IMF: An incomplete multifrontal LU-factorization for element-structured sparse linear systems

Nick Vannieuwenhoven *Karl Meerbergen*

Report TW 581, December 2010



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

IMF: An incomplete multifrontal LU-factorization for element-structured sparse linear systems

Nick Vannieuwenhoven *Karl Meerbergen*

Report TW 581, December 2010

Department of Computer Science, K.U.Leuven

Abstract

We propose an incomplete multifrontal *LU*-factorization (IMF) that extends supernodal multifrontal methods to incomplete factorizations. IMF can be used as a preconditioner in a Krylov-subspace method to solve large-scale sparse linear systems with an underlying element structure. Such systems arise e.g. from a finite element discretization of a partial differential equation. The fact that the element matrices are dense is exploited to increase the computational performance and the robustness of the factorization (through partial pivoting within the dense matrices). We compare IMF with the multilevel ARMS, the level of fill-in ILU, the threshold-based ILUT and the SPAI preconditioner. IMF is demonstrated to be effective on linear systems derived from two incompressible flow simulation model problems, outperforming the aforementioned preconditioners by one order-of-magnitude in one instance. The preconditioner was also applied to solve general sparse systems, without an underlying element structure. It is shown to be effective and robust on some matrices from the University of Florida sparse matrix collection and the Matrix Market, provided that an artificial element structure can be extracted that is similar to a finite element discretization.

Keywords : incomplete factorization, supernodal multifrontal method, multilevel preconditioner, block *LU*-factorization, finite element
MSC : Primary : 65F08, 65N30 Secondary : 65Y05, 65F50, 65N22.

IMF: AN INCOMPLETE MULTIFRONTAL LU-FACTORIZATION FOR ELEMENT-STRUCTURED SPARSE LINEAR SYSTEMS*

NICK VANNIEUWENHOVEN‡ AND KARL MEERBERGEN‡

Abstract. We propose an incomplete multifrontal LU-factorization (IMF) that extends supernodal multifrontal methods to incomplete factorizations. IMF can be used as a preconditioner in a Krylov-subspace method to solve large-scale sparse linear systems with an underlying element structure. Such systems arise e.g. from a finite element discretization of a partial differential equation. The fact that the element matrices are dense is exploited to increase the computational performance and the robustness of the factorization (through partial pivoting within the dense matrices). We compare IMF with the multilevel ARMS, the level of fill-in ILU, the threshold-based ILUT and the SPAI preconditioner. IMF is demonstrated to be effective on linear systems derived from two incompressible flow simulation model problems, outperforming the aforementioned preconditioners by one order-of-magnitude in one instance. The preconditioner was also applied to solve general sparse systems, without an underlying element structure. It is shown to be effective and robust on some matrices from the University of Florida sparse matrix collection and the Matrix Market, provided that an artificial element structure can be extracted that is similar to a finite element discretization.

Key words. incomplete factorization, supernodal multifrontal method, multilevel preconditioner, block LU-factorization, finite element

AMS subject classifications. 65F08, 65N30, 65Y05, 65F50, 65N22

1. Introduction.

In this paper, we propose a preconditioner to solve a large-scale element-structured sparse linear system

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

using Krylov-subspace methods. We say that A admits an *element structure* $\mathcal{E}$ if

$$A := \sum_{e \in \mathcal{E}} P_e A_e P_e^T := \sum_{e \in \mathcal{E}} A^{[e]},$$

where $\mathcal{E}$ is a set of *elements*, A_e is a square dense coefficient matrix, P_e a local-to-global variable mapping and $A^{[e]} := P_e A_e P_e^T$ is a sparse *element matrix*. P_e is a zero matrix with a one in position (i, j) iff i is the jth item in the *index set* $\mathcal{I}(e)$ of $e \in \mathcal{E}$. The sparsity $\mathrm{Sp}(e)$ of an element $e \in \mathcal{E}$, and of its element matrix $A^{[e]}$, is thus $\mathcal{I}(e) \times \mathcal{I}(e)$. The index sets of a set of elements may have a non-zero intersection. Such structured systems arise frequently in practice, e.g. from the discretization of a partial differential equation (PDE) using the finite element method (FEM). In that context, the index set of an element corresponds to its degrees of freedom (dof).

It has long been recognized that effective preconditioners are required to solve sparse linear systems efficiently using Krylov-subspace methods [10]. Some preconditioners exploit knowledge of the element structure. These so-called element-by-element (EBE) preconditioners operate exclusively on the element matrices. The sparse matrix A is never *assembled* (computed). Typically, they compute an approximation to A^{-1} by computing the LU-factorizations of agglomerates of elements, and then combining them [17, 30, 26]. These preconditioners exhibit a high degree of

*This paper was submitted, in modified form, to *SIAM Journal on Scientific Computing*.

‡Department of Computer Science, K.U.Leuven
({nick.vannieuwenhoven, karl.meerbergen}@cs.kuleuven.be)

parallelism and some can be applied to a vector using high-throughput dense matrix-vector multiplications. Unfortunately their quality is unsatisfactory to tackle more difficult linear systems.

A more general preconditioning technique, that does not assume an element structure, is the incomplete LU-factorization. It applies Gaussian elimination to A while discarding non-zeros in the factorization using heuristic rules. The simplest of these approaches is the no fill-in ILU(0) preconditioner [36], which drops all fill-in. This approach is effective on a variety of problems, including scalar elliptic PDEs [10]. The quality of the factorization may be improved by allowing some fill-in values. A strategy that works well in practice, is the dual-threshold strategy employed by ILUT [42]. A significant disadvantage of ILU-type preconditioners is their sequential nature during the forward and backward substitution, which makes it difficult to parallelize them efficiently [27, 2, 4]. Also, a sparse triangular solve attains limited throughput, according to [50].

During the past two decades, a significant amount of research focused on algebraic multigrid [45, 41, 11, 46] and multilevel-type preconditioners [6, 7, 44, 37, 13, 39]. In these approaches, the preconditioner is recursively constructed by applying a two-level scheme [38]. This scheme consists of partitioning the unknowns into two sets: fine and coarse. The fine variables are eliminated and the *reduced system*, which contains only the coarse variables, is approximately computed. Typically, the reduced system is an approximate Schur complement of the partitioned matrix or the Galerkin coarse grid matrix [38], which requires interpolation matrices as in multigrid methods. The two-level scheme is applied recursively to the coarse unknowns. These methods often yield high-quality preconditioners.

Nowadays, a preconditioner is nearly invariably stored as a sparse matrix. The major disadvantage of sparse data structures is their limited throughput in key operations, such as the matrix-vector product [49] and a sparse triangular solve [50]. Clearly, the total time spent on preconditioning, excluding the factorization, is the product of the number of iterations and the execution time per iteration. That implies that it is *equally* important to reduce the number of iterations as increasing the computational performance. We believe that a compromise should be reached between reducing the iteration count, by forming a more accurate factorization, and increasing the computational performance, by imposing additional restrictions to allow for dense operations. Current research appears to be focused on reducing the iteration count. Preconditioners could benefit from an element structure, as it naturally defines the sparse matrix as a sequence of overlapping dense matrices. We believe that this structure is not exploited effectively in available preconditioners; the EBE-type preconditioners are considered too ineffective, the ILU-type preconditioners are not naturally defined in terms of element matrices and the few preconditioners that do employ element-wise algorithms, e.g. [12, 11, 31, 40, 33, 5], use the element structure only during the factorization, whereas the cost of an iterative method is usually dominated by applying the preconditioner many times.

Multifrontal algorithms have already been adapted to compute preconditioners [12, 40, 28]. However, we believe these approaches can be improved. These papers are mainly concerned with an effective factorization phase. In the end, an incomplete LU-factorization is obtained, so that sparse triangular solves are required during the solution phase.

In this work, we propose a multifrontal algorithm to compute a multilevel block-ILU preconditioner for systems with an underlying element structure. The design

of our algorithm was inspired by the ARMS preconditioner [44]. The method we present, shares some characteristics with the multifrontal method by Duff and Reid [19]. In particular, the computational scheme operates element-wise and the Schur complement of (dense) frontal matrices is computed. The method we present, employs a symbolic discard rule for fill-in values. It can be interpreted as a number of supernode eliminations, as in multifrontal methods, organized into a number of levels. The first few levels are factored exactly. From a given level onwards, the factorization is inexact, but still operates in a multifrontal manner, i.e. using the element structure. During the factorization, IMF seeks to combine the quality of multilevel methods with the inherent degree of parallelism and throughput of multifrontal methods. We aim to use the element structure efficiently during the solution phase as well.

The paper is structured as follows. In the next section, the IMF factorization and its application to a vector are described. We discuss the partitioning of the unknowns, the multifrontal computation of the Schur complement and a strategy to discard fill-in values. In Section 3, the spectrum of the preconditioned system is investigated. A modification of the IMF preconditioner for symmetric positive definite systems is presented in Section 4. Thereafter, in Section 6, the numerical experiments are presented. The concluding remarks are presented in Section 7.

Concerning notation, the following conventions apply throughout. Upper-case letters denote matrices (A, M, D, L, U, S) and lower-case boldface letters denote vectors ($\mathbf{x}$, $\mathbf{z}$, $\mathbf{b}$). A set of elements is typeset in calligraphic upper-case letters ($\mathcal{E}$, $\mathcal{D}$) and its elements in lower-case letters (e, d). Upper-case calligraphic letters are also used to denote operators acting on elements ($\mathcal{I}(\cdot)$, $\mathcal{N}(\cdot)$). The letter f is reserved for the final level and $0 \leq k \leq f$ always denotes the level subscript of a matrix, vector, element or element set. An entity with a double subscript, e.g. $D_{k,i}$, is a shorthand for $(D_k)_i$. It does not denote the value $d_{k,i}$. For elements, the level subscript can be understood from the element set to which they belong. Therefore, their subscripts will not be displayed.

2. IMF preconditioning. The IMF preconditioner consists of a multilevel block ILU-factorization that is computed in a multifrontal manner. Our scheme can be considered an extension to the supernodal multifrontal method [19, 34]. In particular, we adapt the idea of frontal matrices to the computation of an incomplete Schur complement. The computations operate exclusively on the element structure $\mathcal{E}$ of the coefficient matrix A. The latter matrix is never assembled into a sparse matrix. Contrary to multifrontal methods, we do not employ the assembly tree [34] to guide the factorization. Rather, we propose an ordering strategy that automatically identifies a set of independent supernodes (also called *fronts*). After the elimination of said supernodes, in a fashion very similar to multifrontal methods, the factorization proceeds recursively with the variables that have not yet been eliminated. This scheme essentially groups the supernodes in a number of distinct levels, similarly to the supernodes at a given depth in the assembly tree. In IMF, every supernode corresponds to the indices of one element. The supernodes in a given level may be eliminated concurrently.

In general, the IMF factorization, when considered in assembled form, is recursively defined as follows. Let $M_0 := A_0 := A$. At level $k \geq 0$, the matrix A_k has the exact block-LU factorization

$$A_k^P := P_k A_k P_k^T := \begin{bmatrix} D_k & U_k \\ L_k & S_k \end{bmatrix} = \begin{bmatrix} I & 0 \\ L_k D_k^{-1} & I \end{bmatrix} \times \begin{bmatrix} D_k & U_k \\ 0 & A_{k+1} \end{bmatrix}, \tag{2.1}$$

where $A_{k+1} = S_k - L_k D_k^{-1} U_k$ and P_k is a permutation matrix such that D_k is a dense block diagonal matrix; i.e. the matrices on the diagonal are dense. We propose to approximate A_k^P in Eq. (2.1) by

$$M_k^P := P_k M_k P_k^T := \begin{bmatrix} \bar{D}_k & \bar{U}_k \\ \bar{L}_k & \bar{S}_k \end{bmatrix} \approx \begin{bmatrix} I & 0 \\ \bar{L}_k \bar{D}_k^{-1} & I \end{bmatrix} \times \begin{bmatrix} \bar{D}_k & \bar{U}_k \\ 0 & M_{k+1} \end{bmatrix}, \tag{2.2}$$

where $M_{k+1} \approx \bar{S}_k - \bar{L}_k \bar{D}_k^{-1} \bar{U}_k \approx A_{k+1}$. At every level k, M_k^P is an approximation to A_k^P. The inverse of the block diagonal matrix is explicitly computed, without dropping, and stored as a sequence of dense matrices. This allows us to employ dense matrix-vector multiplications, which is a key ingredient for computational efficiency during the application of the preconditioner. One may argue that, according to [29], applying the exact inverse of a matrix is less stable than Gaussian elimination with partial pivoting. Potentially losing a few digits of accuracy is not a disaster since M_k is only applied as a preconditioner. Accuracy is traded for computational efficiency here. The factors of $\bar{L}_k \bar{D}_k^{-1}$ are stored, rather than computing the (much denser) product. The matrices $\bar{L}_k$ and $\bar{U}_k$ are assembled and stored as sparse matrices, e.g. using compressed storage. Factorization (2.2) is recursively applied to M_{k+1}. The recursion halts once all nodes have been eliminated. At the final level f, $M_f^P = \bar{D}_f$. In the end, the IMF factorization results in an approximate block-LU factorization with recursive block structure:

$$\begin{aligned} M_{\text{IMF}} &= P \times \left[\begin{array}{c|ccc} I & & & \\ \hline & I & & \\ \bar{L}_0 \bar{D}_0^{-1} & & \ddots & \\ & \bar{L}_1 \bar{D}_1^{-1} & & \\ & & \cdots & \begin{array}{c|c} I & \\ \hline \bar{L}_{f-1} \bar{D}_{f-1}^{-1} & I \end{array} \end{array}\right] \times \left[\begin{array}{ccccc} \bar{D}_0 & & & & \bar{U}_0 \\ \hline & \bar{D}_1 & & & \bar{U}_1 \\ & & \ddots & & \vdots \\ & & & \bar{D}_{f-1} & \bar{U}_{f-1} \\ & & & & \bar{D}_f \end{array}\right] \times P^T \\ &= P L_M U_M P^T. \end{aligned} \tag{2.3}$$

Herein, P is the permutation matrix that combines the actions of the permutation matrices P_k, for all k. For notational convenience, we will drop the bars in Eq. (2.2).

Factorization (2.2) is stated in terms of the assembled matrix M_k. We will not compute the factorization in this manner, however. We propose a multifrontal algorithm to compute this factorization, that efficiently exploits an available element structure. Given an element structure, such an algorithm needs to construct an element structure for the reduced system. Otherwise the algorithm cannot be applied recursively to factor the reduced system. The design of an algorithm that transforms an element structure $\mathcal{E}_k$ of M_k into an element structure $\mathcal{E}_{k+1}$ for the reduced system M_{k+1} is the major topic in this section. We will always assume that an element structure $\mathcal{E}_0$ of M_0 is given. Such an element structure is, for instance, present in linear systems that were derived from a finite element discretization of a PDE.

The remainder of this section is structured as follows. In the next subsection, we briefly investigate the application of the IMF preconditioner to a vector. In Section 2.2, we discuss the selection of the independent supernodes, whose nodes are eliminated. The elimination of the supernodes can proceed in two different manners. We can either form the exact Schur complement, as in the multifrontal method, or an approximate Schur complement wherein certain values are discarded. The former strategy is discussed in Section 2.3, whereas the latter is the subject of Section 2.4. Finally, in Section 2.5, the IMF(κ) preconditioner is presented.

