Parallel Rayleigh Quotient optimization with FSAI-based preconditioning

Luca Bergamaschi, Ángeles Martínez, and Giorgio Pini
Dept. Mathematical Methods and Models for Scientific Applications
University of Padova, via Trieste 63, 35121 Padova, Italy
e-mail luca.bergamaschi@unipd.it, angeles.martinez@unipd.it,
giorgio.pini@unipd.it

Abstract

The present paper describes a parallel preconditioned algorithm for the solution of partial eigenvalue problems for large sparse symmetric matrices, on parallel computers. Namely, we consider the Deflation-Accelerated Conjugate Gradient (DACG) algorithm accelerated by factorized sparse approximate inverse (FSAI) type preconditioners. We present an enhanced parallel implementation of the FSAI preconditioner and made use of the recently developed Block FSAI-IC preconditioner, which combines the FSAI and the Block Jacobi-IC preconditioners. Results onto matrices of large size arising from Finite Element discretization of geomechanical models reveals that DACG accelerated by these type of preconditioners is competitive with respect to the available public parallel *hypre* package, especially in the computation of a few of the leftmost eigenpairs. The parallel DACG code accelerated by FSAI is written in MPI–Fortran 90 language and exhibits good scalability up to one thousand processors.

1 Introduction

The computation by iterative methods of the s leftmost partial eigenspectrum of the generalized eigenproblem:

$$A\mathbf{u} = \lambda B\mathbf{u} \tag{1}$$

where $A, B \in \mathbb{R}^{n\times n}$ are large sparse symmetric positive definite (SPD) matrices, is an important and difficult task in many applications. It has become increasingly wide spread owing to the development in the last twenty years of robust and computationally efficient schemes and corresponding software packages. Among the most well known approaches for the important class of symmetric positive definite (SPD) matrices are the implicitly restarted Arnoldi method (equivalent to the Lanczos technique for this type of matrices) [2, 19], the Jacobi-Davidson (JD) algorithm [21] and schemes based on preconditioned Conjugate Gradient minimization of the Rayleigh quotient [3, 16].

The basic idea of the later is to minimize the Rayleigh quotient:

$$r(\mathbf{x}) = \frac{\mathbf{x}^T A\mathbf{x}}{\mathbf{x}^T B\mathbf{x}} \tag{2}$$

in a subspace which is orthogonal to the previously computed eigenvectors. via a preconditioned CG-like procedure. Among the different variants of this technique we chose to use the Deflation-Accelerated Conjugate Gradient (DACG) scheme [3, 7] which has been shown to be competitive with the Jacobi Davidson method and with the PARPACK package [8]. As in any other approach, for our DACG method, the choice of the preconditioning technique is a key factor to accelerate, and in some cases even to allow for, convergence. To accelerate DACG in a parallel environment we selected the Factorized Sparse Approximate inverse (FSAI) preconditioner introduced in [18]. We have developed a parallel implementation of this algorithm which has displayed excellent performances on both the setup phase and the application phase within a Krylov subspace solver [4, 6, 5]. The effectiveness of the FSAI preconditioner in the acceleration of DACG is compared to that of the Block FSAI-IC preconditioner, recently developed in [13], which combines the FSAI and the Block Jacobi-IC preconditioners obtaining good results on a small number of processors for the solution of SPD linear systems and for the solution of large eigenproblems [11]. We used the resulting parallel codes to compute a few of the leftmost eigenpairs of a set of test matrices of large size arising from Finite Element discretization of geomechanical models. The reported results show that DACG preconditioned with either FSAI or BFSAI is a scalable and robust algorithm for the partial solution of SPD eigenproblems. The parallel performance of DACG is also compared to that of the publicly available parallel package *hypre* [1] which implements a number of preconditioners which can be used in combination with the Locally Optimal Block PCG (LOBPCG) iterative eigensolver [14]. The results presented in this paper shows that the parallel DACG code accelerated by FSAI exhibits good scalability up to one thousand processors, and displays comparable performance with respect to *hypre*, specially when a low number of eigenpairs is sought.

The outline of the paper is as follows: in Section 2 we describe the DACG Algorithm; in Section 3 and 4 we recall the definition and properties of the FSAI and BFSAI preconditioners, respectively. Section 5 contains the numerical results obtained with the proposed algorithm in the eigensolution of very large SPD matrices of size up to almost 7 million unknowns and 3×10^8 nonzeros. A comparison with the *hypre* eigensolver code is also included. Section 6 ends the paper with some conclusions.

2 The DACG iterative eigensolver and implementation

The DACG algorithm sequentially computes the eigenpairs, starting from the leftmost one $(\lambda_1, \boldsymbol{u}_1)$. To evaluate the j-th eigenpair, $j > 1$, DACG minimizes the Rayleigh Quotient (RQ) in a subspace orthogonal to the $j-1$ eigenvectors previously computed. More precisely, DACG minimizes the Rayleigh Quotient:

$$q(\boldsymbol{z}) = \frac{\boldsymbol{z}^T A\boldsymbol{z}}{\boldsymbol{z}^T\boldsymbol{z}}, \tag{3}$$

where

$$\boldsymbol{z} = \boldsymbol{x} - U_j \left(U_j^T \boldsymbol{x}\right), \qquad U_j = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_{j-1}], \ \boldsymbol{x} \in R^n.$$

The first eigenpair $(\lambda_1, \boldsymbol{u}_1)$ is obtained by minimization of (3) with $\boldsymbol{z} = \boldsymbol{x}$ ($U_1 = \emptyset$). Indicating with M the preconditioning matrix, i. e. $M \approx A^{-1}$, the s leftmost eigenpairs are computed by the conjugate gradient procedure [7] described in Algorithm 1.

