BANDED TARGET MATRICES AND RECURSIVE FSAI FOR PARALLEL PRECONDITIONING

L. BERGAMASCHI*† AND A. MARTíNEZ*

Abstract. In this paper we propose a parallel preconditioner for the CG solver based on successive applications of the FSAI preconditioner. We first compute an FSAI factor G_{out} for coefficient matrix A, and then another FSAI preconditioner is computed for either the preconditioned matrix $S = G_{out}AG_{out}^T$ or a sparse approximation of S. This process can be iterated to obtain a sequence of triangular factors whose product forms the final preconditioner. Numerical results onto large SPD matrices arising from geomechanical models account for the efficiency of the proposed preconditioner which provides a reduction of the iteration number and of the CPU time of the iterative phase with respect to the original FSAI preconditioner.

1. Introduction.

The efficient parallel solution of sparse linear systems of equations

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

where $A \in \mathbb{R}^{n\times n}$, $\boldsymbol{x}$ and $\boldsymbol{b} \in \mathbb{R}^n$, is a key issue in many numerical computations in science and engineering. Iterative methods based on Krylov subspaces involve matrix-vector products, dot products, and `daxpy` only, so they can be, at least in principle, almost ideally implemented on parallel computers. However, the computation of an effective preconditioner often is not, and perhaps this is the most decisive effort for the efficient solution to (1.1).

Factorized sparse approximate inverses are inherently parallel since their application to a vector consists in two matrix-by-vector products. 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 and SAINV algorithms [2, 3, 1]. The incomplete biorthogonalization of matrix A used to compute (S)AINV, however, is inherently sequential, and the efforts to parallelize its construction did not reveal completely satisfactory.

Differently, the approximate inverse M computed by minimizing the Frobenius norm of $(I - AM)$ 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 exchange 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 then perform the preconditioner setup with no additional communication overhead.

Among 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 in [16]. The FSAI preconditioner is based on an a-priori determination of the sparsity pattern which is usually selected as that of $\widetilde{A}^d$ where $\widetilde{A}$ is obtained from A by dropping the elements below a prescribed threshold (prefiltration) and $d = 1, 2, \cdots$ is a small positive integer. Once the triangular factor of the preconditioner is obtained, it is furtherly sparsified by a second dropping procedure called postfiltration. Parallelization of FSAI preconditioners has been performed and tested e.g. in [5, 7, 6] where prefiltration and postfiltration have been implemented together with a priori sparsity pattern based on nonzeros of A^d with $d \leq 4$.

The drawback of the FSAI approach can be summarized as follows:

*Department of Mathematical Methods and Models for Scientific Applications, University of Padova, via Trieste 63, 35121 Padova, Italy, `e-mail` berga@dmsa.unipd.it, acalomar@dmsa.unipd.it

†corresponding author

1. The inverse of a sparse matrix may be dense with the entries of A^{-1} in most cases slowly decaying away from the main diagonal.
2. A fixed sparsity pattern, based on small powers of A, can hardly capture all the most important nonzeros in A^{-1}.

For these reasons the FSAI preconditioner, despite of its powerful parallel potential of the setup phase, very often produces slow convergence of iterative methods. Recently, an attempt to enlarge the sparsity patterns without computing higher powers of A have been performed in [12] where an adaptive sparsity pattern is constructed by minimizing the Kaporin number.

In this paper we propose an implicit enlargement of the sparsity pattern using a target matrix B as in [11]. Here we choose B to be a banded matrix: the lower factor of the FSAI preconditioner is obtained by minimizing $\|B - GL\|_F$ over the set of matrices G having a fixed sparsity pattern. Denoting with G_{out} the result of this minimization, we propose to compute explicitly the preconditioned matrix $G_{out}AG_{out}^T$ and then to compute a second FSAI factor G_{in} for this matrix. Thus the final preconditioner can be written as the product $G_{out}G_{in}G_{in}^TG_{out}^T$. This procedure, which we call RFSAI – recursive FSAI – can be iterated a number of times to yield a preconditioner written as a product of several triangular factors.

We present the numerical results obtained using the RFSAI preconditioner in combination with the PCG solver in the solution of large linear systems arising from Finite Element discretization of geomechanical models. We compare the proposed approach with the naive FSAI, showing that, for a fixed number of processors, RFSAI produces on the average a more efficient preconditioner (in terms of iteration number and CPU time) for a given nonzero number. The setup time often increases due to the increasing complexity of the preconditioner but it can be kept under control mainly by proper choice of the prefiltration parameters. We also successfully tried RFSAI to accelerate a PCG-like iterative eigensolver (DACG, see [4]) to compute the leftmost eigenpairs of our SPD test matrices.

The paper is organized as follows. In Section 2 we describe the RFSAI algorithm and its parallel implementation. Section 3 is devoted to the presentation of our test matrices while a numerical comparison between RFSAI and the original FSAI in the solution of linear systems is performed in Section 4 where also the scalability results are reported. In Section 5 we present the parallel results of the RFSAI-DACG solver. The conclusions end the paper in Section 6.