Let $\mathcal{A}$ be an operator on $\mathcal{E}$, then we define, for $\mathcal{S} \subseteq \mathcal{E}$, $\mathcal{A}(\mathcal{S}) := \bigcup_{s\in\mathcal{S}} \mathcal{A}(s)$. To simplify the discussion of the algorithms, we assume, without loss of generality, that the permutation matrix P in Eq. (2.3) has already been applied to A_0. In this manner, the permutation matrix P_k is the identity matrix at every level $k \geq 0$. Therefore, we can write $A_k^P = A_k$ and $M_k^P = M_k$, for all $k \geq 0$.

2.1. Applying the preconditioner. Here, we detail the application of the preconditioner to a vector. To solve $M_k\mathbf{x}_k = \mathbf{b}_k$, we require a block forward and backward substitution with the factors in Eq. (2.2). If we partition $\mathbf{x}_k$ and $\mathbf{b}_k$ compatibly with the block structure in Eq. (2.2),

$$\mathbf{b}_k^T = \begin{bmatrix}\mathbf{b}_{k,1}^T & \mathbf{b}_{k,2}^T\end{bmatrix} \qquad \text{and} \qquad \mathbf{x}_k^T = \begin{bmatrix}\mathbf{x}_{k,1}^T & \mathbf{x}_{k,2}^T\end{bmatrix},$$

then we should compute

$$\begin{aligned}
\mathbf{z}_{k,1} &:= \mathbf{b}_{k,1}, \\
\mathbf{z}_{k,2} &:= \mathbf{b}_{k,2} - L_k(D_k^{-1}\mathbf{z}_{k,1}), && (2.4)\\
\mathbf{x}_{k,2} &= M_{k+1}^{-1}\mathbf{z}_{k,2}, && (2.5)\\
\mathbf{x}_{k,1} &= D_k^{-1}(\mathbf{z}_{k,1} - U_k\mathbf{x}_{k,2}). && (2.6)
\end{aligned}$$

Because D_k^{-1} is stored explicitly, a series of high-throughput dense matrix-vector products can be applied in Eq. (2.4) and Eq. (2.6). Furthermore, indirect addressing is not required. Eq. (2.5) is solved for recursively. Sparse matrix-vector products are used in Eq. (2.4) and Eq. (2.6) to apply L_k and U_k respectively.

Notice that the above scheme does not operate fully element-wise during the solution phase. An element-wise mode of operation could be employed, however, by storing the element structure of L_k and U_k and using an element-by-element matrix-vector product. We did not expect to benefit from this, however. Consequently, the approach was not pursued here. It should be noted that such an element-by-element scheme might be interesting on throughput-oriented massively parallel architectures such as GPGPUs.

2.2. Partitioning the unknowns. In this subsection, we investigate the construction of a permutation P_k such that D_k is a dense block diagonal matrix in Eq. (2.2), given an element structure $\mathcal{E}_k$ of M_k. We propose a scheme wherein the matrices on the block diagonal correspond to element matrices. In this manner, the inverse can be computed explicitly without generating fill-in (as every element matrix is dense). It turns out that this has interesting consequences for the spectrum of the preconditioned matrix, as will be shown in Section 3.

In the terminology of multifrontal methods, we will select a set of independent supernodes, whereby each supernode consists of the variables of one element. In contrast to the multifrontal method, we do not utilize the assembly tree to determine such a set of independent supernodes. Rather, we will work on the graph representation directly. In this *element graph*, each element (or supernode) is represented by a node. Two nodes in this graph are connected if their corresponding elements share variables. We say that those elements are *neighbors*.

DEFINITION 2.1 (Neighborhood). *The set of neighbors of $e \in \mathcal{E}_k$ is*

$$\mathcal{N}(e) := \{c \in \mathcal{E}_k \mid \mathcal{I}(c) \cap \mathcal{I}(e) \neq \emptyset\}.$$

By definition, e neighbors itself.

The next theorem shows that if a subset of elements $\mathcal{D}_k \subseteq \mathcal{E}_k$ are only interconnected via paths of length three or more in the element graph, then numbering their unknowns first results in a block diagonal matrix.

THEOREM 2.2 (Pivotal elements). *Let $\mathcal{D}_k := \langle d_1, d_2, \ldots, d_m \rangle \subseteq \mathcal{E}_k$ and $\mathcal{E}_k$ be a set of elements. If $\mathcal{D}_k$ satisfies*

$$\forall d_i \neq d_j \in \mathcal{D}_k : \mathcal{N}(d_i) \cap \mathcal{N}(d_j) = \emptyset, \tag{2.7}$$

called the element-independence condition*, then the permutation matrix P_k, corresponding to the renumbering*

$$\mathbf{p}_k := \left[\begin{array}{c|c} \mathbf{q}_k & \mathbf{r}_k \end{array}\right] := \left[\begin{array}{ccc|c} \mathcal{I}(d_1) & \mathcal{I}(d_2) & \ldots \; \mathcal{I}(d_m) & \mathcal{I}(\mathcal{E}_k) \backslash \mathcal{I}(\mathcal{D}_k) \end{array}\right], \tag{2.8}$$

results in

$$P_k M_k P_k^T = \begin{array}{c} \\ \mathcal{I}(d_1) \\ \mathcal{I}(d_2) \\ \vdots \\ \mathcal{I}(d_m) \\ \mathbf{r}_k \end{array} \begin{array}{c} \begin{array}{cccc} \mathcal{I}(d_1) & \mathcal{I}(d_2) & \ldots & \mathcal{I}(d_m) \end{array} \\ \left[\begin{array}{cccc|c} D_{k,1} & & & & U_{k,1} \\ & D_{k,2} & & & U_{k,2} \\ & & \ddots & & \vdots \\ & & & D_{k,m} & U_{k,m} \\ \hline L_{k,1} & L_{k,2} & \ldots & L_{k,m} & S_k \end{array}\right] \end{array} := \begin{array}{c} \\ \mathbf{q}_k \\ \mathbf{r}_k \end{array} \begin{array}{c} \begin{array}{cc} \mathbf{q}_k & \mathbf{r}_k \end{array} \\ \left[\begin{array}{c|c} D_k & U_k \\ \hline L_k & S_k \end{array}\right] \end{array}. \tag{2.9}$$

We call $\mathcal{D}_k$ a set of pivotal elements*.*

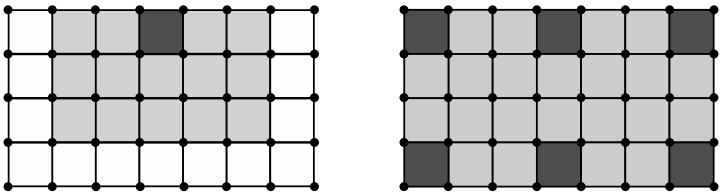
Proof. First, we remark that the sparsity of M_k is the union of the sparsities of the elements. That is because we assume that the numerical values in the element matrices do not sum to analytical zero. To prove the theorem, we demonstrate that if $\mathcal{D}_k$ satisfies condition (2.7), then for distinct $d_p, d_q \in \mathcal{D}_k$ and any combination of $i \in \mathcal{I}(d_p)$ and $j \in \mathcal{I}(d_q)$ it holds that $(M_k)_{i,j} = 0$. Assume, to the end of contradiction, that $\mathcal{D}_k$ satisfies Eq. (2.7), but that the above statement does not hold. Then there must be some element $e \in \mathcal{E}_k$ with $i, j \in \mathcal{I}(e)$. Therefore, $e \in \mathcal{N}(d_p)$ and $e \in \mathcal{N}(d_q)$, which forms a contradiction. Applying the renumbering (2.8) to M_k thus results in Eq. (2.9), demonstrating that D_k is a block diagonal matrix. ∎

Theorem 2.2 reduces the problem of determining a permutation matrix P_k to finding a set of pivotal elements. Condition (2.7) can also be written as

$$\forall d_i \neq d_j \in \mathcal{D}_k : d_j \notin \mathcal{N}_2(d_i),$$

where $\mathcal{N}_2(e) := \mathcal{N}(e) \cup \mathcal{N}(\mathcal{N}(e))$.

The element-independence condition and a set of pivotal elements is illustrated below, in the context of a finite element discretization of some rectangular domain.



Every square represents an element with four nodes. In the left panel, let e denote the dark gray element. The elements shaded in the lighter gray are those in $\mathcal{N}_2(e)$.

In the right panel, the set of dark gray elements is a set of pivotal elements $\mathcal{D}$. The elements in the lighter gray are those in $\mathcal{N}_2(\mathcal{D})$. Notice that the set $\mathcal{D}$ is maximal in the sense that no more elements can be added to it, without violating Eq. (2.7).

In Section 3, we will argue that the quality of factorization (2.2) may be improved by partitioning the unknowns so that the number of variables in the reduced system is as small as possible. Unfortunately, this problem is intractable.

THEOREM 2.3. *Let $\mathcal{E}_k$ be an element structure of A_k. There is no polynomial-time algorithm for the problem of determining the pivotal elements $\mathcal{D}_k^* \subseteq \mathcal{E}_k$ such that the number of variables in the reduced system is minimal, unless the complexity class (see [14]) P equals NP.*

Proof. Let us call this problem the maximum block independent set (MBIS) problem. We will prove the theorem by reducing the maximum independent set (MIS) problem to MBIS. The MIS problem consists of finding the largest set S of vertices in a graph, such that no vertices in S are connected. If the vertices in S are numbered before the other vertices, we obtain a block partitioned matrix in which the upper left matrix is a diagonal matrix. MIS is a well-known NP-Hard optimization problem [32].

Assume, to the end of contradiction, that the MBIS problem is in P; i.e. there exists an algorithm for MBIS that runs in polynomial time. We will demonstrate that the MIS problem can be solved in polynomial time, then. Given the problem of finding the MIS in a graph $G(V, E)$, where V are the vertices and $E \subseteq V \times V$ the edges, we transform it into a MBIS problem, as follows. For every node $v_i \in V$ we introduce a "prime element" $e_i \in \mathcal{E}$. For every edge $e = (v_i, v_j) \in E$, we introduce a "helper element" $h_{i,j} \in \mathcal{H} \subset \mathcal{E}$. Its index set is defined as $\mathcal{I}(h_{i,j}) = \{I_{i,j}^1, I_{i,j}^2\}$. The index set of e_i is extended with $I_{i,j}^1$ and the index set of e_j is extended with $I_{i,j}^2$. The introduced nodes (in the element structure) uniquely identify the connection between e_i and e_j, via the helper element $h_{i,j}$. In this manner, $e_i \in \mathcal{N}_2(e_j)$ iff $(v_i, v_j) \in E$. Finally, we ensure that the helper elements are never selected by an algorithm for MBIS. The index set of every helper element is extended with the single index I. In this manner, $h_{i,j} \in \mathcal{N}(h_{p,q})$ for any combination of $(v_i, v_j) \in E$ and $(v_p, v_q) \in E$. Notice that this modification does not influence the level-2 neighborhood of the prime elements $e \in \mathcal{E}\backslash\mathcal{H}$ in any other way than introducing all helper elements to it. The level-2 neighborhood of the prime element e_i is thus given by $\{e_j \mid (v_i, v_j) \in E\} \cup \mathcal{H}$. Finally, all elements should be extended with enough, globally unique, indices such that every element in $\mathcal{E}$ has an equal amount of indices. These added indices do not alter the neighborhood of any element, because the indices are all unique. Now that all elements have an equal amount of indices, the solution to the MBIS problem is also a solution to the problem of maximizing the number of elements in $\mathcal{D}^*$. If we now solve the MBIS problem for the element structure $\mathcal{E}$, then its output is a set of pivotal elements $\mathcal{D}^*$ such that the number of nodes in the reduced system is minimal, or equivalently, $|\mathcal{I}(\mathcal{D}^*)|$ is maximal. Normally, the output cannot contain helper elements, because their level-2 neighborhood is $\mathcal{E}$. In the case that the output is $h_{i,j} \in \mathcal{H}$, the element should be replaced by the prime element $e_i \in \mathcal{E}\backslash\mathcal{H}$. This modification is valid, because a helper element can only be in the output if the optimal set of pivotal elements contains only one element. Any prime element would then be a solution. In the general case, the output consists entirely of prime elements, because any prime element has the helper elements in their level-2 neighborhood. The solution to the MIS problem is simply the set of nodes corresponding to the prime elements in $\mathcal{D}^*$. Indeed, $\mathcal{D}^*$ is the largest set of elements that is element-independent. By

construction $(v_i, v_j) \notin E$ iff $e_i \notin \mathcal{N}_2(e_j)$. Consequently, the nodes corresponding to the elements in $\mathcal{D}^*$ are the MIS solution in the graph $G(V, E)$.

Demonstrating that MIS can be solved in a polynomial-time complexity given a polynomial-time algorithm for MBIS would conclude the proof. Clearly, the element structure $\mathcal{E}$ can be constructed in a polynomial time. The solution of MBIS can also be transformed in polynomial time to the solution of the MIS problem. Let us assume that $P \neq NP$. Since we know that MIS cannot be solved in polynomial time in this case, and the above algorithm would solve it in polynomial time, we conclude that a polynomial-time algorithm for MBIS cannot exist. ∎

It is thus infeasible to compute $\mathcal{D}_k^*$. Therefore, we will settle for an approximation. Consider what happens if $e \in \mathcal{E}_k$ is to be added to a set of pivotal elements $\mathcal{D}_k$. By selecting it, a node which is connected via an edge to at least one node of e will inevitably appear in the reduced system. Formally, the set of all such nodes is

$$\mathcal{I}_{\mathcal{N}}(e) := \mathcal{I}(\mathcal{N}(e)) \backslash \mathcal{I}(e). \tag{2.10}$$

To the end of minimizing the size of the reduced system, it is natural to prioritize elements whose *degree* $|\mathcal{I}_{\mathcal{N}}(e)|$ is small. Notice that we do not attempt to minimize the number of non-zeros in the reduced system, here. Once e has been added to the set of pivotal elements, the elements in $\mathcal{N}_2(e)$ can no longer be added to $\mathcal{D}_k$. They would otherwise violate independence condition (2.7). The above reflections lead to the following greedy algorithm.

Algorithm 1 Computing a set of pivotal elements

$\mathcal{M}_k \leftarrow \mathcal{E}_k$
$\mathcal{D}_k \leftarrow \emptyset$
while $\mathcal{M}_k \neq \emptyset$ **do**
  $e^* \leftarrow \arg\min_{e \in \mathcal{M}_k} \{|\mathcal{I}_{\mathcal{N}}(e)|\}$
  $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \{e^*\}$
  $\mathcal{M}_k \leftarrow \mathcal{M}_k \backslash \mathcal{N}_2(e^*)$
end while

Algorithm 1 serves a slightly different purpose than the minimum degree ordering algorithm [47, 23]. The latter is often employed to reduce the fill-in that results from a direct solution method. In this paper, however, we are concerned with minimizing the number of variables in the reduced system, because we can show that the spectrum of the preconditioned system improves.

A parallel version of the above algorithm is also possible. In such a setting the set of elements $\mathcal{E}$ would be partitioned and distributed among the available processors. In that manner, each processor would be assigned a subdomain. By marking the elements on the boundary of each subdomain, the above algorithm could be applied, unmodified, at each processor to the local element sets. That is because any unmarked element in one subdomain could not have an unmarked element from another subdomain in its level-2 neighbourhood. The *interface* elements, i.e. the elements on the subdomain boundaries, would then be eliminated in the last steps, as is common in domain decomposition methods.

