SIAM J. SCI. COMPUT.
Vol. 35, No. 1, pp. A270–A293
© 2013 Society for Industrial and Applied Mathematics

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) preconditioner that extends supernodal multifrontal methods to incomplete factorizations. It can be used as a preconditioner in a Krylov-subspace method to solve large-scale sparse linear systems with an element structure, e.g., those arising 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 efficient partial pivoting. IMF is compared with the multilevel ARMS2, the level of fill-in ILU, and the threshold-based ILUTP preconditioners. Our experiments indicate that IMF is competitive with ARMS2 on saddle-point problems arising in the solution of the steady-state Navier–Stokes equation. Experiments with element-structured matrices arising from structural engineering applications, found in the University of Florida Sparse Matrix Collection, illustrate the robustness of IMF. Finally, the computational performance of IMF clearly surpasses that of the related ARMS2 preconditioner.

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

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

DOI. 10.1137/100818996

1. Introduction.

We propose a preconditioner for solving an element-structured sparse linear system $A\mathbf{x} = \mathbf{b}$ using Krylov-subspace methods. Such systems arise frequently in practice, e.g., from a discretization of a partial differential equation in the finite element method (FEM). 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^T is a zero matrix with a 1 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 nonzero intersection. In a FEM, the index set of an element corresponds to its degrees of freedom (dofs).

An incomplete *LU*-factorization is a generally applicable preconditioning technique that does not require an element structure. It applies Gaussian elimination to A while heuristically discarding some nonzero values. ILU(0) [28] is the simplest

*Submitted to the journal's Methods and Algorithms for Scientific Computing section December 21, 2010; accepted for publication (in revised form) September 5, 2012; published electronically January 15, 2013. A preliminary version of this paper appeared as the technical report [44]. 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 authors. This research was also partially funded by the Research Council KU Leuven grants PFV/10/002 (optimization in engineering) and OT/10/038 (multi-parameter model order reduction and its applications).

http://www.siam.org/journals/sisc/35-1/81899.html

†Numerical Approximation and Linear Algebra Group, Department of Computer Science, KU Leuven, B-3000 Leuven, Belgium (nick.vannieuwenhoven@cs.kuleuven.be, karl.meerbergen@cs.kuleuven.be). The first author's research was supported by a Ph.D. fellowship of the Research Foundation–Flanders (FWO).

approach, in which all fill-in values are dropped. Another strategy that works well in practice is the dual threshold strategy employed by ILUT [33]. Nevertheless, a significant disadvantage of ILU-type preconditioners is their sequential nature during forward- and backward substitution, which makes them difficult to parallelize [20]. Furthermore, a sparse triangular solve attains limited throughput [47].

During the past two decades, a significant amount of research focused on algebraic multigrid (see [41] and the references therein) and multilevel preconditioners [4, 5, 9, 29, 31, 37]. In these approaches, the preconditioner is constructed by recursively applying a two-level scheme [30]. It consists of partitioning the unknowns into two sets: fine and coarse. The fine variables are eliminated, and the *reduced system*, containing only coarse variables, is approximated. Typically, the reduced system is an approximate Schur complement or the Galerkin coarse grid matrix [30]. The factorization continues recursively with the reduced system.

Currently, a preconditioner is often stored as a sparse matrix. However, a major disadvantage of these data structures is their limited throughput in matrix-vector products and sparse triangular solves, which can be partially overcome by using optimized kernels that employ transformations to allow operations on dense submatrices [46, 47]. Nevertheless, we believe it is critical to *simultaneously* minimize the number of iterations and the execution time per iteration, because the time spent on preconditioning is the product of the two. A compromise should be sought between these competing goals; forming a more accurate factorization favors the former, whereas imposing additional restrictions to enable operations on dense matrices and allowing fewer nonzeros can decrease the time per iteration. Preconditioners can benefit from an element structure, as it naturally defines the sparse matrix as a set of overlapping dense matrices. We argue, however, that this structure is not yet maximally exploited in available preconditioners; the preconditioners [3, 7, 25, 26, 32] use the element structure during the factorization only, while the cost of an iterative method is usually dominated by repeatedly applying the preconditioner to a vector. We investigate whether an element structure can be profitable during this last phase as well.

In this paper, we describe a new multifrontal[1] algorithm for computing a multilevel block-ILU preconditioner, inspired by ARMS [37], for systems with an element structure. In operating on elements exclusively and computing Schur complements of dense frontal matrices, it is related to the multifrontal method of Duff and Reid [14]. In that context, it can be interpreted as a set of supernode eliminations organized into discrete levels. The first few levels are factored exactly. However, from a given level onward, the factorization is inexact but nevertheless operates on the element structure in a multifrontal manner.

The main contribution of this paper is an extension of the multifrontal method with our symbolic element-based dropping rule to obtain a new multilevel block-ILU preconditioner for element-structured linear systems, called the incomplete multifrontal (IMF) method. It is *robust* due to partial pivoting within frontal matrices, *accurate* by clustering several eigenvalues at unity, *efficient during factorization* by computing the Schur complements with LAPACK and BLAS3, and *efficient during application* by maximally operating on dense matrices and using BLAS2 routines.

This paper is structured as follows. An overview of IMF is presented in the next section. A multifrontal algorithm for computing it is detailed in section 3. Thereafter,

[1] We assume the reader has a basic understanding of (supernodal) multifrontal direct solvers, as reviewed in Liu's article [27].

in section 4, the IMF(κ) preconditioner is presented. It is then compared to related preconditioners in section 5. Implementation details are given in section 6, and our numerical experiments appear in section 7. In section 8, conclusions are presented.

Concerning notation, the following conventions apply throughout. Uppercase letters denote matrices (A, M), and lowercase boldface letters denote vectors ($\mathbf{x}$, $\mathbf{b}$). A set of elements is typeset in calligraphic uppercase letters ($\mathcal{E}$, $\mathcal{D}$) and its elements in lowercase letters (e, d). Uppercase calligraphic letters can also denote operators acting on elements ($\mathcal{I}(\cdot)$, $\mathcal{N}(\cdot)$). The integer f denotes the final level number, and $0 \leq k \leq f$ denotes the level subscript of a matrix, vector, or element set. A matrix with a double subscript, e.g., $D_{k,i}$, is a shorthand for $(D_k)_i$: it denotes the ith block of the matrix D_k. 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. The elements of an ordered set S are denoted as $S = \langle s_1, s_2, s_3 \rangle$.

2. IMF preconditioner. IMF is an incomplete block LU-factorization with multilevel structure that is computed by a multifrontal algorithm. It can be considered an extension of the supernodal multifrontal method [14, 27]. In particular, we adapt the idea of frontal matrices to the computation of an incomplete Schur complement. Our algorithm operates on the element structure $\mathcal{E}$ of A exclusively; the latter is never assembled into a sparse matrix. Contrary to multifrontal methods, we do not employ the elimination tree [27] to guide the factorization. Instead, we consider a dynamic ordering strategy that identifies a set of independent supernodes (also called *fronts*). After multifrontal elimination of these supernodes, the factorization proceeds recursively with the remaining variables. Our algorithm groups the supernodes in distinct levels, similar to the supernodes at a fixed depth in the elimination tree. In IMF, every supernode corresponds to the indices of one element.

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 admits 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 block diagonal matrix with dense matrices on the diagonal. We approximate A_k^P 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 computed without dropping and is stored as a set of dense matrices. This allows us to employ dense matrix-vector multiplications, which can attain higher throughput than triangular solves [12, p. 64]. However, one may argue that, according to [21, p. 260], applying the inverse of a matrix is less stable than solving the triangular systems. Potentially losing a few digits of accuracy is not an issue, as M_k is applied only 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. Factorization (2.2) is applied recursively to M_{k+1} until all unknowns have been eliminated. At the final level f, $M_f^P = \bar{D}_f$. We thus obtain

a block-ILU factorization with multilevel structure:

$$M = P \begin{bmatrix} I & & & & \\ & I & & & \\ \bar{L}_0\bar{D}_0^{-1} & & \ddots & & \\ & \bar{L}_1\bar{D}_1^{-1} & & I & \\ & & \cdots & \bar{L}_{f-1}\bar{D}_{f-1}^{-1} & I \end{bmatrix} \begin{bmatrix} \bar{D}_0 & & & \bar{U}_0 & \\ & \bar{D}_1 & & \bar{U}_1 & \\ & & \ddots & \vdots & \\ & & & \bar{D}_{f-1} & \bar{U}_{f-1} \\ & & & & \bar{D}_f \end{bmatrix} P^T. \tag{2.3}$$

Herein, P is the permutation matrix that combines the actions of the permutation matrices P_k for all k. To simplify the discussion of the algorithms, we assume that P has already been applied to A_0. Consequently, $P_k = I$, $A_k^P = A_k$, and $M_k^P = M_k$ for all k. For notational convenience, we will also drop the bars in (2.2).

3. Multifrontal computation. In this section, we describe a multifrontal algorithm for efficiently computing factorization (2.2). A crucial step is constructing a new element structure $\mathcal{E}_{k+1}$ for the reduced system M_{k+1}, given an element structure $\mathcal{E}_k$ of M_k. We assume an element structure $\mathcal{E}_0$ of M_0 is known, such as, e.g., in linear systems that were derived from a FEM.

In the next subsection, we consider the construction of the permutation matrix P_k, which is induced by a set of independent supernodes $\mathcal{D}_k \subseteq \mathcal{E}_k$. The elimination of supernodes can proceed in one of two ways: either the exact Schur complement is formed (section 3.2), or an approximation to it (section 3.3). Finally, in section 4, the aforementioned techniques are combined, resulting in the IMF(κ) preconditioner.

3.1. Partitioning the unknowns. A scheme is presented wherein the matrices on the block diagonal of D_k correspond to element matrices so that the inverse can be computed without generating fill-in. In the terminology of multifrontal methods, a set of independent supernodes is selected from the element structure $\mathcal{E}_k$. An independent set (IS)—a subset of the nodes in a graph which are not connected to one another [34, 38]—in the graph representation of a sparse matrix induces a permutation such that the matrix admits a 2×2 block structure in which the $(1,1)$ block is a diagonal matrix [38]. Similarly, in the present context, a block diagonal matrix can be obtained from an IS in the *element graph*. This is a graph in which every element in $\mathcal{E}_k$ is represented by a node [14]. Two nodes are connected if their corresponding elements, say $d, e \in \mathcal{E}_k$, are *neighbors*, i.e., if $\mathcal{I}(d) \cap \mathcal{I}(e) \neq \emptyset$. The set of neighbors of $e \in \mathcal{E}_k$ will be denoted by $\mathcal{N}(e)$; the number of neighbors of an element is its *degree*. If $\mathcal{D}_k := \langle d_1, d_2, \ldots, d_m \rangle \subseteq \mathcal{E}_k$ is an IS in the element graph, then the permutation matrix P_k, corresponding to the renumbering

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

results in the required structure (see, e.g., [44])

$$M_k^P = \begin{array}{c} \\ \mathcal{I}(d_1) \\ \mathcal{I}(d_2) \\ \vdots \\ \mathcal{I}(d_m) \\ \mathbf{r}_k \end{array} \begin{array}{c} \begin{array}{cccc} \mathcal{I}(d_1) & \mathcal{I}(d_2) & \cdots & \mathcal{I}(d_m) \end{array} \\ \left[\begin{array}{cccc|c} D_{k,1} & & & & U_{k,1} \\ & D_{k,2} & & & U_{k,2} \\ & & \ddots & & \vdots \\ & & & D_{k,m} & U_{k,m} \\ \hline L_{k,1} & L_{k,2} & \cdots & L_{k,m} & S_k \end{array}\right] \end{array} := \begin{array}{c} \\ \mathbf{q}_k \\ \mathbf{r}_k \end{array} \begin{array}{c} \begin{array}{cc} \mathbf{q}_k & \mathbf{r}_k \end{array} \\ \left[\begin{array}{c|c} D_k & U_k \\ \hline L_k & S_k \end{array}\right] \end{array}. \tag{3.1}$$

We refer to the elements in $\mathcal{D}_k$ as *pivotal elements*.