2. The Recursive FSAI (RFSAI) algorithm. Let $A \in \mathbb{R}^{n\times n}$ be a SPD matrix. The basic idea of the native FSAI is to find a matrix $G \in \mathcal{A}_S$ such that:

$$G = \underset{G\in\mathcal{A}_S}{\operatorname{argmin}} \ \|I - GL\|_F, \tag{2.1}$$

where L is the lower triangular Cholesky factor of A, i.e. $A = LL^T$, and $\mathcal{A}_S$ is the set of matrices with a prescribed lower triangular non-zero pattern, i.e. all matrices such that:

$$G_{ij} = 0 \qquad \forall (i,j) \notin S \tag{2.2}$$

with:

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

The matrix G satisfying (2.1) can be obtained by solving a set of n independent linear systems since

$$\|I - GL\|_F^2 = \sum_{i=1}^{n} \|\boldsymbol{e}_i^T - \boldsymbol{g}_i^T L\|_2^2 = \sum_{i=1}^{n} \|L\boldsymbol{g}_i - \boldsymbol{e}_i\|_2^2 \tag{2.4}$$

where $\boldsymbol{g}_i^T$ is the ith row of matrix G. Minimum of $\|I - GL\|_F$ is obtained by minimizing separately $\|L\boldsymbol{g}_i - \boldsymbol{e}_i\|_2,\ i = 1, \cdots, n$; such a minimization can be accomplished in a lest-squares

sense by solving:

$$A_i^R \boldsymbol{g}_i = L^T \boldsymbol{e}_i, \qquad i = 1, \cdots, n \tag{2.5}$$

where A_R is the $n_i \times n_i$ matrix obtained from A by setting to zero all the rows and columns of A of index j such that $(i,j) \notin \mathcal{A}_S$. Under this constraint, also the right hand side simplifies to $L^T \boldsymbol{e}_i = l_{ii} \boldsymbol{e}_i$, being l_{ii} generally unknown. In practice, the dense linear system to be solved is

$$A_i^R \boldsymbol{g}_i = \boldsymbol{e}_i$$

and matrix G is properly diagonally scaled afterwards.

2.1. 2-step RFSAI preconditioners. RFSAI can be viewed as a generalization of FSAI. Differently from the classical FSAI approach, and following the idea of [11] we look for $G_{out} \in \mathcal{A}_S$ such that:

$$G_{out} = \underset{G \in \mathcal{A}_S}{\operatorname{argmin}} \ \|B - GL\|_F \tag{2.6}$$

where we choose B to be an arbitrary (possibly dense) triangular banded matrix, depending on a positive integer parameter `nband`:

$$b_{ij} = 0 \qquad \Longleftarrow \qquad |i-j| > \texttt{nband}. \tag{2.7}$$

The development that follows is in part inspired by the work in [13] in which the authors choose B to be instead block diagonal where the numbers of blocks equate the number of processors in view of an incomplete Cholesky factorization of the blocks. They obtain a very efficient preconditioner for a relatively small number of processors, where their approach still retains the good convergence properties of IC(0). However this preconditioner is not scalable since the number of iteration increases, sometimes dramatically, with the number of processors.

Reasoning as before, minimization (2.6) is accomplished by solving independently n least square problems

$$A_i^R \boldsymbol{g}_i = l_{ii} \boldsymbol{b}_i. \tag{2.8}$$

Vector $\boldsymbol{b}_i$ is the i-th row of matrix B. In view of (2.7) it plays a number of entries equal to zero, the remaining ones being arbitrary so that we can partition the vector $\boldsymbol{c} = l_{ii}\boldsymbol{b}_i$, and $\boldsymbol{g}_i$ and A_i^R accordingly, as

$$\boldsymbol{c} = \begin{pmatrix} 0 \\ \boldsymbol{c}_2 \end{pmatrix}, \qquad \boldsymbol{g}_i = \begin{pmatrix} \boldsymbol{g}_{out} \\ \boldsymbol{g}_{in} \end{pmatrix} \qquad A_i^R = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}.$$

Since $\boldsymbol{c}_2$ is arbitrary, we are free of choosing the vector $\boldsymbol{g}_{in} \neq 0$. The most natural choice is $\boldsymbol{g}_{in} = \boldsymbol{e}_i$ which reduces system (2.8) to

$$A_{11} \boldsymbol{g}_{out} = -A_{12} \boldsymbol{e}_i.$$

As a consequence, the dense linear systems to be solved by RFSAI will be smaller than those needed to compute FSAI.

Our choice of matrix B will produce a fixed number of iterations, irrespective of the number of processors employed. Once matrix G_{out} has been computed the second step is to explicitly form matrix $B = G_{out} A G_{out}^T$. This matrix should resemble a banded matrix in view of the minimization (2.6), so that we can define a banded matrix $\widetilde{B} = \texttt{drop}(B, \texttt{nband})$ with coefficients defined as

$$\widetilde{b}_{ij} = \begin{cases} b_{ij} & \text{if } |i-j| \leq \texttt{nband} \\ 0 & \text{otherwise} \end{cases}.$$

In general we can not guarantee $\widetilde{B}$ to be SPD, so that an SPD FSAI preconditioner may not exist. However the following theorem states that if we choose for $\widetilde{B}$ the parameter $d = 1$, i.e. we use the lower triangular part of the matrix itself as the sparsity pattern, then the FSAI preconditioner is well defined.

Theorem 2.1. Given an SPD matrix A and its sparsity pattern S_A. Given another sparsity pattern S_B such that $\{(i,i), i = 1, \cdots, n\} \subset S_B \subset S_A$ we define the matrix $B = \{b_{ij}\}$ as

$$b_{ij} = \begin{cases} a_{ij} & \text{if } (i,j) \in S_B \\ 0 & \text{otherwise} \end{cases}.$$

Then the FSAI preconditioner for B using $d = 1$ is the same as that for A, obtained using the lower triangular part of S_B as the sparsity pattern if the same prefiltration and postfiltration parameters are used.