2.3. Computing the exact Schur complement. In this subsection, we investigate a multifrontal algorithm to compute the exact Schur complement $M_{k+1} :=$

$S_k - L_k D_k^{-1} U_k$ of M_k in Eq. (2.9), given an element structure $\mathcal{E}_k$ of M_k and a set of pivotal elements $\mathcal{D}_k \subseteq \mathcal{E}_k$. The variables corresponding to the pivotal elements will be eliminated. Here, we describe a restatement, in the terminology of this paper, of a group of concurrent eliminations in a supernodal multifrontal method. An element structure of the Schur complement is presented. In the next subsection, we propose a strategy to discard fill-in values, based on that element structure.

From Eq. (2.9) it follows that the Schur complement can be written as

$$M_{k+1} = S_k - \sum_{i=1}^{m} L_{k,i} D_{k,i}^{-1} U_{k,i}. \tag{2.11}$$

Let us define the *frontal matrix* associated with $d_i \in \mathcal{D}_k$ as

$$F_{k,i} := \sum_{e \in \mathcal{N}(d_i)} M_k^{[e]} := \sum_{e \in \mathcal{N}(d_i)} \begin{bmatrix} D_k^{[e]} & U_k^{[e]} \\ L_k^{[e]} & S_k^{[e]} \end{bmatrix} := \begin{bmatrix} \hat{D}_{k,i} & \hat{U}_{k,i} \\ \hat{L}_{k,i} & \hat{S}_{k,i} \end{bmatrix}, \tag{2.12}$$

where $F_{k,i}$ is partitioned compatibly with Eq. (2.9). This definition is equivalent to its classical definition in Liu's review paper [34].

The frontal matrix is fully assembled at the indices of the diagonal element d_i, as in the multifrontal method. This is demonstrated next. Consider an element $e \in \mathcal{E}_k$ that does not neighbor d_i. By definition, its index set does not overlap with $\mathcal{I}(d_i)$. Its element matrix can thus not contribute to $L_{k,i}$, $D_{k,i}$ and $U_{k,i}$ in Eq. (2.9). Conversely, a neighbor of d_i cannot contribute to $L_{k,j}$, $D_{k,j}$ and $U_{k,j}$ if $j \neq i$. That is because its index set does not overlap with the index sets of any other pivotal element, as the converse would contradict property (2.7). Consequently, the frontal matrix of d_i is

$$F_{k,i} = \begin{array}{c} \\ \mathcal{I}(d_1) \\ \vdots \\ \mathcal{I}(d_{i-1}) \\ \mathcal{I}(d_i) \\ \mathcal{I}(d_{i+1}) \\ \vdots \\ \mathcal{I}(d_m) \\ \mathbf{r}_k \end{array} \begin{array}{c} \begin{array}{ccccccccc} \mathcal{I}(d_1) & \dots & \mathcal{I}(d_{i-1}) & \mathcal{I}(d_i) & \mathcal{I}(d_{i+1}) & \dots & \mathcal{I}(d_m) & & \mathbf{r}_k \end{array} \\ \left[\begin{array}{ccccccc|c} 0 & & & & & & & & 0 \\ & \ddots & & & & & & & \vdots \\ & & 0 & & & & & & 0 \\ & & & D_{k,i} & & & & & U_{k,i} \\ & & & & 0 & & & & 0 \\ & & & & & \ddots & & & \vdots \\ & & & & & & 0 & & 0 \\ \hline 0 & \dots & 0 & L_{k,i} & 0 & \dots & 0 & & \hat{S}_{k,i} \end{array}\right] \end{array}, \tag{2.13}$$

where $\mathbf{r}_k$ is as in Eq. (2.9). This concludes the proof that the frontal matrix is fully assembled at the indices $\mathcal{I}(d_i)$. Clearly, we are not the first to present such a proof.

By combining Eq. (2.11), Eq. (2.12) and Eq. (2.13) we obtain

$$\begin{aligned} M_{k+1} &= \left(\sum_{e \in \mathcal{H}_k} S_k^{[e]} + \sum_{d_i \in \mathcal{D}_k} \sum_{e \in \mathcal{N}(d_i)} S_k^{[e]} \right) - \sum_{d_i \in \mathcal{D}_k} \hat{L}_{k,i} \hat{D}_{k,i}^{-1} \hat{U}_{k,i} \\ &= \sum_{e \in \mathcal{H}_k} S_k^{[e]} + \sum_{d_i \in \mathcal{D}_k} \left(\hat{S}_{k,i} - \hat{L}_{k,i} \hat{D}_{k,i}^{-1} \hat{U}_{k,i} \right), \end{aligned} \tag{2.14}$$

where $\mathcal{H}_k := \mathcal{E}_k \backslash \mathcal{N}(\mathcal{D}_k)$. If we now define the element matrices at the next level by

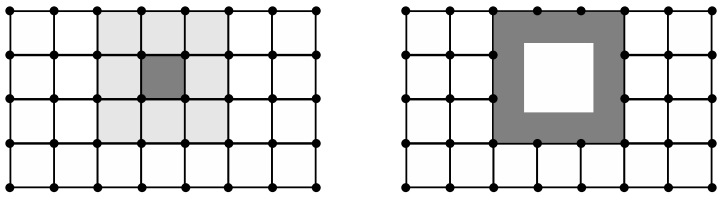
$$M_{k+1}^{[h]} := S_k^{[h]} \text{ if } h \in \mathcal{H}_k \qquad \text{and} \qquad M_{k+1}^{[g_i]} := \hat{S}_{k,i} - \hat{L}_{k,i} \hat{D}_{k,i}^{-1} \hat{U}_{k,i} \text{ if } d_i \in \mathcal{D}_k,$$

where g_i is a new element, then

$$\mathcal{E}_{k+1} := \mathcal{H}_k \cup \mathcal{G}_k \text{ where } \mathcal{G}_k := \{g_i \,|\, d_i \in \mathcal{D}_k\}, \tag{2.15}$$

is an element structure of M_{k+1}. The exact Schur complement of M_k can thus be computed, element-wise, by computing the Schur complements of the (dense) frontal matrices $F_{k,i}$, for $d_i \in \mathcal{D}_k$. The Schur complement of the frontal matrix $F_{k,i}$ can itself be interpreted as the element matrix of a new element g_i. The index set of this new element is $\mathcal{I}_{\mathcal{N}}(d_i)$, wherein the operator is as in Eq. (2.10). In [19], the elements in $\mathcal{G}_k$ are referred to as "generated" or "generalized" elements, while in [34] they are called "update matrices".

As an example, consider the following finite element discretizations.



In the left panel, let us denote the dark gray element by d. The neighbors of d are shaded in the lighter gray. Assume that d is the only pivotal element. By eliminating the dofs[1] of d, we generate a new element g whose index set is $\mathcal{I}_{\mathcal{N}}(d)$. This generated element g corresponds to the dark gray element in the right panel, which is the element structure we obtain after eliminating the pivotal element.

We summarize the above results in Algorithm 2. Notice that the frontal matrices are treated as dense matrices. This approach is feasible since the size of the frontal matrices is very small relative to the size of A. It enables us to compute the direct inverse via LAPACK [3] routines and the matrix product via the high-throughput BLAS3 [18] routine. However, superfluous computations may result from the explicit representation of some structural zeros. Also notice that the iterations in each loop can proceed independently from one another. This enables a large degree of parallelism during the computationally intensive phase of the factorization.

Algorithm 2 Multifrontal computation of the Schur complement

$\mathcal{E}_{k+1} \leftarrow \emptyset$
for $h \in \mathcal{H}_k$ **in parallel do**
  $M_{k+1}^{[h]} \leftarrow S_k^{[h]}$
  $\mathcal{E}_{k+1} \leftarrow \mathcal{E}_{k+1} \cup \{h\}$
end for
for $d_i \in \mathcal{D}_k$ **in parallel do**
  {Create a new element g_i}
  $\mathcal{I}(g_i) \leftarrow \mathcal{I}(\mathcal{N}(d_i)) \backslash \mathcal{I}(d_i)$
  $M_{k+1}^{[g_i]} \leftarrow \hat{S}_{k,i} - \hat{L}_{k,i}\hat{D}_{k,i}^{-1}\hat{U}_{k,i}$
  $\mathcal{E}_{k+1} \leftarrow \mathcal{E}_{k+1} \cup \{g_i\}$
end for

[1] The dofs of an element are the unknowns corresponding to the index set of the element.

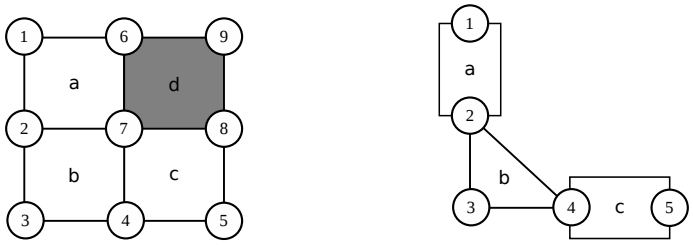
2.4. Computing an approximate Schur complement. The element structure (2.15) assembles to the exact Schur complement of M_k. Applying this technique at every level would result in a multifrontal direct solution method. Here, we propose a symbolic dropping rule to limit the fill-in that is generated by the generated elements. The proposed algorithm operates in a multifrontal manner.

The sparsity of M_k is always the union of the sparsities of the elements in $\mathcal{E}_k$. We believe it is thus not feasible to employ a dropping rule based on single positions. This would just set a value to zero, even though that position would still be in the sparsity pattern of some elements. It seems more natural to approximate the Schur complement by *dropping* elements from element structure (2.15). As fill-in can only occur within the generated elements $\mathcal{G}_k$, we will only drop elements there.

An alternative approach consists of modifying the elements in $\mathcal{G}_k$ structurally, by changing their index sets. For instance, an index could be dropped from an element, if some norm of the corresponding row and column is small w.r.t. the entire matrix. This would allow for more involved dropping strategies, with potentially better performance.

In the next theorem, we propose a multifrontal technique to drop all fill-in values in the Schur complement, while retaining an element structure. First, we outline the approach. Consider again the multifrontal algorithm to compute the exact Schur complement. Therein, a generated element $g_i \in \mathcal{G}_k$ essentially replaces the pivotal element $d_i \in \mathcal{D}_k$ and its neighbors $\mathcal{N}(d_i)$. The incomplete method we propose, on the other hand, will not introduce the generated element g_i to the element structure for the reduced system. Instead, the neighbors of the diagonal element d_i will be added to the element structure of the next level. Their element matrices will be adapted so that they assemble to the non-fill-in values of the generated element g_i. The fill-in values are thus discarded.

Example. Assume that we are given the elements displayed in the left panel.



Herein $d \in \mathcal{D}_k$ is the sole pivotal element. Let us assume that the element matrix of the generated element g, resulting from the elimination of d as described in the previous subsection, is

$$G = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \end{bmatrix} \end{array}.$$

In multifrontal algorithm 2, the element structure of the exact Schur complement would consist solely out of the generated element g. In our incomplete multifrontal method, however, the generated element will not be added to the element structure of the (approximate) Schur complement. The neighbors of the diagonal element (a, b, c) are present in the element structure of the approximate Schur complement;

$\mathcal{E}_{k+1} = \{a, b, c\}$. This is illustrated in the right panel. Notice in particular that the dofs of the pivotal element d have been removed and that d itself is no longer present at the next level. The element matrices of a, b and c will be chosen so that they assemble to G at the positions in the sparsity of $\mathcal{E}_{k+1}$. These element matrices can be constructed algorithmically. We just store the values of G in the first[2] element that can store the values, if any. The element matrices would thus be

$$M_{k+1}^{[a]} = \begin{array}{c} \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \begin{bmatrix} 11 & 12 \\ 21 & 22 \end{bmatrix} \end{array}, \quad M_{k+1}^{[b]} = \begin{array}{c} \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 23 & 24 \\ 32 & 33 & 34 \\ 42 & 43 & 44 \end{bmatrix} \end{array}, \quad M_{k+1}^{[a]} = \begin{array}{c} \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cc} 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 45 \\ 54 & 55 \end{bmatrix} \end{array}.$$

Notice that the element matrices may overlap. For instance, both a and b have position $(2, 2)$ in their sparsity. This introduces some degrees of freedom in selecting the element matrices for the reduced system. The above process of storing the values of a generated element in the other elements will be referred to as *distribution*. In essence, distribution is the inverse of assembling the element matrices. Given the above element matrices, it is easy to verify that they assemble to

$$G' = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 11 & 12 & 0 & 0 & 0 \\ 21 & 22 & 23 & 24 & 0 \\ 0 & 32 & 33 & 34 & 0 \\ 0 & 42 & 43 & 44 & 45 \\ 0 & 0 & 0 & 54 & 55 \end{bmatrix} \end{array},$$

which is exactly equal to the values of G, except for the fill-in values. These are zero in the reduced system.

THEOREM 2.4 (Computing the Schur complement without fill-in). *Let*

$$M_k = \sum_{e \in \mathcal{E}_k} M_k^{[e]} = \begin{bmatrix} D_k & U_k \\ L_k & S_k \end{bmatrix}$$

be the approximation to A_k at the current level. If an element structure $\mathcal{E}_{k+1}$ is obtained by inserting, for every element $e \in \mathcal{E}_k \backslash \mathcal{D}_k$, a new element $e' \in \mathcal{E}_{k+1}$ whose element matrix is defined by

$$M_{k+1}^{[e']} := S_k^{[e]} - \sum_{d_i \in \mathcal{D}_k} \left(\omega_{k+1}^{[e']} \circ \left(\hat{L}_{k,i} \hat{D}_{k,i}^{-1} \hat{U}_{k,i} \right) \right), \tag{2.16}$$

where $\circ$ is the Hadamard matrix multiplication[3], and the matrices $\hat{L}_{k,i}$, $\hat{D}_{k,i}$ and $\hat{U}_{k,i}$ are as in Eq. (2.12), and the set of distribution matrices

$$\Omega := \{\omega_{k+1}^{[e']} \mid e' \in \mathcal{E}_{k+1}, \}$$

satisfies the two constraints

$$\forall (p, q) \in \mathrm{Sp}\left(\mathcal{E}_{k+1}\right) : \sum_{e' \in \mathcal{E}_{k+1}} \left(\omega_{k+1}^{[e']} \right)_{p,q} = 1, \tag{2.17}$$

$$\forall e' \in \mathcal{E}_{k+1} \backslash \mathcal{E}_{p,q} : \left(\omega_{k+1}^{[e']} \right)_{p,q} = 0, \tag{2.18}$$

[2] In this example we assume that a lexicographic ordering is defined on the names of the element.
[3] This product is defined as $[a_{ij} b_{ij}]_{i,j=1}^n$ for $A = [a_{ij}]_{i,j=1}^n$ and $B = [a_{ij}]_{i,j=1}^n$.

wherein

$$\mathcal{E}_{p,q} := \{e' \in \mathcal{E}_{k+1} \,|\, p,q \in \mathcal{I}(e')\} \text{ and } \mathcal{I}(e') := \mathcal{I}(e)\backslash\mathcal{I}(\mathcal{D}_k), \tag{2.19}$$

then

$$\forall e' \in \mathcal{E}_{k+1} : \mathrm{Sp}\left(M_{k+1}^{[e']}\right) = \mathcal{I}(e') \times \mathcal{I}(e'), \tag{2.20}$$

which furthermore entails that

$$\mathrm{Sp}\left(\mathcal{E}_{k+1}\right) := \mathrm{Sp}\left(\mathcal{E}_k\right) \cap \left(\mathcal{I}(\mathcal{E}_k)\backslash\mathcal{I}(\mathcal{D}_k)\right) \times \left(\mathcal{I}(\mathcal{E}_k)\backslash\mathcal{I}(\mathcal{D}_k)\right),$$

and $\mathcal{E}_{k+1}$ *is an element structure of*

$$\left(M_{k+1}\right)_{p,q} = \begin{cases} \left(S_k - L_k D_k^{-1} U_k\right)_{p,q} & \text{if } (p,q) \in \mathrm{Sp}\left(\mathcal{E}_{k+1}\right) \\ 0 & \text{otherwise} \end{cases}, \tag{2.21}$$

i.e.

$$M_{k+1} = \sum_{e' \in \mathcal{E}_{k+1}} M_{k+1}^{[e']}. \tag{2.22}$$

Proof. First, we show that the set of equations Eq. (2.17) and Eq. (2.18) has a solution Ω. Clearly, Eq. (2.18) is satisfied trivially. We should demonstrate that an assignment consistent with Eq. (2.18) does not violate the possibility to satisfy Eq. (2.17). The only way in which Eq. (2.17) could not be satisfied is when all matrices $\omega_{k+1}^{[e']}$ are zero for some $(p,q) \in \mathrm{Sp}\left(\mathcal{E}_{k+1}\right)$, due to Eq. (2.18). However, that would entail that (p,q) is not in the sparsity of any element in $\mathcal{E}_{k+1}$, which forms a contradiction. Hence, there are always enough degrees of freedom to satisfy both equations. Furthermore, it is easy to show that the number of terms, and hence degrees of freedom, in satisfying Eq. (2.17) for a given $(p,q) \in \mathrm{Sp}\left(\mathcal{E}_{k+1}\right)$ equals the number of elements in $\mathcal{E}_{p,q}$.

