SIAM J. SCI. COMPUT. © 2010 Society for Industrial and Applied Mathematics
Vol. 32, No. 5, pp. 2468–2484

A BLOCK FSAI-ILU PARALLEL PRECONDITIONER FOR SYMMETRIC POSITIVE DEFINITE LINEAR SYSTEMS[*]

CARLO JANNA[†], MASSIMILANO FERRONATO[†], AND GIUSEPPE GAMBOLATI[†]

Abstract. A novel parallel preconditioner for symmetric positive definite matrices is developed coupling a generalized factored sparse approximate inverse (FSAI) with an incomplete LU (ILU) factorization. The generalized FSAI, called block FSAI, is derived by requiring the preconditioned matrix to resemble a block-diagonal matrix in the sense of the minimal Frobenius norm. An incomplete block Jacobi algorithm is then effectively used to accelerate the convergence of a Krylov subspace method. The block FSAI-ILU preconditioner proves superior to both FSAI and the incomplete block Jacobi by themselves in a number of realistic finite element test cases and is fully scalable for a given number of blocks.

Key words. Krylov subspace methods, preconditioning, factorized sparse approximate inverse, incomplete factorization, parallel computing

AMS subject classifications. 65F10, 65F50, 65Y05

DOI. 10.1137/090779760

1. Introduction.

The efficient parallel solution of sparse linear systems of equations

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

where $A \in \mathbb{R}^{n\times n}$ is the coefficient matrix, $\mathbf{x} \in \mathbb{R}^n$ the unknown vector, and $\mathbf{b} \in \mathbb{R}^n$ the given right-hand side, is a central issue in many numerical computations in science and engineering. Typically, but not exclusively, A arises from the linearized discretization of partial differential equations with n frequently growing up to a few millions, thus making the use of iterative methods and parallel architectures virtually mandatory.

Iterative methods based on Krylov subspaces involve matrix-vector products, dot products, and vector updates only, so they can be, at least in principle, almost ideally implemented on parallel computers. However, the computation and application of an effective preconditioner often is not, and perhaps this is the most decisive effort for the efficient solution to (1.1), depending of course on the nature and structure of the coefficient matrix A. The most popular general-purpose algebraic preconditioners can be roughly subdivided into two classes: (1) incomplete LU (ILU) factorizations and (2) sparse approximate inverses.

ILU-based preconditioners were first introduced in the 1960s, but were exploited in connection to Krylov subspace methods in the late 1970s only [23, 29], and soon extended and improved with the aim of controlling the fill-in degree. For a nice review, see, for instance, [3]. The main drawback, however, of ILU strategies for large sized problems is their difficult parallelization. ILU preconditioners suffer from the

[*]Received by the editors December 10, 2009; accepted for publication (in revised form) June 2, 2010; published electronically August 12, 2010. This work was partially funded by the Italian MIUR project (PRIN) "Advanced numerical methods and models for environmental fluid-dynamics and geomechanics."
http://www.siam.org/journals/sisc/32-5/77976.html

[†]Department of Mathematical Methods and Models for Scientific Applications, University of Padova, via Trieste 63, 35121 Padova, Italy (janna@dmsa.unipd.it, ferronat@dmsa.unipd.it, gambo@dmsa.unipd.it).

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

intrinsic sequentiality of their computation and, most of all, the forward and backward substitutions required by their application. Tricky attempts of parallel ILU implementations, e.g., the level-based scheduling by Anderson and Saad [1] and the method proposed by Hysom and Pothen [20], are generally based on special reorderings of the problem unknowns. For example, the so-called red-black ordering provides satisfactory results for a five-point stencil discretization [22], although a deterioration of the preconditioned conjugate gradient (PCG) convergence can result [12]. Generally speaking, the aim of such reorderings is at subdividing A into a number of possibly overlapping groups of unknowns with a limited communication [35]. The simplest alternative is the one with no overlapping at all, thus generating the incomplete block Jacobi algorithm, which typically exhibits an excellent degree of parallelism but a poor solver rate of convergence.

Sparse approximate inverses are conceptually parallel in nature, as they provide an explicit approximation of A^{-1} which can be applied to a vector by a matrix-by-vector product only [2, 10, 17]. In a parallel computation, however, the algorithm bottleneck can be the approximate inverse setup. Two main approaches are followed: incomplete biorthogonalization and Frobenius norm minimization. Factored approximate inverses can be efficiently constructed by an incomplete Gram–Schmidt procedure, which provides an approximation of the triangular factorization of A^{-1} relying on the A entries only. This is the basis of the so-called AINV algorithm [4, 5]. The incomplete A biorthogonalization used to compute AINV, however, is inherently sequential, and some degree of parallelism can be gained by graph partitioning [6]. By distinction, the approximate inverse P computed by minimizing the Frobenius norm of $(I - AP)$ can be obtained from the solution of n independent least-squares problems subject to some sparsity constraints. In a distributed computing environment, each processor can acquire the matrix coefficients needed to form the local least-squares problems at a preliminary stage. Hence, the corresponding algorithm can be efficiently implemented on a parallel machine because each processor can perform the preconditioner setup with no communication overhead. Quite interestingly, these preconditioners can be viewed as the nullspace of an appropriate linear map between two matrix spaces [19], thus giving rise to a more general formulation of the approximate inverse problem [7]. Amongst the preconditioners belonging to this class, a most effective one for a wide range of problems is the factored sparse approximate inverse (FSAI), which was originally developed for symmetric positive definite (SPD) matrices [24, 25, 26, 37] and later extended to nonsymmetric problems [38].

In the present paper, a novel parallel preconditioner combining FSAI and ILU for SPD matrices is developed. A generalized FSAI, denoted as block FSAI (BFSAI) in what follows, is derived by requiring the preconditioned matrix to resemble as much as possible an SPD block diagonal matrix in the sense of the minimal Frobenius norm. A second preconditioning is then applied using an incomplete block Jacobi strategy, which makes use of an incomplete Cholesky (IC) decomposition with partial controlled fill-in. The BFSAI-IC preconditioner turns out to be a parallel hybrid of FSAI and IC that proves generally superior to FSAI. This is verified with some large sized test matrices arising from the discretization of partial differential equations in mechanics and fluid dynamics. Finally, a few remarks close the paper.

2. The BFSAI-IC algorithm.

In what follows, the following notation is used:

- $[A]_{ij}$ denotes the coefficient in row i and column j of matrix A;
- $\mathbf{v}[\mathcal{I}]$ denotes the subvector of $\mathbf{v}$ made of the components with index belonging to the set $\mathcal{I}$;

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

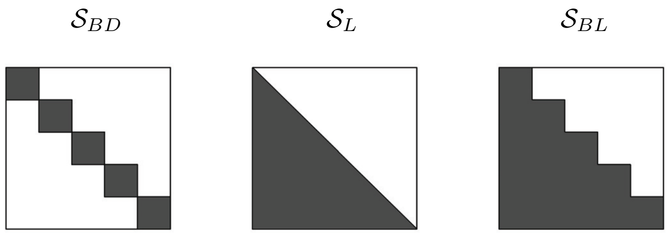

FIG. 2.1. *Schematic representation of the nonzero patterns $\mathcal{S}_{BD}$, $\mathcal{S}_L$, and $\mathcal{S}_{BL}$.*

- $A[\mathcal{I},\mathcal{J}]$ denotes the submatrix of A made of the coefficients $[A]_{ij}$ such that $i \in \mathcal{I}$ and $j \in \mathcal{J}$;
- $A[\mathcal{I},:]$ denotes the submatrix of A made of the rows with index belonging to the set $\mathcal{I}$;
- $A[:,\mathcal{J}]$ denotes the submatrix of A made of the columns with index belonging to the set $\mathcal{J}$;
- $\|A\|_F = \sqrt{\mathrm{Tr}(AA^T)} = \sqrt{\mathrm{Tr}(A^TA)} = \sqrt{\sum_{i,j=1}^n [A]_{ij}^2}$ denotes the Frobenius norm of matrix A.

