A class of multilevel parallel preconditioning strategies

Laura Grigori, Pawan Kumar, Frédéric Nataf, Ke Wang

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A class of multilevel parallel preconditioning strategies

Laura Grigori — Pawan Kumar — Frederic Nataf — Ke Wang

N° 7410

Octobre 2010

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--7410--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SACLAY - ÎLE-DE-FRANCE

A class of multilevel parallel preconditioning strategies

Laura Grigori* , Pawan Kumar† , Frederic Nataf‡ , Ke Wang§

Thème NUM — Systèmes numériques
Équipe-Projet Grand-large

Rapport de recherche n° 7410 — Octobre 2010 — 25 pages

Abstract: In this paper, we introduce a class of recursive multilevel preconditioning strategies suited for solving large sparse linear systems of equations on modern day architectures. They are based on a reordering of the input matrix into a nested bordered block diagonal form, which allows a nested formulation of the preconditioners. The first one, which we refer to as nested SSOR (NSSOR), requires only the factorization of diagonal blocks at the innermost level of the recursive formulation. Hence, its construction is embarassingly parallel, and the memory requirements are very limited. Next two are nested versions of Modified ILU preconditioner with row sum (NMILUR) and colsum (NMILUC) property. We compare these methods in terms of iteration number, memory requirements, and overall solve time, with ILU(0) with natural ordering and nested dissection ordering, and MILU. We find that NSSOR compares favorably with ILU(0) with nested dissection ordering, while NMILUR and NMILUC outperform the other methods for certain matrices in our test set.

It is proved that the NSSOR method is convergent when the input matrix is SPD. The preconditioners are designed to be suitable for parallel computing.

Key-words: preconditioner, linear system, GMRES, nested structure, incomplete LU

* INRIA Saclay - Ile de France, Laboratoire de Recherche en Informatique, Université Paris-Sud 11, France (Email: Laura.grigori@inria.fr). Part of the work of this author has been supported by French National Research Agency (ANR) through COSINUS program (project PETAL no ANR-08-COSI-009).

† INRIA Saclay - Ile de France, Laboratoire de Recherche en Informatique, Université Paris-Sud 11, France (Email: pawan.kumar@lri.fr)

‡ Laboratoire J. L. Lions, CNRS UMR7598, Université Paris 6, France, (Email: nataf@ann.jussieu.fr). Part of the work of this author has been supported by French National Research Agency (ANR) through COSINUS program (project PETAL no ANR-08-COSI-009).

§ Department of Mathematics, College of Sciences, Shanghai University, Shanghai 200444, P.R. China (Email: kwang@shu.edu.cn). The work of this author was performed as a postdoctoral researcher at INRIA Saclay, funded by French National Research Agency (ANR) through COSINUS program (project PETAL no ANR-08-COSI-009

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université
4, rue Jacques Monod, 91893 ORSAY Cedex
Téléphone : +33 1 72 92 59 00

Une classe de préconditionneurs parallèles multiniveaux

Résumé : Dans ce papier nous décrivons une classe de préconditionneurs multiniveaux parallèles pour résoudre des systèmes linéaires de grande taille. Ils se basent sur une renumérotation de la matrice d'entrée en forme block diagonale bornée et emboîtée, qui permet une définition emboîtée des préconditionneurs.

Nous prouvons qu'un des préconditionneurs, NSSOR, converge quand la matrice d'entrée est symmétrique et définie positive. Les préconditionneurs sont adaptés au calcul parallèle.

Mots-clés : système linéaire, préconditionneur, GMRES, factorisation LU incomplète

1 Introduction

The problem of finding an approximate solution of a large sparse linear system of the form

$$Ax = b \tag{1}$$

is an important operation in many scientific applications. Consequently, a considerable amount of research focuses on iterative methods, in particular on Krylov subspace type methods [15]. The convergence of these methods mainly depends on how well the given problem is preconditioned. Given a linear system of the form (1) above, we seek an approximation B to A and consider the following preconditioned problem

$$B^{-1}Ax = B^{-1}b, \tag{2}$$

where B is chosen such that the spectrum of $B^{-1}A$ is "favorable" for the Krylov subspace type methods [15].

A particular class of methods for solving sparse linear systems of equations are the algebraic multigrid methods [14, 18, 20]. They have proved to be successful for certain classes of problems as for example elliptic type PDEs. However, they can involve a high setup cost and hence other alternatives can be sometimes more efficient. Preconditioned Krylov subspace methods with incomplete LU (ILU) (see for example [2, 15]) as preconditioner are designed to be general purpose methods for solving arbitrary sparse linear systems of equations. They tend to work on problems where the above methods fail. However, the main drawback of ILU type preconditioners are their poor convergence rate with increasing problem size. Moreover, they usually need tuning of parameters for different problem types. Recently, several multilevel methods based on ILU have been designed, that use a combination of techniques from direct and iterative methods [6, 9, 16]. Another unique approach is based on a direct approximation of the error propagation matrix, namely $I - B^{-1}A$. This method commonly known as SPAI (see for example [3, 4]) is very promising.

In this work we introduce and compare three recursive multilevel parallel preconditioners, which are shown to be efficient for a range of problems. The new preconditioners are the following:

- NSSOR : nested SSOR
- NMILUR : nested MILU with rowsum constraint
- NMILUC : nested MILU with colsum constraint

The methods consider that the input matrix has a nested bordered block diagonal structure, which allows a nested definition of the preconditioner. In addition it is suitable for parallel computation. The methods can be seen as a multilevel extension of classical preconditioners as SSOR and modified ILU (MILU) (see for example [2, 15]).

The method of NSSOR is built by approximating the Schur complements simply by the diagonal blocks of the original matrix. This preconditioner is attractive because the construction cost and storage requirements are relatively minimal. Moreover its construction is embarassingly parallel.

The methods NMILUR and NMILUC can be seen as nested extensions of the classic MILU method with a rowsum or a colsum property. They can also be seen as an extension of the Nested Factorization preconditioner introduced in [1] for matrices arising from the discretization of PDEs on structured grids. A relaxed version of Nested Factorization is presented in [12]. The NMILUR and NMILUC preconditioners satisfy the rowsum and colsum property respectively. We say that a preconditioner B satisfies rowsum filtering property for a given vector $\mathbf{1} = [1, 1, \ldots, 1]$ if the following holds:

$$A\mathbf{1}^T = B\mathbf{1}^T \tag{3}$$

On the other hand, colsum property is defined as follows

$$\mathbf{1}A = \mathbf{1}B \tag{4}$$

It is proved that the NSSOR method is convergent for a given SPD problem. We also show that NMILUR and NMILUC satisfy the respective filtering properties.

The nested bordered block diagonal form can be obtained by k-way partitioning techniques, as for example implemented in Metis graph partitionning package [10]. In addition of allowing a recursive formulation of the preconditioner, this form has several advantages. First, it allows for a parallel construction and application of the preconditioner in the iterative process. Second, such reordering creates a structure which itself presents several advantages. One of the key steps in solving linear systems of the form (1) via iterative

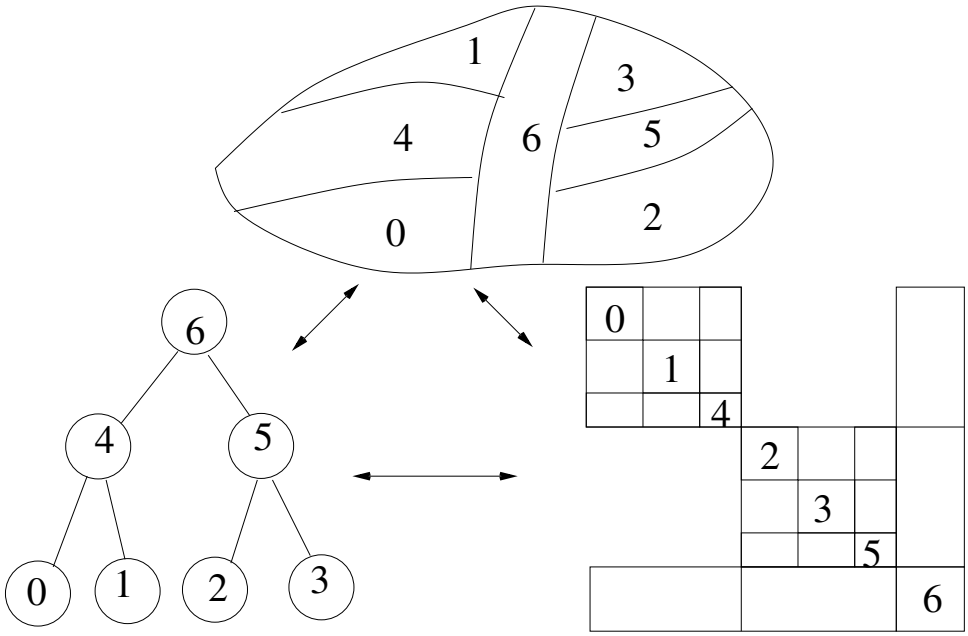

Figure 1: Graph partitioned into 4 parts, its corresponding separator tree and matrix obtained after reordering.

methods is the multiplication of a sparse matrix with a dense vector. The reordering based on k-way partitioning helps in minimizing communication for this critical matrix vector operation for distributed memory model [8]. It also creates data locality, which is important for exploiting well the memory hierarchy and for shared memory machines. Third, the ordering reduces the fill incured by the factorization of the diagonal blocks, and hence it is important as in the case of direct methods.

The methods are compared with ILU using natural ordering, ILU using 2-way nested dissection ordering, and MILU preconditioners. Our test matrices include matrices arising from convection-diffusion problems on structured grids, finite element problems generated by FreeFEM++ package [13] on unstructured meshes, and several problems from University of Florida Sparse Matrix Collection [5]. We find that NSSOR compares favorably with ILU based on nested dissection. When applied to matrices arising from problems with smooth coefficients (see the results for matrices airfoil2d, chipcool1, Lap1, Lap2, mat_mem, and mat_heat in Tables 4, 5), the NMLIUR/C methods perform very well. They are strongly scalable and this is most probably due to the filtering property.

We note that domain decomposition methods are also well suited for parallelism and are very efficient for many classes of problems. The methods discussed in this paper have the same level of parallelism. In addition, the methods NMILUR and NMILUC are relatively stable with respect to the number of domains, similar to scalable domain decomposition methods. While in our case this is due to satisfying a filtering property, in domain decomposition this is ensured by using a coarse grid correction [19, 17].