Proof. In the construction of row i of the FSAI preconditioner for B, the matrix B_R in dense system

$$B_R \boldsymbol{g}_i = \boldsymbol{e}_i.$$

has as coefficients a_{kj} such that $(i,j) \in S_B$ and $(i,k) \in S_B$, which are exactly the same as if we computed the ith row of the FSAI preconditioner for A with S_B as the sparsity pattern. ∎

The second step of our preconditioner setup can therefore be summarized as follows

- Compute an FSAI preconditioner for $\widetilde{B}$ using the lower triangular part of this matrix as the sparsity pattern.

$$G_{in} G_{in}^T \approx \widetilde{B}^{-1}.$$

The final 2-step RFSAI preconditioner is given by the sequential application of G_{out} and G_{in} to A. The resulting preconditioned matrix is:

$$G_{in} G_{out} A G_{out}^T G_{in}^T = WAW^T. \tag{2.9}$$

If we chose higher powers to determine the sparsity pattern for this second preconditioner factor, the existence of the FSAI preconditioner would not be guaranteed since $\widetilde{B}$ is not SPD. This can limit the possibility of finding an efficient approximation of $\widetilde{B}^{-1}$ and, consequently, of developing a performing overall preconditioner.

To overcome this problem, we will investigate the following variant of the preconditioner just described: once the G_{out} matrix has been provided, we can compute the whole matrix product $B = G_{out} A G_{out}^T$ and then compute a preconditioner factor G_{in} for B:

$$G_{in} G_{in}^T \approx B^{-1}.$$

The preconditioned matrix takes on the same form as in (2.9). In this second approach the preconditioner setup CPU time is expected to be larger due to computation of matrix B, however we have more freedom in the choice of the sparsity pattern of the G_{in} preconditioner factor hence we expect to produce a more efficient preconditioner.

2.2. Recursive FSAI preconditioners. The procedure just described can be iterated a number of times on order to define a sequence of triangular factors

$$G^{(1)}, G^{(2)}, \cdots,$$

as described in Algorithms 2.1 and 2.2. We denote as FSAIout the procedure which computes the G_{out} factor by minimizing (2.6), as described in the previous section. FSAIout depends on the `nband` parameter in addition to the usual FSAI parameters δ_{out}, prefiltration threshold, d_{out},

power of $\widetilde{A}$ defining the sparsity pattern and ε_{out} postfiltration parameter. With $\texttt{drop}(A, \texttt{nband})$ we denote the matrix B such that

$$b_{ij} = \begin{cases} a_{ij} & \text{if } |i-j| \leq \texttt{nband} \\ 0 & \text{otherwise} \end{cases}.$$

The RFSAI1 algorithm implements recursively the first approach described in §2.1. The second preconditioner factor, G_{in}, is obtained by applying the canonical FSAI procedure to the matrix $\texttt{drop}(G_{out}^{(k)} A^{(k-1)} {G_{out}^{(k)}}^T, \texttt{nband})$ with parameters $\delta_{in} = 0, d_{in} = 1$ and ε_{in}.

In the implementation of the second approach, RFSAI2, the G_{in} factor is the result of the FSAI procedure applied to the whole product matrix $G_{out}^{(k)} A^{(k-1)} {G_{out}^{(k)}}^T$, with parameters δ_{in}, d_{in} and ε_{in}.

Algorithm 2.1: RFSAI1

Set $A^{(0)} = A$
FOR $k = 1$ TO K DO
$G_{out}^{(k)} = \text{FSAIout}(A^{(k-1)}, \texttt{nband}, \delta_{out}, d_{out}, \varepsilon_{out})$
$A^{(k)} = \texttt{drop}(G_{out}^{(k)} A^{(k-1)} {G_{out}^{(k)}}^T, \texttt{nband})$
$G_{in}^{(k)} = \text{FSAI}(A^{(k)}, 0.0, 1, \varepsilon_{in})$
END FOR.

Algorithm 2.2: RFSAI2

Set $A^{(0)} = A$
FOR $k = 1$ TO K DO
$G_{out}^{(k)} = \text{FSAIout}(A^{(k-1)}, \texttt{nband}, \delta_{out}, d_{out}, \varepsilon_{out})$
$A^{(k)} = G_{out}^{(k)} A^{(k-1)} {G_{out}^{(k)}}^T$
$G_{in}^{(k)} = \text{FSAI}(A^{(k)}, \delta_{in}, d_{in}, \varepsilon_{in})$
END FOR.

In both algorithms, the final preconditioner takes on the form:

$$WW^T = \prod_{k=1}^{K} G_{in}^{(k)} G_{out}^{(k)} \prod_{k=K}^{1} {G_{out}^{(k)}}^T {G_{in}^{(k)}}^T$$

whose application consists in $4K$ matrix-vector products.

Remark 2.1. Setting $\texttt{nband} = 1$ in RFSAI1 we obtain the classical FSAI preconditioner.

Remark 2.2. Our RFSAI2 algorithm shares a number of common features with the DFSAI (Double FSAI) preconditioner which was proposed in [15]. However, differently from [15], we first construct an approximate and sparser FSAI G_{out} so as to have a sparser product $S = G_{out} A G_{out}$. Then we compute the exact FSAI preconditioner for S whereas within the DFSAI preconditioner the sparse approximate inverse for S is computed by iteratively solving to a low accuracy the dense linear systems (2.5).