In section 4.1, 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 minimal.[2] To this end, we apply Algorithm 2.2 from [34] to the element graph. For the sake of completeness, it is presented below in the notation of the present paper.

ALGORITHM 3.1. Constructing a set of pivotal elements.

1: Sort elements in $\mathcal{E}_k$ by their degree in the element graph.
2: Initialize: $\mathcal{M}_k \leftarrow \emptyset$, and $\mathcal{D}_k \leftarrow \emptyset$.
3: **for** $e \in \mathcal{E}_k$ **in sorted sequence do**
4: **if** $e \notin \mathcal{M}_k$ **then**
5: Add element: $\mathcal{D}_k \leftarrow \mathcal{D}_k \cup \{e\}$.
6: Mark neighbors: $\mathcal{M}_k \leftarrow \mathcal{M}_k \cup \mathcal{N}(e) \cup \mathcal{N}(\mathcal{N}(e))$.
7: **end if**
8: **end for**

3.2. Computing the exact Schur complement. We discuss an adaption of the classic supernodal multifrontal elimination algorithm [27] for computing the exact Schur complement of M_k in (3.1), given an element structure $\mathcal{E}_k$ of M_k and a set of pivotal elements $\mathcal{D}_k \subseteq \mathcal{E}_k$. In the next subsection, we propose a strategy for discarding fill-in values.

The *frontal matrix* [27] associated with a supernode $d_i \in \mathcal{D}_k$ is defined as

$$F_{k,i} := \sum_{e \in \mathcal{N}(d_i)} M_k^{[e]} := \sum_{e \in \mathcal{N}(d_i)} \begin{bmatrix} D_k^{[e]} & U_k^{[e]} \\ L_k^{[e]} & S_k^{[e]} \end{bmatrix} := \begin{bmatrix} \hat{D}_{k,i} & \hat{U}_{k,i} \\ \hat{L}_{k,i} & \hat{S}_{k,i} \end{bmatrix}, \tag{3.2}$$

where $F_{k,i}$ is partitioned compatibly with (3.1).[3] It is well established that the frontal matrix is fully assembled at the indices of the diagonal element d_i [27, 44]. That is, comparing with (3.1), the frontal matrix of d_i is given by

$$F_{k,i} = \left[\begin{array}{c|c} \hat{D}_{k,i} & \hat{U}_{k,i} \\ \hline \hat{L}_{k,i} & \hat{S}_{k,i} \end{array}\right] = \begin{array}{c} \\ \mathcal{I}(d_1) \\ \vdots \\ \mathcal{I}(d_i) \\ \vdots \\ \mathcal{I}(d_m) \\ \mathbf{r}_k \end{array} \begin{array}{c} \begin{array}{ccccc} \mathcal{I}(d_1) & \cdots & \mathcal{I}(d_i) & \cdots & \mathcal{I}(d_m) \end{array} \quad \mathbf{r}_k \\ \left[\begin{array}{ccc|c} & & & \\ & & & \\ & D_{k,i} & & U_{k,i} \\ & & & \\ & & & \\ \hline & L_{k,i} & & \hat{S}_{k,i} \end{array}\right] \end{array}, \tag{3.3}$$

where $\mathbf{r}_k$ is as in (3.1). Note that the $(2,2)$ block is only partially assembled; see [44] for details. By combining (3.1), (3.2), and (3.3) we obtain

$$M_{k+1} := S_k - L_k D_k^{-1} U_k = \sum_{d_i \in \mathcal{D}_k} (\hat{S}_{k,i} - \hat{L}_{k,i} \hat{D}_{k,i}^{-1} \hat{U}_{k,i}) + \sum_{e \in \mathcal{H}_k} S_k^{[e]}, \tag{3.4}$$

[2] Algorithm 3.1 has a goal different from that of the minimum degree ordering algorithm, which reorders the unknowns to *reduce fill-in* [18]. We are concerned with *reducing the number of variables* because we can show that the spectrum of the preconditioned system improves.

[3] Contrary to [27], we define a frontal matrix of $d \in \mathcal{D}_k$ as a sparse matrix which is nonzero only at the positions $(\mathcal{I}(d) \cup \mathcal{I}_{\mathcal{N}}(d)) \times (\mathcal{I}(d) \cup \mathcal{I}_{\mathcal{N}}(d))$, rather than by using an extend-add operation. Naturally, we treat the frontal matrices as dense matrices in our implementation.

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

$$M_{k+1}^{[h]} := S_k^{[h]} \text{ if } h \in \mathcal{H}_k \quad \text{and} \quad M_{k+1}^{[g_i]} := \hat{S}_{k,i} - \hat{L}_{k,i}\hat{D}_{k,i}^{-1}\hat{U}_{k,i} \text{ if } d_i \in \mathcal{D}_k, \tag{3.5}$$

where g_i is a new element whose index set is $\mathcal{I}(g_i) = \mathcal{I}_{\mathcal{N}}(d_i) := \mathcal{I}(\mathcal{N}(d_i)) \backslash \mathcal{I}(d_i)$, resulting from the elimination of $d_i \in \mathcal{D}_k$, then

$$\mathcal{E}_{k+1} := \mathcal{H}_k \cup \mathcal{G}_k, \qquad \text{where } \mathcal{G}_k := \{g_i \,|\, d_i \in \mathcal{D}_k\}, \tag{3.6}$$

is an element structure of M_{k+1}. The unknowns of the pivotal elements $\mathcal{D}_k$ can thus be eliminated, in a multifrontal manner, by computing the Schur complement of the dense frontal matrices $F_{k,i}$ for $d_i \in \mathcal{D}_k$. Such a Schur complement matrix can then be interpreted as the element matrix of a new element $g_i \in \mathcal{G}_k$. In [14], such new elements are referred to as "generated" elements, while in [27] they are called "update matrices."

Example. Consider the following finite element discretizations:



In the left panel, the neighbors of d are shaded. Let $d \in \mathcal{D}$ be the only pivotal element. By eliminating the dofs of d, we generate a new element $g \in \mathcal{G}$, whose index set is $\mathcal{I}_{\mathcal{N}}(d)$. This generated element g is shown in the right panel, which depicts the element structure we obtain after eliminating d.

We mold (3.5) and (3.6) into Algorithm 3.2; see lines 3 and 6.

ALGORITHM 3.2. Computation of the Schur complement.

1: Initialize: $\mathcal{E}_{k+1} \leftarrow \emptyset$, and $\mathcal{H}_k \leftarrow \mathcal{E}_k \backslash \mathcal{N}(\mathcal{D}_k)$
2: **for** $h \in \mathcal{H}_k$ **in parallel do**
3: Copy element: $M_{k+1}^{[h]} \leftarrow S_k^{[h]}$, and $\mathcal{E}_{k+1} \leftarrow \mathcal{E}_{k+1} \cup \{h\}$
4: **end for**
5: **for** $d_i \in \mathcal{D}_k$ **in parallel do**
6: Create new element g_i: $\mathcal{I}(g_i) \leftarrow \mathcal{I}_{\mathcal{N}}(d_i)$, and $M_{k+1}^{[g_i]} \leftarrow \hat{S}_{k,i} - \hat{L}_{k,i}\hat{D}_{k,i}^{-1}\hat{U}_{k,i}$
7: Add to next level: $\mathcal{E}_{k+1} \leftarrow \mathcal{E}_{k+1} \cup \{g_i\}$
8: **end for**

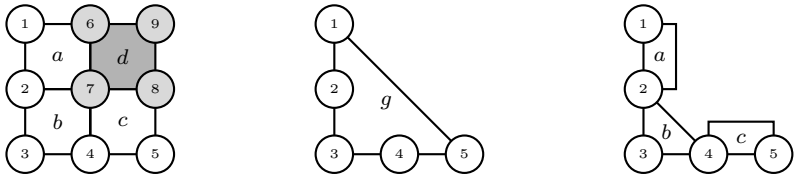
The frontal matrices are treated as dense matrices. We compute the inverse via the LAPACK [2] routines `dgetrf` and `dgetri`, which use partial pivoting, and the matrix product via the BLAS3 [13] routine `dgemm`. Both loops (lines 2–4 and 5–8) can proceed independently of each other; in addition, the iterations of each loop can be executed in parallel.

3.3. Computing an approximate Schur complement. The element structure (3.6) assembles to the exact Schur complement of M_k. Applying this technique at every level results in a supernodal multifrontal direct method. Here, we propose a new symbolic dropping rule to limit the fill-in that originates from the generated elements. The proposed algorithm operates in a multifrontal manner.

The sparsity of M_k is the union of the sparsities of the elements in $\mathcal{E}_k$. It thus seems natural to approximate the Schur complement by dropping elements from element structure (3.6). As fill-in can occur only within the generated elements $\mathcal{G}_k$, we will only drop elements there. The *incomplete* method we propose does not introduce the generated element $g_i \in \mathcal{G}_k$, resulting from the elimination of a pivotal element

$d_i \in \mathcal{D}_k$, to the element structure of the reduced system. Instead, the element matrices of the neighbors of the pivotal element d_i are modified so that they assemble to the non–fill-in values of the generated element g_i. Only the fill-in values are thus effectively discarded. An example illustrates this idea.