The paper is organized as follows. In section 2, we explain the partitioning and reordering used to define the methods proposed. We focus on the usage of 2-way nested dissection. In section 3, we discuss and explain the methods in detail. In section 4, we prove some results for these methods. Finally, in section 5 a comparison is done for all the methods discussed.

2 Partitioning, reordering, and associated notations

The multilevel preconditioners presented in this paper consider that the input matrix has a structure referred to as nested bordered block diagonal form. This structure can be obtained by reordering the matrix based on nested k-way partitioning (see for example [11]), that we present briefly in this section.

The undirected graph $G(V, E)$ of a symmetric matrix A of size $n \times n$ is formed by a set of n vertices V and a set of edges E. There is a vertex $v_i \in V$ for each row/column i of A and an edge (v_i, v_j) for each nonzero element A_{ij}. A subset S of V is called a vertex separator of G, if the removal of all the nodes of S from the graph leaves the graph disconnected into two or more components. In k-way partitioning, a separator S is found that separates the graph into k disconnected components. Each of these components

can be further divided into k disconnected components, and this process can be repeated for desired number of times. In this paper we consider the case when $k = 2$, and this partitioning technique is referred to as nested dissection. It is implemented for example in METIS graph partitioning library [10].

Figure (1) shows a pictorial representation of a graph corresponding to nested dissection ordering and its associated separator tree. On the right, the matrix corresponding to the graph is shown. In the separator tree, the nodes numbered 6, 5, and 4 correspond to separators. Once a separator is found we would like to group them together by renumbering the nodes. For our purpose, we are concerned only with the undirected graph or symmetric matrices and the actual weights (values in the matrix) are irrelevant. Renumbering the nodes in a matrix means a symmetric permutation of the rows and columns such that the resulting adjacency graph remains isomorphic to the original one i.e., no new nodes or edges are created or destroyed. In figure 1, the separator block number 6 is numbered last after the left subtree of nodes in {4, 0, 1} and right subtree made up of {5, 2, 3 } are numbered. This process of finding the separator nodes and later renumbering them last after the two disconnected components have been numbered leads to the matrix with special structure as shown in the Figure (1) on the left.

We now introduce a general convenient notation. After obtaining a suitable separator and renumbering the child nodes and subsequently renumbering the separator nodes we obtain a bordered block diagonal matrix

$$P^T AP = \begin{pmatrix} T_1^1 & & U_1^1 \\ & T_1^2 & U_1^2 \\ L_1^1 & L_1^2 & S_1^1 \end{pmatrix}. \tag{5}$$

Here P is the symmetric permutation matrix that renumbers the nodes. The interior nodes of the separator tree obtained after nested dissetion have two children, namely, left and right child. The block S_1^1 corresponds to the vertices of the separator associated with the root node. Here the subscript refers to the level of dissection 1 in the separator tree, and 1 refers to the number of the node at this level. The blocks corresponding to the two independent partitions obtained after one dissection are T_1^1 and T_2^1 respectively. These blocks are associated with the two children nodes of the root node. Each of these children nodes are connected via lower blocks L_1^1 and L_2^1 and upper blocks U_1^1 and U_2^1 to the root separator. In the notation of T, L, and U, the subscript correspnds to the level of dissection in the separator tree, and the superscript refers to the number of the child node, that is the number of the node in the next level of the separator tree.

For a matrix with nested bordered block diagonal form, the blocks T_1^1 and T_1^2 have recursively a bordered block diagonal form. The matrix is denoted recursively in a similar fashion for each node and their children.

For K levels of nested dissection, that is a separator tree of height K, we denote by $L_k, U_k, 1 \leq k \leq K$ the lower and upper matrices of same size as original matrix A, where only those blocks which represent the connection between the separators of level k and $k+1$ are present. Also, we denote by D the block diagonal part of A. The additive decomposition of A can be written as

$$P^T AP = D + \sum_{k=1}^{K} (L_k + U_k).$$

If we consider two levels of nested dissection, the permuted matrix has the following structure

$$P^T AP = D + \sum_{k=1}^{2} (L_k + U_k) = \left(\begin{array}{cc|cc|c} T_2^1 & & U_2^1 & & \\ & T_2^2 & U_2^2 & & U_1^1 \\ \hline L_2^1 & L_2^2 & S_2^1 & & \\ \hline & & & T_2^3 \quad U_2^3 & \\ & & & T_2^4 \quad U_2^4 & U_1^2 \\ & & & L_2^3 \quad L_2^4 \quad S_2^2 & \\ \hline \multicolumn{2}{c|}{L_1^1} & \multicolumn{2}{c|}{L_1^2} & S_1^1 \end{array}\right), \tag{6}$$

where

$$D = \left(\begin{array}{cc|cc|c} T_2^1 & & & & \\ & T_2^2 & & & \\ \hline & & S_2^1 & & \\ \hline & & & T_2^3 & \\ & & & & T_2^4 \\ \hline & & & & S_2^2 \\ \hline & & & & S_1^1 \end{array}\right),$$

$$L_1 = \left(\begin{array}{cc|c|cc|c} 0 & & & & & \\ & 0 & & & & \\ \hline & & 0 & & & \\ \hline & & & 0 & & \\ & & & & 0 & \\ \hline & & & & & \\ \hline L_1^1 & & & L_1^2 & & 0 \end{array}\right), L_2 = \left(\begin{array}{cc|c|cc|c} 0 & & & & & \\ & 0 & & & & \\ \hline L_2^1 & L_2^2 & 0 & & & \\ \hline & & & 0 & & \\ & & & & 0 & \\ \hline & & & L_2^3 & L_2^4 & 0 \\ \hline & & & & & 0 \end{array}\right),$$

and U_1, U_2 are defined in a similar way to L_1, L_2 respectively.

3 Nested preconditioners

In this section we present the nested preconditioners in detail. They are based on a nested bordered block diagonal form of the input matrix A. This form can be obtained by partitionning and reordering techniques presented in section 2, which are for example implemented in the METIS graph partitioner [10]. We first describe the algebra of an exact decomposition of the input matrix A, on which our preconditioners are based. For the ease of understanding, we condiser that the input matrix has 2 independent domains, a form that can be obtained by 2-way nested dissection partitionning and reordering. However the methods are easily generalized to a nested bordered block diagonal form with any number of diagonal blocks at each level of the partitioning.

3.1 Nested exact factorization

We consider first a matrix that has a bordered block diagonal form obtained after applying one level of nested dissection, as follows:

$$A = L_1 + D + U_1 = \begin{pmatrix} T_1^1 & & U_1^1 \\ & T_1^2 & U_1^2 \\ L_1^1 & L_1^2 & S_1^1 \end{pmatrix}.$$

where

$$D = \begin{pmatrix} T_1^1 & & \\ & T_1^2 & \\ & & S_1^1 \end{pmatrix}, L_1 = \begin{pmatrix} 0 & & \\ & 0 & \\ L_1^1 & L_1^2 & 0 \end{pmatrix}, U_1 = \begin{pmatrix} 0 & & U_1^1 \\ & 0 & U_1^2 \\ & & 0 \end{pmatrix}.$$

An exact factorization of A can be written as

$$\begin{aligned} A &= (L_1 + F_1)F_1^{-1}(F_1 + U_1) - L_1F_1^{-1}U_1, \\ F_1 &= D. \end{aligned}$$

Note that due to the bordered structure of A, the Schur complement $L_1F_1^{-1}U_1$ uses the first two diagonal blocks of F_1 and modifies the third diagonal block of F_1. Hence, the exact factorization of A can also be written as

$$A = (L_1 + F_1)F_1^{-1}(F_1 + U_1), \quad (7)$$
$$F_1 = D - L_1F_1^{-1}U_1, \quad (8)$$

where F_1 has the following block diagonal structure:

$$F_1 = \begin{pmatrix} T_1^1 & & \\ & T_1^2 & \\ & & S_1^1 - L_1^1(T_1^1)^{-1}U_1^1 - L_1^2(T_1^2)^{-1}U_1^2 \end{pmatrix}$$

The latter decomposition in equations (7) (8) makes more apparent the factored form of A, which is usefull in solving equations of the form $Ax = b$, and this approach will be used in the rest of the paper.

Consider that the matrix A has a nested bordered block diagonal structure, that is the submatrices T_1^1 and T_1^2 have themselves a nested bordered block diagonal structure. With the notation introduced in section 2, the additive decomposition of A is given as $A = D + \sum_{k=1}^{K}(L_k + U_k)$, and its exact factorization can be written in a nested way as following:

$$\begin{aligned} A &= F_0, && (9)\\ F_k &= (L_{k+1} + F_{k+1})F_{k+1}^{-1}(F_{k+1} + U_{k+1}),\ for\ k = 0 \dots K-1 && (10)\\ F_K &= D - \sum_{k=1}^{K} L_k F_k^{-1} U_k. && (11) \end{aligned}$$

This recursive procedure can start from a certain level ($k \geq 1$). Once all the Schur complements are computed in the expression of F_K, we get a factorization of A in terms of block lower and block upper factors.

We note that the expression of F_K involves terms of the recurrence F_k for $k = 1 \dots K$. The computation is possible due to the structure of L_k, F_k, and U_k matrices. The factored form of F_{k+1} gives to each of the subdomains T_k a factored form in terms of block lower triangular and block upper triangular factors. Then the Schur complement $L_k F_k^{-1} U_k$ can be computed, it uses the factored forms of the blocks corresponding to T_k, and modifies the blocks corresponding to the separator S_k. The computation will become more clear shortly, when it will be illustrated in the context of the proposed preconditioners.

3.2 Preconditioners

We introduce now two types of preconditioners, that we refer to as NSSOR and NMILU. In an exact factorization, the blocks F_k need to be inverted to compute the Schur complements in equation (11). These inversions introduce fill in the matrix F_k, and are costly in terms of both storage requirements and computation time. Hence the goal of a nested preconditioner is to find suitable approximations to the inverse of matrices F_k in equations (10) and (11).