Second, we demonstrate that Eq. (2.20) holds for any $e' \in \mathcal{E}_{k+1}$. Clearly, the sparsity of $S_k^{[e]}$ is $\mathcal{I}(e') \times \mathcal{I}(e')$. From the definition of the Hadamar product, we only need to demonstrate that

$$\mathrm{Sp}\left(\omega_{k+1}^{[e']}\right) \subseteq \mathcal{I}(e') \times \mathcal{I}(e'),$$

holds. From Eq. (2.18) and Eq. (2.19) it follows that if $p \notin \mathcal{I}(e')$ or $q \notin \mathcal{I}(e')$ then $(\omega_{k+1}^{[e']})_{p,q} = 0$. The contraposition of the last statement concludes the proof. This entails that every element in $\mathcal{E}_{k+1}$ can still be stored as an index set and a dense matrix, containing only the values in $\mathcal{I}(e') \times \mathcal{I}(e')$.

Finally, we demonstrate that Eq. (2.22) holds. First, let $(p,q) \in \mathrm{Sp}\left(\mathcal{E}_{k+1}\right)$. Consider now Eq. (2.16) at this position. By summing over all elements in $\mathcal{E}_{k+1}$ we obtain

$$\left(\sum_{e' \in \mathcal{E}_{k+1}} M_{k+1}^{[e']}\right)_{p,q} = \left(\sum_{e \in \mathcal{E}_k \backslash \mathcal{D}_k} S_k^{[e]}\right)_{p,q} - \left(\sum_{e' \in \mathcal{E}_{k+1}} \left(\omega_{k+1}^{[e']}\right)_{p,q}\right) \circ \sum_{d_i \in \mathcal{D}_k} \left(\hat{L}_{k,i} \hat{D}_{k,i}^{-1} \hat{U}_{k,i}\right)_{p,q}$$

In the right-hand side we note that, by Eq. (2.17), the distribution matrices sum to one at (p, q). Furthermore we note that $S_k^{[e]}$ is zero if $e \in \mathcal{D}_k$. Combining those observations with Eq. (2.14), it follows that the above equation reduces to

$$\left(\sum_{e' \in \mathcal{E}_{k+1}} M_{k+1}^{[e']} \right)_{p,q} = \left(S_k - L_k D_k^{-1} U_k \right)_{p,q} = \left(M_{k+1} \right)_{p,q} \qquad \text{if } (p,q) \in \mathrm{Sp}\left(\mathcal{E}_{k+1} \right),$$

which proves the first case in Eq. (2.21). Next, let $(p, q) \notin \mathrm{Sp}\left(\mathcal{E}_{k+1}\right)$. The sparsity of $S_{k+1}^{[e']}$ is always a subset of the sparsity of $\mathcal{E}_{k+1}$, and so $(S_{k+1}^{[e']})_{p,q} = 0$. Furthermore, from Eq. (2.19) it follows that the set $\mathcal{E}_{p,q}$ is empty. Therefore, it follows from Eq. (2.18) that $\omega_{k+1}^{[e']}$ is zero at (p, q) for any $e' \in \mathcal{E}_{k+1}$. Consequently,

$$\left(\sum_{e' \in \mathcal{E}_{k+1}} M_{k+1}^{[e']} \right)_{p,q} = 0 = \left(M_{k+1} \right)_{p,q} \qquad \text{if } (p,q) \notin \mathrm{Sp}\left(\mathcal{E}_{k+1} \right).$$

This proves the second case in Eq. (2.21) and concludes the proof. ∎

We proceed to describe a multifrontal scheme to compute the element structure of the reduced system, based on the previous theorem. Let us define the *modified frontal matrix* associated with $d_i \in \mathcal{D}_k$ as

$$F_{k,i}^0 := \sum_{e \in \mathcal{N}(d_i)} \begin{bmatrix} D_k^{[e]} & U_k^{[e]} \\ L_k^{[e]} & 0 \end{bmatrix} := \begin{bmatrix} \hat{D}_{k,i} & \hat{U}_{k,i} \\ \hat{L}_{k,i} & 0 \end{bmatrix}$$

The matrices $\hat{L}_{k,i}$, $\hat{D}_{k,i}$ and $\hat{U}_{k,i}$ are exactly those in Eq. (2.12). The modified frontal matrix $F_{k,i}^0$ differs from the frontal matrix $F_{k,i}$ in Eq. (2.12) only in the lower-right block. Consider now the Schur complement of the modified frontal matrix $F_{k,i}^0$. This matrix can, as in the exact case, be interpreted as the element matrix of a (modified) generated element g_i^0. In fact, the sparsity of this element is $\mathcal{I}_{\mathcal{N}}(d_i) \times \mathcal{I}_{\mathcal{N}}(d_i)$, as in the exact case. In contrast to the exact case, the generated element g_i^0 is not added to the element structure of the reduced system. Instead, the values of its element matrix are distributed over the elements in $\mathcal{E}_k \backslash \mathcal{D}_k$, which are added to $\mathcal{E}_{k+1}$. Eq. (2.16) can thus be computed in a multifrontal manner, very similar to the exact case. For every pivotal element $d_i \in \mathcal{D}_k$, the modified frontal matrix $F_{k,i}^0$ is constructed. Its Schur complements results in a modified generated element g_i^0. An element $e \in \mathcal{E}_k \backslash \mathcal{D}_k$ is then updated by adding the Hadamard product of its distribution matrix and the modified generated elements to its element matrix.

It should be noted that many of the Hadamard products in Eq. (2.16) are actually zero. Consider an element $e \in \mathcal{E}_k \backslash \mathcal{D}_k$ and its corresponding element $e' \in \mathcal{E}_{k+1}$. The sparsity of its distribution matrix is $\mathcal{I}(e') \times \mathcal{I}(e')$. The sparsity of the modified generated element g_i^0 resulting from the elimination of the pivotal element $d_i \in \mathcal{D}_k$, on the other hand, is $\mathcal{I}_{\mathcal{N}}(d_i) \times \mathcal{I}_{\mathcal{N}}(d_i)$. Clearly, the Hadamard product

$$\omega_{k+1}^{[e']} \circ (\hat{L}_{k,i} \hat{D}_{k,i}^{-1} \hat{U}_{k,i}),$$

will be zero, unless there is some matching index $p \in \mathcal{I}(e') \cap \mathcal{I}_{\mathcal{N}}(d_i)$. Recall that $\mathcal{I}_{\mathcal{N}}(d_i)$ is the set of indices directly connected to at least one of the indices in $\mathcal{I}(d_i)$ (in

the element structure $\mathcal{E}_k$). Therefore, $\mathcal{I}(e') \cap \mathcal{I}_{\mathcal{N}}(d_i) \neq \emptyset$ if e neighbors[4] an element that itself neighbors d_i. Formally,

$$\text{if } d_i \in \mathcal{N}_2(e) \text{ then } \mathcal{I}(e') \cap \mathcal{I}_{\mathcal{N}}(d_i) \neq \emptyset.$$

Consequently, we can rewrite Eq. (2.16) to

$$M_{k+1}^{[e']} := S_k^{[e]} - \sum_{d_i \in \mathcal{R}(e)} \left(\omega_{k+1}^{[e']} \circ \left(\hat{L}_{k,i} \hat{D}_{k,i}^{-1} \hat{U}_{k,i} \right) \right) \text{ where } \mathcal{R}(e) := \mathcal{D}_k \cap \mathcal{N}_2(e). \quad (2.23)$$

This scheme is much more economical.

The distribution matrices are more useful as a theoretical device to prove the correctness of our approach, than as a practical means to update the element matrices. In practice, we distribute the numerical values as follows. Consider the modified generated element g_i^0, associated with the pivotal element $d_i \in \mathcal{D}_k$. For every level-2 neighbor $e \in \mathcal{N}_2(d_i)$ we add, to the matching positions of $M_{k+1}^{[e]}$, the values of the element g_i^0. At the corresponding positions in g_i^0 zero is stored. In this scheme, the distribution matrices are entirely zero except for $\omega_{p,q}^{[e]} = 1$ iff e is the first element in $\mathcal{E}_{p,q}$, assuming some total order on the elements. The existence of these distribution matrices proves the correctness of Algorithm 3.

We can combine the above ingredients into a practical multifrontal algorithm to compute the approximate Schur complement (2.21). It is presented as Algorithm 3. As in Algorithm 2, the computation of the Schur complements of the (modified) frontal matrices may proceed concurrently, using BLAS3 and LAPACK routines. However, the updates to the elements should be synchronized, as it is possible that an element $e' \in \mathcal{E}_{k+1}$ is updated via multiple modified generated elements.

2.5. The IMF(κ) preconditioner. We propose a multifrontal multilevel block-ILU preconditioner, using the algorithms from the previous subsections. The preconditioner has a single parameter $\kappa \geq 0$ which controls two major properties: the amount of fill-in and the quality of the factorization. Higher values of κ result in more fill-in and more eigenvalues will be clustered around unity.

The IMF(κ) factorization constructs a preconditioner $M \approx A$ for the linear system (1.1). At every level $k \geq 0$, Algorithm 1 constructs a set of pivotal elements $\mathcal{D}_k$ from $\mathcal{E}_k$. To compute the element structure $\mathcal{E}_{k+1}$ of the reduced system, IMF(κ) distinguishes between two cases. In the case $k < \kappa$, Algorithm 2 computes the exact Schur complement. In the case $k \geq \kappa$, the approximate Schur complement is computed using Algorithm 3. In either case, the reduced system is recursively factored. The recursion halts at level k if $\mathcal{D}_k = \mathcal{E}_k$, i.e. the pivotal elements make up the entire element set. In this case, only the inverse of the block diagonal matrix should be computed. The Schur complement is empty.

3. Spectrum. In this section we investigate the spectrum of the preconditioned system $M^{-1}A$ when IMF(κ) is employed as preconditioner.

In IMF(κ), the *level of fill-in* parameter κ controls the quality of the approximation. One interesting property of this preconditioner is that

$$M_k = A_k \text{ for all } 0 \leq k \leq \kappa, \quad (3.1)$$

[4]The neighborhood relation should be considered in the element structure $\mathcal{E}_k$, as d_i is only defined therein.

Algorithm 3 Multifrontal computation of the approximate Schur complement

$\mathcal{E}_{k+1} \leftarrow \emptyset$
for $d_i \in \mathcal{D}_k$ **in parallel do**
  $G_{k,i} \leftarrow -\hat{L}_{k,i}\hat{D}_{k,i}^{-1}\hat{U}_{k,i}$
  for $e \in \mathcal{N}_2(d_i)\backslash\{d_i\}$ **in sequence do**
    {Create a new element e'}
    $\mathcal{I}(e') \leftarrow \mathcal{I}(e)\backslash\mathcal{I}(\mathcal{D}_k)$
    $M_{k+1}^{[e']} \leftarrow S_k^{[e]}$
    for $(p,q) \in \mathrm{Sp}\,(e')$ **and** $(p,q) \in \mathrm{Sp}\,(G_{k,i})$ **do**
      $\left(M_{k+1}^{[e']}\right)_{p,q} \leftarrow \left(M_{k+1}^{[e']}\right)_{p,q} + (G_{k,i})_{p,q}$
      $(G_{k,i})_{p,q} \leftarrow 0$
    end for
    $\mathcal{E}_{k+1} \leftarrow \mathcal{E}_{k+1} \cup \{e'\}$
  end for
end for
for $e \in \mathcal{E}_k\backslash\mathcal{N}_2(\mathcal{D}_k)$ **in parallel do**
  $M_{k+1}^{[e]} \leftarrow S_k^{[e]}$
  $\mathcal{E}_{k+1} \leftarrow \mathcal{E}_{k+1} \cup \{e\}$
end for

which can be proved easily. Therefore, IMF(∞) is a restatement of the supernodal multifrontal method, albeit with a primitive ordering strategy.

We can expect more accurate factorizations when κ is increased, as we now demonstrate. If we let $0 \leq k \leq \kappa$, then from Eq. (2.1), Eq. (2.2) and Eq. (3.1), we find

$$\begin{aligned} M_k^{-1}A_k &= \begin{bmatrix} D_k^{-1} & -D_k^{-1}U_kM_{k+1}^{-1} \\ 0 & M_{k+1}^{-1} \end{bmatrix} \times \begin{bmatrix} I & 0 \\ -L_kD_k^{-1} & I \end{bmatrix} \times \begin{bmatrix} D_k & U_k \\ L_k & S_k \end{bmatrix} \\ &= \begin{bmatrix} I & -D_k^{-1}U_k(I - M_{k+1}^{-1}A_{k+1}) \\ 0 & M_{k+1}^{-1}A_{k+1} \end{bmatrix}. \end{aligned}$$

The matrix in the lower right corner is again of the same form if $k < \kappa$. Finally at level κ, we approximate $A_{\kappa+1}$ by Eq. (2.21), so that $M_{\kappa+1}^{-1}A_{\kappa+1}$ is no longer an upper triangular matrix. From the above description we learn that the IMF(κ) preconditioned system is, in general, of the following form

$$M^{-1}A = \left[\begin{array}{c|ccc} I & \multicolumn{3}{c}{X_0} \\ \hline & I & \multicolumn{2}{c}{X_1} \\ \cline{2-4} & & \ddots & \vdots \\ \cline{3-4} & & I & X_\kappa \\ \cline{3-4} & & & M_{\kappa+1}^{-1}A_{\kappa+1} \end{array}\right], \tag{3.2}$$

where $X_k = -D_k^{-1}U_k(I - M_{k+1}^{-1}A_{k+1})$. The spectrum of the preconditioned system

is therefore $\sigma(M^{-1}A) = \{1\} \cup \sigma(M_{\kappa+1}^{-1}A_{\kappa+1})$, where eigenvalue 1 has multiplicity

$$\sum_{k=0}^{\kappa} |\mathcal{I}(\mathcal{D}_k)|.$$

It can be beneficial to maximize this quantity, as is attempted by Algorithm 1. Notice that even the no-fill IMF(0) preconditioner has some eigenvalues at unity, assuming exact arithmetic.