Let $A \in \mathbb{R}^{n\times n}$ be an SPD matrix. The basic idea of the native FSAI is to find a matrix $G \in \mathcal{W}_{\mathcal{S}_L}$ such that

$$\|I - GL\|_F \to \min, \tag{2.1}$$

where L is the lower triangular factor of A, i.e., $A = LL^T$, and $\mathcal{W}_{\mathcal{S}_L}$ is the set of matrices with a prescribed lower triangular nonzero pattern, i.e., all matrices such that

$$[G]_{ij} = 0 \qquad \forall (i,j) \notin \mathcal{S}_L \tag{2.2}$$

with

$$\{(i,j) : 1 \le i = j \le n\} \subseteq \mathcal{S}_L \subseteq \{(i,j) : 1 \le j \le i \le n\}. \tag{2.3}$$

The matrix G satisfying (2.1) can be obtained by solving a set of n independent linear systems [24].

BFSAI can be viewed as a generalization of FSAI. Let $\mathcal{W}_{\mathcal{S}_{BL}}$ be the set of matrices with an arbitrary nonzero pattern $\mathcal{S}_{BL}$. We look for $F \in \mathcal{W}_{\mathcal{S}_{BL}}$ such that

$$\|D - FL\|_F \to \min, \tag{2.4}$$

where D is an arbitrary full-rank block diagonal matrix. For the sake of simplicity, assume that D consists of n_b diagonal blocks with size $m = n/n_b$. Hence, $D \in \mathcal{W}_{\mathcal{S}_{BD}}$ with

$$\mathcal{S}_{BD} = \{(i,j) : 1 \le i \le n \quad \text{and} \quad m(i_b - 1) + 1 \le j \le m i_b\}, \tag{2.5}$$

where i_b is the integer part of $(i/m + 1)$ and denotes the block index (see Figure 2.1).

Two useful lemmas are now introduced.

LEMMA 2.1. *Let $A, B \in \mathbb{R}^{n\times n}$; then*

$$\frac{\partial}{\partial [A]_{ij}} \mathrm{Tr}(AB) = [B^T]_{ij}. \tag{2.6}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Proof. Write by components the trace and the matrix-by-matrix product:

$$\mathrm{Tr}(AB) = \sum_{k=1}^{n} [AB]_{kk} = \sum_{k=1}^{n} \left(\sum_{l=1}^{n} [A]_{kl}[B]_{lk} \right).$$

In the double summation, $[A]_{ij}$ appears only once multiplying $[B]_{ji}$, so

$$\frac{\partial}{\partial [A]_{ij}} \mathrm{Tr}(AB) = [B]_{ji} = [B^T]_{ij},$$

proving (2.6). ◻

LEMMA 2.2. *Let $A, B \in \mathbb{R}^{n\times n}$ with $B = B^T$; then*

$$\frac{\partial}{\partial [A]_{ij}} \mathrm{Tr}(ABA^T) = 2[AB]_{ij}. \tag{2.7}$$

Proof. Write again by components the trace and the matrix-by-matrix product:

$$\mathrm{Tr}(ABA^T) = \sum_{k=1}^{n} [ABA^T]_{kk} = \sum_{k=1}^{n} \left(\sum_{l=1}^{n} \sum_{m=1}^{n} [A]_{kl}[B]_{lm}[A^T]_{mk} \right).$$

Recalling that $[A^T]_{mk} = [A]_{km}$, $[A]_{ij}$ appears twice in the triple summation,

$$\frac{\partial}{\partial [A]_{ij}} \mathrm{Tr}(ABA^T) = \sum_{m=1}^{n} [B]_{jm}[A]_{im} + \sum_{l=1}^{n} [A]_{il}[B]_{lj}.$$

Because of the symmetry of B, $[B]_{jm} = [B]_{mj}$; hence

$$\sum_{m=1}^{n} [B]_{jm}[A]_{im} + \sum_{l=1}^{n} [A]_{il}[B]_{lj} = \sum_{m=1}^{n} [A]_{im}[B]_{mj} + \sum_{l=1}^{n} [A]_{il}[B]_{lj} = 2[AB]_{ij},$$

proving (2.7). ◻

Matrix F is found by differentiating the square of the left-hand side of (2.4) and setting to zero:

$$\begin{aligned} \frac{\partial}{\partial [F]_{ij}} \|D - FL\|_F^2 &= \frac{\partial}{\partial [F]_{ij}} \left[\mathrm{Tr}(DD^T) - 2\mathrm{Tr}(FLD^T) + \mathrm{Tr}(FAF^T) \right] \\ &= 0 \qquad \forall (i,j) \in \mathcal{S}_{BL}. \end{aligned} \tag{2.8}$$

Noting that $\mathrm{Tr}(DD^T)$ does not depend on F and applying Lemmas 2.1 and 2.2 yields

$$[FA]_{ij} = [DL^T]_{ij} \qquad \forall (i,j) \in \mathcal{S}_{BL}. \tag{2.9}$$

If $D \equiv I$ and $\mathcal{S}_{BL} \equiv \mathcal{S}_L$, then (2.9) reduces to the equation used to compute G; hence $F \equiv G$.

Set $\mathcal{S}_{BL} = \mathcal{S}_L \cup \mathcal{S}_{BD}$ (see Figure 2.1). As DL^T is block upper triangular, the vector $\mathbf{f}_i$ of the nz_i nonzeros of the ith row of F is the solution of the dense linear system

$$A[\mathcal{J}_i, \mathcal{J}_i]\mathbf{f}_i = \mathbf{v}, \tag{2.10}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

where $\mathcal{J}_i = \{k : (i,k) \in \mathcal{S}_{BL}\}$ and $\mathbf{v}$ is a vector with the first $(nz_i - m)$ components zero and the last m equal to $[DL^T]_{ij}$ with $m(i_b - 1) + 1 \leq j \leq mi_b$. However, as D is arbitrary, the coefficients of DL^T, and hence $\mathbf{v}$, are also. The system (2.10) can be rewritten in a block form by defining the set

$$\overline{\mathcal{P}}_i = \{k : (i,k) \in \mathcal{S}_{BD}\} \tag{2.11}$$

and its complement set

$$\mathcal{P}_i = \mathcal{J}_i \setminus \overline{\mathcal{P}}_i \tag{2.12}$$

and separating accordingly the unknowns of $\mathbf{f}_i$:

$$\begin{bmatrix} A[\mathcal{P}_i, \mathcal{P}_i] & A[\mathcal{P}_i, \overline{\mathcal{P}}_i] \\ A[\overline{\mathcal{P}}_i, \mathcal{P}_i] & A[\overline{\mathcal{P}}_i, \overline{\mathcal{P}}_i] \end{bmatrix} \begin{Bmatrix} \mathbf{f}_{i1} \\ \mathbf{f}_{i2} \end{Bmatrix} = \begin{Bmatrix} \mathbf{0} \\ \mathbf{v}_2 \end{Bmatrix}. \tag{2.13}$$

