Parallelization of Multilevel ILU Preconditioners on Distributed-Memory Multiprocessors

José I. Aliaga[1], Matthias Bollhöfer[2],
Alberto F. Martín[1], and Enrique S. Quintana-Ortí[1]

[1] Dpto. de Ingen. y Ciencia de Computadores, Universidad Jaume I, Spain
{aliaga,martina,quintana}@icc.uji.es
[2] Institute of Computational Mathematics, TU-Braunschweig, Germany
m.bollhoefer@tu-braunschweig.de

Abstract. In this paper we investigate the parallelization of the ILUPACK library for the solution of sparse linear systems on distributed-memory multiprocessors. The parallelization approach employs multilevel graph partitioning algorithms in order to identify a set of concurrent tasks and their dependencies, which are then statically mapped to processors. Experimental results on a cluster of Intel QuadCore processors report remarkable speed-ups.

Keywords: Sparse linear system, iterative solver, preconditioner, ILU decomposition, MPI, distributed-memory multiprocessor.

1 Introduction

The solution of sparse linear systems is a computational bottleneck in many scientific computing application problems. While sparse direct methods have proven to be extremely efficient for a wide range of applications, the increasing size of the problems arising from 3D PDEs applications asks for fast and efficient iterative solution techniques. This in turn requires alternative techniques like approximate factorizations combined with Krylov subspace methods, because of their moderate computational and memory requirements [10]. Among these, ILUPACK[1] (*Incomplete LU decomposition PACKage*) is a software package mainly based on ILU factorizations with improved robustness in conjunction with Krylov subspace methods.

Although the application of a preconditioner has the potential of accelerating the convergence rate of iterative solvers, the computational cost per iteration increases. Moreover, the time of computing the preconditioner also needs to be taken into account. To compensate for this, high-performance computing techniques can be applied to speed-up the computation of both the preconditioner and the iterative procedure. The parallelization of ILUPACK-based preconditioners on shared-memory multiprocessors, and scaling studies with up to 16 cores, are discussed in previous work [1,2,3]. As in sparse direct methods [7], this parallelization is inspired by a nested-dissection hierarchy of the initial system

[1] http://ilupack.tu-bs.de

K. Jónasson (Ed.): PARA 2010, Part I, LNCS 7133, pp. 162–172, 2012.

which allows to map independent tasks concurrently to cores within each level. This paper demonstrates that the same parallelization carries over to distributed-memory multiprocessors, reporting remarkable performance on up to 32 cores.

The paper is structured as follows. ILUPACK is briefly reviewed in Sect. 2. Details on the parallelization of this package on distributed-memory multiprocessors are given in Sect. 3. Finally, Sect. 4 contains experimental results collected from the parallel algorithm and offers a few concluding remarks.

2 Computation of Preconditioners in ILUPACK

Preconditioning in ILUPACK relies on the so-called *inverse-based approach*, which improves the robustness of classical ILU factorizations bounding the growth of the entries in the inverses of the triangular factors. To justify this, consider the ILU factorization

$$A = \tilde{L}\tilde{D}\tilde{U} + R \ , \tag{1}$$

where $\tilde{L}, \tilde{U}^T$ are unit lower triangular matrices, $\tilde{D}$ is diagonal, and R is the error matrix which collects those entries that were dropped during the factorization. Applying the preconditioner $M = \tilde{L}\tilde{D}\tilde{U}$, we obtain the preconditioned matrix

$$\tilde{L}^{-1}A\tilde{U}^{-1} = \tilde{D} + \tilde{L}^{-1}R\tilde{U}^{-1} \ . \tag{2}$$

Although dropping typically results in some "relatively small" error matrix R, both $\tilde{L}^{-1}$ and $\tilde{U}^{-1}$ may exhibit very large norms, so that application of the preconditioning can significantly amplify the size of the entries in R. This may directly impact the convergence rate of the preconditioned iterative solver.

The inverse-based preconditioning approach relies on approximate factorizations with "bounded" inverse triangular factors; i.e., factorizations with $\left\|L^{-1}\right\| \leq \kappa$ and $\left\|U^{-1}\right\| \leq \kappa$, for some prescribed small threshold $\kappa > 1$. In practical applications, an ILU factorization of the system at hand does not typically satisfy this requirement, so that pivoting is necessary to bound the inverse triangular factors during the computation. Pivoting is accommodated in a multilevel framework in order to construct a hierarchy of partial inverse-based approximations, as sketched in the following multilevel algorithm:

