TITLE

ILUPACK

BYLINE

Matthias Bollhöfer
Institut Computational Mathematics
TU Braunschweig
38106 Braunschweig
Germany
m.bollhoefer@tu-bs.de

José I. Aliaga, Alberto F. Martín, Enrique S. Quintana-Ortí
Dpto. de Ingeniería y Ciencia de Computadores
Universitat Jaume I
12.071-Castellón
Spain
{aliaga,martina,quintana}@icc.uji.es

DEFINITION

ILUPACK is the abbreviation for **I**ncomplete **LU** factorization **PACK**age. It is a software library for the iterative solution of large sparse linear systems. It is written in FORTRAN 77 and C and available at `http://ilupack.tu-bs.de`. The package implements a multilevel incomplete factorization approach (multilevel ILU) based on a special permutation strategy called "inverse-based pivoting" combined with Krylov subspace iteration methods. Its main use consists of application problems such as linear systems arising from partial differential equations (PDEs). ILUPACK supports single and double precision arithmetic for real and complex numbers. Among the structured matrix classes that are supported by individual drivers are symmetric and/or Hermitian matrices that may or may not be positive definite and general square matrices. An interface to MATLAB (via MEX) is available. The main drivers can be called from C, C++ and FORTRAN.

DISCUSSION

Introduction

Large sparse linear systems arise in many application areas such as partial differential equations, quantum physics or problems from circuit and device simulation. They all share the same central task which consists of efficiently solving large sparse systems of equations. For a large class of application problems, sparse direct solvers have proven to be extremely efficient. However, the enormous size of the underlying applications arising in 3-D PDEs or the large number of devices in integrated circuits currently requires fast and efficient iterative solution techniques, and this need will be exacerbated as the dimension of these systems increases. This in turn demands for alternative approaches and, often, approximate factorization techniques, combined with iterative methods based on Krylov subspaces, reflect an attractive alternative for these kind of application problems. A comprehensive overview over iterative methods can be found in [15].

The ILUPACK software is mainly built on incomplete factorization methods (ILUs) applied to the system matrix in conjunction with Krylov subspace methods. The ILUPACK hallmark is the so-called *inverse-based approach*. It was initially developed to connect the ILUs and their approximate inverse factors. These relations are important since, in order to solve linear systems, the inverse triangular factors resulting from the factorization are applied rather than the original incomplete factors themselves. Thus, information extracted from the inverse factors will in turn help to improve the robustness for the incomplete factorization process. While this idea has been successfully used to improve robustness, its downside was initially that the norm of the inverse factors could become large such that small entries could hardly be dropped during Gaussian elimination. To overcome this shortcoming, a multilevel strategy was developed to limit the growth of the inverse factors. This has led to the inverse-based approach and hence the incomplete factorization process that has eventually been implemented in ILUPACK benefits from the information of bounded inverse factors while being efficient at the same time [6].

A parallel version of ILUPACK on shared-memory multiprocessors, including current multicore architectures, is under development and expected to be released in the near future. The ongoing development of the parallel code is inspired by a nested-dissection hierarchy of the initial system which allows to map tasks concurrently to independent threads within each level.

The Multilevel Method

To solve a linear system $Ax = b$, the multilevel approach of ILUPACK performs the following steps:

1. The given system A is scaled by diagonal matrices D_l and D_r and reordered by permutation matrices Π_l, Π_r as
$$A \to D_l A D_r \to \Pi_l^T D_l A D_r \Pi_r = \hat{A}.$$
These operations can be considered as a preprocessing prior to the numerical factorization. They typically include scaling strategies to equilibrate the system, scaling and permuting based on maximum weight matchings [7] and, finally, fill-reducing orderings such as nested dissection [14], (approximate) minimum degree [11, 4] and some more.

2. An incomplete factorization $A \approx LDU$ is next computed for the system $\hat{A}$, where L, U^T are unit lower triangular factors and D is diagonal. Since the main objective of ILUPACK (*inverse-based approach*) is to limit the norm of the inverse triangular factors, L^{-1} and U^{-1}, the approximate factorization process is interlaced with a pivoting strategy that cheaply estimates the norm of these inverse factors. The pivoting process decides in each step to reject a factorization step if an imposed bound κ is exceeded, or to accept a pivot and continue the approximate factorization otherwise. The set of rejected rows and columns is permuted to the lower and right end of the matrix. This process is illustrated in Figure 1.

