OPT-*i*
An International Conference on
Engineering and Applied Sciences Optimization
M. Papadrakakis, M.G. Karlaftis, N.D. Lagaros (eds.)
Kos Island, Greece, 4-6 June 2014

MORSy – A NEW TOOL FOR SPARSE MATRIX REORDERING

Anna Yu. Pirova[1], **Iosif B. Meyerov** [1]

[1] Lobachevsky State University of Nizhni Novgorod
23 Gagarin av., 603950, Nizhni Novgorod, Russia
{pirova, meerov}@vmk.unn.ru

Keywords: multilevel nested dissection, sparse matrix reordering, reducing fill-in, vertex separator, Cholesky factorization.

Abstract. *When direct methods for solving sparse linear systems of equations are used, an important stage of the solution is to reorder matrix rows and columns to reduce the number of non-zero elements of the matrix factor. We present MORSy – a new tool for reordering symmetric sparse matrices. It is based on multilevel nested dissection algorithm with modifications for vertex separators. Experimental results prove that MORSy is competitive to METIS and Scotch libraries both in ordering quality and performance.*

1 INTRODUCTION

Systems of linear equations $Ax = b$ with sparse symmetric positive definite matrix A arise in a wide range of research and engineering problems in physics, chemistry, economics, finance, and other domains. There are direct and iterative methods of solving such systems. Direct methods are based on factorization of the matrix A into two triangular matrices ($A = LL^T$) [1], while iterative methods are based on step-by-step approximation to the solution x [2]. When direct methods are used, so-called fill-in of the matrix occurs – as a rule, the number of non-zero elements of the factor is much greater than the number of non-zero elements of initial matrix. This effect can lead to significant memory requirements for factor storage and critical increase in factorization time. For reduction of factor fill-in, ordering of the matrix rows and columns is applied. Finding correct ordering that minimizes factor fill-in is an NP-complete problem of discrete optimization [3]. In practice, there are two commonly used heuristic approaches for performing reordering to reduce factor fill-in: nested dissection and minimum degree algorithms.

The minimum degree algorithm was proposed by Tinney and Walker in 1969 [4]. It models the Gaussian elimination process and is based on the local factor minimization strategy. At each step of the algorithm the vertex with the smallest degree is eliminated and numbered, while its neighbors are contracted to the clique. The most time consuming operation of this algorithm is vertex degree recalculation after every step. Since 1980s a number of modifications of the minimum degree algorithm for improving its runtime and quality has been developed, including Multiple Minimum Degree [5] (Liu, 1985), Approximate Minimum Degree [6] (Amestoy, Davis, Duff, 1996), Column Approximate Minimum Degree [7] (Davis, Gilbert at el., 2004) and others.

The nested dissection algorithm for finite element meshes was proposed by George in 1973 [8] and was generalized for irregular graphs by Lipton, Rose, Tarjian [9], and George, Liu [10]. It is based on the global factor minimization strategy. The notion of a *separator* is central to this algorithm. It is a set of graph vertexes, removal of which divides the graph into two disconnected parts. The nested dissection algorithm is as follows: to divide the matrix graph into two disconnected subgraphs by a small vertex separator, to number separator vertexes from highest available indexes, and then to process the produced subgraphs recursively. Finding a small separator that divides the graph into two roughly equal subgraphs determines the quality of ordering. Since 1993 modifications of the nested dissection algorithm based on the multilevel graph partition procedure are widely used. This approach was proposed by Bui and Jonse [11] and improved by Karypis and Kumar [12], Hendrickson and Leland [13], Hendrickson and Rothberg [14] and others. When the multilevel method for determining the separator is used, finding graph separator consists of three stages: coarsening, partitioning, and uncoarsening. During the *coarsening* stage matching techniques are used to construct a sequence of graphs, where the structure of each graph coarsens the structure of the previous one. During the *partitioning* stage the separator of the coarsest graph is determined. It is performed by either direct finding vertex separator or finding an edge separator followed by computing a vertex separator from the edge separator. During the *uncoarsening* stage the separator of the coarsest graph is projected back to the original graph through the sequence of intermediate graphs. At each step of the uncoarsening stage the separator is refined to reduce its size and to balance subgraphs. Usually it is performed by the modifications of the local optimization algorithm by Kernighan-Lin [15] or Fiduccia-Mattheyses [16]. The advantage of the multilevel scheme is that time-consuming separation algorithms are applied to small graphs, and refinement of projected separators is carried out by an iterative algorithm starting from good initial approximation. This allows to reduce reordering time and improve its quality in comparison with the classical nested dissection method.