1. **Preprocessing step.** Matrix A is scaled by diagonal matrices D_l and D_r and reordered by permutation matrices P_l and P_r,

$$\hat{A} = P_l^T D_l A D_r P_r \ .$$

2. **Factorization step.** At each step of the Crout variant of the ILU factorization, the method is interlaced with a pivoting strategy which yields a nonexpensive estimation of the norm of a new row/column of the inverse factors. If the estimation exceeds the threshold κ, the current pivot is rejected and the corresponding row/column are moved to the bottom/right-end of the matrix. Otherwise, the pivot is accepted and dropping is applied to the current row/column before they are incorporated to the factors. This is illustrated

in Fig. 1. Collecting the permutations due to inverse-based pivoting on P, we obtain the following partial ILU factorization of a permuted matrix:

$$P^T \hat{A} P = \begin{bmatrix} \tilde{L}_B & 0 \\ \tilde{L}_E & \mathrm{I} \end{bmatrix} \begin{bmatrix} \tilde{D}_B & 0 \\ 0 & \tilde{S}_C \end{bmatrix} \begin{bmatrix} \tilde{U}_B & \tilde{U}_F \\ 0 & \mathrm{I} \end{bmatrix} + \begin{bmatrix} R_B & R_F \\ R_E & 0 \end{bmatrix} \ .$$

The method applies additional dropping to the approximate Schur complement $\tilde{S}_C$, so that we actually compute

$$\hat{S}_C = \tilde{S}_C + R_C = C - \left(\tilde{L}_E \tilde{D}_B \tilde{U}_F \right) + R_C \ .$$

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

For a more detailed description of the numerical approach which lays the foundation of ILUPACK and its theoretical properties see [4,5].

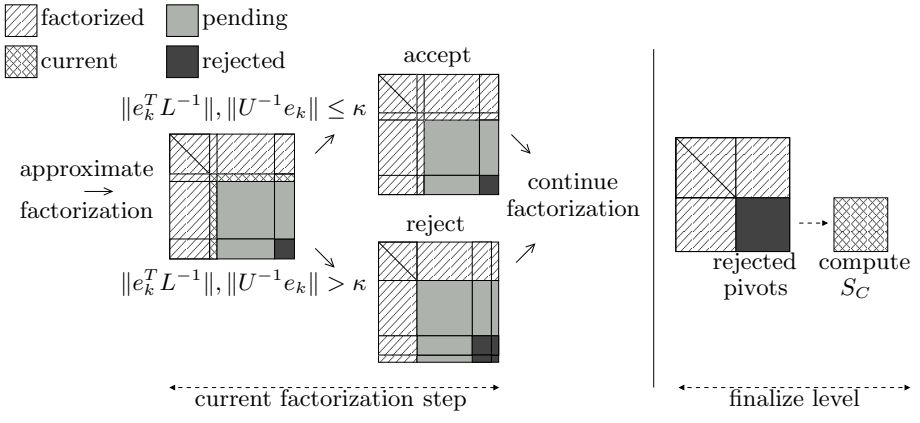

Fig. 1. ILUPACK pivoting strategy

3 Parallelization of ILUPACK

To design a parallel version of ILUPACK, we decompose the computation of the preconditioner into tasks, identify the dependencies among them, and apply static mapping to these operations.

For sparse linear systems, it is possible to apply graph-based symmetric reorderings to find a permutation Π such that

$$\Pi^T A \Pi = \left[\begin{array}{cc|c} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ \hline A_{31} & A_{32} & A_{33} \end{array} \right] \ . \tag{3}$$

Computing the ILU decomposition of the leading blocks A_{11} and A_{22}, we obtain the following partial approximation

$$\left[\begin{array}{cc|c} \tilde{L}_{11} & 0 & 0 \\ 0 & \tilde{L}_{22} & 0 \\ \hline \tilde{L}_{31} & \tilde{L}_{32} & \mathrm{I} \end{array}\right] \left[\begin{array}{cc|c} \tilde{D}_{11} & 0 & 0 \\ 0 & \tilde{D}_{22} & 0 \\ \hline 0 & 0 & \tilde{S}_{33} \end{array}\right] \left[\begin{array}{cc|c} \tilde{U}_{11} & 0 & \tilde{U}_{13} \\ 0 & \tilde{U}_{22} & \tilde{U}_{23} \\ \hline 0 & 0 & \mathrm{I} \end{array}\right] + \left[\begin{array}{cc|c} R_{11} & 0 & R_{13} \\ 0 & R_{22} & R_{23} \\ \hline R_{31} & R_{32} & 0 \end{array}\right] \ ,$$

where the approximate Schur complement is given by

$$\hat{S}_{33} = A_{33} - \left(\tilde{L}_{31}\tilde{D}_{11}\tilde{U}_{13}\right) - \left(\tilde{L}_{32}\tilde{D}_{22}\tilde{U}_{23}\right) + R_{33} \ ; \tag{4}$$

proceeding with the ILU factorization of $\hat{S}_{33}$, the ILU decomposition of $\Pi^T A \Pi$ is completed. The structure of $\Pi^T A \Pi$ allows the explotation of parallelism during this computation. In particular, we can disassemble $\Pi^T A \Pi$ into two submatrices

$$\left[\begin{array}{c|c} A_{11} & A_{13} \\ \hline A_{31} & A^1_{33} \end{array}\right] \ , \ \left[\begin{array}{c|c} A_{22} & A_{23} \\ \hline A_{32} & A^2_{33} \end{array}\right] \ , \ A^1_{33} + A^2_{33} = A_{33} \ , \tag{5}$$

