Sparse Matrix Ordering with SCOTCH*

François Pellegrini and Jean Roman

LaBRI, URA CNRS 1304
Université Bordeaux I
351, cours de la Libération, 33405 TALENCE, FRANCE
{pelegrin|roman}@labri.u-bordeaux.fr

Abstract. Finding good orderings is a critical issue for the efficient factorization of sparse symmetric matrices, both in terms of space usage and solution time. Several ordering techniques have been proposed, among which nested dissection is gaining increasing popularity, due to its suitability for parallel solving.
This paper presents the sparse matrix ordering capabilities of SCOTCH, a software package for static mapping, graph partitioning, and sparse matrix ordering. SCOTCH uses nested dissection ordering, and computes small vertex separators from edge separators by computing minimum covers on the bipartite graphs associated with the edge-cuts.
We give brief descriptions of our implementation of nested dissection and of the separation and ordering methods that it uses, and compare its performance to other ordering programs.

1 Introduction

Many scientific and engineering problems can be modeled by sparse linear systems, which are solved either by iterative or direct methods. Direct methods are popular because of their generality and robustness, especially in some application fields such as linear programming and structural mechanics where iterative methods are not applicable. However, to achieve efficiency with direct methods, one must minimize the fill-in induced by factorization. This fill-in is a direct consequence of the order in which the unknowns of the linear system are numbered, and its effects are critical both in terms of memory and computation costs.

An efficient way to compute fill reducing orderings of symmetric sparse matrices is to use recursive nested dissection [9]. It amounts to computing a vertex set S that separates the graph into two parts A and B, ordering S last, and proceeding recursively on parts A and B until their sizes become smaller than some threshold value. This ordering guarantees that no non-zero term can appear in the factorization process between unknowns of A and unknowns of B.

The main issue of the nested dissection ordering algorithm is thus to find small vertex separators that balance the remaining subgraphs as evenly as possible. Most often, vertex separators are computed from edge separators [23, and

* This work was supported by the French GDR PRS

included references] by minimum cover techniques [5, 13] or heuristics [17], but other techniques such as spectral vertex partitioning have also been used [24]. Provided that good vertex separators are found, the nested dissection algorithm produces orderings which, both in terms of fill-in and operation count, compare very favorably [11, 14] to the ones obtained with the multiple minimum degree algorithm [18]. Moreover, the elimination trees induced by nested dissection are broader, shorter, and better balanced than the ones issued from multiple minimum degree, and therefore exhibit much more concurrency in the context of parallel Cholesky factorization [2, 7, 8, 10, 28, and included references].

Scotch is a project carried out at the *Laboratoire Bordelais de Recherche en Informatique* (LaBRI) of the Université Bordeaux I, by the ALiENor (ALgorithmics and ENvironments for parallel computing) team. Its goal is to study the applications of graph theory to scientific computing, using a "divide and conquer" approach. It focused first on static mapping, and has resulted in the development of the Dual Recursive Bipartitioning (or DRB) mapping algorithm and in the study of several graph bipartitioning heuristics [19], all of which have been implemented in the Scotch software package [22]. We are now focusing on the computation of high-quality vertex separators for nested dissection ordering, by extending the work that has been done on graph (edge-)partitioning in the context of static mapping.

This paper presents the results that have been obtained to date on ordering within the Scotch project. The rest of the paper is organized as follows. Section 2 outlines the most important aspects of our implementation of nested dissection, section 3 describes the Scotch 3.2 software package, and section 4 compares its performance to a multiple minimum degree algorithm and to O-MeTiS 2.0 [15], a nested dissection ordering software developed at the University of Minnesota. Then follows the conclusion.