Currently, there exist a number of open source implementations of sparse matrix ordering that can be used sequentially or in parallel for distributed memory systems. We compare our results with widely used METIS [17] and Scotch [18] libraries that are based on the multilevel nested dissection algorithm. Most of sparse linear solvers have interfaces for running external reordering libraries and their own ordering implementations. For example, the modifications of minimum degree and nested dissection orderings are implemented in Intel MKL Pardiso [19], SuperLU [20], MUMPS [21], CHOLMOD [22], and others. We present MORSy – a new open source fill-in minimization software based on the multilevel nested dissection algorithm with modifications for using vertex separator at all steps of multilevel scheme.

The remainder of the paper is organized as follows. Section 2 defines the fill-in minimization problem. Section 3 describes multilevel nested dissection implementation in MORSy. Section 4 presents the experimental results of MORSy and compares its performance and orderings quality with that of METIS and Scotch libraries. Section 5 gives a summary of experimental results and observes future work.

2 PROBLEM STATEMENT

Let $A = (a_{ij})$ be a sparse symmetric n by n matrix. Let us construct a matrix graph $G = (V, E)$ with the set of vertexes V and the set of edges E, where each vertex v_i is associated with matrix row i ($i = 1, 2, ..., n$), and each edge (v_i, v_j) is associated with non-zero element of matrix, i.e. $(v_i, v_j) \in E$ if and only if $a_{ij} \neq 0$ ($i, j = 1, 2, ..., n$; $i \neq j$). The set of vertexes that are adjacent to a vertex v is denoted by Adj(v).

When *elimination* of vertex v from graph G is performed, edges between vertexes adjacent to vertex v are added to the graph so that they become a clique, vertex v is deleted from the set of vertexes together with all incident edges:

$$V = V \setminus v;$$

$$E = E \setminus \{(u, v): u \in \mathrm{Adj}(v)\} \cup \{(u_1, u_2): u_1, u_2 \in \mathrm{Adj}(v)\}$$

The added edges are associated with the elements that became non-zero during Gaussian elimination of v-th matrix row. Let $\pi = (\pi_1, \pi_2, ..., \pi_n)$ be a permutation of the set of vertexes V. *Fill-in* $F(\pi)$ generated by the permutation π is a set of edges added during the consequent elimination of vertexes $\pi_1, \pi_2, ..., \pi_n$. Problem of finding the permutation π^* that minimizes number of edges in produced fill-in is NP-complete [3]:

$$\pi^* = \mathrm{argmin}\ \{|F(\pi)|\}$$

Let us denote the *quality of ordering* as the number of nonzero elements of the Cholesky factor of the matrix after applying this ordering.

3 MULTILEVEL NESTED DISSECTION IN MORSY

Reordering in MORSy is based on the classical multilevel scheme with modification of stages for vertex separator (Figure 1).

```
Program MultilevelNestedDissection ($G_A(V_A, E_A)$, Iperm)
Input: $G_A(V_A, E_A)$ – a graph constructed from the sparse symmetric matrix $A$ structure.
Output: Iperm – a new numbering of $G_A$ vertexes ($A$ rows).

    1   $G(V, E)$ = Compress($G_A$)
```

```
2   while $G$ isn' t numbered do
3     $G_0(V_0, E_0)$ = current subgraph of $G$
4     if $|V_0|$ is small enough then
5       number the nodes in $G$ by automatic nested dissection
6       $V = V \setminus V_0$;
7     else
8       $i = 0$;
9       while $G_i$ is big enough do
10        $G_{i+1}(V_{i+1}, E_{i+1})$ = Coarse ($G_i$); $i$++;
11      end while
12      $m = i$;
13      $P_m(S_m, V_{m,1}, V_{m,2})$ = InitializePartition($G_m$) with separator $S_m$;
14      for $i = m$ downto 1 do
15        $P_{i-1}(S_{i-1}, V_{i-1,1}, V_{i-1,2})$ = ProjectPartition($P_i$);
16        $P_{i-1}(S_{i-1}, V_{i-1,1}, V_{i-1,2})$ = RefinePartition($P_{i-1}$);
17      end for
18      Number vertexes from $S_0$;
19      $V = V \setminus S_0$;
20    end if
21  end while
22  $Iperm$ = ProjectNumbers($G_A$);
```

Figure 1: Structure of multilevel nested dissection in MORSy. Stages are modified for use of a vertex separator.