so that the ILU decomposition of the leading block of both submatrices can be concurrently obtained,

$$\left[\begin{array}{c|c} A_{11} & A_{13} \\ \hline A_{31} & A^1_{33} \end{array}\right] = \left[\begin{array}{c|c} \tilde{L}_{11} & 0 \\ \hline \tilde{L}_{31} & \mathrm{I} \end{array}\right] \left[\begin{array}{c|c} \tilde{D}_{11} & 0 \\ \hline 0 & \tilde{S}^1_{33} \end{array}\right] \left[\begin{array}{c|c} \tilde{U}_{11} & \tilde{U}_{13} \\ \hline 0 & \mathrm{I} \end{array}\right] + \left[\begin{array}{cc} R_{11} & R_{13} \\ R_{31} & 0 \end{array}\right]$$

$$\left[\begin{array}{c|c} A_{22} & A_{23} \\ \hline A_{32} & A^2_{33} \end{array}\right] = \left[\begin{array}{c|c} \tilde{L}_{22} & 0 \\ \hline \tilde{L}_{32} & \mathrm{I} \end{array}\right] \left[\begin{array}{c|c} \tilde{D}_{22} & 0 \\ \hline 0 & \tilde{S}^2_{33} \end{array}\right] \left[\begin{array}{c|c} \tilde{U}_{22} & \tilde{U}_{23} \\ \hline 0 & \mathrm{I} \end{array}\right] + \left[\begin{array}{cc} R_{22} & R_{23} \\ R_{32} & 0 \end{array}\right] \ .$$

Then, we can also compute in parallel the Schur complements corresponding to both partial approximations

$$\hat{S}^1_{33} = A^1_{33} - \left(\tilde{L}_{31}\tilde{D}_{11}\tilde{U}_{13}\right) + R^1_{33} \ , \ \hat{S}^2_{33} = A^2_{33} - \left(\tilde{L}_{32}\tilde{D}_{22}\tilde{U}_{23}\right) + R^2_{33} \ .$$

However, the construction of (4) requires communication before the addition of these two blocks can be computed

$$R_{33} \approx R^1_{33} + R^2_{33} \ \rightarrow \ \hat{S}_{33} \approx \hat{S}^1_{33} + \hat{S}^2_{33} \ . \tag{6}$$

Finally, the sequential ILU factorization of $\hat{S}_{33}$ completes the parallel approximate factorization of $\Pi^T A \Pi$.

To expose a higher degree of parallelism, we need to identify a larger number of independent diagonal blocks. We can do this by applying a permutation similar to Π on the two leading blocks, and then reordering and renaming the blocks,

$$\left[\begin{array}{ccc|ccc|c} \hat{A}_{11} & 0 & \hat{A}_{13} & 0 & 0 & 0 & * \\ 0 & \hat{A}_{22} & \hat{A}_{23} & 0 & 0 & 0 & * \\ \hat{A}_{31} & \hat{A}_{32} & \hat{A}_{33} & 0 & 0 & 0 & * \\ \hline 0 & 0 & 0 & \bar{A}_{11} & 0 & \bar{A}_{13} & * \\ 0 & 0 & 0 & 0 & \bar{A}_{22} & \bar{A}_{23} & * \\ 0 & 0 & 0 & \bar{A}_{31} & \bar{A}_{32} & \bar{A}_{33} & * \\ \hline * & * & * & * & * & * & * \end{array}\right] \rightarrow \left[\begin{array}{cccc|cc|c} A_{11} & 0 & 0 & 0 & A_{15} & 0 & A_{17} \\ 0 & A_{22} & 0 & 0 & A_{25} & 0 & A_{27} \\ 0 & 0 & A_{33} & 0 & 0 & A_{36} & A_{37} \\ 0 & 0 & 0 & A_{44} & 0 & A_{46} & A_{47} \\ \hline A_{51} & A_{52} & 0 & 0 & A_{55} & 0 & A_{57} \\ 0 & 0 & A_{63} & A_{64} & 0 & A_{66} & A_{67} \\ \hline A_{71} & A_{72} & A_{73} & A_{74} & A_{75} & A_{76} & A_{77} \end{array}\right] \ . \tag{7}$$

Figure 2 illustrates the dependency tree for the factorization of the diagonal blocks in the right-hand side of (7); there, the nodes which lie at the same height can be factored in parallel and the edges of the graph define the dependencies between the diagonal blocks, other words, the order in which the blocks of the matrix have to be processed. We identify three classes of nodes in the figure:

1. The **Leaf nodes**, which are responsible for the factorization of the four leading diagonal blocks in parallel.
2. The **Intermediate nodes**, which factorize in parallel the next two intermediate diagonal blocks, A_{55} and A_{66}. These blocks cannot be factorized unless the leading diagonal blocks corresponding to its children have been already factorized, i.e., A_{11} - A_{22} and A_{33} - A_{44} respectively.
3. The **Root node**, which sequentially factorizes the last diagonal block, A_{77}. This approximation can be only computed when all the preceding diagonal blocks have been processed.