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



$d \in \mathcal{D}_0$ is the sole pivotal element. Assume that the element matrix of the generated element g in the middle panel, resulting from the elimination of d in the left panel, is G, and that the matrix resulting from dropping all fill-in values[4] in G is G':

$$G = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \\ 41 & 42 & 43 & 44 & 45 \\ 51 & 52 & 53 & 54 & 55 \end{bmatrix} \end{array}, \quad \text{and} \quad G' = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 11 & 12 & 0 & 0 & 0 \\ 21 & 22 & 23 & 24 & 0 \\ 0 & 32 & 33 & 34 & 0 \\ 0 & 42 & 43 & 44 & 45 \\ 0 & 0 & 0 & 54 & 55 \end{bmatrix} \end{array}. \tag{3.7}$$

In Algorithm 3.2, the element structure of the exact Schur complement consists solely of the generated element g. In our incomplete multifrontal method, however, the generated element g is not added to the element structure of the approximate Schur complement. Instead, the neighbors a, b, and c of the pivotal element d are added to the element structure of the approximate Schur complement: $\mathcal{E}_1 = \{a, b, c\}$. This is illustrated in the right panel. The dofs of the pivotal element d have been removed, and d is not present at the next level. The element matrices of a, b, and c are chosen so that they assemble to G at the positions in the sparsity of $\mathcal{E}_1$. These element matrices can be constructed algorithmically. We simply add the values of G to the first[5] element that can store the values, if any. Thus, the element matrices are

$$M_1^{[a]} = \begin{array}{c} \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cc} 1 & 2 \end{array} \\ \begin{bmatrix} 11 & 12 \\ 21 & 22 \end{bmatrix} \end{array}, \quad M_1^{[b]} = \begin{array}{c} \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 23 & 24 \\ 32 & 33 & 34 \\ 42 & 43 & 44 \end{bmatrix} \end{array}, \quad M_1^{[c]} = \begin{array}{c} \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cc} 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 45 \\ 54 & 55 \end{bmatrix} \end{array}.$$

Note that the element matrices may overlap. For instance, both a and b have position $(2, 2)$ in their sparsity. This introduces some freedom in selecting the element matrices for the reduced system. The above process of storing the values of a generated element in 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', which is exactly equal to G except for the fill-in values, which are zero in G'.

From the above example, we extract a multifrontal scheme to compute the element structure of the reduced system. Define the *modified frontal matrix* of $d_i \in \mathcal{D}_k$ as

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

[4] These consist of all values at positions that do not appear in the sparsity defined by the current element set $\mathcal{E}_0$.

[5] In this example we assume that a lexicographic ordering is defined on the names of the elements.

The matrices $\hat{L}_{k,i}$, $\hat{D}_{k,i}$, and $\hat{U}_{k,i}$ are exactly those in (3.2). $F^0_{k,i}$ differs from $F_{k,i}$ in (3.2) only in the $(2,2)$ block. Consider the Schur complement of the modified frontal matrix $F^0_{k,i}$. This matrix can be interpreted as the element matrix of a modified generated element g^0_i. In fact, the sparsity of this element is the same as in the exact case. In contrast to the exact case, g^0_i is not added to the element structure of the reduced system. Instead, the values of its element matrix are distributed over the elements in $\mathcal{E}_k \backslash \mathcal{D}_k$. Thereafter, both the pivotal element $d_i \in \mathcal{D}_k$ and the generated element g^0_i are discarded.

LEMMA 3.1 (Theorem 2.3 of [44]). *If the Schur complement is computed as detailed above, then the element structure $\mathcal{E}_{k+1}$ assembles to the exact Schur complement at the positions in the sparsity of $\mathcal{E}_{k+1}$, and to zero elsewhere. That is,*

$$\sum_{e \in \mathcal{E}_{k+1}} \left(M^{[e]}_{k+1} \right)_{i,j} = \begin{cases} \left(S_k - L_k D_k^{-1} U_k \right)_{i,j} & \text{if } (i,j) \in \mathrm{Sp}\left(\mathcal{E}_k\right) \cap (\mathbf{r}_k \times \mathbf{r}_k), \\ 0 & \text{otherwise} \end{cases} \tag{3.8}$$

with $\mathbf{r}_k := \mathcal{I}(\mathcal{E}_k) \backslash \mathcal{I}(\mathcal{D}_k)$.

The proof of this lemma is straightforward but technical and notation-heavy. It consists of carefully determining the values of the approximate Schur complement. We refer the interested reader to [44].

The distribution of numerical values is not unique. We describe the approach that was adopted in our implementation and presented as Algorithm 3.3. It is not hard to prove that the values should be distributed over only the level-2 neighborhood $\mathcal{N}_2(d_i) := \mathcal{N}(d_i) \cup \mathcal{N}(\mathcal{N}(d_i))$ of a pivotal element $d_i \in \mathcal{D}_k$, instead of $\mathcal{E}_k \backslash \mathcal{D}_k$ [44, p. 14]. Moreover, the entire set $\mathcal{N}_2(d_i)$ is required only in a few exceptional cases;[6] in practice, $\mathcal{N}(d_i) \subset \mathcal{N}_2(d_i)$ of d_i suffices to distribute all non–fill-in values of g^0_i (line 4). This decreases the computational cost, as the values of the modified frontal matrix are distributed over fewer elements. Our experiments indicate that this simplification hardly affects the convergence rate, while decreasing the cost of the factorization. The distribution of the values of g^0_i (line 3) can be accomplished by sequentially processing the elements in $\mathcal{N}(d_i)$ (lines 4–10) as follows. For every $e \in \mathcal{N}(d_i)$ we add, at the matching positions of $M^{[e]}_{k+1}$ (line 7), the values of the element g^0_i. At the corresponding positions in g^0_i, zero is stored (line 8). g^0_i is discarded along with the pivotal element d_i, but the neighbors of the latter are added to $\mathcal{E}_{k+1}$ (lines 5–6).

ALGORITHM 3.3. Computation of an approximate Schur complement.

1: Initialize: $\mathcal{E}_{k+1} \leftarrow \emptyset$
2: **for** $d_i \in \mathcal{D}_k$ **in parallel do**
3: Compute local Schur complement: $G_{k,i} \leftarrow -\hat{L}_{k,i} \hat{D}^{-1}_{k,i} \hat{U}_{k,i}$
4: **for** $e \in \mathcal{N}(d_i) \backslash \{d_i\}$ **in sequence do**
5: Create a reduced element e': $\mathcal{I}(e') \leftarrow \mathcal{I}(e) \backslash \mathcal{I}(\mathcal{D}_k)$, and $M^{[e']}_{k+1} \leftarrow S^{[e]}_k$
6: Add to next level: $\mathcal{E}_{k+1} \leftarrow \mathcal{E}_{k+1} \cup \{e'\}$
7: **for** $(p,q) \in \mathrm{Sp}\left(e'\right) \cap \mathrm{Sp}\left(G_{k,i}\right)$ **do**
8: Distribute: $\left(M^{[e']}_{k+1}\right)_{p,q} \leftarrow \left(M^{[e']}_{k+1}\right)_{p,q} + (G_{k,i})_{p,q}$, and $(G_{k,i})_{p,q} \leftarrow 0$
9: **end for**
10: **end for**
11: **end for**
12: **for** $e \in \mathcal{E}_k \backslash \mathcal{N}(\mathcal{D}_k)$ **in parallel do**
13: Copy to next level: $M^{[e]}_{k+1} \leftarrow S^{[e]}_k$, and $\mathcal{E}_{k+1} \leftarrow \mathcal{E}_{k+1} \cup \{e\}$
14: **end for**

[6] This is related to small gaps in the element structure, which is uncommon in a FEM.

As in Algorithm 3.2, the computation of the Schur complements of the modified frontal matrices is by BLAS3 and LAPACK routines and may proceed in parallel (lines 2–11 and 12–14). We distribute the values of g_i^0 over only the neighbors of d_i.

4. IMF(κ) preconditioner. We propose a multifrontal multilevel block-ILU preconditioner with a symbolic element-based discard rule. The preconditioner has a single integer 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 are clustered around unity.

The IMF(κ) algorithm builds a preconditioner $M \approx A$ for the element-structured sparse matrix A. At every level $k \geq 0$, Algorithm 3.1 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: if $k < \kappa$, then Algorithm 3.2 computes the exact Schur complement; otherwise, an approximate Schur complement is computed using Algorithm 3.3. Both algorithms use partial pivoting within the matrices on the block diagonal. The reduced system is factored recursively until $\mathcal{D}_k = \mathcal{E}_k$ at some level k. Only the inverse of the block diagonal matrix is then computed.

In our approach, a level k fill-in value is a value which is not in the sparsity of the element structure $\mathcal{E}_{k-1}$ but does appear in the sparsity of $\mathcal{E}_k$. An IMF(0) preconditioner thus contains no fill-in values *outside of the sparsity pattern defined by* $\mathcal{E}_0$. This element structure may, however, include structural zeros.

The *level of fill-in* parameter κ should be set to a small value, as the amount of fill-in quickly increases with κ, especially in problems with a high interelement connectivity. If $\kappa = \infty$, a direct supernodal multifrontal method is obtained.

4.1. Spectrum. The parameter κ influences the approximation quality. An interesting property of IMF(κ) is, recalling (2.1) and (2.2), that

$$\bar{D}_k = D_k, \quad \bar{L}_k = L_k, \quad \text{and} \quad \bar{U}_k = U_k \quad \text{for all } 0 \leq k \leq \kappa, \tag{4.1}$$

which can be verified readily. We may expect more accurate factorizations when κ is increased. Indeed, if $0 \leq k \leq \kappa$, then from (2.1), (2.2), and (4.1), we find that

$$M_k^{-1} A_k = \begin{bmatrix} D_k & U_k \\ 0 & M_{k+1} \end{bmatrix}^{-1} \times \begin{bmatrix} D_k & U_k \\ 0 & A_{k+1} \end{bmatrix} = \begin{bmatrix} I & D_k^{-1} U_k (I - M_{k+1}^{-1} A_{k+1}) \\ 0 & M_{k+1}^{-1} A_{k+1} \end{bmatrix}. \tag{4.2}$$

The $(2,2)$ block is again of the same form if $k < \kappa$. At level κ, we approximate $A_{\kappa+1}$ by (3.8), so that $M_{\kappa+1}^{-1} A_{\kappa+1}$ is no longer an upper triangular matrix. Therefore,

$$M^{-1} A = \left[\begin{array}{c|ccc} I & \multicolumn{3}{c}{X_0} \\ \hline & \ddots & \multicolumn{2}{c}{\vdots} \\ \hline & & I & X_\kappa \\ \hline & & & M_{\kappa+1}^{-1} A_{\kappa+1} \end{array}\right], \text{ with } X_k = D_k^{-1} U_k (I - M_{k+1}^{-1} A_{k+1}). \tag{4.3}$$

The spectrum of the IMF(κ) preconditioned system is therefore given by $\sigma(M^{-1}A) = \{1\} \cup \sigma(M_{\kappa+1}^{-1} A_{\kappa+1})$, 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 3.1. Even the IMF(0) preconditioner has some eigenvalues at unity, assuming exact arithmetic.

4.2. Application. To apply IMF(κ) as a preconditioner in an iterative method, one could perform a standard multilevel block forward- and backward substitution, as follows. Let $M_k^P \mathbf{x}_k = \mathbf{b}_k$ be the system at level k. If we partition $\mathbf{x}_k$ and $\mathbf{b}_k$ compatibly with the block structure in (2.2), i.e., $\mathbf{b}_k^T = \begin{bmatrix}\mathbf{b}_{k,1}^T & \mathbf{b}_{k,2}^T\end{bmatrix}$ and $\mathbf{x}_k^T = \begin{bmatrix}\mathbf{x}_{k,1}^T & \mathbf{x}_{k,2}^T\end{bmatrix}$, then we compute

$$\mathbf{z}_{k+1} = \mathbf{b}_{k,2} - L_k(D_k^{-1}\mathbf{b}_{k,1}), \quad M_{k+1}\mathbf{x}_{k,2} = \mathbf{z}_{k+1}, \quad \mathbf{x}_{k,1} = D_k^{-1}(\mathbf{b}_{k,1} - U_k\mathbf{x}_{k,2}), \tag{4.4}$$

recursively solving the second equation. We refer to this as the *full system approach*.