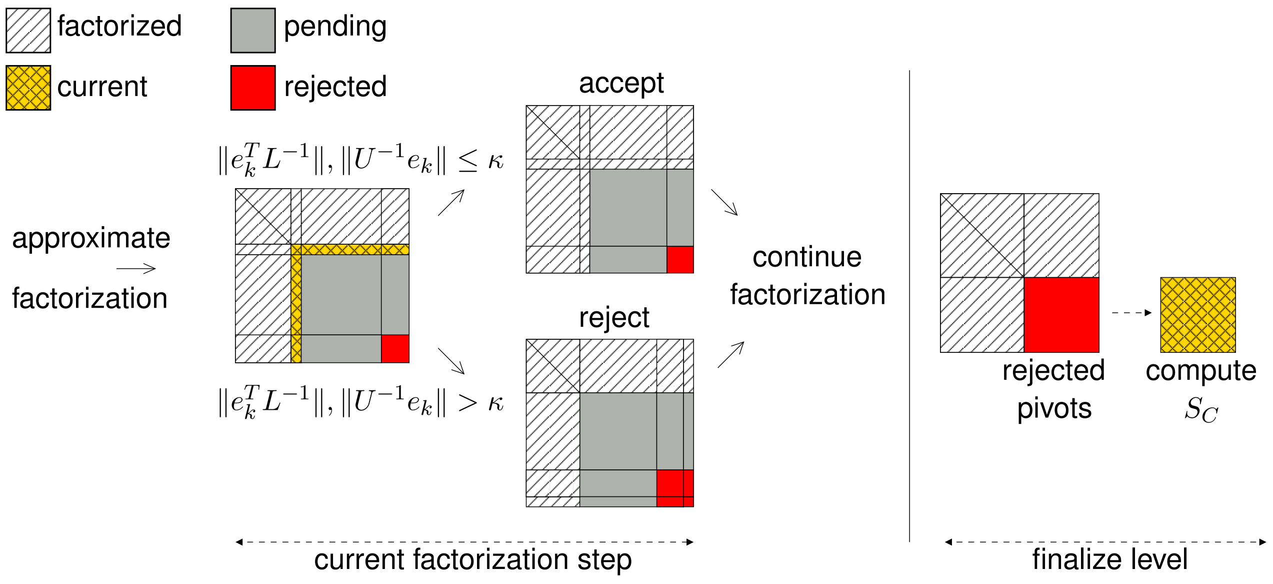

Figure 1: ILUPACK pivoting strategy.

The pivoting strategy computes a partial factorization

$$P^T \hat{A} P = \begin{pmatrix} B & E \\ F & C \end{pmatrix} = \begin{pmatrix} L_B & 0 \\ L_F & I \end{pmatrix} \begin{pmatrix} D_B U_B & D_B U_E \\ 0 & S_C \end{pmatrix} + R,$$

where R is the error matrix which collects those entries of $\hat{A}$ that were dropped during the factorization and "S_C" is the Schur complement consisting of all rows and columns associated with the rejected pivots. By construction the inverse triangular factors satisfy

$$\left\| \begin{pmatrix} L_B & 0 \\ L_F & I \end{pmatrix}^{-1} \right\|, \left\| \begin{pmatrix} U_B & U_E \\ 0 & I \end{pmatrix}^{-1} \right\| \lessapprox \kappa.$$

3. Steps 1 and 2 are successively applied to $\hat{A} = S_C$ until S_C is void or "sufficiently dense" to be efficiently factorized by a level 3 BLAS-based direct factorization kernel.

When the multilevel method is applied over multiple levels a cascade of factors L_B, D_B, U_B as well as matrices E, F are usually obtained (cf. Figure 2). Solving linear systems via a multilevel ILU requires a hierarchy of forward and backward substitutions, interlaced with reordering and scaling stages. ILUPACK employs Krylov subspace methods to incorporate the approximate factorization into an iterative method.

The computed multilevel factorization is adapted to the structure of the underlying system. For real symmetric positive definite (SPD) matrices, an incomplete Cholesky decomposition is used in conjunction with the conjugate gradient method. For the symmetric and indefinite case, symmetric maximum weight matchings [8] allow to build 1×1 and 2×2 pivots. In this case ILUPACK constructs a blocked symmetric multilevel ILU and relies on the simplified QMR [10] as iterative solver. The general (unsymmetric) case typically uses GMRES [16] as default driver. All drivers in ILUPACK support real/complex and single/double precision arithmetic.

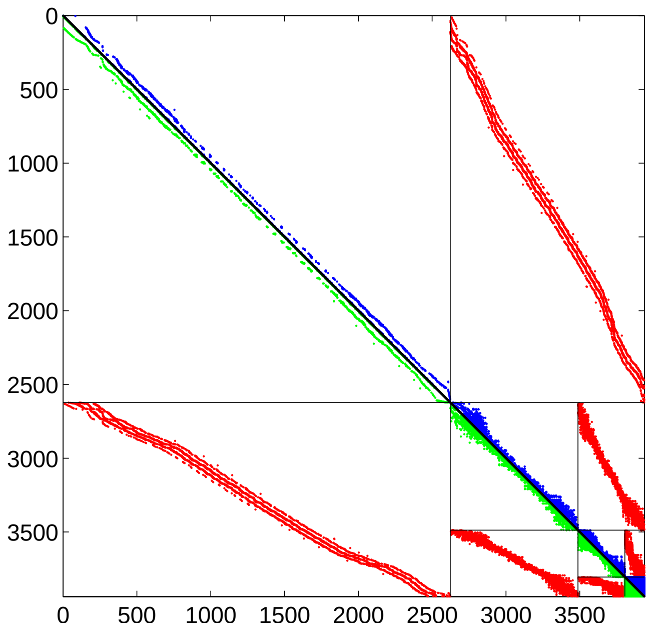

Figure 2: ILUPACK multilevel factorization.

Mathematical Background

The motivation for inverse-based approach of ILUPACK can be explained in two ways. First, when the approximate factorization

$$A = LDU + R$$

is computed for some error matrix R, the inverse triangular factors L^{-1} and U^{-1} have to be applied to solve a linear system $Ax = b$. From this point of view,

$$L^{-1}AU^{-1} = D + L^{-1}RU^{-1}$$

ensures that the entries in the error matrix R are not amplified by some large inverse factors L^{-1} and U^{-1}. The second and more important aspect links ILUPACK's multilevel ILU to algebraic multilevel methods. The inverse $\hat{A}^{-1}$ can be approximately written as

$$\hat{A}^{-1} \approx P \left(\begin{pmatrix} (L_B D_B U_B)^{-1} & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} -U_B^{-1} U_E \\ I \end{pmatrix} S_C^{-1} \begin{pmatrix} -L_F L_B^{-1} & I \end{pmatrix} \right) P^T.$$