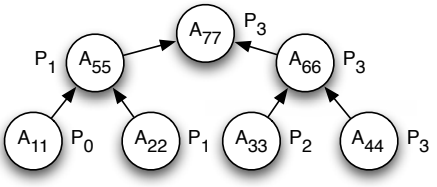

Fig. 2. Dependency tree of the diagonal blocks

The parallel computation of the preconditioner also commences by disassembling A, with one submatrix for each leaf node,

$$\left[\begin{array}{c|cc} A_{11} & A_{15} & A_{17} \\ \hline A_{51} & A_{55}^1 & A_{57}^1 \\ \hline A_{71} & A_{75}^1 & A_{77}^1 \end{array}\right], \left[\begin{array}{c|cc} A_{22} & A_{25} & A_{27} \\ \hline A_{52} & A_{55}^2 & A_{57}^2 \\ \hline A_{72} & A_{75}^2 & A_{77}^2 \end{array}\right], \left[\begin{array}{c|cc} A_{33} & A_{36} & A_{37} \\ \hline A_{63} & A_{66}^3 & A_{67}^3 \\ \hline A_{73} & A_{76}^3 & A_{77}^3 \end{array}\right], \left[\begin{array}{c|cc} A_{44} & A_{46} & A_{47} \\ \hline A_{64} & A_{66}^4 & A_{67}^4 \\ \hline A_{74} & A_{76}^4 & A_{77}^4 \end{array}\right],$$

$$A_{55} = A_{55}^1 + A_{55}^2 \ , \ A_{66} = A_{66}^3 + A_{66}^4 \ , \ A_{77} = A_{77}^1 + A_{77}^2 \ + A_{77}^3 + A_{77}^4 \ .$$

Thus, the partial factorization of these submatrices can be computed concurrently. For example, computing the ILU of A_{11}, we obtain the following partial approximation

$$\left[\begin{array}{c|cc} \tilde{L}_{11} & 0 & 0 \\ \hline \tilde{L}_{51} & \mathrm{I} & 0 \\ \hline \tilde{L}_{71} & 0 & \mathrm{I} \end{array}\right] \left[\begin{array}{c|cc} \tilde{D}_{11} & 0 & 0 \\ \hline 0 & \tilde{S}_{55}^1 & \tilde{S}_{57}^1 \\ \hline 0 & \tilde{S}_{75}^1 & \tilde{S}_{77}^1 \end{array}\right] \left[\begin{array}{c|cc} \tilde{U}_{11} & \tilde{U}_{15} & \tilde{U}_{17} \\ \hline 0 & \mathrm{I} & 0 \\ \hline 0 & 0 & \mathrm{I} \end{array}\right] + \left[\begin{array}{c|cc} R_{11} & R_{15} & R_{17} \\ \hline R_{51} & 0 & 0 \\ \hline R_{71} & 0 & 0 \end{array}\right] .$$

When the partial factorizations of all the leaf nodes are completed, the processes in charge of these tasks send the local Schur complement to the corresponding intermediate node, which then accumulates them to continue the process,

$$\left[\begin{array}{c|c} \hat{S}^1_{55} & \hat{S}^1_{57} \\ \hline \hat{S}^1_{75} & \hat{S}^1_{77} \end{array}\right] + \left[\begin{array}{c|c} \hat{S}^2_{55} & \hat{S}^2_{57} \\ \hline \hat{S}^2_{75} & \hat{S}^2_{77} \end{array}\right] = \left[\begin{array}{c|c} \hat{S}_{55} & \hat{S}_{57} \\ \hline \hat{S}_{75} & \hat{S}^{12}_{77} \end{array}\right] \ ,$$

$$\left[\begin{array}{c|c} \hat{S}^3_{66} & \hat{S}^3_{67} \\ \hline \hat{S}^3_{76} & \hat{S}^3_{77} \end{array}\right] + \left[\begin{array}{c|c} \hat{S}^4_{66} & \hat{S}^4_{67} \\ \hline \hat{S}^4_{76} & \hat{S}^4_{77} \end{array}\right] = \left[\begin{array}{c|c} \hat{S}_{66} & \hat{S}_{67} \\ \hline \hat{S}_{76} & \hat{S}^{34}_{77} \end{array}\right] \ .$$

The matrix resulting from assembling these two submatrices presents the same structure as that defined in (3)

$$\left[\begin{array}{c|c} \hat{S}_{55} & \hat{S}_{57} \\ \hline \hat{S}_{75} & \hat{S}^{12}_{77} \end{array}\right] \oplus \left[\begin{array}{c|c} \hat{S}_{66} & \hat{S}_{67} \\ \hline \hat{S}_{76} & \hat{S}^{34}_{77} \end{array}\right] = \left[\begin{array}{cc|c} S_{55} & 0 & S_{57} \\ 0 & S_{66} & S_{67} \\ \hline S_{75} & S_{76} & S_{77} \end{array}\right] \ , \quad S_{77} = S^{12}_{77} + S^{34}_{77} \ , \qquad (8)$$

and the process continues as described above.

