NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2016; **23**:61–82
Published online 24 July 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/nla.2006

An algebraic multifrontal preconditioner that exploits the low-rank property

Artem Napov[1,*,†] and Xiaoye S. Li[2]

[1]*Service de Métrologie Nucléaire, Université Libre de Bruxelles (C.P. 165/84), 50, Av. F.D. Roosevelt, B-1050 Brussels, Belgium*
[2]*Computational Research Division, Lawrence Berkeley National Laboratory, 1 Cyclotron Rd., Berkeley, CA 94720, USA*

SUMMARY

We present an algebraic structured preconditioner for the iterative solution of large sparse linear systems. The preconditioner is based on a multifrontal variant of sparse LU factorization used with nested dissection ordering. Multifrontal factorization amounts to a partial factorization of a sequence of logically dense frontal matrices, and the preconditioner is obtained if structured factorization is used instead. This latter exploits the presence of low numerical rank in some off-diagonal blocks of the frontal matrices. An algebraic procedure is presented that allows to identify the hierarchy of the off-diagonal blocks with low numerical rank based on the sparsity of the system matrix. This procedure is motivated by a model problem analysis, yet numerical experiments show that it is successful beyond the model problem scope. Further aspects relevant for the algebraic structured preconditioner are discussed and illustrated with numerical experiments. The preconditioner is also compared with other solvers, including the corresponding direct solver. Copyright © 2015 John Wiley & Sons, Ltd.

Received 23 March 2014; Revised 4 May 2015; Accepted 16 June 2015

KEY WORDS: preconditioning; iterative methods; sparse matrix; structured factorization; incomplete factorization

1. INTRODUCTION

We consider a structured factorization preconditioner for the iterative solution of large sparse linear systems,

$$A\mathbf{u} = \mathbf{b}\,, \tag{1.1}$$

where A is a nonsingular $n \times n$ matrix. The preconditioner is based on a sparse multifrontal LU factorization and exploits numerical rank deficiency of the given off-diagonal blocks in the factor – the so-called low-rank property.

Solution methods that rely on a low-rank property are becoming increasingly popular. They mainly target applications arising from discretization of PDEs, because the low-rank property is then known to hold with numerical rank being independent of, or slowly varying with, the block size [1–3] (see also [4, 5] for the related results). The property is further exploited through a suitable data-sparse representation to reduce the operation count and the memory requirements. Well-known representations include $\mathcal{H}$-matrix and $\mathcal{H}^2$-matrix structures [6–8], sequential and hierarchical semiseparable (HSS) representations [9–12]; a more recent block low-rank format is described in [13].

*Correspondence to: Artem Napov, Service de Métrologie Nucléaire, Université Libre de Bruxelles (C.P. 165/84), 50, Av. F.D. Roosevelt, B-1050 Brussels, Belgium.
†E-mail: anapov@ulb.ac.be

Yet, the existing structured factorizations have two limitations. First, most of them require some knowledge of the underlying problem in addition to the system (1.1) and therefore are not fully algebraic. This may range from the geometric location of degrees of freedom to the implicit assumption on the rank properties. Second, low-rank approximations are often performed with high accuracy, so that the resulting factorization is used as a direct solver. Instead, relaxing this requirement while using the factorization as a preconditioner usually improves both computational time and storage.

In this work, we consider a structured preconditioner that does not suffer from these limitations. It is derived from a multifrontal variant of sparse LU factorization [14, 15] combined with a nested dissection ordering of unknowns [16, 17]. Nested dissection recursively partitions the nodes of the connectivity graph of A into subsets called separators, whereas multifrontal factorization simultaneously eliminates the unknowns corresponding to one separator at a time by partially factorizing a logically dense frontal matrix.

Multifrontal factorization combined with nested dissection ordering represents on its own an effective direct solver. This is because sparse factorizations based on nested dissection are known to have attractive storage requirements and operation count [18], whereas multifrontal factorization is rich in matrix–matrix operations and therefore allows to minimize the time per operation.

Now, the structured preconditioner is obtained from this direct solver if the dense frontal matrices arising during the multifrontal factorization are approximated by an HSS matrix prior to their partial factorization. The HSS approximation amounts to representing some block rows and columns devoid of diagonal block by a rank-deficient, or low-rank, matrix. The hierarchical nature of the approximation comes with the property that some blocks are recursively defined as a combination of others; it is expressed via the corresponding compression tree, also called HSS tree.

The algebraic nature of the preconditioner mainly comes with an algebraic procedure for the construction of the compression tree, that is, with a procedure – based solely on a system matrix A – that determines what rows and columns of the dense frontal matrix should be grouped together to effectively exploit the low-rank property.

In present work, the algebraic procedure requires only the connectivity graph of A. It recursively partitions the nodes of the separator based on the corresponding connectivity subgraph of A that is further completed with some length-two connections. The compression tree is then defined from this partition. The use of the subgraph corresponding to the separator is motivated by the model problem analysis based on model separators. On the other hand, enrichment with additional connections allows to extend the algebraic procedure beyond the analysis scope.

Note that the so far described design choices make the preconditioner almost independent of the initial ordering of unknowns. This is because both the nested dissection ordering and the just-described algebraic procedure only rely on the connectivity information from the system matrix A. This insensitivity to the initial ordering is a key difference from similar structured factorizations available in the literature, as, for example, [19]. Now, considering more specifically the approach in [19], we further note that it uses a different structured partial factorization of HSS matrices. However, despite these differences, the algebraic procedure for the compression tree may be used with this approach as well.

Some other features motivated by the algebraic structure of the preconditioner are as follows.

- The HSS approximation is combined with the structured partial factorization step (corresponding to a so-called ULV factorization) into a single structured partial factorization stage.
- Arbitrary compression trees are allowed. This is a generalization of the traditional HSS format [9, 11] that requires complete binary compression trees. Besides, numerical experiments show that ternary trees yield a slightly faster factorization.
- Symbolic compression is used to ensure that the resulting preconditioner requires less memory than the corresponding direct solver. This is needed because no *a priori* assumption can be made over the presence of the low-rank property.

Eventually, we present some numerical experiments with the resulting structured preconditioner. First, the experiments show that the algebraic procedure for the construction of the compression tree can be effective in reducing both the computational time and the memory needed for the

DOI: 10.1002/nla

factors. Second, the experiments allow to properly discuss the main features and the choice of the main parameters of the preconditioner. Eventually, they allow to assess the performance of the preconditioner and to compare it with that of other solvers, including the corresponding direct solver.

The remainder of the paper is organized as follows. In Section 2 ,we briefly introduce the multifrontal factorization based on nested dissection ordering. Section 3 is devoted to the structured preconditioner, whereas Section 4 introduces and motivates the algebraic construction of the compression tree. Further practical considerations are discussed in Section 5, hand in hand with numerical experiments. They are followed by the concluding remarks in Section 6.

Notation

For a finite set $\mathcal{A}$, $|\mathcal{A}|$ is the number of its elements. For $i \leqslant j$, $[i, j] = \{i, i+1, \ldots, j\}$ stands for the ordered set of integers ranging from i to j. For an $m \times n$ matrix $B = (b_{ij})$ and ordered sets $\mathcal{B} = \{i_1, \ldots, i_b\} \subset [1, m]$, $\mathcal{C} = \{j_1, \ldots, j_c\} \subset [1, n]$, $B|_{\mathcal{B}\times\mathcal{C}}$ is defined as

$$B|_{\mathcal{B}\times\mathcal{C}} = \begin{pmatrix} b_{i_1 j_1} & \cdots & b_{i_1 j_c} \\ \vdots & \ddots & \vdots \\ b_{i_b j_1} & \cdots & b_{i_b j_c} \end{pmatrix}.$$

For any matrix $C = (c_{ij})$, $\mathrm{sp}(C)$ is its sparsity pattern, that is, the set of couples (i, j) corresponding to entries c_{ij} that are considered nonzero. For any matrix D, D^T is its transpose. I stands for an identity matrix, and O stands for a zero matrix (e.i., a matrix whose entries are zeros).

2. MULTIFRONTAL FACTORIZATION

We first describe an exact multifrontal LU factorization that relies on a nested dissection ordering; the structured preconditioner based on this factorization is introduced in the next section. The material of this section is not new and is only outlined (more details can be found in [15, 18]).

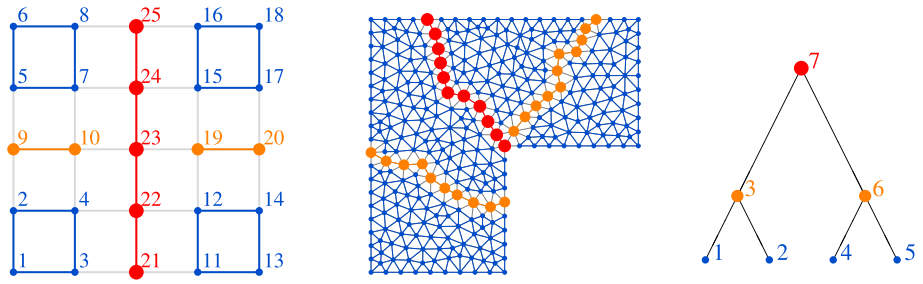
In what follows, we also restrict ourselves to a system matrix A with a symmetric sparsity pattern; that is, $(i, j) \in \mathrm{sp}(A)$ implies $(j, i) \in \mathrm{sp}(A)$. In practice, the matrix A may always be padded with zeros until it reaches the sparsity pattern of $A + A^T$, which is symmetric.

Nested dissection ordering is best described by means of the connectivity graph of A. This latter is an undirected graph having n nodes and such that any two nodes i, j $(i \neq j)$ are connected if and only if $(i, j) \in \mathrm{sp}(A)$. The ordering is obtained by recursively partitioning the nodes of the connectivity graph of A into three disjoint subsets, two of which are not connected to each other but potentially connected to the third, called separator. The nodes of the disconnected subsets are enumerated first, those of the separator are enumerated last, and the procedure is applied recursively to each disconnected subset until it becomes small enough. The no-longer-partitioned subsets are also called separators, so that the union of all separators corresponds to the set of all nodes of the connectivity graph of A. Examples of separators resulting from two recursive partitionings are given on Figure 1(a) and (b); Figure 1(a) also shows the resulting ordering.

With nested dissection ordering, the unknowns corresponding to different disconnected subsets may be eliminated during the factorization independently of each other and before the unknowns corresponding to the separator. This elimination dependency is commonly described by a tree labeled in a postorder; that is, the separator produced after a partitioning step is the parent of, and has a higher label than, the separators produced when partitioning the resulting disconnected subsets; an example of nested dissection tree is presented on Figure 1(c). The labels then define the order in which separators are eliminated. We also assume that the unknowns are ordered accordingly; that is, if k, k' are separator labels and $k > k'$, then unknowns corresponding to the separator k have higher indices than those of separator k'.

Multifrontal factorization for a nested dissection ordering amounts to eliminating the unknowns corresponding to a given separator k by partially factorizing a square logically dense 2×2 *frontal* matrix

DOI: 10.1002/nla



(a) (b) (c)

Figure 1. Two recursive steps of nested dissection partitioning for a graph corresponding (a) to a cartesian 5×5 grid (together with the induced ordering) and (b) to an irregular grid, as well as (c) the corresponding nested dissection tree; big •, midsize •, and small • markers represent first-level, second-level, and third-level separators, respectively.

$$F^{(k)} = \left(F_{ij}^{(k)}\right)_{i,j=1,2} \tag{2.1}$$

with $F_{11}^{(k)}$, $F_{22}^{(k)}$ being square. We first outline the partial factorization of this matrix and then explain how this matrix is computed.