2 Nested Dissection with Scotch

Our ordering program implements the nested dissection ordering algorithm. As did several other authors [5, 15], we compute the vertex separator of a graph by first computing an edge separator, and then turning it into a vertex separator by using the method proposed by Pothen and Fang [23]. This method requires the computation of maximal matchings in the bipartite graphs associated with the edge cuts, which are built using Duff's variant [5] of the Hopcroft and Karp algorithm [13]. The graph partitioning algorithms that we use make up the core of our static mapping and graph partitioning software [21]. These are graph heuristics only; graph geometry is never used, even when it is available. Several graph bipartitioning methods have been implemented to date, including greedy algorithms to compute initial bipartitions and an improved version of the Fiduccia–Mattheyses heuristic [6], all of which can be used in a multilevel manner. The multi-level approach, which has been used by several authors for graph partitioning [3, 12, 15, 20], dramatically improves the performance of the Fiduccia–Mattheyses algorithm, since it makes possible for it to account for topological structures of the graph that would otherwise be of too high a level for it to encompass in its local optimization process. As for our mapping program, it is possible to apply several different graph partitioning methods in sequence, each one starting from the result of the previous one, and to select the methods with respect to the characteristics of the subgraphs, leading to the definition of *vertex partitioning strategies*.

The ordering program is completely parametrized by two strategies: the vertex separation strategy, which defines the partitioning methods to apply to build the separator tree, according to parameters of the subgraphs to separate (*e.g.* size[2] and average degree), and the ordering strategy. The vertex separation strategy enables the user to take advantage of all of the graph partitioning routines that have been developed in the earlier stages of the Scotch project. Internal ordering strategies for the separators are relevant in the case of sequential or parallel [10, 25, 26, 27] block solving, to select ordering algorithms that minimize the number of extra-diagonal blocks [4], thus allowing for efficient use of BLAS3 primitives, and to reduce inter-processor communication.

3 The Scotch Software Package

Scotch 3.2 is a software package for static mapping, graph partitioning, and sparse matrix ordering, which embodies the algorithms developed within the Scotch project. Apart from the static mapper and the sparse matrix orderer themselves, the Scotch package contains programs to build and test graphs, compute decompositions of target architectures for mapping, and visualize mapping and ordering results. Advanced command-line interface and vertex labeling capabilities make them easy to interface with other programs.

The Scotch 3.2 academic distribution may be obtained from the Scotch WWW page at `http://www.labri.u-bordeaux.fr/~pelegrin/scotch/`, or by `ftp` at `ftp.u-bordeaux.fr` in directory `/pub/Local/Info/Software/Scotch`. It is labeled `scotch_3.2A.tar.gz`. It contains the executables for several machines and operating systems, along with documentation and sample files.

4 Performance evaluations

The quality of orderings is evaluated with respect to several criteria. The first one, NNZ, is the number of non-zero terms of the factored reordered matrix to store. The second one, OPC, is the operation count, that is the number of arithmetic operations required to factor the matrix. To comply with existing RISC processor technology, the operation count that we consider in this paper accounts

[2] This parameter allows the user to end the recursion, by selecting a void separation method when graph size becomes too small.

for all operations (additions, subtractions, multiplications, divisions) required by Cholesky factorization, except square roots; it is equal to $\sum_c n_c^2$, where n_c is the number of non-zeros of column c of the factored matrix, diagonal included. A third criterion for quality is the shape of the elimination tree; concurrency in parallel solving is all the higher as the elimination tree is broad and short. To measure its quality, several parameters can be defined: $h_{\min}$, $h_{\max}$, and h_{avg} denote the minimum, maximum, and average heights of the tree[3], respectively, and h_{dlt} is the variance, expressed as a percentage of h_{avg}. Since small separators result in small chains in the elimination tree, h_{avg} should also indirectly reflect the quality of separators.

Name	Vertices	Edges	Description
144	144649	1074393	3D finite element mesh
3ELT	4720	13722	2D finite element mesh
598A	110971	741934	3D finite element mesh
AUTO	448695	3314611	3D finite element mesh
BCSSTK29	13992	302748	3D stiffness matrix
BCSSTK30	28924	1007284	3D stiffness matrix
BCSSTK31	35588	572914	3D stiffness matrix
BCSSTK32	44609	985046	3D stiffness matrix
BODY	45087	163734	3D finite element mesh
BRACK2	62631	366559	3D finite element mesh
BUMP	9800	28989	2D finite element mesh
M14B	214765	3358036	3D finite element mesh
PWT	36519	144793	3D finite element mesh
ROTOR	99617	662431	3D finite element mesh
TOOTH	78136	452591	3D finite element mesh

Table 1. Various matrices used for our tests.