Definition 3.1 *Let A be a matrix of size $n \times n$ which has a nested bordered block diagonal structure and whose additive decomposition can be written as $A = D + \sum_{k=1}^{K}(L_k + U_k)$. A nested SSOR preconditioner B_{NSSOR} is defined as*

$$\begin{aligned} B_{NSSOR} &= G_0, && (12)\\ G_k &= (L_{k+1} + G_{k+1})G_{k+1}^{-1}(G_{k+1} + U_{k+1}),\ for\ k = 0 \dots K-1 && (13)\\ G_K &= D, && (14) \end{aligned}$$

where we suppose that the matrices G_k for $k = 1, \dots, K$ are invertible.

In the NSSOR preconditioner, the Schur complements which appear in an exact factorization in equation (11) are simply dropped. That is, there is no explicit coupling term in between the different partitions T_k at different levels k of the nested factorization. We note that NSSOR is relatively cheap to compute, and highly parallel. In fact, once the diagonal blocks D are factored, the preconditioner is ready to be applied in the iterative solver. We describe later in Algorithm (1) the details of the construction of the preconditioner.

We now introduce two variants of a nested modified ILU preconditioner. The first variant, NMILUR, ensures that the rowsum property is satisfied, that is $\mathbf{1}A = \mathbf{1}B_{NMILUR}$. The second variant, NMILUC, ensures that the colsum property is satisfied, that is $A\mathbf{1}^T = B_{NMILUC}\mathbf{1}^T$. We give formal proofs for these properties in the analysis section 4. In the following, $Diag(v)$ is the diagonal matrix formed from vector v.

Definition 3.2 *Let A be a matrix of size $n \times n$ which has a nested bordered block diagonal structure and whose additive decomposition is written as $A = D + \sum_{k=1}^{K}(L_k + U_k)$. An NMILU preconditioner B_{NMILU} is defined as*

$$\begin{aligned} B_{NMILU} &= G_0, && (15)\\ G_k &= (L_{k+1} + G_{k+1})G_{k+1}^{-1}(G_{k+1} + U_{k+1}),\ for\ k = 0 \dots K-1 && (16)\\ G_K &= D - \sum_{k=1}^{K} H_k && (17) \end{aligned}$$

where we suppose that the matrices G_k, for $k = 1, \ldots, K$, are invertible. For NMILUR preconditioner, the matrices H_k are defined as

$$H_k = rowsum(L_k G_k^{-1} U_k) = Diag(L_k G_k^{-1} U_k \mathbf{1}),\ for\ k = 1, \ldots, K, \tag{18}$$

while for NMILUC preconditioner, the matrices H_k are defined as

$$H_k = colsum(L_k G_k^{-1} U_k) = Diag(\mathbf{1}^T L_k G_k^{-1} U_k),\ for\ k = 1, \ldots, K \tag{19}$$

where $\mathbf{1} = [1, 1, \ldots, 1]$ is a vector of appropriate dimension.

Consider a level $k \geq 1$ of the computation of NMILU preconditioner. Two approximations are used. The approximation of the factorization of the blocks correspoding to different parts of T_k is given by the expression of G_{k+1}. The Schur complements involved in the computation of the blocks of S_k that couple the domains of T_k are approximated by the formulas (18), (19).

In this paper, we use the same approach as in modified ILU preconditioner, in which the terms dropped are added to the diagonal of the preconditioner, such that the rowsum or the colsum property is satisfied. However, other approximations can be used for the inverse of G_k matrices, as for example the approximation presented in [7].

3.3 Algorithms for nested preconditioners

We present algorithms to compute the NSSOR and NMILU preconditioners and to apply them during the iterative process. Although they are sequential, we also outline the parallelism available in this computation.

Algorithm 1 BuildNSSOR($T, level, K$): recursive procedure to build B_{NSSOR} preconditioner for a matrix A.

Input: K is the height of the separator tree obtained from nested dissection, *level* is the current level of computation in the separator tree. If $level \leq K$, then T is partitioned as

$$T = \begin{pmatrix} T^1 & & U^1 \\ & T^2 & U^2 \\ L^1 & L^2 & S^1 \end{pmatrix}$$

Output: Updated factored form of G_K
if $level > K$ **then**
  Factor(T)
else
  BuildNSSOR($T^1, level + 1, K$)
  BuildNSSOR($T^2, level + 1, K$)
  Factor(S^1)
end if

Algorithms 1 and 2 present the construction of NSSOR and NMILUR preconditioners in a recursive manner, as a postorder traversal of the separator tree. The construction of NMILUC is similar to NMILUR. In both algorithms, *Factor* stands for exact factorization, but in practice an incomplete factorization can be used.

NSSOR preconditioner is build by a call to $BuildNSSOR(A, 1, K)$ in algorithm 1, where K is the height of the separator tree obtained from nested dissection. Once the diagonal blocks corresponding to $G_K = D$ matrix (equation (14)) are factored, the preconditioner is ready to be applied in the iterative solver. Since the factorizations of the diagonal blocks are independent computations, this algorithm is embarassingly parallel.

NMILUR preconditioner is computed through a call to $BuildNMILUR(A, 1, K)$ in Algorithm 2, where K is the height of the separator tree obtained from nested dissection. At each level k of the recursion, the input matrix T has a bordered block diagonal form. The goal is to compute an approximate factorization $\tilde{T}$ of T. This is achieved by computing approximate factorizations of the two subdomains T^1, T^2 through recursive calls to $BuildNMILUR$. Then the approximate Schur complement for matrix S^1 is computed, which corresponds to a diagonal block of matrix G_K.

The solution procedure for solving with these nested preconditioners is the same, once the factored form of matrix G_K in equations 14 and 17 is computed. The solve will involve calling recursively the forward and backward sweep routines shown in Algorithms (3) and (4) respectively.

Algorithm 2 BuildNMILUR($T, level, K$): recursive procedure to build B_{NMILUR} preconditioner for a matrix A.

Input: K is the height of the separator tree obtained from nested dissection, $level$ is the current level of computation in the separator tree. If $level \leq K$, then T is partitioned as

$$T = \begin{pmatrix} T^1 & & U^1 \\ & T^2 & U^2 \\ L^1 & L^2 & S^1 \end{pmatrix}$$

Output: Updated factored form of G_K, approximate factorization $\tilde{T}$ of T
if $level > K$ **then**
$\tilde{T} = Factor(T)$
else
Call BuildNMILUR($T^1, level-1, K$) to compute the factored form $\tilde{T}^1$
Call BuildNMILUR($T^2, level-1, K$) to compute the factored form $\tilde{T}^2$
Compute Schur complement

$$\tilde{S}^1 \quad = \quad S^1 - rowsum(L^1(\tilde{T}^1)^{-1}U^1) - rowsum(L^2(\tilde{T}^2)^{-1}U^2)$$

Let $\tilde{T}$ be formed as

$$\tilde{T} = \begin{pmatrix} \tilde{T}^1 & & \\ & \tilde{T}^2 & \\ L^1 & L^2 & \tilde{S}^1 \end{pmatrix} \cdot \begin{pmatrix} \tilde{T}^1 & & \\ & \tilde{T}^2 & \\ & & \tilde{S}^1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} \tilde{T}^1 & & U^1 \\ & \tilde{T}^2 & U^2 \\ & & \tilde{S}^1 \end{pmatrix}$$

end if

We describe more in detail the solution procedure for NSSOR preconditioner. Recall that $B_{NSSOR} = (L_1 + G_1)(I + G_1^{-1}U_1) = B_L B_U$, where G_1 has recursively a similar factored form until some level. The solve $B_{NSSOR}x = b$ is computed by a call to ForwardSweep $(B_L, b, 1, K)$, followed by a call to BackwardSeep$(B_U, y, 1, K)$, where K is the height of the separator tree. We notice here that for both forward and backward sweep, the solve with the subdomains $\tilde{T}^1$ and $\tilde{T}^2$ involve a recursive call to forward and backward sweeps within these subdomains. The recursion stops when the last level of the multilevel factorization of B_{NSSOR} is attained. At each level of the factorization, the solves within the two subdomains $\tilde{T}^1$ and $\tilde{T}^2$ can be performed in parallel.

Algorithm 3 ForwardSweep($\tilde{T}_L, \bar{b}, level, K$): recursive procedure to solve $\tilde{T}_L\bar{y} = \bar{b}$ using nested preconditioner $B = B_L B_U$. This routine is used when solving the equation $B_L B_U x = b$, where $B_U x = y$.

Input: K is the height of the separator tree obtained from nested dissection, $level$ is the current level of computation in the separator tree. If $level \leq K$, then the procedure solves the system

$$\tilde{T}_L\bar{y} = \bar{b},\ that\ is \quad \begin{pmatrix} \tilde{T}^1 & & \\ & \tilde{T}^2 & \\ L^1 & L^2 & \tilde{S}^1 \end{pmatrix} \cdot \begin{pmatrix} \bar{y}^1 \\ \bar{y}^2 \\ \bar{y}^3 \end{pmatrix} = \begin{pmatrix} \bar{b}^1 \\ \bar{b}^2 \\ \bar{b}^3 \end{pmatrix}.$$

Output: $\bar{y}$
if $level > K$ **then**
Solve $\tilde{T}_L\bar{y} = \bar{b}$
else
Solve $\tilde{T}^1\bar{y}^1 = \bar{b}^1$ by calling ForwardSweep$(\tilde{T}^1, \bar{b}^1, level+1, K)$
Solve $\tilde{T}^2\bar{y}^2 = \bar{b}^2$ by calling ForwardSweep$(\tilde{T}^2, \bar{b}^2, level+1, K)$
Solve $\tilde{S}^1\bar{y}^3 = \bar{b}^3 - L^1\bar{y}^1 - L^2\bar{y}^2$
end if

4 Analysis

In this section, we collect some of the results on the methods presented in section 3. By SPD we shall mean symmetric positive definite matrix. The input matrix A is reordered using techniques described in section

Algorithm 4 BackwardSweep($\tilde{T}_U, \bar{y}, level, K$): recursive procedure to solve $\tilde{T}_U \bar{x} = \bar{y}$ using nested preconditioner $B = B_L B_U$. This routine is used when solving the equation $B_L B_U x = b$, where $B_U x = y$.

