Hybrid Sparse Linear Solutions with Substituted Factorization

Joshua Dennis Booth and Padma Raghavan
{jdb5132, raghavan}@cse.psu.edu

Computer Science and Engineering,
The Pennsylvania State University,
University Park PA 16802, USA

Abstract. We develop a computationally less expensive alternative to the direct solution of a large sparse symmetric positive definite system arising from the numerical solution of elliptic partial differential equation models. Our method, *substituted factorization*, replaces the computationally expensive factorization of certain dense submatrices that arise in the course of direct solution with sparse Cholesky factorization with one or more solutions of triangular systems using substitution. These substitutions fit into the tree-structure commonly used by parallel sparse Cholesky, and reduce the initial factorization cost at the expense of a slight increased cost in solving for a right-hand side vector. Our analysis shows that substituted factorization reduces the number of floating-point operations for the model $k \times k$ 5-point finite-difference problem by 10% and empirical test show execution time reduction on average of 24.4%. On a test suite of three-dimensional problems we observe execution time reduction as high as 51.7% and 43.1% on average.

1 Introduction

The solution of sparse linear systems arising from finite discretization of second-order elliptic partial differential equations (PDEs) dominates the execution time of scientific codes [12]. These systems of linear equations may be solved by two broad categories of solvers, namely direct or iterative [6, 7, 9, 15]. In application, a third hybrid category may appear, which tries to balance trade-offs between direct and iterative solvers [3, 4]. They make no claim on the number of iterations needed to solve the system, and therefore hybrid methods can be seen as a preconditioned iterative method. We provide a new formulation used as a direct method by making substitutions in place of factorization, and demonstrate how this formulation results in savings over the direct method sparse Cholesky while providing a robust solution.

This paper provides a detailed understanding and analysis of our new *substituted factorization* (*SF*), and demonstrates our method as an alternative direct method formulation for solving systems arising from finite discretization of a second-order elliptic PDE on quasi-uniform grids. Although other methods exist, the choice of the best method depends on the number of solves, robustness, and memory constraints. Direct methods provide a desired robustness and low cost for multiple solves that many iterative solvers cannot provide. However, the high computational cost of factorization in direct methods makes iterative method more desirable unless this cost can be amortized with multiple solves. Our method provides a middle ground in terms of the number of solves needed to amortize factorization cost while still providing the desirable robustness.

2 Background and Related Work

A number of methods exist to solve systems arising from second-order elliptic PDE models. Sparse direct solvers [6, 9] use sparse factorization to compute robust solutions. Iterative solvers, such as Krylov space based Conjugate Gradients (CG) [7, 15], attempt to solve systems faster by iteratively converging to a solution. Many elliptic problems can be efficiently solved using domain decomposition methods (DDM) by splitting the problem into smaller domains. DDM may solve the arising systems or be used to precondition Krylov space iterative methods, and the effectiveness varies with problem and decomposition.

When using an iterative method similar to CG, the number of iterations needed to solve for a single right-hand side vector (RHS) depends on the condition number of the matrix. The condition number of a matrix is defined as: $\delta(A) = \|A\|_2 \|A^{-1}\|_2$ [7]. In particular, the number of iterations needed for CG to converge is $O(\sqrt{\delta(A)})$ [2].

For symmetric systems of equations arising from finite discretization of a second-order elliptic PDE on regular grids, the condition of the matrix is known to be greater than $O(h^{-2})$. Generally, h^{-1} equals $n^{1/2}$ for two-dimensional (2D), and $n^{1/3}$ for three-dimensional (3D) grids. However, the condition number for submatrices used may be better. Mansfield [13] demonstrates the condition number for the Schur complement is bounded by $O(h^{-1})$ for the broad class of linear systems arising from finite discretization of a second-order elliptic PDE on regular grids. *SF* uses this bound to fix the number of iterations for CG.

Hybrid solvers [3, 4] use various combinations of direct and iterative methods. These hybrid solvers partition the matrix into submatrices linked by other submatrices that correspond to separators in the graph representation, and the resulting form may be similar to:

$$\begin{bmatrix} A_{11} & 0 & A_{31}^T \\ 0 & A_{22} & A_{32}^T \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_1 = x_1^s + x_1^d \\ x_2 = x_2^s + x_2^d \\ x_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}. \quad (1)$$

Sparse factorization or incomplete factorization of these submatrices are used to compute a solution vector.