The first set of experiments that we have carried out regards the incompleteness of nested dissection. We have run our orderer on several test graphs (see table 1) with various upper bounds on the subgraph size under which recursive separation stops. The ordering strategy used by Scotch consists in doing simple (consecutive) ordering of the separator vertices, and ordering the leaves of the separation tree by a multiple minimum degree algorithm [18]. Selected threshold values range from 5, below which nested dissection can be considered as complete, to 200, above which fill-in becomes much too important. Results for graph *598A* are shown in table 2.

The first phenomenon to notice is the impact of the size threshold on fill-in.

[3] We do not consider as leaves the disconnected vertices that are present in some meshes, since they do not participate in the solving process.

	Size threshold					
	10	20	30	50	100	200
NNZ	2.74772e+07	2.76578e+07	2.76858e+07	2.82091e+07	2.78283e+07	2.78870e+07
OPC	2.06490e+10	2.10677e+10	2.08486e+10	2.22195e+10	2.11645e+10	2.11594e+10
h_{nbr}	11620	10723	10353	10267	10901	11815
h_{min}	2595	2609	2600	2618	2607	2619
h_{max}	3229	3302	3328	3303	3222	3386
h_{avg}	2868.43	2886.24	2877.24	2924.06	2890.01	2882.89
h_{dlt}	4.14	4.59	4.77	4.66	4.70	4.31

Table 2. Ordering results for graph *598A* with different threshold values.

As could be expected, the smaller the threshold is, the smaller NNZ is. However, for higher threshold values, the multiple minimum degree algorithm compensates for the increase of NNZ due to the earlier termination of the nested dissection process. As nested dissection is often more efficient than multiple minimum degree, NNZ eventually increases when the threshold gets higher.

Another interesting point regards OPC, whose minimum value is not achieved for complete nested dissection, but rather for threshold values of a few tens of vertices. This is due to the distribution of the values of n_c at the last stages of the nested dissection process: compared to the fill-in generated by multiple minimum degree ordering, complete nested dissection leads to more unbalanced distributions whose largest terms, once squared, dominate the sums.

From the above experiments, we have chosen a default size threshold value of 30, for which OPC is often minimal, and fill-in is no more than one percent above the value that we obtain with complete nested dissection. Considering what we said about the behavior of NNZ and OPC, one could think of using higher threshold values, but doing so would result in a larger dispersion of the non zeroes over the sub-matrices, thus reducing the efficiency of block computations.

O-MeTiS is a sparse matrix ordering software which is part of the MeTiS package [15] developed by Karypis and Kumar at the University of Minnesota. O-MeTiS performs incomplete nested dissection, with a size threshold value of 50, and does multiple minimum degree ordering on the leaves of the separation tree. The orderings computed by O-MeTiS have been proven more efficient on average than the ones computed by the multiple minimum degree ordering for a wide variety of graphs [11, 14].

Table 3 summarizes the values of NNZ and OPC obtained by Scotch, O-MeTiS[4], and a multiple minimum degree algorithm (MMD), for all of our test graphs. All the experiments were run on a 190MHz R10000-based SGI Onyx

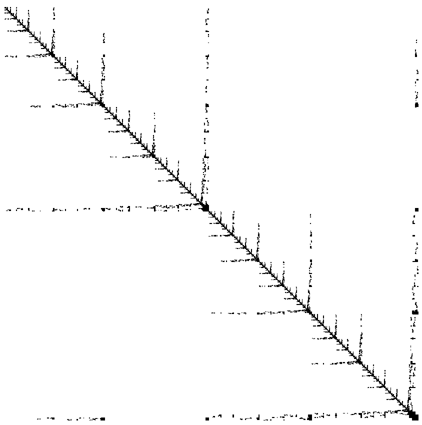
[4] The OPC that we report for O-MeTiS differs from the one output by O-MeTiS itself, which is $\sum_c n_c^2 - n_c + 1$.

machine with 128 Mb of main memory.