Algorithm 1 DACG Algorithm.

Choose tolerance ε, set $U = 0$.

DO $j = 1, s$

1. Choose $\boldsymbol{x}_0$ such that $U^T \boldsymbol{x}_0 = 0$; set $k = 0$, $\beta_0 = 0$;
2. $\boldsymbol{x}_0^A = A\boldsymbol{x}_0, \quad \gamma = \boldsymbol{x}_0^T \boldsymbol{x}_0^A, \quad \eta = \boldsymbol{x}_0^T \boldsymbol{x}_0, \quad q_0 \equiv q(\boldsymbol{x}_0) = \gamma/\eta, \quad \boldsymbol{r}_0 = \boldsymbol{x}_0^A - q_0 \boldsymbol{x}_0$;
3. REPEAT
   - 3.1 $\boldsymbol{g}_k \equiv \nabla q(\boldsymbol{x}_k) = \dfrac{2}{\eta} \boldsymbol{r}_k$;
   - 3.2 $\boldsymbol{g}_k^M = M \boldsymbol{g}_k$;
   - 3.3 IF $k > 0$ THEN $\beta_k = \dfrac{\boldsymbol{g}_k^T (\boldsymbol{g}_k^M - \boldsymbol{g}_{k-1}^M)}{\boldsymbol{g}_{k-1}^T \boldsymbol{g}_{k-1}^M}$;
   - 3.4 $\tilde{\boldsymbol{p}}_k = \boldsymbol{g}_k^M + \beta_k \boldsymbol{p}_{k-1}$;
   - 3.5 $\boldsymbol{p}_k = \tilde{\boldsymbol{p}}_k - U \left(U^T \tilde{\boldsymbol{p}}_k\right)$
   - 3.6 $\boldsymbol{p}_k^A = A \boldsymbol{p}_k$,
   - 3.7 $\alpha_k = \underset{t}{\operatorname{argmin}} \{q(\boldsymbol{x}_k + t\boldsymbol{p}_k)\} = \dfrac{\eta d - \gamma b + \sqrt{\Delta}}{2(bc - ad)}$, with
     $a = \boldsymbol{p}_k^T \boldsymbol{x}_k^A, \qquad b = \boldsymbol{p}_k^T \boldsymbol{p}_k^A, \qquad c = \boldsymbol{p}_k^T \boldsymbol{x}_k, \qquad d = \boldsymbol{p}_k^T \boldsymbol{p}_k,$
     $\Delta = (\eta d - \gamma b)^2 - 4(bc - ad)(\gamma a - \eta c)$;
   - 3.8 $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k, \quad \boldsymbol{x}_{k+1}^A = \boldsymbol{x}_k^A + \alpha_k \boldsymbol{p}_k^A$;
   - 3.9 $\gamma = \gamma + 2a\,\alpha_k + b\,\alpha_k^2, \qquad \eta = \eta + 2c\,\alpha_k + d\,\alpha_k^2$;
   - 3.10 $q_{k+1} \equiv q(\boldsymbol{x}_{k+1}) = \gamma/\eta$;
   - 3.11 $k = k + 1$;
   - 3.12 $\boldsymbol{r}_k = \boldsymbol{x}_k^A - q_k \boldsymbol{x}_k$

   UNTIL $\dfrac{q_{k+1} - q_k}{q_{k+1}} < tol$;
4. $\lambda_j = q_k, \qquad \boldsymbol{u}_j = \boldsymbol{x}_k / \sqrt{\eta}, \qquad U = [U, \boldsymbol{u}_j]$.

END DO

The schemes relying on the Rayleigh quotient optimization are quite attractive for parallel computations, however preconditioning is an essential feature to ensure practical convergence. When seeking for an eigenpair $(\lambda_j, \boldsymbol{u}_j)$ it can be proved that the number of iterations is proportional to the square root of the condition number $\xi_j = \kappa(H_j)$ of the Hessian of the Rayleigh Quotient in the stationary point $\mathbf{u}_j$ [3]. It

turns out that H_j is similar to $(A-\lambda_j I)M$ which is not SPD. However, H_j operates on the orthognal space spanned by the previous eigenvectors, so that the only important eigenvalues are the positive ones. In the non-preconditioned case (i.e. $M = I$) we would have

$$\kappa(H_j) \approx \frac{\lambda_N}{\lambda_{j+1} - \lambda_j}$$

where in the ideal case $M \equiv A^{-1}$, we have

$$\kappa(H_j) \approx \frac{\lambda_j}{\lambda_{j+1} - \lambda_j} = \xi_j \ll \frac{\lambda_N}{\lambda_{j+1} - \lambda_j}$$

Therefore, even though A^{-1} is not the optimal preconditioner for $A - \lambda_j I$, however, if M is a good preconditioner of A then the condition number $\kappa(H_j)$ will approach ξ_j.