Input: K is the height of the separator tree obtained from nested dissection, *level* is the current level of computation in the separator tree. If $level \leq K$, then the procedure solves the system

$$\tilde{T}_U \bar{x} = \bar{y}, \text{ that is } \begin{pmatrix} I & & (\tilde{T}^1)^{-1}U^1 \\ & I & (\tilde{T}^2)^{-1}U^2 \\ & & I \end{pmatrix} \cdot \begin{pmatrix} \bar{x}^1 \\ \bar{x}^2 \\ \bar{x}^3 \end{pmatrix} = \begin{pmatrix} \bar{y}^1 \\ \bar{y}^2 \\ \bar{y}^3 \end{pmatrix}.$$

Output: $\bar{x}$
if $level > K$ **then**
  Solve $\tilde{T}_U \bar{x} = \bar{y}$
else
  Set $\bar{x}^3 = \bar{y}^3$
  Find $\bar{x}^1 = \bar{y}^1 - (\tilde{T}^1)^{-1}U^1\bar{x}^3$.
  Find $\bar{x}^2 = \bar{y}^2 - (\tilde{T}^2)^{-1}U^2\bar{x}^2$.
end if

2 into a nested bordered block diagonal form. Let K be the height of the separator tree of nested dissection ordering. For example in Figure (1), K is equal to 2. The node {6} is situated at level 1, the nodes {4,5} are at level 2 and subsequently the leaf nodes {0,1,2,3} are at level 3 in the separator tree. Using the notations introduced in section 2, $P^T AP$ has the following additive decomposition

$$P^T AP = D + \sum_{k=1}^{K} (L_k + U_k).$$

Further, recall that **1** denotes [1,1,...,1].

Theorem 4.1 *For a given SPD matrix A, the NSSOR preconditioner B_{NSSOR} is SPD and $\rho(B_{NSSOR}^{-1} P^T AP) < 1$.*

Proof: The preconditioner B_{NSSOR} can be seen as a multilevel preconditioner as in definition 3.1. If the original matrix A is SPD, then $P^T AP$ is SPD. The proof of the theorem follows from definition, i.e., we have $(P^T APx, Px) = (APx, Px) > 0, \ for\ x \neq 0$. Notice that $P^2 = I$ since P is a permutation matrix, and hence P is non-singular thus $Px \neq 0, for\ x \neq 0$.

Consequently, $G_K = D$ is SPD. We have

$$G_k = (L_{k+1} + G_{k+1}) G_{k+1}^{-1} (G_{k+1} + L_{k+1}^T),$$

It is easy to see that G_{k+1} is symmetric if A is symmetric. Also, G_{k+1} is a block diagonal matrix and $G_{k+1} + L_{k+1}$ is a lower block triangular matrix with the same diagonal blocks as that of G_{k+1}. Thus the eigenvalues of $G_{k+1} + L_{k+1}$ counting multiplicities are same as the eigenvalues of G_{k+1} and hence $G_{k+1} + L_{k+1}$ is non-singular, since G_{k+1} is SPD. Thus it follows that

$$((G_{k+1} + L_{k+1}) G_{k+1}^{-1} (G_{k+1} + L_{k+1}^T)x, x) = (G_{k+1}^{-1}(G_{k+1} + L_{k+1}^T)x, (G_{k+1} + L_{k+1}^T)x) > 0, \ for\ x \neq 0$$

Thus we have proved that G_k is SPD given that G_{k+1} is SPD and hence B_{NSSOR} is SPD. Also, we have

$$\begin{aligned} B_{NSSOR} &= P^T AP + \sum_{k=1}^{K} L_k G_k^{-1} L_k^T, \\ (B_{NSSOR}x, x) &= (P^T APx, x) + \sum_{k=1}^{K} (G_k^{-1} L_k^T x, L_k^T x), \\ &\geq (P^T APx, x), \\ &> 0, \forall x \neq 0. \end{aligned}$$

Thus $\lambda_i(B_{NSSOR}^{-1} P^T AP) \in (0, 1]$. □

The following theorem shows that NMILUR preconditioners satisfy a particular filtering property.

Theorem 4.2 *For a given matrix A for which NMILUR (NMILUC) as defined in definition 3.2 exists, NMILUR (NMILUC) satisfies the following right (left) filtering property*

$$\begin{aligned} B_{NMILUR}\mathbf{1^T} &= P^T AP\mathbf{1^T} \ (rowsum\ property) \\ \mathbf{1}B_{NMILUC} &= \mathbf{1}P^T AP \ (colsum\ property) \end{aligned}$$

Proof: The preconditioners B_{NMILUR} and B_{NMILUC} can be seen as multilevel preconditioners as in Definition 3.2. For NMILUR we have,

$$G_K = D - \sum_{k=1}^{K} rowsum(L_k G_k^{-1} U_k),$$

while for NMILUC we have,

$$G_K = D - \sum_{k=1}^{K} colsum(L_k G_k^{-1} U_k).$$

Recall that $rowsum(G) = Diag(G\mathbf{1}^T)$ and $colsum(G) = Diag(\mathbf{1}G)$, where $Diag(v)$ is the diagonal matrix formed from vector v. We shall prove the row sum property for B_{NMILUR} and the colsum property can be proved in a similar way for B_{NMILUC}. Writing

$$B_{NMILUR} - P^T AP = \sum_{k=1}^{K} (L_k G_k^{-1} U_k - rowsum(L_k G_k^{-1} U_k))$$

and observing the fact that $rowsum(L_i G_i^{-1} L_i^T)\mathbf{1}^T = (L_i G_i^{-1} L_i^T)\mathbf{1}^T$, the proof follows. □

5 Numerical Results with Matlab

This section presents numerical results for the three nested preconditioners when applied to several real world problems. The numerical results were performed in MATLAB 7.7 in double precision arithmetic on a dual core intel processor with multi-threading enabled. The iterative scheme used is restarted GMRES(60). The algorithm is stopped whenever the relative norm $\|b - Ax_k\|/\|b\|$ is less than 10^{-8}. The exact solution is generated randomly. The maximum Krylov subspace dimension allowed is 1000. For all our experiments in MATLAB, we equilibrate the matrix by scaling the rows and columns by their respective norms. From our experience, we find that this is important for several problems. We refer to this equilibration routine as unsymmetric equilibration (UE), since it can destroy the symmetry of the input matrix. We also discuss results obtained with an equilibration that preserves symmetry. Given a symmetric matrix A, we use a symmetric equilibration (SE) $\widehat{A} = RAR$, where R is a diagonal matrix such that

$$R_{ii} = 1/sqrt(max(abs(a_{i1}, a_{i1}, \ldots, a_{in}))).$$

Here $a_{i,j}$ is the $(i,j)th$ entry of A and n is the size of the square matrix A.

The input matrix is reordered using the nested dissection routine from the Metis graph partitioner [10], which is called inside MATLAB via a mex interface. The local sub-domain solver is built using LU routine of MATLAB, while the GMRES routine is the one available at *http://www-users.cs.umn.edu/ saad/software/*.

The methods are denoted in the tables as following:

- MILU: modified ILU, with colsum constraint
- NMILUR: nested MILU with rowsum constraint
- NMILUC: nested MILU with colsum constraint
- NSSOR: nested SSOR
- ILUND: ILU(0) after the input matrix is reordered using 2-way nested dissection
- ILUNO: ILU(0) with natural ordering of the input matrix

Table 1: Test matrices.

MATRICES	**STANDS FOR**	**Size**	**Non-zeros**	**Symmetric**
2DNHm	2-dimensional non-homogenous problem discretized on $m \times m$ grid	m^2	$\approx 5(m^2)$	Yes
2DADm	2-dimensional advection diffusion problem discretized on $m \times m$ grid	m^2	$\approx 5(m^2)$	No
2DSKYm	2-dimensional sky scrapper problem discretized on $m \times m$ grid	m^2	$\approx 5(m^2)$	Yes
2DCSm	2-dimensional convective skyscrapper discretized on $m \times m$ grid	m^2	$\approx 5(m^2)$	No
3DSKYm	3-dimensional skyscrapper problem discretized on $m \times m \times m$ grid	m^3	$\approx 7(m^3)$	Yes
3DCONSKYm	3-dimensional convective skyscrapper discretized on $m \times m \times m$ grid	m^3	$\approx 7(m^3)$	No
3DANIm	3-dimensional anisotropic problem discretized on $m \times m \times m$ grid	m^3	$\approx 7(m^3)$	No
mat_heat	Heat equation on unstructured mesh	19770	136152	Yes
mat_mem	Equilibrium of a membrane under load	31365	358431	Yes
crystm03	matrices from crystal simulation	24696	583770	Yes
chipcool1	convective thermal flow	20082	281150	No
airfoil_2d	Unstructured 2D mesh (airfoil)	14214	259688	No
Lap1	Laplace 3D unstructured	26082	362328	Yes
Lap2	Laplace 3D unstructured	34960	501394	Yes
bodyy4	Structural problem, Florida matrix market	17546	121550	Yes
bodyy5	Structural problem, Florida matrix market	18589	128853	Yes

5.1 Test matrices

We test the efficiency of our preconditioners on several matrices, ranging from convection-diffusion problems for 2D and 3D case, finite element problems on unstructured meshes, and some miscellaneous problems from Florida matrix market collection [5]. Table 1 displays for each matrix its application domain, its size and number of nonzeros and its numerical symmetry. In the following we describe more in detail the convection-diffusion problems and the finite element problems on unstructured meshed.

Convection diffusion problem We consider the following boundary value problem

$$\begin{array}{r} \eta(x)u + div(\mathbf{a}(x)u) - div(\kappa(x)\nabla u) = f \ \ in \ \ \Omega \\ u = 0 \ \ on \ \ \partial\Omega_D \\ \frac{\partial u}{\partial n} = 0 \ \ on \ \ \partial\Omega_N \end{array} \tag{20}$$

where $\Omega = [0,1]^n$ ($n = 2$, or 3), $\partial\Omega_N = \partial\Omega \setminus \partial\Omega_D$. The function η, the vector field $\mathbf{a}$, and the tensor κ are the given coefficients of the partial differential operator. In 2D case, we have $\partial\Omega_D = [0,1] \times \{0,1\}$, and in 3D case, we have $\partial\Omega_D = [0,1] \times \{0,1\} \times [0,1]$.

The following five cases are considered:

`Case 4.1:` *The advection-diffusion problem with a rotating velocity in two dimensions:*
The tensor κ is the identity, and the velocity is $\mathbf{a} = (2\pi(x_2 - 0.5), 2\pi(x_1 - 0.5))^T$. The function η is zero. The uniform grid with $n \times n$ nodes, $n = 100, 200, 300, 400$ nodes are tested respectively.