In this expression, all terms on the right hand side are constructed to be bounded except S_C^{-1}. Since in general $\hat{A}^{-1}$ has a relatively large norm, so does S_C^{-1}. In principle this justifies that eigenvalues of small modulus are revealed by the approximate Schur complement S_C (a precise explanation is given in [5]). S_C serves as some kind of coarse grid system in the sense of discretized partial differential equations. This observation goes hand in hand with the observation that the inverse-based pivoting approach selects pivots similar to coarsening strategies in algebraic multilevel methods, which is demonstrated for the following simple model problem

$$-10^{-2}u_{xx}(x,y) - u_{yy}(x,y) = f(x,y) \text{ for all } (x,y) \in [0,1]^2, u(x,y) = 0 \text{ on } \partial[0,1]^2.$$

This partial differential equation can be easily discretized on a square grid $\Omega_h = \{(kh, lh) : \ k, l = 0, \ldots, N+1\}$, where $h = \frac{1}{N+1}$ is the mesh size, using standard finite difference techniques. Algebraic approaches to multilevel methods roughly treat the system as if the term $-10^{-2}u_{xx}(x,y)$ is hardly present, i.e., the coarsening process treats it as if it were a sequence of one-dimensional differential equations in the y-direction. Thus, semi-coarsening in the y-direction is the usual approach to build a coarse grid. ILUPACK inverse-based pivoting algebraically picks and rejects the pivots precisely in the same way as in semi-coarsening. In Figure 3, this is illustrated for a portion of the grid Ω_h using $\kappa = 3$. In the y-direction, pivots are rejected after 3 to 5 steps while in the x-direction all pivots are kept or rejected (in blue and red, resp.).

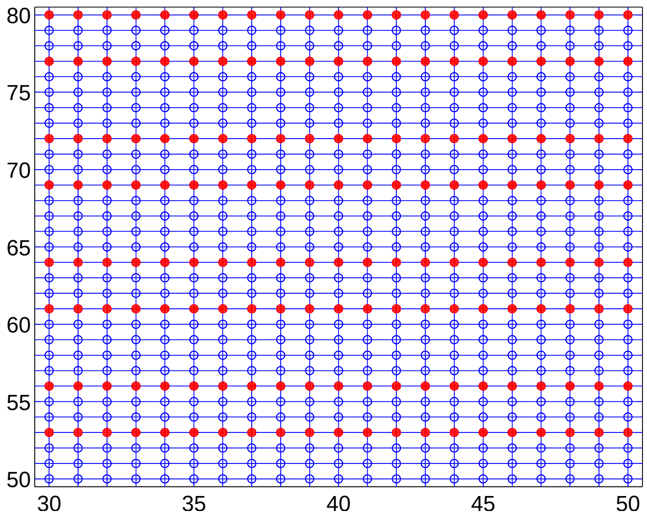

Figure 3: ILUPACK pivoting for partial differential equations. Blue and red dots denote, respectively, accepted and rejected pivots.

The number of rejected pivots in ILUPACK strongly depends on the underlying application problem. As a rule of thumb, the more rows (and columns) of the coefficient matrix satisfy $|a_{ii}| \gg \sum_{j \neq i} |a_{ij}|$, the less pivots tend to be rejected.

The Parallelization Approach

Parallelism in the computation of approximate factorizations can be exposed by means of graph-based symmetric reordering algorithms, such as graph coloring or graph partitioning techniques. Among these classes of algorithms, nested dissection orderings enhance parallelism in the approximate factorization of A by partitioning its associated adjacency graph $G(A)$ into a hierarchy of vertex separators and independent subgraphs. For example, in Figure 4, $G(A)$ is partitioned after two levels of recursion into four independent subgraphs, $G_{(3,1)}$, $G_{(3,2)}$, $G_{(3,3)}$ and $G_{(3,4)}$, first by separator $S_{(1,1)}$, and then by separators $S_{(2,1)}$ and $S_{(2,2)}$. This hierarchy is constructed so that the size of vertex separators is minimized while simultaneously balancing the size of the independent subgraphs. Therefore, relabeling the nodes of $G(A)$ according to the levels in the hierarchy, leads to a reordered matrix, $A \rightarrow \Phi^T A\Phi$, with a structure amenable to efficient parallelization. In particular, the leading diagonal blocks of $\Phi^T A\Phi$ associated with the independent subgraphs can be first eliminated independently; after that, $S_{(2,1)}$ and $S_{(2,2)}$ can be eliminated in parallel, and finally separator $S_{(1,1)}$ is processed. This type of parallelism can be expressed by a binary task dependency tree, where nodes represent concurrent tasks and arcs dependencies among them. State-of-the-art reordering software packages as, e.g., METIS[1] or SCOTCH[2], provide fast and efficient multilevel variants [14] of nested dissection orderings.