4. Preconditioning symmetric positive definite (spd) systems. Next, we consider the case where A is an spd matrix. If the preconditioner M is also spd, then the preconditioned conjugate gradient method can be employed to solve the system. For spd systems the following factorization can be employed. It is a minor variant of Eq. (2.2).

$$M_k := \begin{bmatrix} D_k & E_k^T \\ E_k & S_k \end{bmatrix} \approx \begin{bmatrix} L_k & 0 \\ E_k L_k^{-T} & M_{k+1}^{1/2} \end{bmatrix} \times \begin{bmatrix} L_k^T & L_k^{-1}E_k^T \\ 0 & (M_{k+1}^{1/2})^T \end{bmatrix} := M_k^{1/2}(M_k^{1/2})^T, \quad (4.1)$$

where $M_{k+1} = M_{k+1}^{1/2}(M_{k+1}^{1/2})^T$ is an incomplete Cholesky decomposition of $M_{k+1} \approx S_k - E_k L_k^{-T} L_k^{-1} E_k^T$. $D_k = L_k L_k^T$ is the Cholesky decomposition of the block diagonal matrix D_k. The approximation M_{k+1} is again computed either via Algorithm 2 or Algorithm 3, which preserves symmetry. Positive definiteness is preserved by Algorithm 2 but not necessarily by Algorithm 3. Assuming that the Cholesky decomposition of D_k exists, the above factorization is mathematically equivalent to Eq. (2.2). Applying the former recursively at every level, yields a symmetric version of Eq. (2.3),

$$A \approx M = L_M L_M^T. \quad (4.2)$$

The preconditioner is thus an incomplete Cholesky decomposition. Notice that D_k is factored using a complete, rather than an incomplete, Cholesky decomposition. Its existence is guaranteed whenever D_k is spd, i.e. no negative pivots occur during the factorization of D_k. It is straightforward to prove that this condition is satisfied for M- and H-matrices.

THEOREM 4.1. *If A is an M-matrix (see [36]) or an H-matrix (see [35]), then the incomplete Cholesky decomposition (4.2) exists.*

Proof. The proof follows by noticing that factorization Eq. (4.1) is an incomplete Cholesky decomposition of M_k. Let A be an M- or H-matrix. If we assume that M_k is an M- or H-matrix then this property also holds for M_{k+1} which was essentially proven in [36] and [35] for M- and H-matrices respectively. That is because the computation of M_{k+1} can be understood as a sequence of single pivot eliminations. ∎

In case A is spd but not an M- or H-matrix, factorization (4.1) can be modified using standard techniques to preserve spd-ness in the reduced system [10]. For instance, the diagonal elements could be shifted with the one-norm of the discarded fill-in values, in every row.

The spectrum of the preconditioned system is, in the spd case, given by

$$\sigma(L_M^{-1}AL_M^{-T}) = \{1\} \cup \sigma(M_{\kappa+1}^{-1/2}A_{\kappa+1}(M_{\kappa+1}^{1/2})^{-T}).$$

Let $A = L_A L_A^T$ be the Cholesky decomposition of A. Clearly, $L_M^{-1}L_A L_A^T L_M^{-T}$ is symmetric. Therefore, the condition number of the preconditioned system is

$$\kappa(L_M^{-1}AL_M^{-T}) = \left| \frac{\lambda_{\max}(L_M^{-1}AL_M^{-T})}{\lambda_{\min}(L_M^{-1}AL_M^{-T})} \right| = \frac{\max\{1, |\lambda_{\max}(R_{\kappa+1})|\}}{\min\{1, |\lambda_{\min}(R_{\kappa+1})|\}},$$

where $R_{\kappa+1} := M_{\kappa+1}^{-1/2} A_{\kappa+1} (M_{\kappa+1}^{1/2})^{-T}$. The above bound is a particular case of Theorem 9 in [38].

5. Implementation issues. In this section, we discuss some technical details of our implementation and point out discrepancies between it and the algorithms presented in this paper.

We employ a variant of Algorithm 1 wherein the degrees of the elements are not updated. Consequently, we only need to sort the elements once. This simpler strategy turns out to be competitive with an implementation of Algorithm 1 using a binary heap to update the degrees. The latter often required less iterations to converge than the former. Except for linear systems that proved to be difficult to precondition with IMF, the time gained from the reduced iteration count, by updating the degrees, was offset by an increase of the computation time. For more difficult problems, updating the degrees often improved the overall execution time.

According to Theorem 2.4, we should distribute the values of the generated element $g_i \in \mathcal{G}_k$ over the level-2 neighbors $\mathcal{N}_2(d_i)$ of the pivotal element $d_i \in \mathcal{D}_k$. However, only in a few exceptional situations is the entire level-2 neighborhood necessary. In practice, the neighborhood of d_i most often suffices to distribute all non-fill-in values of g_i. If Eq. (2.23), which is equivalent to Eq. (2.16) in Theorem 2.4, were modified so that $\mathcal{R}(e) = \mathcal{D}_k \cap \mathcal{N}(e)$, then the main results of Section 3 will still hold. In particular, Eq. (3.1) and Eq. (3.2) and their implications on the multiplicity of eigenvalue unity are still valid. The spectrum of $M_{\kappa+1}^{-1} A_{\kappa+1}$ will be perturbed, however. Distributing the values of g_i over only the neighbors of d_i has several advantages. Foremost, it increases the degree of parallelism, because no element can be updated twice. Synchronization between updates from multiple fronts is no longer a concern. Furthermore, the modification decreases the computational cost; the values of the modified frontal matrix have to be distributed over less elements. In our numerical experiments we found that this simplification did not greatly affect the rate of convergence, while decreasing the cost of the factorization measurably. The convergence was hardly affected in our experiments, because only in a few exceptional situations the neighbors of d_i do not suffice to distribute the non-fill-in values over.

The permutations P_k are applied to the preconditioning matrix at every level k in the factorization phase. That is, we store $M_k^P := P_k M_k P_k^T$. Meanwhile, the permutations are accumulated in one permutation matrix P. We can then apply the action of M^{-1} to $\mathbf{x}$ by permuting $\mathbf{x}$ with P^T, solving for M_k^P at every level, and permuting the result with P. In this manner, only one permutation is required per preconditioner application. This is an advantage because permutations cannot be applied (practically) with high throughput. That is due to the indirect memory accesses and potentially large jumps in the memory, which significantly hampers cache performance. Clearly, the number of permutation operations could be reduced further, by permuting A and the right-hand side $\mathbf{b}$ with P as well.

In a Krylov subspace method, the coefficient matrix A has to be applied in every iteration. We employed the matrix-vector product implementation of GLAS, which is a straightforward CSR-based implementation. The coefficient matrix was reordered with Reverse Cuthill-McKee [15], to improve the profile of the matrix.

We implemented our preconditioner in C++, using the Generic Linear Algebra Software (GLAS) [24] and Boost libraries. GLAS provides basic data structures such as compressed storage for sparse matrices and dense matrices. It also provides the reference implementation of BiCgSTAB [48], presented in [9]. We used the netlib LAPACK and BLAS libraries to compute the inverse (with partial pivoting) and

matrix product, respectively. These were compiled using the `gcc` 4.4.3 compiler, with the `-O3` optimization flag. Our C++ code was compiled using `gcc` 4.4.3 with `-O3`.

We will compare IMF with some preconditioners from the ITSOL package[5]. This package was compiled using `gcc` 4.4.3 with optimizations (`-O3`) enabled. We experimented with the SPAI preconditioner, from the SPAI library[6]. It was compiled using the `gcc` 4.4.3 compiler with `-O3` enabled.

6. Numerical experiments. We present the results of our numerical experiments in this section. The experiments were conducted on a computer system that consisted of an Intel Core i5 M560 processing unit running at 2.67GHz (2×256KB L2 cache and 3MB L3 cache) and 4GB main memory (clocked at 1333MHz). We will compare the performance of ARMS, ILU, ILUT, IMF and SPAI where BiCgSTAB [48] is used as iterative method. In the next section, we present two model problems. Thereafter, the throughput of the solution phase is assessed. In Section 6.3, we investigate the convergence of aforementioned preconditioners on the model problems. Finally, some preliminary experiments are presented wherein IMF is applied to solve general sparse linear systems.

First, we briefly describe the preconditioners with which we compare. ILUT(p, κ) is a dual-threshold incomplete LU-factorization preconditioner [42]. It consists of row-wise Gaussian elimination. A numerical dropping rule is used to discard all values that are smaller than κ times the 1-norm of the current row. In each row, at most the p largest values are retained. The preconditioner can be applied by a forward and backward substitution. Unless stated otherwise, we took $p = 300$.

ARMS(b, p, κ, w, q) is a multilevel block-ILU preconditioner [44], similar to IMF. Its factorization is given by

$$P_k A_k P_k^T = \begin{bmatrix} D_k & U_k \\ L_k & S_k \end{bmatrix} \approx \begin{bmatrix} V_k & 0 \\ L_k W_k^{-1} & I \end{bmatrix} \times \begin{bmatrix} W_k & V_k^{-1} U_k \\ 0 & A_{k+1} \end{bmatrix}, k \geq 0, \tag{6.1}$$

where $D_k \approx V_k W_k$ (an incomplete LU-decomposition) and $A_{k+1} \approx S_k - G_k H_k$ with $G_k \approx L_k W_k^{-1}$ and $H_k \approx V_k^{-1} U_k$. D_k is a sparse block diagonal matrix, which is factored using ILUT(p, κ). A heuristic algorithm seeks blocks of size at least b, to form D_k, while trying to satisfy a relative diagonal dominance criterion involving the parameter $0 \leq w \leq 1$. Higher values of w result in blocks that are more diagonally dominant. To compute the matrices G_k, H_k and A_{k+1}, the ILUT(p, κ) dual-threshold dropping rule is again used. The factorization is applied recursively to A_{k+1}. The recursive application is halted if the number of levels equals the maximum number of levels q. The final system A_q is factored using ILUT(p, κ), resulting in the factors V_q and W_q. The preconditioner can be applied by a recursive block forward and backward substitution. ARMS-ND is a variant of ARMS that uses a nested dissection strategy to determine the partitioning of the unknowns. This approach often results in less levels in the multilevel factorization. Unless stated otherwise, we set $b = 500$, $p = 1000$, $q = 20$ and $w = 0.3$ in our experiments.

ILU(κ) is a level of fill-in based incomplete LU-factorization preconditioner [43]. Fill-in values whose level is greater than κ are discarded. The preconditioner is applied by a forward and backward substitution. A multifrontal algorithm to compute this factorization was proposed in [12].

[5]The source code may be obtained from Y. Saad's software page:
`http://www-users.cs.umn.edu/~saad/software/`

[6]The source code may be obtained from M.J. Grote's software page:
`http://www.computational.unibas.ch/software/spai/`

SPAI(κ, p, b) is the block sparse approximate inverse preconditioner [8, 25]. The SPAI algorithm seeks a minimizer to $\|I - MA\|_F$. The sparse matrix M serves as the preconditioner, which can be applied by a simple matrix-vector multiplication. The parameter κ controls the quality of the approximation, p is the number of improvement steps in the optimization and b is the block size. For more details on these parameters, see [8, 25]. We set $p = 5$ and $b = 12$.

6.1. Model problems. We have selected two model problems, `NS5` and `NS3` [20, Appendix A] from the IFISS toolbox [21] to demonstrate the performance of the preconditioners. `NS5` models the flow of an incompressible Newtonian fluid, driven by a pressure difference, around a square obstacle in a rectangular channel. The domain Ω is $[0, 8] \times [-1, 1]$, with the square obstruction at $[7/4, 9/4] \times [-1/4, 1/4]$ removed. On this domain, the Navier-Stokes equation

$$\begin{aligned} -\nu\nabla^2\mathbf{u} + \mathbf{u}\cdot\nabla\mathbf{u} + \nabla p &= \mathbf{f}, \\ \nabla\cdot\mathbf{u} &= 0, \end{aligned}$$

is specified. The parameter ν is the kinematic viscosity. The vector-valued function $\mathbf{u}$ represents the flow velocity and the scalar function p is the pressure. The no-flow Dirichlet boundary condition, $\mathbf{u} = 0$, is specified on the boundaries of the obstruction and the top and bottom walls of the channel. At the inflow boundary, $x = 0$, the flow velocity $\mathbf{u} = (1 - y^2, 0)$ is specified. At the outflow boundary, $x = 8$, a Neumann condition that sets the mean outflow pressure to zero is applied. This models a flow through a rectangular channel, with a square obstruction, that is driven by a pressure difference between the two ends of the channel. The flow in contact with the top and bottom boundaries and the obstruction is stationary, due to friction, while the center of the flow moves at speed unity.

The second problem, `NS3`, models the watertight driven-cavity flow. The domain is $\Omega = (-1, 1)^2$. On this domain, the Navier-Stokes equation is specified. The no-flow Directlet boundary condition is applied to the left, right and bottom boundaries. At the top boundary, $y = 1$, $-1 < x < 1$, the fluid flows at speed unity, $\mathbf{u} = (1, 0)$.

The PDE in both problems is discretized using pressure stabilized $\mathbf{Q}_1$–$\mathbf{Q}_1$ elements. This results in element matrices of size 12×12. For `NS5`, there were 16128 elements and `NS3` was discretized with 40000 elements. To solve the steady-state flow problem, a nonlinear system of equations is solved iteratively. In every step, a linear system of equations is solved. We exported[7] the element structure of this system, which was then used as input to our preconditioner. The system that we exported for `NS5`, is the system one obtains after 6 Picard iteration steps and 5 Newton steps. It has 49440 unknowns and 1243530 non-zero values (regardless of the parameter ν). For the `NS3` problem, we exported the system that is obtained after 2 Picard iteration steps followed by 4 Newton steps. The system has 121203 unknowns and 3153593 non-zeros. Both systems are real, unsymmetric and not diagonally dominant. More details can be found in [20, 22].

An important quantity in incompressible fluid flow dynamics is the Reynolds number. In our case, the Reynolds number of the `NS5` flow was $\mathrm{Re} = 3.2\nu^{-1}$ whereas `NS3` had $\mathrm{Re} = 2\nu^{-1}$. It is well known that the linear systems become more difficult to solve for higher values of the Reynolds number.

[7] IFISS does not provide routines to directly export the element structure, so we have written our own routine.