First, compressing of graph structure [14] is performed to reduce reordering time (Figure 1, line 1). Then, in the main loop of the algorithm (Figure 1, lines 2-21) a separator is found for each subgraph G_0 of the initial graph G, which is constructed during the processing of nested dissection ordering. If the number of graph vertexes is big enough, its separator is defined using the multilevel technique (Figure 1, lines 8-17).

At the coarsening stage (Figure 1, lines 8-11) a sequence of graphs G_1, G_2, ..., G_m, is formed, where each following graph flows out of the previous one by contracting edges and merging their incident vertexes. Heavy edge matching or random matching [12] are used for this purpose.

At the partitioning stage (Figure 1, line 13) the vertex separator of the graph is fined by building a rooted level structure from pseudoperipheral vertex [23].

At the uncoarsening stage (Figure 1, lines 14-17) the Primitive moves method is used for partition refinement. This method is a modification of the iterative Kernighan-Lin method adapted for vertex separator by Ashcraft and Liu [24] and modified for the multilevel scheme by Hendrickson and Rothberg [14]. The method uses the notion of a partition $P = (S, V_1, V_2)$ as a union of free disjoint sets of graph vertexes, were S is separator and V_1, V_2 are the sets of vertexes of disconnected subgraphs produced after separator's deleting. The essence of the method is as follows: each vertex $s \in S$ from the partition $P = (S, V_1, V_2)$ is associated with "gain" – change of separator size when moving this vertex to one of the parts V_1, V_2 (let us denote it gain($s \rightarrow V_1$), gain($s \rightarrow V_2$) respectively). Then as per rule from Figure 2 a series of separator vertex moves is carried out:

```
1. $M = \{v \in V$: $v$ have not been selected during this sequence of moves$\}$
2. $s_1$ = argmax {gain($u \rightarrow V_1$), $u \in S \cap M$ }; maxV1 = gain($s_1 \rightarrow V_1$);
3. $s_2$ = argmax {gain($u \rightarrow V_2$), $u \in S \cap M$ }; maxV2 = gain($s_2 \rightarrow V_2$);
```

```
4.  if maxV1 > maxV2 then
5.    Move $s_1$ to $V_1$; $M = M / s_1$;
6.  else
7.    if maxV2 > max V1 then
8.      Move $s_2$ to $V_2$; $M = M / s_2$;
9.    else // maxV1 = maxV2
10.     if $|V_1| < |V_2|$ then
11.       Move $s_1$ to $V_1$; $M = M / s_1$;
12.     else
13.       Move $s_2$ to $V_2$; $M = M / s_2$;
14.     end if
15.   end if
16. end if
```

Figure 2: The rule of moving the separator vertexes from the Primitive move algorithm [1].

If moving of vertice *s* improves the partition, then the best found partition *P** is updated. Then the process is repeated for a newly received partition *P**. Thus, the algorithm consists of two nested loops, the external one corresponds to the loop on various partitions, and the internal one corresponds to the loop on vertex moves from the current partition.

The rule of moving of separator vertexes significantly influences the quality of resulting orderings. When using rule (Figure 2) it is possible that the imbalance of partition received at the previous iteration of the internal loop of the algorithm has to be compensated at the next iteration. We changed the rule of vertex moves for most effective balancing tracking so that at each iteration of the external loop moves are carried out only into one part of partition (smaller one at the beginning of the external loop). It also reduces storage requirements as it is not necessary to store the gains of moving vertexes to another part of the partition.

For reduction of run time of the algorithm at the uncoarsening stage, partition refinement can be used not for every intermediate graph from the sequence of G_1, G_2, ..., G_m, but once per several iterations. Besides, it has been experimentally established that limiting the number of algorithm iterations also reduces its run time with an insignificant loss in quality for the majority of graphs.

4 EXPERIMENTAL RESULTS

4.1 Test environment

We tested MORSy with the matrices from The University of Florida Sparse Matrix Collection [25]. Matrix sizes varied from 200 000 to 1 500 000. Table 1 gives the description of the set of matrices.