Since the vector $\mathbf{v}_2 \in \mathbb{R}^m$ is arbitrary but certainly not null because of the nonsingularity of D, we can set arbitrarily m components of $\mathbf{f}_i$. We choose to take $[F]_{ii}$ equal to 1 and all the remaining components of $\mathbf{f}_{i2}$ equal to 0. This choice corresponds to requiring F to be a lower block triangular matrix in the form

$$\begin{bmatrix} I & 0 & \cdots & 0 \\ F_{21} & I & & \vdots \\ \vdots & & \ddots & \vdots \\ F_{n_b1} & \cdots & \cdots & I \end{bmatrix}; \tag{2.14}$$

i.e., it actually reduces to a unit lower triangular matrix. To grasp the rationale underlying such a choice, consider a full-rank 4-block lower triangular matrix:

$$M = \begin{bmatrix} A & 0 \\ B & C \end{bmatrix}. \tag{2.15}$$

We look for a lower triangular block matrix H such that

$$HM = \begin{bmatrix} X & 0 \\ Y & Z \end{bmatrix} \begin{bmatrix} A & 0 \\ B & C \end{bmatrix} = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \tag{2.16}$$

with D_1 and D_2 nonsingular matrices. The condition (2.16) generates the equations

$$XA = D_1, \tag{2.17}$$
$$ZC = D_2, \tag{2.18}$$
$$YA + ZB = 0 \Rightarrow Y = -ZBA^{-1}. \tag{2.19}$$

The extradiagonal block Y is generally dense. By contrast, as D_1 and D_2 are arbitrary, X and Z are also, with the only constraint that they be nonsingular. The simplest and cheapest option is to set $X = Z = I$ with $Y = -BA^{-1}$. The same argument can be easily extended by induction to matrices with a larger number of blocks.

In this way, system (2.13) reduces to

$$A[\mathcal{P}_i, \mathcal{P}_i]\mathbf{f}_{i1} = -A[\mathcal{P}_i, i], \tag{2.20}$$

where $A[\mathcal{P}_i, i]$ is the $(nz_i - m) \times 1$ matrix with the coefficients of the ith column of A with row index in $\mathcal{P}_i$. Hence, the algorithm for the computation of F is fully

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

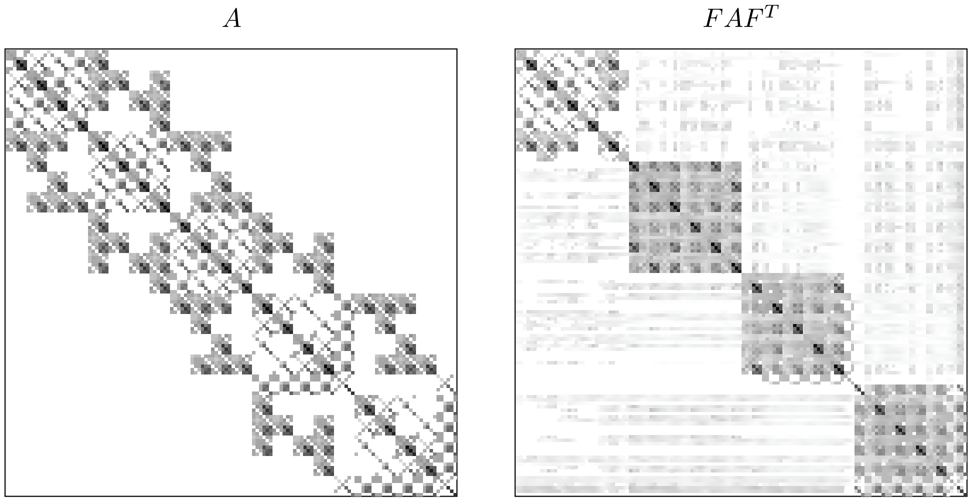

FIG. 2.2. *Nonzero pattern of A (left) and FAF^T (right) for a 132×132 SPD matrix. The darker a point, the larger the absolute value of the corresponding coefficient.*

parallelizable, as it requires the solution of n independent dense subsystems in the form (2.20). Moreover, existence and uniqueness of F is guaranteed by the following.

THEOREM 2.3. *Let $A \in \mathbb{R}^{n\times n}$ be SPD and F computed so as to satisfy* (2.9) *and* (2.14) *for any given $\mathcal{S}_{BL}$. Then F exists and is unique.*

Proof. Let $\mathbf{v}$ be an arbitrary vector in $\mathbb{R}^{nz_i - m}$; i.e., $\mathbf{v}$ has the same size as $\mathbf{f}_{i1}$ in (2.20), and $\mathcal{P}_i$ is the set defined in (2.12). If $\mathcal{P}_i$ is empty, the thesis is trivial, as $\mathbf{f}_i = I[i,:]$. If $\mathcal{P}_i$ is not empty, let $\overline{\mathbf{v}}$ be a vector in $\mathbb{R}^n$ such that

$$\overline{v}_k = 0 \qquad \forall k \notin \mathcal{P}_i, \qquad \text{and } \overline{v}_k = v_k \qquad \forall k \in \mathcal{P}_i;$$

hence $\overline{\mathbf{v}}[\mathcal{P}_i] = \mathbf{v}$. As A is SPD,

$$\mathbf{v}^T A[\mathcal{P}_i, \mathcal{P}_i]\mathbf{v} = \overline{\mathbf{v}}^T[\mathcal{P}_i] A[\mathcal{P}_i, \mathcal{P}_i] \overline{\mathbf{v}}[\mathcal{P}_i] = \overline{\mathbf{v}}^T A \overline{\mathbf{v}} > 0.$$

Therefore, $A[\mathcal{P}_i, \mathcal{P}_i]$ is SPD, too, and the solution to system (2.20) exists and is unique, thus proving the thesis. ∎

As D is arbitrary, there is still no a priori reason to prefer the preconditioned matrix FAF^T to A in a conjugate gradient iteration. However, FAF^T has the nice property for a parallel implementation that the interaction between distinct blocks is weak. An example is shown in Figure 2.2, where the patterns of A (left) and FAF^T (right) are represented with a grey scale apportioned to the coefficient absolute values. The effect of F over A is to concentrate the largest entries into the diagonal blocks. Using a number of blocks at least equal to the number of available processors, we may think of preconditioning each block of FAF^T separately with an efficient sequential preconditioner such as, for example, ILU [23, 27, 28, 34]. To this aim, we have to compute the diagonal blocks B_{i_b} of FAF^T:

$$B_{i_b} = F[\mathcal{J}_{i_b}, :]AF^T[:, \mathcal{J}_{i_b}], \qquad i_b = 1, \dots, n_b, \tag{2.21}$$

with

$$\mathcal{J}_{i_b} = \{k : m(i_b - 1) + 1 \le k \le m i_b\}. \tag{2.22}$$

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

It can be easily seen that B_{i_b} is SPD. Let $\mathbf{x}$ be a nonzero vector in $\mathbb{R}^m$, and let $\mathbf{y} = F^T[:, \mathcal{J}_{i_b}]\mathbf{x}$. As $F^T[:, \mathcal{J}_{i_b}]$ is a full-rank rectangular matrix by construction, $\mathbf{y}$ is a nonzero vector in $\mathbb{R}^n$; hence

$$\mathbf{x}^T B_{i_b} \mathbf{x} = \mathbf{y}^T A \mathbf{y} > 0. \tag{2.23}$$

Because of (2.23), the IC decomposition of each block B_{i_b} can be computed:

$$B_{i_b} \simeq L_{i_b} L_{i_b}^T. \tag{2.24}$$

Collecting all matrices L_{i_b}, $i_b = 1, \dots, n_b$, provides an incomplete block Jacobi preconditioner:

$$J = J_L J_L^T = \begin{bmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_{n_b} \end{bmatrix} \begin{bmatrix} L_1^T & & & \\ & L_2^T & & \\ & & \ddots & \\ & & & L_{n_b}^T \end{bmatrix}. \tag{2.25}$$

The final BFSAI-IC preconditioner is given by the sequential application of F and J_L to A. The resulting preconditioned matrix is

$$J_L^{-1} F A F^T J_L^{-T} = W A W^T, \tag{2.26}$$

where the inner preconditioner F is intended to reduce the interaction between the diagonal blocks of A, while the outer preconditioner J^{-1} improves the conditioning index of each block. The overall preconditioner $M^{-1} = W^T W$ is a parallelizable hybrid between FSAI and ILU. BFSAI-IC coincides with FSAI when the number of blocks n_b is equal to the system size n, while it reduces to IC if only one block is used. Therefore, the BFSAI-IC performance is expected to be intermediate between FSAI and IC according to the number of blocks selected. The choice $F \equiv I$ is also admissible, with $W^T W$ simply resulting in a standard incomplete block Jacobi preconditioner computed on n_b diagonal blocks of A.