SF is related to our earlier work, Booth, Chatterjee, Raghavan, & Frasca. [3], which uses equation 1, and we refer to this method as DI. In DI, sparse Cholesky is applied to A_{11} and A_{22}, incomplete Cholesky to the submatrix corresponding to the separator, and A_{31} and A_{32} remain unfactored. After factorization, DI applies direct forward/backwards solves to find x_1^d and x_2^d, and uses preconditioned CG to solve for x_3. Finally, DI uses forward/backward solves to find x_1^s and x_2^s. *SF* uses DI's framework, and provides a new formulation by expanding A_{33} and using factorization on A_{31} and A_{32}.

3 Substituted Factorization Formulation

In this section, we present our alternative direct method known as *substituted factorization* (*SF*). For our method, we consider a sparse linear system $Ax = f$, where A is a $n \times n$ sparse symmetric positive definite matrix ($A \in \mathbb{R}^{n\times n}$), $f \in \mathbb{R}^{n\times 1}$, and $x \in \mathbb{R}^{n\times 1}$. We wish to solve for x, and we rely on a graph representation of the sparse matrix A to construct a tree-structure. In this graph representation, each row/column represents a node and an undirected edge exists iff $A(i, j) \neq 0$. Additionally, self edges, i.e., $A(i, i) \neq 0$, are removed.

Nodes in the graph are split into two disconnected sets of nodes (Ω_0 and Ω_1) and a small set of nodes (Λ_0) that separates them. Λ_0 is commonly called a separator and Ω_is the domains. In a tree-structure, each separator is the parent of two domains. These domains can be split recursively. When done recursively, the resulting tree has leaf nodes of disconnected domains and internal parent nodes of separators. This tree-structure may impose a new ordering for the sparse matrix A by numbering the nodes corresponding to separators after those of their children. This ordering is known as nested dissection (ND) [5, 11].

By splitting the graph twice and reordering, we rewrite A and its associated Cholesky factorization as follows:

$$\begin{pmatrix} A_{11} & & A_{31}^T & & & & A_{71}^T \\ & A_{22} & A_{32}^T & & & & A_{72}^T \\ A_{31} & A_{32} & A_{33} & & & & A_{73}^T \\ & & & A_{44} & & A_{64}^T & A_{74}^T \\ & & & & A_{55} & A_{65}^T & A_{75}^T \\ & & & A_{64} & A_{65} & A_{66} & A_{76}^T \\ A_{71} & A_{72} & A_{73} & A_{74} & A_{75} & A_{76} & A_{77} \end{pmatrix} \& \begin{pmatrix} L_{11} & & & & & & \\ & L_{22} & & & & & \\ L_{31} & L_{32} & L_{33} & & & & \\ & & & L_{44} & & & \\ & & & & L_{55} & & \\ & & & L_{64} & L_{65} & L_{66} & \\ L_{71} & L_{72} & L_{73} & L_{74} & L_{75} & L_{76} & L_{77} \end{pmatrix},$$

where $A = LL^T$. We divide the submatrices of the sparse Cholesky into two groups, namely domains and separators.

The domain ($i \in \{1, 2, 4, 5\}$) and their corresponding off-diagonal entries can be written as $A_{ii} = L_{ii}L_{ii}^T$ and $A_{ji} = L_{ji}L_{ii}^T$ where j is an index of a separator such that $j \in \{3, 6, 7\}$. Additionally, the diagonal entries corresponding to our domains will be relatively sparse compared to our separators after factorization. Therefore, we perform sparse Cholesky on both the diagonal and off-diagonal domain submatrices.

The separators ($i \in \{3, 6, 7\}$) submatrices have the following equations

$$A_{73} = L_{71}L_{31}^T + L_{72}L_{32}^T + L_{73}L_{33}^T, \tag{2}$$

$$A_{76} = L_{74}L_{64}^T + L_{75}L_{65}^T + L_{76}L_{66}^T, \tag{3}$$

$$A_{77} = L_{71}L_{71}^T + L_{72}L_{72}^T + L_{73}L_{73}^T + L_{74}L_{74}^T + L_{75}L_{75}^T + L_{76}L_{76}^T + L_{77}L_{77}^T. \tag{4}$$

Because these submatrices become dense during factorization, we avoid computing their factorization. The matrices, $L_{jj}L_{jj}^T$, for the top levels are commonly referred to as the Schur complement, and denote them as $\overline{A_{jj}}$. To avoid factorization, we solve each $\overline{A_{jj}}$ with associated right-hand side using a fixed number of iterations of CG guarantee to converge, thus making our method direct.