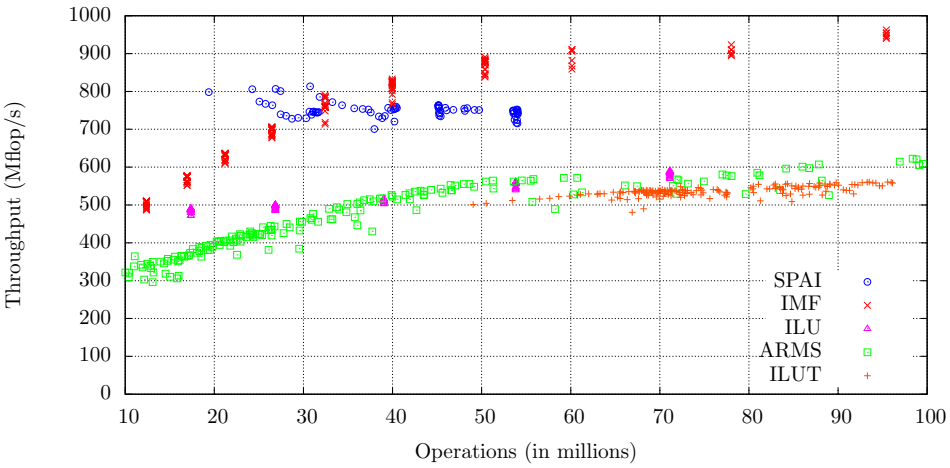
6.2. Computational efficiency. First, the composition of the operation count for the application of the preconditioners is determined. Let "SpTriSol" denote a sparse triangular solve, "SpMxV" a sparse matrix-vector product and "BMxV" a block diagonal matrix-vector product. Let the ARMS preconditioner be as in Eq. (6.1), the ILU preconditioner $M_{\text{ILU}} = L_{\text{ILU}}U_{\text{ILU}}$, the ILUT preconditioner $M_{ILUT} = L_{\text{ILUT}}U_{\text{ILUT}}$, the IMF preconditioner as in Eq. (2.2) and the SPAI preconditioner M_{SPAI}. The floating-point operation counts (flops) for one application of these preconditioners are given, respectively, by

$$\begin{aligned}
\text{ops}_{\text{ARMS}} &= 2\sum_{i=0}^{l}(\underbrace{2\,\text{nnz}(V_i)}_{\text{SpTriSol}} + \underbrace{2\,\text{nnz}(W_i)}_{\text{SpTriSol}}) + \sum_{i=0}^{l-1}(\underbrace{2\,\text{nnz}(L_i)}_{\text{SpMxV}} + \underbrace{2\,\text{nnz}(U_i)}_{\text{SpMxV}}),\\
\text{ops}_{\text{ILU}} &= \underbrace{2\,\text{nnz}(L_{\text{ILU}})}_{\text{SpTriSol}} + \underbrace{2\,\text{nnz}(U_{\text{ILU}})}_{\text{SpTriSol}},\\
\text{ops}_{\text{ILUT}} &= \underbrace{2\,\text{nnz}(L_{\text{ILUT}})}_{\text{SpTriSol}} + \underbrace{2\,\text{nnz}(U_{\text{ILUT}})}_{\text{SpTriSol}},\\
\text{ops}_{\text{IMF}} &= 2\sum_{i=0}^{m}\underbrace{2\,\text{nnz}(D_i)}_{\text{BMxV}} + \sum_{i=0}^{m-1}\underbrace{2\,\text{nnz}(L_i)}_{\text{SpMxV}} + \underbrace{2\,\text{nnz}(U_i)}_{\text{SpMxV}}, \text{ and}\\
\text{ops}_{\text{SPAI}} &= \underbrace{2\,\text{nnz}(M_{\text{SPAI}})}_{\text{SpMxV}}.
\end{aligned}$$

Herein, l and m are the number of levels in ARMS and IMF, respectively. We notice that ARMS and IMF require more operations than the other preconditioners for the same number of non-zeros in the factorization. That is due to the double application of the inverse of the block diagonal matrix, i.e. in Eq. (2.4) and Eq. (2.6). The operation count of ILU and ILUT consists of two SpTriSols, which are not trivially parallelized [27]. ARMS improves upon this by replacing these sequential solves by two SpMxVs and four block-SpTriSols, per level. Both of which allow for a greater degree of parallelism. IMF extends the parallelism by replacing the SpTriSols by BMxVs. Since every matrix on the diagonal is dense, a dense matrix-vector product can be employed. If the preconditioner is to be applied in a block Krylov-subspace method, then matrix multiplication via BLAS3 is even possible. Both ARMS and IMF require a synchronization, however, before applying the preconditioner recursively at the next level. The SPAI preconditioner avoids this, as it requires just one SpMxV. This increases the degree of parallelism even further.

We now present some results of the throughput of the five preconditioners on the `NS5` model problem. We measured the throughput for one BiCgSTAB iteration. The operation count includes the two matrix-vector products with A, the 6 dot products (counting norms as well) and the two preconditioning operations. The cost of the other operations in BiCgSTAB was neglected. In Figure 6.1, the throughput of the preconditioners in function of the operation count is plotted. The data points come from solving the model problem with different values for the Reynolds number ($\text{Re} = 1200, 1600, \ldots, 10000$) and different settings for the preconditioners; only κ, which influences the amount of fill-in, varied. Our data show that, for a comparable number of operations, ARMS, ILU and ILUT are very close. IMF and SPAI consistently outperform the other preconditioners, all of which require sparse triangular solves. Notice that IMF is competitive with SPAI, especially when there are more non-zeros,

Figure 6.1: The throughput of one BiCgSTAB iteration in function of the number of operations of one application of the preconditioner. The results were gathered from multiple solves of the model problem with different Reynolds numbers (Re = $1200, 1600, \ldots, 10000$).



and thus operations, in the preconditioner. IMF's throughput is between 40% and 60% better than the similar ARMS preconditioner. Its advantage stems from the block matrix-vector products. The size of the matrices on the block diagonal increases with the number of fill-in levels and thus the number of operations. Therefore, the dense matrix-vector products can attain a higher throughput. Furthermore, the ratio of the non-zeros in the block diagonal matrix to the total non-zeros in the preconditioner increases. On this problem, that ratio increased from 23.6% for IMF(1) up to 54.5% for IMF(13). This shifts the dominant operation from the SpMxVs to the BMxV. The throughput of the latter being higher than the former.

6.3. Convergence. We investigate the convergence of the preconditioners on the model problems, for different values of the Reynolds number.

We solved the model problems using BiCgSTAB. The stopping criterion consists of a relative tolerance of 10^{-8} and a maximum iteration count of 2000. As (artificial) right-hand side we took A times the vector of all ones. ILU used RCM [15] as ordering, the other preconditioners specified their own ordering.

Speedup. In Table 6.1, we display the minimum, (geometric) average, maximum and total speedup of the IMF preconditioner over the other preconditioners, using the parameter settings that resulted in the smallest total solution time (includes factorization and convergence). The total speedup is determined with respect to the total time to solve the model problem for all the different Reynolds numbers, whereas the other speedups measure the speedup of one particular linear system solve. The statistical data in Table 6.1 include results from experiments with different parameter settings than those presented in Table 6.3 and Table 6.2. The "failed" column indicates the number of linear systems that could not be solved by a given preconditioner, with any of the parameter settings that we tested, within the maximum number of iterations and within 20 minutes.

The data demonstrate that IMF was a very fast preconditioner. IMF solved the `NS5` model problem about one order-of-magnitude faster than the other precondition-

Table 6.1: Speedup of IMF with respect to the other preconditioners.

Problem	Preconditioner	Failed	Speedup			
			Min.	Avg.	Max.	Total
NS3	IMF	0/12	1.00	1.00	1.00	1.00
	ARMS	0/12	0.73	2.10	3.62	2.29
	ILU(κ)	4/12	2.58	3.36	4.71	3.38
	ILUT	10/12	2.01	2.90	4.19	3.15
	SPAI	10/12	8.41	13.62	22.05	8.41
NS5	IMF	0/19	1.00	1.00	1.00	1.00
	ARMS	0/19	1.01	9.06	27.09	13.99
	ARMS-ND	13/19	2.65	8.60	68.89	23.86
	ILU(κ)	1/19	5.11	8.03	17.28	8.18
	ILUT	5/19	3.19	19.85	48.25	26.97
	SPAI	13/19	9.40	25.40	74.30	34.79

ers. Both ARMS and IMF were robust, so that all linear systems could be solved, with suitable parameter settings. This was not the case for ILUT and SPAI which experienced considerable troubles to solve the NS3 model problem.

Model problem NS3. Results of our experiments with the NS3 model problem are presented in Table 6.2. We have only retained the most competitive preconditioners.

IMF offers a fast solution time over the whole range of Reynolds numbers. However, this speedup comes at the cost of increased memory consumption. ARMS was the only preconditioner that managed to solve the system with very low fill factors, even at higher Reynolds numbers. Notice that the fill factor of ARMS varies rather smoothly with the Reynolds number. The maximum fill factor that ARMS required to solve a system was 2.89, whereas IMF needed 5.58.

The factorization time of IMF was competitive with ARMS and ILUT, especially at lower fill factors, even though IMF requires more operations to compute its factorization. This is a consequence of the high-throughput BLAS and LAPACK routines which operate on the dense frontal matrices.

Notice how the quality of the IMF factorization increases with its level of fill-in parameter. Except for higher Reynolds numbers, increasing the level of fill-in reliably reduces the number of iterations required for convergence. Increasing κ also resulted in an increase of the fill factor and the factorization time.

Notice that the factorizations of IMF(κ) and ILU(κ) are different, even though the symbolic dropping rule of the former resembles the level of fill-in rule of the latter. In particular, the fill factor of IMF is lower than ILU's for a given level of fill-in. This enables a more fine-grained control over the amount of fill-in, in IMF.

Model problem NS5. Results of the experiments with NS5 are shown in Table 6.3. We let T_f denote the time to compute the preconditioner, T_s the time to solve the system (excluding T_f), "its" the number of iterations to reach convergence and "fill" the ratio of the number of non-zero values in the preconditioner and the system. A "$*$" indicates that the method did not converge within the maximum number of iterations. In every row, the smallest value is highlighted in boldface. Only the preconditioners that converged are taken into consideration, to determine this value.

Here, the results indicate that IMF is a powerful preconditioner over the entire range of Reynolds numbers. Especially at higher values of the Reynolds number, IMF proved to be very efficient in reducing the time to solve the system. At low Reynolds

Table 6.2: The convergence of the preconditioners on model problem `NS3`.

Re		IMF(κ)				ARMS(κ)				ILU(κ)	
	κ	3	4	5	6	0.030	0.010	0.005	0.003	3	4
400	T_f	3.90	5.43	7.95	13.77	**1.44**	2.37	3.95	6.37	10.29	19.80
	T_s	18.62	15.07	12.57	**11.78**	41.59	28.38	23.77	24.73	*	*
	its	178	132	95	**75**	368	175	108	88	*	*
	fill	3.41	4.38	5.58	7.07	**1.05**	1.92	3.01	4.43	6.78	9.83
1200	T_f	3.92	5.30	7.87	13.44	1.57	**2.49**	3.81	5.63	10.25	19.82
	T_s	36.40	*	14.72	15.01	*	28.23	16.74	**10.94**	*	*
	its	348	*	112	98	*	168	78	**42**	*	*
	fill	3.41	4.38	5.58	7.07	0.84	**1.59**	2.40	3.31	6.78	9.83
2000	T_f	3.89	5.32	7.88	13.33	**1.60**	4.36	9.12	17.13	10.22	19.79
	T_s	*	*	17.62	**16.84**	55.31	146.59	153.82	175.19	221.13	*
	its	*	*	135	**110**	488	598	395	330	1145	*
	fill	3.41	4.38	5.58	7.07	**1.19**	3.40	6.85	10.79	6.78	9.83
2800	T_f	3.96	5.39	7.99	13.63	1.52	**3.45**	7.65	15.05	10.56	20.25
	T_s	*	19.97	21.23	**19.49**	*	88.38	122.67	153.15	161.39	*
	its	*	175	160	**125**	*	410	355	310	825	*
	fill	3.41	4.38	5.58	7.07	1.11	**2.89**	5.74	10.10	6.78	9.83
3600	T_f	4.03	5.49	8.24	14.11	**1.49**	3.50	7.32	14.05	10.73	20.43
	T_s	*	41.92	33.29	**15.96**	68.49	126.73	147.60	152.66	130.84	*
	its	*	358	245	**98**	648	582	432	318	655	*
	fill	3.41	4.38	5.58	7.07	**1.06**	2.84	5.73	9.60	6.78	9.83
4000	T_f	3.95	5.42	8.05	13.89	**1.61**	4.00	7.89	13.62	11.66	20.71
	T_s	56.79	37.16	20.64	**15.53**	79.85	116.09	172.88	151.74	112.27	*
	its	535	312	155	**98**	692	490	490	305	538	*
	fill	3.41	4.38	5.58	7.07	**1.08**	2.87	5.54	9.05	6.78	9.83
4400	T_f	3.94	5.37	8.02	13.66	**1.55**	3.64	6.81	12.70	10.52	19.98
	T_s	192.61	*	25.42	**16.88**	104.86	114.55	184.40	138.79	*	*
	its	1800	*	190	**108**	982	530	582	308	*	*
	fill	3.41	4.38	5.58	7.07	**1.09**	2.87	5.21	8.67	6.78	9.83
4800	T_f	3.93	5.30	7.93	13.46	**1.56**	3.55	5.90	11.80	10.23	19.83
	T_s	*	*	23.80	**18.04**	74.84	*	172.05	141.14	*	*
	its	*	*	182	**118**	712	*	578	328	*	*
	fill	3.41	4.38	5.58	7.07	**1.09**	2.82	4.62	8.00	6.78	9.83
5600	T_f	3.89	5.34	7.88	13.42	**1.70**	3.23	6.79	11.70	10.26	19.90
	T_s	*	45.61	23.21	**22.18**	62.62	153.58	120.88	120.56	96.21	*
	its	*	408	178	**145**	575	788	402	282	500	*
	fill	3.41	4.38	5.58	7.07	**1.12**	2.50	4.92	8.46	6.78	9.83
6400	T_f	3.94	5.32	7.91	13.47	**1.76**	2.85	8.81	14.20	10.23	19.86
	T_s	79.17	37.93	**22.62**	37.80	57.52	71.44	123.05	175.31	99.02	*
	its	765	338	**172**	245	512	402	398	405	512	*
	fill	3.41	4.38	5.58	7.07	**1.14**	2.17	5.68	8.99	6.78	9.83
7200	T_f	3.92	5.32	7.94	13.30	2.27	**3.03**	9.38	15.92	10.17	19.74
	T_s	*	45.80	82.26	**28.34**	*	72.32	127.13	184.91	97.46	*
	its	*	408	632	**185**	*	402	410	438	508	*
	fill	3.41	4.38	5.58	7.07	1.18	**2.20**	5.59	8.74	6.78	9.83
8000	T_f	4.07	5.47	8.10	13.72	**2.59**	3.10	10.78	14.99	10.41	20.11
	T_s	*	*	**27.20**	*	148.77	78.34	122.86	198.47	87.88	*
	its	*	*	**202**	*	1342	445	390	490	445	*
	fill	3.41	4.38	5.58	7.07	**1.29**	2.15	5.90	8.35	6.78	9.83

Table 6.3: The convergence of the preconditioners on model problem `NS5`.