3 The FSAI Preconditioner

The FSAI preconditioner, initially proposed in [17] and [18], has been later developed and implemented in parallel by Bergamaschi et al. in [4]. Here, we only shortly recall the main features of this preconditioner. Given and SPD matrix A the FSAI preconditioner approximately factorize its inverse as a product of two sparse triangular matrices as

$$A^{-1} \approx W^T W.$$

The choice of nonzeros in W are based on a sparsity pattern which in our work may be the same as $\tilde{A}^d$ where $\tilde{A}$ is the result of *prefiltration* [6] of A i.e. dropping of all elements below of a threshold parameter δ. The entries of W are computed by minimizing the Frobenius norm of $I - WL$ where L is the exact Cholesky factor of A, without forming explicitly the matrix L. The computed W is then sparsified by dropping all the elements which are below a second tolerance parameter (ε). The final FSAI preconditioner is therefore related to the following three parameters: δ, prefiltration threshold; d, power of A generating the sparsity pattern (we allow $d \in \{1, 2, 4\}$ in our experiments); ε, postfiltration threshold.

3.1 Parallel implementation of FSAI-DACG.

We have developed a parallel code written in FORTRAN 90 and which exploits the MPI library for exchanging data among the processors. We used a block row distribution of all matrices (A, W and W^T), that is, with complete rows assigned to different processors. All these matrices are stored in static data structures in CSR format.

Regarding the preconditioner computation, we stress that any row i of matrix W of FSAI preconditioner is computed independently of each other, by solving a small SPD dense linear system of size n_i equal to the number of nonzeros allowed in row i of W. Some of the rows which contribute to form this linear system may be non local to processor i and should be received from other processors. To this aim we implemented a routine called *get_extra_rows* which carries out all the row exchanges among the processors, before starting the computation of W, which proceed afterwards entirely in

parallel. Since the number of non local rows needed by each processor is relatively small we chose to temporarily replicate these rows on auxiliary data structures. Once W is obtained a parallel transposition routine provides every processor with its part of W^T.

The DACG iterative solver is essentially based on scalar and matrix-vector products. We made use of an optimized parallel matrix-vector product which has been developed in [20] showing its effectiveness up to 1024 processors.

4 Block FSAI-IC preconditioning

The Block FSAI-IC preconditioner, BFSAI-IC in the following, is a recent development for the parallel solution to Symmetric Positive Definite (SPD) linear systems. Assume that D is an arbitrary non-singular block diagonal matrix consisting of n_b equal size blocks.

Let $\mathcal{S}_L$ and $\mathcal{S}_{BD}$ be a sparse lower triangular and a dense block diagonal non-zero pattern, respectively, for a $n \times n$ matrix. Even though not strictly necessary, for the sake of simplicity assume that $\mathcal{S}_{BD}$ consists of n_b diagonal blocks with equal size $m = n/n_b$ and let $D \in \mathbb{R}^{n \times n}$ be an arbitrary full-rank matrix with non-zero pattern $\mathcal{S}_{BD}$.

Consider the set of lower block triangular matrices F with a prescribed non-zero pattern S_{BL} and minimize over F the Frobenius norm:

$$\|D - FL\|_F \tag{4}$$

where L is the exact lower Cholesky factor of an SPD matrix A. A matrix F satisfying the minimality condition (4) for a given D is the lower block triangular factor of BFSAI-IC. Recalling the definition of the classical FSAI preconditioner, it can be noticed that BFSAI-IC is a block generalization of the FSAI concept.

The differentiation of (4) with respect to the unknown entries $[F]_{ij}$, $(i,j) \in S_{BL}$, yields the solution to n independent dense subsystems which do not require the explicit knowledge of L. The effect of applying F to A is to concentrate the largest entries of the preconditioned matrix FAF^T into n_b diagonal blocks. However, as D is arbitrary, it is still not ensured that FAF^T is better than A in an iterative method, so it is necessary to precondition FAF^T again. As FAF^T resembles a block diagonal matrix, an efficient technique relies on using a block diagonal matrix which collects an approximation of the inverse of each diagonal block B_{i_b} of FAF^T.

It is easy to show that F is guaranteed to exist with SPD matrices and B_{i_b} is SPD, too [13]. Using an IC decomposition with partial fill-in for each block B_{i_b} and collecting in J the lower IC factors, the resulting preconditioned matrix reads:

$$J^{-1}FAF^TJ^{-T} = WAW^T \tag{5}$$

with the final preconditioner:

$$M = W^TW = F^TJ^{-T}J^{-1}F \tag{6}$$

M in equation (6) is the BFSAI-IC preconditioner of A.