Traditional direct methods that implement sparse Cholesky have a forward and backward stage when solving for a given right-hand side (f). During the forward and backward stages, an updated right-hand side must be formed for each of the separating pieces, i.e., $j \in \{3,5,7\}$. These new right-hand sides (denoted $\hat{f}_j$) need updates by all $L_{j,i}$ where i corresponds to all node underneath j in the tree-structure, e.g., $\hat{f}_3 = f_3 - L_{31}L_{11}^{-1}f_1 - L_{32}L_{22}^{-1}f_2$. However, we save operations by not finding the factorization $L_{jj}L_{jj}^T$ and its corresponding off-diagonal entries for the top levels in our tree-structure. Instead, *SF* keeps the following equations from (2) and (3) as:

$$E_{73} = L_{73}L_{33}^T = A_{73} - L_{71}L_{31}^T - L_{72}L_{32}^T, \tag{5}$$

$$E_{76} = L_{76}L_{66}^T = A_{76} - L_{74}L_{64}^T - L_{75}L_{65}^T, \tag{6}$$

and substitutes them when forming the associated right-hand side for $\overline{A}_{77}$.

Using substitution, we can write the solution to the set of linear equations in two sets, namely forward and backward. The forward set is defined as:

$$x_1^f = L_{11}^{-1}f_1, \qquad x_4^f = L_{44}^{-1}f_4, \qquad x_3^f = \overline{A_{33}}^{-1}\hat{f}_3,$$

$$x_2^f = L_{22}^{-1}f_2, \qquad x_5^f = L_{55}^{-1}f_5, \qquad x_6^f = \overline{A_{66}}^{-1}\hat{f}_6,$$

$$x_7 = \overline{A_{77}}^{-1}\hat{f}_7,$$

where

$$\hat{f}_3 = f_3 - L_{31}x_1^f - L_{32}x_2^f,$$

$$\hat{f}_6 = f_6 - L_{64}x_4^f - L_{65}x_5^f,$$

$$\hat{f}_7 = f_7 - L_{71}x_1^f - L_{72}x_2^f - L_{74}x_4^f - L_{75}x_5^f - E_{73}x_3^f - E_{76}x_6^f.$$

Using this forward set, the backward set is defined as:

$$x_1 = L_{11}^{-T}(x_1^f - L_{31}^T x_3 - L_{71}^T x_7), \qquad x_5 = L_{55}^{-T}(x_5^f - L_{65}^T x_6 - L_{75}^T x_7),$$

$$x_2 = L_{22}^{-T}(x_2^f - L_{32}^T x_3 - L_{72}^T x_7), \qquad x_3 = x_3^f - \overline{A_{33}}^{-1}E_{73}^T x_7,$$

$$x_4 = L_{44}^{-T}(x_4^f - L_{64}^T x_6 - L_{74}^T x_7), \qquad x_6 = x_6^f - \overline{A_{66}}^{-1}E_{76}^T x_7.$$

4 Cost Analysis

In this section, we provide the cost analysis in terms of the number of multiplication operations (Ops) needed by *SF*. This analysis focuses on applying *SF* on the model $k \times k$ finite-difference 5-point stencil problem. We will examine the setup cost, i.e., the cost of factorization, and solve cost for *SF* compared to sparse Cholesky with ND ordering (*Chol*). The setup and solve cost are denoted as Γ_{setup}^i and Γ_{solve}^i where i is either *SF* or *Chol*. Additionally, we provide a comparison of the cost to solve multiple RHS, and provide a break even point between the two methods in terms of the number of RHS.

Setup Cost. We first consider the setup cost of *Chol* and *SF*. The setup cost to apply sparse Cholesky to a matrix A approximately equals $1/2\sum_{i=1}^n nnz(i)^2$ where $nnz(i)$ is the number of nonzeros in the ith column of L, such that $A = LL^T$ [6]. When A is reordered with ND, George [5] provides that $\Gamma_{setup}^{Chol} \approx \frac{267}{28}k^3 - 17k^2\log_2 k + \frac{847}{28}n^2 + O(k\log_2 k)$.