`Case 4.2:` *Non-Homogeneous problems with large jumps in the coefficients in two dimensions:*
The coefficient η and $\mathbf{a}$ are both zero. The tensor κ is isotropic and discontinuous. It jumps from the constant value 10^3 in the ring $\frac{1}{2\sqrt{2}} \leq |x - c| \leq \frac{1}{2}$, $c = (\frac{1}{2}, \frac{1}{2})^T$, to 1 outside. We tested uniform grids with $n \times n$ nodes, $n = 100, 200, 300, 400$.

`Case 4.3:` *Skyscraper problems:*
The tensor κ is isotropic and discontinuous. The domain contains many zones of high permeability which are isolated from each other. Let $[x]$ denote the integer value of x. In 2D, we have

$$\kappa(x) = \begin{cases} 10^3 * ([10 * x_2] + 1), & if \ \ [10 * x_i] = 0 \ mod(2) \ , \ i = 1, 2, \\ 1, & otherwise. \end{cases}$$

and in 3D

$$\kappa(x) = \begin{cases} 10^3 * ([10 * x_2] + 1), & if \ \ [10 * x_i] = 0 \ mod(2) \ , \ i = 1, 2, 3, \\ 1, & otherwise. \end{cases}$$

`Case 4.4:` *Convective skyscraper problems:*
The same with the Skyscraper problems except that the velocity field is changed to be $\mathbf{a} = (1000, 1000, 1000)^T$.

`Case 4.5:` *Anisotropic layers:*
The domain is made of 10 anisotropic layers with jumps of up to four orders of magnitude and an anisotropy ratio of up to 10^3 in each layer. For 3D problem, the cube is divided into 10 layers parallel to $z = 0$, of size 0.1, in which the coefficients are constant. The coefficient κ_x in the ith layer is given by $v(i)$, the latter being the ith component of the vector $v = [\alpha, \beta, \alpha, \beta, \alpha, \beta, \gamma, \alpha, \alpha]$, where $\alpha = 1$, $\beta = 10^2$ and $\gamma = 10^4$. We have $\kappa_y = 10\kappa_x$ and $\kappa_z = 1000\kappa_x$. The velocity field is zero.

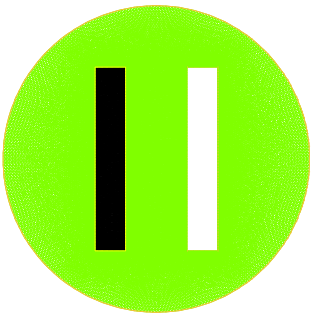

Figure 2: Mesh for heat exchanger problem

Finite element problems on unstructured mesh The Finite element problems are generated by the package FreeFEM++ [13]. We consider the following examples for the unstructured case which are copied

from the FreeFEM++ user guide.
Case A *Equilibrium of a membrane under load:*
The vertical displacement is assumed to satisfy a Poisson equation

$$-\Delta\phi = f \ in \ \Omega. \tag{21}$$

where Ω is elipse with major axis length $a = 2$ and minor axis length $b = 1$. The equation is discretized by a P1- finite element method yielding a linear system that will be referred to as mat_mem in the tables.

Case B *: Heat exchanger problem*
Consider the following equation,

$$\nabla(\kappa\nabla u) = 0 \ in \ \Omega, \ u_\Gamma = g. \tag{22}$$

where the meshed domain Ω is shown on figure 5.1 and κ is discontinuous function with a jump of size 5. The equation is discretized by a P1- finite element method yielding a linear system that will be referred to as mat_heat in the tables.

5.2 Discussion of numerical results

Tables 2, 4, 5, and 6 display test results for NSSOR, NMILUR, NMILUC, and ILUND respectively, with varying number of levels of the nested preconditioners. That is, the nested dissection is stopped when 16, 32, and 64 independend domains are obtained, respectively. The unsymmetric equilibration routine is used for these tests. We present iteration number, error, time spent in construction, time spent in solving phase, and fill-factor (the ratio of the nonzeros in the preconditioner and non-zeros in original matrix A).

We note that the non-zero pattern of the Schur complements for the methods NMILUR, NMILUC, and NSSOR are similar. Hence a similar fill-factor is obtained for all these methods for a given number of domains. We also observe that in general the fill-factor decreases with increasing number of domains.

When comparing NMILUR and NMILUC in Tables 4 and 5, we find that in terms of both iteration count and total time, NMILUR is usually far superior to NMILUC. For symmetric matrices, the NMILUR and NMILUC approximations are same.

We also observe that NSSOR has a faster convergence than NMILUR. However, for matrices Lap1, mat_mem, mat_heat, bodyy4, and bodyy5, NMILUR is faster than NSSOR in terms of both total time and iteration count. For all other cases, NSSOR is superior to both NMILUR and NMILUC. Here we cannot compare the time of ILUND with our nested preconditioners since in ILUND, for ILU(0) Matlab uses a well optimized compiled routines for the factorization and solve with ILU(0) factors. We find a surprising case of Lap1. For this problem, the method fails to converge for 32 domains within maximum iterations allowed. However, it converges again for 64 domains. This is probably, due to the fact that some of the diagonal blocks are very ill-conditioned for 32 domains. This is an aspect that requires further investigation. Moreover, for this problem the fill factor for all the methods for all the partitions remains highest.

Table 3 shows test results for NSSOR with PCG method for symmetric matrices, equilibrated using symmetric equilibration (SE). We observe some improvement for the skyscrapper problem and for the finite element problems, namely, mat_mem, mat_heat. Otherwise, for our test matrices we do not observe any significant reduction in iteration count with PCG compared to GMRES results in Table 2. We have tried SE and PCG with other methods like NMILUR, NMILUC, NFF, ILUND, ILU(0), and MILU and compared with GMRES with UE, we do not observe any advantage in terms of iteration count and thus we have not included those results.

Table 7 displays the results obtained for ILU(0) and MILU for natural ordering of the unknowns. In general ILU(0) is more stable than MILU, which in this case satisfies rowsum property. However, for some cases, e.g. 3dCS20, 2dAD100, 2dANI100, 2dNH100, mat_mem, and airfoil_2d, MILU takes approximately half or less than half iterations compared to ILU(0). Comparing ILU(0) and MILU with natural ordering with NSSOR, we find that in general NSSOR takes more iterations than ILU(0). Exception being mat_mem, where both MILU and NSSOR take less than half the number of iterations of ILU(0). For the problem mat_heat, MILU fails to converge within 1000 iterations, and NSSOR is better than ILU(0). On the other hand, the multilevel extension of MILU with rowsum property, namely NMILUR, performs significantly better than ILU(0) and MILU for mat_mem, mat_heat, body4, body5, airfoil_2d, and Lap2 for matrix partitioned into 16 domains. As mentioned before, in general, we find the performance of NMILUC to be very poor compared to other methods. We could have chosen incomplete LU with drop tolerance but this choice remains very problem dependent and we compare only with parameter free methods. In general, the fill factor of NSSOR decreases very rapidly with increasing number of domains, while the iteration count does not vary much for most of the problems.

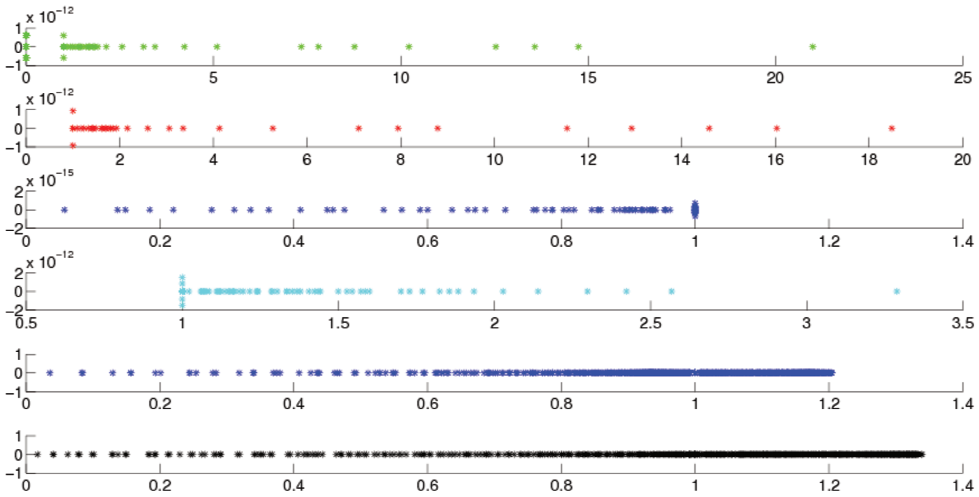

Figure 3: Spectrum plot of from top to bottom : NMILUR, NMILUC, NSSOR, ILUNO, ILUND, MILU for the matrix 2dAD30, nparts=8. Horizontal axis : real part of eigenvalues, vertical axis : imaginary part of eigenvalues

In Figures 3 and 4, we plot the spectrum of the preconditioned matrix, with NMILUR, NMILUC, NSSOR, and NILUND preconditioners. In Figure 3, both NMILUR and NMILUC have similar spectrum, this is because the problem 2dAD30 is close to be symmetric. For NMILUR, there are some eigenvalues close to zero. However, for NMILUC the real part of all the eigenvalues is greater than or equal to one. For NSSOR, the real part of eigenvalues lies between zero and one, where most of the eigenvalues are clustered around one, while some close to zero. The matrix 2dAD30 is SPD and hence the eigenvalues of NSSOR lie between zero and one (see Theorem 4.1 for the proof). For both ILUND and ILUNO, the spectrum is relatively spread out, with most of the eigenvalues clustered around one. On the other hand, in Figure 4, we plot the spectrum for the 3D problem 3dCS15. In contrast, here for NMILUR, the eigenvalues are larger than one. On the other hand for NMILUC, there are negative eigenvalues as well, this is the reason why NMILUC fails to converge within 1000 steps for a similar convective skyscrapper problem 3dCS20, see Table 5. For NSSOR the eigenvalues lies between zero and one as expected since the problem 3dCS15 is symmetric positive definite (see Theorem 4.1).