2.3. Parallel implementation. We have developed a parallel code which implements the construction and application inside parallel PCG, of the two FSAI based algorithms described so far along with the original FSAI preconditioner. The resulting program is written in Fortran 90 and exploits the MPI library for exchanging data among the processors. We use a block row distribution of all matrices, that is, with complete rows assigned to different processors. All the matrices are stored in static data structures in CSR format.

The FSAI and the RFSAI preconditioners will be used to accelerate the PCG solver. In our implementation, we made use of an optimized parallel matrix-vector product which has been developed in [17] showing its effectiveness up to 1024 processors.

3. Test problems. We report in the subsequent sections the results of our experiments with RFSAI1 and RFSAI2 preconditioners in the solution of a set of problems of large size.

The test cases are all realistic examples of large size arising from 2D and 3D FE discretization of geomechanical problems. In detail:

1. FAULT-639: arises from the numerical solution by a linear FE of the inequality-constrained minimization problem governing the mechanical equilibrium of a3D body with contact surfaces. The contact is solved with the aid of a penalty formulation that gives rise to an SPD ill-conditioned linear system.
2. EMILIA-923: arises from the regional geomechanical model of a deep hydrocarbon reservoir. It is obtained discretizing the structural problem with tetrahedral Finite Elements. Due to the complex geometry of the geological formation it was not possible to obtain a computational grid characterized by regularly shaped elements.
3. GEO-1438: arises from a regional geomechanical model of the sedimentary basin underlying the Venice lagoon discretized by a linear FE with randomly heterogeneous properties. The computational domain is a box with an areal extent of 50 x 50 km and 10 km deep consisting of regularly shaped tetrahedral Finite Elements.

These matrices are publicly available in the University of Florida Sparse Matrix Collection at `http://www.cise.ufl.edu/research/sparse/matrices`. The size and number of nonzero terms for each matrix is provided in Table 3.1.

Table 3.1
Size n and number of nonzeros `nnz` *of the test matrices.*

name	n	nnz
FAULT-639	638 812	14 626 683
EMILIA-923	923 136	41 005 206
GEO-1438	1 437 960	63 156 690

All tests are performed on the IBM SP6/5376 cluster at the CINECA Centre for HCP, equipped with IBM Power6 processors at 4.7 GHz with 168 nodes, 5376 computing cores, and 21 Tbytes of internal network RAM. The Fortran 90 is compiled with `-O4 -q64 -qarch=pwr6 -qtune=pwr6 -qnoipa -qstrict -bmaxdata:0x70000000` options.

4. Numerical results. PCG solution of linear systems. The linear system (1.1) is solved by PCG using the exact solution as a vector of all ones. The exit test for the iterative solver is $\dfrac{\|\boldsymbol{r}_k\|}{\|\boldsymbol{b}\|} \leq 10^{-10}$, $\boldsymbol{r}_k$ being the relative residual at iteration k. Each matrix has been preliminarily reordered by a Reverse Cuthill McKee (RCM) algorithm [10]. In all the forthcoming tables we provide the number of iteration (iter), two indicators of the density of the RFSAI preconditioner:

$$\rho_1 = \frac{2\mathtt{nnz}(G_{out}) - n}{\mathtt{nnz}(A)}, \qquad \rho_2 = \frac{2\mathtt{nnz}(G_{in}) - n}{\mathtt{nnz}(A)},$$

as well as three CPU times. In particular we report: T_{prec}, the cost of FSAI computation, T_{sol}, the cost of the iterative solver and $T_{tot} = T_{prec} + T_{sol}$, the total time. For a fixed test case all the runs have been performed using a fixed number of processors ($p = 16$). All the results reported in this Section are obtained using $K = 1$.

4.1. Role of the parameter `nband` in the RFSAI1 algorithm. The parameter `nband` plays a twofold role:

- The complexity of the G_{out} preconditioner is directly proportional to its value.
- The complexity of the G_{in} preconditioner is inversely proportional to `nband`.

Regarding the number of iterations, we can expect a slight increasing when `nband` grows.

Table 4.1
Timings and iteration number for RFSAI1-PCG. Matrix GEO-1438.

G_{out}			`nband`	G_{in}			ρ_1	ρ_2	iter	T_{prec}	T_{sol}	T_{tot}
δ	d	ε		δ	d	ε						
0.1	4	0.1	10000	0.0	1	0.05	0.16	0.19	641	17.05	13.57	30.62
0.1	4	0.1	1000	0.0	1	0.05	0.21	0.15	602	19.19	12.69	31.88
0.1	4	0.1	100	0.0	1	0.05	0.25	0.11	601	22.30	13.34	35.64
0.1	4	0.1	10	0.0	1	0.05	0.27	0.09	601	22.68	12.62	35.30
0.1	4	0.1	2	0.0	1	0.05	0.33	0.04	589	23.56	12.47	36.03
0.1	4	0.1					0.34		585	20.37	11.89	32.25

Table 4.1 shows the RFSAI1 behavior for a number of values of `nband` between 2 and 10000 for matrix GEO-1438. From Table 4.1 we note that both the CPU time for the linear solver and the number of iterations are roughly constant, while high values of `nband` produce a reduction in the setup CPU time. This suggests that the RFSAI1 algorithm may be used to reduce the setup CPU time and the density of a given FSAI preconditioner, without significantly affecting the number of iterations.