SF has the same number of Ops as the previous analysis minus the cost to factor the top level "+" separator which accounts for A_{33}, A_{66}, and A_{77}. The cost to factor the "+" separator is $\sum_{i=k}^{3k/2} i^2 - \frac{1}{2}\sum_{i=1}^{k} i^2 = \frac{23}{24}k^3 + \frac{7}{8}k^2 + \frac{1}{6}k$. Therefore, our setup cost equals:

$$\Gamma_{setup}^{SF} \approx \frac{1441}{168}k^3 - 17k^2 \log_2 k + \frac{235}{8}k^2 + O(k \log_2 k). \quad (7)$$

As our result, setup cost for *SF* has approximately **10**% fewer Ops than *Chol*.
Solve Cost. When solving using *Chol*, the number of Ops is on the order of the number of nonzeros. CG with no precondition requires $O(n^{1.5})$ Ops to solve the model problem [2]. George and Liu [6] state $\Gamma_{solve}^{Chol} = 31/4k^2 \log_2 k + O(k^2)$.

SF performs sparse Cholesky on the leaf nodes and a fixed number of CG steps on the parent nodes. These parent nodes correspond to the Schur complements, and these nodes have a bounded condition number less than that of the whole system. For the model problem, the condition number is $O(h^{-2})$, and the condition number for our Schur complement will be less than $O(h^{-1}) \approx k$ [13]. Additionally, we know that CG will theoretically converge to a solution in $\sqrt{\delta(A)}$ with exact arithmetic [2]. Using these two facts, we set our fixed number of iterations to $\gamma = \beta\sqrt{k}$ where $\beta > 0$ is some unknown constant. For *SF*, the cost for a solve is

$$\Gamma_{solve}^{SF} = \frac{31}{4}k^2 \log_2 k + (c-2)k^2 + 2\beta k^{5/2} \quad (8)$$

where c is the constant from Γ_{solve}^{Chol}.
Number of Solves. Despite slightly more Ops during solving, *SF* has fewer Ops for factorization and solving some m RHS. The m is easily seen to be $\frac{23}{24}k^3 \approx 2\beta k^{5/2} m$, which translates to $m \approx \frac{23}{48\beta}k^{1/2}$.

5 Experiments and Evaluation

In this section, we present numerical experimentation of *SF*. *SF* is constructed using DSCPACK [14] for factorization, and using WSMP [10] for CG. The code is constructed using *C* and *FORTRAN*, and compiled using *gnu 4.7.2* compilers with *O3* optimization. GotoBLAS2 [8] is used by all solvers. We compare Ops to DSCPACK (DSC), the direct symmetric WSMP, and the hybrid solver from Booth et al. [3] (DI). However, we only compare times against DSC and WSMP, since DI is coded in MATLAB.
Test Suites. Our experimentation evaluates *SF* over a collection of 2D and 3D problems. Test matrices are divided into two test suites. *Test suite A* comprises of $k \times k$ 5-point stencil Laplacian problems (*2DL*) of varying size. *Test suite A* evaluates the analysis of the previous section using perfect separators similar to those in the analysis. *Test suite B* contains 3D problems, and uses separators found using a multi-level partitioning scheme from WSMP. *Test suite B* comprises of 3D regular grid Laplacian problems (*3DL*) and stiffness matrices for a brick in 3D using 20 node serendipity elements and the equations of linear elasticity (*Brick*). Brick is generated using *ex10.c* from PETSc [1]. These matrices are found in Table 1 and 2.

Table 1: *Test Suite A*

Matrix	n	nonzeros
2DL300	90,000	448,800
2DL500	250,000	1,248,000
2DL700	490,000	2,447,200

Table 2: *Test Suite B*

Matrix	n	nonzeros
3DL40	64,000	401,896
3DL50	125,000	802,376
Brick20	13,860	1,912,944
Brick30	44,640	6,500,784

Metrics for Evaluation. We evaluate methods in terms of the number of Ops and time to factor and solve (T). When comparing Ops, we will commonly use relative improvement ($ROps(x)$) to compare method x to WSMP, and compare the time for setup and solving m RHS for method x to WSMP as $RTime(x)$, i.e., $ROps(x) = \frac{\text{Ops(WSMP)}-\text{Ops(x)}}{\text{Ops(WSMP)}}$, $RTime(s) = \frac{\text{T(WSMP)}-\text{T(x)}}{\text{T(WSMP)}}$.