Re		IMF(κ)		ARMS(κ)		ILUT(κ)	ILU(κ)		SPAI(κ)	
	κ	3	4	0.005	0.003	0.95	4	5	0.2	0.1
400	T_f	**1.32**	1.80	1.45	2.00	8.17	7.54	13.23	226.24	265.20
	T_s	2.17	1.76	2.08	**1.67**	2.98	*	*	15.92	15.62
	its	58	42	25	**18**	30	*	*	240	218
	fill	3.42	4.31	**2.45**	3.02	8.74	9.70	13.18	8.88	9.72
1200	T_f	**1.34**	1.79	4.32	7.52	21.47	7.62	13.19	221.70	272.09
	T_s	2.55	**2.02**	12.81	15.81	9.02	29.41	*	18.75	21.22
	its	68	**48**	82	72	78	305	*	292	295
	fill	**3.42**	4.31	7.81	13.52	11.16	9.70	13.18	8.61	9.74
2000	T_f	1.33	**1.80**	26.73	58.52	53.52	7.65	13.23	220.87	275.95
	T_s	*	**21.27**	95.97	113.27	*	*	*	*	*
	its	*	498	570	**495**	*	*	*	*	*
	fill	3.42	**4.31**	13.79	21.44	12.41	9.70	13.18	8.55	9.75
2800	T_f	**1.35**	1.86	32.13	56.27	37.18	7.63	13.24	208.41	280.04
	T_s	7.63	**3.12**	25.31	25.39	45.05	*	*	82.39	50.33
	its	198	**72**	155	122	355	*	*	1358	695
	fill	**3.42**	4.31	13.85	19.90	12.06	9.70	13.18	8.11	9.74
3600	T_f	**1.32**	1.81	39.65	67.26	38.10	7.87	13.83	222.31	292.54
	T_s	4.26	**3.22**	21.74	38.48	89.86	*	*	*	*
	its	112	**78**	140	195	722	*	*	*	*
	fill	**3.42**	4.31	15.05	20.86	12.25	9.70	13.18	8.01	9.74
4000	T_f	**1.33**	1.80	45.08	70.93	39.62	7.80	13.54	210.81	288.08
	T_s	4.62	**3.32**	20.38	33.94	81.48	*	26.31	*	*
	its	122	**78**	130	175	650	*	210	*	*
	fill	**3.42**	4.31	16.02	20.86	12.38	9.70	13.18	7.99	9.73
4800	T_f	**1.34**	1.81	60.42	84.37	43.30	7.99	13.74	208.00	281.93
	T_s	5.56	**3.63**	*	32.41	*	*	25.46	*	*
	its	148	**88**	*	165	*	*	208	*	*
	fill	**3.42**	4.31	18.27	22.32	12.48	9.70	13.18	7.97	9.70
5600	T_f	**1.35**	1.84	66.15	94.74	47.75	19.40	34.41	201.23	277.26
	T_s	6.74	**4.46**	225.41	43.09	*	*	45.32	*	*
	its	175	**102**	1375	220	*	*	255	*	*
	fill	**3.42**	4.31	18.69	23.26	12.65	9.70	13.18	7.96	9.66
6400	T_f	**1.34**	1.92	65.23	101.20	50.56	7.55	13.37	200.84	276.09
	T_s	9.77	**5.14**	*	55.05	*	*	29.93	*	*
	its	242	**110**	*	272	*	*	248	*	*
	fill	**3.42**	4.31	18.08	23.33	12.69	9.70	13.18	7.96	9.65
7200	T_f	**1.32**	1.81	57.66	99.18	52.20	7.53	13.38	200.82	275.65
	T_s	12.08	**5.47**	79.11	*	*	*	45.98	*	*
	its	320	**128**	500	*	*	*	380	*	*
	fill	**3.42**	4.31	18.24	23.84	12.77	9.70	13.18	7.96	9.65
7600	T_f	**1.32**	1.81	54.91	118.58	52.96	7.54	13.22	200.28	275.36
	T_s	14.13	**5.41**	*	*	180.35	29.35	*	*	*
	its	372	**130**	*	*	1388	305	*	*	*
	fill	**3.42**	4.31	17.96	25.31	12.78	9.70	13.18	7.96	9.64
8000	T_f	**1.36**	1.84	71.83	117.27	56.63	7.55	13.21	200.40	275.38
	T_s	16.91	**5.81**	99.02	*	*	*	60.92	*	*
	its	440	**138**	582	*	*	*	502	*	*
	fill	**3.42**	4.31	20.10	24.75	12.97	9.70	13.18	7.95	9.64

numbers, the ARMS preconditioner was faster, while using less memory. ARMS was also able to converge with fill factors less than 1, for low Reynolds numbers, a feat not accomplished by IMF. Both ILUT and SPAI showed difficulties in solving the model problem at Reynolds numbers higher than 4000. Note that Table 6.3 does not include the results for the lower Reynolds numbers.

IMF spends much less time on the factorization than the other preconditioners. This is partly the consequence of its low fill factor. The other preconditioners required such a high fill factor, because otherwise they would not converge. When the fill factors were approximately equal, the factorization times of ARMS, ILUT and IMF were comparable, with ILU being somewhat faster than the others. For comparable fill factors, the SPAI preconditioner always resulted in the slowest factorization. The data show that IMF achieves roughly the same factorization time as ARMS and ILUT, while requiring a larger number of operations to compute the factorization. That is by virtue of the high-throughput BLAS3 and LAPACK routines, which are employed in the computationally most expensive part of the factorization.

The quality of the IMF factorization appears to increase with its level of fill-in parameter κ, as we could expect from theoretical arguments. Except for higher Reynolds numbers, increasing κ reliably reduces the number of iterations required for convergence. Increasing κ also increases the fill factor and the factorization time.

Notice that the factorizations of IMF(κ) and ILU(κ) are different, even though the symbolic dropping rule of the former resembles the level of fill-in rule of the latter. In particular, the fill factor of IMF is lower than ILU's for a given level of fill-in. This enables a more fine-grained control over the amount of fill-in, in IMF.

One advantage of IMF over the other preconditioners is its consistent performance at a fixed memory consumption. ARMS and ILUT both use numerical dropping strategies. The table clearly shows an increase of the fill factor with the Reynolds number. For instance, the fill factor of ARMS(0.005) increases from a modest 3.02 at Re = 400 up to 24.75 for Re = 8000. This dependence makes it more difficult to select an appropriate ARMS or ILUT preconditioner. The symbolic approaches, ILU, IMF and SPAI, are able to keep the memory consumption fixed, but suffer from a deterioration of the performance as the Reynolds number is increased. IMF demonstrates, however, that a symbolic preconditioner can be effective, with a reasonable memory consumption, on difficult problems.

In the previous subsection, it was demonstrated that the SPAI preconditioner had a very competive throughput on this model problem. However, the data in Table 6.3 indicate that it was unable to solve the harder linear systems within the maximum number of iterations. In fact, the preconditioner often quickly reduced the norm of the residual a few orders, after which the convergence stagnated.

Not shown in Table 6.3 is the performance of the ARMS-ND preconditioner. We found that ARMS-ND did not perform significantly better than ARMS, on this problem. We also found that the fill factor of ARMS-ND was more difficult to control than ARMS'. The fill factor of ILUT was also very hard to control on this problem. Even with $\tau = 0.999$, the fill factor was higher than 7, for any Reynolds number. On a diagonally dominant matrix, this setting would keep at most 2 values in every row.

6.4. Some experiments with general sparse matrices. In this subsection, we apply the IMF preconditioner to some general sparse matrices which were obtained from the University of Florida sparse matrix collection [16]. The test matrices and some of their properties are presented in Table 6.4.

To apply IMF to a general sparse matrix, we need an element structure of that

Table 6.4: Test matrices from the University of Florida sparse matrix collection.

Name	n	**Nnz**	**Sym.** (%)	**Application area**
af23560	23560	460598	0.0	Computational fluid dynamics problem
bcsstk39	46772	2060662	100.0	Structural problem
ct20stif	52329	2600295	100.0	Structural problem
d_pretok	182730	1641672	100.0	2D/3D problem
matrix_9	103430	1205518	17.5	Semiconductor device problem
rim	22560	1014951	0.0	Computational fluid dynamics problem
rma10	46835	2329092	23.6	Computational fluid dynamics problem
ship_003	121728	3777036	100.0	Structural problem
shipsec1	140874	3568176	100.0	Structural problem
shipsec5	179860	4598604	100.0	Structural problem
shipsec8	114919	3303553	100.0	Structural problem
stokes128	49666	558594	100.0	Computational fluid dynamics problem
turon_m	189924	1690876	100.0	2D/3D problem

matrix. In this paper, we consider a very simple scheme, not unlike the element recovery algorithm in [19, Algorithm 2.1]. We generate an artificial element structure from a general sparse matrix as follows. For every row i, we determine the indices at which the non-zero values are located. These indices, along with i, form the index set $\mathcal{I}(e)$ of the new artificial element e. Its values are those in the general sparse matrix. Next, the values in the sparse matrix at the positions $\mathcal{I}(e) \times \mathcal{I}(e)$ are set to zero. The element is added to the artificial element structure. The algorithm proceeds with the next row. In the end, every element is again inspected. Every index for which the corresponding row and column is zero, is removed from the artificial element. Elements whose index set is empty, are removed from the element structure. For sparse matrices derived from a finite element application, the artificial elements usually span multiple finite elements. One difficulty with this simple algorithm, however, is that the elements so generated are often sparser than elements derived from an actual finite element discretization.

An alternative approach consists of representing each value $a_{i,j}$ in the sparse matrix by an element with indices $\{i, j\}$. In this manner, much less structural zeros are stored. Unfortunately, this element structure results in an impractical amount of elements. It leads to a slow factorization phase, while in the solution phase only limited throughput could be attained. We do not pursue this approach.

The artificial element structures of most of the test matrices in Table 6.4 exhibited the properties of a finite element discretization. That includes small, very dense element matrices and a low coupling between the element matrices. It is common for IMF to be outperformed by other preconditioners, if the artificial element structure does not exhibit these properties.

The test problems were solved using BiCgSTAB, as before. Additionally, the method was stopped if the total execution time exceeded 15 minutes. The results are shown in Table 6.5 and Table 6.6. A serious failure during the solution phase, e.g. if the solution vector contains NaN or infinity, is denoted by "†". Breakdown during the factorization is denoted by "‡".

The data in Table 6.5 and Table 6.6 show that IMF was more robust than ARMS, ARMS-ND, ILUT and ILU. Our experiments indicated that ARMS-ND often was more robust than ARMS and ILUT. Unfortunately, this robustness comes at the price of increased factorization time. The IMF factorization did not result in breakdowns

Table 6.5: Comparison of IMF, ARMS, ARMS-ND, ILUT and ILU on the test matrices from Table 6.4.

		IMF(κ)		ARMS(κ)		ARMS-ND(κ)		ILUT(κ)		ILU(κ)	
bcsstk39	τ	5	6	0.05	0.005	0.05	0.01	0.2	0.01	1	2
	T_f	6.44	12.62	30.37	48.54	43.10	92.26	2.46	5.82	**0.29**	0.61
	T_s	$*$	31.86	†	†	$*$	$*$	28.25	3.38	0.58	**0.37**
	its	$*$	355	†	†	$*$	$*$	368	27	38	**15**
	fill	5.64	7.22	7.90	8.99	8.62	12.37	3.25	7.17	**2.38**	4.42
ct20stif	τ	2	3	0.005	0.0005	0.05	0.01	0.5	0.01	1	3
	T_f	3.97	10.01	7.70	16.90	12.69	237.49	31.73	96.47	**1.40**	4.47
	T_s	37.59	33.65	$*$	$*$	$*$	$*$	$*$	$*$	89.72	**9.46**
	its	605	415	$*$	$*$	$*$	$*$	$*$	$*$	1390	**92**
	fill	3.02	4.48	4.04	10.71	3.46	11.32	4.42	6.62	**1.72**	3.61
ship_003	τ	1	3	0.001	0.0001	0.005	0.001	0.3	0.005	2	3
	T_f	**7.54**	52.76	6.35	30.86	10.95	151.58	18.67	70.19	8.68	19.99
	T_s	113.31	41.31	$*$	$*$	$*$	$*$	$*$	44.59	37.23	**25.30**
	its	848	149	$*$	$*$	$*$	$*$	$*$	156	170	**88**
	fill	**4.30**	11.98	4.31	12.41	4.51	15.18	4.43	8.97	4.99	7.99
shipsec1	κ	1	2	0.001	0.0001	0.03	0.01	0.25	0.05	2	3
	T_f	**3.41**	5.47	42.18	36.16	21.37	85.88	18.64	53.24	4.39	8.18
	T_s	29.06	**21.77**	†	$*$	$*$	†	$*$	$*$	32.68	33.30
	its	273	172	†	$*$	$*$	†	$*$	$*$	180	**142**
	fill	**3.10**	4.50	7.55	14.50	3.04	6.56	4.76	8.60	3.68	5.42
shipsec5	κ	0	1	0.005	0.0005	0.03	0.02	0.75	0.05	1	2
	T_f	**2.82**	4.80	4.15	139.85	13.03	105.94	41.67	70.12	3.33	8.18
	T_s	55.69	35.56	$*$	$*$	$*$	$*$	†	$*$	112.71	**34.75**
	its	494	253	$*$	$*$	$*$	$*$	†	$*$	633	**125**
	fill	**1.70**	3.23	1.83	10.27	2.52	5.47	13.00	13.14	2.59	4.93
shipsec8	τ	2	3	0.001	0.0001	0.02	0.01	0.5	0.05	1	2
	T_f	8.98	22.15	24.75	34.02	17.10	128.00	32.74	81.34	**2.50**	6.18
	T_s	82.43	46.99	†	$*$	$*$	$*$	†	$*$	23.29	**14.18**
	its	591	248	†	$*$	$*$	$*$	†	$*$	188	**75**
	fill	5.68	8.65	7.12	14.75	3.53	8.56	9.81	10.85	**2.60**	4.84

on any of the test matrices, for any parameter setting that we tested. For none of the parameter settings, IMF resulted in a serious arithmetic failure during the solution phase. The robustness of IMF is a consequence of the partial pivoting employed by the LAPACK `dgetrf` routine. However, IMF is not completely breakdown free. The robustness of IMF could be improved further by implementing a pivoting strategy across levels. That is, if one of the frontal matrices results in a breakdown, we can simply remove the pivotal element that causes this breakdown. The element might get updated by some of the other eliminations, so that the instability could disappear at the next level. Furthermore, full pivoting could be used to compute the inverse of the matrices on the block diagonal. Another technique to increase the robustness consists of grouping some elements into a super-element and computing the Schur complement of the frontal matrix associated with this super-element.

In eight out of thirteen cases, IMF was able to converge, within 2000 iterations, with less memory than ARMS, ARMS-ND, ILU, ILUT and SPAI, for the parameter

settings that we tested. IMF's capability to converge with less memory is a result of a more accurate approximation to the inverse of the linear system, at a given memory consumption.

Concerning the factorization time, we observed that IMF was faster than ARMS, ARMS-ND and ILUT, at small fill factors (≤ 5). For higher fill factors it was slower than the aforementioned preconditioners, but it remained competitive. The factorization time of ARMS-ND was significantly larger than ARMS's, especially for higher fill factors.

The structural problems presented in Table 6.5 proved to be very difficult to solve using the threshold-based preconditioners, ARMS, ARMS-ND and ILUT. They failed to converge for reasonable fill factors. However, the IMF approach, which is very similar to ARMS, did not experience difficulties. The ILU preconditioner was very competitive on these problems, outperforming all other preconditioners.

Notice again the good performance of IMF on the matrices derived from CFD problems. IMF was the only preconditioner able to solve the `rim` and `rma10` systems. Both systems resulted from a FEM discretization of a 3D domain.

We also experimented with the SPAI preconditioner on all of these problems. However, we were not able to select parameter settings for which said preconditioner would converge, within reasonable time.

More CFD problems. As a final testament to IMF's robustness, we consider some matrices from the Matrix Market [1], in table Table 6.7. In this case, the underlying element structure was perfectly reconstructed from the assembled matrix.

From the table it is clear that IMF was more robust than the other preconditioners; it never experienced breakdown nor arithmetic failures. IMF was also very effective on this problem. It converged at low fill factors for which the other preconditioners would not converge or broke down.

Only IMF and ILUT were able to solve the linear system for all values of the Reynolds number. However, the fill factor of ILUT was impractically high; it required at least 6.5 times the memory of IMF.

Table 6.6: Comparison of the convergence of IMF, ARMS, ARMS-ND and ILUT on the test matrices in Table 6.4.