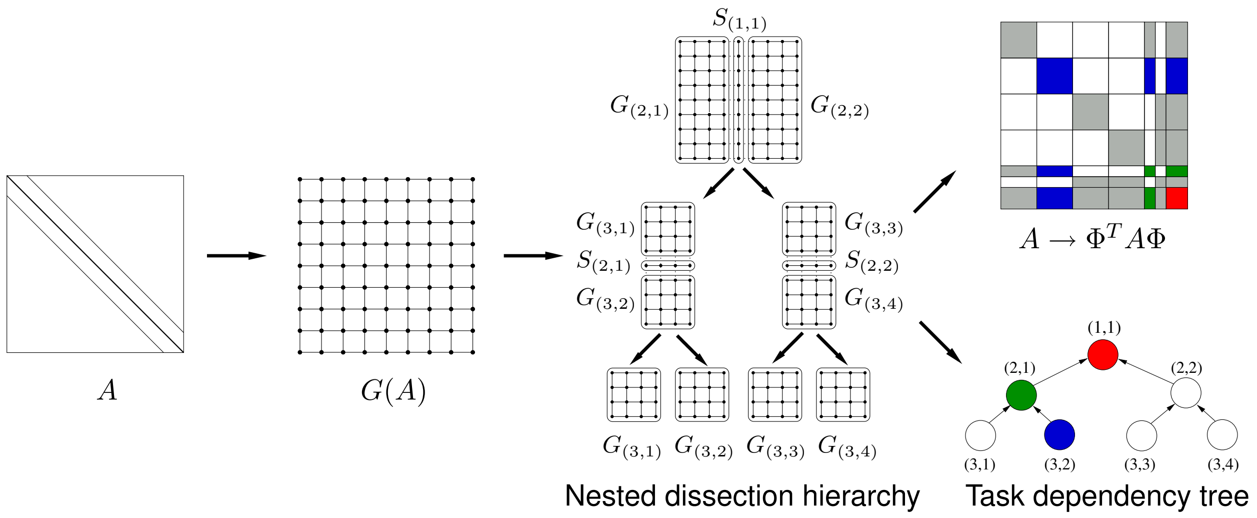

Figure 4: Nested dissection reordering.

The dependencies in the task tree are resolved while the computational data and results are generated and passed from the leaves towards the root. The leaves are responsible for approximating the leading diagonal blocks of $\Phi^T A\Phi$, while those blocks which will be later factorized by their ancestors are updated. For example, in Figure 4, colors are used to illustrate the correspondence between the blocks of $\Phi^T A\Phi$ to be factorized by tasks (3,2), (2,1) and (1,1). Besides, (3,2) only updates those blocks that will be later factorized by tasks (2,1) and (1,1). Taking this into consideration, $\Phi^T A\Phi$ is decomposed into the sum of several submatrices, one local block per each leaf of the tree, as shown in Figure 5. Each local submatrix is composed of the blocks to be factorized by the corresponding task, together with its local contributions to the blocks which are later factorized by its ancestors, hereafter referred as contribution blocks. This strategy significantly increases the degree of parallelism, because the updates from descendants nodes to an ancestor node can also be performed locally/independently by its descendants.

The parallel multilevel method considers the following partition of the local submatrices into 2×2 block

[1] http://glaros.dtc.umn.edu/gkhome/views/metis
[2] http://www.labri.fr/perso/pelegrin/scotch

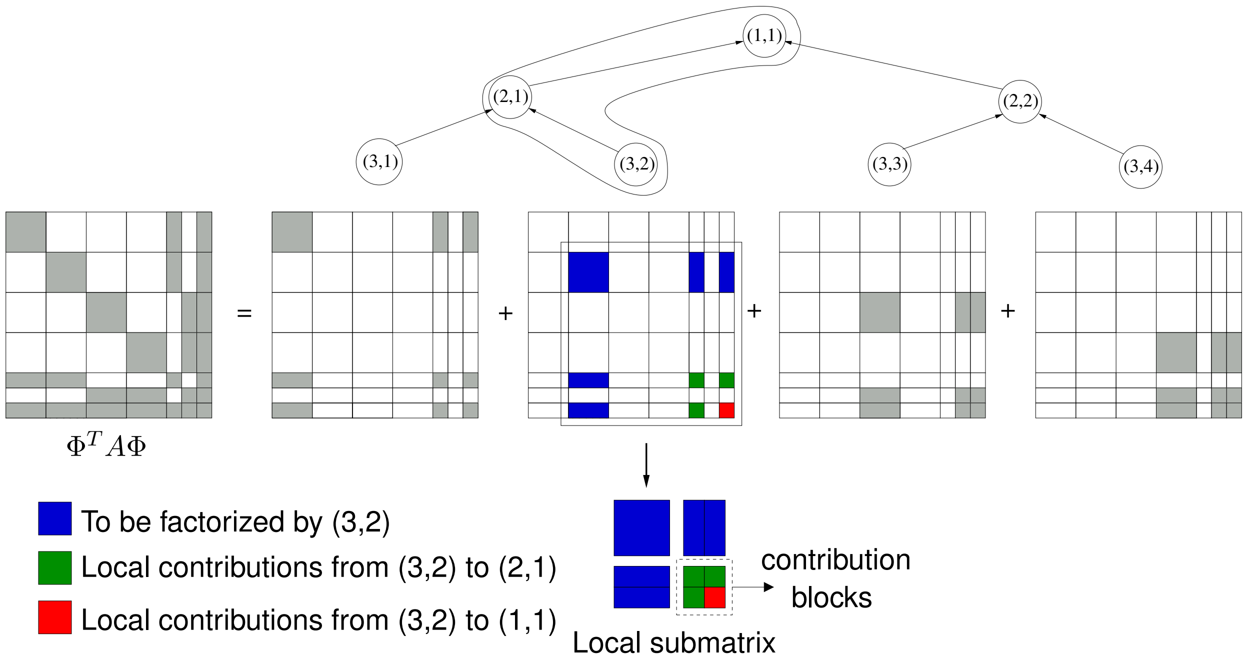
(1,1)
(2,1)
(2,2)
(3,1)
(3,2)
(3,3)
(3,4)
$\Phi^T A \Phi$
=
+
+
+
🟦 To be factorized by (3,2)
🟩 Local contributions from (3,2) to (2,1)
🟥 Local contributions from (3,2) to (1,1)
contribution blocks
Local submatrix

Figure 5: Matrix decomposition and local submatrix associated to a single node of the task tree.