3. Numerical implementation.

3. Numerical implementation. The procedure for the BFSAI-IC computation is summarized in Algorithm 3.1. The main body of the procedure is represented by a loop over i_b, i.e., over the blocks. For each block, the ith row of F, $i \in \mathcal{J}_{i_b}$, is computed by solving a small dense SPD linear system (lines 4 to 7). This task can be efficiently carried out by using third level BLAS. Afterwards, F is sparsified by an a posteriori drop of the smallest coefficients with the implementation of a postfiltration strategy [26] (line 8). Though this is not essential for the BFSAI computation, according to our experience, a large part of the entries computed for F is often unnecessary and can be discarded with a limited loss of effectiveness [15]. The postfiltration strategy is also important for making both the computation of B_{i_b} and the preconditioner application faster. The block B_{i_b} is computed through two matrix-by-matrix products, thus resulting generally much denser than the corresponding blocks of A. For such a reason, a second dropping strategy proves appropriate for neglecting the smallest B_{i_b} entries (line 9). Finally, an IC decomposition with partial fill-in of the dropped B_{i_b} provides J_L. There are several effective strategies for computing IC using level-of-fill, tolerance, or fill-in based techniques for selecting the L_{i_b} entries. In this implementation, we use an algorithm similar to those proposed in [34] and [28] based on the selection of a user-specified rowwise number of entries for L_{i_b} in addition to

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

Algorithm 3.1. BFSAI-IC computation

```
 1. Select $n_b$ and $\mathcal{S}_{BL}$
 2. $m = n/n_b$
 3. **Do** $i_b = 1, n_b$
 4.     **Do** $k = 1, m$
 5.         $i = m(i_b - 1) + k$
 6.         **Solve** $A[\mathcal{P}_i, \mathcal{P}_i]\mathbf{f}_{i1} = -A[\mathcal{P}_i, i]$
 7.     **End Do**
 8.     Compute $F[\mathcal{J}_{i_b}, :] = \text{drop}(F[\mathcal{J}_{i_b}, :])$ {parameter $\delta$}
 9.     Compute $B_{i_b} = F[\mathcal{J}_{i_b}, :]AF^T[:, \mathcal{J}_{i_b}]$ {parameter $\rho_B$}
10.     Compute $L_{i_b} L_{i_b}^T \simeq B_{i_b}$ {parameter $\rho_L$}
11. **End Do**
```

those of B_{i_b}. This allows for full control of the memory occupation. Moreover, a level-of-fill based IC that selects the nonzero entries using the adjacency graph only often is not very efficient with ill-conditioned matrices, while the most appropriate drop value selection may prove difficult in a tolerance-based IC. Therefore, the preconditioner density is controlled by three user-specified parameters:

1. δ is the postfiltration parameter denoting the relative tolerance below which the entries of F are discarded;
2. ρ_B is the parameter used to drop the smallest B_{i_b} entries denoting the maximum allowable number of nonzeros for each row of B_{i_b} in addition to the corresponding number of entries of A;
3. ρ_L is the parameter used to fill L_{i_b} denoting the maximum allowable number of nonzeros for each row of L_{i_b} in addition to the corresponding number of entries of B_{i_b}.

Algorithm 3.2 provides the application of BFSAI-IC to a vector $\mathbf{r}$:

$$\mathbf{z} = M^{-1}\mathbf{r} = W^T W \mathbf{r} = F^T J_L^{-T} J_L^{-1} F \mathbf{r}. \tag{3.1}$$

Two loops over i_b are required. First, the product of $F[\mathcal{J}_{i_b}, :]$ by $\mathbf{r}$ is performed to get the subvector $\mathbf{u}[\mathcal{J}_{i_b}]$ (line 2), followed by a forward and backward substitution with L_{i_b} and $L_{i_b}^T$ (lines 3 and 4) to get $\mathbf{w}[\mathcal{J}_{i_b}]$. All the subvectors $\mathbf{w}[\mathcal{J}_{i_b}]$ are then collected to form $\mathbf{w}$. Second, the product $F^T[\mathcal{J}_{i_b}, :]$ by $\mathbf{w}$ is performed to get $\mathbf{z}[\mathcal{J}_{i_b}]$ (line 7). The subvectors $\mathbf{z}[\mathcal{J}_{i_b}]$ form the preconditioned vector $\mathbf{z}$ of (3.1). The algorithm is split into two distinct loops over i_b because the whole vector $\mathbf{w}$ is needed to perform the product in line 7.

BFSAI-IC can be fully implemented in a parallel environment by scheduling a set of blocks $\mathcal{K}_p$ to each processor. In fact, in Algorithm 3.1, one processor needs neither F nor J_L entries computed by another processor. Hence, the preconditioner setup can also be efficiently performed in a distributed computing environment, provided that the A coefficients necessary to form the local dense subsystems are preliminarily collected by each computing core. A little dependence over i_b arises in Algorithm 3.2 only because a few components of $\mathbf{r}$ and $\mathbf{w}$ have to be exchanged among the processors to perform the products in lines 2 and 7, respectively.

The BFSAI-IC algorithms require the number of blocks n_b to be larger than or equal to the number of computing processors n_p. In fact, a parallel implementation with $n_b < n_p$ would be much more difficult and less efficient, as it would require the incomplete factorization of blocks shared by more than one processor. In practice, for a given n_p, the most convenient choice in terms of both wall clock time and iteration

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

ALGORITHM 3.2. BFSAI-IC application

```
1. Do $i_b = 1, n_b$
2.     $\mathbf{u}[\mathcal{J}_{i_b}] = F[\mathcal{J}_{i_b}, :]\mathbf{r}$
3.     Solve $L_{i_b}\mathbf{v}[\mathcal{J}_{i_b}] = \mathbf{u}[\mathcal{J}_{i_b}]$
4.     Solve $L_{i_b}^T\mathbf{w}[\mathcal{J}_{i_b}] = \mathbf{v}[\mathcal{J}_{i_b}]$
5. End Do
6. Do $i_b = 1, n_b$
7.     $\mathbf{z}[\mathcal{J}_{i_b}] = F^T[\mathcal{J}_{i_b}, :]\mathbf{w}$
8. End Do
```

count is to have the blocks as large as possible, so that BFSAI-IC is closer to IC. This implies setting $n_b = n_p$.

We choose to adopt a shared memory programming (SMP) paradigm by using the OpenMP application programming interface (API) [8, 32] that is becoming increasingly popular for its ease of use in multithread applications. The system matrix A and the right-hand side $\mathbf{b}$ are shared variables, while F and J_L are scattered by rows among the processors as private variables. Let $\mathcal{I}_p$ be the set of row indices defined as

$$\mathcal{I}_p = \{i : i \in \mathcal{J}_{i_b} \text{ for } i_b \in \mathcal{K}_p\} \tag{3.2}$$

with $\mathcal{J}_{i_b}$ defined in (2.22). Each processor is the owner of the solution subvectors $\mathbf{x}[\mathcal{I}_p]$ and the other subarrays used in a PCG iteration. Local components are shared with other processors to perform the matrix-by-vector products and apply BFSAI-IC. The interprocessor communications can be significantly reduced if a bandwidth optimization reordering algorithm is used, e.g., reverse Cuthill–McKee (RCM) [11]. Anyway, BFSAI-IC can be efficiently implemented also in a distributed memory environment, e.g. by a message passing (e.g., MPI [30, 33]) implementation.