This is even more evident in Table 4.2 where a more dense G_{out} is computed. Passing from `nband` $= 1$ (original FSAI) to `nband` $= 10000$, the setup time is halved, and also the solution time is reduced.

Table 4.2
Timings and iteration number for RFSAI1-PCG. Matrix GEO-1439.

G_{out}			`nband`	G_{in}			ρ_1	ρ_2	iter	T_{prec}	T_{sol}	T_{tot}
δ	d	ε		δ	d	ε						
0.05	4	0.1	10000	0.0	1	0.05	0.17	0.19	626	143.54	13.45	147.99
0.05	4	0.1					0.36		585	261.62	18.90	280.52

4.2. Comparisons between RFSAI2 and FSAI. In this section we compare the performances of RFSAI2 and FSAI for a fixed number of processors. Table 4.3 reports the results for matrix EMILIA-923. For this matrix we note that the best RFSAI requires three times less iterations than the best FSAI and this is accomplished by setting to 1 the value of `nband`. For this combination of parameters, however, there is a high setup CPU time. Regarding the time for the iterative solution only, the reduction provided by RFSAI is by a factor 2 or more.

Considering now matrix GEO-1438, whose results are summarized in Table 4.4 we may note that the only improvement provided by RFSAI2 is in terms of iteration count. However, the larger setup CPU time and the increased density of the resulting preconditioner makes our approach not competitive in terms of CPU time.

Matrix FAULT-639 is once again an example of the potential of the RFSAI2 approach. As it can be seen from Table 4.5, using `nband` $= 1$, RFSAI2 produces an important reduction of the

Table 4.3
Timings and iterations for FSAI and RFSAI2. Matrix EMILIA-923.

G_{out}			nband	G_{in}			ρ_1	ρ_2	iter	T_{prec}	T_{sol}	T_{tot}
δ	d	ε		δ	d	ε						
0.05	4	0.05					0.40		1634	85.60	41.52	127.12
0.1	4	0.05					0.35		1630	7.22	41.61	48.83
0.1	4	0.1					0.24		1695	7.36	37.68	45.04
0.01	2	0.05					0.25		2972	6.69	53.67	60.36
0.1	4	0.1	5000	0.01	1	0.05	0.16	0.24	994	9.23	19.92	29.15
0.1	4	0.1	1000	0.01	1	0.05	0.17	0.23	828	11.65	13.61	25.26
0.1	4	0.1	1	0.05	2	0.05	0.25	0.32	554	18.63	10.66	29.29
0.1	4	0.1	1	0.01	1	0.05	0.25	0.25	813	15.92	13.69	29.61

Table 4.4
Timings and iterations for FSAI and RFSAI2. Matrix GEO-1438.

G_{out}			nband	G_{in}			ρ_1	ρ_2	iter	T_{prec}	T_{sol}	T_{tot}
δ	d	ε		δ	d	ε						
0.1	4	0.1					0.34		585	20.37	11.89	32.25
0.1	2	0.1					0.21		766	1.18	24.94	26.12
0	2	0.05					0.49		629	13.66	15.76	29.32
0.1	4	0.1	1000	0.05	1	0.05	0.21	0.27	509	23.00	18.99	41.99
0.1	4	0.1	1	0.05	2	0.05	0.34	0.33	382	52.90	17.05	69.95
0.1	4	0.1	1	0.01	1	0.05	0.34	0.33	403	39.28	18.12	57.40

number of iterations as well as a smaller CPU time of the PCG solver We note that the density of the resulting preconditioners remains very low. Here RFSAI2 is the winner also considering the total CPU time which is reduced from 8.23 (Best FSAI) to 6.41 (Best RFSAI2). The last two rows in Table 4.5 consider using `nband` = 1000 parameter. There are no important differences between the case `nband` = 1 and the case `nband` = 1000. However, as expected, using `nband` = 1000 the setup time slightly reduces, due to the decreased complexity of the preconditioner, while the number of iterations increases.

Table 4.5
Timings and iterations for FSAI and RFSAI2. Matrix FAULT-639.

G_{out}			nband	G_{in}			ρ_1	ρ_2	iter	T_{prec}	T_{sol}	T_{tot}
δ	d	ε		δ	d	ε						
0.1	4	0.01					1.32		673	5.07	9.62	14.69
0.2	4	0.01					0.18		986	0.30	8.61	8.91
0.2	4	0.1					0.11		1022	0.32	7.91	8.23
0.2	2	0.01					0.10		1667	0.26	21.86	22.02
0	2	0.01					1.41		938	6.42	13.34	19.76
0.2	4	0.1	1	0.1	4	0.1	0.11	0.14	390	6.90	3.72	10.62
0.2	4	0.01	1	0.1	4	0.1	0.18	0.14	377	9.34	5.38	14.72
0.2	4	0.1	1	0.05	2	0.05	0.11	0.33	349	4.75	3.64	8.39
0.2	4	0.1	1	0.1	2	0.05	0.11	0.17	423	2.42	3.99	6.41
0.2	4	0.1	1000	0.1	4	0.1	0.06	0.19	760	6.72	9.44	16.16
0.2	4	0.01	1000	0.1	4	0.1	0.09	0.18	622	7.83	5.82	13.65