The partial factorization exploits a 2×2 block form of (2.1) in which the indices of the first block correspond to the unknowns $\mathcal{S}_k$ in the separator k, whereas those of the second represent the indices $\mathcal{C}_k$ of rows that need to be updated in the factor after the elimination of this separator. It amounts to the following decomposition:

$$\begin{pmatrix} F_{11}^{(k)} & F_{12}^{(k)} \\ F_{21}^{(k)} & F_{22}^{(k)} \end{pmatrix} = \begin{pmatrix} L_{11}^{(k)} & \\ L_{21}^{(k)} & I \end{pmatrix} \begin{pmatrix} I & \\ & C^{(k)} \end{pmatrix} \begin{pmatrix} U_{11}^{(k)} & U_{12}^{(k)} \\ & I \end{pmatrix}, \tag{2.2}$$

where $L_{11}^{(k)}$, $U_{11}^{(k)}$ are square easy-to-invert matrices and $C^{(k)}$ is the Schur complement of $F^{(k)}$.

In the case of the exact factorization, the matrices $L_{11}^{(k)}$, $U_{11}^{(k)}$ are obtained by computing an LU factorization with partial pivoting[‡] of $F_{11}^{(k)}$; the other submatrices, namely, $L_{21}^{(k)}$, $U_{12}^{(k)}$, and $C^{(k)}$, stem from the equalities $L_{21}^{(k)}U_{11}^{(k)} = F_{21}^{(k)}$, $L_{11}^{(k)}U_{12}^{(k)} = F_{12}^{(k)}$, and $C^{(k)} = F_{22}^{(k)} - L_{21}^{(k)}U_{12}^{(k)}$, which follow directly from (2.2). The use of partial factorization implies that the overall pivoting is restricted to the indices $\mathcal{S}_k$ of the separator k; this is similar to the restricted pivoting strategy (see [20] for details).

The matrices $L_{11}^{(k)}$, $L_{21}^{(k)}$, $U_{11}^{(k)}$, and $U_{12}^{(k)}$ arising from the partial factorization (2.2) represent the given blocks of the sparse factors, and moreover, the sparse factors are progressively formed from these matrices. Regarding the matrix $C^{(k)}$, it is used in the computation of the frontal matrices (2.1) corresponding to the parent of the separator k in the nested dissection tree, as explained in the following paragraph.

Now, the computation of a frontal matrix $F^{(k)}$ is as follows (see [15] for details). If k corresponds to a leaf in the nested dissection tree (e.i., it corresponds to a no-longer-partitioned subset), the frontal matrix is given by

$$F^{(k)} := \begin{pmatrix} A|_{\mathcal{S}_k \times \mathcal{S}_k} & A|_{\mathcal{S}_k \times \mathcal{C}_k} \\ A|_{\mathcal{C}_k \times \mathcal{S}_k} & \end{pmatrix},$$

with the updated indices $\mathcal{C}_k$ being

$$\mathcal{C}_k := \{\, i \mid i > \max(\mathcal{S}_k) \text{ and } (i,j) \in \mathrm{sp}(A) \text{ for some } j \in \mathcal{S}_k \,\}. \tag{2.3}$$

[‡]The pivoting is incorporated into $L_{11}^{(k)}$, which therefore corresponds to a lower triangular matrix whose rows are permuted; $U_{11}^{(k)}$ is upper triangular.

DOI: 10.1002/nla

If k is a not a leaf and has children c_1, c_2, we set

$$\mathcal{C}_k \;:=\; \tilde{\mathcal{C}}_k \;\cup\; \mathcal{C}_{c_1}\backslash\mathcal{S}_k \;\cup\; \mathcal{C}_{c_2}\backslash\mathcal{S}_k \;,$$

where $\tilde{\mathcal{C}}_k$ is defined as in (2.3), and

$$F^{(k)} \;:=\; \begin{pmatrix} A|_{\mathcal{S}_k,\mathcal{S}_k} & A|_{\mathcal{S}_k,\mathcal{C}_k} \\ A|_{\mathcal{C}_k,\mathcal{S}_k} & \end{pmatrix} \;\oplus\; C^{(c_1)} \;\oplus\; C^{(c_2)} \,,$$

where $\oplus$ is an extended add operation. Note that the matrices $C^{(c_1)}$ and $C^{(c_2)}$ are already available during the step k because the elimination is performed in a postorder.

It should be noted for the sake of completeness that the multifrontal factorization, whose description is tailored here for the nested dissection case, can be used as well with other fill reducing orderings. The unknowns are then still usually partitioned into subsets (or supernodes), and the elimination dependency is still described by a tree, although this latter is not necessary a binary tree. Likewise, although the approach developed in this work is presented in the case of nested dissection ordering, the resulting ideas can of course be applied to other partitionings of unknowns related by a proper elimination tree, including partitionings obtained with a fill reducing ordering.

3. STRUCTURED PRECONDITIONER

The structured preconditioner is obtained from the exact factorization by replacing the partial factorization (2.2) with a structured equivalent, which we now describe. We consider structured factorizations based on the HSS representation [12]. Traditionally, such factorizations are applied in two stages: first, the HSS representation of the frontal matrix F is obtained (because one frontal matrix is considered at a time, we omit the separator index k); then a so-called ULV decomposition is computed. Here, we combine both steps into a single structured partial factorization stage; this allows to rewrite the factorization in a more algebraic fashion while enhancing its data locality.

An important building block of structured partial factorization is a low-rank approximation, called compression. It amounts to approximating a given $m \times d$ matrix, say M, by a product

$$M \;\approx\; V\,W$$

of $m \times r$ and $r \times d$ matrices so as to keep the approximation rank r ($r \leqslant \min(m, d)$) reasonably low. This operation usually depends on a threshold parameter ε that controls the accuracy of the approximation.

Compression may be performed using a rank revealing QR decomposition [21–24], with V then having orthogonal columns. In the present work, we use the Householder QR factorization with Businger–Golub column pivoting, as implemented in `dgeqp3` routine from LAPACK [25]. The routine computes one Householder reflector at a time; the factorization process is stopped when, after computing r reflectors, all the partial column norms are lower than ε times the largest initial column norm. The reflectors produced by the routine implicitly define an orthogonal matrix $Q \;=\; (\,\tilde{V}\;\; V\,)$, from which the matrix V itself can be extracted; however, as is made clear in the following text, only the matrix Q is required for the partial factorization step.

Structured partial factorization exploits the low-rank property of a given set $\mathcal{P}$ of (not necessarily consecutive) rows and columns of the frontal matrix F devoid of the corresponding diagonal block $F|_{\mathcal{P}\times\mathcal{P}}$; that is, if s is the number of rows of F and $\mathcal{P}^c = [1, s]\backslash\mathcal{P}$ is the set of indices of F that are not in $\mathcal{P}$, then the matrix that undergoes compression is

$$\left(\, F|_{\mathcal{P}\times\mathcal{P}^c} \;\; F|^T_{\mathcal{P}^c\times\mathcal{P}} \,\right) . \tag{3.1}$$

It is made clear in Section 4.1 that if $\mathcal{P}$ is chosen appropriately, this matrix can be accurately approximated, at least in the model problem context, by a low-rank matrix with a reasonable approximation rank.

One step of structured partial factorization is then as follows. For a given set $\mathcal{P}$, the compression of (3.1) is first performed and amounts to

DOI: 10.1002/nla

$$\left(F|_{\mathcal{P}\times\mathcal{P}^c} \;\; F|_{\mathcal{P}^c\times\mathcal{P}}^T \right) \;\approx\; V \left(\tilde{F}_{12} \;\; \tilde{F}_{21}^T \right), \tag{3.2}$$

where V is a $|\mathcal{P}| \times r$ matrix, $\tilde{F}_{12}$ and $\tilde{F}_{21}^T$ are each a $r \times |\mathcal{P}^c|$ matrix and r is the approximation rank. Because V has orthogonal columns, it can further be extended to form an orthogonal matrix

$$Q \;=\; \left(\tilde{V} \;\; V \right); \tag{3.3}$$

in our case, Q is defined by the reflectors computed during the rank-revealing QR factorization. Eventually, the $|\mathcal{P}| - r$ first rows of the $|\mathcal{P}| \times |\mathcal{P}|$ matrix $Q^T F|_{\mathcal{P}\times\mathcal{P}} Q$ are partially eliminated by determining easy-to-invert D_L and D_R matrices as well as D_{12}, D_{21} and $\tilde{F}_{11}$ such that the following factorization holds

$$Q^T F|_{\mathcal{P}\times\mathcal{P}} Q \;=\; \begin{pmatrix} D_L & \\ D_{21} & I \end{pmatrix} \begin{pmatrix} I & \\ & \tilde{F}_{11} \end{pmatrix} \begin{pmatrix} D_R & D_{12} \\ & I \end{pmatrix};$$

as in the case of exact factorization (2.2), this may be performed using LU factorization with partial pivoting.

Now, putting together the aforementioned operations and ordering the indices in $\mathcal{P}$ before those in $\mathcal{P}^c$, one has

$$\begin{aligned} F \;&=\; \begin{pmatrix} F|_{\mathcal{P}\times\mathcal{P}} & F|_{\mathcal{P}\times\mathcal{P}^c} \\ F|_{\mathcal{P}^c\times\mathcal{P}} & F|_{\mathcal{P}^c\times\mathcal{P}^c} \end{pmatrix} \\ &\approx\; \begin{pmatrix} F|_{\mathcal{P}\times\mathcal{P}} & V\tilde{F}_{12} \\ \tilde{F}_{21}V^T & F|_{\mathcal{P}^c\times\mathcal{P}^c} \end{pmatrix} \\ &=\; \begin{pmatrix} Q & \\ & I \end{pmatrix} \begin{pmatrix} Q^T F|_{\mathcal{P}\times\mathcal{P}} Q & \begin{pmatrix} O \\ \tilde{F}_{12} \end{pmatrix} \\ \begin{pmatrix} O & \tilde{F}_{21} \end{pmatrix} & F|_{\mathcal{P}^c\times\mathcal{P}^c} \end{pmatrix} \begin{pmatrix} Q^T & \\ & I \end{pmatrix} \\ &=\; \left(\begin{array}{cc|c} Q \begin{pmatrix} D_L & \\ D_{21} & I \end{pmatrix} & & \\ & & I \end{array} \right) \left(\begin{array}{cc|c} I & & \\ & & \\ \hline & \tilde{F}_{11} \quad \tilde{F}_{12} & \\ & \tilde{F}_{21} \quad F|_{\mathcal{P}^c\times\mathcal{P}^c} & \end{array} \right) \left(\begin{array}{cc|c} \begin{pmatrix} D_R & D_{12} \\ & I \end{pmatrix} Q^T & & \\ & & \\ \hline & & I \end{array} \right); \end{aligned} \tag{3.4}$$

this is further illustrated with Figure 2. The original frontal matrix is therefore approximated by a product of two easy-to-invert matrices and a block diagonal matrix whose bottom right block still needs to be partially factorized. Hence, we can repeat the earlier steps again while choosing another set $\mathcal{P}$ of block rows.