Figures 5 and 6 display the convergence curves for some of the problems. In the figures, we plot the iteration count of NSSOR, NMILUR, NMILUC, and ILUND versus the norm of relative residual at each iteration. As we see from the plots, for some cases, NMILUC does not converge. For cases where it does not converge within 1000 steps, we have omitted the curve of NMILUC, so that we can see the convergence curves for other methods in detail. For advection-diffusion problem, we see from the plots that ILUND shows large plateaus, that indicate presence of very small eigenvalues. The advection-diffusion problems are very close to be symmetric and thus the convergence behavior of NMILUC and NMILUR are close. Here NSSOR performs better when compared to ILUNO. For a difficult skyscrapper problem namely, 2dSKY100, which has large jumps in the coefficients, we observe that most of the methods have difficulties to converge, see last subfigure of Figure 5, and they converge only after 400 steps. We now consider the matrices taken from Florida matrix market collection. In the case of bodyy4, NMILUR and NMILUC performs better compared to other methods. Here ILUND performs worse. For a problem from crystal simulation, i.e. for crystm03, ILUNO has the best performance closely followed by NSSOR. Here both NMILUR and NMILUC perform worse and take more than 100 steps to converge. For another problem from computational fluid dynamics field, namely airfoil_2d, we find that NMILUR and NMILUC perform better compared to other methods. Here ILUND performs worse.

6 Conclusions

In this paper we have presented a class of recursive multilevel preconditioners. These preconditioners are based on a nested formulation that is enabled by a nested bordered block diagonal form of the input matrix. In addition, this form is well suited for parallel computation.

Table 2: Test Results for NSSOR with unsymmetric equil. in MATLAB, tol. $= 10^{-8}$, t_{con}= construction time in seconds, t_{tot}=total time. Restart parameter for GMRES is 60 and maximum iterations allowed is 1000. Here its = number of iterations required for convergence, err = error in the solution, and mem = nnz(B_NSSOR)/nnz(A)

	16 parts					32 parts					64 parts				
Mat./Resul.	its	err	t_{con}	t_{tot}	mem	its	err	t_{con}	t_{tot}	mem	its	err	t_{con}	t_{tot}	mem
3dCS20	88	e-7	0.03	80.5	3.79	92	e-7	0.01	158.6	2.26	96	e-7	0.01	287.0	1.6
3dANI20	46	e-7	0.03	46.3	3.5	49	e-07	0.01	85.4	2.2	51	e-7	0.01	150.4	1.6
3dSK20	165	e-5	0.04	151.0	4.0	193	e-04	0.02	312.0	2.3	215	e-5	0.01	635.8	1.6
Lap1	46	e-6	1.3	449.2	9.2	85	e-6	0.5	987.2	5.7	115	e-7	111.8	1903.8	3.8
2dAD100	53	e-7	0.03	54.3	3.7	63	e-7	0.02	125.3	2.8	69	e-6	0.01	271.4	2.1
2dANI100	191	e-5	0.03	191.8	3.9	259	e-5	0.02	497.9	2.9	304	e-5	0.01	1105.3	2.1
2dNH100	53	e-7	0.02	16.8	3.7	61	e-7	0.01	39.3	2.8	68	e-6	0.01	95.2	2.1
mat_mem	89	e-5	0.2	367.9	3.6	116	e-6	0.16	851.9	2.9	135	e-5	0.12	1803.9	2.5
mat_heat	60	e-7	0.09	125.6	4.5	79	e-6	0.07	288.7	3.6	93	e-6	0.05	668.1	2.9
bodyy4	32	e-7	0.16	100.3	5.0	38	e-7	0.1	209.2	3.9	44	e-6	0.08	454.4	3.0
bodyy5	86	e-6	0.19	290.6	5.3	77	e-6	0.1	460.7	4.1	87	e-6	0.09	939.2	3.0
crystm03	6	e-9	1.3	55.1	5.8	7	e-8	0.8	95.2	4.3	7	e-8	0.3	97.5	3.0
airfoil2d	43	e-7	0.26	140.6	3.9	52	e-7	0.17	304.4	2.9	58	e-7	0.20	962.0	2.0
Lap2	32	e-7	2.8	654.5	9.3	34	e-7	1.3	854.8	6.3	36	e-7	0.7	1220.1	4.5

Table 3: Test Results for NSSOR with symmetric equil. MATLAB, tol. $= 10^{-8}$, t_{con}= construction time in seconds, t_{tot}=total time. PCG used for symmetric matrices which are scaled by SE. Maximum number of iterations allowed is 1000. Here its = number of iterations required for convergence, err = error in the solution, and mem = nnz(B_NSSOR)/nnz(A)

	16 parts					32 parts					64 parts				
Mat./Resul.	its	err	t_{con}	t_{tot}	mem	its	err	t_{con}	t_{tot}	mem	its	err	t_{con}	t_{tot}	mem
3dANI20	46	e-7	0.03	87.7	3.2	49	e-07	0.02	154.7	1.8	51	e-7	0.04	286.3	1.14
3dSK20	124	e-4	0.04	229.9	3.71	135	e-04	0.02	439.0	1.9	150	e-4	0.01	897.9	1.15
Lap1	49	e-7	2.7	983.5	9.1	91	e-7	1.13	2160.9	5.4	104	e-7	0.48	3294.3	3.5
2dNH100	55	e-7	0.03	45.5	3.6	63	e-7	0.02	104.8	2.7	69	e-7	0.02	251.1	1.94
mat_mem	83	e-7	0.2	417.3	3.5	95	e-7	0.18	845.4	2.9	109	e-7	0.16	1946.1	2.3
mat_heat	61	e-7	0.1	150.9	4.47	70	e-7	0.08	301.0	3.57	81	e-7	0.07	663.5	2.7
bodyy4	34	e-7	0.24	71.8	4.8	40	e-7	0.08	179.7	3.7	44	e-6	0.08	454.4	3.0
bodyy5	95	e-6	0.14	231.4	5.2	81	e-7	0.1	356.6	4.0	91	e-6	0.07	761.5	2.9
crystm03	6	e-9	2.5	127.8	5.7	7	e-8	1.5	217.7	4.1	7	e-8	0.78	303.4	2.7
airfoil2d	43	e-7	0.26	140.6	3.9	52	e-7	0.17	304.4	2.9	58	e-7	0.20	962.0	2.0
Lap2	33	e-8	2.0	432.0	9.0	34	e-8	0.9	566.5	6.0	37	e-8	0.54	863.6	4.15

Table 4: Test Results for NMILUR with unsymmetric equil. in MATLAB, tol. $= 10^{-8}$, t_{con}= construction time in seconds, t_{tot}=total time. Restart parameter for GMRES is 60 and maximum iterations allowed is 1000. Here its = number of iterations required for convergence, err = error in the solution, and mem = nnz(B_NMILUR)/nnz(A)

	16 parts					32 parts					64 parts				
Mat./Resul.	its	err	t_{con}	t_{tot}	mem	its	err	t_{con}	t_{tot}	mem	its	err	t_{con}	t_{tot}	mem
3dCS20	116	e-8	1.2	114.8	4.1	147	e-8	1.7	256.5	2.4	168	e-8	3.0	511.2	1.7
3dANI20	1000	e-8	1.3	1006.4	3.8	1000	e-4	4.2	1703.6	2.4	1000	e-6	3.1	2972.6	1.7
3dSK20	179	e-9	1.17	163.1	4.3	402	e-9	1.6	643.2	2.5	1000	e-8	2.9	2935.5	1.7
Lap1	20	e-7	18.6	212.5	9.6	23	e-7	15.9	281.5	6.1	23	e-7	16.9	366.7	4.0
2dAD100	95	e-8	1.1	97.6	3.8	104	e-8	2.0	207.4	2.9	119	e-8	3.8	466.6	2.2
2dANI100	402	e-8	1.1	403.4	4.0	1000	e-5	2.0	1916.3	3.0	1000	e-7	3.6	3623.8	2.2
2dNH100	102	e-8	0.36	32.1	3.8	109	e-8	0.6	70.0	2.9	122	e-8	1.3	170.6	2.2
mat_mem	45	e-8	5.8	206.5	3.7	44	e-7	8.15	328.7	3.0	53	e-8	13.6	717.7	2.5
mat_heat	50	e-7	2.4	102.0	4.6	50	e-8	3.9	188.0	3.7	56	e-8	7.0	394.9	2.9
bodyy4	10	e-8	3.9	35.9	5.1	12	e-8	5.9	71.5	4.0	11	e-8	10.3	121.4	3.0
bodyy5	10	e-7	4.3	39.6	5.3	11	e-7	6.3	71.4	4.1	18	e-7	10.8	203.3	3.0
crystm03	141	e-8	18.2	1309.0	5.9	371	e-7	17.5	5003.5	4.4	1000	e-3	15.2	13658.7	3.1
chipcool1	24	e-8	9.04	124.7	6.4	24	e-8	8.6	184.8	4.39	26	e-8	14.2	307.4	2.9
airfoil2d	15	e-8	8.4	50.0	4.0	16	e-8	6.6	96.5	3.0	17	e-8	21.2	362.2	2.1
Lap2	33	e-8	46.5	700.3	9.7	34	e-8	34.4	883.6	6.7	35	e-8	37.4	1214.3	4.7

Table 5: Test Results for NMILUC with unsymmetric equil. in MATLAB, tol. $= 10^{-8}$, t_{con}= construction time in seconds, t_{tot}= total time. Restart parameter for GMRES is 60 and maximum iterations allowed is 1000. Here its = number of iterations reqd. for convergence, err = error in the solution, and mem = nnz(B_NMILUC)/nnz(A)