Matrix name	N	NZ	Description
pwtk	217 918	11 524 432	structural problem
msdoor	415 863	19 173 163	structural problem
parabolic_fem	525 825	3 674 625	computational fluid dynamics problem
tmt_sym	726 713	5 080 961	electromagnetics problem
boneS10	914 898	40 878 708	3D structural problem
Emilia_923	923 136	40 373 538	3D structural problem
audikw_1	943 695	77 651 847	3D problem

bone010	986 703	47 851 783	3D problem
ecology2	999 999	4 995 991	2D structural problem
thermal2	1 228 045	8 580 313	unstructured FEM
StocF-1465	1 465 137	21 005 389	computational fluid dynamics problem
Hook_1498	1 498 023	59 374 451	3D structural problem
Flan_1565	1 564 794	114 165 372	3D structural problem
G3_circuit	1 585 478	7 660 826	circuit simulation problem

Table 1: Description of the test matrices.
N is the number of matrix rows, NZ is the number of non-zero elements of the matrix.

All experiments were performed on Intel Xeon E5-2690 CPU (8 cores, 2.9 GHz) with 64 GB of RAM, running OS Linux. MORSy was compiled with Intel® C++ Composer (from Intel Parallel Studio XE 2013); Intel MKL library was used for random number generation.

The quality of orderings was evaluated with respect to the number of non-zero elements in the factor of reordered matrix and time needed for the ordering.

4.2 Reordering parameters

Reordering in MORSy allows various parameter setting (Table 2), with prioritization of run time minimization and reordering quality maximization. Values of the parameters used in experiments are described in Table 2.

Parameter name	Range of values
Coarsening method	Random matching, heavy edge matching
Number of coarsening steps	10, 15
Partition quality evaluation function coefficient	0.20, 0.25, 0.30
Step of partitioning refinement during Uncoarsening process	1 (for every intermediate graph), 2 (for every second intermediate graph)
Limit of the number of iterations of the partition refinement algorithm at intermediate Uncoarsening steps	No limits, Limited – not more than one iteration

Table 2: MORSy parameters used in experiments.

4.3 Comparison with other ordering libraries

Table 3 shows the quality of orderings produced by MORSy, METIS and Scotch. MORSy was run with the best parameter configuration with respect to factor fill-in for each matrix. We denote this parameter configuration as “quality-priority”. METIS and Scotch were run with the default parameters of ordering routines. Scotch was run under the METIS-compatible interface.

In comparison with METIS, in 7 test matrices out of 14 MORSy provided orderings with a better quality. For *ecology2* matrix the size of the factor is 26% better, for other matrices it is 1-3% better. For the remaining 7 matrices from the test set MORSy provides orderings with a worse quality than METIS. The size of the factor is 9-10% larger for two matrices (boneS10, Hook_1498), for other matrices it is 1-3% larger. Thus, in 12 matrices out of 14 MORSy parameter adjustment allows orderings that are very close to orderings of METIS or better in quality.

In comparison with default reordering in Scotch, MORSy orderings are better for all test matrices. The factor size advantage is 20 to 60% for 5 matrices, 10 to 20% for 5 matrices, and less than 10% for 4 matrices (19% on average).

Matrix name	N	Factor NZ, METIS	Factor NZ, MORSY	Factor NZ, Scotch
pwtk	217 918	47 124 530	46 784 875	56 116 478
msdoor	415 863	51 483 893	52 085 831	83 374 463
parabolic_fem	525 825	25 607 853	24 923 337	28 575 855
tmt_sym	726 713	29 507 621	28 741 732	35 754 899
boneS10	914 898	267 940 257	295 723 050	339 280 809
Emilia_923	923 136	1 636 886 316	1 650 689 751	1 715 779 992
audikw_1	943 695	1 216 865 448	1 200 984 910	1 204 126 326
bone010	986 703	1 049 932 740	1 035 907 995	1 249 173 615
ecology2	999 999	35 641 736	30 081 507	43 675 655
thermal2	1 228 045	50 430 085	51 356 445	58 403 914
StocF-1465	1 465 137	1 037 743 963	1 072 623 748	1 111 688 726
Hook_1498	1 498 023	1 484 282 865	1 617 131 106	1 863 646 578
Flan_1565	1 564 794	1 456 370 148	1 415 670 363	1 546 352 973
G3_circuit	1 585 478	90 916 423	91 292 425	107 035 058

Table 3: Comparison of number of factor non-zero elements after performing reordering.
N is number of matrix rows, NZ is number of non-zero elements after factorization.