Now, to obtain an approximate partial factorization of the form (2.2) for a block 2×2 matrix $F = (F_{ij})_{i,j=1,2}$, the indices $\mathcal{P}$ of the compressed rows and columns must be chosen only from the first block F_{11}; that is, if $\mathcal{S}$ is the separator corresponding to F, then F_{11} is a $|\mathcal{S}| \times |\mathcal{S}|$ matrix, and one requires $\mathcal{P} \subset [1, |\mathcal{S}|]$ to hold. If this is the case, then repeating the aforementioned structured partial step for the different index sets $\mathcal{P}$ yields an approximate factorization of the form

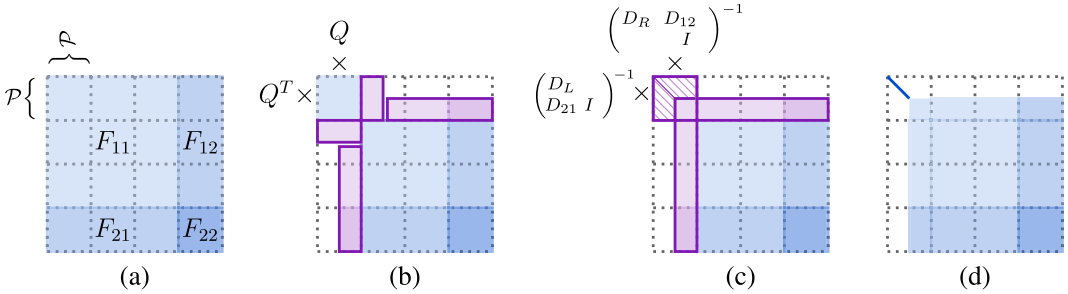


Figure 2. One step of structured partial factorization: frontal matrix (a) before and (b) after the compression step, (c) after the product of rows and columns in $\mathcal{P}$ with Q^T and Q, and (d) after a partial factorization of the diagonal block $Q^T F|_{\mathcal{P}\times\mathcal{P}} Q$. For simplicity, $\mathcal{P}$ is depicted as the set of the first $|\mathcal{P}|$ indices.

DOI: 10.1002/nla

$$F \approx \begin{pmatrix} M_1 & \\ & I \end{pmatrix} \left(\begin{pmatrix} I & \\ & \tilde{F}_{11} \\ \tilde{F}_{21} & \end{pmatrix} \begin{pmatrix} \tilde{F}_{12} \\ F_{22} \end{pmatrix} \right) \begin{pmatrix} M_2 & \\ & I \end{pmatrix}, \tag{3.5}$$

with M_1, M_2 being easy-to-invert, but not necessarily triangular matrices; these matrices are available implicitly as a product of easy-to-invert factors from (3.4) determined for different index sets $\mathcal{P}$. Eventually, if no compression remains to be performed, an exact partial factorization of the bottom right 2×2 block in the central factor of (3.5) is computed, yielding

$$\begin{pmatrix} \tilde{F}_{11} & \tilde{F}_{12} \\ \tilde{F}_{21} & F_{22} \end{pmatrix} = \begin{pmatrix} \tilde{L}_{11} & \\ \tilde{L}_{21} & I \end{pmatrix} \begin{pmatrix} I & \\ & \tilde{C} \end{pmatrix} \begin{pmatrix} \tilde{U}_{11} & \tilde{U}_{12} \\ & I \end{pmatrix}. \tag{3.6}$$

Combining this latter with (3.5) gives an approximate factorization of the same form as (2.2); the matrix $\tilde{C}$ for a given separator k plays then the same role as $C^{(k)}$ in the case of exact factorization from Section 2.

Now, the approximation accuracy ε of the compression step is commonly chosen to be uniformly bounded by a specific user-defined threshold parameter. Although we are not aware of any analysis that supports such a threshold strategy for the considered preconditioner, numerical experiments in Section 5 show that such a strategy works well for the problems at hands. Moreover, the analysis for similar structured factorizations in [26] also suggests that the accuracy control of individual approximations may be a proper tool for controlling the overall quality of the preconditioner.

Besides, the low-rank property of the block (3.1) only depends on the choice of $\mathcal{P}$, and not on the initial ordering of unknowns. Indeed, a reordering of unknowns amounts to some reordering of rows and columns of the block (3.1); the approximation (3.2) is then still possible by applying the same row reordering to the matrix V and the same column reordering to the matrix $\begin{pmatrix} \tilde{F}_{12} & \tilde{F}_{21}^T \end{pmatrix}$. Hence, if the procedure that determines the sets $\mathcal{P}$ is itself insensitive to the initial ordering of unknowns (as, e.g., the algebraic procedure described later in Section 4), then the resulting structured preconditioner becomes insensitive to that ordering.

Eventually, we note that an alternative structured factorization approach is described in [27]. It is similar in that it operates on a dense matrix and progressively reduces the dense part by a repetitive combination of two-sided orthogonal transformation and partial factorization steps. However, [27] considers different orthogonal transformations, and different orthogonal matrices are applied on both sides (as opposite to the application of the same matrix Q here, see (3.4)). The use of different orthogonal transformations aims at reducing the impact of approximations on the Schur complement, but it also requires some additional computation.

4. ALGEBRAIC CONSTRUCTION OF A COMPRESSION TREE

In this section, we present an algebraic procedure for the construction of the compression tree. This is carried out in three steps corresponding to the following three subsections.

We start with the model problem analysis based on model separators. The analysis indicates that the numerical rank of the matrix (3.1) corresponding to a given set $\mathcal{P}$ is related to a quantity that depends on $\mathcal{P}$ and on the connectivity graph of the separator. Because this requires only the knowledge of the system matrix, it is especially relevant for an algebraic solver.

Next, we introduce the compression tree and explain, based on the earlier analysis, why it is sensible to choose the sets $\mathcal{P}$ so that they form a compression tree.

Eventually, we show that the low-rank property can be ensured with a compression tree obtained from a recursive subdivision of the connectivity graph of the separator. Regarding the separators obtained using a graph partitioning tool and which are typically beyond the scope of the analysis, the separator graph is further enriched with length-two connections from the connectivity graph of A before it is used for the construction of the compression tree. This latter construction is used by default in our structured preconditioner.

DOI: 10.1002/nla

4.1. On the low-rank property

We now explain why matrices in (3.1) may exhibit a low-rank property and how the indices $\mathcal{P}$ should be chosen to ensure this property. This is carried out in a special case where the system matrix is symmetric positive definite (SPD) and has the following $n_b \times n_b$ block form:

$$A = \begin{pmatrix} G & -I & \\ -I & G & \ddots \\ & \ddots & \ddots \end{pmatrix}, \tag{4.1}$$

with G being square. In particular, the two-dimensional and three-dimensional (2D and 3D) Poisson model problems discretized using respectively five-point and seven-point finite difference schemes on a cartesian grid and with nodes ordered in a lexicographical ordering fit into this framework; n_b is then the number of nodes in the last coordinate direction. Note that, although the ordering of unknowns that leads to the block form (4.1) is suitable for the analysis in the following text, the presence of the low-rank property does not depend on the initial ordering, as previously stated in Section 3.

Now, we further restrict ourselves to the case of the frontal matrix corresponding to the root separator, that is, the separator with the largest label in the nested dissection tree. In this particular case, the frontal matrix F is also the Schur complement of A corresponding to the separator. Moreover, F also coincides with its top left block F_{11}.

Regarding the separator $\mathcal{S}$ itself, a natural choice corresponds to the set of indices formed by any of the n_b blocks in (4.1) except the first and the last. Indeed, indices belonging to a block j ($1 < j < n_b$) separate the connectivity graph into two disconnected parts, one being the first $j-1$ blocks and the other corresponding to the last $n_b - j$ ones. In the case of Poisson model problems, such choice leads to a geometric separator given by the j-th line (2D) or the j-th plane (3D) of the grid.

In this setting, the frontal matrix is given by

$$F = G - K_{j-1}^{-1} - K_{n_b-j}^{-1}, \tag{4.2}$$

where

$$K_j = G - K_{j-1}^{-1}, \quad K_1 = G. \tag{4.3}$$

The next theorem from [3] relates the rank property of K_∞ to that of G. The assumption of $G-2I$ being SPD simply ensures that A is SPD for any number n_b of blocks.

Theorem 4.1 ([3]*)*
Let A be given by (4.1) with G being square and $G - 2I$ being SPD. Then, for any index subset $\mathcal{P}$ (with complementary subset $\mathcal{P}^c = [1, |\mathcal{S}|]\backslash\mathcal{P}$) and threshold parameter ε, there exists an approximation $\tilde{K}_\infty|_{\mathcal{P}\times\mathcal{P}^c}$ of $K_\infty|_{\mathcal{P}\times\mathcal{P}^c}$ of rank $r(\tilde{K}_\infty, \mathcal{P})$ such that

$$\|\tilde{K}_\infty|_{\mathcal{P}\times\mathcal{P}^c} - K_\infty|_{\mathcal{P}\times\mathcal{P}^c}\| \leqslant \varepsilon \tag{4.4}$$

and

$$r(\tilde{K}_\infty, \mathcal{P}) \leqslant r(G, \mathcal{P}) \left(1 + 8\ln^4\left(\frac{3\|A\|}{\varepsilon}\right)\right), \tag{4.5}$$

where $r(G, \mathcal{P})$ is the rank of $G|_{\mathcal{P}\times\mathcal{P}^c}$.

Whereas in practice neither $j-1$ nor $n_b - j$ reaches infinity, it is reasonable to assume that $K_{j-1} \approx K_{n_b-j} \approx K_\infty$ if both indices are big enough. We implicitly make this assumption in what follows, referring the interested reader to [3] for a more rigorous explanation on why $\|K_j - K_\infty\|$ becomes small for big j. Note however that the low-rank property is also present for small j. For

DOI: 10.1002/nla

instance, although $K_1 = G$ may differ significantly from K_∞, the rank of any block of K_1 is equal to that of the corresponding block of G, and a rank bound similar to (4.5) trivially holds.

The following corollary provides an upper bound on the approximation rank of the submatrix $F|_{\mathcal{P}\times\mathcal{P}^c}$ in the case where $j-1, n_b - j$ both go to infinity; because the matrix F is symmetric, this is also the rank of the submatrix (3.1). This rank essentially depends on the number $\mathrm{cnx}(G,\mathcal{P})$ of nodes in $\mathcal{P}$, which are connected to the rest of the connectivity graph of G (see (4.8) for a formal definition). Note that connectivity graph of G is also the connectivity graph of A restricted to the nodes of the separator, and hence, $\mathrm{cnx}(G,\mathcal{P})$ can be computed algebraically.