	16 parts					32 parts					64 parts				
Mat./Resul.	its	err	t_{con}	t_{tot}	mem	its	err	t_{con}	t_{tot}	mem	its	err	t_{con}	t_{tot}	mem
3dCS20	1000	e-8	1.3	993.6	4.1	1000	e-8	1.8	1726.1	2.4	1000	e-8	3.1	3002.5	1.7
3dANI20	1000	e-8	1.3	1006.4	3.9	1000	e-4	4.2	1703.6	2.4	1000	e-6	3.2	3001	1.7
3dSK20	1000	e-9	1.2	911.0	4.3	1000	e-9	1.7	1604.5	2.5	1000	e-8	3.1	2935.6	1.7
Lap1	56	e-7	20.7	557.5	9.6	298	e-7	17.3	3454.3	6.1	237	e-7	18.5	3628.4	4.0
2dAD100	104	e-8	1.1	106.8	3.8	113	e-8	2.1	225.5	2.9	127	e-8	4.0	497.8	2.1
2dANI100	1000	e-8	1.19	1005.5	4.0	1000	e-5	2.0	1916.3	3.0	1000	e-7	3.9	3627.9	2.2
2dNH100	102	e-8	0.4	32.3	3.8	109	e-8	0.7	70.3	2.9	122	e-8	1.5	171.0	2.2
mat_mem	1000	e-8	5.7	4080.0	3.7	1000	e-7	8.8	7342.4	3.0	1000	e-8	14.7	13301.3	2.5
mat_heat	655	e-7	2.6	1359.3	4.6	1000	e-8	4.2	3660.2	3.7	1000	e-8	7.5	7090.5	2.9
bodyy4	11	e-8	4.0	38.1	5.1	15	e-8	6.2	88.2	4.0	13	e-8	10.7	141.9	3.0
bodyy5	11	e-7	4.5	43.3	5.4	13	e-7	6.7	83.7	4.2	20	e-7	11.4	225.4	3.1
crystm03	120	e-8	19.6	1102.3	6.0	473	e-7	20.7	6531.5	4.5	1000	e-3	16.9	13703.5	3.1
chipcool1	1000	e-8	9.6	4865.4	6.4	1000	e-8	9.4	7673.6	4.39	1000	e-8	14.4	10594.6	2.9
Lap2	1000	e-8	50.8	19970.6	9.7	1000	e-8	37.7	24943.6	6.7	1000	e-8	40.1	33631	4.7

Table 6: Test Results for ILUND in MATLAB, tol. = 10^{-8}, t_{con}= construction time in seconds, t_{tot}=total time. Restart parameter for GMRES is 60 and maximum iterations allowed is 1000. Here its = number of iterations reqd. for convergence, err = error in the solution, and mem = (nnz(L)+nnz(U))/nnz(A), where L, and U are ILU(0) factors of A

	16 parts					32 parts					64 parts				
Mat./Resul.	its	err	t_{con}	t_{tot}	mem	its	err	t_{con}	t_{tot}	mem	its	err	t_{con}	t_{tot}	mem
3dCS20	103	e-7	0.002	1.08	1	104	e-7	0.002	1.08	1	104	e-7	0.003	1.03	1
3dANI20	71	e-7	0.003	0.79	1	74	e-7	0.003	0.87	1	76	e-7	0.003	0.85	1
3dSK20	162	e-05	0.002	1.5	1	172	e-5	0.002	1.5	1	176	e-4	0.002	1.6	1
Lap1	202	e-5	0.03	6.17	1	201	e-5	0.03	6.1	1	198	e-5	0.03	6.10	1
2dAD100	173	e-6	0.002	1.87	1	174	e-6	0.002	1.90	1	167	e-6	0.002	1.80	1
2dANI100	747	e-5	0.002	7.36	1	781	e-5	0.002	7.76	1	789	e-5	0.002	7.8	1
2dNH100	176	e-6	0.001	1.5	1	177	e-6	0.001	1.64	1	179	e-6	0.001	1.6	1
mat_mem	248	e-5	0.02	7.53	1	254	e-5	0.02	7.53	1	1000	e-1	0.15	25.4	1
mat_heat	233	e-5	0.007	4.44	1	240	e-5	0.007	4.3	1	238	e-5	0.008	4.87	1
bodyy4	69	e-6	0.01	2.76	1	69	e-6	0.01	2.81	1	68	e-6	0.01	2.76	1
bodyy5	143	e-5	0.01	4.82	1	140	e-5	0.01	4.75	1	142	e-6	0.013	5.07	1
crystm03	9	e-8	0.06	0.22	1	9	e-8	0.06	0.22	1	9	e-8	0.06	0.22	1
airfoil2d	69	e-6	0.02	1.66	1	70	e-6	0.02	1.61	1	70	e-6	0.02	1.60	1
Lap2	56	e-7	0.06	6.24	1	55	e-7	0.06	5.67	1	56	e-7	0.06	6.15	1

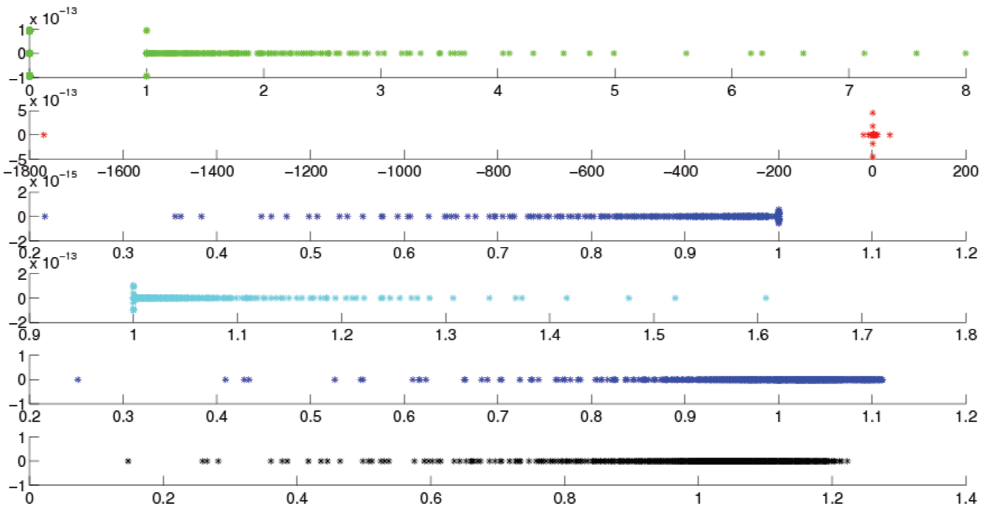

Figure 4: Spectrum plot of from top to bottom : NMILUR, NMILUC, NSSOR, ILUNO, ILUND, and MILU for the matrix 3dCS15, nparts=8. Horizontal axis: real part of eigenvalues, vertical axis: imaginary part of eigenvalues

Table 7: Test results with ILU(0), MILU with natural ordering and with unsymmetric equilibration, The iteration is stopped when the rel. res. is below 10^{-8}, and the maximum number of iterations allowed is 1000, retart parameter for GMRES is 60.

Matrix/Method	ILU(0)		MILU	
	its	err.	its	err.
3dCS20	49	e-6	42	e-8
3dSK20	87	e-5	206	e-8
3dANI20	26	e-7	29	e-8
Lap1	20	e-6	31	e-6
Lap2	36	e-7	38	e-8
2dAD100	80	e-6	44	e-8
2dANI100	231	e-5	63	e-8
2dNH100	77	e-6	41	e-8
mat_mem	217	e-5	84	e-8
mat_heat	138	e-5	769	e-7
bodyy4	23	e-7	30	e-7
bodyy5	39	e-6	35	e-7
crystm03	2	e-9	2	e-8
airfoil_2d	46	e-7	27	e-7

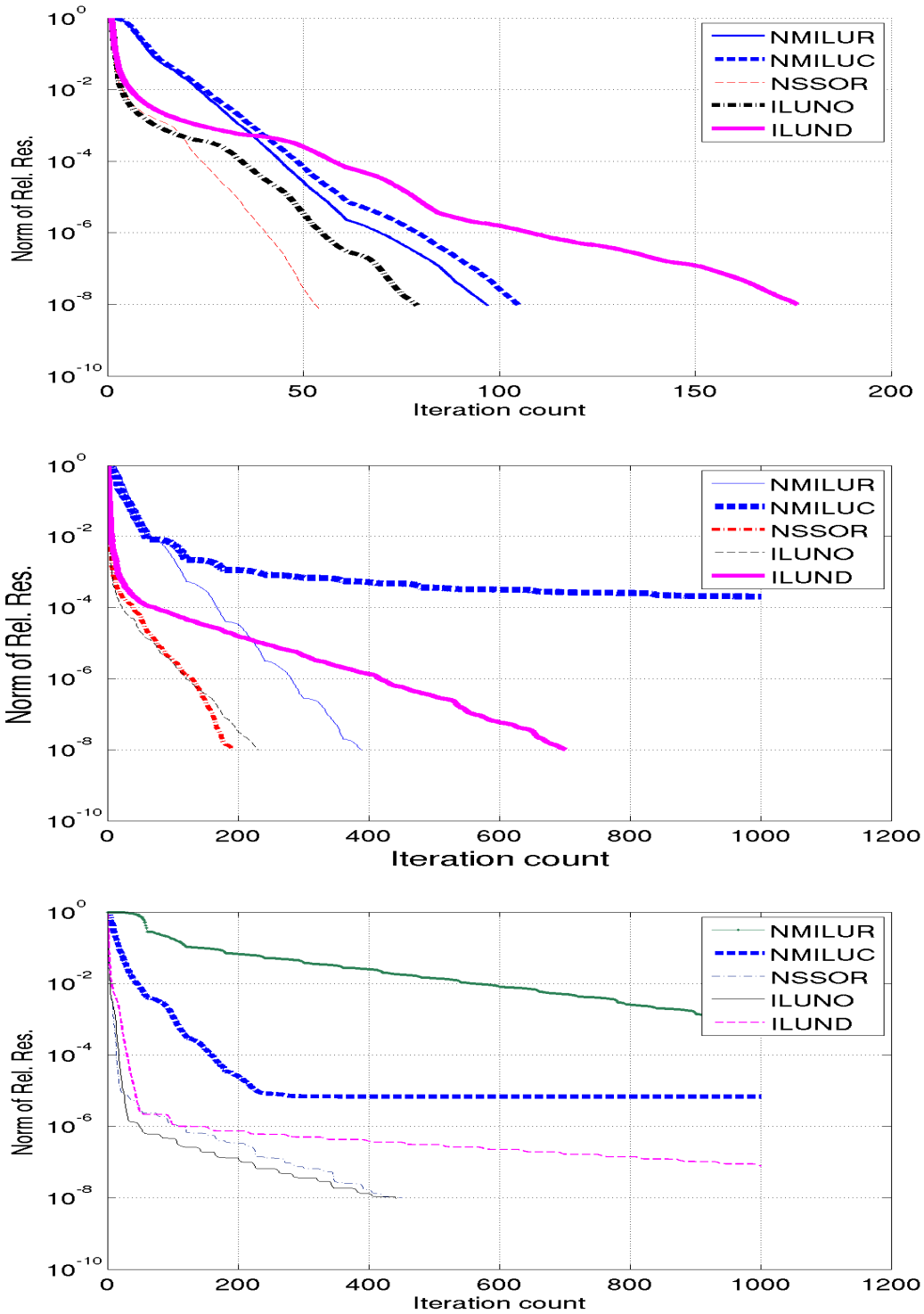

Figure 5: Convergence curves for three test matrices, from top to bottom: 2dAD100, 2dANI100, 2dSKY100.