4.3. Parallel results and scalability. We investigate in this Section the parallel efficiency of the PCG solver preconditioned by FSAI and RFSAI. We will use a strong scaling measure to see how CPU times vary with the number of processors for a fixed total problem size. Denoting with T_p the total CPU elapsed times expressed in seconds on p processors, we define relative measures of the parallel efficiency and speedup of our code. We define as $S_p^{(\bar{p})}$ the pseudo speedup computed with respect to the smallest number of processors ($\bar{p}$) used to solve a given problem and $E_p^{(\bar{p})}$ the corresponding efficiency:

$$S_p^{(\bar{p})} = \frac{T_{\bar{p}}\bar{p}}{T_p}, \qquad E_p^{(\bar{p})} = \frac{S_p^{(\bar{p})}}{p} = \frac{T_{\bar{p}}\bar{p}}{T_p p}.$$

Table 4.6
Timings, speedups and efficiencies of FSAI-PCG (top table) and RFSAI2-PCG (bottom table) in the solution of problem EMILIA-923.

	p	T_{prec}	T_{sol}	T_{tot}	$S_p^{(2)}$	$E_p^{(2)}$
FSAI-PCG	4	17.0	140.6	157.7		
	8	10.5	61.6	72.2	8.7	1.09
	16	7.4	37.0	44.4	14.2	0.89
	32	5.3	19.3	24.6	25.6	0.80
	64	3.6	9.4	13.0	48.4	0.76
	128	2.7	5.8	8.5	74.6	0.58
RFSAI2-PCG	4	35.8	44.2	80.0		
	8	19.7	30.1	49.8	6.4	0.80
	16	11.6	13.6	25.3	12.7	0.79
	32	7.7	9.5	17.1	18.7	0.58
	64	5.2	6.6	11.7	27.3	0.43
	128	3.5	4.5	8.0	40.1	0.31

Fig. 4.1. Number of communicating processors vs process ID for A, G_{out} and G_{in} multiplications times a vector in problem EMILIA-923 when using $p = 128$ processors.

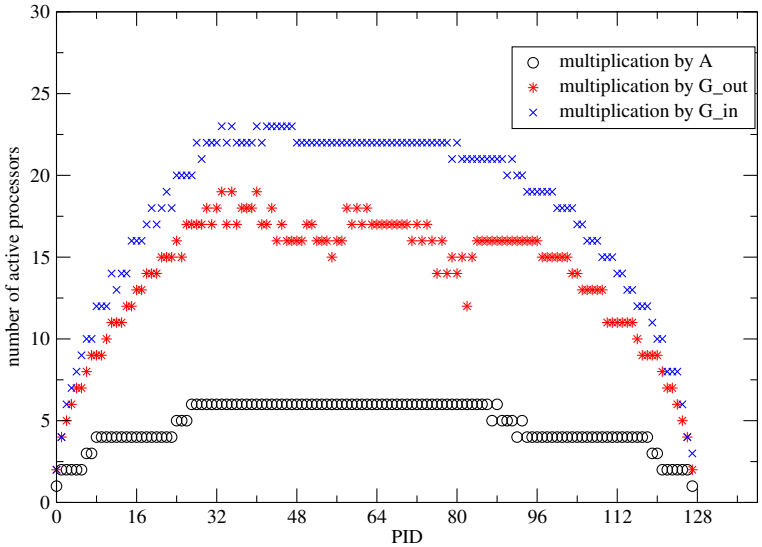
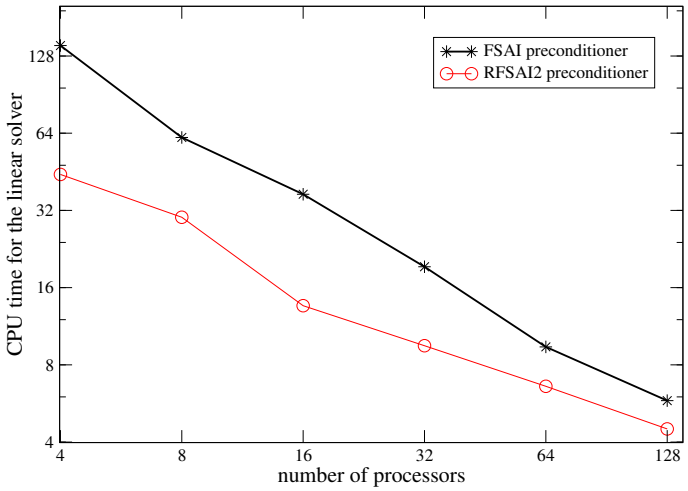


Table 4.7 reports in summary the scalability analysis in solving problem EMILIA-923 with

FSAI-PCG and RFSAI2-PCG, respectively. The parameters used are: $d = 4, \delta = 0.1$ and $\varepsilon = 0.1$ for G_{out} in both cases. RFSAI2 uses `nband` $= 1000$ and $\delta = 0.01$, $d = 1$ and $\varepsilon = 0.05$ for the G_{in} factor.

Fig. 4.2. T_{sol} for FSAI and RFSAI2 in solving problem EMILIA-923 vs the number of processors.



This test case is very hard from the point of view of communication among processors. In the setup phase as well as in the iterative phase each processor has to exchange data with a very high number of other processors. As an example, to construct factor G_{in} with $p = 128$, each processor interacts from 2 to 23 other processors, depending on its PID number. For this reason the scalability of RFSAI2 is worst than that of FSAI, mostly in the iterative phase.