matrices

$$A_{\text{par}} = \left(\begin{array}{c|c} A_X & A_V \\ \hline A_W & A_Z \end{array} \right),$$

where the partitioning lines separate the blocks to be factorized by the task, i.e., A_X, A_W, and A_V, and its contribution blocks, i.e., A_Z. It then performs the following steps:

1. Scaling and permutation matrices are only applied to the blocks to be factorized

$$A_{\text{par}} \to \left(\begin{array}{c|c} \Pi_l^T & 0 \\ \hline 0 & \text{I} \end{array} \right) \left(\begin{array}{c|c} D_l & 0 \\ \hline 0 & \text{I} \end{array} \right) \left(\begin{array}{c|c} A_X & A_V \\ \hline A_W & A_Z \end{array} \right) \left(\begin{array}{c|c} D_r & 0 \\ \hline 0 & \text{I} \end{array} \right) \left(\begin{array}{c|c} \Pi_r & 0 \\ \hline 0 & \text{I} \end{array} \right) = \left(\begin{array}{c|c} \hat{A}_X & \hat{A}_V \\ \hline \hat{A}_W & \hat{A}_Z \end{array} \right) = \hat{A}_{\text{par}}.$$

2. These blocks are next approximately factorized using inverse-based pivoting, while $\hat{A}_Z$ is only updated. For this purpose, during the incomplete factorization of $\hat{A}_{\text{par}}$, rejected rows and columns are permuted to the lower and right end of the leading block $\hat{A}_X$. This is illustrated in Figure 6.

   This step computes a partial factorization

$$\begin{aligned} \left(\begin{array}{c|c} P^T & 0 \\ \hline 0 & \text{I} \end{array} \right) \hat{A}_{\text{par}} \left(\begin{array}{c|c} P & 0 \\ \hline 0 & \text{I} \end{array} \right) &= \left(\begin{array}{cc|c} B_X & E_X & E_V \\ F_X & C_X & C_V \\ \hline F_W & C_W & C_Z \end{array} \right) \\ &= \left(\begin{array}{cc|c} L_{B_X} & 0 & 0 \\ L_{F_X} & I & 0 \\ \hline L_{F_W} & 0 & I \end{array} \right) \left(\begin{array}{cc|c} D_{B_X} U_{B_X} & D_{B_X} U_{E_X} & D_{B_X} U_{E_V} \\ 0 & S_{C_X} & S_{C_V} \\ \hline 0 & S_{C_W} & S_{C_Z} \end{array} \right) + R, \end{aligned}$$

   where the inverse triangular factors approximately satisfy

$$\left\| \left(\begin{array}{cc|c} L_{B_X} & 0 & 0 \\ L_{F_X} & \text{I} & 0 \\ \hline L_{F_W} & 0 & \text{I} \end{array} \right)^{-1} \right\|, \left\| \left(\begin{array}{cc|c} U_{B_X} & U_{E_X} & U_{E_V} \\ 0 & \text{I} & 0 \\ \hline 0 & 0 & \text{I} \end{array} \right)^{-1} \right\| \lessapprox \kappa.$$

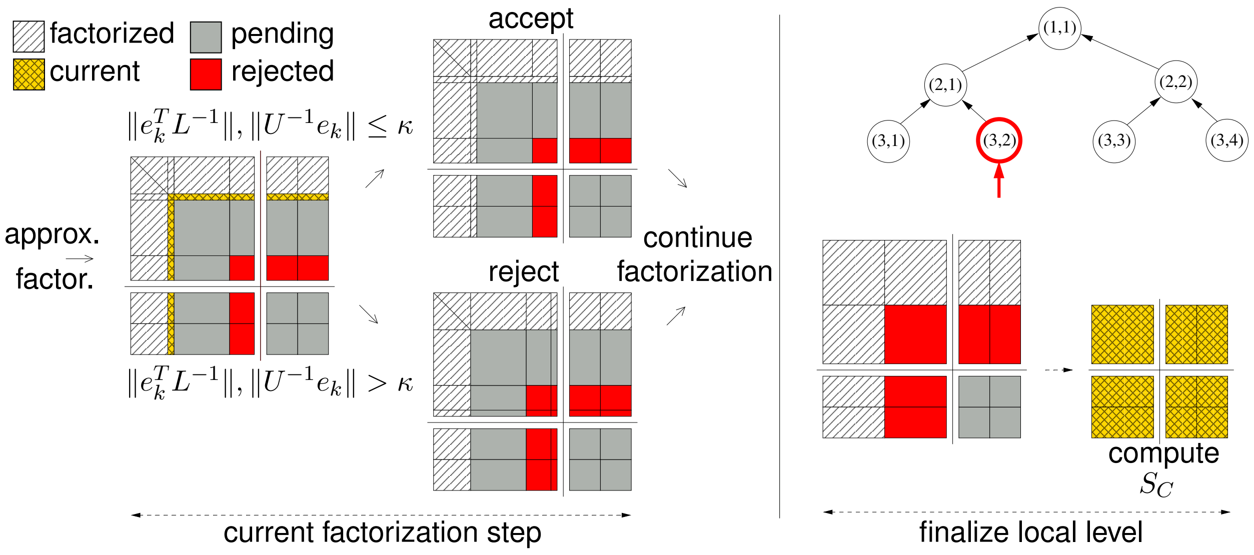

Figure 6: Local incomplete factorization computed by a single node of the task tree.