It is well known that the quality of any approximate inverse depends strongly on the choice of the nonzero pattern $\mathcal{S}_{BL}$. The optimal a priori selection of $\mathcal{S}_{BL}$ is still an open issue. For example, in [31], a matrix aggregation technique is proposed with the aim of capturing an optimal pattern for the FSAI computation. A widely accepted choice is the nonzero pattern of A^k for small values of k [9, 18]. In practical problems, however, the largest feasible pattern is that of A^2 because the number of nonzero entries in each row nz_i increases very quickly with k. As the FSAI computation requires the solution of n dense $nz_i \times nz_i$ systems, the overall numerical complexity is proportional to nz_i^3. In the BFSAI computation, we use a similar strategy. We set $\mathcal{S}_L$ (see (2.3)) as the lower triangular part of the nonzero pattern of A^k, $k \geq 1$, and $\mathcal{S}_{BL} = \mathcal{S}_L \cup \mathcal{S}_{BD}$ (see (2.5) and Figure 2.1). The dense systems (2.20) have size $(nz_i - m)$, i.e., smaller than that required by the standard FSAI. Hence, a larger power of A, say 3 or 4, can be used for $\mathcal{S}_{BL}$, with a potential improvement of the BFSAI quality.

4. Preconditioner analysis. BFSAI-IC needs four user-specified parameters to be set: n_b, δ, ρ_B, and ρ_L. The most decisive one is n_b, i.e., the number of blocks, which in turn is dictated by the number of available processors n_p. If n_b, hence n_p, is small, BFSAI-IC is closer to IC than to FSAI and is generally more effective than the latter. If n_b is large, BFSAI-IC approaches the standard FSAI. The other parameters control the BFSAI-IC density and obviously also impact the preconditioner performance.

The BFSAI-IC effectiveness is evaluated by the number of PCG iterations required to converge to a relative residual equal to 10^{-10} using a right-hand side of all ones.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

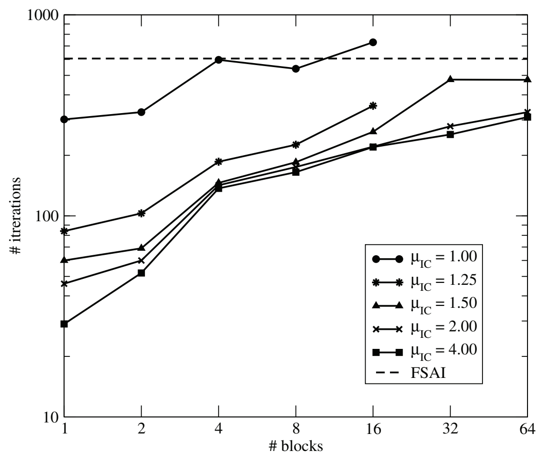

FIG. 4.1. *Number of PCG iterations to converge versus n_b for different values of μ_{IC}.*

As a numerical example, an SPD matrix arising from a three-dimensional (3D) finite element (FE) geomechanical simulation of faulted porous media is selected [13]. The matrix A has size 320,493 with 14,849,397 nonzeros. Both FSAI and BFSAI-IC in these tests are computed using the pattern of A^2 with no postfiltration ($\delta = 0.00$).

Denote by μ_{IC} the density of J_L:

$$\mu_{IC} = \frac{\text{nnz}(J_L)}{\text{nnz}(A)}, \tag{4.1}$$

where nnz$(\cdot)$ is the function providing the number of nonzeros of a matrix. Observe that nnz(J_L), and hence μ_{IC}, is controlled by the user-specified parameters ρ_B and ρ_L. Figure 4.1 shows the number of PCG iterations to converge versus n_b for different values of μ_{IC} in a log-log plot. Increasing n_b, the BFSAI-IC performance deteriorates, though it remains below the horizontal dashed line denoting the standard FSAI. For sparser J_L, negative terms may arise in the $J_L J_L^T$ diagonal. This is a well-known outcome of symmetric incomplete factorizations, causing a slow convergence rate or even a failure [21]. In this case, such an occurrence takes place for $\mu_{IC} \leq 1.25$ and $n_b \geq 32$. By distinction, for larger μ_{IC}, no negative terms are generated in the $J_L J_L^T$ diagonal with similar performances. The iteration count no longer decreases for $\mu_{IC} \geq 4.00$. Therefore, there exists a threshold value of μ_{IC} beyond which the performance of BFSAI-IC cannot be further improved by increasing the J_L density only.

Such a behavior can be better understood looking at Figure 4.2, which shows the number of iterations versus μ_{IC} for a variable n_b. With $n_b = 1$, i.e., BFSAI-IC coincides with IC, the iteration count grows up very quickly for a low density and still keeps on decreasing for larger μ_{IC}. With $n_b \geq 2$, the iteration count does not improve for large μ_{IC}.

The influence of the B_{i_b} density on the overall BFSAI-IC effectiveness is shown in Figure 4.3, which provides the number of iterations to converge versus ρ_B for different values of ρ_L and n_b. As for the J_L density, a threshold value for ρ_B exists beyond which a ρ_B increase gives no benefit over the iteration count. It can also be observed that ρ_B is more important for a large n_b with no appreciable dependency on ρ_L.

Finally, the overall quality of BFSAI-IC as a preconditioner for A is evaluated by the spectral analysis of WAW^T. To this aim, we use a smaller matrix arising

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

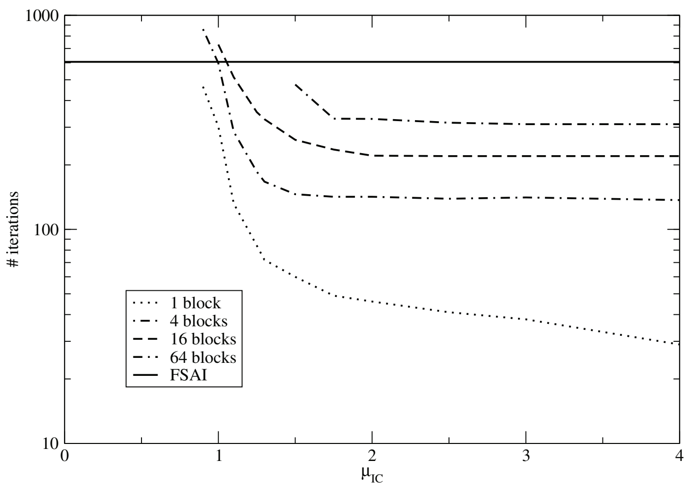

FIG. 4.2. *Number of PCG iterations to converge versus μ_{IC} for different values of n_b.*

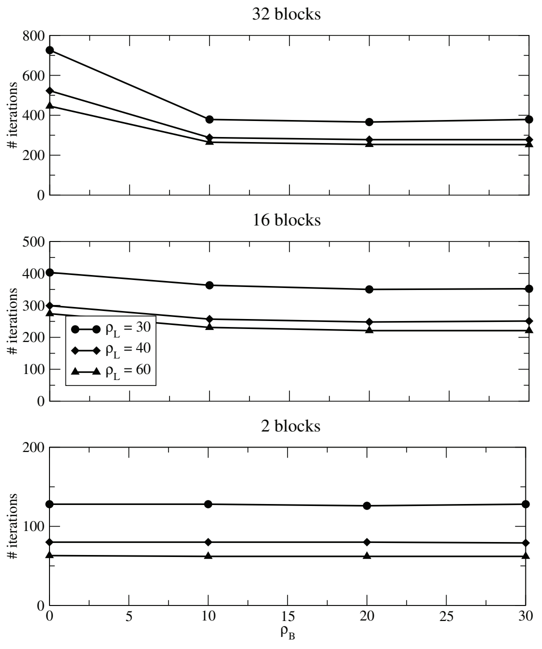

FIG. 4.3. *Number of PCG iterations to converge versus ρ_B for different values of n_b and ρ_L.*

from a 3D FE mechanical application with size 27,158 and 1,162,926 nonzeros. Figure 4.4 shows the corresponding eigenvalue distribution for various n_b values, including $n_b = 1$ (IC) and $n_b = n$ (FSAI), while Table 4.1 provides the corresponding maximum and minimum eigenvalues λ_{max} and λ_{min}, respectively, and the conditioning index $\kappa = \lambda_{max}/\lambda_{min}$. As expected, the best clustering and the minimum κ are obtained with IC, and the worst with FSAI, with BFSAI-IC exhibiting an intermediate behavior. Note that the κ deterioration is mainly due to λ_{min}, as λ_{max} is practically unchanged. This is a typical behavior of preconditioners based on the Frobenius norm minimization. BFSAI-IC tends to mitigate such an occurrence since condition (2.4)