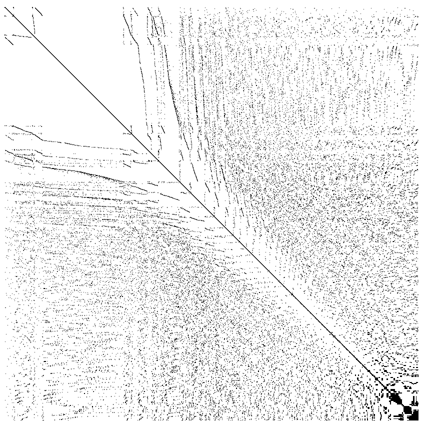
SCOTCH 3.2 outperforms O-METIS 2.0 in 80 % of the runs. On average, its NNZ is smaller by 2%, and its OPC by 4%. These figures, which may seem small, amount, for large graphs, to gains of millions of terms and billions of operations. For instance, for graph *M14B*, the gain over O-METIS is of about 120 thousand terms and 1 billion operations. As far as we can see, this is primarily due to the quality of the partitions computed by SCOTCH, which are consistently better than the ones computed by the graph partitioning routines of P-METIS [20]. At the time being, SCOTCH is slower than O-METIS by a factor 1.5 to 2; careful rewriting of critical portions of code should reduce this difference.

These experiments confirm the efficiency of state-of-the-art nested dissection methods compared to multiple minimum degree algorithms. SCOTCH 3.2 outperforms the MMD algorithm in about 70 % of the runs. On average, its NNZ is smaller by 19%, and its OPC by 47%. For graph *AUTO*, the gains for NNZ and OPC are of 53 % and 77 % respectively, which represents savings of about 274 million terms and 1.9 trillion operations.

The main advantage of MMD resides in its speed, which is 1.5 to 7 times that of current nested dissection algorithms. However, the elimination trees generated by nested dissection are far better balanced than the ones computed by MMD (the good balance of elimination trees for nested dissection can be inferred from the pattern of the factored matrices; see for instance figure 1.a), and therefore are much more suitable for parallel Cholesky factorization [10, 26, 27]. Moreover, the MMD algorithm is inherently sequential, so nested dissection seems much more promising for parallelization [16].



a. SCOTCH 3.2 (S+MMD).



b. MMD.

Fig. 1. Patterns of the factored matrices built from orderings of graph *BUMP* computed by SCOTCH 3.2 and the MMD algorithms.

Graph	Scotch 3.2			O-MeTiS 2.0		
	NNZ	OPC	Time	NNZ	OPC	Time
144	**5.04302e+07**	**6.30637e+10**	54.18	5.14774e+07	6.63160e+10	32.07
3ELT	**9.34520e+04**	**2.91077e+06**	0.44	9.78490e+04	3.43192e+06	0.27
598A	2.80753e+07	2.15109e+10	32.85	**2.80390e+07**	**2.10550e+10**	22.69
AUTO	**2.42459e+08**	**5.45092e+11**	673.78	–	–	–
BCSSTK29	**1.95601e+06**	**4.98316e+08**	5.05	2.00891e+06	5.22292e+08	2.88
BCSSTK30	**4.70394e+06**	**1.35818e+09**	16.79	4.91682e+06	1.48445e+09	10.26
BCSSTK31	**5.22795e+06**	1.84196e+09	12.64	5.27408e+06	**1.72527e+09**	7.73
BCSSTK32	**6.31493e+06**	**1.69557e+09**	20.12	6.93198e+06	2.23416e+09	12.20
BODY	1.21614e+06	1.08353e+08	8.45	**1.10336e+06**	**7.92548e+07**	4.44
BRACK2	7.38112e+06	**2.54711e+09**	15.07	**7.20380e+06**	2.56784e+09	9.19
BUMP	**2.67353e+05**	**1.48797e+07**	1.02	2.73569e+05	1.55700e+07	0.63
M14B	**6.77465e+07**	**7.08264e+10**	102.05	6.89223e+07	7.18790e+10	52.67
PWT	**1.43140e+06**	**1.22120e+08**	6.05	1.53126e+06	1.40441e+08	3.76
ROTOR	**1.76431e+07**	**1.11101e+10**	28.88	1.81358e+07	1.16120e+10	18.36
TOOTH	**1.31559e+07**	**9.02613e+09**	19.48	1.32536e+07	9.44619e+09	12.34