For its computation BFSAI-IC needs the selection of n_b and $\mathcal{S}_{\overline{L}}$. The basic requirement for the number of blocks n_b is to be larger than or equal to the number of computing cores p. From a practical viewpoint, however, the most efficient choice in terms of both wall clock time and iteration count is to keep the blocks as large as possible, thus implying $n_b = \text{p}$. Hence, n_b is by default set equal to p. By distinction, the choice of $\mathcal{S}_{\overline{L}}$ is theoretically more challenging and still not completely clear. A widely accepted option for other approximate inverses, such as FSAI or SPAI, is to select the non-zero pattern of A^d for small values of d on the basis of the Neumann series expansion of A^{-1}. Using a similar approach, in the BFSAI construction we select $\mathcal{S}_{\overline{L}}$ as the lower block triangular pattern of A^d. As the nonzeros located in the diagonal blocks are not used for the computation of F a larger value of d, say 3 or 4, can still be used.

Though theoretically not necessary, three additional user-specified parameters are worth introducing in order to better control the memory occupation and the BFSAI-IC density:

1. ε is a post-filtration parameter that allows for dropping the smallest entries of F. In particular, $[F]_{ij}$ is neglected if $[F]_{ij} < \varepsilon \|\mathbf{f}_i\|_2$, where $\mathbf{f}_i$ is the ith row of F;

2. ρ_B is a parameter that controls the fill-in of B_{i_b} and determines the maximum allowable number of nonzeros for each row of B_{i_b} in addition to the corresponding entries of A. Quite obviously, the largest ρ_B entries only are retained;

3. ρ_L is a parameter that controls the fill-in of each IC factor $\tilde{L}_{i_b}$ denoting the maximum allowable number of nonzeros for each row of $\tilde{L}_{i_b}$ in addition to the corresponding entries of B_{i_b}.

An OpenMP implementation of the algorithms above is available in [12].

5 Numerical Results

In this section we examine the performance of the parallel DACG preconditioned by both FSAI and BFSAI in the partial solution of four large size sparse eigenproblems. The test cases, which we briefly describe below, are taken from different real engineering mechanical applications. In detail,

- FAULT-639: is obtained from a structural problem discretizing a faulted gas reservoir with tetrahedral Finite Elements and triangular Interface Elements [10]. The Interface Elements are used with a Penalty formulation to simulate the faults behavior. The problem arises from a 3D discretization with three displacement unknowns associated to each node of the grid.

- PO-878: arises in the simulation of the consolidation of a real gas reservoir of the Po Valley, Italy, used for underground gas storage purposes (for details, see [9]).

- GEO-1438: is obtained from a geomechanical problem discretizing a region of the earth crust subject to underground deformation. 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. The problem arises from a 3D discretization with three displacement unknowns associated to each node of the grid [22].

- CUBE-6091: arises from the equilibrium of a concrete cube discretized by a regular unstructured tetrahedral grid.

Matrices FAULT-639 and GEO-1438 are publicly available in the University of Florida Sparse Matrix Collection at `http://www.cise.ufl.edu/research/sparse/matrices`.

Table 1: Size, number of nonzeros and three representative eigenvalues of the test matrices.

	Size	Nonzeros	λ_1	λ_{10}	λ_N
FAULT-639	638,802	28,614,564	$6.99 \cdot 10^6$	$1.73 \cdot 10^7$	$2.52 \cdot 10^{16}$
PO-878	878 355	38 847 915	$1.46 \cdot 10^6$	$4.45 \cdot 10^6$	$5.42 \cdot 10^{15}$
GEO-1438	1,437,960	63,156,690	$7.81 \cdot 10^5$	$1.32 \cdot 10^6$	$1.11 \cdot 10^{13}$
CUBE-6091	6,091,008	270,800,586	$1.82 \cdot 10^1$	$3.84 \cdot 10^2$	$1.05 \cdot 10^{07}$

The computational performance of FSAI is compared to the one obtained by using BFSAI as implemented in [13]. The comparison is done evaluating the number of iterations n_{iter} to converge at the same tolerance, the wall clock time in seconds T_{prec} and T_{iter} for the preconditioner computation and the eigensolver to converge, respectively, with the total time $T_{tot} = T_{prec} + T_{iter}$. 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 FSAI-DACG code is written in Fortran 90 and compiled with `-O4 -q64 -qarch=pwr6 -qtune=pwr6 -qnoipa -qstrict -bmaxdata:0x70000000` options. For the BFSAI–IC code only an OpenMP implementation is presently available.

To study parallel performance we will use a strong scaling measure to see how the CPU times vary with the number of processors for a fixed total problem size. Denote with T_p the total CPU elapsed times expressed in seconds on p processors. We introduce a relative measure of the parallel efficiency achieved by the code, $S_p^{(\bar{p})}$, which is the pseudo speedup computed with respect to the smallest number of processors ($\bar{p}$) used to solve a given problem. Accordingly, we will denote $E_p^{(\bar{p})}$ the corresponding efficiency:

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

5.1 FSAI-DACG results

In this section we report the results of our FSAI-DACG implementation in the computation of the 10 leftmost eigenpairs of the 4 test problems. We used the exit test described in the DACG algorithm (see Fig. 1) with $tol = 10^{-10}$. The results are summarized in Table 2. As the FSAI parameters, we choose $\delta = 0.1$, $d = 4$ and $\varepsilon = 0.1$ for all the test matrices. This combination of parameters produces, on the average, the

best (or close to the best) performance of the iterative procedure. Note that the number of iterations does not change with the number of processors, for a fixed problem. The scalability of the code is very satisfactory in both the setup stage (preconditioner computation) and the iterative phase.