A more economical scheme exists due to IMF's structure, which we call the *reduced system approach*. Considering (4.3), it can be seen that IMF(κ) performs an exact elimination of the variables on the first κ levels of the factorization. The approximation error is restricted to the *reduced system* $M_{\kappa+1}^{-1}A_{\kappa+1}$. Consequently, only the reduced system should be solved iteratively. To exploit this observation, we consider the preconditioned system $M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}$ with

$$\mathbf{x} = \begin{bmatrix}\mathbf{x}_1^T & \mathbf{x}_2^T & \cdots & \mathbf{x}_{\kappa+1}^T\end{bmatrix}^T \quad \text{and} \quad \tilde{\mathbf{b}} := M^{-1}\mathbf{b} = \begin{bmatrix}\tilde{\mathbf{b}}_1^T & \tilde{\mathbf{b}}_2^T & \cdots & \tilde{\mathbf{b}}_{\kappa+1}^T\end{bmatrix}^T$$

partitioned compatibly with the block structure in (4.3). Because this system is block upper triangular, we can apply a block backward substitution:

$$M_{\kappa+1}^{-1}A_{\kappa+1}\mathbf{x}_{\kappa+1} = \tilde{\mathbf{b}}_{\kappa+1} \quad \text{and} \quad \mathbf{x}_k = \tilde{\mathbf{b}}_k - D_k^{-1}U_k(I - M_{k+1}^{-1}A_{k+1})\mathbf{x}_{k+1}^{\rightarrow}$$

for all $0 \le k \le \kappa$, where $\mathbf{x}_{k+1}^{\rightarrow} := [\mathbf{x}_{k+1}^T \ \mathbf{x}_{k+2}^T \ \cdots \ \mathbf{x}_{\kappa+1}^T]^T$. The right equation can be computed efficiently by noting from (4.2) that $M_{k+1}^{-1}A_{k+1}\mathbf{x}_{k+1}^{\rightarrow} = \tilde{\mathbf{b}}_{k+1}^{\rightarrow}$, where $\tilde{\mathbf{b}}_{k+1}^{\rightarrow}$ is defined analogously as $\mathbf{x}_{k+1}^{\rightarrow}$. For handling the left equation, on the other hand, first note that $M\tilde{\mathbf{b}} = \mathbf{b}$, and from (4.4) it follows that $M_{\kappa+1}\tilde{\mathbf{b}}_{\kappa+1} = \mathbf{z}_{\kappa+1}$. The left equation consequently reduces to $A_{\kappa+1}\mathbf{x}_{\kappa+1} = \mathbf{z}_{\kappa+1}$, which can thus be handled by an iterative method using $M_{\kappa+1}$ as preconditioner. Explicitly computing $A_{\kappa+1}$ is not practical due to fill-in; nevertheless, $A_{\kappa+1}$ can be applied efficiently due to (4.1):

$$A_{\kappa+1} = SPAP^TS^T - \sum_{k=0}^{\kappa}(SL_k)D_k^{-1}(U_kS^T), \tag{4.5}$$

with P as in (2.3) and $S = \left[\begin{smallmatrix}0 & 0\\ 0 & I\end{smallmatrix}\right]$ a square matrix that selects only the variables remaining in $A_{\kappa+1}$; i.e., the size of I is the size of $A_{\kappa+1}$. We compute $SPAP^TS^T$ explicitly and discard A; the latter is no longer needed.

The foremost advantage of the reduced system approach is its lower cost per iteration. A large reduction stems from the inverse of the first κ diagonal matrices D_k, $0 \le k \le \kappa$, which are applied only once per iteration, whereas in the full approach they are applied twice.

5. Related work.

Campbell and Davis [7]. An adaptation of the unsymmetric pattern multifrontal algorithm [10] called the multifrontal incomplete LU-factorization (MFILU) was proposed by Campbell and Davis [7] for computing an ILU preconditioner. MFILU has no separate analyze and factorize phase; the ordering of the unknowns is determined dynamically and incorporates numerical pivoting. A pivot that is sufficiently numerically stable and limits fill-in is selected heuristically. Thereafter, its associated frontal matrix is assembled. Several pivots are eliminated within this matrix to increase computational efficiency, while a symbolic dropping strategy may discard the

generated fill-in values. A new pivot is sought whenever the front has been completely eliminated, or no suitable pivot can be found within the frontal matrix.

MFILU and IMF both use supernode eliminations, pivoting within frontal matrices, and employ LAPACK and BLAS3 for computational efficiency. However, they also differ in several aspects. First, MFILU constructs an ILU factorization, whereas IMF groups the independent frontal matrices in levels to obtain a multilevel block-ILU factorization. This avoids the sequential triangular solves of MFILU by replacing them with sparse and dense matrix-vector products. Second, IMF stores the factors of $L_k D_k^{-1}$, while MFILU computes the product and may discard fill-in values. Furthermore, in MFILU the local Schur complement is not computed if the *resulting* fill-in level would be too high; however, not all of these values are fill-in. As a result, some updates to non–fill-in positions are not incorporated, whereas in IMF these updates are always performed, which improves the accuracy. Third, MFILU is applicable to general sparse unsymmetric matrices, whereas IMF can be applied efficiently only to element-structured matrices with a symmetric sparsity pattern.

Qu and Fish [32]. Qu and Fish [32] extended MFILU with a numerical dropping rule for symmetric sparse matrices. The multifrontal threshold incomplete LU-factorization (MFILUT) employs a separate analyze and factorize phase. In the former, an elimination tree is constructed to identify initial supernodes, and in the latter, the input matrix is appropriately scaled and the frontal matrices are processed.

The same remarks as for MFILU apply to MFILUT: some fill-in values that IMF represents in factored form are discarded, the factorization results in unstructured sparse lower and upper triangular matrices, and some values in the Schur complement of a frontal matrix are not computed, regardless of whether those values are fill-in or not.

Saad and Suchomel [37]. IMF was inspired by the ARMS-type preconditioners [34, 36, 37]. We discuss ARMS2, proposed in [37] by Saad and Suchomel. ARMS2(b, p, τ, q, α) is a multilevel block-ILU preconditioner, given by

$$P_k A_k P_k^T = \begin{bmatrix} D_k & U_k \\ L_k & S_k \end{bmatrix} \approx \begin{bmatrix} V_k & 0 \\ L_k W_k^{-1} & I \end{bmatrix} \times \begin{bmatrix} W_k & V_k^{-1} U_k \\ 0 & A_{k+1} \end{bmatrix}, \text{ with } 0 \le k < q, \tag{5.1}$$

whereby $D_k \approx V_k W_k$ represents 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 $H_k \approx V_k^{-1} U_k$. First, row scaling and column scaling are applied to A_k. Then, a heuristic algorithm constructs a set of independent blocks of size at least b and which are sufficiently diagonally dominant, employing $0 \le \alpha \le 1$ as thresholding parameter [37]. This results in the permutation matrix P_k, which is applied to A_k. Thereafter, the sparse block diagonal matrix D_k is factored using ILUT(p, τ) [33]. To compute the matrices G_k, H_k, and A_{k+1}, the ILUT dual-threshold dropping rule is used again. Finally, the factorization is recursively applied to A_{k+1} until the number of levels equals the maximum number of levels q. The final system A_q is factored using ILUT with column pivoting (ILUTP) [35, sect. 10.4.4], resulting in the factors V_q and W_q. ARMS2 is applied by a recursive block forward- and backward substitution.

Aside from differences between pointwise and multifrontal algorithms, ARMS2 and IMF process the block diagonal matrix differently. In ARMS2, a heuristic algorithm seeks *sparse blocks*, whereas IMF selects small dense blocks from the element structure. The latter can invert the dense blocks exactly with partial pivoting because no fill-in is generated. In the former, however, this is infeasible due to fill-in; a dropping strategy is required. As a result, the approximation error is spread out over the

entire matrix in ARMS2, whereas in IMF it is restricted to the reduced system A_{k+1}. In addition, ARMS2 could discard values in G_k and H_k such that some updates in the Schur complement are inexact even though that update would not generate fill-in. IMF, on the other hand, always computes the exact Schur complement. Another difference is the solution phase, wherein IMF employs dense matrix-vector products to apply the preconditioner and ARMS2 requires sparse triangular solves. Furthermore, as ARMS2 generally discards some fill-in values in the approximate factorization of D_k, it is infeasible to employ a reduced system approach as in IMF.

6. Implementation details. The permutations P_k are applied to M_k at every level k so that we store only $M_k^P := P_k M_k P_k^T$. 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 global permutation is required per preconditioner application. This is advantageous because it avoids many indirect memory accesses and potentially large jumps in the memory, both of which hamper cache performance.[7]

To improve the throughput of the solution phase, we permute the block diagonal matrix D_k during the factorization phase so that the matrices on the block diagonal appear in order of increasing size. To apply the block diagonal matrix to a vector, optimized unrolled C++ code applies the matrices on the block diagonal whose size is smaller than 12×12, while the remainder are handled by BLAS3 calls. Our experiments indicate throughput increases 10%–35%.

We implemented IMF in C++, using the Generic Linear Algebra Software (GLAS) [19] and Boost libraries. GLAS provides basic data structures, such as Compressed Sparse Row (CSR) and the reference implementation [43] of IDR(s) [40]. We used the optimized ATLAS [48, 49] implementation of the BLAS interface for computing the matrix product required by the Schur complement, as well as the ATLAS implementation of `dgetrf` and `degtri` for computing the inverse of a frontal matrix, with partial pivoting. The Optimized Sparse Kernel Interface (OSKI) [45] library computes the sparse matrix-vector products. The ITSOL package[8] provides several preconditioners with which we compare IMF. The sparse data structures in ITSOL were converted to OSKI data structures. Our codes and all employed libraries were compiled using the `gcc` 4.4.3 compiler, with `-O3 -msse4.1` optimization flags enabled.

7. Numerical experiments. We compare the performance of IMF with preconditioners from the ITSOL package: ARMS2, ILU, and ILUTP.[9] The system matrix is reordered with RCM [17, 42] for improving its profile, increasing the throughput of the matrix-vector products; the preconditioners were constructed from the unpermuted system, unless stated otherwise. In all experiments, IDR(s) [40] with $s = 8$ was used as the iterative method.[10] Unless stated otherwise, the stopping criterion consisted of a maximum number of 10,000 matrix-vector products, or, equivalently, 1,250 IDR outer iterations, and a relative residual of 10^{-10}. As artificial right-hand side, we selected $A\mathbf{r}$, where $\mathbf{r}$ is a vector whose components were drawn from a standard

[7] It is possible to permute the system only once, so that the iterative process can be performed without permutations. However, the preconditioners from ITSOL apply the permutation as part of the preconditioning operation. We used this strategy in IMF as well, to ensure a fair comparison.

