October 2017

Spectral acceleration of parallel iterative eigensolvers for large scale scientific computing

Luca BERGAMASCHI [a,1], and Ángeles MARTÍNEZ [b]
[a] *Department of Civil Environmental and Architectural Engineering University of Padua*
[b] *Department of Mathematics "Tullio Levi-Civita", University of Padua*

Abstract. The computation of a number of the smallest eigenvalues of large and sparse matrices is crucial in various scientific applications, as the Finite Element solution of PDEs, electronic structure calculations or Laplacian of graphs, to mention a few. We propose in this contribution a parallel algorithm which is based on the spectral low-rank modification of a factorized sparse inverse preconditioner (RFSAI) to accelerate Newton-based iterative eigensolvers. Numerical results onto matrices arising from various realistic problems with size up to 5 million unknowns and 2.2×10^8 nonzero elements account for the efficiency and the scalability of the proposed RFSAI–updated preconditioner.

Keywords. Eigenpairs, Newton's method, preconditioners, approximate inverses

1. Acceleration of eigensolvers by spectral preconditioners

Consider a symmetric positive definite (SPD) matrix A, which is also assumed to be large and sparse. We will denote as $0 < \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_m \leq \ldots \leq \lambda_n$ the eigenvalues of A and $v_1, v_2, \ldots, v_m, \ldots, v_n$ the corresponding (normalized) eigenvectors.

To compute the $j-$th leftmost eigenpair we choose the DACG-Newton method [2]. The DACG solver[3], is used to assess a rough initial approximation of the eigenvector which is refined by the subsequent projected Newton method. In this paper we propose a parallel implementation of a new preconditioning strategy as proposed in [1] for accelerating the Newton stage, namely the Newton method in the unit sphere:

$$\text{solve by PCG} \quad J_k^{(j)} s_k = -r_k; \qquad \text{Set} \quad t = u_k + s; \qquad u_{k+1} = t\|t\|^{-1}$$

$$\text{where} \quad J_k^{(j)} = (I - QQ^\top)(A - \theta_k I)(I - QQ^\top), \; r_k = Au_k - \theta_k u_k, \quad \theta_k = u_k^\top A u_k$$

$$Q = \begin{bmatrix} v_1 \; v_2 \ldots v_{j-1} \; u_k \end{bmatrix}$$

The idea is to use the initial DACG approximation not only as an initial eigenvector guess for the Newton phase, but also to construct a spectral preconditioner for the efficient

[1] Corresponding Author: Luca Bergamaschi, Department of Civil Environmental and Architectural Engineering, University of Padua, E-mail: luca.bergamaschi@unipd.it.

solution of the correction equation to be solved at each step of this projected Newton method. We assume that the DACG method has provided the m leftmost eigenpairs of A (to a low relative accuracy specified by parameter τ) satisfying

$$A\tilde{v}_j = \lambda_j \tilde{v}_j + \mathbf{res}_j, \qquad \|\mathbf{res}_j\| \leq \tau\lambda_j,\ j = 1,\ldots,m, \tag{1}$$

Then, for a generic eigenvalue λ_j $(j < m)$ we define the following tuned preconditioner, which will be kept constant throughout the Newton iterations, to accurately compute the j-th eigenpair:

$$\widehat{P}_j = \widehat{P}_0 - W\left(W^\top A V_j\right)^{-1} W^\top, \qquad \text{with} \qquad W = \widehat{P}_0 A V_j - V_j \tag{2}$$

where $V_j = \left[\tilde{v}_{j+1}\ldots\tilde{v}_m\right]$ and $\widehat{P}_0$ is an initial approximate inverse of A. A direct computation shows that $\widehat{P}_j$ is a tuned preconditioner ([4]) i.e. satisfies: $\widehat{P}_j A V_j = V_j$, irrespective of the error introduced by the computation of V_j. This means that the preconditioned matrix $\widehat{P}_j A$ has the eigenvalue 1 with at least multiplicity $m-j$. When $\widehat{P}_j$ is used to accelerate the Newton iteration, it must be projected in the space orthogonal to the previously computed eigenvectors yielding: $P_j = \left(I - QQ^\top\right)\widehat{P}_j\left(I - QQ^\top\right)$. Theorem 1.1, whose proof can be found in [1], accounts for the clustering of eigenvalues of the preconditioned Jacobian provided by the spectral acceleration:

Theorem 1.1 *Let $\widehat{P}_j$ a tuned preconditioner satisfying $\widehat{P}_j A V_j = V_j$, then $\tilde{v}_s, s = j+1,\ldots,m$, is an approximate eigenvector of $P_j J_k^{(j)}$ corresponding to the approximate eigenvalue $1 - \dfrac{\theta}{\lambda_s} \approx 1 - \dfrac{\lambda_j}{\lambda_s}$.*