Graph	MMD		
	NNZ	OPC	Time
144	9.81030e+07	2.62934e+11	28.30
3ELT	**8.98200e+04**	**2.84402e+06**	0.06
598A	4.63319e+07	6.43375e+10	19.96
AUTO	5.16945e+08	2.41920e+12	139.14
BCSSTK29	**1.69480e+06**	**3.930592+08**	0.76
BCSSTK30	**3.84344e+06**	**9.28353e+08**	1.58
BCSSTK31	5.30825e+06	2.55099e+09	1.96
BCSSTK32	**5.24635e+06**	**1.10873e+09**	2.14
BODY	**1.00908e+06**	8.93073e+07	1.25
BRACK2	7.30755e+06	2.99122e+09	3.90
BUMP	2.81482e+05	1.76288e+07	0.12
M14B	1.22359e+08	2.56226e+11	44.07
PWT	1.59060e+06	1.75944e+08	0.70
ROTOR	2.60982e+07	2.86029e+10	11.27
TOOTH	1.56226e+07	1.65121e+10	5.06

Table 3. Orderings produced by Scotch 3.2, O-MeTiS 2.0, and a multiple minimum degree (MMD) algorithm. Bold values in the Scotch and MeTiS sub-arrays indicate best results among these two sub-arrays. Bold values in the MMD sub-array indicate best results over all sub-arrays. Dashes indicate abnormal termination due to lack of memory.

5 Conclusion

In this paper, we have presented the sparse matrix ordering capabilities of Scotch 3.2, a software package for static mapping and graph partitioning. The experiments that have been carried out to date show its great efficiency in terms of fill-in and balance of the elimination trees.

The vertex-separation capabilities of our software have already found applications other than sparse matrix ordering. Scotch is currently used to compute separators and distribute workload for a parallel implementation of a graph coloring algorithm being developed at the LaBRI.

Work in progress includes the development of new separation and ordering strategies. We are planning to implement the Approximate Minimum Degree algorithm of Amestoy, Davis, and Duff [1] to compute the orderings of the leaves of the separation tree. Ordering strategies that minimize the number of extra-diagonal blocks are also being considered, to provide good block orderings to a parallel sparse block Cholesky solver, also under development at the LaBRI.