Table 2: Number of iterations, timings and scalability indices for FSAI-DACG in the computation of the 10 leftmost eigenpairs of the four test problems.

	p	iter	T_{prec}	T_{iter}	T_{tot}	$S_p^{(4)}$	$E_p^{(4)}$
	4	4448	25.9	261.4	287.3		
	8	4448	13.2	139.0	152.2	7.6	0.94
FAULT-639	16	4448	6.6	69.4	76.0	15.1	0.95
	32	4448	4.0	28.2	32.2	35.7	1.11
	64	4448	1.9	15.5	17.4	66.1	1.03
	128	4448	1.1	9.4	10.5	109.0	0.85
	4	5876	48.1	722.5	770.6		
	8	5876	25.2	399.8	425.0	7.3	0.91
PO-878	16	5876	11.4	130.2	141.6	21.8	1.36
	32	5876	6.8	65.8	72.5	42.5	1.33
	64	5876	4.1	30.1	34.1	90.3	1.41
	128	5876	1.9	19.1	21.0	146.8	1.15
	4	6216	90.3	901.5	991.7		
	8	6216	47.5	478.9	526.4	7.5	0.94
	16	6216	24.7	239.4	264.1	15.0	0.94
GEO-1437	32	6216	13.6	121.0	134.6	29.5	0.92
	64	6216	8.2	60.9	69.1	57.4	0.90
	128	6216	4.2	29.5	33.8	117.5	0.92
	256	6216	2.3	19.1	21.4	185.4	0.72
	p	iter	T_{prec}	T_{iter}	T_{tot}	$S_p^{(16)}$	$E_p^{(16)}$
	16	15796	121.5	2624.8	2746.2		
	32	15796	62.2	1343.8	1406.0	31.3	0.98
	64	15796	32.5	737.0	769.5	57.1	0.89
CUBE-6091	128	15796	17.3	388.4	405.7	108.3	0.85
	256	15796	9.1	183.9	192.9	227.8	0.89
	512	15796	5.7	106.0	111.7	393.5	0.77
	1024	15796	3.8	76.6	80.4	546.6	0.53

5.2 BFSAI-IC-DACG results

We present in this Section the results of DACG accelerated by the BFSAI-IC preconditioner for the approximation of the $s = 10$ leftmost eigenpairs of the matrices described above.

Table 3 provides iteration count and total CPU time for BFSAI-DACG with different combinations of the parameters needed to construct the BFSAI-IC preconditioner

Table 3: Performance of BFSAI-DACG for matrix PO-878 with 2 to 8 processors and different parameter values.

				p= 2		p= 4		p= 8	
d	ρ_B	ε	ρ_L	iter	T_{tot}	iter	T_{tot}	iter	T_{tot}
2	10	0.01	10	2333	385.76	2877	286.24	3753	273.77
2	10	0.05	10	2345	415.81	2803	245.42	3815	**142.93**
2	10	0.05	20	2186	**370.86**	2921	276.41	3445	257.18
2	10	0.00	10	2328	445.16	2880	241.23	3392	269.41
2	20	0.05	10	2340	418.20	2918	**224.32**	3720	253.98
3	10	0.05	10	2122	375.17	2638	228.39	3366	149.59
3	10	0.05	20	1946	433.04	2560	304.43	3254	263.51
3	10	0.05	30	1822	411.00	2481	321.30	3176	179.67
3	10	0.05	40	1729	439.47	2528	346.82	3019	188.13
4	10	0.05	10	2035	499.45	2469	350.03	3057	280.31

for matrix PO-878, and using from 2 to 8 processors. It can be seen from Table 3 that the assessment of the optimal parameters, ε, ρ_B and ρ_L, is not an easy task, since the number of iterations may highly vary depending on the number of processors. We chose in this case the combination of parameters producing the second smallest total time with $\mathrm{p} = 2, 4, 8$ processors. After intensive testing for all the test problems, we selected similarly the "optimal" values which are used in the numerical experiments reported in Table 4:

The user-specified parameters for BFSAI-IC given above provide evidence that it is important to build a dense preconditioner based on the lower non-zero pattern of A^3 (except for CUBE-6091, which is built on a regular discretization) with the aim at decreasing the number of DACG iterations. Anyway, the cost for computing such a dense preconditioner appears to be almost negligible with respect to the wall clock time needed to iterate to convergence.

Table 4: Number of iterations for BFSAI-DACG in the computations of the 10 leftmost eigenpairs.

	n_b									
matrix	2	4	8	16	32	64	128	256	512	1024
FAULT-639	1357	1434	1594	2002	3053	3336	3553			
PO-878	2122	2638	3366	4157	4828	5154	5373			
GEO-1438	1458	1797	2113	2778	3947	4647	4850	4996		
CUBE-6091			5857	6557	7746	8608	9443	9996	10189	9965

- FAULT-639: $d = 3$, $\varepsilon = 0.05$, $\rho_B = 10$, $\rho_L = 60$

- PO-878 $d = 3$, $\varepsilon = 0.05$, $\rho_B = 10$, $\rho_L = 10$
- GEO-1438: $d = 3$, $\varepsilon = 0.05$, $\rho_B = 10$, $\rho_L = 50$
- CUBE-6091: $d = 2$, $\varepsilon = 0.01$, $\rho_B = 0$, $\rho_L = 10$

We recall that, presently, the code BFSAI-IC is implemented in OpenMP, and the results in terms of CPU time are significant only for p $\leq$ 8. For this reason the number of iterations reported in Table 4 are obtained with increasing number of blocks n_b and with p = 8 processors. This iteration numbers accounts for a potential implementation of BFSAI-DACG under the MPI (or hybrid OpenMP – MPI) environment as the number of iterations depends only on the number of blocks, irrespective of the number of processors.