Evaluation of Test Suite A: 2D Laplacian problems. We evaluate our cost analysis of the 2D stencil problem in Fig. 1 using 1 and 20 solves. For this evaluation, we assume that β is 1/2 and therefore use the fixed number of $1/2\sqrt{k}$ CG iterations. For all RHS, the solution found with *SF* has error on the same order of magnitude as WSMP. We notice in Fig. 1 (a) that *SF* reduces the number of Ops similar to predicted in the analysis section by approximately 10%. The slight increase of reduced Ops with size comes from Ops reduction during solve. This growing reduction with problem size demonstrates the usefulness of *SF* for large problems, and this is further demonstrated in Fig. 1 (b). Fig. 1 (b) demonstrates the Ops for solving 20 RHS will be better for *SF* on larger problems than *Chol* . Note from our analysis and assumption of β, the number of RHS while costing less is $\approx .96\sqrt{k}$.

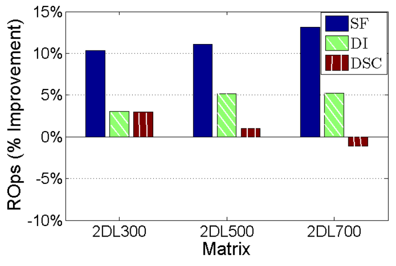

Fig. 1 (a) ROps 1RHS

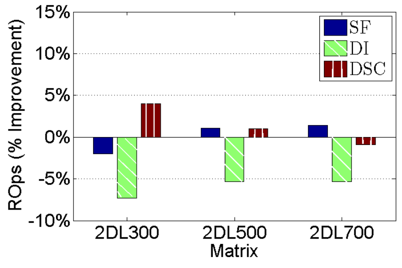

Fig. 1 (b) ROps 20RHS

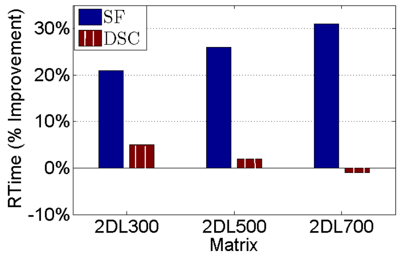

Fig. 2 (a) RTime 1RHS

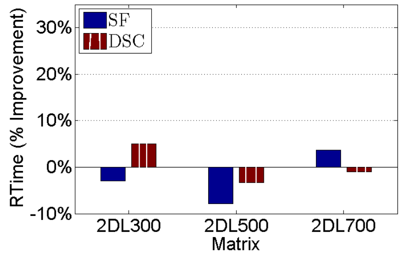

Fig. 2 (b) RTime 20RHS

In further investigation, we examine the execution time for solving 1 and 20 RHS in Fig. 2 (a) and (b). We first notice *SF*'s improvement over other methods when performing one solve and how *SF*'s improvement increases with the size of the problem from 21% to 31%. Even when solving 20 RHS, our method outperforms sparse Cholesky when $k > 500$.

Evaluation of Test Suite B: 3D Laplacian and Brick. Our second set of problems represent a key area, because they are more difficult for iterative methods, e.g., CG, and ND using multi-level partitioning schemes may have larger separators. For this set, the fix number of CG iterations is set to 1000 based on observed condition number, however we allow CG to stop if the relative error of the solution below $O(10^{-12})$. All the solutions have error similar in magnitude to *Chol*.

Figure 3 presents the relative time for *SF* and DSC relative to WSMP for 1 and 50 RHS. We first notice the execution times are better than those from *test suite A*. The execution time is greater than 40% for all matrices solving once. This better performance is in part due to the larger Schur complement in the top levels of our tree. For Brick30, the top level separator dimension is 3960 where the largest 2D top level separator dimension is 700.

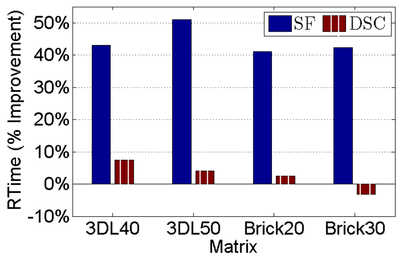

Fig. 3 (a) RTime 1 RHS

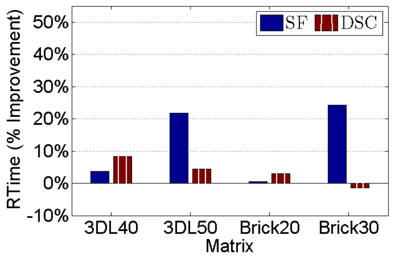

Fig. 3 (b) RTime 50 RHS

Additionally, we observe that 3D problems allow *SF* to solve a greater number of RHS while costing less than sparse Cholesky. In Fig. 3 (b), *SF* solves almost 50 RHS before costing more than sparse Cholesky, and demonstrates a trade-off between the number of solves and the number of nonzeros.