Table 4 presents the comparison of run time of MORSy, METIS and Scotch for obtaining the above mentioned orderings. On 5 matrices out of the 14 MORSy works 1.13 to 1.99 times faster than METIS, and on 4 matrices it is 1.02 to 1.64 times slower, and on 5 matrices it works 2.11 to 2.53 times slower. In comparison with Scotch, MORSy has 1.04 to 2.64 times performance advantage on the half of the test matrices and 1.05 to 1.92 times disadvantage on other matrices.

Matrix name	N	Reordering time, METIS	Reordering time, MORSY	Reordering time, Scotch
pwtk	217 918	0,44	0,34	0,46
msdoor	415 863	0,59	0,60	0,60
parabolic_fem	525 825	2,75	5,95	3,76
tmt_sym	726 713	4,04	9,77	5,10
boneS10	914 898	5,43	8,92	5,95
Emilia_923	923 136	5,48	3,74	5,64
audikw_1	943 695	8,27	4,16	10,99
bone010	986 703	7,08	8,26	7,86
ecology2	999 999	4,50	9,66	5,99
thermal2	1 228 045	7,31	18,46	10,44
StocF-1465	1 465 137	14,96	31,52	26,81
Hook_1498	1 498 023	9,73	8,62	8,98
Flan_1565	1 564 794	11,47	10,07	12,37
G3_circuit	1 585 478	9,19	9,43	13,33

Table 4: Comparison of the run time of METIS, MORSy, Scotch.
N is number of matrix rows, NZ is number of non-zero elements after factorization. All times are in seconds.

Table 5 shows MORSy quality and run time with the parameter configuration that minimizes reordering time while keeping factor increase less than 10%, in comparison with the previous results. Let us denote this parameters configuration “time-priority”.

Matrix name	N	Factor NZ, MORSy	Reordering time, MORSy	Factor increase	Time decrease
Pwtk	217 918	48 112 201	0.31	2.8%	9.7%
Msdoor	415 863	53 500 238	0.5	2.7%	20.0%
parabolic_fem	525 825	25 450 749	3.45	2.1%	72.5%
tmt_sym	726 713	29 517 952	6.28	2.7%	55.6%
boneS10	914 898	315 302 577	5.81	6.6%	53.5%
Emilia_923	923 136	1 713 036 243	3.45	3.8%	8.4%
audikw_1	943 695	1 200 984 910	4.16	0.0%	0.0%
bone010	986 703	1 081 234 524	5.38	4.4%	53.5%
ecology2	999 999	33 076 977	5.6	10.0%	72.5%
thermal2	1 228 045	53 088 061	11.8	3.4%	56.4%
StocF-1465	1 465 137	1 159 553 480	19.57	8.1%	61.1%
Hook_1498	1 498 023	1 682 587 980	8.11	4.0%	6.3%
Flan_1565	1 564 794	1 464 331 428	8.08	3.4%	24.6%
G3_circuit	1 585 478	92 706 938	9.27	1.5%	1.7%

Table 5: Comparison of MORSy quality and run time with parameters for minimizing run time of reordering (time-priority), with parameters for maximizing quality (quality-priority). *N* is the number of matrix rows. *Factor NZ, MORSy* is the number of matrix factor non-zero elements after "time-priority" reordering. *Reordering time, MORSy* - " time-priority" reordering run time. *Factor increase* is an increase in the size of the matrix factor in comparison with the "quality-priority" configuration. All times are in seconds.

MORSy run time reduction by 1.06 to 1.72 times allows obtaining the ordering that gives at most 10% of excess non-zero factor elements in comparison with the best MORSy results. For 3 test matrices MORSy produced orderings with 1-3% better factor fill-in compared with METIS, for 8 matrices factor fill-in is 0.5-5% worse, and for 3 matrices it is 12-18% worse. However, for a half of the test matrices MORSy works 1.18-1.99 times faster than METIS (1.44 times faster on average). With other matrices the difference is by 1.01 - 1.61 times (1.29 times on average). In comparison with Scotch, orderings made by MORSy with "time-quality" parameters provide a 0.2-35% better fill-in for 13 matrices (12% on average), for one matrix it is 4% worse. Thus, for 12 matrices run time is worse than with Scotch by 2-62% (25% on average). For the rest two matrices the difference is 13% and 23%.