Corollary 4.1
Let the assumptions of Theorem 4.1 hold and set $F_\infty = G - 2K_\infty^{-1}$. Then, for any index subset $\mathcal{P}$ and threshold parameter ε, there exists an approximation $\tilde{F}_\infty|_{\mathcal{P}\times\mathcal{P}^c}$ of $F_\infty|_{\mathcal{P}\times\mathcal{P}^c}$ of rank $r(\tilde{F}_\infty,\mathcal{P})$ such that

$$\|\tilde{F}_\infty|_{\mathcal{P}\times\mathcal{P}^c} - F_\infty|_{\mathcal{P}\times\mathcal{P}^c}\| \leqslant 2\varepsilon \tag{4.6}$$

and

$$r(\tilde{F}_\infty,\mathcal{P}) \leqslant \mathrm{cnx}(G,\mathcal{P})\left(2 + 8\ln^4\left(\frac{3\|A\|}{\varepsilon}\right)\right), \tag{4.7}$$

where

$$\mathrm{cnx}(G,\mathcal{P}) = \left|\{\, i\in\mathcal{P} \mid \exists j\notin\mathcal{P} \text{ such that } G_{ij}\in \mathrm{sp}(G)\}\right|. \tag{4.8}$$

Proof
First, note that for the rank $r(G,\mathcal{P})$ of $G|_{\mathcal{P}\times\mathcal{P}^c}$, there holds $r(G,\mathcal{P}) \leqslant \mathrm{cnx}(G,\mathcal{P})$. Then observe that (4.2) and (4.3) imply $F_\infty = G - 2K_\infty^{-1} = 2K_\infty - G$. The result then follows from the previous theorem by setting

$$\tilde{F}_\infty|_{\mathcal{P}\times\mathcal{P}^c} = 2\tilde{K}_\infty|_{\mathcal{P}\times\mathcal{P}^c} - G|_{\mathcal{P}\times\mathcal{P}^c}, \tag{4.9}$$

with $\tilde{K}_\infty$ being the low-rank approximation of K_∞ satisfying (4.4) and (4.5). Indeed, this matrix satisfies (4.6), while its rank is at most the sum of the ranks of each of the terms in (4.9). □

Note that the aforementioned corollary also holds if $\mathrm{cnx}(G,\mathcal{P})$ is replaced by $\mathrm{cnx}(G,\mathcal{P}^c)$, where $\mathcal{P}^c$ the complement of $\mathcal{P}$ with respect to the separator set. On the other hand, both $\mathrm{cnx}(G,\mathcal{P})$ and $\mathrm{cnx}(G,\mathcal{P}^c)$ are bounded above by the number of connection between $\mathcal{P}$ and $\mathcal{P}^c$.

4.2. Compression tree

Let us first illustrate the results of the previous corollary with the examples from Figure 3(a) and (b). We begin by considering the system matrix of the form (4.1) with the connectivity graph being as on Figure 3(a); we assume that each block in (4.1) corresponds to a vertical line of nodes and the separator is chosen to be the central line.

In the following text, we also assume that the rank of (3.1) is bounded similarly to its asymptotical value in (4.7) and hence that $\mathrm{cnx}(G,\mathcal{P})$ is a correct indicator of the rank value. This may sound debatable for the small 7×7 grid in the considered example; however, the small grid is considered here only for simplicity, and the observations mentioned later remain valid in the case of bigger rectangular grid as encountered in practice.

In this example, the subsets $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_4$ of the separator are chosen to be connected sets, and hence, the number of nodes $\mathrm{cnx}(G,\mathcal{P}_i)$ in $\mathcal{P}_i$ connected to the separator nodes outside $\mathcal{P}_i$ is bounded by 2; it is even bounded by 1 in the case of $\mathcal{P}_1$ and $\mathcal{P}_4$. Now, setting $\mathcal{P}_3 = \mathcal{P}_1 \cup \mathcal{P}_2$ in the same example, we note that again $\mathrm{cnx}(G,\mathcal{P}_3) \leqslant 1$, this bound being the same as for $\mathcal{P}_1$ and similar to the one for $\mathcal{P}_2$. As a result, it makes sense to compress the union of some subsets if the union is also a connected set. This compression is especially useful if the rows and columns in $\mathcal{P}_1$ and $\mathcal{P}_2$ have already been compressed, because then the number of rows in the approximated block decreases.

DOI: 10.1002/nla

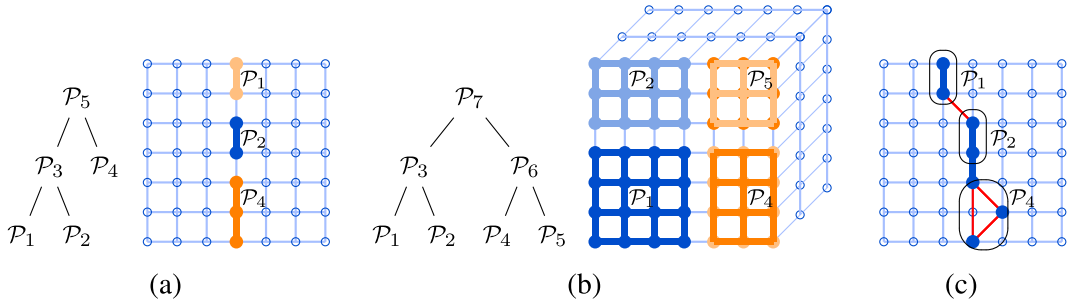


Figure 3. Geometric separator and the corresponding compression tree (a) on a cartesian 7×7 grid, (b) on a cartesian $7 \times 7 \times 3$ grid, and (c) an irregular separator typically obtained with a graph partitioning tool; in this latter case, thick (dark blue) lines represent the separator graph whereas thin (red) lines are the length-two connections added to obtain the enriched graph.

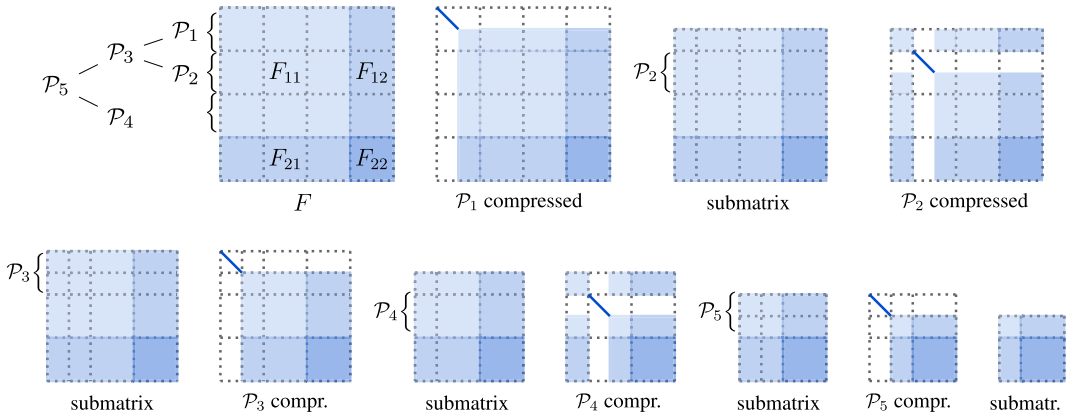


Figure 4. Structured partial factorization algorithm (without the last partial compression step (3.6)) for the compression tree in the top left corner. For simplicity, $\mathcal{P}_i$ are depicted as sets of subsequent indices.

The tree structure naturally accounts for the nested character of subsets $\mathcal{P}_i$ arising in the aforementioned example. The corresponding tree is referred to as compression tree. Each node of the tree corresponds to a given set $\mathcal{P}_i$.

The set $\mathcal{P}_p$ is considered to be a parent of the sets $\mathcal{P}_i, i \in \text{child}(p)$, if $\mathcal{P}_p = \cup_{i \in \text{child}(p)} \mathcal{P}_i$. Moreover, if the sets $\mathcal{P}_i$ in the tree are labeled in a postorder, the parent set is only compressed once its children have been compressed. Now, the set at the root of the compression tree commonly corresponds to the whole separator; the last compression step is then performed on the F_{12} submatrix of the frontal matrix, at least if this block is not empty. An example of a possible compression tree is given on Figure 3(a). The structured partial factorization corresponding to this tree is represented on Figure 4.

4.3. Algebraic construction

We now consider an algebraic construction of the compression tree. To motivate this construction, let us first take a look at the example depicted in Figure 3(b); it represents a 3D cartesian grid with the separator corresponding to its front face. The compression tree $\mathcal{P}_i, i = 1, \ldots, 7$, from the same figure ensures a reasonable low-rank property if the quantities $\text{cnx}(G, \mathcal{P}_i)$ are reasonably small. Considering first $\text{cnx}(G, \mathcal{P}_3)$ and $\text{cnx}(G, \mathcal{P}_6)$, we note that they are indirectly minimized if the sets $\mathcal{P}_3 = \mathcal{P}_1 \cup \mathcal{P}_2$ and $\mathcal{P}_6 = \mathcal{P}_4 \cup \mathcal{P}_5$ – disjoint sets whose union corresponds to the whole separator – are chosen in a way that minimizes the number of connections between them. The latter problem of subdividing a given graph into two subgraphs so as to minimize the number of connections between them is a well-known problem of graph partitioning. Next, considering for example $\text{cnx}(G, \mathcal{P}_1)$ and $\text{cnx}(G, \mathcal{P}_2)$, we note that these quantities are again indirectly minimized if the aforementioned

DOI: 10.1002/nla

minimization problem is solved reasonably well and, in addition, if the number of connections between $\mathcal{P}_1$ and $\mathcal{P}_2$ is made small. This latter problem is also the graph partitioning problem but for the connectivity subgraph corresponding to $\mathcal{P}_3$.

The above example highlights a general approach for the construction of the compression tree. It is based on a graph $\mathcal{G}$ whose nodes correspond to the unknowns in the separator. The compression tree is defined similarly to the nested dissection tree: starting from the set of all nodes of the separator, the tree is obtained by partitioning the graph of the parent set $\mathcal{P}_p$ into $|\text{child}(p)|$ subgraphs corresponding to the children $\mathcal{P}_i$, $i \in \text{child}(p)$. The procedure is stopped when $|\mathcal{P}_p|$ goes below $|\text{child}(p)|$ times a given minimal block size $|\mathcal{P}|_{\min}$. In this work, the graph partitioning problem is solved using Metis graph partitioning routines [28].

In the case of geometric separators $\mathcal{S}$ from Figure 3(a) and (b), the graph $\mathcal{G}$ may be chosen as a connectivity graph of $G = A|_{\mathcal{S}\times\mathcal{S}}$, in agreement with the analysis from Section 4.1. However, in more general cases, this latter graph may be composed of several disjoint subgraphs, as depicted on Figure 3(c). Although some of these subgraphs are geometrically close, while others are not, this proximity cannot be deduced from the connectivity graph of $A|_{\mathcal{S}\times\mathcal{S}}$. To reduce the occurrence of the fragmented graphs, the construction of the compression tree is therefore based on the enriched graph $\mathcal{G}_\beta(\mathcal{S})$, where $\beta > 0$ is an integer parameter. This graph has $|\mathcal{S}|$ nodes and $(i, j) \in \mathcal{G}_\beta(\mathcal{S})$ if there is a path in the connectivity graph of A, which does contain at most $\beta + 1$ nodes and, except for the first and last, these nodes do not belongs to $\mathcal{S}$. $\beta = 2$ is used as a default choice in our structured preconditioner; an example of the enriched graph $\mathcal{G}_2(\mathcal{S})$ corresponding to this choice is given on Figure 3(c). We note that $\mathcal{G}_\beta(\mathcal{S})$ is related to the connectivity graph of $A^\beta|_{\mathcal{S}\times\mathcal{S}}$ but is usually sparser than this latter; for instance, the graph on Figure 3(c) would need an additional connection between nodes 3 and 5 (ordering from bottom to top) to match that of $A^2|_{\mathcal{S}\times\mathcal{S}}$.