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

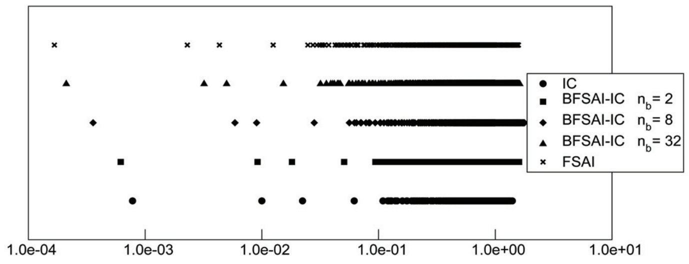

FIG. 4.4. *Eigenvalue distribution for IC, FSAI, and BFSAI-IC for different values of n_b.*

TABLE 4.1
Conditioning index for IC, FSAI, and BFSAI-IC for different values of n_b.

	IC	BFSAI-IC $n_b = 2$	BFSAI-IC $n_b = 8$	BFSAI-IC $n_b = 32$	FSAI
λ_{max}	$1.40{\cdot}10^{+0}$	$1.60{\cdot}10^{+0}$	$1.78{\cdot}10^{+0}$	$1.62{\cdot}10^{+0}$	$1.58{\cdot}10^{+0}$
λ_{min}	$7.08{\cdot}10^{-4}$	$6.18{\cdot}10^{-4}$	$3.58{\cdot}10^{-4}$	$2.11{\cdot}10^{-4}$	$1.67{\cdot}10^{-4}$
κ	$1.80{\cdot}10^{+3}$	$2.59{\cdot}10^{+3}$	$4.97{\cdot}10^{+3}$	$7.68{\cdot}10^{+3}$	$9.50{\cdot}10^{+3}$

requires FA to approach an arbitrary block diagonal matrix D instead of the identity, as required by (2.1).

5. Numerical results.

The BFSAI-IC performance has been experimented with in four SPD test matrices:

- FAULT-639: This arises from the numerical solution by a linear FE of the inequality-constrained minimization problem governing the mechanical equilibrium of a 3D body with contact surfaces [14]. The contact is solved with the aid of a penalty formulation that gives rise to an SPD ill-conditioned linear system of equations.
- STOCF-729: This arises from the FE integration of the diffusion partial differential equation governing the 3D transient flow of groundwater in saturated porous media. The problem is solved assuming a stochastic distribution of the hydraulic conductivity tensor with a large permeability contrast in adjacent elements [16].
- GEO-1438: This arises from a regional geomechanical model of the sedimentary basin underlying the Venice lagoon discretized by a linear FE with randomly heterogeneous properties [36].
- FLAN-1565: This arises from the mechanical equilibrium of a steel flange discretized by a 3D 8-node brick FE [21].

All of the test cases above concern real engineering applications. The size and number of nonzero terms for each matrix is provided in Table 5.1. The linear system is solved by PCG using a vector of all ones as the right-hand side and an exit relative residual smaller than 10^{-10}. Each matrix has been preliminarily reordered by an RCM algorithm. All tests are performed on the IBM SP6/5376 cluster at the CINECA Centre for High Performance Computing, equipped with IBM Power6 processors at 4.7 GHz with 168 nodes, 5376 computing cores, and 21 Tbyte of internal network RAM.

The BFSAI-IC performance is evaluated by the number of iterations, the wall clock time in seconds T_p and T_s for the preconditioner computation and the PCG to converge, respectively, with the total time $T_t = T_p + T_s$. As to the memory occupation,

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

TABLE 5.1
Size and number of nonzeros of the test matrices.

	FAULT-639	STOCF-729	GEO-1438	FLAN-1565
Size	638,812	729,400	1,437,960	1,564,794
# of nonzeros	14,626,683	10,765,586	63,156,690	117,406,044

TABLE 5.2
*BFSAI-IC best performances with FAULT-*639*, STOCF-*729*, GEO-*1438*, and FLAN-*1565*. k is the power of A used to select the $\mathcal{S}_{BL}$ pattern.*

FAULT-639											
$n_b = n_p$	k	δ	ρ_B	ρ_L	# iter.	T_p	T_s	T_t	μ_F	μ_{IC}	μ_T
1	-	-	-	50	153	84.71	48.30	133.01	-	1.58	1.58
2	2	0.00	0	50	256	51.05	37.27	88.32	0.06	1.58	1.64
4	2	0.05	10	50	303	25.88	22.24	48.12	0.03	1.57	1.60
8	2	0.05	10	50	376	13.83	14.33	28.16	0.03	1.56	1.59
16	2	0.05	10	50	482	7.31	9.03	16.34	0.05	1.53	1.58
STOCF-729											
$n_b = n_p$	k	δ	ρ_B	ρ_L	# iter.	T_p	T_s	T_t	μ_F	μ_{IC}	μ_T
1	-	-	-	20	107	18.29	13.76	32.05	-	1.89	1.89
2	2	0.00	10	10	328	5.33	18.73	24.06	0.09	1.22	1.31
4	2	0.00	0	10	383	2.70	11.64	14.34	0.14	1.21	1.35
8	3	0.01	0	10	404	3.56	6.34	9.90	0.17	1.20	1.37
16	3	0.01	0	10	518	2.06	4.45	6.51	0.28	1.17	1.45
GEO-1438											
$n_b = n_p$	k	δ	ρ_B	ρ_L	# iter.	T_p	T_s	T_t	μ_F	μ_{IC}	μ_T
1	-	-	-	0	275	22.11	120.37	142.48	-	0.50	0.50
2	1	0.00	0	0	315	16.29	54.96	71.25	0.03	0.50	0.53
4	1	0.00	0	0	357	8.42	32.23	40.65	0.03	0.50	0.53
8	1	0.00	0	0	419	4.60	20.65	25.25	0.05	0.49	0.54
16	1	0.00	0	0	526	2.69	13.01	15.70	0.08	0.49	0.57
FLAN-1565											
$n_b = n_p$	k	δ	ρ_B	ρ_L	# iter.	T_p	T_s	T_t	μ_F	μ_{IC}	μ_T
1	-	-	-	40	1365	194.98	2102.17	2297.15	-	1.04	1.04
2	2	0.00	10	30	1917	142.33	769.65	911.98	0.03	0.91	0.94
4	2	0.00	10	30	1988	70.76	425.64	496.40	0.06	0.91	0.97
8	2	0.00	10	30	2089	35.84	246.28	282.12	0.11	0.91	1.02
16	2	0.00	10	30	2284	23.16	156.58	179.73	0.21	0.92	1.13

along with μ_{IC} of (4.1), we define the F density as

$$\mu_F = \frac{\text{nnz}(F)}{\text{nnz}(A)} \tag{5.1}$$

and the total density as $\mu_T = \mu_{IC} + \mu_F$. Table 5.2 provides the best BFSAI-IC performance from the different tests with 1 to 16 processors and $n_b = n_p$. When $n_p = 1$, BFSAI-IC coincides with IC; hence there is no need for setting the user-specified parameters k, δ, and ρ_B. If we set $F = I$, the resulting incomplete block Jacobi preconditioning on A does not allow for convergence with $n_b = n_p \geq 2$ in all examples. By contrast, although F is very sparse ($\mu_F \ll 1$, especially for low n_p values), the incomplete block Jacobi preconditioning proves very efficient on FAF^T, with generally a limited increase of the iteration count relative to the $n_p = 1$ case.