Figure 3 shows the quality of orderings produced by METIS, Scotch and MORSy with "quality-priority" and "time-priority" parameter configurations. Figure 4 presents the time for obtaining these orderings. It has been shown that there is a parameter configuration that provides orderings with a better fill-in, than obtained by METIS, during a possibly longer period of time, and orderings with a not critically worse fill-in, during shorter periods of time for all test matrices. In comparison with Scotch, orderings received using MORSy with various parameter settings are of a better quality and, for a half of the tested matrices, are less time-consuming.

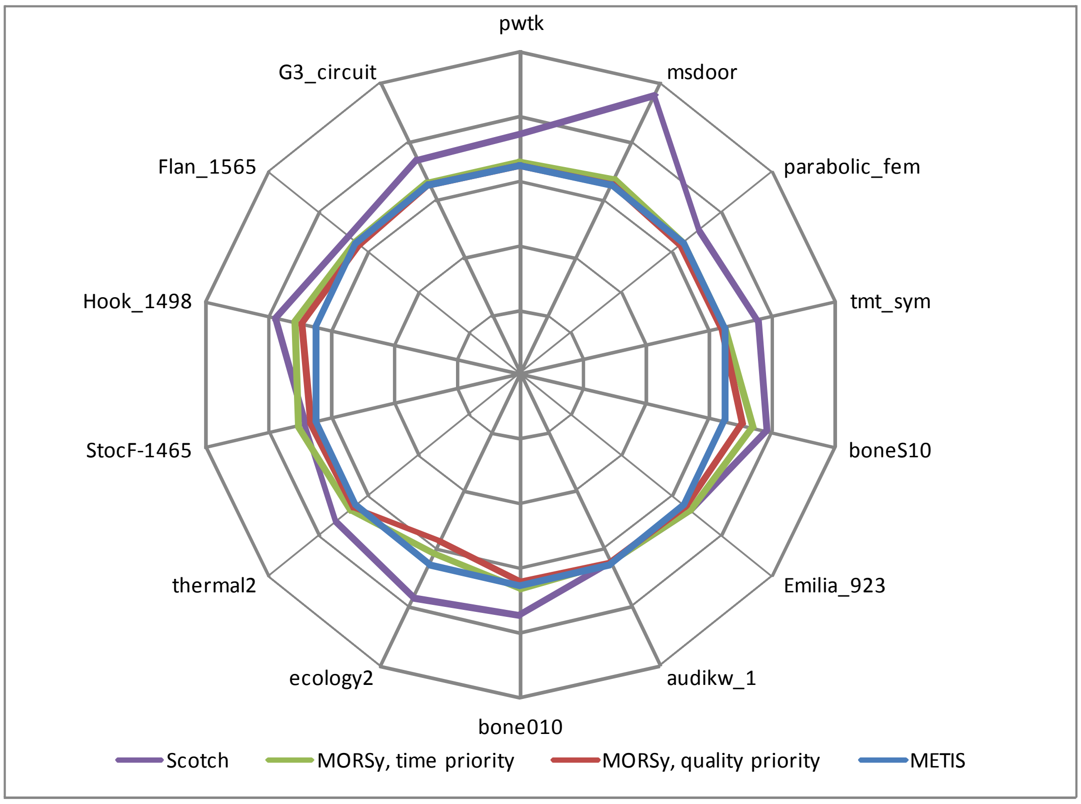

Figure 3: Comparison of number of factor non-zero elements received using MORSy, METIS, Scotch. Results of MORSy and Scotch are given with relation to the number of non-zero elements of the factor received using METIS.

