1.00 | 1.89 | 3.46 | 6.07 | 11.15 | 15.06 | 13.70 |
| WANG4 | Minimum | 78.67 | 39.94 | 20.57 | 11.32 | 6.88 | 5.43 | 6.02 |
| | Average | 78.74 | 39.97 | 20.64 | 11.36 | 6.91 | 5.45 | 6.15 |
| | Maximum | 78.94 | 40.03 | 20.69 | 11.38 | 6.96 | 5.51 | 6.27 |
| | Speed up | 1.00 | 1.97 | 3.82 | 6.93 | 11.40 | 14.44 | 12.80 |

### *2.4. Other limitations of SuperLU solvers*

Although the existing versions of SuperLU work well for many matrices, they need to be improved for certain types of sparse matrices. For example, we generated a new unsymmetric matrix HELM2D03LOWER_20K, shown in Figure 6, which consists of the lower triangular part of a symmetric matrix HELM2D03 from the University of Florida sparse matrix collection and an upper subdiagonal with 20000 distance from the main diagonal. Although SuperLU_DIST works well for the matrix HELM2D03on the Linux Nehalem Cluster (see [16]) available at UHeM, it produces segmentation fault for HELM2D03LOWER_20K. We found the same results with several similar matrices that we generated.

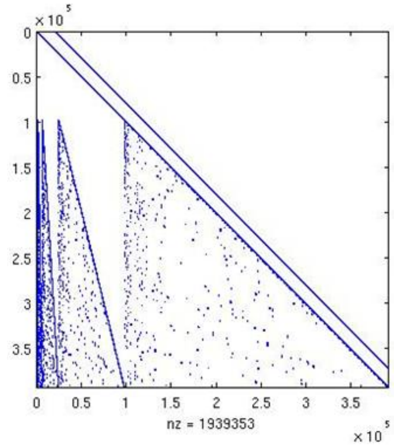


Fig. 6. Matrix picture of HELM2D03LOWER_20K

## 3. Conclusions

We believe that there is no unique solver that fits all our needs for every matrices because of the rich pattern spectrum of matrices and the NP-complete problem of best reordering for minimum fill-in. We need always a better solver as multiscale modelling develops.

SuperLU_DIST has shown scalable speed-up between 256 and 512 cores for many test matrices. On the other hand, for randomly populated large sparse matrices, we find a peak of wall clock time around 7 for the number of nonzeros per row related to the ability to find supernodes. After 7, it decreases gradually as sparsity level decreases to 100 nonzeros per row. Moreover, we find that the memory overhead coming from ParMeTiS becomes one of the dominating factors in the distribution of wall clock time on n-diagonal sparse matrices for certain large number of cores. Furthermore, we generated new unsymmetric matrices which consists of the lower triangular part of a symmetric matrix and an upper subdiagonal with $d$ distance from the main diagonal. While SuperLU_DIST performs properly for the symmetric matrices, it produces segmentation fault for the corresponding new unsymmetric matrices.

The code of SuperLU_MT has been tested up to 64 threads for all sparse matrices in the list on HP Integrity Superdome SD32B (see [17]) computing server. A scalability between 4 and 32 is achieved depending on the sparsity level, total number of nonzeros and structural symmetry, as shown by Demmel et al. [7] with different machine. Finally, we find very large sparse matrices with less sparsity for which SuperLU_DIST works well while SuperLU_MT gives segmentation fault them related to memory usage.

Based on these results, we designed a new hybrid algorithm utilizing the MPI+OpenMP hybrid programming approach among other modifications to solve large sparse linear systems so that we can avoid extra communication overhead with MPI within node and we could have a better scalability than both pure MPI and OpenMP.

## 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), referenced as GSLU also, a part of LinBox package. GSLU contains 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. 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.

8. X.S. Li. Evaluation of sparse LU factorization and triangular solution on multicore platforms. Computing for Computational Science-VECPAR 2008, Springer.

9. 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.

10. 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.

11. I.S. Duff and J. Koster, The design and use of algorithms for permuting large entries to the diagonal of sparse matrices, SIAM J. Matrix Anal. Appl. 20 (4) (1999) 889–901.

12. G. Karypis, K. Schloegel, and V. Kumar. ParMeTiS: Parallel graph partitioning and sparse matrix ordering library, version 3.1. University of Minnesota, 2003. http://www-users.cs.umn.edu/~karypis/metis/parmetis/.

13. G. Karypis and V. Kumar. MeTiS, a software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0. University of Minnesota, September 1998. http://www-users.cs.umn.edu/~karypis/metis/.

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

15. M.S. Celebi, A. Duran, M.Tuncel, and 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.

16. http://www.uybhm.itu.edu.tr/eng/inner/duyurular.html#karadeniz

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