This procedure can be generalized to obtain the same number of leaf nodes as process/processors, so that the ILU factorization of each leaf node can be mapped to a specific process. The performance of the parallel computation of the preconditioner will be improved if the load is balanced among the leaf nodes is optimum and the communication time is reduced. Mapping nodes to processors as in Fig. 2, yields a high degree of parallelism if the computational cost is concentrated on the leaf nodes of the dependency tree, and the cost is evenly distributed among the leaf nodes. In practice, it is not possible to know the cost of the multilevel ILU factorization *a priori*, but we can estimate this cost from the number of rows/columns and nonzeros per node. Therefore, we must find a permutation of A that minimizes the number of rows/columns of non-leaf nodes, while simultaneously balancing those of the leaf nodes.

The MLND (*Multilevel Nested Dissection*) algorithm [8] is a recursive procedure that, at each step, splits a graph into two disjoint subgraphs connected by the nodes included in the separator. Some conditions hold for the result of this computation; e.g., the size of the separator may feature some minimum criteria and/or the size of the subgraphs can be made equal up to a certain degree. The recursion can be continued on the subgraphs until their size is relatively small. By viewing a sparse matrix as a graph, this procedure generates a reordered matrix similar to that shown in left-hand side of (7).

There exist several implementations of MLND (e.g., in METIS[2], SCOTCH[3]), which usually lead to balanced elimination trees that exhibit a higher degree of concurrency. There also exist parallel versions of these packages (ParMETIS [9] and PT-SCOTCH [6]) that exploit several types of parallelism during the computation of the permutation. To illustrate the quality of current partitioning packages, we applied ParMETIS to a benchmark matrix of dimension 10^6 (see Sect. 4 for details), in order to generate a tree with the structure shown in Fig. 2; the result is shown in Fig. 3.

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

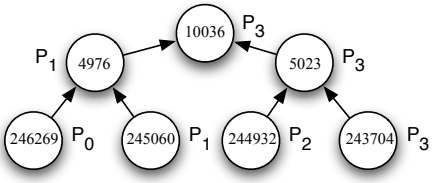

Fig. 3. Number of rows/columns of the dependency tree corresponding to a matrix of size 10^6 arising from the finite-difference discretization of the Laplace 3D PDE

The described parallel algorithm forces a certain order of elimination to expose a high degree of concurrence during the approximate factorization of the reordered matrix. In particular, the leaf nodes first factorize the leading diagonal blocks, and then the corresponding Schur complements are received and accumulated by the corresponding intermediate nodes. This idea is recursively applied till the root node is reached. However, this parallel process does not control the growth of the norms of the inverse triangular factors during the computation of the preconditioner. In order to accommodate the inverse-based preconditioning approach, we first restrict the **preprocessing** and **factorization** steps to the leading block of each submatrix, so that only the rows/columns of this block are preprocessed, and rejected rows/columns are moved to the bottom/right-end of the leading block. Thus, e.g., the **factorization** step applied to the left-most submatrix in (5) results in the following partial approximation,

$$\left[\begin{array}{c|c} P_{11} & 0 \\ \hline 0 & \mathrm{I} \end{array}\right]^T \left[\begin{array}{c|c} A_{11} & A_{13} \\ \hline A_{31} & A^1_{33} \end{array}\right] \left[\begin{array}{c|c} P_{11} & 0 \\ \hline 0 & \mathrm{I} \end{array}\right] = \left[\begin{array}{cc|c} B_{11} & F_{11} & F_{13} \\ E_{11} & C_{11} & C_{13} \\ \hline E_{31} & C_{31} & A^1_{33} \end{array}\right] =$$

$$\left[\begin{array}{cc|c} \tilde{L}_{B,11} & 0 & 0 \\ \tilde{L}_{E,11} & \mathrm{I} & 0 \\ \hline \tilde{L}_{E,31} & 0 & \mathrm{I} \end{array}\right] \left[\begin{array}{cc|c} \tilde{D}_{B,11} & 0 & 0 \\ 0 & \tilde{S}_{C,11} & \tilde{S}_{C,13} \\ \hline 0 & \tilde{S}_{C,31} & \tilde{S}^1_{C,33} \end{array}\right] \left[\begin{array}{cc|c} \tilde{U}_{B,11} & \tilde{U}_{F,11} & \tilde{U}_{F,13} \\ 0 & \mathrm{I} & 0 \\ \hline 0 & 0 & \mathrm{I} \end{array}\right] + \left[\begin{array}{cc|c} R_{B,11} & R_{F,11} & R_{F,13} \\ R_{E,11} & 0 & 0 \\ \hline R_{E,31} & 0 & 0 \end{array}\right] ,$$

and then the multilevel process is recursively applied on the matrix

$$\left[\begin{array}{c|c} \hat{S}_{C,11} & \hat{S}_{C,13} \\ \hline \hat{S}_{C,31} & \hat{S}^1_{C,33} \end{array}\right] = \left[\begin{array}{c|c} \tilde{S}_{C,11} & \tilde{S}_{C,13} \\ \hline \tilde{S}_{C,31} & \tilde{S}^1_{C,33} \end{array}\right] + \left[\begin{array}{c|c} R_{C,11} & R_{C,13} \\ \hline R_{C,31} & R^1_{C,33} \end{array}\right] . \tag{9}$$