The only meaningful comparison between FSAI-DACG and BFSAI-DACG can be carried out in terms of iteration numbers which are smaller for BFSAI-DACG for a small number of processors. The gap between FSAI and BFSAI iterations reduces when the number of processors increases.

5.3 Comparison with the LOBPCG eigensolver provided by *hypre*

In order to validate the effectiveness of our preconditioning in the proposed DACG algorithm with respect to already available public parallel eigensolvers, the results given in Tables 2 and 4 are compared with those obtained by the schemes implemented in the *hypre* software package [1]. The Locally Optimal Block Preconditioned Conjugate Gradient method (LOBPCG) [14] is experimented with, using the different preconditioners developed in the *hypre* project, i.e. algebraic multigrid (AMG), diagonal scaling (DS), approximate inverse (ParaSails), additive Schwarz (Schwarz), and incomplete LU (Euclid). The *hypre* preconditioned CG is used for the inner iterations within LOBPCG. For details on the implementation of the LOBPCG algorithm, see for instance [15]. The selected preconditioner, ParaSails, is on its turn based on the FSAI preconditioner, so that the different FSAI-DACG and ParaSails-LOBPCG performances should be ascribed mainly to the different eigensolvers rather than to the preconditioners.

Table 5: Iterations and CPU time for the iterative solver of LOBPCG-*hypre* preconditioned by Parasails with different values of `bl` and $p = 16$ processors.

	`bl` = 10		`bl` = 11		`bl` = 12		`bl` = 15	
matrix	iter	T_{iter}	iter	T_{iter}	iter	T_{iter}	iter	T_{iter}
FAULT-639	156	79.5	157	85.3	157	96.1	160	128.1
PO-878	45	117.0	41	131.6	38	151.3	35	192.6
GEO-1438	23	123.7	72	173.7	30	152.5	121	291.1
CUBE-6091	101	1670.5	143	2414.0	38	1536.7	35	1680.9

We first carried out a preliminary set of runs with the aim of assessing the optimal value of the block size `bl` parameter, i.e. the size of the subspace where to seek for

the eigenvectors. Obviously it must be `bl` $\geq s = 10$. We fixed to 16 the number of processors and obtained the results summarized in Table 5 with different values of `bl` $\in [10, 15]$. We found that only in problem CUBE-6091, a value of `bl` larger than 10, namely `bl` $= 12$ yields an improvement in the CPU time. Note that we also made this comparison with different number of processors, and we obtain analogous results.

Table 6: Number of iterations, timings and scalability of LOBPCG-*hypre* preconditioned by Parasails.

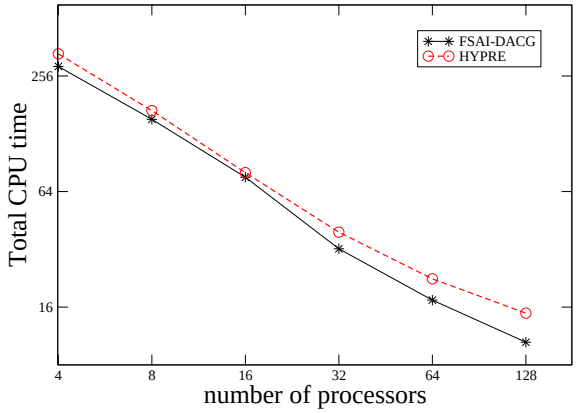
	p	iter	T_{prec}	T_{iter}	T_{tot}	$S_p^{(4)}$	$E_p^{(4)}$
	4	155	2.5	331.2	333.7		
	8	156	1.3	167.6	168.9	7.9	0.99
FAULT-639	16	156	0.8	79.5	80.3	16.6	1.04
pcgitr= 5	32	150	0.5	38.8	39.3	34.0	1.06
	64	145	0.3	22.2	22.5	59.4	0.93
	128	157	0.1	14.8	14.9	89.7	0.70
	4	45	3.3	438.4	441.7		
	8	50	1.3	232.3	234.0	7.6	0.94
PO-878	16	45	1.0	117.0	118.0	15.0	0.94
pcgitr = 30	32	45	0.7	63.2	63.9	27.6	0.86
	64	47	0.4	34.4	34.8	50.8	0.79
	128	41	0.3	19.44	19.74	89.5	0.70
	4	26	7.7	478.0	485.7		
	8	22	4.0	256.8	260.8	7.5	0.93
GEO-1438	16	23	2.1	123.7	125.8	15.4	0.96
pcgitr = 30	32	28	1.2	73.1	74.3	26.2	0.82
	64	23	0.8	35.5	36.3	53.5	0.84
	128	25	0.5	20.3	20.8	93.2	0.73
	256	26	0.3	12.9	13.2	147.2	0.57
	p	iter	T_{prec}	T_{iter}	T_{tot}	$S_p^{(16)}$	$E_p^{(16)}$
	16	38	9.2	1536.7	1545.9		
	32	36	4.7	807.5	812.2	30.5	0.95
CUBE-6091	64	38	3.2	408.2	411.4	60.1	0.94
	128	41	1.6	251.4	253.0	97.8	0.76
	256	35	0.9	105.9	106.8	231.6	0.90
	512	39	0.6	65.3	65.9	375.3	0.73
	1024	37	0.3	37.7	38.0	650.9	0.64