3. Steps 1 and 2 are successively applied to the Schur complement until S_{C_X} is void or "sufficiently small", i.e., A_{par} and its 2×2 block partition are redefined as

$$A_{\text{par}} = \left(\begin{array}{c|c} A_X & A_V \\ \hline A_W & A_Z \end{array}\right) := \left(\begin{array}{c|c} S_{C_X} & S_{C_V} \\ \hline S_{C_W} & S_{C_Z} \end{array}\right).$$

4. The task completes its local computations and the result A_{par} is sent to the parent node in the task dependency tree. When a parent node receives the data from its children, it must first construct its local submatrix A_{par}. To achieve this, it incorporates the pivots rejected from its children and accumulates their contribution blocks, as shown in Figure 7. If the parent node is the root of the task dependency tree, it applies the sequential multilevel algorithm to the new constructed submatrix A_{par}. Otherwise, the parallel multilevel method is restarted on this matrix at step 1.

To compare the cascade of approximations computed by the sequential multilevel method and its parallel variant, it is helpful to consider the latter as an algorithmic variant which enforces a certain order of elimination. Thus, the parallel variant interlaces the algebraic levels generated by the pivoting strategy of ILUPACK with the nested dissection hierarchy levels in order to expose a high degree of parallelism. The leading diagonal blocks associated with the last nested dissection hierarchy level are first factorized using inverse-based pivoting, while those corresponding to previous hierarchy levels are only updated, i.e., the nodes belonging to the separators are rejected by construction. The multilevel algorithm is restarted on the rejected nodes, and only when it has eliminated the bulk of the nodes of the last hierarchy level, it starts approximating the blocks belonging to previous hierarchy levels. Figure 8 compares the distribution of accepted and rejected nodes (in blue and red, resp.) by the sequential and parallel inverse-based incomplete factorizations when they are applied to the Laplace PDE with discontinuous coefficients discretized with a standard 200×200 finite-difference grid. In both cases, the grid was reordered using nested dissection. These diagrams confirm the strong similarity between the sequential inverse-based incomplete factorization and its parallel variant

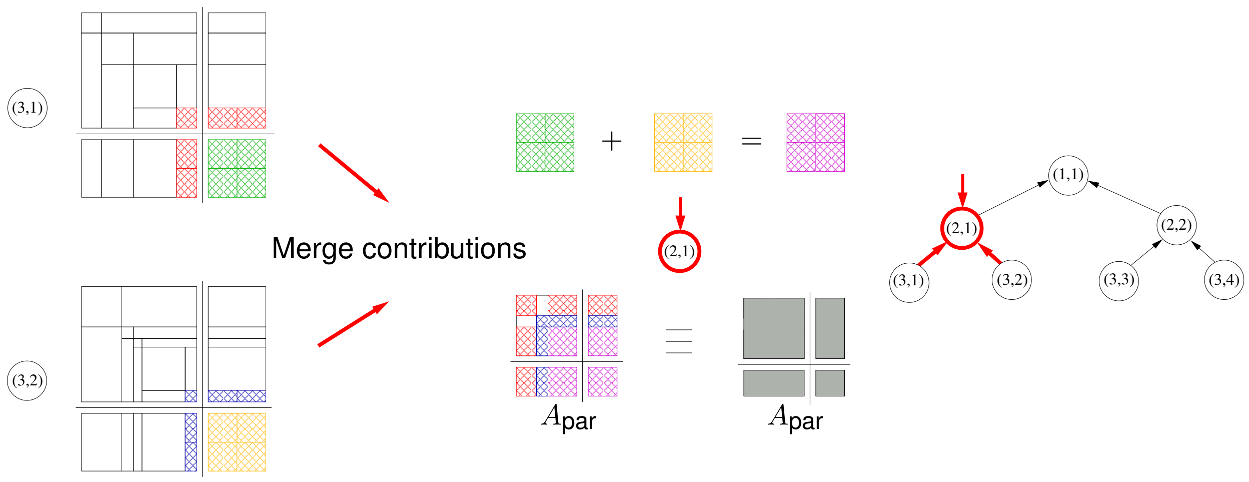

Figure 7: Parent nodes construct their local submatrix from the data generated by their children.

for this particular example. The experimentation with this approach in the SPD case reveals that this compromise has a negligible impact on the numerical properties of the inverse-based preconditioning approach, in the sense that the convergence rate of the preconditioned iterative method is largely independent on the number of processors involved in the parallel computation.

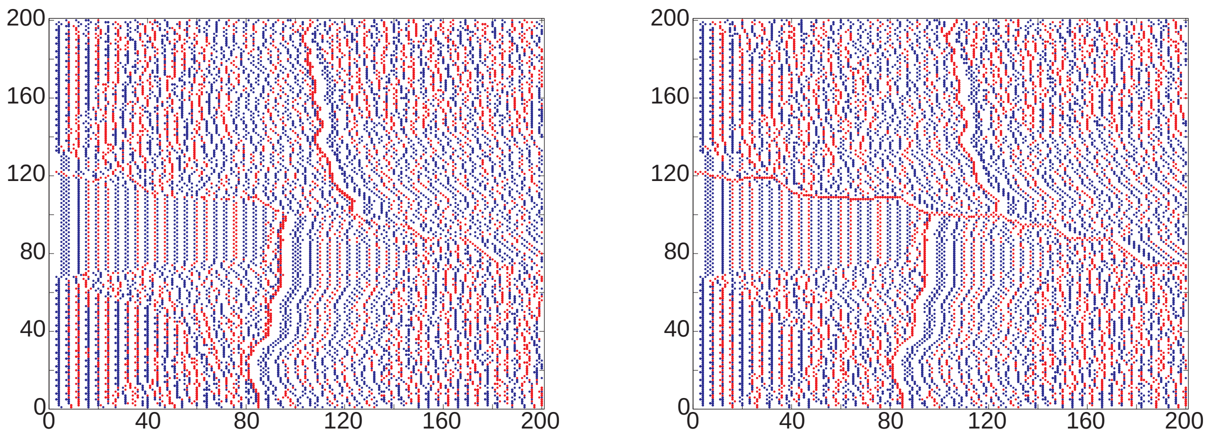

Figure 8: ILUPACK pivoting strategy (left) and its parallel variant (right). Blue and red dots denote, respectively, accepted and rejected nodes.