As expected from the analysis carried out in the previous section and shown by the iteration count in Table 5.2, the BFSAI-IC preconditioning progressively deteriorates as n_p, and hence n_b, grows. Such a deterioration appears to be problem-dependent.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

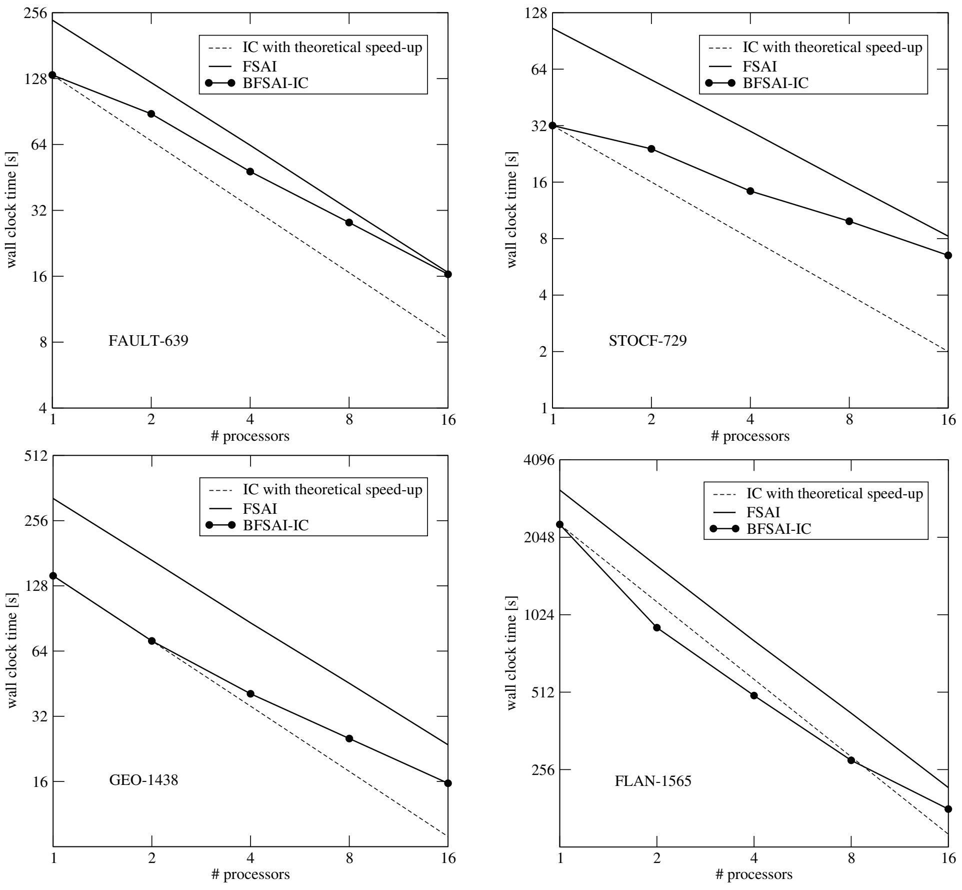

FIG. 5.1. *Wall clock time versus the number of processors for the optimal FSAI and BFSAI-IC. The theoretical IC profile with perfect speedup is also shown.*

For any n_p selection, however, BFSAI-IC always outperforms the standard FSAI preconditioning. Figure 5.1 compares the overall wall clock time elapsed using FSAI, with the pattern and postfiltration parameters providing the best performance, and BFSAI-IC. For the sake of the comparison, the theoretical IC profile obtained if IC were perfectly scalable with n_p is also reported. The BFSAI-IC profile turns out to be generally comprised between the theoretical IC and FSAI, thus proving a hybrid preconditioner superior to FSAI alone. As n_p grows, the IC influence progressively decreases, and BFSAI-IC tends to behave like FSAI. In the largest problems (GEO-1438 and FLAN-1565), however, FSAI with 16 processors still requires a wall clock time 1.5 times larger than BFSAI-IC. The overall gain of BFSAI-IC versus n_p is summarized in Figure 5.2, which shows the ratio between T_t provided by FSAI and BFSAI-IC. BFSAI-IC proves even more than twice faster than FSAI according to the problem and the number of processors.

Finally, the BFSAI-IC parallel efficiency is addressed, fixing the number of blocks ($n_b = 16$) and varying n_p. Figure 5.3 shows the wall clock time versus the number of processors. Once n_b is set, BFSAI-IC exhibits an excellent degree of parallelism up to $n_p = n_b$. In particular, both the BFSAI-IC computation and the application are almost ideally parallel, as shown in Table 5.3, where speed-up values between 11.5 and 13.4 are obtained with 16 processors.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

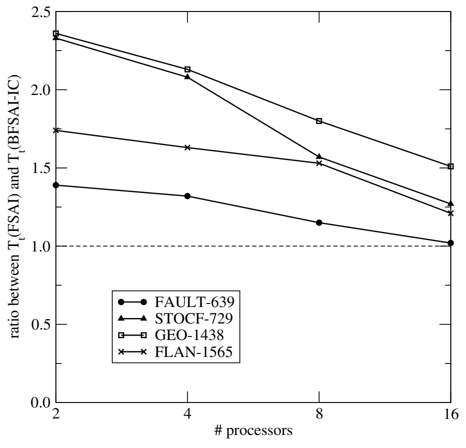

FIG. 5.2. *Ratio between the total wall clock time required by FSAI and BFSAI-IC versus the number of processors.*

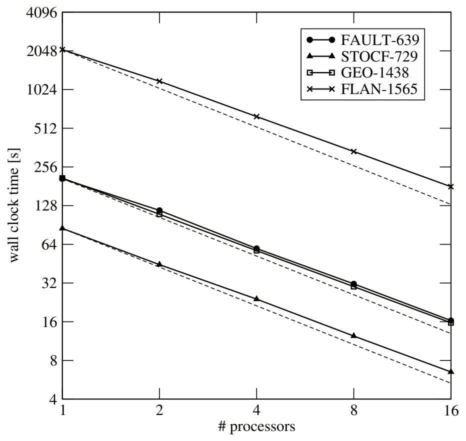

FIG. 5.3. *Wall clock time versus the number of processors for the optimal BFSAI-IC with $n_b = 16$. The theoretical speed-up profile is plotted by a dashed line.*

TABLE 5.3
Speed-up values using BFSAI-IC with $n_b = 16$. The user-specified parameters of Table 5.2 are used. S_p, S_s, and S_t are the speedups obtained for the preconditioner computation, for the PCG to converge, and for the whole solution process, respectively.