Table 4.7
Timings, speedups and efficiencies of FSAI-PCG (top table) and RFSAI1-PCG (bottom table) in the solution of problem GEO-1438.

	p	T_{prec}	T_{sol}	T_{tot}	$S_p^{(2)}$	$E_p^{(2)}$
FSAI-PCG		74.4	41.7	116.1		
	8	44.0	21.4	65.4	7.1	0.89
	16	20.4	11.9	32.3	14.4	0.90
	32	11.1	8.3	19.4	23.9	0.75
	64	6.6	4.6	11.2	41.5	0.65
	128	3.1	2.4	5.5	84.9	0.66
	256	1.9	1.6	3.5	132.3	0.52
RFSAI1-PCG	4	73.4	64.0	137.4		
	8	42.5	24.3	66.8	8.2	1.03
	16	19.1	12.7	31.8	17.3	1.08
	32	10.1	12.4	22.6	24.3	0.76
	64	6.0	4.2	10.1	54.4	0.85
	128	3.0	2.4	5.4	101.0	0.79
	256	2.0	1.9	3.8	143.5	0.56

Figure 4.1 gives evidence of the amount of communication among processors. For $p = 128$ we plot the number of active processors vs processor identification index (PID). Application of both factors G_{in} and G_{out} at each iteration of the PCG solver requires a far higher number of interactions between processors with respect to the product by coefficient matrix A, with

G_{in} slightly more demanding in terms of data exchange. However, even using 128 processor, RFSAI2 CPU time for the iterative part remains significantly smaller (4.5 vs 5.8 seconds) with respect to the FSAI preconditioner. This is also accounted by Figure 4.2 where the T_{sol} times are plotted for both the codes.

In Table 4.7 we report the scalability analysis in the solution of problem GEO-1438 with FSAI-PCG and RFSAI1-PCG, respectively. The parameters used are: $d = 4, \delta = 0.1$ and $\varepsilon = 0.1$ for G_{out} in both cases. RFSAI1 uses `nband` $= 1000$ and $\varepsilon = 0.05$ for the G_{in} factor. We note from Table 4.7 that our code scales very well up to 256 processors. The parallel efficiency is always larger than 50% for both codes, and this is true for both setup and iteration phases.

Differently from the previous scalability case, here we do not note any significant difference between parallel efficiencies of FSAI and RFSAI1. This is so since the G_{in} preconditioner is banded and hence its nonzeros are very close to the main diagonal. In fact, the number of active processors in the application of G_{in} are in this case only 2, irrespective of the number of processors employed.

5. RFSAI as a preconditioner for eigensolvers. We report in this section the results of an eigenvalue solver based on PCG optimization, accelerated with the RFSAI preconditioner. As the eigenvalue solver we choose DACG [4] which seeks the leftmost eigenvalues of an SPD matrix, sequentially, by minimizing the Rayleigh Quotient

$$q(\boldsymbol{x}) = \frac{\boldsymbol{x}^T A \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$$

onto a subspace orthogonal to the previously computed eigenvectors. Sequential DACG has been shown competitive with the Jacobi-Davidson method or the ARPACK package [9]. More recently, a parallel implementation of FSAI-DACG has been successfully compared in [8] with the LOBPCG method [14] as implemented within the HYPRE package.

The preconditioner is computed only once as an FSAI (RFSAI) inverse approximation of A. For this reason, we expect that the gap between FSAI and RFSAI2 will grow, depending on the number of eigenpairs being sought, since the cost of the preconditioner setup (on the average larger for RFSAI2) is only a small percentage of the overall CPU time.

We report the results of FSAI-DACG and RFSAI2-DACG in the computation of the 10 leftmost eigenpairs of matrices EMILIA-923 and FAULT-679. In Tables 5.1 and 5.2 we provide the outcome of a number of runs obtained by varying the FSAI parameters, keeping the number of processors to a constant value of 16. It is shown that RFSAI2 acceleration provides on the average a reduction of three times the number of iteration and twice the overall CPU time for problem EMILIA-923. As for the FAULT-639 matrix, the reduction is more moderate (40% the iteration number and 20% the overall CPU time).

Table 5.1

Timings and iterations for FSAI and RFSAI2. Computation of the 10 leftmost eigenpairs of matrix EMILIA-923 using 16 processors.

G_{out}			`nband`	G_{in}			ρ_1	ρ_2	iter	T_{prec}	T_{sol}	T_{tot}
δ	d	ε		δ	d	ε						
0.1	4	0.05					0.35		17537	9.03	426.40	435.56
0.2	4	0.1					0.21		19570	0.81	520.44	521.38
0.1	4	0.1					0.26		19268	9.12	433.10	442.22
0.05	2	0.05					0.24		31324	2.19	692.16	694.45
0.1	4	0.1	1000	0.01	1	0.05	0.17	0.24	8693	13.64	236.27	250.59
0.1	4	0.1	1	0.05	2	0.05	0.26	0.30	6165	21.01	225.39	247.12

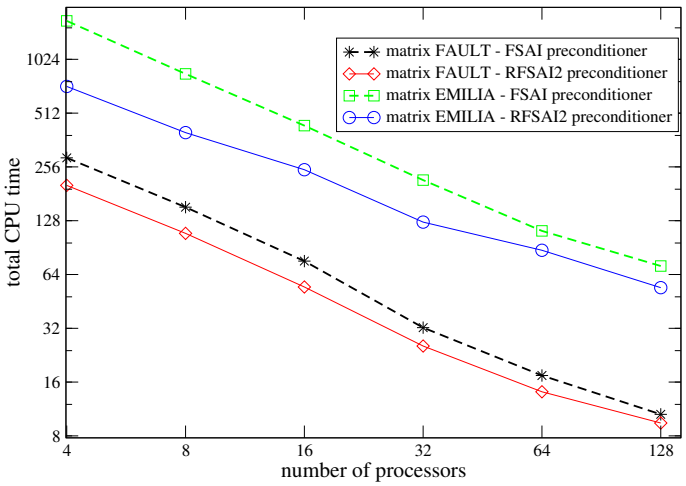
We finally report in Figure 5.1 the scalability results obtained using the best runs (in terms of CPU time) taken from the previous tables. In this figure we plot the total CPU time vs

Table 5.2
Timings and iterations for FSAI and RFSAI2. Computation of the 10 leftmost eigenpairs of matrix FAULT-639.