Numerical Example

The example illustrates the parallel and memory scalability of the parallel multilevel method on a SGI Altix 350 CC-NUMA shared-memory multiprocessor with 16 Intel Itanium2@1.5 GHz processors sharing 32 GBytes of RAM connected via a SGI NUMAlink network. The nine linear systems considered in this experiment are derived from the linear finite element discretization of the irregular 3-D elliptic PDE $[-\mathrm{div}\,(A\ \mathrm{grad}\ u) = f]$ in a 3-D domain, where $A(x, y, z)$ is chosen with positive random coefficients. The size of the systems ranges from $n = 1,709$ to $5,413,520$ equations/unknowns.

The parallel execution of the task tree on shared-memory multiprocessors is orchestrated by a runtime which dynamically maps tasks to threads (processors) in order to improve load balance requirements during the computation of the multilevel preconditioner. Figure 9 displays two lines for each number of processors: dashed lines are obtained for a perfect binary tree with the same number of leaves as processors, while solid lines correspond to a binary tree with (potentially) more leaves than processors (up to $2p$ leaves). The higher speed-ups revealed in the solid lines demonstrate the benefit of dynamic scheduling on shared-memory multiprocessors.

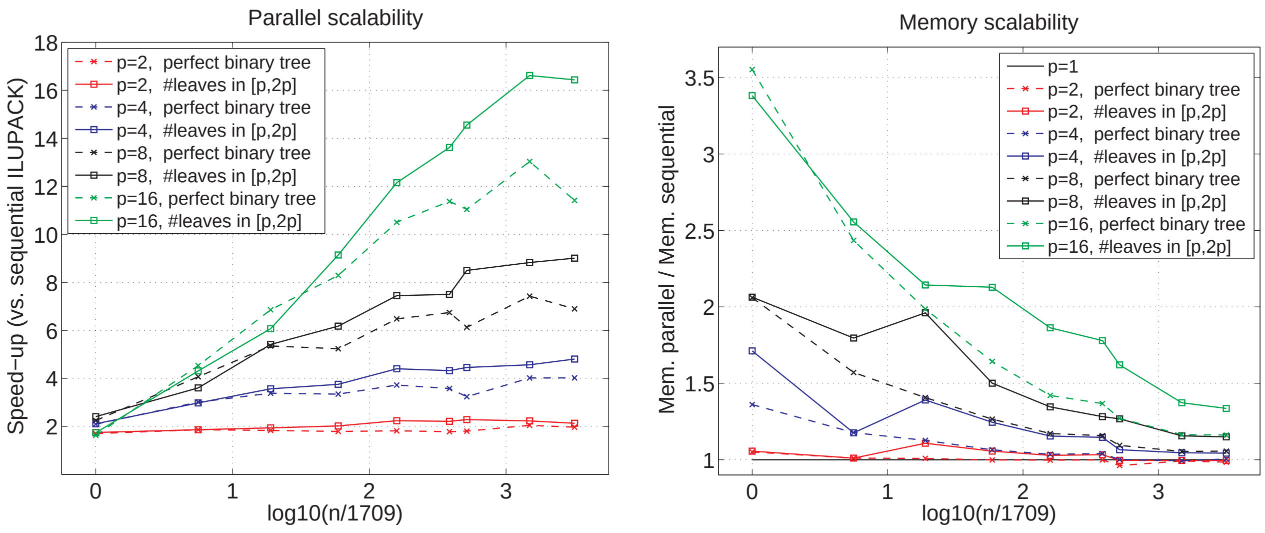

Figure 9: Parallel speed-up as a function of problem size (left) and ratio among the memory consumed by the parallel multilevel method and that of the sequential method (right) for a discretized 3-D elliptic PDE.

As shown in Figure 9 (left), the speed-up always increases with the number of processors for a fixed problem size, and the parallel efficiency rapidly grows with problem size for a given number of processors. Besides, as shown in Figure 9 (right), the memory overhead associated with parallelization relatively decreases with the problem size for a fixed number of processors, and it is below 1.5 for the two largest linear systems; this is very moderate taking into consideration that the amount of available physical memory typically increases linearly with the number of processors. These observations confirm the excellent scalability of the parallelization approach up to 16 cores.

Future Research Directions

ILUPACK is currently parallelized for SPD matrices only. Further classes of matrices such as symmetric and indefinite matrices or unsymmetric, but symmetrically structured matrices are subject to ongoing research. The parallelization of these cases shares many similarities with the SPD case. However, maximum weight matching, or more generally, methods to rescale and reorder the system to improve the size of the diagonal entries (resp. diagonal blocks) are hard to parallelize [9]. Although the current implementation of ILUPACK is designed for shared-memory multiprocessors using OpenMP, the basic principle is currently being transferred to distributed-memory parallel architectures via MPI. Future research will be devoted to block-structured algorithms to improve cache performance. Block structures have been proven to be extremely efficient for direct methods, but their integration into incomplete factorizations remains a challenge; on the other hand, recent research indicates the high potential of block-structured algorithms also for ILUs [12, 13].

RELATED ENTRIES

Shared-Memory Multiprocessor, NUMA Shared-Memory Machines, Load Balancing, Parallel Algorithms, Sparse Matrix Algorithms, Linear Algebra Software, Graph Partitioning.