Eventually, we summarize the whole procedure for the construction of the compression tree for the case of the separator from Figure 3(c); the resulting compression tree is the same as in Figure 3(a). The input parameter is the set $\mathcal{S}$ of nodes corresponding to the separator; it is represented with filled circles on the figure. First, an enriched graph $\mathcal{G}_2(\mathcal{S})$ is constructed. Two nodes i and j are connected in the graph if there is a node $k \notin \mathcal{S}$ such that $(i, k), (j, k) \in \text{sp}(A)$; the graph connections are depicted on the figure with line segments of different thickness connecting the nodes of the separator. Next, the graph $\mathcal{G}_2(\mathcal{S})$ is used with a partitioning tool such as Metis to subdivide the separator $\mathcal{S}$ into (say) two subsets; possible subsets correspond to $\mathcal{P}_3 = \mathcal{P}_1 \cup \mathcal{P}_2$ and $\mathcal{P}_4$ on the figure. Moreover, the children of the root node of the compression tree are set to $\mathcal{P}_3$ and $\mathcal{P}_4$. Now, if the index sets $\mathcal{P}$ are expected to contain at least $|\mathcal{P}|_{\min} = 2$ elements, we no longer subdivide $\mathcal{P}_4$ (it becomes the leaf of the compression tree) but apply the partitioning tool to the subgraph of $\mathcal{G}_2(\mathcal{S})$ corresponding to $\mathcal{P}_3$; possible resulting subsets are $\mathcal{P}_1$ and $\mathcal{P}_2$, which become the children of $\mathcal{P}_3$ in the compression tree. Because these subsets can no longer be subdivided into subsets of acceptable size (that is, into subset of size at least $|\mathcal{P}|_{\min} = 2$), they become leafs, and the tree construction procedure stops.

5. PRACTICAL CONSIDERATIONS AND NUMERICAL EXPERIMENTS

5.1. General setting

In this section, we discuss several practical aspects that are relevant for the implementation of the structured multifrontal preconditioner. The discussion is accompanied with the numerical experiments, and most of them use the system matrix A corresponding to one of the following two model problems (experiments with other problems are provided starting from Section 5.7). Regarding the vector of right-hand sides, its entries are randomly generated based on normal distribution $\mathcal{N}(0, 1)$.

Problem MOD2D : *2D Poisson problem $-\Delta u = f$ with homogeneous Dirichlet boundary conditions defined on a square domain and discretized using a five-point finite difference scheme on a cartesian $(n_x + 2) \times (n_x + 2)$ grid. The condition number of the resulting system matrix is $\kappa(A) = \sin^2(\frac{\pi n_x}{2n_x+2}) / \sin^2(\frac{\pi}{2n_x+2})$, which is approximately given by $6.5\,10^5$ for $n_x = 4000$.*

DOI: 10.1002/nla

Table I. Time, operation counts for the numerical factorization stage, iteration counts, and memory requirements for the structured preconditioner ($\mathrm{mf}_{\mathrm{str}}(\varepsilon)$) and for the direct solver (mf) applied to 2D and 3D model problems.

	MOD2D ($n_x = 4000$)		MOD3D ($n_x = 100$)	
	$\mathrm{mf}_{\mathrm{str}}(10^{-5})$	mf	$\mathrm{mf}_{\mathrm{str}}(10^{-1})$	mf
ND time (s)	180	180	41	41
Sb.F. time (s)	10	6	2	1
Nm.F. time (s)	193	425	526	1494
Sol. time (s)	27	12	86	4
Tot. time (s)	410	623	655	1540
Nm.F. flops [$\times 10^{12}$]	0.42	2.7	1.2	11
It.	3	1	58	1
nnz factors [$\times 10^9$]	1.2	1.9	0.41	1.7
nnz overhead [$\times 10^9$]	0.12	0.052	0.51	0.29

ND, nested dissection; Sb.F., symbolic factorization; Nm.F., numerical factorization; Sol., solution; Tot., total; It., iteration; nnz., number of nonzeros. The ND stage is the same for both solvers.

Problem MOD3D : *3D Poisson problem* $-\Delta u + 10^{-1}u = f$ *with homogeneous Neumann boundary conditions defined on a cubic domain and discretized using seven-point finite difference scheme on a Cartesian* $n_x \times n_x \times n_x$ *grid. The condition number of the resulting system matrix is approximately*[§] *given by* $1.2\,10^6$ *for* $n_x = 100$*; for other values of* n_x*, it can be roughly estimated by* $120\,n_x^2$.

Numerical experiments are performed in double precision (unless stated otherwise) with a code written in C and executed on a single AMD MagnyCours 2.1-GHz processor (peak performance: $8.4\,10^9$ flop per second) with 32-GB RAM memory. The preconditioner solves the linear system (1.1) by performing several distinct stages.

- Nested dissection ordering is first computed by recursively partitioning the connectivity graph of A (see Section 2). This is carried out using Scotch graph partitioner [29], as this latter provides the nested dissection tree in addition to the ordering itself.
- Symbolic factorization amounts to setting up the symbolic structure of the factors and defining the compression trees. In particular, the compression tree for a given separator is obtained by recursively partitioning the separator graph enriched with some connections of length $\beta = 2$ (see Section 4.3).
- Numerical factorization computes the factors of the structured preconditioner by applying the structured partial factorization to a sequence of dense matrices. The compression is performed using Businger–Golub rank revealing QR decomposition, which is stopped when all the partial column norms are less than ε times the largest initial column norm (see Section 3).
- Solution uses GMRES(30) iteration [30] with the structured preconditioner and zero initial guess to solve the system (1.1); the iteration is stopped if the relative residual norm decreases below 10^{-6}. The reported iteration count corresponds to the number of preconditioner applications.

The code requires only the information available in the system (1.1) and is therefore purely algebraic. In what follows, it is always used with the same values of parameters, unless stated otherwise. An exception is made for the compression threshold parameter ε : this latter is always chosen to be the best value in the set $\{1, 10^{-1}, 10^{-2}, \ldots, 10^{-10}\}$; that is, the value that yields the preconditioner with the smallest total time. In particular, $\varepsilon = 10^{-5}$ is used for MOD2D, whereas $\varepsilon = 10^{-1}$ is specified for MOD3D.

Table I gives the time needed for the four stages of the preconditioner when applied to the MOD2D and MOD3D problems. Here and in the following text, time values correspond to a

[§]Eigenvalue estimates are obtained with Matlab *eigs* function.

DOI: 10.1002/nla

specific run, and up to 5% variation may be observed from one run to another. Now, whereas the time required by the symbolic factorization stage is negligible in 2D and 3D cases, the other time contributions are significant. In the 2D case, the ND, numerical factorization, and solution times are comparable. In the 3D case, the numerical factorization time is dominant; whereas a further reduction in ε may help reduce this contribution, a significant decrease in ε may also lead to a stagnation in convergence. It should be noted that other 2D and 3D test problems considered in Section 5.7 exhibit similar relative times of the four stages as those given in Table I of MOD2D and MOD3D, respectively.

5.2. Comparison with other solvers, part I

The aforementioned structure in four stages may also be used to describe the stages of the direct multifrontal solver based on a nested dissection ordering; the corresponding times are also available in Table I. The table further provides, both for the direct solver and the structured preconditioner, the number of floating point operations (flops) required to compute the factors as well as the memory needed to store them.

The results in Table I show that the improvement in the numerical factorization time from using the preconditioner as compared with that of the direct solver is much smaller than what can be concluded from the numerical factorization flop count. This is due to the fact that multifrontal solver is rich in matrix–matrix operations, and for big enough model problems, the number of flops per second comes close to the peak performance. On the other hand, the structured preconditioner spends a significant portion of time in the compression operations; these are essentially matrix–vector or vector–vector operations whose flop rate is far from the peak. Moreover, the matrix–matrix operations that are carried out by the preconditioner have lower granularity and hence lower flop intensity. This difference between the direct solver and the corresponding preconditioner is further illustrated in Figure 5 for the MOD2D and MOD3D problems of different grid size n_x.

Memory requirements in Table I (reported in number of nonzeros, or nnz) deserve a particular comment. Regarding the memory needed for the factors, it decreases when going from the direct solver to the preconditioner, thanks to the use of low-rank approximations. This decrease is however less significant than the corresponding decrease in the number of flops; moreover, it is more important for MOD3D than for MOD2D problem.

Note than during the factorization stage, more memory may be required than what is needed for the factors. The memory overhead is mainly due to the temporary storage of the Schur complements $C^{(k)}$ and may therefore occur in both the structured preconditioner and the direct solver. Some representative values of the overhead (compared with the memory of the corresponding factors) are reported in Table I. In particular, the memory overhead is larger for the structured factorization preconditioner than that for the direct solver; this is mainly because the storage for the dense representation and for the structured representation of the frontal matrix is simultaneously required by the preconditioner, whereas only the dense storage is needed for the direct solver. Now, this storage overhead is especially an issue for the 3D problems. It may, however, be avoided either partially, by using a compressed representation of the Schur complements[¶] or, totally, by using a left-looking variant of the factorization.

Now, as mentioned in the Section 1, the structured factorization is best if used as a preconditioner. Yet, it can, in principle, also be used as a direct solver if the relative residual norm accuracy required for the solution is not high. Table II provides time and relative residual norm reported for one application of the structured factorization, as a function of the dropping tolerance ε. Note that only the values of ε are considered for which the factorization requires less time than the direct solver. As of the reported relative residual norm, it corresponds to a given (randomly generated) right-hand side, and therefore, its value is rather indicative.

Eventually, we compare in Table III the preconditioned GMRES(30) iteration with the one used without preconditioner. We note that for both model problems, the stand-alone iteration converges slowly because of the fact that the considered problems are ill conditioned.

[¶]Although the compression of the Schur complements may improve the memory usage, its straightforward implementation will likely slow down the factorization stage.

DOI: 10.1002/nla

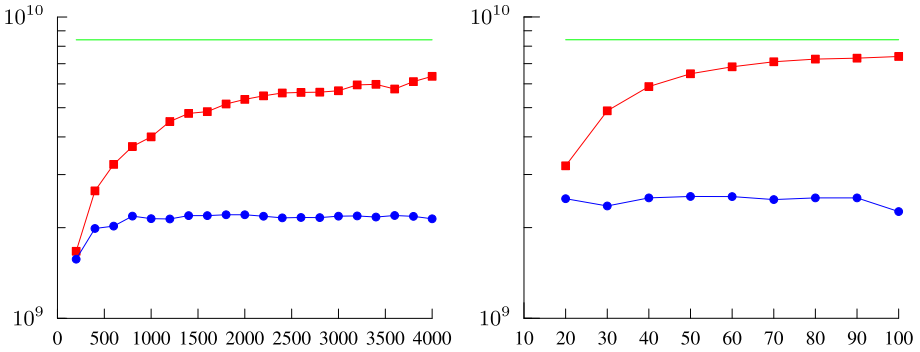


Figure 5. Number of floating point operations (flops) per second during the factorization stage as a function of the grid size n_x in one direction for the MOD2D (left) and MOD3D (right) problems; curves connecting • markers correspond to the structured preconditioner; those connecting ■ markers are for multifrontal solver, and the top solid — curve is the maximal flop rate.