G_{out}			nband	G_{in}			ρ_1	ρ_2	iter	T_{prec}	T_{sol}	T_{tot}
δ	d	ε		δ	d	ε						
0.1	4	0.1					0.24		4448	6.63	67.78	74.50
0.0	2	0.05					0.40		5323	8.13	88.07	96.30
0.2	4	0.15	1	0.1	4	0.1	0.09	0.16	2916	11.69	47.03	59.07
0.2	4	0.1	1	0.05	2	0.05	0.11	0.33	2714	4.81	49.31	54.48

number of processors, showing that RFSAI2 is more efficient than FSAI irrespective on the number of processor employed for both the test cases.

Fig. 5.1. Scalability of FSAI-DACG and RFSAI2-DACG in solving problems FAULT-639 and EMILIA-923. The total CPU time is plotted vs the number of processors.



6. Conclusions. We have proposed and developed a recursive preconditioner (RFSAI) with the aim of improving the efficiency of the original FSAI preconditioner. Two variants are described in this paper: RFSAI1 and RFSAI2. Numerical results in the solution of large, realistic and ill-conditioned matrices arising from discretization of geomechanical models reveal that RFSAI1 produces on the average a very cheap and perfectly scalable preconditioner. This approach can be used to reduce the setup time without significantly affecting the number of iterations. The second approach, RFSAI2, is more powerful since it is shown to greatly reduce the number of iterations and the iterative solution phase CPU time though at the price of

1. an increased complexity of the setup phase and
2. a possible reduction of the parallel efficiency due to the increased number of nonzero entries of the preconditioner which are far away from the main diagonal.

As a preconditioner for the DACG eigenvalue solver, where the cost of preconditioner setup phase is relatively less important, RFSAI2 reveals more efficient than FSAI also in terms of total CPU time.

Acknowledgments. We acknowledge the CINECA Iscra Award SCALPREC (2011) for the availability of HPC resources and support.

REFERENCES

[1] M. Benzi, J. K. Cullum, and M. Tůma, *Robust approximate inverse preconditioning for the conjugate gradient method*, SIAM J. Sci. Comput., 22 (2000), pp. 1318–1332.

[2] M. Benzi, C. D. Meyer, and M. Tůma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., 17 (1996), pp. 1135–1149.

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

[4] L. Bergamaschi, G. Gambolati, and G. Pini, *Asymptotic convergence of conjugate gradient methods for the partial symmetric eigenproblem*, Numer. Lin. Alg. Appl., 4 (1997), pp. 69–84.

[5] L. Bergamaschi and A. Martínez, *Parallel acceleration of Krylov solvers by factorized approximate inverse preconditioners*, in VECPAR 2004, M. Daydè et al., ed., vol. 3402 of Lecture Notes in Computer Sciences, Heidelberg, 2005, Springer-Verlag, pp. 623–636.

[6] ———, *Parallel inexact constraint preconditioners for saddle point problems*, in Euro-Par 2011, Bordeaux (France), R. N. E. Jeannot and J. Roman, eds., vol. 6853, Part II of Lecture Notes in Computer Sciences, Heidelberg, 2011, Springer, pp. 78–89.

[7] L. Bergamaschi, A. Martínez, and G. Pini, *An efficient parallel MLPG method for poroelastic models*, CMES: Computer and Modeling in Engineering & Sciences, 49 (2009), pp. 191–216.

[8] ———, *Parallel Rayleigh Quotient optimization with FSAI-based preconditioning*, J. Appl. Math., (2011). Submitted.

[9] L. Bergamaschi and M. Putti, *Numerical comparison of iterative eigensolvers for large sparse symmetric matrices*, Comp. Methods App. Mech. Engrg., 191 (2002), pp. 5233–5247.

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

[11] R. M. Holland, A. J. Wathen, and G. J. Shaw, *Sparse approximate inverses and target matrices*, SIAM J. Sci. Comput., 26 (2005), pp. 1000–1011 (electronic).

[12] C. Janna and M. Ferronato, *Adaptive pattern research for block FSAI preconditioning*, SIAM J. Sci. Comput., 33 (2011), pp. 3357–3380.

[13] 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.

[14] A. Knyazev, *Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method*, SIAM J. Sci. Comput., 23 (2001), pp. 517–541.

[15] L. Yu. Kolotilina, A. A. Nikishin, and A. Yu. Yeremin, *Factorized sparse approximate inverse preconditionings. IV: Simple approaches to rising efficiency*, Numer. Lin. Alg. Appl., 6 (1999), pp. 515–531.

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

[17] A. Martínez, L. Bergamaschi, M. Caliari, and M. Vianello, *A massively parallel exponential integrator for advection-diffusion models*, J. Comput. Appl. Math., 231 (2009), pp. 82–91.