	$n_p = 1$			$n_p = 16$					
	T_p	T_s	T_t	T_p	T_s	T_t	S_p	S_s	S_t
FAULT-639	95.68	111.74	207.42	7.31	9.03	16.34	13.1	12.4	12.7
STOCF-729	26.76	58.39	85.15	2.06	4.45	6.51	13.0	13.1	13.1
GEO-1438	35.52	174.17	209.69	2.69	13.01	15.70	13.2	13.4	13.4
FLAN-1565	292.89	1801.19	2094.08	23.16	156.58	179.74	12.6	11.5	11.7

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

6. Conclusions.

A hybrid preconditioner coupling a generalized FSAI with an incomplete block Jacobi algorithm has been developed for the efficient parallel solution to SPD linear systems of equations. The block FSAI preconditioner F is computed using a sparse lower block triangular pattern $\mathcal{S}_{BL}$ with the aim of minimizing the distance of FAF^T from an arbitrary block diagonal matrix D in the sense of the Frobenius norm. An incomplete block Jacobi algorithm is then effectively applied for preconditioning FAF^T. The resulting BFSAI-IC preconditioner is therefore made of an inner (F) and an outer (J_L) preconditioner which can be efficiently computed and used on parallel computer architectures.

The results from a number of realistic test cases from engineering applications show the following:

- The BFSAI-IC quality as a preconditioner progressively deteriorates as the number of blocks increases. In the limiting case with $n_b = n$, however, it coincides with FSAI; hence BFSAI-IC proves generally superior to FSAI for any realistic number of blocks.
- The incomplete block Jacobi preconditioner is very attractive from a computational viewpoint because of its simplicity and excellent parallel efficiency; however, it often allows for a very poor solver acceleration. In the examples discussed in the present paper, the incomplete block Jacobi algorithm by itself does not yield convergence for any fill-in degree. By contrast, the preliminary BFSAI application makes it an efficient technique for ill-conditioned problems as well.
- For a given number of blocks, the BFSAI-IC preconditioner can be both computed and applied in a parallel environment exhibiting very good speed-up values.
- BFSAI-IC appears to be a robust and efficient parallel preconditioner for SPD matrices arising from a variety of engineering problems. It is significantly superior to a standard parallel preconditioner such as FSAI for small n_p (2 to 8) and, depending on the problem nature and size, can be still competitive for $n_p \geq 16$ as well.

REFERENCES

[1] E. Anderson and Y. Saad, *Solving sparse triangular systems on parallel computers*, Int. J. High Speed Com., 1 (1989), pp. 73–95.

[2] S. T. Barnard, L. M. Bernardo, and H. D. Simon, *An MPI implementation of the SPAI preconditioner on the T3E*, Int. J. High Perform. Comput. Appl., 13 (1999), pp. 107–123.

[3] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, J. Comput. Phys., 182 (2002), pp. 418–477.

[4] M. Benzi, C. D. Meyer, and M. Tůma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput. 17 (1995), pp. 1135–1149.

[5] M. Benzi and M. Tůma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput. 19 (1998), pp. 968–994.

[6] M. Benzi and M. Tůma, *A parallel solver for large-scale Markov chains*, Appl. Numer. Math. 41 (2002), pp. 135–153.

[7] M. Byckling and M. Huhtanen, *Approximate Factoring of the Inverse*, http://math.tkk.fi/∼mhuhtane/ (2002).

[8] B. Chapman, G. Jost, and R. Van Der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*, The MIT Press, Cambridge, MA, 2008.

[9] E. Chow, *A priori sparsity patterns for parallel sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., 21 (2000), pp. 1804–1822.

[10] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., 19 (1998), pp. 995–1023.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.

[11] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings of the 1969 24th National Conference, ACM, New York, 1969, pp. 157–172.

[12] I. S. Duff and G. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–657.

[13] M. Ferronato, G. Gambolati, C. Janna, and P. Teatini, *Numerical modelling of regional faults in land subsidence prediction above gas/oil reservoirs*, Int. J. Numer. Anal. Meth. Geomech., 32 (2008), pp. 633–657.

[14] M. Ferronato, C. Janna, and G. Gambolati, *Mixed constrained preconditioning in computational contact mechanics*, Comput. Methods Appl. Mech. Engrg., 197 (2008), pp. 3922–3931.

[15] M. Ferronato, C. Janna, and G. Pini, *Parallel solution to ill-conditioned FE geomechanical problems*, Int. J. Numer. Anal. Meth. Geomech., submitted.

[16] M. Ferronato, G. Pini, and C. Janna, *A shifted FSAI preconditioner for the efficient parallel solution of transient groundwater flow models*, Adv. Eng. Softw., submitted.

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

[18] T. Huckle, *Approximate sparsity patterns for the inverse of a matrix and preconditioning*, Appl. Numer. Math., 30 (1999), pp. 291–303.

[19] M. Huhtanen, *Factoring matrices into the product of two matrices*, BIT, 47 (2007), pp. 793–808.

[20] D. Hysom and A. Pothen, *Efficient parallel computation of ILU(k) preconditioners*, in Proceedings of the 1999 ACM/IEEE Conference on Supercomputing, article 29.

[21] C. Janna, A. Comerlati, and G. Gambolati, *A comparison of projective and direct solvers for finite elements in elastostatics*, Adv. Eng. Softw., 40 (2009), pp. 675–685.

[22] M. T. Jones and P. E. Plassmann, *Scalable iterative solution of sparse linear systems*, Parallel Comput., 20 (1994), pp. 753–773.

[23] D. S. Kershaw, *The incomplete Cholesky-conjugate gradient method for the iterative solution of systems of linear equations*, J. Comput. Phys., 26 (1978), pp. 43–65.

[24] L. Yu. Kolotilina and A. Yu. Yeremin, *Factorized sparse approximate inverse preconditionings* I. *Theory*, SIAM J. Matrix Anal. Appl., 14 (1993), pp. 45–58.

[25] L. Yu. Kolotilina and A. Yu. Yeremin, *Factorized sparse approximate inverse preconditionings* II. *Solution of* 3*D FE systems on massively parallel computers*, Int. J. High Speed Comput., 7 (1995) pp. 191–215.

[26] L. Yu. Kolotilina and A. Yu. Yeremin, *Factorized sparse approximate inverse preconditioning* IV, Numer. Linear Algebra Appl., 6 (1999), pp. 515–531.

[27] N. Li, Y. Saad, and E. Chow, *Crout versions of ILU for general sparse matrices*, SIAM J. Sci. Comput., 25 (2003), pp. 716–728.

[28] C.-J. Lin and J. J. Moré, *Incomplete Cholesky factorizations with limited memory*, SIAM J. Sci. Comput., 21 (1999), pp. 24–45.

[29] 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.

[30] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, version 2.2, http://www.mpi-forum.org (2009).

[31] A. A. Nikishin and A. Yu. Yeremin, *Prefiltration technique via aggregation for constructing low-density high-quality factorized sparse approximate inverse preconditioning*, Numer. Linear Algebra Appl., 10 (2003), pp. 235–246.

[32] OpenMP website, http://www.openmp.org.

[33] P. S. Pacheco, *Parallel Programming with MPI*, Morgan Kaufmann, San Francisco, CA, 1997.

[34] Y. Saad, *ILUT: A dual threshold incomplete ILU factorization*, Numer. Linear Algebra Appl., 1 (1994), pp. 387–402.

[35] R. D. Smith, P. E. Bjørstad, and W. D. Gropp, *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*, Cambridge University Press, Cambridge, UK, 1996.

[36] P. Teatini, M. Ferronato, G. Gambolati, D. Baù, and M. Putti, *Anthropogenic Venice uplift by seawater pumping into a heterogeneous aquifer system*, Water Resour. Res., to appear.

[37] A. Yu. Yeremin, L. Yu. Kolotilina, and A. A. Nikishin, *Factorized sparse approximate inverse preconditionings* III. *Iterative construction of preconditioners*, J. Math. Sci., 101 (2000), pp. 3237–3254.

[38] A. Yu. Yeremin and A. A. Nikishin, *Factorized-sparse-approximate-inverse preconditionings of linear systems with unsymmetric matrices*, J. Math. Sci., 121 (2004), pp. 2448–2457.

Copyright © by SIAM. Unauthorized reproduction of this article is prohibited.