Table II. Numerical factorization and solution times (in seconds), as well as relative residual norm corresponding to one application of the structured factorization, reported here as a function of the dropping tolerance ε for the 2D and 3D model problems.

MOD2D ($n_x = 4000$)				MOD3D ($n_x = 100$)			
Solver	Nm.F. time	Sol. time	rel.res.	Solver	Nm.F. time	Sol. time	rel.res.
$\text{mf}_{\text{str}}(10^{-1})$	112	9.8	$7\,10^{-2}$	$\text{mf}_{\text{str}}(1)$	90	0.9	$1\,10^{-1}$
$\text{mf}_{\text{str}}(10^{-3})$	145	9.9	$3\,10^{-2}$	$\text{mf}_{\text{str}}(10^{-1})$	526	1.5	$4\,10^{-2}$
$\text{mf}_{\text{str}}(10^{-5})$	193	10.3	$3\,10^{-3}$	$\text{mf}_{\text{str}}(10^{-2})$	888	2.2	$2\,10^{-2}$
$\text{mf}_{\text{str}}(10^{-7})$	273	10.5	$3\,10^{-5}$	$\text{mf}_{\text{str}}(10^{-3})$	1390	2.9	$1\,10^{-2}$
$\text{mf}_{\text{str}}(10^{-9})$	329	10.7	$3\,10^{-7}$				
$\text{mf}_{\text{str}}(10^{-11})$	395	10.9	$2\,10^{-9}$				
mf	425	12	$2\,10^{-13}$	mf	1494	3.5	$2\,10^{-14}$

Nm.F., numerical factorization; Sol., solution; rel.res., relative residual. mf corresponds to the direct multifrontal solver.

Table III. Total time and number of GMRES(30) iterations with and without the use of the structured preconditioner.

	MOD2D ($n_x = 4000$)		MOD3D ($n_x = 100$)	
	$\text{mf}_{\text{str}}(10^{-5})$	GMRES(30)	$\text{mf}_{\text{str}}(10^{-1})$	GMRES(30)
Tot. time (s)	410	$-^a$	655	2695
It.	7	$-^a$	58	20049

Tot., total; It., iteration. $-^a$ means that the method has not converged in 4 hours.

5.3. *Symbolic compression*

One of the advantages of the structured preconditioner is its smaller memory requirement for the factors compared with that of the multifrontal factorization. Such a memory decrease is however not guaranteed in principle. Indeed, going back to the structured partial factorization as described in Section 3, an additional memory may be required both during the compression step (3.2), when a $|\mathcal{P}| \times 2|\mathcal{P}^c|$ matrix is approximated (but also replaced) by a product of a $|\mathcal{P}| \times r$ matrix V and a $r \times 2|\mathcal{P}^c|$ matrix , and during the step (3.3) , when the matrix V is further extended to a $|\mathcal{P}| \times |\mathcal{P}|$ matrix. The overall memory then decreases only if

$$2|\mathcal{P}||\mathcal{P}^c| \;>\; 2r|\mathcal{P}^c| + |\mathcal{P}|^2.$$

DOI: 10.1002/nla

To avoid the memory increase, we only perform compression if the aforementioned condition is satisfied. More precisely, we compute a low-rank approximation, and if the aforementioned condition is not met, the low-rank approximation is discarded. Now, this latter case can be viewed as a symbolic compression step during which the given block is replaced by a product of identity matrix with this block; the former matrix is orthogonal and therefore plays the same role as V except that it does not have to be stored explicitly.

5.4. Minimal block size

The compression tree is obtained by recursively subdividing the nodes of the separator so that the size $|\mathcal{P}|$ of the resulting subsets does not go well below $|\mathcal{P}|_{\min}$; that is, more precisely, $\mathcal{P}$ is subdivided into K subsets (here $K = 2$) if $|\mathcal{P}| \geqslant |\mathcal{P}|_{\min} K$.

The choice of $|\mathcal{P}|_{\min}$ has an impact on the performance of the preconditioner. Small values of $|\mathcal{P}|_{\min}$ lead to smaller memory use. Yet, $|\mathcal{P}|_{\min}$ should not be chosen too small if the factorization and solution times are of importance. Indeed, small values of $|\mathcal{P}|$ may trigger the symbolic compression step; this means that the cost of the compression will not be compensated by the reduction in the number of rows and columns as expected during the compression step. Now, even if the symbolic compression does not occur, the gain in the number of operations for an individual compression step essentially behaves like $r/|\mathcal{P}|$, with r being bounded or grow slowly with $|\mathcal{P}|$. This gain should be big enough to overcome the loss in performance implied by the use of compression.

Table IV gives the time and memory requirements of the structured preconditioner for a few possible values of $|\mathcal{P}|_{\min}$; it also provides the storage needed for the root separator, because this latter is among the biggest in the case of model problems. The reported results shows that $|\mathcal{P}|_{\min} = 64$ – the default choice for our structured preconditioner – gives reasonable results for numerical factorization and solution times. Although it is the best choice for the 2D model problem among those in the table, the choice $|\mathcal{P}|_{\min} = 256$ leads to a faster code in 3D, meaning that the optimal value of $|\mathcal{P}|_{\min}$ is problem dependent.

Table IV. Numerical factorization and solution times, as well as memory needed to store the factor and the frontal matrix corresponding to the root separator, given as a function of minimal block size $|\mathcal{P}|_{\min}$.

	MOD2D ($n_x = 4000$)			MOD3D ($n_x = 100$)		
$\|\mathcal{P}\|_{\min}$	16	64	256	16	64	256
Nm.F. time (s)	300	193	221	700	526	493
Sol. time (s)	31	27	31	393	86	99
It.	3	3	3	209	58	54
nnz factors (% mf)	53	62	79	19	23	39
nnz($F_{\text{str}}^{(\text{root})}$) (% mf)	3	6	23	2	3	7

Nm.F., numerical factorization. Sol., solution; It., iteration; nnz., number of nonzeros. The memory requirements are expressed as a percentage of the corresponding requirements of the direct solver.

Table V. Times and iteration count as described in Table IV for the separator size threshold set to $|\mathcal{P}|_{\min}$, $2|\mathcal{P}|_{\min}$, and $8|\mathcal{P}|_{\min}$, with $|\mathcal{P}|_{\min} = 64$.

	MOD2D ($n_x = 4000$)			MOD3D ($n_x = 100$)		
	$\|\mathcal{P}\|_{\min}$	$2\|\mathcal{P}\|_{\min}$	$8\|\mathcal{P}\|_{\min}$	$\|\mathcal{P}\|_{\min}$	$2\|\mathcal{P}\|_{\min}$	$8\|\mathcal{P}\|_{\min}$
Nm.F. time (s)	198	193	227	525	526	567
Sol. time (s)	29	27	31	104	86	98
It.	3	3	3	68	58	55

Nm.F., numerical factorization; Sol., solution; It., iteration.

DOI: 10.1002/nla

Table VI. Time, iteration count, and memory usage parameters as described in Table IV for the compression trees having $K = 2$, 3, and 4 children.

	MOD2D ($n_x = 4000$)			MOD3D ($n_x = 100$)		
K	2	3	4	2	3	4
Nm.F. time (s)	193	184	184	526	448	428
Sol. time (s)	27	29	29	86	86	84
It.	3	3	3	58	59	58
nnz factors (% mf)	62	63	64	23	24	24
nnz($F_{\text{str}}^{(\text{root})}$) (% mf)	6	7	12	3	4	4

Nm.F., numerical factorization; Sol., solution; It., iteration; nnz., number of nonzeros.

Clearly, the structured partial factorization needs only to be applied if the separator size is above $|\mathcal{P}|_{\min}$; however, higher threshold values are also possible. In Table V, we report the results for the preconditioner used with the default $|\mathcal{P}|_{\min} = 64$ value and the separator size threshold set to $|\mathcal{P}|_{\min}$, $2|\mathcal{P}|_{\min}$ and $8|\mathcal{P}|_{\min}$. Although the best overall timing (that is, Nm.F. time + Sol. time) corresponds to the choice $2|\mathcal{P}|_{\min}$ – the default separator size threshold value – the other choices are equally competitive. Now, even larger threshold values are less attractive because the preconditioner properties are then even closer to those of the direct solver.

Note that a similar threshold for structured partial factorization is also used in a model problem setting [11] by employing partial factorization only above the so-called switching level in the nested dissection tree. The use of the switching level is related to the fact that in the case of model problems, bigger separators occur higher in the nested dissection tree.

5.5. *Ternary compression trees*

So far, we have not specified the number K of children of a given parent node in the compression tree; in other words, we have not specified how many subsets result from one step of recursive subdivision of a given set during the construction of the compression tree. The traditional HSS format allows for $K = 2$ children, yielding binary compression trees; this is also our default choice, because the numerical results then also correspond to the HSS format. However, the use of the ternary trees usually allows for a reduction in the computation time.

This is illustrated in Table VI for the case of MOD2D and MOD3D problems. The results show that when using the ternary trees, the improvement in 2D case is not as significant as that in 3D case. The results also indicate a slight increase in memory requirements.

5.6. *Algebraic construction of compression tree*

We now study the effectiveness of the algebraic procedure for the construction of the compression tree described in Section 4.3. For simplicity, we only consider the binary trees, with hence $K = 2$, which is the default value. For this study, we consider two alternative procedures that recursively subdivide a given separator based exclusively on the initial ordering of unknowns; hence, these alternative procedures are ordering-dependent. More precisely, starting from the set of all unknowns of the separator, the tree is obtained by recursively partitioning the parent set into two subsets, one containing the half of the unknowns with lower indices in the ordering and the other formed by the remaining half with higher indices. The tests are performed with a MOD3D problem, and two separator orderings are considered: the ordering induced by the global lexicographical ordering of unknowns in A and the random ordering. Moreover, two variants of nested dissection ordering are considered: one produced by Scotch graph partitioner (default choice) and the other corresponding the geometric plane separators (a 3D equivalent of the separators from Figure 1(a)).

Factorization and solution times as well as iteration counts are given in Table VII. The algebraic construction of the compression tree outperforms the other two options both in time and in memory usage. Comparing the results for separators generated by Scotch and for geometric separators, we

DOI: 10.1002/nla

Table VII. Time, iteration count, and memory usage parameters as described in Table IV for the solution of MOD3D ($n_x = 100$) problem. Compression tree is constructed using the algebraic procedure described in Section 4.3 (alg) as well as separator set bipartirtion based on initial ordering of unknowns (lex) and on the random reordering (rnd); separators from nested dissection are those generated by Scotch [29] (left) and geometric plane separators (right).

	Scotch			Geometric		
	alg	lex	rnd	alg	lex	rnd
Nm.F. time (s)	526	1043	4997	589	1076	5925
Sol. time (s)	86	118	206	242	296	594
It.	58	57	56	148	145	148
nnz factors (% mf)	23	26	44	21	22	42
$\mathrm{nnz}(F_{\mathrm{str}}^{(\mathrm{root})})$ (% mf)	3	12	14	3	12	14

Nm.F., numerical factorization; Sol., solution; It., iteration; nnz., number of nonzeros.

note that they are similar in the case of algebraic construction, except for the iteration counts and, hence, also for the solution times; this means that the low-rank property is correctly exploited even if the separators are not geometric and hence are outside the scope of the analysis from Section 4.1. Eventually, the higher iteration counts in the geometric case are likely due to the compression performed on the smallest separators, that is, on the separators just above the separator size threshold. This can therefore be cured by increasing the value of the threshold.