		IMF(τ)		ARMS(τ)		ARMS-ND(τ)		ILUT(τ)		ILU(κ)	
af23560	κ	1	2	0.05	0.01	0.05	0.03	0.5	0.3	1	2
	T_f	3.92	4.73	0.39	0.78	1.11	1.89	0.44	0.67	**0.29**	0.61
	T_s	$*$	44.41	2.23	1.12	3.19	2.41	0.26	**0.25**	0.58	0.37
	its	$*$	1545	108	32	105	62	11	**8**	38	15
	fill	4.46	7.21	**1.43**	3.34	4.00	5.99	4.69	6.48	2.38	4.42
d_pretok	κ	0	2	0.5	0.1	0.025	0.01	0.5	0.25	3	4
	T_f	**2.42**	4.62	5.47	11.55	5.33	393.33	2.75	4.19	‡	‡
	T_s	15.67	12.26	†	†	$*$	$*$	30.27	**5.59**	‡	‡
	its	201	105	†	†	$*$	$*$	316	**50**	‡	‡
	fill	**1.03**	4.60	3.57	6.66	3.19	18.08	4.92	6.45	‡	‡
matrix_9	κ	0	1	0.05	0.02	0.025	0.01	0.85	0.1	0	1
	T_f	5.06	10.53	1.05	2.70	3.65	9.85	6.62	51.50	**0.42**	0.58
	T_s	2.17	2.50	12.44	11.35	31.18	108.21	$*$	$*$	1.48	**1.04**
	its	43	33	202	108	271	485	$*$	$*$	48	**25**
	fill	2.06	5.29	1.49	4.19	4.19	11.34	10.91	17.51	**1.00**	2.11
rim	κ	2	3	0.15	0.1	0.1	0.05	0.015	0.01	4	5
	T_f	**1.83**	4.98	6.21	4.33	6.73	54.71	1.90	2.41	‡	‡
	T_s	8.77	**5.70**	†	†	†	†	†	†	‡	‡
	its	289	**138**	†	†	†	†	†	†	‡	‡
	fill	**4.42**	6.53	5.20	4.68	5.25	10.70	5.53	6.05	‡	‡
rma10	τ	4	5	0.5	0.1	0.1	0.05	0.1	0.01	5	6
	T_f	**14.61**	34.14	68.41	490.91	72.05	343.47	73.55	251.02	‡	‡
	T_s	50.22	**32.76**	†	†	$*$	$*$	†	†	‡	‡
	its	544	**275**	†	†	$*$	$*$	†	†	‡	‡
	fill	**6.26**	8.55	7.90	15.45	6.12	11.83	5.46	6.71	‡	‡
stokes128	κ	0	2	0.001	0.0005	0.003	0.002	0.2	0.05	2	3
	T_f	1.63	1.94	0.69	1.03	2.17	1082.6	0.41	1.29	**0.38**	0.52
	T_s	18.00	**9.68**	20.42	19.37	†	$*$	$*$	14.36	65.36	41.45
	its	892	302	328	**238**	†	$*$	$*$	250	1965	1152
	fill	**1.03**	4.23	4.00	5.74	8.32	10.26	4.05	9.06	3.46	4.49
turon_m	κ	0	1	0.95	0.1	0.1	0.01	0.95	0.25	5	6
	T_f	**2.44**	3.22	5.51	22.53	1.69	64.50	4.41	10.17	‡	‡
	T_s	23.16	26.81	†	†	$*$	$*$	4.91	**3.70**	‡	‡
	its	289	268	†	†	$*$	$*$	40	**20**	‡	‡
	fill	**1.04**	2.56	3.39	14.18	1.05	10.59	6.51	10.63	‡	‡

Table 6.7: Comparison of IMF, ARMS, ARMS-ND, ILUT and ILU on the `e40rXXXX` test matrices from the Matrix Market.

Re		IMF(κ)		ARMS(κ)		ARMS-ND(κ)		ILU(κ)		ILUT(κ)	
	κ	0	1	0.005	0.0001	0.005	0.003	3	4	0.05	0.01
500	T_f	**0.15**	0.42	0.18	2.08	0.30	0.33	‡	‡	2.46	5.55
	T_s	1.88	3.49	∗	9.22	∗	∗	‡	‡	**1.53**	1.91
	its	232	280	∗	150	∗	∗	‡	‡	35	**30**
	fill	**1.03**	2.59	1.04	8.76	2.04	2.37	‡	‡	8.82	15.71
1000	T_f	**0.15**	0.44	0.20	3.01	0.36	0.42	‡	‡	2.76	5.73
	T_s	1.96	1.72	†	21.34	∗	∗	‡	‡	1.74	**1.43**
	its	248	140	†	322	∗	∗	‡	‡	40	**22**
	fill	**1.03**	2.59	1.12	9.62	2.35	2.88	‡	‡	9.05	15.85
2000	T_f	**0.15**	0.42	0.37	5.66	9.17	10.38	‡	‡	5.03	6.81
	T_s	3.14	5.68	†	54.56	∗	∗	‡	‡	9.30	**1.90**
	its	398	468	†	612	∗	∗	‡	‡	162	**28**
	fill	**1.03**	2.59	1.86	14.88	11.08	12.55	‡	‡	13.66	18.28
3000	T_f	0.16	**0.43**	1.39	12.63	47.77	54.20	‡	‡	6.37	7.43
	T_s	∗	4.55	†	∗	∗	∗	‡	‡	23.63	**2.72**
	its	∗	372	†	∗	∗	∗	‡	‡	362	**38**
	fill	1.03	**2.59**	3.74	25.14	19.86	21.97	‡	‡	16.76	19.50
4000	T_f	0.15	**0.43**	5.14	24.80	80.92	84.16	‡	‡	7.94	9.21
	T_s	∗	4.32	†	∗	†	†	‡	‡	∗	**3.17**
	its	∗	350	†	∗	†	†	‡	‡	∗	**40**
	fill	1.03	**2.59**	10.37	30.55	23.51	25.84	‡	‡	19.66	21.81
5000	T_f	**0.15**	0.44	5.42	29.25	85.81	100.46	‡	‡	8.93	9.52
	T_s	11.42	**6.00**	†	∗	†	†	‡	‡	∗	7.82
	its	1412	482	†	∗	†	†	‡	‡	∗	**98**
	fill	**1.03**	2.59	8.47	34.54	23.89	26.26	‡	‡	20.81	22.18

7. Conclusions. We proposed an incomplete multifrontal method (IMF) that computes an approximate multilevel block-LU factorization of an element-structured linear system. It can be regarded as an extension of the multifrontal method to approximate factorizations.

Numerical experiments indicate that this approach is robust and powerful on some linear systems arising from the numerical simulation of an incompressible Newtonian flow in a 2D domain. In one problem, an order-of-magnitude speedup was achieved over ARMS, ARMS-ND, ILU, ILUT and SPAI.

The throughput of IMF was shown to be competitive with, or better than, the SPAI method, which is credited for its computational efficiency. As IMF resembles the multifrontal method, it could benefit from the parallelization techniques employed by the latter. A parallel implementation has yet to be investigated.

We applied IMF to solve general linear systems, by extracting an artificial element structure. Provided that this artificial element structure has the properties of a finite element discretization, i.e. small dense matrices with a low interconnectivity, the results were promising. IMF was shown to be more robust than ARMS, ARMS-ND, ILU, ILUT and SPAI, even in the presence of indefiniteness, at a comparable memory consumption. We recognize that a better technique must be designed to solve general sparse matrices. To this end, we believe that the techniques employed by the supernodal multifrontal method are a promising direction.

The IMF approach was unsuccessful on some linear systems, a problem often indicated by a high number of levels in the factorization and high fill factors at very small values of κ. We found the problem to be slow coarsening. That is, the number of variables in the reduced system does not decrease sufficiently fast. We believe that a more involved partitioning technique is required. The nested dissection approach of ARMS-ND, for instance, never displayed this behavior. Also, different discard policies may need to be investigated to tackle this issue.

Acknowledgements. This paper presents research results of the Belgian Network DYSCO (Dynamical Systems, Control, and Optimization), funded by the Interuniversity Attraction Poles Programme, initiated by the Belgian State Science Policy Office. The scientific responsibility rests with its author(s).

The authors would like to thank Patrick Amestoy for encouraging discussions.

REFERENCES

[1] *The matrix market.* http://math.nist.gov/MatrixMarket/index.html.

[2] F. L. Alvarado and R. Schreiber, *Optimal parallel solution of sparse triangular systems*, SIAM Journal on Scientific Computing, 14 (1993), pp. 446–460.

[3] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, PA, third ed., 1999.

[4] E. Anderson and Y. Saad, *Solving sparse triangular linear systems on parallel computers*, International Journal of High Speed Computing, 1 (1989), pp. 73–95.

[5] O. Axelsson, R. Blaheta, and M. Neytcheva, *Preconditioning of boundary value problems using elementwise Schur complements*, SIAM Journal on Matrix Analysis and Applications, 31 (2009), pp. 767–789.

[6] O. Axelsson and P. S. Vassilevski, *Algebraic multilevel preconditioning methods. I*, Numerische Mathematik, 56 (1989), pp. 157–177.

[7] ———, *Algebraic multilevel preconditioning methods, II*, SIAM Journal on Numerical Analysis, 27 (1990), pp. 1569–1590.

[8] S. Barnard and M. J. Grote, *A block version of the spai preconditioner*, in 9th SIAM Conference on Parallel Processing for Scientific Computing, 1999.

[9] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*, SIAM, Philadelphia, PA, 1994.

[10] M. Benzi, *Preconditioning techniques for large linear systems: a survey*, Journal of Computational Physics, 182 (2002), pp. 418–477.

[11] M. Brezina, A. J. Cleary, R. D. Falgout, V. E. Henson, J. E. Jones, T. A. Manteuffel, S. F. McCormick, and J. W. Ruge, *Algebraic multigrid based on element interpolation (AMGe)*, SIAM Journal on Scientific Computing, 22 (2000), pp. 1570–1592.

[12] Y. E. Campbell and T. A. Davis, *Incomplete LU factorization: A multifrontal approach*, Tech. Rep. TR-95-024, Computer and Information Sciences Department, University of Florida, Gainesville, FL, 32611 USA, October 1995.

[13] E. Chow and P. S. Vassilevski, *Multilevel block factorizations in generalized hierarchical bases*, Numerical Linear Algebra with Applications, 10 (2003), pp. 105–127.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, The MIT Press, second ed., 2001.

[15] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in ACM '69: Proceedings of the 1969 24th national conference, New York, NY, USA, 1969, ACM, pp. 157–172.

[16] T. A. Davis and Y. Hu, *The university of florida sparse matrix collection.* ACM Transactions on Mathematical Software (to appear).

[17] M. J. Daydé, J.-Y. L'Excellent, and N. I. M. Gould, *Element-by-element preconditioners for large partially separable optimization problems*, SIAM Journal on Scientific Computing, 18 (1997), pp. 1767–1787.

[18] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff, *A set of level 3 basic linear algebra subprograms*, ACM Transactions on Mathematical Software, 16 (1990), pp. 1–17.

[19] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear systems*, ACM Transactions on Mathematical Software, 9 (1983), pp. 302–325.

[20] H. C. Elman, A. Ramage, and D. Silvester, *Incompressible flow iterative solution software (ifiss) installation & software guide.* http://www.cs.umd.edu/ elman/ifiss/ifiss_guide.pdf, 2005.

[21] H. C. Elman, A. Ramage, and D. J. Silvester, *Algorithm 866: Ifiss, a matlab toolbox for modelling incompressible flow*, ACM Transactions on Mathematical Software, 33 (2007).

[22] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics*, Oxford University Press, Oxford, UK, 2005.

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

[24] GLAS, *Generic Linear Algebra Software.* https://www.cs.kuleuven.be/ karlm/glas, 2005.

[25] M. J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.

[26] I. Gustafsson and G. Lindskog, *A preconditioning technique based on element matrix factorizations*, Computer Methods in Applied Mechanics and Engineering, 55 (1986), pp. 201–220.

[27] M. T. Heath, E. Ng, and B. W. Peyton, *Parallel algorithms for sparse linear systems*, SIAM Review, 33 (1991), pp. pp. 420–460.

[28] P. Hénon, F. Pellegrini, P. Ramet, J. Roman, and Y. Saad, *Applying parallel direct solver techniques to build robust high performance preconditioners*, in Applied Parallel Computing, J. Dongarra, K. Madsen, and J. Wasniewski, eds., vol. 3732 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2006, pp. 611–619.

[29] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, 2nd ed., 2002, p. 260.

[30] T. J. R. Hughes, I. Levit, and J. Winget, *An element-by-element solution algorithm for problems of structural and solid mechanics*, Computer Methods in Applied Mechanics and Engineering, 36 (1983), pp. 241–254.

[31] J. E. Jones and P. S. Vassilevski, *AMGe based on element agglomeration*, SIAM Journal on Scientific Compututing, 23 (2001), pp. 109–133.

[32] R. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. Miller and J. Thatcher, eds., Plenum Press, 1972, pp. 85–103.

[33] J. K. Kraus, *Algebraic multilevel preconditioning of finite element matrices using local Schur complements*, Numerical Linear Algebra with Applications, 13 (2006), pp. 49–70.

[34] J. W. H. Liu, *The multifrontal method for sparse matrix solution: theory and practice*, SIAM Review, 34 (1992), pp. 82–109.

[35] T. A. Manteuffel, *An incomplete factorization technique for positive definite linear systems*, Mathematics of Computation, 34 (Apr., 1980), pp. 473–497.

[36] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Mathematics of Computation, 31 (1977), pp. 148–162.
[37] Y. Notay, *Robust parameter free algebraic multilevel preconditioning*, Numerical Linear Algebra with Applications, 9 (2002), pp. 409–428.
[38] ———, *Algebraic multigrid and algebraic multilevel methods: a theoretical comparison*, Numerical Linear Algebra with Applications, 12 (2005), pp. 522–543.
[39] ———, *Aggregation-based algebraic multilevel preconditioning*, SIAM Journal on Matrix Analysis and Applications, 27 (2006), pp. 998–1018.
[40] Y. Qu and J. Fish, *Multifrontal incomplete factorization for indefinite and complex symmetric systems*, International Journal for Numerical Methods in Engineering, 53 (2002), pp. 1433–1459.
[41] J. W. Ruge and K. Stüben, *Algebraic multigrid (AMG)*, in Multigrid Methods, Frontiers in Applied Mathematics, S. McCormick, ed., vol. 5, SIAM, 1986.
[42] Y. Saad, *ILUT: a dual threshold incomplete ILU factorization*, Numerical Linear Algebra with Applications, 1 (1994), pp. 387–402.
[43] ———, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.
[44] Y. Saad and B. Suchomel, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Numerical Linear Algebra with Applications, 9 (2002), pp. 359–378.
[45] K. Stüben, *Algebraic multigrid (AMG): Experiences and comparisons*, Applied Mathematics of Computation, 13 (1983), pp. 419–452.
[46] ———, *A review of algebraic multigrid*, Journal of Computational and Applied Mathematics, 128 (2001), pp. 281–309.
[47] W. F. Tinney and J. W. Walker, *Direct solutions of sparse network equations by optimally ordered triangular factorization*, Proceedings of the IEEE, 55 (1967), pp. 1801–1809.
[48] H. A. van der Vorst, *BI-CGSTAB: a fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems*, SIAM Journal on Scientific and Statistical Computing, 13 (1992), pp. 631–644.
[49] R. Vuduc, J. W. Demmel, K. A. Yelick, S. Kamil, R. Nishtala, and B. Lee, *Performance optimizations and bounds for sparse matrix-vector multiply*, in Proc. ACM/IEEE Conf. Supercomputing (SC), Baltimore, MD, USA, November 2002.
[50] R. Vuduc, S. Kamil, J. Hsu, R. Nishtala, J. W. Demmel, and K. A. Yelick, *Automatic performance tuning and analysis of sparse triangular solve*, in Proc. Wkshp. Performance Optimization of High-level Languages and Libraries (POHLL), at ACM Int'l. Conf. Supercomputing (ICS), New York, USA, June 2002.