Experimentation with Iterative Solvers. *SF* outperforms DI which is shown to be better for multiple solves than precondition CG in Booth et al. [3].

6 Substituted Factorization Contributions

SF provides two improvements from the work Booth et al. [3]. First, we construct a direct method by bounding the number of iterations and using CG. Second, we provide a new formulation that allows for the factorization of the off-diagonal entries, thus removing the need for both forward/backward solves at multiple steps. Additionally, the off-diagonal entries allow for our method to expand terms related to A_{33} from equation 1 into submatrices that will be solved using our derived substitutions. This expansion reduces Ops and provides better matrix conditioning as observed in the experimental section.

7 Conclusions

Finding the most efficient solutions of sparse linear systems arising from finite discretization of second-order elliptic PDEs is very important to the performance of scientific codes. We have developed a *substituted factorization* (SF) that finds solutions with the robustness of direct methods in less time than a direct solution based on sparse Cholesky factorization for a given number of solves. Specifically, the number of float-point operations during factorization will reduce by 10% for the $k \times k$ finite-difference problem. Additionally, experiments on our test suites demonstrate that execution time may reduce by as much as 48.9% on 3D problems and 31.7% on our 2D problems. *SF* method provides a needed method between sparse Cholesky and CG in terms of the number of solves needed to amortize factorization or preconditioning cost.

Acknowledgment. We would like to thank Anshul Gupta for all his assistance with WSMP that made our experiments possible. Additionally, we acknowledge the support of NSF grants CCF-0830679, CCF-1319448, CNS-1017882.

References

1. Balay, S., Brown, J., Buschelman, K., Eijkhout, V., Gropp, W.D., Kaushik, D., Knepley, M.G., McInnes, L.C., Smith, B.F., Zhang, H.: PETSc users manual. Tech. Rep. ANL-95/11 - Revision 3.4, Argonne National Laboratory (2013)
2. Bern, M., Gilbert, J., Hendrickson, B., Nguyen, N., Toledo, S.: Support-graph preconditioners. SIAM J. Matrix Anal. and Appl. 27(4), 930–951 (2006)
3. Booth, J.D., Chatterjee, A., Raghavan, P., Frasca, M.: A multilevel cholesky conjugate gradients hybrid solver for linear systems with multiple right-hand sides. Procedia Computer Science 4(0), 2307 – 2316 (2011)
4. Gaidamour, J., Henon, P.: A parallel direct/iterative solver based on a schur complement approach. In: Computational Science and Engineering, 2008. CSE '08. 11th IEEE International Conference on. pp. 98–105 (2008)
5. George, A.: Nested dissection of a regular finite element mesh. SIAM J. Numerical Analysis (2), 24–45 (April 1973)
6. George, A., Liu, J.: Computer Solution of Large Sparse Positive Definite Systems. Prentice-Hall, Englewood Cliffs, NJ, USA (1981)
7. Golub, G.H., Van Loan, C.F.: Matrix computations (3rd ed.). Johns Hopkins University Press, Baltimore, MD, USA (1996)
8. Goto, K., van de Geijn, R.A.: High-performance implementation of the level-3 blas. ACM Trans. Math. Softw. 35(1) (2008)
9. Gupta, A., Kumar, V.: A scalable parallel algorithm for sparse cholesky factorization. In: Supercomputing '94., Proceedings. pp. 793–802 (1994)
10. Gupta, A., Joshi, M.: Wsmp: A high-performance shared- and distributed-memory parallel sparse linear equation solver (2001)
11. Heath, M.T., Raghavan, P.: A cartesian parallel nested dissection algorithm (1994)
12. Heroux, M., Raghavan, P., Simon, H.: Parallel Processing for Scientific Computing
13. Mansfield, L.: On the conjugate gradient solution of the schur complement system obtained from domain decomposition. SIAM J. Num. Anal. 27(6), 1612–1620 (1990)
14. Raghavan, P.: DSCPACK. Tech. Rep. CSE-02-004, Penn State (2002)
15. Saad, Y.: Iterative Methods for Sparse Linear Systems. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2nd edn. (2003)