[8] See Saad's software page: http://www-users.cs.umn.edu/∼saad/software/.

[9] We additionally experimented with ARMS-C [36]. On the problems in this paper, ARMS2 generally performed better than ARMS-C; therefore, we omit the latter's results.

[10] Our experiments indicated that IDR(s) was a particularly suitable accelerator for the problems in section 7.1, outperforming Bi-CGSTAB and restarted GMRES.

normal distribution. All experiments were conducted on a single core of a computer system comprising two Intel Xeon E5645 hexa-core processors running at 2.40GHz and 48GB of main memory.

ARMS2 was described in section 5. We chose the parameters following guidelines from [37]. Our experiments indicated that $b = 500$ provided a good tradeoff between factorization speed and the number of levels. Parameter $p = 5{,}000$ was set sufficiently high to ensure that nonzero values were generally discarded only by the numerical dropping rule. The maximum number of levels $q = 25$ was sufficiently large to ensure that the final system contained few variables, while it remained sufficiently small to prevent an increase of the solution phase time.[11] In most experiments, the actual number of levels in ARMS2 was smaller than the maximum we imposed. The diagonal dominance criterion was $\alpha = 0.3$. Setting α higher requires more levels and increases the factorization time significantly, whereas a small α could render the factorization susceptible to instabilities.

ILUTP(p, τ) is the dual-threshold incomplete LU-factorization [35]. The matrix is initially reordered with RCM. Rowwise Gaussian elimination with column pivoting is then applied to the row scaled and column scaled matrix. A numerical dropping rule discards all values that are smaller than τ times the 1-norm of the current row. In each row, at most the p largest values are retained. We set $p = 1{,}000$ in all experiments, ensuring that the numerical dropping rule controls most of the dropping [33].

ILU(κ) is the level of fill-in incomplete LU-factorization [35]. We initially permute the system matrix by the multilevel ordering produced by IMF(κ). That allows us to investigate the relative merit of IMF's dropping strategy.

In the succeeding experiments, the reduced and full system approach of IMF will be denoted, respectively, by IMF-red and IMF-full. Unless stated otherwise, IMF-red is employed in the experiments.

The next subsection illustrates that IMF is competitive with ARMS2 on unsymmetric indefinite problems arising from the solution of a steady-state two-dimensional Navier–Stokes equation. Section 7.2 presents results from three-dimensional structural engineering problems arising in real applications, which exemplify the robustness of IMF. Finally, the computational performance of IMF is experimentally studied in section 7.3.

Throughout this section, the following notation applies: ϕ denotes the fill-in factor, i.e., the number of nonzeros in the preconditioner divided by the number of nonzeros in the system matrix A; T_f is the time, in seconds, needed to compute the preconditioner; T_s is the time, in seconds, needed to solve the system; υ denotes the number of matrix-vector products with A; and ρ is the relative residual $\|A\mathbf{x}_* - \mathbf{b}\|/\|\mathbf{b}\|$ with $\mathbf{x}_*$ the computed solution.

7.1. IFISS experiment. Saddle-point problems arising from the discretized Navier–Stokes equation are challenging problems; several special-purpose block preconditioners have been proposed for these problems [6]. In this section, we examine whether IMF and the ITSOL preconditioners perform well on these unsymmetric indefinite ill-conditioned problems. The Navier–Stokes problem we consider here allows us to vary the indefiniteness and ill-conditionedness through a single parameter: the Reynolds number. As an example, consider Figure 7.1, which displays the spectrum of a saddle-point problem arising from the numerical solution of the problem described

[11] An ARMS2 factorization with a high number of levels often has smaller blocks on the block diagonals, decreasing the throughput and increasing the overhead associated with processing levels.

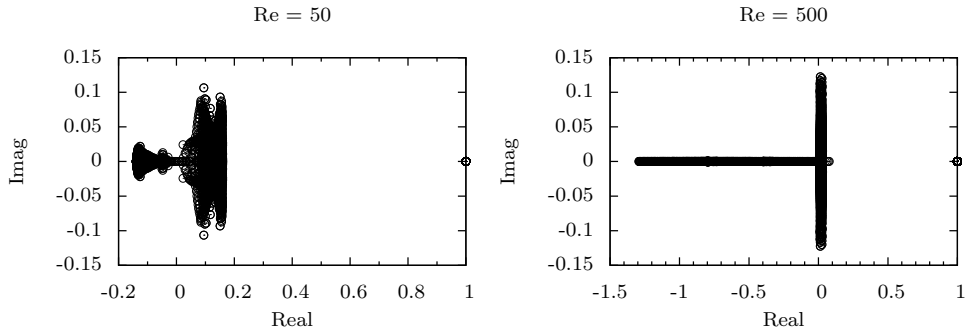


FIG. 7.1. *Distribution of the eigenvalues of a saddle-point problem arising in the numerical solution of the flow around an obstacle problem at* Re = 50 *and* Re = 500.

in the next paragraph.[12] We observe that by increasing the Reynolds number the eigenvalues in the left half-plane are spread out more towards negative infinity, while those in the right half-plane are pushed towards the imaginary axis. This can seriously hamper the convergence of Krylov methods [6] and cause instabilities in incomplete factorization preconditioners [6, 8].

We investigate the performance of the preconditioners on a typical steady-state fluid flow scenario using IFISS [15]—a MATLAB software package that provides several typical fluid flow configurations, various discretization schemes, and a nonlinear Navier–Stokes solver. We selected problem NS5 [39, sect. A], which models the flow of a pressure-driven incompressible Newtonian fluid around a square obstacle in a rectangular channel. The domain Ω is $[0, 8] \times [-1, 1]$, with the obstacle at $[7/4, 9/4] \times [-1/4, 1/4]$ removed. The Navier–Stokes equation

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

is specified on this domain. The parameter ν is the kinematic viscosity, $\mathbf{u}$ represents the flow velocity, and p is the pressure. The Dirichlet boundary condition $\mathbf{u} = 0$ is imposed on the boundaries of the obstacle 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. The domain is discretized with 258,048 pressure-stabilized Q_1-Q_1 elements;[13] the element matrices are 12×12. The nonlinear system of equations representing the steady-state flow problem is solved iteratively by solving a linear saddle-point problem in every step. We solve the unsymmetric indefinite 778,368 $\times$ 778,368 matrix, which has 20,452,690 nonzeros, that arises in the second Newton step following 25 Picard steps. For more details, see [16, 39].

The results of our experiments are shown in Table 7.1. The ILU preconditioner is not included, because it encountered pivot breakdowns in the $(1, 1)$ block on the first level. Partial pivoting within the frontal matrices, as in IMF, is thus necessary for avoiding breakdowns in these problems.

IMF consistently delivers a relative residual that is one order better than requested. This is due to a difference in the stopping criterion in IMF-red. We solve

[12] This problem arises in the first Newton step following 3 Picard steps of the NS5 problem, discretized by 252 pressure-stabilized Q_1-Q_1 elements.

[13] The dofs of these elements consist of the pressure and velocity unknowns at each physical node.

TABLE 7.1

Comparison of the preconditioners on the flow around an obstacle problem. The parameter is κ for IMF, and τ for ARMS2 and ILUTP. The Info column provides additional information about the factorization phase: † indicates a pivot breakdown, ⋆ indicates that pivoting was triggered, and ⋄ denotes that infinity or NaN appeared in the final approximation. A - indicates that the iterative method did not converge within 10,000 *matrix-vector products.*

Re	Precond.	Parameter	ϕ	T_f	T_s	υ	ρ	Info
1,000	IMF-red	0	1.02	9.93	319.	3282	$5.3 \cdot 10^{-12}$	
		1	2.04	19.3	143.	1121	$2.2 \cdot 10^{-12}$	
	ARMS2	$2.3991210 \cdot 10^{-2}$	1.02	22.0	187.	862	$9.0 \cdot 10^{-11}$	
		$8.1816758 \cdot 10^{-3}$	2.03	36.1	111.	426	$8.9 \cdot 10^{-11}$	
	ILUTP	$1.5425586 \cdot 10^{-1}$	1.03	11.1	595.	3980	$1.0 \cdot 10^{-10}$	
		$6.2506309 \cdot 10^{-2}$	2.05	18.6	344.	1772	$9.5 \cdot 10^{-11}$	
2,000	IMF-red	0	1.02	9.90	421.	4426	$3.7 \cdot 10^{-12}$	
		1	2.04	19.2	138.	1097	$2.0 \cdot 10^{-12}$	
	ARMS2	$4.9760180 \cdot 10^{-2}$	1.02	31.3	530.	2390	$9.0 \cdot 10^{-11}$	
		$2.2926859 \cdot 10^{-2}$	2.07	36.1	401.	1594	$9.6 \cdot 10^{-11}$	
	ILUTP	$2.3542015 \cdot 10^{-1}$	1.02	11.3	-	-	$1.3 \cdot 10^{+46}$	
		$1.2801780 \cdot 10^{-1}$	2.00	19.2	-	-	$1.3 \cdot 10^{+55}$	
3,000	IMF-red	0	1.02	9.88	675.	7120	$3.8 \cdot 10^{-12}$	
		1	2.04	19.2	138.	1183	$2.2 \cdot 10^{-12}$	
	ARMS2	$4.400000 \cdot 10^{-2}$	1.01	19.4	683.	2776	$8.0 \cdot 10^{-11}$	
		$1.870000 \cdot 10^{-2}$	2.02	37.5	407.	1338	$9.5 \cdot 10^{-11}$	
	ILUTP	$2.890625 \cdot 10^{-1}$	1.02	11.5	-	-	$2.3 \cdot 10^{+39}$	⋆
		$2.513915 \cdot 10^{-1}$	2.03	57.2	-	-	-	⋆, †, ⋄

the reduced system, rather than the original system, to a relative tolerance of 10^{-10}; i.e., we measure the progress of $\|A_{\kappa+1}\mathbf{x}_{\kappa+1}^{(i)} - \mathbf{z}_{\kappa+1}\|$ rather than $\|A\mathbf{x}^{(i)} - \mathbf{b}\|$, where the superscript (i) denotes the iteration number. It is feasible to compute the correct residual by performing the block backward substitution in every iteration; however, this negates the computational advantage the reduced approach has over the full system approach. Therefore, we solve the reduced system to the desired accuracy and afterwards compute ρ to ensure the residuals of IMF-red are directly comparable with those of the other preconditioners. A complete analysis of the numerical stability of the reduced system approach is beyond the scope of the current paper.

ILUTP produces unstable factors[14] for Re = 2,000, causing IDR(8) to diverge. Increasing the Reynolds number to 3,000 furthermore triggers excessive pivot searches that are responsible for a stark increase of the factorization time. Notwithstanding pivoting, breakdowns still occur; however, in the ITSOL implementation of ILUTP, these zero pivots are replaced by a small value, allowing the factorization to continue.