Figure 4 illustrates the computation of the partial ILU factorization of A_{22} in (7), computed by a single leaf node of the dependency tree. The **restarting** step is also adapted, because it recursively applies the restricted steps until $\hat{S}_{C,11}$ in (9) is void or "sufficiently small". Finally, the intermediate node assembles the approximate Schur complement computed by its children as:

$$\left[\begin{array}{ccc} \hat{S}_{C,11} & 0 & \hat{S}_{C,13} \\ 0 & \hat{S}_{C,22} & \hat{S}_{C,23} \\ \hat{S}_{C,31} & \hat{S}_{C,32} & \hat{S}_{C,33} \end{array}\right] = \left[\begin{array}{c|c} \hat{S}_{C,11} & \hat{S}_{C,13} \\ \hline \hat{S}_{C,31} & \hat{S}^1_{C,33} \end{array}\right] \oplus \left[\begin{array}{c|c} \hat{S}_{C,22} & \hat{S}_{C,23} \\ \hline \hat{S}_{C,32} & \hat{S}^1_{C,33} \end{array}\right] .$$

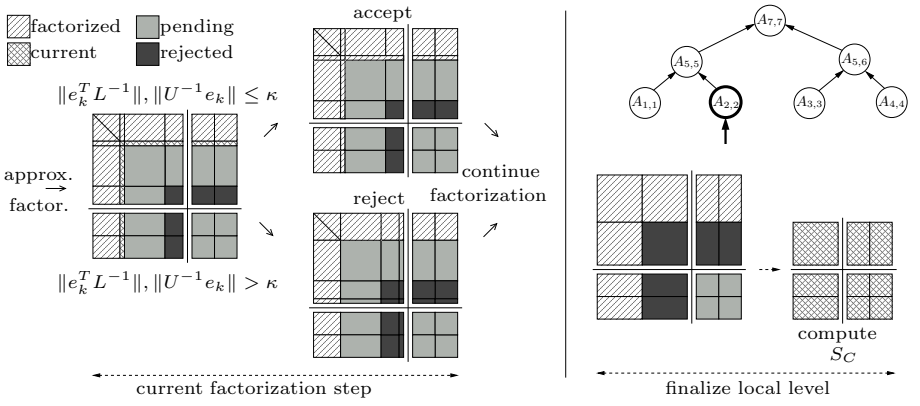

Fig. 4. Local incomplete factorization computed by a single node of the task tree

Unlike (6), the parent node must now consider the pivots rejected by its children, which are incorporated into the submatrix constructed by the former. This is illustrated in Tab. 1, which reports the structure of the parallel ILU factorization corresponding to the problem/tree in Fig. 3. We can observe that the leaves and the root have to build several partial inverse-based ILUs. Moreover, the number of levels and the number of rejected pivots in each leaf can be different.

Table 1. Number of accepted pivots by inverse-based pivoting in each level, and number of rejected pivots which are pushed upwards for the tree in Fig. 3

	Leaves						Intermediates		Root	
Proc	level1	level2	level3	level4	level5	rejected	level1	rejected	level1	level2
P_0	184977	33739	19065	8462		26				
P_1	184073	33529	17952	9291	215	0	5001	1		
P_2	183783	33123	18925	9070		31				
P_3	182922	33071	18608	9074		29	5083	0	9905	132
	735755	133462	74550	35897	215	86	10084	1	9905	132
	979879					86	10084	1	10037	

4 Experimental Results and Conclusions

All experiments in this section were obtained using IEEE double-precision arithmetic, on a cluster interconnected by an InfiniBand network with 4 nodes. Each node contains two Intel QuadCore *Nehalem* processors (8 cores), at 2.27 GHz and with 24 Gbytes of RAM. We used the OpenMPI message-passing library tuned for the InfiniBand network. The dependency tree was computed using ParMETIS (routine `ParMETIS_V3_NodeND` with defaults parameters).