Table 6 presents the number of iterations and timings using the LOBPCG algorithm in the *hypre* package. The LOBPCG wall clock time is obtained with the preconditioner allowing for the best performance in the specific problem at hand, i.e. ParaSails for all the problems. Using AMG as the preconditioner did not allow for convergence in three cases out of four, with the only exception of the FAULT-639 problem, in which the CPU timings were however very much larger than using ParaSails.

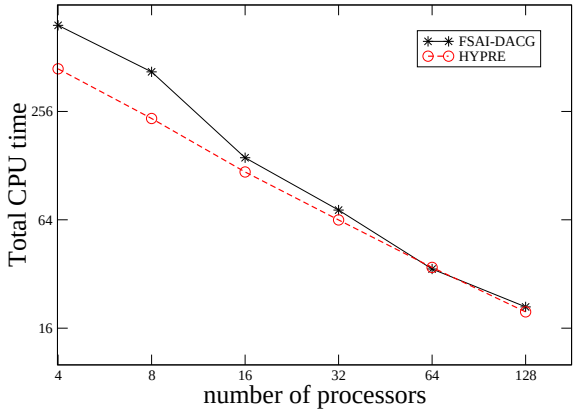
All matrices have to be preliminarily scaled by their maximum coefficient in order to allow for convergence. To make the comparison meaningful, the outer iterations of the different methods are stopped when the average relative error measure of the computed leftmost eigenpairs gets smaller than 10^{-10}, in order to obtain a comparable accuracy as in the other codes. We also report in Table 6 the number of inner preconditioned CG iterations (pcgitr).

Figure 1: Comparison between FSAI-DACG and LOBPCG-*hypre* in terms of total CPU time for different number of processors.

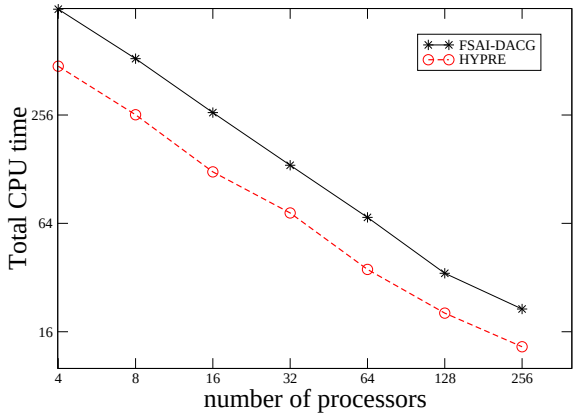
FAULT-639



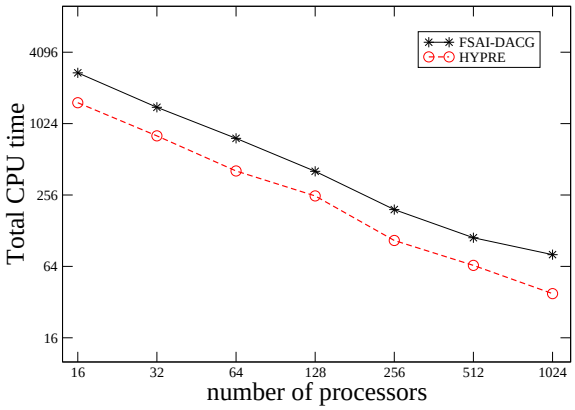
PO-878



GEO-1439



CUBE-6091



To better compare our FSAI DACG with the LOBPCG method, we depict in Figure 1 the total CPU time vs the number of processor for the two codes. FSAI-DACG and LOBPCG provide very similar scalability, being the latter code a little bit more performing on the average. On the FAULT-639 problem, DACG reveals faster than LOBPCG, irrespective of the number of processors employed.

Finally, we have carried out a comparison of the two eigensolvers in the computation of only the leftmost eigenpair. Differently from LOBPCG, which performs a simulta-

neous approximation of all the selected eigenpairs, DACG proceeds in the computation of the selected eigenpairs in a sequential way. For this reason, DACG should be the better choice, at least in principle, when just one eigenpair is sought. We investigate this feature, and the results are summarized in Table 7. We include the total CPU time and iteration count needed by LOBPCG and FSAI-DACG to compute the leftmost eigenpair with 16 processors. For the LOBPCG code we report only the number of outer iterations.