ARMS2 outperforms IMF in number of iterations if the fill-in factor is 1.02. IMF is nonetheless competitive with ARMS2, because of its increased throughput and fewer operations per iteration; for Re = 2,000, for instance, ARMS2 performs only 4.5 preconditioned iterations per second, whereas IMF attains 10.5 iterations per second. Some parameter tuning was necessary to solve the system with ARMS2 for Re = 3,000. With the default parameters, the matrix at the final level was too large, and its ILUTP factorization produced unstable factors, causing divergence. To alleviate this, we set $\alpha = 0.15$ and $q = 35$.

IMF was competitive with the threshold-based preconditioners and even outperformed them for high values of the Reynolds number. In particular, we note its

[14] We will loosely refer to the problem of unstable triangular solves caused by poorly conditioned L and U factors [8] as "unstable factors" or "unstable solves."

TABLE 7.2
University of Florida Sparse Matrix Collection test matrices.

Group	Name	n	# elements	# nonzeros
Janna	Emilia_923	923,136	1,801,926	40,785,928
	Fault_639	638,802	1,210,902	27,250,120
	Hook_1498	1,498,023	2,578,916	59,383,287
DNVS	M_T1	97,578	5,328	9,753,570
	SHIP_001	34,920	3,431	3,940,718
	SHIP_003	121,728	45,464	4,052,002
	SHIPSEC1	140,874	41,037	3,905,312
	SHIPSEC5	179,860	52,272	5,256,616
	SHIPSEC8	114,919	35,280	3,560,205
	THREAD	29,736	2,176	4,447,216
	X104	108,384	6,019	8,722,160

competitive factorization time, solve time, and number of iterations. In contrast to ILUTP, IMF did not produce unstable factors at higher Reynolds numbers, nor did pivot breakdowns occur as in ILU and ILUTP. Both ARMS2 and IMF were sufficiently robust to solve the problem for all Reynolds numbers.

7.2. University of Florida Sparse Matrix Collection experiments. The next set of matrices can be found in the University of Florida Sparse Matrix Collection [11] in assembled format. Janna (see [23, 24]) kindly provided the element structures of three of the matrices found in the Janna group, while Hogg and Scott [22] recently made the matrices from the DNVS group available in element format.[15] All of these matrices arise from structural engineering problems; they are symmetric and positive definite.[16] The other properties of these matrices are summarized in Table 7.2. The results of our experiments appear in Tables 7.3 and 7.4; they provide an overview of the most important findings. In the remainder, a detailed analysis of these results is performed, explaining the various difficulties we encountered in solving these problems, and techniques to overcome them. This material may be skimmed on a first reading.

In Table 7.3, the results of experiments with the matrices from the Janna group are displayed. The results of ILU are not included because it encountered zero pivots. Emilia_923 was difficult to solve using the Schur complement–based preconditioners: IMF converges only after many iterations, and ARMS2 could not solve this problem.

Fault_639 could not be solved by IMF-red. IDR converged on the reduced system, but the final residual did not satisfy the convergence criterion $\rho < 10^{-10}$. Increasing the accuracy with which the reduced system was solved did not improve the final residual, indicating that the block backward substitution involved ill-conditioned block diagonals. Therefore, we present the results of IMF-full; it is the only preconditioner solving the system at $\phi \approx 1.00$, as the threshold-based preconditioners encountered unstable pivots during the factorization. At higher fill-in factors, however, ILUTP clearly outperformed IMF. ARMS2 could not solve this problem with the parameter settings we tested; we present results with $\alpha = 0.3$ and $q = 100$. Even with fill-in factors up to 10.0, much pivoting was required on the final level, and the resulting factors proved unstable.

Hook_1478 could be solved satisfactorily by all preconditioners; they require a similar number of iterations for $\phi \approx 1.00$. IMF requires the smallest solve time, due

[15] They can be obtained from ftp://ftp.numerical.rl.ac.uk/pub/matrices/element/.
[16] This information is not exploited by the preconditioners.

TABLE 7.3

Comparison of the preconditioners on the problems from the Janna group. The parameter is κ for IMF, and τ for ARMS2 and ILUTP. The Info column provides extra information about the factorization phase: † indicates a pivot breakdown, $\star$ indicates that pivoting was triggered, and $\diamond$ denotes that infinity or NaN appeared in the final approximation. A - indicates that the iterative method did not converge within 10,000 *matrix-vector products.*

Problem	Precond.	Parameter	ϕ	T_f	T_s	v	ρ	Info
Emilia_923	IMF-red	0	1.00	71.0	1850.	7796	$5.5 \cdot 10^{-11}$	
		1	2.67	120.	1710.	4706	$1.9 \cdot 10^{-11}$	
	ARMS2	$2.00000 \cdot 10^{-2}$	1.00	145.	-	-	$6.2 \cdot 10^{-01}$	$\star$
		$3.80000 \cdot 10^{-3}$	2.64	224.	-	-	$5.0 \cdot 10^{-03}$	$\star$
	ILUTP	$8.51376 \cdot 10^{-3}$	0.99	108.	47.7	200	$1.0 \cdot 10^{-10}$	
		$8.11484 \cdot 10^{-4}$	2.68	273.	26.7	78	$9.5 \cdot 10^{-11}$	
Fault_639	IMF-full	0	1.01	48.5	386.	2391	$8.6 \cdot 10^{-11}$	
		1	2.61	79.8	1800.	7476	$7.0 \cdot 10^{-11}$	
	ARMS2	$3.25000 \cdot 10^{-2}$	1.01	58.6	-	-	-	$\star, \dagger, \diamond$
		$6.00000 \cdot 10^{-3}$	2.61	96.3	-	-	$1.7 \cdot 10^{+01}$	$\star$
	ILUTP	$2.97852 \cdot 10^{-2}$	1.00	36.0	-	-	$3.6 \cdot 10^{+06}$	$\star$
		$4.75640 \cdot 10^{-3}$	2.62	88.4	117.	492	$9.8 \cdot 10^{-11}$	$\star$
Hook_1478	IMF-red	0	1.00	120.	570.	1438	$1.7 \cdot 10^{-11}$	
		1	2.38	175.	683.	1250	$1.1 \cdot 10^{-11}$	
	ARMS2	$3.90000 \cdot 10^{-2}$	1.00	124.	1020.	1758	$8.8 \cdot 10^{-11}$	
		$5.99900 \cdot 10^{-3}$	2.42	167.	340.	460	$9.6 \cdot 10^{-11}$	
	ILUTP	$6.28000 \cdot 10^{-2}$	1.01	92.7	619.	1240	$9.5 \cdot 10^{-11}$	
		$1.37000 \cdot 10^{-2}$	2.34	141.	384.	584	$8.8 \cdot 10^{-11}$	

to its higher throughput and fewer iterations. At higher fill-in factors, the threshold-based preconditioners produce better factorizations, requiring fewer iterations than IMF.

The results of our experiments with the matrices from the DNVS group are displayed in Table 7.4. SHIP_001 and THREAD are not included, because none of the preconditioners could solve them.

For SHIP_003, ILUTP encountered many small pivots, thus producing unstable factors. ARMS2 could solve SHIP_003 only at fill-in factors higher than those reported in the table. At the fill-in values displayed, the factorization was too inaccurate and required more than 1,250 IDR outer iterations to converge to the required relative residual. IMF had no difficulties in solving this problem. The fill-in factor of IMF(0) is not near 1.00 because the element structure contains many actual zeros, while our definition of fill-in divides the actual number of nonzeros of the preconditioner by that of the system matrix. This remark also applies to the SHIPSEC matrices.

SHIPSEC1 appeared to be more difficult to solve than SHIP_003. ILUTP's dropping strategy failed to generate a sufficient amount of fill-in, leading to several pivot breakdowns. An additional problem was the excessive pivoting, as more than 45,000 pivot searches were activated in each ILUTP displayed in Table 7.4. The resulting lower and upper triangular factors were very unstable and caused the iterative method to diverge. We were unable to determine suitable parameter settings so that ARMS2 could solve the system, even though there are no apparent reasons for its failure to converge. The rate of coarsening, i.e., the decrease of the size of the $(2, 2)$ block at every level, was very good, and only modest pivoting—less than 150 pivot searches—was required at the final level. Increasing the amount of fill-in did not improve the rate of convergence. IMF, on the other hand, solved this system without difficulties, converging in a few iterations and at low fill-in factors. The behavior of the preconditioners was similar on the other SHIPSEC matrices.

TABLE 7.4

Comparison of the preconditioners on the problems from the DNVS group. The parameter is κ for IMF, and τ for ARMS2 and ILUTP. The Info column provides extra information about the factorization phase: † indicates a pivot breakdown, ⋆ indicates that pivoting was triggered, and ⋄ denotes that infinity or NaN appeared in the final approximation. A - indicates that the iterative method did not converge within 10,000 *matrix-vector products.*

Problem	Precond.	Parameter	ϕ	T_f	T_s	v	ρ	Info
SHIP_003	IMF-red	0	1.41	3.19	40.0	1521	$1.8 \cdot 10^{-11}$	
		1	2.96	6.02	56.6	1812	$8.4 \cdot 10^{-12}$	
	ARMS2	$8.50000 \cdot 10^{-3}$	1.40	4.79	-	-	$5.0 \cdot 10^{-10}$	⋆
		$1.81800 \cdot 10^{-3}$	2.92	8.65	-	-	$4.6 \cdot 10^{-10}$	⋆
	ILUTP	$1.47398 \cdot 10^{-1}$	1.48	7.94	-	-	$7.5 \cdot 10^{+01}$	⋆, †
		$1.14978 \cdot 10^{-1}$	2.97	22.2	-	-	$1.3 \cdot 10^{+04}$	⋆, †
SHIPSEC1	IMF-red	0	1.35	2.69	19.2	817	$1.6 \cdot 10^{-12}$	
		1	2.61	4.67	10.7	417	$3.0 \cdot 10^{-12}$	
	ARMS2	$7.12500 \cdot 10^{-3}$	1.34	4.50	-	-	$1.1 \cdot 10^{-06}$	⋆
		$2.51600 \cdot 10^{-3}$	2.64	8.30	-	-	$9.6 \cdot 10^{-04}$	⋆
	ILUTP	$2.12891 \cdot 10^{-1}$	1.36	4.60	-	-	$1.1 \cdot 10^{+09}$	⋆, †
		$1.89304 \cdot 10^{-1}$	2.60	11.4	-	-	-	⋆, †, ⋄
SHIPSEC5	IMF-red	0	1.33	3.41	17.3	631	$2.2 \cdot 10^{-12}$	
		1	2.67	6.42	13.7	355	$2.1 \cdot 10^{-12}$	
	ARMS2	$7.88000 \cdot 10^{-3}$	1.33	6.22	-	-	$6.3 \cdot 10^{-05}$	⋆
		$2.10000 \cdot 10^{-3}$	2.67	11.3	-	-	$2.2 \cdot 10^{-04}$	⋆
	ILUTP	$2.12600 \cdot 10^{-1}$	1.32	6.21	-	-	$2.1 \cdot 10^{+16}$	⋆, †
		$1.88800 \cdot 10^{-1}$	2.68	16.9	-	-	-	⋆, †, ⋄
SHIPSEC8	IMF-red	0	1.35	2.50	20.0	878	$1.6 \cdot 10^{-11}$	
		1	2.54	4.27	11.6	443	$6.5 \cdot 10^{-11}$	
	ARMS2	$8.45000 \cdot 10^{-3}$	1.35	4.15	-	-	$5.8 \cdot 10^{-05}$	⋆
		$2.71491 \cdot 10^{-3}$	2.54	7.26	-	-	$2.1 \cdot 10^{-05}$	⋆
	ILUTP	$2.39701 \cdot 10^{-1}$	1.35	4.37	-	-	$5.7 \cdot 10^{+02}$	⋆, †
		$2.19640 \cdot 10^{-1}$	2.55	11.0	-	-	$4.3 \cdot 10^{+02}$	⋆, †
M_T1	IMF-full	1	2.32	8.62	63.5	1141	$9.7 \cdot 10^{-11}$	
		2	3.95	29.0	44.2	568	$8.2 \cdot 10^{-11}$	
	ARMS2	$3.38979 \cdot 10^{-1}$	2.34	42.7	-	-	-	⋆, †, ⋄
		$3.25000 \cdot 10^{-1}$	3.90	134.	-	-	-	⋆, †, ⋄
	ILUTP	$3.72500 \cdot 10^{-1}$	2.30	41.4	-	-	-	⋆, †, ⋄
		$2.76000 \cdot 10^{-1}$	3.96	117.	-	-	-	⋆, †, ⋄
X104	IMF-full	3	6.02	53.0	308.	3251	$8.6 \cdot 10^{-09}$	
		4	8.54	98.4	217.	1706	$7.9 \cdot 10^{-09}$	
	ARMS2	$4.79300 \cdot 10^{-3}$	6.12	227.	-	-	-	⋆, ⋄
		$4.30000 \cdot 10^{-3}$	8.55	475.	-	-	$3.5 \cdot 10^{+00}$	⋆
	ILUTP	$2.32000 \cdot 10^{-1}$	6.03	133.	-	-	-	⋆, †, ⋄
		$8.72000 \cdot 10^{-2}$	8.56	281.	-	-	-	⋆, †, ⋄