We consider a standard benchmark problem for the solution of PDEs: the Laplacian equation $-\Delta u = f$ in a $3D$ unit cube $\Omega = [0,1]^3$ with Dirichlet

boundary conditions $u = g$ on $\partial\Omega$. Although this regular problem is known to be best-suited for multigrid methods, we have selected it due to its large dimension and applicability. The problem is discretized using a uniform mesh of size $h = \frac{1}{N+1}$. The computational domain Ω is replaced by a grid $\Omega_h = \{(x_i, y_j, z_k) = (ih, jh, kh) |\ i, j, k = 1, \ldots, N\}$, and the differential operator is replaced by finite differences

$$\Delta u\,(x_i, y_j, z_k) \approx \tfrac{1}{h^2}\,(-\,u_{i-1,j,k} - u_{i,j-1,k} - u_{i,j,k-1} + 6u_{i,j,k} - u_{i+1,j,k} - u_{i,j+1,k} - u_{i,j,k+1})\ ,$$

where $u_{ijk} \approx u(x_i, y_j, z_k)$. Because of the Dirichlet boundary conditions, any unknown u_{ijk} such that $i, j, k \in \{0, N+1\}$ is explicitly available and becomes part of the right-hand side vector. The resulting linear system $Au = b$ presents a sparse symmetric positive definite (SPD) coefficient matrix with seven nonzero elements per row, and $n = N^3$ unknowns. We set N=100, 126, 159, 200, and 252 in our experiments, which results in five SPD linear systems with roughly n =1, 2, 4, 8, and 16 millions of unknowns. We also consider four large-scale SPD benchmark matrices (bmwcra_1, af_shell3, ldoor and G3_circuit) from the UF sparse matrix collection[4]. We have selected these to evaluate the performance of our parallelization approach with irregularly structured problems.

Figure 5 shows the speed-up of the parallel ILU preconditioner for the different matrices, number of nodes, and number of cores per node. The total number of cores equals the product of the number of nodes and cores per node. The dependency tree is generated so that its number of leaves equals the number of cores. Thus, those combinations of "number of nodes-cores per node" which result in the same number of cores, utilize the same dependency tree to exploit parallelism, but a different mapping of MPI processes to cores. From this figure we can conclude that the parallel ILUPACK implementation exhibits reasonable strong scaling, as the parallel efficiency drops moderately as the number of cores grows. Moreover, the performance almost remains constant when the same number of cores are involved in the parallel computation, revealing a mild influence of the distribution of the cores among the nodes; only for the largest matrices, a small performance reduction is observed when using eight cores per node (see, e.g., for $N = 252^3$, drop from 4-4 to 2-8). We believe that this is due, to some extent, to contention caused by the fully utilization of the resources in a node.

At first glance, it might appear that the factor that contributes more to the drop in efficiency observed in Fig. 5 is the lack of parallelism in the higher levels of the dependency tree. Although this factor can (asymptotically) limit the strong scalability of our approach, in practice, we observed a moderate reduction of the computational cost concentrated on the leaves of the tree as the number of cores increases, so that even a single type of parallelism (e.g., tree parallelism) can provide a reasonable degree of parallelism for a multilevel ILU preconditioner. Table 2 reports the percentage of the aggregated computational cost which is concentrated on the leaves and the non-leaf nodes of the tree vs. the number of cores. This cost is defined as the aggregation of the computational cost of all

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

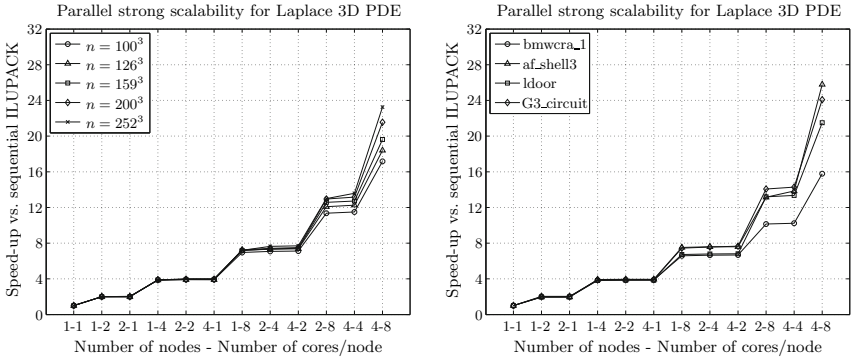

Fig. 5. Speed-up of the parallel multilevel ILU algorithm for the five Laplace 3D PDE matrices (left) and the four matrices from the UF sparse matrix collection (right)

tasks in the tree, so that overheads associated with parallelism (e.g., communication or idling) were not accounted in Tab. 2. The experiment clearly reveals a moderate reduction of the computational cost concentrated on the leaves with the number of cores for all matrices except bmwcra_1 (the smallest test matrix). For bmwcra_1, this reduction does not solely justify the performance drop observed in Fig. 5.