BIBLIOGRAPHIC NOTES AND FURTHER READING

Around 1985 a first package also called ILUPACK was developed by H.D. Simon [18] which used reorderings, incomplete factorizations and iterative methods in one package. Nowadays, as the development of preconditioning methods has advanced significantly by novel approaches like improved reorderings, multilevel methods, maximum weight matchings and inverse-based pivoting, incomplete factorization methods have changed completely and gained wide acceptance among the scientific community. Beside ILUPACK there exist several software packages based on incomplete factorizations and on multilevel factorizations. E.g., Y. Saad[3] et al. developed the software packages SPARSKIT, ITSOL and pARMS which also inspired the development of ILUPACK. In particular, pARMS is a parallel code based on multilevel ILU interlaced with reorderings. J. Mayer[4] developed ILU++, also for multilevel ILU. MRILU, by F.W. Wubs[5] et al., uses multilevel incomplete factorizations and has successfully been applied to problems arising from partial differential equations. There are many other scientific publications on multilevel ILU methods, most of them especially tailored for the use in partial differential equations.

ILUPACK has been successfully applied to several large scale application problems in particular to the Anderson model of localization [17] or the Helmholtz equation [5]. Its symmetric version is integrated into the software package JADAMILU[6], which is a Jacobi-Davidson-based eigenvalue solver for symmetric eigenvalue problems.

More details on the design aspects of the parallel multilevel preconditioner (including dynamic scheduling) and experimental data with a benchmark of irregular matrices from the UF sparse matrix collection[7] can be found in [1, 2]. The computed parallel multilevel factorization is incorporated to a Krylov subspace solver, and the underlying parallel structure of the former, expressed by the task dependency tree, is exploited for the application of the preconditioner as well as other major operations in

[3] `http://www-users.cs.umn.edu/~saad/software/`

[4] `http://iamlasun8.mathematik.uni-karlsruhe.de/~ae04/iluplusplus.html`

[5] `http://www.math.rug.nl/~wubs/mrilu/`

[6] `http://homepages.ulb.ac.be/~jadamilu/`

[7] `http://www.cise.ufl.edu/research/sparse/matrices/`

iterative solvers [2]. The experience with these parallel techniques has revealed that the sequential computation of the nested dissection hierarchy (included in, e.g., METIS or SCOTCH) dominates the execution time of the whole solution process when the number of processors is large. Therefore, additional types of parallelism have to be exploited during this stage in order to develop scalable parallel solutions; in [3], two parallel partitioning packages, ParMETIS[8] and PT-SCOTCH[9], are evaluated with this purpose.

BIBLIOGRAPHY

[1] J. I. Aliaga, M. Bollhöfer, A. F. Martín, and E. S. Quintana-Ortí. Design, tuning and evaluation of parallel multilevel ILU preconditioners. In J. Palma, P. Amestoy, M. Dayde, M. Mattoso, and J. C. Lopes, editors, *High Performance Computing for Computational Science - VECPAR 2008*, number 5336 in Lecture Notes in Computer Science, pages 314–327. Springer, 2008.

[2] J. I. Aliaga, M. Bollhöfer, A. F. Martín, and E. S. Quintana-Ortí. Exploiting thread-level parallelism in the iterative solution of sparse linear systems. Technical report, Dpto. de Ingeniería y Ciencia de Computadores, Universitat Jaume I, Castellón, 2008. submitted for publication.

[3] J. I. Aliaga, M. Bollhöfer, A. F. Martín, and E. S. Quintana-Ortí. Evaluation of parallel sparse matrix partitioning software for paral lel multilevel ILU preconditioning on shared-memory multiprocessors. In B. C. et al., editor, *Parallel Computing: From Multicores and GPUs to Petascale*, volume 19 of *Advances in Parallel Computing*, pages 125–132. IOS Press, 2009.

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

[5] M. Bollhöfer, M. J. Grote, and O. Schenk. Algebraic Multilevel Preconditioner for the Helmholtz Equation in Heterogeneous Media. *SIAM J. Sci. Comput.*, 31(5):3781–3805, 2009.

[6] M. Bollhöfer and Y. Saad. Multilevel Preconditioners Constructed from Inverse–Based ILUs. *SIAM J. Sci. Comput.*, 27(5):1627–1650, 2006.

[7] 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):889–901, 1999.

[8] I. S. Duff and S. Pralet. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM J. Matrix Anal. Appl.*, 27(2):313–340, 2005.

[9] I. S. Duff and B. Uçar. Combinatorial problems in solving linear systems. Technical Report TR/PA/09/60, CERFACS, August 2009.

[10] R. Freund and N. Nachtigal. Software for simplified Lanczos and QMR algorithms. *Appl. Numer. Math.*, 19(3):319–341, 1995.

[11] A. George and J. W. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31(1):1–19, 1989.

[8] `http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview`

[9] `http://www.labri.fr/perso/pelegrin/scotch`

[12] A. Gupta and T. George. Adaptive techniques for improving the performance of incomplete factorization preconditioning. *SIAM J. Sci. Comput.*, 32(1):84–110, 2010.

[13] P. Hénon, P. Ramet, and J. Roman. On finding approximate supernodes for an efficient block-ILU(k) factorization. *Parallel Computing*, 34(6-8):345–362, 2008.

[14] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[15] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM Publications, second edition, 2003.

[16] Y. Saad and M. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Scientific and Statistical Computing*, 7:856–869, 1986.

[17] O. Schenk, M. Bollhöfer, and R. A. Römer. Awarded SIGEST paper: On large scale diagonalization techniques for the Anderson model of localization. *SIAM Review*, 50:91–112, 2008.

[18] H. D. Simon. User guide for ILUPACK: Incomplete LU factorization and iterative methods. Technical Report ETA-LR-38, Boeing Computer Services, January 1985.