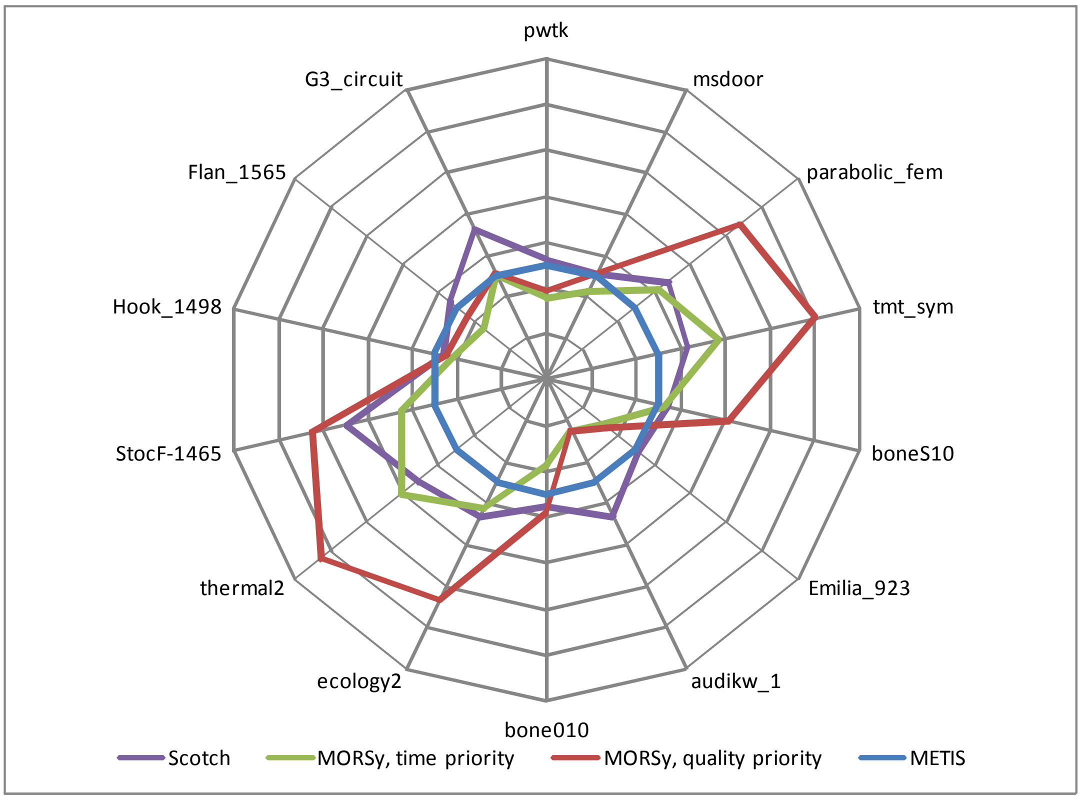

Figure 4: The run time of MORSy, METIS, Scotch. Time of MORSy and Scotch is shown in relation to METIS time.

5 CONCLUSIONS AND FUTURE WORK

We have presented MORSy – a new reordering tool for reducing sparse matrix fill-in based on the multilevel nested dissection algorithm with modifications for vertex separator at all steps of the multilevel scheme. Our experiments demonstrate that MORSy is competitive with the widely used open source ordering libraries METIS and Scotch. MORSy is cross-platform and is publically available [26]. The software is used in the High Performance Computing Center of the State University of Nizhny Novgorod [27] for solving sparse systems of linear equations in the process of finite element simulation of heart activity.

We plan to improve MORSy performance with quality-priority settings to achieve METIS and Scotch performance on the full set of test matrices. The main line of future research is to develop a parallel version of MORSy for shared-memory systems. While there are a number of successful implementations for distributed memory systems (ParMetis [28], PT-Scotch [29]), developing parallel reordering tools for shared memory systems is still an open question. The recent reports presented parallel versions of multilevel graph partitioning algorithms by Scotch [30] and METIS [31] which are designed to the relative problem. While multicore and manycore systems is widely used today, developing parallel reordering algorithms that will combine high quality of orderings with efficient use of computational resources of modern shared-memory systems, are of great importance.

ACKNOWLEGEMENTS

The study was partially supported by the RFBR, research project No. 14-01-3145514.

REFERENCES

[1] T. A. Davis, Direct methods for sparse linear systems. Vol. 2. Siam, 2006.

[2] Y. Saad, Iterative methods for sparse linear systems. Siam, Philadelphia, 2003.

[3] M. Yannakakis, Computing the minimum fill-in is NP-complete. *SIAM J. on Algebraic and Discrete Methods*, **2(1)**, 77–79, 1981.

[4] W. Tinney, J. Walker, Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proceedings of the IEEE*, **55(11)**, 1801–1809, 1967.

[5] J. W. H. Liu, Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Software*, **11(2)**, 141–153, 1985.

[6] P. R. Amestoy, T. Davis, I. Duff, An approximate minimum degree ordering algorithm. *SIAM. J. on Matrix Anal. Appl.*, **17(4)**, 886–905, 1996.