Table 2. Percentage of the aggregated computational cost which is concentrated on the leaves (left) and non-leaf nodes (right) of the dependency tree

matrix	1 core	2 cores	4 cores	8 cores	16 cores	32 cores
$n = 100^3$	(100.0,0.0)	(99.76,0.24)	(99.35,0.65)	(98.36,1.64)	(96.75,3.25)	(93.87,6.13)
$n = 126^3$	(100.0,0.0)	(99.84,0.16)	(99.46,0.54)	(98.71,1.29)	(97.25,2.75)	(95.20,4.80)
$n = 159^3$	(100.0,0.0)	(99.81,0.19)	(99.58,0.42)	(98.95,1.05)	(97.88,2.12)	(96.11,3.89)
$n = 200^3$	(100.0,0.0)	(99.90,0.10)	(99.63,0.37)	(99.15,0.85)	(98.24,1.76)	(96.93,3.07)
$n = 252^3$	(100.0,0.0)	(99.89,0.11)	(99.71,0.29)	(99.32,0.68)	(98.61,1.39)	(97.52,2.48)
bmwcra_1	(100.0,0.0)	(99.99,0.01)	(99.46,0.54)	(97.83,2.17)	(94.69,5.31)	(90.51,9.49)
af_shell3	(100.0,0.0)	(99.87,0.13)	(99.86,0.14)	(99.66,0.34)	(99.18,0.82)	(97.97,2.03)
ldoor	(100.0,0.0)	(99.96,0.04)	(99.74,0.26)	(99.27,0.73)	(98.76,1.24)	(97.58,2.42)
G3_circuit	(100.0,0.0)	(99.99,0.01)	(99.75,0.25)	(99.63,0.37)	(99.12,0.88)	(98.06,1.94)

We believe that the main key factor for the performance drop observed in Fig. 5 is the parallel overhead due to idle MPI processes, which in turn is caused by an unbalanced distribution of the computational work associated with the leaf nodes in the tree. Table 3 reports, for a parallel execution with 32 cores, how much computational time is concentrated on the most and least expensive computational leaves; this is expressed as a percentage relative to the parallel execution time in the rows labeled as $leaf_{max}$ and $leaf_{min}$. This table also reports the aggregated parallel overhead relative to the aggregated parallel execution time, with the latter defined as the product of the parallel execution

time and the number of cores. The aggregated parallel overhead was estimated by substracting the aggregated parallel execution time and the aggregation of the computational costs of all tasks in the tree (i.e., the useful computation). The table clearly correlates load unbalance in the computation of the leaves and parallel overhead; see e.g., values for bmwcra_1. Future developments will require additional techniques to improve load balancing in the computation of the leaves.

Table 3. Amount of computational time concentrated on the most and least computationally expensive leaves, and relative aggregated parallel overhead

	100^3	126^3	159^3	200^3	252^3	bmwcra_1	af_shell3	ldoor	G3_circuit
$leaf_{max}$	65.28	68.61	73.28	77.75	80.80	60.27	87.67	82.50	84.38
$leaf_{min}$	50.00	53.28	58.40	64.03	68.98	27.40	53.42	62.50	71.88
overhead	39.15	35.83	32.18	26.47	23.01	54.88	36.77	26.80	19.63

Acknowledgments. This research has been supported by the CICYT project TIN2008-06570-C04-01 and by the Fundación Caixa-Castelló/Bancaixa and UJI project P1-1B2009-31.

References

1. Aliaga, J.I., Bollhöfer, M., Martín, A.F., Quintana-Ortí, E.S.: Parallelization of multilevel preconditioners constructed from inverse-based ILUs on shared-memory multiprocessors. In: Parallel Computing: Architectures, Algorithms and Applications. Advances in Parallel Computing, vol. 38, pp. 287–294. NIC (2007)
2. Aliaga, J.I., Bollhöfer, M., Martín, A.F., Quintana-Ortí, E.S.: Design, Tuning and Evaluation of Parallel Multilevel ILU Preconditioners. In: Palma, J.M.L.M., Amestoy, P.R., Daydé, M., Mattoso, M., Lopes, J.C. (eds.) VECPAR 2008. LNCS, vol. 5336, pp. 314–327. Springer, Heidelberg (2008)
3. Aliaga, J.I., Bollhöfer, M., Martín, A.F., Quintana-Ortí, E.S.: Exploiting thread-level parallelism in the iterative solution of sparse linear systems. Parallel Computing (2010) (in press, accepted manuscript)
4. Bollhöfer, M., Grote, M.J., Schenk, O.: Algebraic multilevel preconditioner for the helmholtz equation in heterogeneous media. SIAM Journal on Scientific Computing 31(5), 3781–3805 (2009)
5. Bollhöfer, M., Saad, Y.: Multilevel preconditioners constructed from inverse–based ILUs. SIAM J. Sci. Comput. 27(5), 1627–1650 (2006); special issue on the 8–th Copper Mountain Conference on Iterative Methods
6. Chevalier, C., Pellegrini, F.: PT-SCOTCH: A tool for efficient parallel graph ordering. Parallel Comput. 34(6-8), 318–331 (2008)
7. Davis, T.A.: Direct Methods for Sparse Linear Systems. SIAM (2006)
8. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. 20(1), 359–392 (1998)
9. Karypis, G., Kumar, V.: A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. J. Parallel Distrib. Comput. 48(1), 71–95 (1998)
10. Saad, Y.: Iterative Methods for Sparse Linear Systems. SIAM Publications (2003)