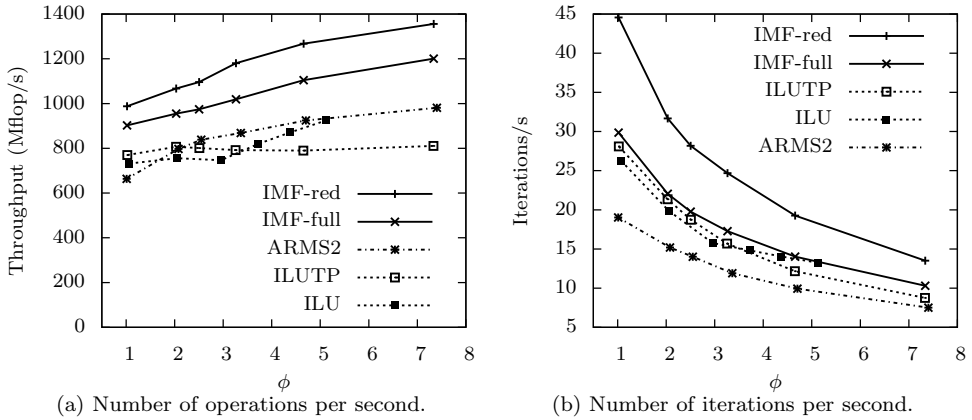
M_T1 and X104 are two challenging problems for which numerical pivoting proved crucial to their solution. These are the only problems in Table 7.4 for which the direct solver MUMPS v4.9.2 [1] required off-diagonal pivots and had to delay the elimination of several unknowns. We were unable to solve X104 to a relative tolerance of 10^{-10} with the preconditioners; the stopping criterion was relaxed to $\rho < 10^{-08}$. IMF-red failed on both M_T1 and X104; while it converged to the required tolerance on the reduced system, it did not satisfy the tolerance on the final residual. Increasing the accuracy to which the reduced system was solved did not alleviate the problem. However, solving the system using IMF-full allows the Krylov method to deal with the instability of the preconditioner, which proved to be sufficient to guarantee convergence. We thus display the results of IMF-full for these problems in Table 7.4. IMF is the only preconditioner to successfully solve these problems. ILUTP's dropping

strategy was unsuccessful on M_T1. It required excessive pivoting (more than 20,000 pivot searches) and generated insufficient fill-in to prevent breakdowns. As a result, the triangular solves are extremely unstable, and IDR diverged in the first iteration. Precisely the same pattern was observed for X104. The difficulties of ARMS2 on M_T1 and X104 were obvious from its slow coarsening during the factorization phase; the number of unknowns after $q = 25$ reductions was still more than 22%, respectively, 9%, of the total unknowns. For good performance, the size of the final system should be no larger than a few thousand unknowns. The main parameter controlling the rate of coarsening in ARMS2 is the diagonal dominance criterion α. Using a small α and a small fill-in factor, the rate of coarsening is satisfactory. However, these settings resulted in an enormous number of breakdowns in the final ILUTP factorization; i.e., the final system was almost entirely zero. Increasing the fill-in factor resolves this problem but introduces another: the factorization proceeds extremely slowly. The reason is that the generated fill-in values are large in magnitude and cause instabilities; the heuristic algorithm that constructs the permutation vector cannot find suitable pivots, even with these small values of α. Consequently, the $(1, 1)$ block often consists of only a few tens of unknowns, thus requiring hundreds of levels. The factorization time rises accordingly. However, the solves are still not stable, and the iterative method diverges to infinity after a few iterations. With $\alpha = 0.5$, which improves the diagonal dominance within the $(1, 1)$ block and should avoid small pivots, we encountered breakdowns in this matrix for some values of τ. In other cases, breakdowns occurred in the final system, again causing unstable solves. Our experiments with ARMS-C indicated similar unresolved issues.

The DNVS problems illustrate some of the advantages of IMF(0) over ILU(0). The latter was unable to solve these problems: on M_T1 and X104, it encountered zero pivots, and for the remainder of the problems, some of the values in the upper or lower triangular matrix were extremely large in magnitude. IMF(0), on the other hand, solves most problems satisfactorily. Because of its pivoting within frontal matrices, no breakdowns were observed for M_T1 and X104. The results suggest that IMF's approximation of the Schur complement is also more stable than ILU's, because IMF did not produce values of very large magnitude. Another advantage of IMF over ILU is property (4.3). ILU eliminates unknowns in a pointwise manner and thus drops values in $L_k D_k^{-1}$. This approximation error is precisely the reason why an ILU(0) with the same ordering as IMF(0) does not obey property (4.3).

7.3. Computational performance. We evaluate the throughput of the solve phase of the preconditioners. The implementation of the ITSOL preconditioners was modified such that optimized OSKI [45] sparse matrix-vector product (SpMV) and sparse triangular solve (SpTrS) kernels were employed. We always let OSKI optimize the data structures, indicating that the matrix is structurally symmetric and contains multiple dense blocks. This modification greatly boosted performance.

Two measures of throughput are studied: the number of floating-point operations, respectively, iterations, per second that can be performed by the preconditioned IDR method. The former includes the application of A to a vector, the application of the preconditioner M, and the $8 + \frac{1}{4}$ dot products and $16 + \frac{2}{9}$ vector updates per iteration required by IDR(8) [43]. We measured the performance on the regularized lid-driven cavity problem from IFISS [39, sect. A] with Reynolds number 1,000, which was discretized by 65,536 elements and resulted in a 198,147 $\times$ 198,147 matrix with 5,168,125 nonzeros. We selected this problem because none of the preconditioners experienced breakdown or unstable solves. The results are presented in Figure 7.2.



(a) Number of operations per second. (b) Number of iterations per second.

FIG. 7.2. *Two measures of the throughput of the solve phase.*

Consider Figure 7.2(a), where three trends are obvious: (1) the throughput increases with the fill-in factor; (2) IMF outperforms the other preconditioners; and (3) the reduced system approach attains higher throughput than the full system approach.

We observed that the throughput of computing an SpMV with A was typically lower than that of applying M to a vector. The optimization algorithm in OSKI decided that blocking was not profitable for A; it selected a CSR data structure. The increase of throughput for ARMS2 and ILU can be attributed to OSKI's capability to select more appropriate data structures for the preconditioner. The case of ILU is very illustrative; no gain in throughput is observed until the fill-in factor surpasses 3.5, when OSKI switches to a block compressed sparse row (BCSR) data structure. Using BCSR, the throughput of the SpTrSs rapidly increases with the fill-in factor. In the case of ARMS2, the throughput increases mainly because the matrices on the block diagonal are relatively dense, and OSKI immediately selects an optimized BCSR data structure. This is particularly profitable for ARMS2, as a significant number of nonzeros appear on the block diagonal matrices, and because the latter are applied twice every iteration. The throughput of ILUTP clearly does not increase; OSKI stores the preconditioner using CSR storage, presumably because the sparsity pattern was unsuitable for OSKI's optimizations. In IMF, finally, a permutation is determined such that a large number of nonzero values are stored in a dense block diagonal matrix—25%–31% in this problem. The throughput of applying these block diagonal matrices to a vector was determined to be higher than the throughput of the other SpMVs involved in applying IMF and the system matrix.

The difference in throughput between IMF-red and IMF-full can be partially explained by the difference in the fraction of the total number of operations that is spent on an SpMV with A. In IMF-red(0), for instance, only 26.27% of the total number of operations is spent on the SpMV with A, whereas the full approach spends 47.18% of the total number of operations on this SpMV. Another aspect of the difference concerns the sparsity pattern of the block diagonal matrices. In IMF-red(κ), the first κ block diagonal matrices are applied only once. However, the matrices on these first few levels are typically smaller than the matrices on the block diagonal matrix of later levels, because such large element matrices can arise only by eliminating smaller elements. The reduced approach thus spends relatively more operations on applying

TABLE 7.5

The number of operations, in millions, required to apply the preconditioner and matrix to a vector on the lid-driven cavity problem with $\mathrm{Re} = 1{,}000$. *The bar indicates that no data was available for a fill-in factor comparable to the other preconditioners.*

	ϕ	Mflop	ϕ	Mflop	ϕ	Mflop	ϕ	Mflop	ϕ	Mflop
IMF-red	1.02	15.70	2.03	27.22	2.50	32.43	3.26	41.34	4.66	59.32
ILU	1.07	21.38	2.06	31.59	-	-	-	-	-	-
ILUTP	1.02	20.92	2.03	31.30	2.51	36.29	3.25	43.98	4.65	58.41
IMF-full	1.02	23.76	2.03	36.83	2.50	42.83	3.26	52.54	4.66	72.27
ARMS2	1.01	28.39	2.09	45.98	2.55	53.34	3.36	66.44	4.70	86.60

larger dense matrices, which can be applied by the BLAS2 `dgemv` routine with higher throughput than that of very small matrices.