References

1. P. Amestoy, T. Davis, and I. Duff. An approximate minimum degree ordering algorithm. Technical Report RT/APO/95/5, ENSEEIHT-IRIT, 1995. To appear in SIAM Journal of Matrix Analysis and Applications.
2. C. Ashcraft, S. Eisenstat, J. W.-H. Liu, and A. Sherman. A comparison of three column based distributed sparse factorization schemes. In *Proc. Fifth SIAM Conf. on Parallel Processing for Scientific Computing*, 1991.
3. S. T. Barnard and H. D. Simon. A fast multilevel implementation of recursive spectral bisection for partitioning unstructured problems. *Concurrency: Practice and Experience*, 6(2):101–117, 1994.
4. P. Charrier and J. Roman. Algorithmique et calculs de complexité pour un solveur de type dissections emboîtées. *Numerische Mathematik*, 55:463–476, 1989.
5. I. Duff. On algorithms for obtaining a maximum transversal. *ACM Trans. Math. Software*, 7(3):315–330, September 1981.
6. C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In *Proc. 19th Design Autom. Conf.*, pages 175–181. IEEE, 1982.
7. G. A. Geist and E. G.-Y. Ng. Task scheduling for parallel sparse Cholesky factorization. *International Journal of Parallel Programming*, 18(4):291–314, 1989.
8. A. George, M. T. Heath, J. W.-H. Liu, and E. G.-Y. Ng. Sparse Cholesky factorization on a local memory multiprocessor. *SIAM Journal on Scientific and Statistical Computing*, 9:327–340, 1988.
9. J. A. George and J. W.-H. Liu. *Computer solution of large sparse positive definite systems*. Prentice Hall, 1981.
10. A. Gupta, G. Karypis, and V. Kumar. Highly scalable parallel algorithms for sparse matrix factorization. TR 94-063, University of Minnesota, 1994. To appear in *IEEE Trans. on Parallel and Distributed Systems*, 1997.
11. A. Gupta, G. Karypis, and V. Kumar. Scalable parallel algorithms for sparse linear systems. In *Proc. Stratagem'96, Sophia-Antipolis*, pages 97–110. INRIA, July 1996.
12. B. Hendrickson and R. Leland. The chaco user's guide – version 2.0. Technical Report SAND94–2692, Sandia National Laboratories, 1994.

13. J. Hopcroft and R. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal of Computing*, 2(4):225–231, December 1973.
14. G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. TR 95-035, University of Minnesota, June 1995.
15. G. Karypis and V. Kumar. METIS – *Unstructured Graph Partitioning and Sparse Matrix Ordering System – Version 2.0.* University of Minnesota, June 1995.
16. G. Karypis and V. Kumar. Parallel multilevel k-way partitioning scheme for irregular graphs. TR 96-036, University of Minnesota, 1996.
17. C. Leiserson and J. Lewis. Orderings for parallel sparse symmetric factorization. In *Third SIAM Conference on Parallel Processing for Scientific Computing, Tromsø.* SIAM, 1987.
18. J. W.-H. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Software*, 11(2):141–153, 1985.
19. F. Pellegrini. Static mapping by dual recursive bipartitioning of process and architecture graphs. In *Proc. SHPCC'94, Knoxville*, pages 486–493. IEEE, May 1994.
20. F. Pellegrini. Application of graph partitioning techniques to static mapping and domain decomposition. In *ETPSC 3, Faverges-de-la-Tour*, August 1996. To appear in a special issue of Parallel Computing.
21. F. Pellegrini. SCOTCH 3.2 User's guide. Technical Report, LaBRI, Université Bordeaux I, April 1997. Available at URL `http://www.labri.u-bordeaux.fr/~pelegrin/papers/scotch_user3.2.ps.gz`.
22. F. Pellegrini and J. Roman. SCOTCH: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs. In *Proceedings of HPCN'96, Brussels,* LNCS 1067, pages 493–498, April 1996.
23. A. Pothen and C.-J. Fan. Computing the block triangular form of a sparse matrix. *ACM Trans. Math. Software*, 16(4):303–324, December 1990.
24. A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM Journal of Matrix Analysis*, 11(3):430–452, July 1990.
25. E. Rothberg. Performance of panel and block approaches to sparse Cholesky factorization on the iPSC/860 and Paragon multicomputers. In *Proceedings of SH-PCC'94, Knoxville*, pages 324–333. IEEE, May 1994.
26. E. Rothberg and A. Gupta. An efficient block-oriented approach to parallel sparse Cholesky factorization. In *Supercomputing'93 Proceedings.* IEEE, 1993.
27. E. Rothberg and R. Schreiber. Improved load distribution in parallel sparse Cholesky factorization. In *Supercomputing'94 Proceedings.* IEEE, 1994.
28. R. Schreiber. Scalability of sparse direct solvers. Technical Report TR 92.13, RIACS, NASA Ames Research Center, May 1992.