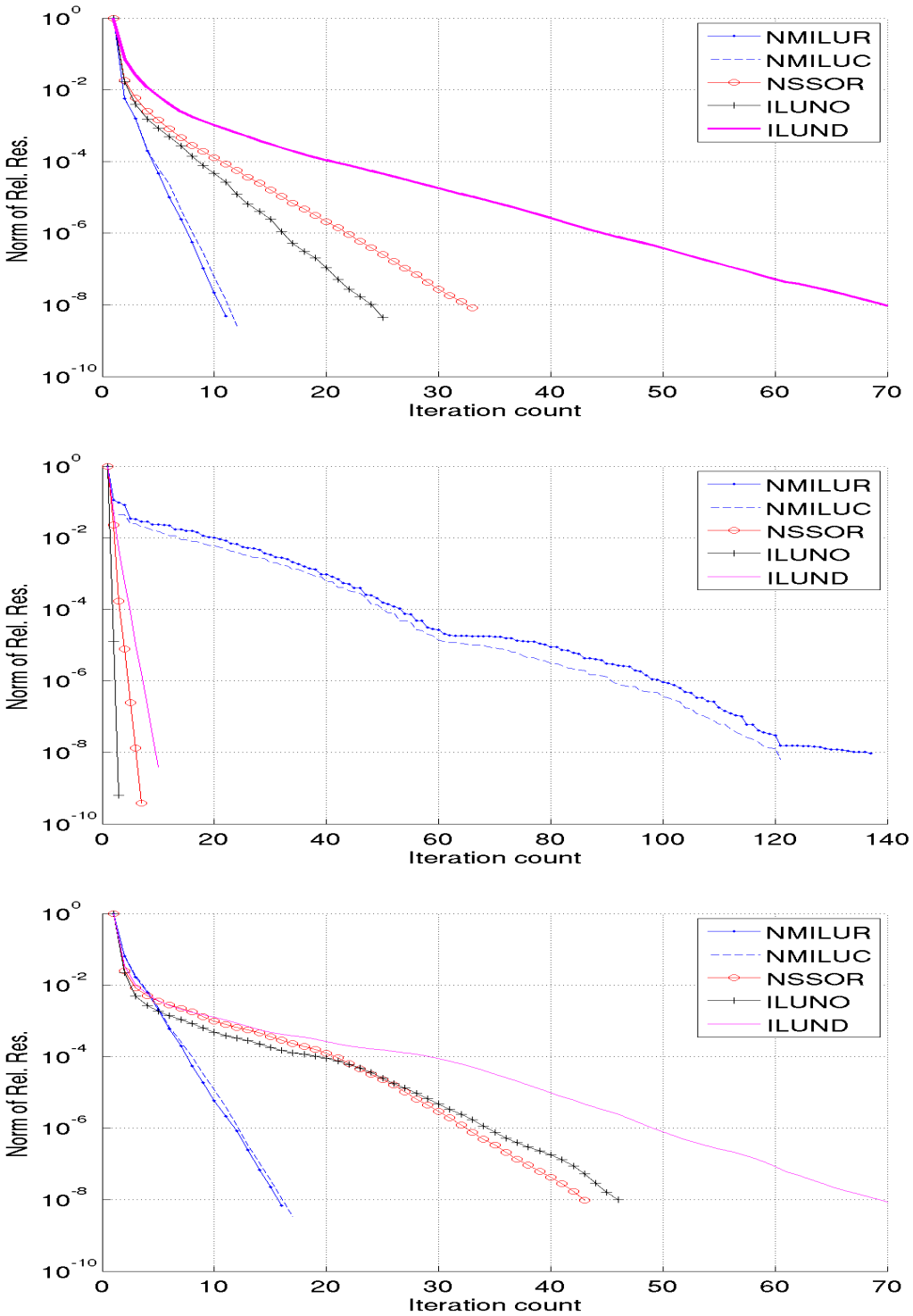

Figure 6: Convergence curves for three test matrices, from top to bottom: bodyy4, crystm03, airfoil2d

Table 8: Test Results for pARMS in C with GMRES, tol. $= 10^{-8}$, t_{con}= construction time in seconds, t_{tot}=total time. Restart parameter for GMRES is 60 and maximum number of iterations allowed is 1000. Here its = number of iterations reqd. for convergence, err = error in the solution, and mem = nnz(B_pARMS)/nnz(A)

	16 parts					32 parts					64 parts				
Mat./Resul.	its	err	t_{con}	t_{tot}	mem	its	err	t_{con}	t_{tot}	mem	its	err	t_{con}	t_{tot}	mem
3dCS20	13	e-4	0.01	0.29	2.9	15	e-4	0.00	0.63	2.48	16	e-5	0.00	1.41	2.31
3dANI20	2	e-1	0.0	0.06	1.94	2	e-1	0.00	0.10	1.81	3	e-2	0.0	0.29	1.7
3dSK20	44	e-3	0.01	0.71	3.72	39	e-3	0.0	1.21	3.08	56	e-3	0.0	3.59	2.71
2dAD100	15	e-7	0.01	0.31	3.9	14	e-8	0.0	0.51	3.85	14	e-7	0.00	1.09	3.19
2dANI100	2	e-1	0.0	0.06	1.94	2	e-1	0.0	0.10	1.81	3	e-2	0.0	0.29	1.72
2dNH100	15	e-8	0.01	0.29	3.9	14	e-8	0.0	0.5	3.85	14	e-7	0.0	1.03	3.19
mat_mem	34	e-7	0.16	2.18	2.36	34	e-6	0.02	2.73	2.35	32	e-7	0.01	4.03	2.34
mat_heat	20	e-8	2.5	44.0	4.5	22	e-7	0.14	0.80	3.61	21	e-7	0.01	1.04	3.51
bodyy4	2	e-7	0.00	0.13	1.29	2	e-7	0.0	0.24	1.36	2	e-7	0.00	0.29	1.39
IFP1	28	e-11	0.09	0.99	3.63	36	e-11	0.01	1.98	3.63	32	e-11	0.00	3.32	3.09
crystm03	62	e-16	0.48	5.60	1.6	62	e-16	0.75	7.8	1.71	62	e-16	0.51	8.57	1.83
airfoil2d	26	e-11	0.06	1.0	1.9	24	e-11	0.01	1.28	1.93	24	e-11	0.00	2.57	1.81
Lap2	1	e-1	0.28	0.59	2.26	1	e-1	0.17	0.56	2.23	1	e-1	0.01	0.53	2.19

As future work, we are interested in developing parallel codes for these preconditioners and compare them with other state-of-art preconditioners [6, 9, 16].

References

[1] J.R Appleyard and I.M. Cheshire. Nested factorization. In *Seventh SPE Symposium on Reservoir Simulation*, pages 315–324, 1983. paper number 12264.

[2] O. Axelsson. *Iterative solution methods*. Cambridge University Press, Cambridge, 1994.

[3] M. Benzi, C. D. Meyer, and M. Tuma. A sparse approximate preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput.*, 17(5):1135–1149, 1996.

[4] M. Benzi and A. M. Tuma. A comparative study of sparse approximate inverse preconditioners. *Appl. Numer. Math.*, 30:305–340, 1999.

[5] T. Davis. University of Florida Sparse Matrix Collection. NA Digest, vol. 92, no. 42, October 16, 1994, NA Digest, vol. 96, no. 28, July 23, 1996, and NA Dige st, vol. 97, no. 23, June 7, 1997. http://www.cise.ufl.edu/research/sparse/matrices.

[6] L. Giraud and A. Haidar. Parallel algebraic hybrid solvers for large 3d convection-diffusion problems. *Numerical Algorithms*, 51(2):151–177, 2009.

[7] L. Grigori, P. Kumar, and F. Nataf. Nested filtering factorization. *In preparation*, 2010.

[8] A. Gupta, G. Karypis, and V. Kumar. Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Transactions on Parallel and Distributed Systems*, 8(5):502–520, 1997.

[9] P. Henon and Y. Saad. A Parallel Multilevel ILU Factorization based on a Hierarchical Graph Decomposition. *SIAM J. on Sci. Comp.*, 28(6):2266–2293, 2006.

[10] G. Karypis and V. Kumar. Metis: A software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices - verstion 4.0, 1998. See http://www-users.cs.umn.edu/karypis/metis.

[11] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359 – 392, 1999.

[12] P. Kumar, L. Grigori, F. Nataf, and Q. Niu. Combinative preconditioning based on relaxed nested factorization and tangential filtering preconditioner. *INRIA report-6955*, 2009. available at http://hal.inria.fr.

[13] O. Pironneau, F. Hecht, A. Le Hyari, and J. Morice. *FreeFEM++*. Universite Pierre et Marie Curie. available at www.freefem.org.

[14] J. W. Ruge and K. Stüben. Algebraic multigrid. In *Multigrid methods*, volume 3 of *Frontiers Appl. Math.*, pages 73–130. SIAM, Philadelphia, PA, 1987.

[15] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS publishing company, Boston, MA, 1996.

[16] Y. Saad and B. Suchomel. Arms: an algebraic recursive multilevel solver for general sparse linear systems. *Numerical Linear Algebra with Applications*, 9(5):359–378, 2002.

[17] B. F. Smith, P. E. Bjørstad, and W. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[18] K. Stüben. A review of algebraic multigrid. *J. Comput. Appl. Math.*, 128(1-2):281–309, 2001. Numerical analysis 2000, Vol. VII, Partial differential equations.

[19] Andrea Toselli and Olof Widlund. *Domain Decomposition Methods - Algorithms and Theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer, 2004.

[20] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press Inc., San Diego, CA, 2001. With contributions by A. Brandt, P. Oswald and K. Stüben.

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 Orsay Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399