The parameters used to construct the FSAI preconditioner for these experiments are as follows:

1. FAULT-639. $\delta = 0.1$, $d = 2$, $\varepsilon = 0.05$
2. PO-878. $\delta = 0.2$, $d = 4$, $\varepsilon = 0.1$
3. GEO-1438. $\delta = 0.1$, $d = 2$, $\varepsilon = 0.1$
4. CUBE-6091. $\delta = 0.0$, $d = 1$, $\varepsilon = 0.05$

These parameters differ from those employed to compute the FSAI preconditioner in the assessment of the 10 leftmost eigenpairs, and have been selected in order to produce a preconditioner relatively cheap to compute. This is so because otherwise the setup time would prevail over the iteration time. Similarly, to compute just one eigenpair with LOBPCG we need to setup a different value for pcgitr, the number of inner iterations. As it can be seen from Table 7, in the majority of the test cases, LOBPCG takes less time to compute 2 eigenpairs than just only 1. FSAI-DACG reveals more efficient than the best LOBPCG on problems PO-878 and GEO-1438. On the remaining two problems the slow convergence exhibited by DACG is probably due to the small relative separation ξ_1 between λ_1 and λ_2.

Table 7: Performance of LOBPCG–*hypre* with Parasails and 16 processors in the computation of the smallest eigenvalue using `bl` $= 1$ and `bl`$= 2$, and FSAI-DACG.

	LOBPCG, `bl`$= 1$		LOBPCG, `bl`$= 2$		FSAI-DACG	
	iter	T_{tot}	iter	T_{tot}	iter	T_{tot}
FAULT-639	144	10.1	132	17.5	1030	15.4
PO-878	99	43.2	34	29.1	993	20.4
GEO-1493	55	40.2	26	37.3	754	27.0
CUBE-6091	144	5218.8	58	522.4	3257	561.1

6 Conclusions

We have presented the parallel DACG algorithm for the partial eigensolution of large and sparse SPD matrices. The scalability of DACG, accelerated with FSAI-type preconditioners, has been studied on a set of test matrices of very large size arising from

real engineering mechanical applications. Our FSAI-DACG code has shown comparable performances with the LOBPCG eigensolver within the well known public domain package, *hypre*. Numerical results reveal that not only the scalability achieved by our code is roughly identical to that of *hypre* but also, in some instances, FSAI-DACG proves more efficient in terms of absolute CPU time. In particular, for the computation of the leftmost eigenpair, FSAI-DACG is more convenient in 2 problems out of 4.

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

References

[1] *Lawrence Livermore National Laboratory, hypre user manual, software version 1.6.0*. Center for Applied Scientific Computing (CASC), University of California, 2001.

[2] P. Arbenz, U. Hetmaniuk, R. Lehoucq, and R. Tuminaro, *A comparison of eigensolvers for large-scale 3D modal analysis using AMG-preconditioned iterative methods*, Int. J. Numer. Methods Engrg., 64 (2005), pp. 204–236.

[3] 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.

[4] 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.

[5] ——, *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.

[6] 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.

[7] L. Bergamaschi, G. Pini, and F. Sartoretto, *Approximate inverse preconditioning in the parallel solution of sparse eigenproblems*, Numer. Lin. Alg. Appl., 7 (2000), pp. 99–116.

[8] 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.

[9] N. Castelletto, M. Ferronato, G. Gambolati, C. Janna, P. Teatini, D. Marzorati, E. Cairo, D. Colombo, A. Ferretti, A. Bagliani, and S. Mantica, *3D geomechanics in UGS projects: a comprehensive study in northern Italy*, in Proceedings of the 44th US Rock Mechanics Symposium, Salt Lake City (UT), 2010.

[10] M. Ferronato, C. Janna, and G. Gambolati, *Mixed constraint preconditioning in computational contact mechanics*, Comp. Methods App. Mech. Engrg., 197 (2008), pp. 3922–3931.

[11] M. Ferronato, C. Janna, and G. Pini, *Efficient parallel solution to large-size sparse eigenproblems with block FSAI preconditioning*, Numerical Linear Algebra with Applications, (2011). published online on 18 OCT 2011.

[12] C. Janna, M. Ferronato, and N. Castelletto, *BFSAI-IC OpenMP implementation.* Available online at: `http://www.dmsa.unipd.it/~ferronat/software.html`. Release V1.0, January 2011.

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

[15] A. V. Knyazev and M. E. Argentati, *Implementation of a preconditioned eigensolver using hypre*, 2005. http://math.cudenver.edu/ccm/reports/rep220.pdf.

[16] A. V. Knyazev and A. L. Skorokhodov, *The preconditioned gradient-type iterative methods in a subspace for partial generalized symmetric eigenvalue problem*, SIAM J. Numerical Analysis, 31 (1994), pp. 1226–1239.

[17] 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.

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

[19] R. B. Lehoucq and D. C. Sorensen, *Deflation techniques for an implicit restarted Arnoldi iteration*, SIAM J. Matrix Anal., 17 (1996), pp. 789–821.

[20] 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.

[21] G. L. G. Sleijpen and H. A. van der Vorst, *A Jacobi-Davidson method for linear eigenvalue problems*, SIAM J. Matrix Anal., 17 (1996), pp. 401–425.

[22] P. Teatini, M. Ferronato, G. Gambolati, D. Bau, and M. Putti, *Anthropogenic Venice uplift by seawater pumping into a heterogeneous aquifer system*, Water Resour. Res., 46 (2010).