[7] T. A. Davis, J. R. Gilbert, S. I. Larimore, E. G. Ng, A column approximate minimum degree ordering algorithm. ACM Trans. Math. Software, **30(3)**, 353–376, 2004.

[8] A. George, Nested dissection of a regular finite element mesh. *SIAM J. on Numerical Analysis*, **10(2)**, 345–363.

[9] R. J. Lipton, D. J. Rose, R. E. Tarjan, Generalized nested dissection. *SIAM J. on Numerical Analysis*, **16(2)**, 346–358, 1979.

[10] A. George, J. W. H. Liu, An automatic nested dissection algorithm for irregular finite element problems, *SIAM J. on Numerical Analysis*, **15(5)**, 1053–1069, 1978.

[11] T. N. Bui, C. Jones, A Heuristic for Reducing Fill-In in Sparse Matrix Factorization. *PPSC*, 445-452, 1993.

[12] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. on Scientific Computing*, **20(1)**, 359–392, 1998.

[13] B. Hendrickson, R. Leland, A multilevel algorithm for partitioning graphs. *Technical Report SAND93-1301*, Sandia National Laboratories, 1993.

[14] B. Hendrickson, E. Rothberg, Improving the runtime and quality of nested dissection ordering. *SIAM J. on Scientific Computing*, **20**, 468–489, 1999.

[15] B. W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, **29**, 291–307, 1970.

[16] C. M. Fiduccia, R. M. Mattheyses, A linear time heuristic for improving network partitions. *Proceedings 19th IEEE Design Automation Conference*, 175–181, 1982.

[17] G. Karipis, METIS. A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices. Version 5.0. *Technical report*, University of Minnesota, Department of Computer Science and Engineering, 2011. http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/manual.pdf

[18] F. Pellegrini, Scotch and libScotch 6.0 User's Guide. *Technical Report LaBRI*, 2012. https://gforge.inria.fr/docman/view.php/248/8260/scotch_user6.0.pdf

[19] Intel Math Kernel Library Reference Manual.
http://software.intel.com/sites/products/documentation/hpc/mkl/mklman.pdf.

[20] X. S. Li, J. W. Demmel, J. R. Gilbert, L. Grigori, M. Shao, I. Yamazaki, SuperLU Users' guide. *Technical report LBNL-44289*, 2011.
http://crd-legacy.lbl.gov/~xiaoye/SuperLU/superlu_ug.pdf

[21] MUltifrontal Massively Parallel Solver (MUMPS 4.10.0) User's guide. *Technical report ENSEEINT-IRIT*, 2011. http://mumps.enseeiht.fr/doc/userguide_4.10.0.pdf

[22] T. A. Davis, User Guide for CHOLMOD: a sparse Cholesky factorization and modification package. Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL, USA, 2008.

[23] A. George, J. W. H. Liu, Computer solution of large sparse positive definite systems. Prentice-Hall, Englewood Cliffs, New York, 1981.

[24] C. Aschcraft, J. W. H. Liu, A partition improvement algorithm for generalized nested dissection. *Boeing Computer Services*, Seattle, WA, Tech. Rep. BCSTECH-94-020, 1994.

[25] The University of Florida Sparse Matrix Collection:
http://cise.ufl.edu/research/sparse/matrices/

[26] MORSy – Sparse Matrix Ordering Software for reducing fill-in: http://hpc-education.unn.ru/research/overview/sparse-algebra/morsy.

[27] S. Bastrakov, I. Meyerov, V. Gergel et al. High Performance Computing in Biomedical Applications. *Procedia Computer Science*, **18**, 10–19, 2013.

[28] G. Karypis, V. Kumar, ParMETIS: Parallel graph partitioning and sparse matrix ordering library. *Technical Report TR 97-060*, Department of Computer Science, University of Minnesota, 1997.

[29] C. Chevalier, F. Pellegrini, PT-Scotch: A tool for efficient parallel graph ordering. *Parallel Computing*, **34(6)**, 318–331, 2008.

[30] F. Pellegrini, Shared memory parallel algorithms in Scotch 6. *MUMPS Users Group Meeting*, May 29th-30th, EDF, Clamart, France, 2013. http://graal.ens-lyon.fr/MUMPS/doc/ud_2013/Pellegrini.pdf.

[31] D. LaSalle, Karypis G., *Multi-threaded graph partitioning. Parallel & Distributed Processing (IPDPS)*, 2013 IEEE 27th International Symposium on IEEE, 2013.