5.7. Additional test problems

We now study the performance of the structured preconditioner for a broader set of test problems.

Problem CD2D [31]: *A 2D convection–diffusion equation* $-\nu\Delta u + \mathbf{v}\cdot\nabla u = f, \mathbf{v} \in \mathbb{R}^2$, *with Dirichlet boundary conditions defined on a unit square domain and discretized using an upwind five-point finite difference scheme on a cartesian* $(n_x + 2) \times (n_x + 2)$ *grid.* $\mathrm{CD2D}_1$ *corresponds to*

$$\mathbf{v} = \begin{pmatrix} x(1-x)(2y-1) & y(1-y)(2x-1) \end{pmatrix}^T,$$

whereas for $\mathrm{CD2D}_2$, *we set*

$$\mathbf{v} = \begin{pmatrix} \cos(\pi(x-1/3))\sin(\pi(y-1/3)) & \sin(\pi(x-1/3))\cos(\pi(y-1/3)) \end{pmatrix}^T$$

inside the circle of center $(1/3, 1/3)$ *and radius* $1/4$ *and* $\mathbf{v} = 0$ *outside; unless stated otherwise,* $n_x = 4000$ *the viscosity parameter is* $\nu = 10^{-4}$.

Problem L(r, p) [32] : *A 2D Poisson equation* $-\Delta u = f$ *on an L-shaped domain* $[-1, 1]^2 \backslash [0, 1] \times [-1, 0]$ *with homogeneous Dirichlet boundary conditions; the discretization is carried out using Lagrangian finite elements of order* p *on an unstructured mesh with simplex size progressively decreased near the reentering corner, in such a way that the mesh size in its neighborhood is about* 10^r *times smaller. In particular, the mesh of* $L(0, 4)$ *problem have been obtained by uniformly refining five times the mesh from Figure 1(b).*

Problem EDG2D : *A 2D curl–curl equation* $\nabla \times \nabla \times \mathbf{u} + 10^{-2}\,\mathbf{u} = \mathbf{f}$, $\mathbf{f} \in \mathbb{R}^2$, *with homogeneous Dirichlet boundary conditions defined on a unit square domain and discretized using bilinear Whitney elements on a* $(n_x + 2) \times (n_x + 2)$ *grid with* $n_x = 2500$*; in this setting, the degrees of freedom correspond to the grid edges.*

Problem CD3D [31] : *A 3D variant of the* CD2D *problem on a unit cube domain with* $102 \times 102 \times 102$ *mesh and with*

$$\mathbf{v} = \begin{pmatrix} 2x(1-x)(2y-1)z & -y(1-y)(2x-1) & -(2x-1)(2y-1)z(1-z) \end{pmatrix}^T.$$

DOI: 10.1002/nla

Table VIII. Number of unknowns and number of nonzero elements per row of A.

	CD2D	L(0, 4)	L(5, 2)	EDG2D	CD3D	SPH3D	TDR
n [$\times 10^6$]	16.0	4.7	12.6	12.5	1.0	0.5	1.1
$\mathrm{nnz}(A)/n$	5.0	23.5	11.5	7.0	6.9	14.3	39.4

Table IX. Memory requirements, operation and iteration counts, and time for the considered problems, expressed both in absolute values and as a percentage of the corresponding requirements of the direct solver.

		nnz factors		Nm.F. flops		Nm.F. time		Solution		Tot. time	
Problem	ε	$\times 10^8$	% mf	$\times 10^{10}$	% mf	s	% mf	s	It.	s	% mf
$\mathrm{CD2D}_1$	10^{-4}	12	64	44	19	190	50	27	3	407	70
$\mathrm{CD2D}_2$	10^{-4}	12	63	38	16	171	44	56	6	416	71
L(0, 4)	10^{-5}	5.0	74	12	27	55	66	25	3	240	92
L(5, 2)	10^{-5}	13	68	38	20	173	53	10	3	451	77
EDG2D	10^{-8}	9.7	66	22	14	111	42	36	5	322	72
CD3D	1	2.7	15	16	1	93	6	384	444	511	31
SPH3D	10^{-1}	2.3	26	38	10	162	31	137	183	314	58
TDR	10^{-4}	5.1	71	65	81	209	156	34	17	307	156

nnz., number of nonzeros; Nm.F., numerical factorization; Tot., total; It., iteration. The relative residual norm of the solutions obtained with the direct solver (mf) is around 10^{-7} for EDG2D, around 10^{-11} for TDR, around 10^{-13} for L and CD2D, and around 10^{-14} for CD3D and SPH3D.

Problem SPH3D [32] : *A discontinuous 3D Poisson equation* $-\nabla D \nabla u = f$ *with homogeneous Dirichlet boundary conditions defined on a cubic* $[-2.5, 2.5]^3$ *domain with* $D = 10^3$ *inside a sphere of radius 1 and* $D = 1$ *outside; it is discretized on an unstructured quasi-uniform tetrahedral mesh.*

Problem TDR [33] : *A 3D curl–curl equation* $\nabla \times \nabla \times \mathbf{u} - \sigma^2 \mathbf{u} = \mathbf{f}$ *arising in the modeling of accelerator cavities and discretized using Nedelec finite elements on unstructured tetrahedral meshes.*

The number of unknowns and the avarage number of nonzero elements per row of A for the considered problems are given in Table VIII.

The results of the application of the structured preconditioner are reported in Table IX, where they are also compared with those of the direct solver. As before, the right-hand sides are randomly generated based on normal $\mathcal{N}(0, 1)$ distribution, whereas the preconditioner is used with the same default set of parameters, except for the compression threshold parameter ε, which is chosen as the best value in the set $\{1, 10^{-1}, 10^{-2}, \cdots, 10^{-10}\}$.

First, note that the structured preconditioner always requires less memory for the factors than the corresponding direct solver; this is a consequence of symbolic compression. Next, in all the considered examples, it also needs less floating point operations. Favorable flop count usually leads to a faster numerical factorization stage, but due to the lower operation intensity of the compression step, this is not always the case, as illustrated by the results of TDR problem in Table IX.

On the other hand, going back to Table I, one may check that the ratio of numerical factorization and solution times per iteration is more important for 3D than that for 2D problems. This is why in Table IX, lower values of ε are preferred for 3D problems, as this allows to decrease the important factorization cost, even at the expense of increasing the iteration count.

5.8. Mixed precision

In this section, we explore the use of the single precision version of the structured factorization as a preconditioner for the double precision GMRES(30) iteration. In particular, we compare in Table X the single precision structured preconditioner ($\mathrm{smf}_{\mathrm{str}}(\varepsilon)$) with the corresponding single precision

DOI: 10.1002/nla

Table X. Time and iteration counts for both the single precision structured preconditioner ($\text{smf}_{\text{str}}(\varepsilon)$) and the single precision direct solver (smf) used as a preconditioner for the double precision GMRES(30) iteration.

		Nm.F. time (s)		Sol. time (s)		It		Tot. time (s)	
Problem	ε	$\text{smf}_{\text{str}}(\varepsilon)$	smf	$\text{smf}_{\text{str}}(\varepsilon)$	smf	$\text{smf}_{\text{str}}(\varepsilon)$	smf	$\text{smf}_{\text{str}}(\varepsilon)$	smf
MOD2D	10^{-5}	144	246	24	27	3	3	357	458
CD2D_1	10^{-4}	139	224	24	19	3	2	352	429
CD2D_2	10^{-4}	127	226	46	19	6	2	362	431
L(0, 4)	10^{-5}	41	52	9	7	3	2	211	217
L(5, 2)	10^{-5}	129	193	22	18	3	2	402	456
MOD3D	10^{-1}	316	807	58	2	73	1	432	851
CD3D	1	70	851	310	2	444	1	414	886
SPH3D	10^{-1}	100	288	114	3	183	3	229	304
TDR	10^{-4}	163	80	29	6	17	5	255	147

Nm.F., numerical factorization; Sol., solution; It., iteration; Tot., total.

multifrontal solver (smf). Except for the MOD3D and CD3D problems, the latter is also used as a preconditioner, because one solution step does not decreases the relative residual norm below the required 10^{-6} threshold. The results for EDG2D are not reported because the corresponding system matrix is numerically singular in single precision.

Comparing the results from Table X with those in Table IX, it is clear that single precision solvers require less time than the double precision ones. Moreover, the single precision arithmetic does not deteriorate the quality of the structured preconditioner, yielding the same iteration counts as in double precision. In particular, it can be checked that virtually the same rank structure is obtained in single and double precisions; therefore, the results for the number of nonzeros and operation counts needed for the factors in single precision are also the same as those in double precision (which are already reported in Table IX); note, however, that single precision format requires two times less memory and single precision operations are faster than than those in double precision. On the other hand, although the structured preconditioner is still faster than the direct solver in single precision for almost all the considered problems, its relative speed-up is less significant.

5.9. Comparison with other solvers, part II

Here, we report the comparison of the structured factorization preconditioner with the incomplete LU factorization (ILU) preconditioner from the SuperLU package [34]. The SuperLU ILU algorithm is a threshold-based ILUTP proposed by Saad, which combines a dual dropping strategy with numerical pivoting ('T' stands for threshold, and 'P' stands for pivoting). It improved upon the original ILUTP method mainly in two ways: (1) a new dropping strategy that accommodates the use of supernodal structures in the factored matrix, and (2) an 'area-based' fill control heuristic for the secondary dropping strategy [35]. The two parameters that affect performance the most are drop tolerance (τ) and the maximum amount of fill (relative to $\text{nnz}(A)$) allowed in the factored matrices (γ). During factorization, the entries in the factors that are smaller than τ are dropped (separate metrics are used for L and U matrices). Secondly, at each panel factorization step, the amount of fill is dynamically monitored. If the present fill ratio exceeds the parameter γ , more smallest entries are further dropped in the current supernode to maintain the fill ratio below γ.

Table XI shows the performance of the ILU preconditioned GMRES(30) solver for the set of test problems. In order to have a fair comparison with the structured preconditioner, in the ILU experiments, partial pivoting was replaced by diagonal threshold pivoting of threshold 10^{-1}, and the same nested dissection ordering (generated by Scotch) was used[‖]. Moreover, fill ratio parameter γ was fixed to 20 , whereas the value of τ was set to 10^{-4}. This setting gives the best results for most

[‖]Note that both features have positive impact on the factorization time of the incomplete LU factorization preconditioner.

DOI: 10.1002/nla

Table XI. Memory requirements, operation and iteration counts, and time for the ILU preconditioned GMRES(30) solver used with the additional test problems.

		nnz factors	Nm.F. flops	Nm.F. time	Solution		Tot. time
Problem	(τ, γ)	$\times 10^8$	$\times 10^{10}$	s	s	It.	s
$CD2D_1$	$(10^{-4}, 20)$	7.8	16.7	306	259	22	794
$CD2D_2$	$(10^{-4}, 20)$	8.3	15.0	356	605	48	1148
L(0, 4)	$(10^{-4}, 20)$	4.0	8.0	132	131	29	436
L(5, 2)	$(10^{-4}, 20)$	9.3	15.3	377	641	53	1274
EDG2D	$(10^{-8}, 20)$	8.0	145.2	1333	883	99	2391
CD3D	$(10^{-4}, 20)$	1.2	29.8	351	130	137	514
SPH3D	$(10^{-8}, 100)$	7.2	362.0	3538	321	78	3898
TDR	$(10^{-5}, 40)$	9.2	212.2	2386	64	12	2489

nnz., number of nonzeros; Nm.F., numerical factorization; Tot., total; It., iteration. The relative residual norm of the solutions is less than 10^{-6}.

problems. However, for EDG2D, SPH3D, and TDR, it did not lead to satisfactory convergence. Then, we found the other setting, which led to satisfactory convergence, with the respective parameters displayed in the table.