In Table 7.5, the total number of operations to apply both A and M to a vector is displayed for various fill-in factors. In the case of ILU and ILUTP, the number of operations is simply $2 \cdot (1 + \phi) \cdot \mathrm{nnz}(A)$; for ARMS2 and IMF-full, an additional $2 \cdot \mathrm{nnz}(D)$ operations should be added, where $\mathrm{nnz}(D)$ represents the total number of nonzeros on the block diagonal matrices. The reduced system approach for IMF(κ) can improve thereupon, because the first κ block diagonal matrices should be applied only once, thus subtracting this number of operations from the operation count of IMF-full. Additionally, only a part of the system matrix should be applied in every iteration, namely $SPAS^TP^T$, thereby further reducing the operation count. As can be seen in Table 7.5, these modifications reduce the number of operations considerably; IMF-red requires 18%–34% fewer operations than IMF-full.

The combined effect of Table 7.5 and Figure 7.2(a) is displayed in Figure 7.2(b), where we plot the number of preconditioned IDR(8) iterations that can be performed per second. Comparing IMF-red, IMF-full, and ARMS2, the benefits of using an element structure are clear. ARMS2 attains a throughput similar to ILU-based preconditioners but also requires substantially more operations per iteration; thus, it performs fewer iterations per second. The throughput of IMF-full is higher than that of ARMS2 because the element structure allows IMF to use optimized BLAS2 routines for applying a large fraction of the total number of operations. However, IMF-full requires more operations to apply than ILU and ILUTP with similar amounts of fill-in; as a result, it is competitive with these preconditioners only in terms of the number of iterations per second. IMF-red exploits the fact that the matrices on the block diagonal are dense so that the exact inverse can be computed without fill-in, which can be accomplished efficiently if an element structure is available. This, in turn, allows IMF-red to solve only a reduced system, significantly reducing the total number of operations for performing one iteration. Combined with its elevated throughput, IMF-red is 33%–50% faster than IMF-full, and 70%–130% faster than ARMS2 in terms of the number of preconditioned IDR iterations performed per second.

We observed similar relations between the preconditioners on the problems from section 7.2; the differences in throughput became smaller, however, because the increased connectivity of three-dimensional problems allowed OSKI to select more appropriate blocked data structures, allowing for higher throughput and smaller differences with optimized ATLAS codes. The throughput of the ILUTP preconditioner also increased with the fill-in factor for these problems.

8. Conclusions. The incomplete multifrontal method (IMF) 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 fac-

torizations. An exact factorization is computed on the first few levels; thereafter, a symbolic dropping rule discards all fill-in values.

IMF has practical advantages over related approaches. As in multifrontal direct solvers, the factorization phase can be implemented using high-throughput operations on dense frontal matrices, allowing for efficient numerical pivoting within these matrices. Numerical experiments with matrices from the University of Florida Sparse Matrix Collection confirm the robustness and efficiency of IMF's factorization. The preconditioned system has a block upper triangular structure so that only a reduced system of smaller size has to be solved iteratively. This greatly reduces the cost per iteration with respect to other multilevel preconditioners. Furthermore, numerical experiments indicate that the throughput of the solve phase of IMF is competitive with an OSKI optimized solve phase of ARMS2. Experiments with unsymmetric indefinite ill-conditioned systems arising from saddle-point problems indicate that IMF is competitive with ARMS2, outperforming it at higher fill-in factors.

Despite IMF's favorable properties, some challenges warrant further research. The symbolic dropping rule may produce large amounts of fill-in for highly interconnected element problems. To address this issue, a numerical dropping rule should be investigated. Our experiments suggest that the reduced system approach may not always be applicable if the factorization of the first levels is inaccurate. Techniques should be proposed to moderate these instabilities.

Acknowledgments. The authors would like to thank P. Amestoy for encouraging discussions. K. Vuik is heartily thanked for feedback on our technical report [44]. We thank the editor, H. Elman, for carefully handling this manuscript, and the two anonymous reviewers for their detailed questions and remarks that considerably improved this manuscript.

REFERENCES

[1] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, *A fully asynchronous multifrontal solver using distributed dynamic scheduling*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 15–41.

[2] 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*, 3rd ed., SIAM, Philadelphia, 1999.

[3] O. Axelsson, R. Blaheta, and M. Neytcheva, *Preconditioning of boundary value problems using elementwise Schur complements*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 767–789.

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

[5] O. Axelsson and P. S. Vassilevski, *Algebraic multilevel preconditioning methods,* II, SIAM J. Numer. Anal., 27 (1990), pp. 1569–1590.

[6] M. Benzi, G. Golub, and J. Liesen, *Numerical solution of saddle point problems*, Acta Numer., 14 (2005), pp. 1–137.

[7] Y. E. Campbell and T. A. Davis, *Incomplete LU Factorization: A Multifrontal Approach*, Technical Report TR-95-024, Computer and Information Sciences Department, University of Florida, Gainesville, FL, 1995.

[8] E. Chow and Y. Saad, *Experimental study of ILU preconditioners for indefinite matrices*, J. Comput. Appl. Math., 86 (1997), pp. 387–414.

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

[10] T. A. Davis and I. S. Duff, *An unsymmetric-pattern multifrontal method for sparse LU factorization*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 140–158.

[11] T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Trans. Math. Software, 38 (2011), 1.

[12] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[13] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. S. Duff, *A set of level* 3 *basic linear algebra subprograms*, ACM Trans. Math. Software, 16 (1990), pp. 1–17.
[14] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear systems*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.
[15] H. C. Elman, A. Ramage, and D. J. Silvester, *Algorithm* 866: *IFISS, a Matlab toolbox for modelling incompressible flow*, ACM Trans. Math. Software, 33 (2007), 14.
[16] 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.
[17] A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall Series in Computational Mathematics, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[18] A. George and J. W. H. Liu, *The evolution of the minimum degree ordering algorithm*, SIAM Rev., 31 (1989), pp. 1–19.
[19] *GLAS*, *Generic Linear Algebra Software*, https://www.cs.kuleuven.be/~karlm/glas (2005).
[20] M. T. Heath, E. Ng, and B. W. Peyton, *Parallel algorithms for sparse linear systems*, SIAM Rev., 33 (1991), pp. 420–460.
[21] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002.
[22] J. D. Hogg and J. A. Scott, *An efficient analyse phase for element problems*, Numer. Linear Algebra Appl., to appear.
[23] C. Janna and M. Ferronato, *Adaptive pattern research for block FSAI preconditioning*, SIAM J. Sci. Comput., 33 (2011), pp. 3357–3380.
[24] C. Janna, M. Ferronato, and G. Gambolati, *A block FSAI-ILU parallel preconditioner for symmetric positive definite linear systems*, SIAM J. Sci. Comput., 32 (2010), pp. 2468–2484.
[25] J. E. Jones and P. S. Vassilevski, *AMGe based on element agglomeration*, SIAM J. Sci. Comput., 23 (2001), pp. 109–133.
[26] J. K. Kraus, *Algebraic multilevel preconditioning of finite element matrices using local Schur complements*, Numer. Linear Algebra Appl., 13 (2006), pp. 49–70.
[27] J. W. H. Liu, *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM Rev., 34 (1992), pp. 82–109.
[28] 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*, Math. Comp., 31 (1977), pp. 148–162.
[29] Y. Notay, *Robust parameter free algebraic multilevel preconditioning*, Numer. Linear Algebra Appl., 9 (2002), pp. 409–428.
[30] Y. Notay, *Algebraic multigrid and algebraic multilevel methods: A theoretical comparison*, Numer. Linear Algebra Appl., 12 (2005), pp. 522–543.
[31] Y. Notay, *Aggregation-based algebraic multilevel preconditioning*, SIAM J. Matrix Anal. Appl., 27 (2006), pp. 998–1018.
[32] Y. Qu and J. Fish, *Multifrontal incomplete factorization for indefinite and complex symmetric systems*, Internat. J. Numer. Methods Engrg., 53 (2002), pp. 1433–1459.
[33] Y. Saad, *ILUT: A dual threshold incomplete ILU factorization*, Numer. Linear Algebra Appl., 1 (1994), pp. 387–402.
[34] Y. Saad, *ILUM: A multi-elimination ILU preconditioner for general sparse matrices*, SIAM J. Sci. Comput., 17 (1996), pp. 830–847.
[35] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.
[36] Y. Saad, *Multilevel ILU with reorderings for diagonal dominance*, SIAM J. Sci. Comput., 27 (2005), pp. 1032–1057.
[37] Y. Saad and B. Suchomel, *ARMS: An algebraic recursive multilevel solver for general sparse linear systems*, Numer. Linear Algebra Appl., 9 (2002), pp. 359–378.
[38] Y. Saad and J. Zhang, *BILUM: Block versions of multielimination and multilevel ILU preconditioner for general sparse linear systems*, SIAM J. Sci. Comput., 20 (1999), pp. 2103–2121.
[39] D. J. Silvester, H. C. Elman, and A. Ramage, *Incompressible Flow Iterative Solution Software (IFISS) Installation & Software Guide*, version 3.2, 2012.
[40] P. Sonneveld and M. B. van Gijzen, *IDR(s): A family of simple and fast algorithms for solving large nonsymmetric systems of linear equations*, SIAM J. Sci. Comput., 31 (2008), pp. 1035–1062.
[41] K. Stüben, *A review of algebraic multigrid*, J. Comput. Appl. Math., 128 (2001), pp. 281–309.
[42] W. F. Tinney and J. W. Walker, *Direct solutions of sparse network equations by optimally ordered triangular factorization*, Proc. IEEE, 55 (1967), pp. 1801–1809.

[43] M. B. van Gijzen and P. Sonneveld, *Algorithm* 913: *An elegant* IDR*(s) variant that efficiently exploits biorthogonality properties*, ACM Trans. Math. Software, 38 (2011), 5.

[44] N. Vannieuwenhoven and K. Meerbergen, *IMF: An Incomplete Multifrontal LU-Factorization for Element-Structured Sparse Linear Systems*, Technical Report TW581, KU Leuven, Leuven, Belgium, 2010.

[45] R. Vuduc, J. W. Demmel, and K. A. Yelick, *The Optimized Sparse Kernel Interface (OSKI) Library User's Guide for Version* 1.0.1*h*, Technical report, Berkeley Benchmarking and Optimization Group, University of California, Berkeley, CA, 2007.

[46] 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 Proceedings of the ACM/IEEE Conference on Supercomputing, Baltimore, MD, 2002, pp. 1–35.

[47] 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 Proceedings of the Workshop on Performance Optimization of High-level Languages and Libraries (POHLL) at ACM International Conference on Supercomputing (ICS), New York, 2002.

[48] R. C. Whaley and A. Petitet, *Minimizing development and maintenance costs in supporting persistently optimized BLAS*, Softw. Pract. Exper., 35 (2005), pp. 101–121.

[49] R. C. Whaley, A. Petitet, and J. J. Dongarra, *Automated empirical optimization of software and the ATLAS project*, Parallel Comput., 27 (2001), pp. 3–35.