Comparing with Table IX, we notice that although the ILU preconditioner has slight advantage in terms of the nonzeros of the factors and the flop count for some problems, its factorization time and the solution time are often worse than the corresponding times of the structured factorization preconditioner.

For problem SPH3D, we need to allow a much larger fill factor ($\gamma = 100$) in order for ILU to converge, which translates into much worse memory requirement and runtime. For problem TDR, diagonal threshold pivoting results in either zero pivots or small pivots on the diagonal, leading to an unstable preconditioner. On the other hand, partial pivoting causes too many off-diagonal pivots and many more fill-ins, exhausting memory. What worked is to use MC64 [36] to pre-pivot large entries of A to the diagonal before proceeding to the usual factorization.

5.10. On simultaneous compression

Here, we give some motivation behind the simultaneous approximation of the blocks $F|_{\mathcal{P}\times\mathcal{P}^c}$ and $F|^T_{\mathcal{P}^c\times\mathcal{P}}$ in the compression step (3.2). To make the arguments clearer, let us first introduce a possible alternative approach based on separate compression of these blocks. In the alternative approach, the step (3.2) of the structured factorization scheme is replaced by two compression steps

$$F|_{\mathcal{P}\times\mathcal{P}^c} \approx V_l \tilde{F}_{12}, \qquad F|^T_{\mathcal{P}^c\times\mathcal{P}} \approx V_r \tilde{F}_{21}^T, \tag{5.1}$$

with the rank r being the maximum of the two compression ranks. Both V_l and V_r have orthogonal columns and can be completed to form the orthogonal matrices $Q_l = (\tilde{V}_l \; V_l)$ and $Q_r = (\tilde{V}_r \; V_r)$. The remaining of the factorization follows the same lines, except that the the left multiplication always involves Q_l (instead of Q), whereas the right multiplication involves Q_r (instead of Q).

Now, the (exact) rank of the matrix $\begin{pmatrix} F|_{\mathcal{P}\times\mathcal{P}^c} & F|^T_{\mathcal{P}^c\times\mathcal{P}} \end{pmatrix}$ is between the maximum and the sum of the (exact) ranks of $F|_{\mathcal{P}\times\mathcal{P}^c}$ and $F|^T_{\mathcal{P}^c\times\mathcal{P}}$. Hence, although the numerical rank may behave in a slightly different way, neither of the approaches is likely significantly cheaper than the other. In particular, the simultaneous approach is more attractive for (almost) symmetric matrices, because the simultaneous compression rank is then (almost) the same as those from individual compression steps, whereas the matrices V and Q are computed and stored only once. However, for strongly non-symmetric matrices, the alternative approach may be attractive.

Another reason for using simultaneous compression is that the resulting preconditioner does not breakdown for strongly non-symmetric convection–diffusion problems. This is illustrated in Table XII for two such problems with progressively decreasing viscosity parameter ν.

DOI: 10.1002/nla

Table XII. Iteration count for the structured factorization preconditioner used with either the simultaneous (sml, the default approach) or separate (sep) compression for $CD2D_1$ and $CD2D_2$ problems with various values of viscosity parameter ν.

	$CD2D_1$ ($n_x = 2000$)		$CD2D_2$ ($n_x = 2000$)	
	sml	sep	sml	sep
$\nu = 10^{-2}$	4	5	4	5
$\nu = 10^{-3}$	3	4	4	4
$\nu = 10^{-4}$	3	$> 10^3$	4	$> 10^3$

6. CONCLUSIONS

We have presented and studied an algebraic structured preconditioner, which is based on a multifrontal factorization and nested dissection ordering. Nested dissection subdivides the unknowns into separators based on the connectivity graph of A, whereas multifrontal factorization performs a partial factorization of the corresponding frontal matrices. The structured preconditioner is obtained if structured partial factorization that exploits the low-rank property of given blocks of rows and columns devoid of diagonal block is used instead. The blocks to be compressed and their compression order are represented with a compression tree.

For the structured preconditioner to be algebraic, the compression tree has to be determined from the information available in the system matrix. In present work, this has been achieved by a recursive partitioning of the enriched graph corresponding to the separator and obtained from the connectivity graph of A. This approach is motivated by the analysis of model problems with model root separators, yet numerical experiments have confirmed that it works beyond the model problem context.

We have also discussed a number of practical aspects that are important for the efficiency and memory usage of the preconditioner. The effectiveness of the preconditioner has further been illustrated with numerical experiments for problems arising in various PDE applications. These experiments show, among others, that the considered preconditioner is competitive when compared with some reference direct and iterative solvers.

ACKNOWLEDGEMENTS

We thank Ming Gu for the numerous discussions and Yvan Notay for his generator of convection–diffusion test problems. The work of the second and (partially) the first author was supported by the Director, Office of Science, Office of Advanced Scientific Computing Research of the US Department of Energy under contract no. DE-AC02-05CH11231. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under contract no. DE-AC02-05CH11231.

REFERENCES

1. Bebendorf M. Why finite element discretizations can be factored by triangular hierarchical matrices. *SIAM Journal on Numerical Analysis* 2007; **45**:1472–1494.
2. Grasedyck L, Kriemann R, Le Borne S. Domain decomposition based $\mathcal{H}$-LU preconditioning. *Numerische Mathematik* 2009; **112**:565–600.
3. Chandrasekaran S, Dewilde P, Gu M, Somasunderam N. On the numerical rank of the off-diagonal blocks of Schur complements of discretized elliptic PDEs. *SIAM Journal on Matrix Analysis and Applications* 2010; **31**:2261–2290.
4. Bebendorf M, Hackbusch W. Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. *Numerische Mathematik* 2003; **95**:1–28.
5. Börm S. Approximation of solution operators of elliptic partial differential equations by $\mathcal{H}$- and $\mathcal{H}^2$-matrices. *Numerische Mathematik* 2010; **115**:165–193.
6. Börm S, Grasedyck L, Hackbusch W. Introduction to hierarchical matrices with applications. *Engineering Analysis with Boundary Elements* 2003; **27**:405–422.

DOI: 10.1002/nla

7. Bebendorf M. *Hierarchical Matrices*, Lecture Notes in Computational Science and Engineering (LNCSE), vol. 63. Springer-Verlag: Berlin, 2008.
8. Hackbusch W, Khoromskij B, Sauter S. On $\mathcal{H}^2$-matrices. In *Lectures on Applied Mathematics*. Springer-Verlag: Berlin, 2000; 9–29.
9. Chandrasekaran S, Gu M, Lyons W. A fast adaptive solver for hierarchically semiseparable representations. *Calcolo* 2005; **42**:171–185.
10. Chandrasekaran S, Gu M, Plas T. A fast ULV decomposition solver for hierarchically semiseparable representations. *SIAM Journal on Matrix Analysis and Applications* 2006; **28**:603–622.
11. Xia J, Chandrasekaran S, Gu M, Li XS. Superfast multifrontal method for large structured linear systems of equations. *SIAM Journal on Matrix Analysis and Applications* 2009; **31**:1382–1411.
12. Xia J, Chandrasekaran S, Gu M, Li XS. Fast algorithms for hierarchically semiseparable matrices. *Numerical Linear Algebra with Applications* 2009; **17**:953–976.
13. Amestoy P, Ashcraft C, Boiteau O, Buttari A, L'Excellent JY, Weisbecker C. Improving multifrontal methods by means of block low-rank representations INPT-IRIT. *Technical Report RT/APO/12/6*, 2012.
14. Duff IS, Reid JK. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Transactions on Mathematical Software* 1982; **9**:302–325.
15. Liu JWH. The multifrontal method for sparse matrix solution: theory and practice. *SIAM Review* 1992; **34**:82–109.
16. George A. Nested dissection of a regular finite-element mesh. *SIAM Journal on Numerical Analysis* 1973; **10**: 345–363.
17. Lipton RJ, Rose DJ, Tarjan RE. Generalized nested dissection. *SIAM Journal on Numerical Analysis* 1979; **16**: 346–358.
18. George A, Liu JWH. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall: Englewood Cliffs, 1981.
19. Xia J. Efficient structured multifrontal factorization for general large sparse matrices. *SIAM Journal of Scientific Computing* 2013; **35**:A832–A860.
20. Hogg JD, Scott JA. Pivoing strategies for tough sparse indefinite systems. *ACM Transactions on Mathematical Software* 2013; **40**(1):4:1–4:19.
21. Businger P, Golub GH. Linear least squares solutions by Householder transformations. *Numerische Mathematik* 1965; **7**:269–276.
22. Chan TF. Rank revealing QR factorizations. *Linear Algebra and its Applications* 1987; **88/89**:67–82.
23. Chandrasekaran S, Ipsen ICF. On rank-revealing factorizations. *SIAM Journal on Matrix Analysis and Applications* 1994; **15**:592–622.
24. Gu M, Eisenstat SC. Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM Journal of Scientific Computing* 1996; **17**:848–869.
25. Anderson E, Bai Z, Bischof C, Blackford S, Demmel J, Dongarra J, Du Croz J, Greenbaum A, Hammarling S, McKenney A, Sorensen D. *LAPACK Users' Guide* (3rd edn). SIAM: Philadelphia, PA, 1999.
26. Napov A. Conditioning analysis of incomplete Cholesky factorizations with orthogonal dropping. *SIAM Journal on Matrix Analysis and Applications* 2013; **34**:1148–1173.
27. Li S, Gu M, Cheng L. Fast structured LU factorization for nonsymmetric matrices. *Numerische Mathematik* 2014; **127**:35–55.
28. Karypis G, Kumar V. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal of Scientific Computing* 1999; **20**:359–392.
29. Pellegrini F. SCOTCH: Static mapping, graph partitioning, and sparse matrix block ordering package. (Available online at: http://www.labri.fr/pelegrin/scotch/) [accessed on 20 May 2011].
30. Saad Y, Schultz MH. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing* 1986; **7**:856–869.
31. Notay Y. An aggregation-based algebraic multigrid method. *Electronic Transactions on Numerical Analysis* 2010; **37**:123–146.
32. Napov A, Notay Y. An algebraic multigrid method with guaranteed convergence rate. *SIAM Journal of Scientific Computing* 2012; **43**:A1079–A1109.
33. Lee L, Li Z, Ng C, Ko K. Omega3P : a parallel finite-element eigenmode analysis code for accelerator cavities. *Technical Report SLAC–PUB–13529*, SLAC National Accelerator Laboratory: Menlo Park, CA, 2009.
34. Li XS. An overview of SuperLU: algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software* 2005; **31**(3):302–325.
35. Li XS, Shao M. A supernodal approach to incomplete LU factorization with partial pivoting. *ACM Transactions on Mathematical Software* 2011; **37**(4).
36. Duff IS, Koster J. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM Journal on Matrix Analysis and Applications* 1999; **20**(4):889–901.

DOI: 10.1002/nla