2. Algorithmic issues

In order to yield an efficient implementation of our spectral preconditioner, the following issues should be taken into account:

1. **Limited memory implementation**. If the number of eigenpairs being sought is large, it is convenient to limit the number of eigenvectors used for the update. To this end we fix the maximum column dimension of matrix V_j, parameter $l_{\max}$.
2. **Adding columns to matrix** W. In assessing an eigenpair whose index is close to m, the size of matrix V_j is necessarily small and only $m-j$ eigenvalues of the preconditioned matrix will be characterized by Theorem 1.1. In particular when $j = m$, V_j is the empty matrix. To make the proposed approach effective also for such eigenpairs a second variant consists in computing an additional number (win) of approximated eigenpairs by the DACG procedure.

Taking into account these variants, in the computation of the j-th eigenpair we will use $V_j = \left[\tilde{v}_{j+1}\ldots\ \tilde{v}_{j_{end}}\right]$ with $j_{end} = \min\{m + \mathrm{win}, l_{\max} + j\}$ to get the final expression for our spectral preconditioner which from now on we will denote as $P_0^{(j)}$:

$$\widehat{P}_0^{(j)} = P_{RFSAI} - W\left(W^\top A V_j\right)^{-1} W^\top, \qquad \text{with} \qquad W = P_{RFSAI} A V_j - V_j \tag{3}$$

$$P_0^{(j)} = (I - QQ^\top)\widehat{P}_j^{(0)}(I - QQ^\top) \tag{4}$$

being P_{RFSAI} the initial RFSAI preconditioner which will be described later. The DACG-Newton algorithm with spectral preconditioner is then sketched in Algorithm 1.

Algorithm 1 DACG-Newton with spectral preconditioner.

1. INPUT: A, m, ε, ITMAX, τ, τ_{PCG}, ITMAX$_{PCG}$, $l_{\max}$, win.
2. Compute an RFSAI preconditioner for A: P_{RFSAI}.
3. $V := [\,]$.
4. FOR $j := 1$ TO $m + \text{win}$
   (a) Choose x_0 such that $V^\top x_0 = 0$.
   (b) Compute $\tilde{v}_j$ by DACG with initial vector x_0, preconditioner P_{RFSAI} and tolerance τ.
   (c) Set $V := [V\ \tilde{v}_j]$

   END FOR
5. $\tilde{Q} := [\,]$.
6. FOR $j := 1$ TO m
   (a) $k := 0$, $u_0 = \tilde{v}_j$, $\theta_0 := u_0^\top A u_0$.
   (b) $Q := [\tilde{Q}\ u_0]$.
   (c) Set $j_{end} = \min\{m + \text{win}, l_{\max} + j\}$, $V_j = \left[\tilde{v}_{j+1} \ldots \tilde{v}_{j_{end}}\right]$
   (d) Compute $\widehat{P}_0^{(j)}$ using (4) and set $P_0^{(j)} = (I - QQ^\top)\widehat{P}_j^{(0)}(I - QQ^\top)$;
   (e) WHILE $\|Au_k - \theta_k u_k\| > \varepsilon\theta_k$ AND $k <$ IMAX DO
      1. Solve $J_k s_k = -r_k$ for $s_k \perp Q$ by PCG with preconditioner $P_j^{(0)}$.
      2. $u_{k+1} := \dfrac{u_k + s_k}{\|u_k + s_k\|}$, $\theta_{k+1} = u_{k+1}^\top A u_{k+1}$.
      3. $k := k + 1$
      4. $Q := [\tilde{Q}\ u_k]$.
   (f) END WHILE
   (g) Assume $v_j = u_k$ and $\lambda_j = \theta_k$. Set $\tilde{Q} := [\tilde{Q}\ v_j]$

   END FOR

2.1. Repeated application of the spectral preconditioning technique

In principle every eigenvalue solver may take advantage of the spectral preconditioning technique to update a given preconditioner. In our case the idea is to run twice the DACG method: in the first run a very rough approximation of the leftmost $m + \text{win}$ eigenpairs: $\tilde{v}_1^{(0)}, \tilde{v}_2^{(0)}, \ldots, \tilde{v}_{m+\text{win}}^{(0)}$ is provided. To this end we define a tolerance $\mu\,(> \tau)$ and iterate until the test on the residual $\|A\tilde{v}_j^{(0)} - q(\tilde{v}_j^{(0)})\tilde{v}_j^{(0)}\| \le \mu q(\tilde{v}_j^{(0)})$ is satisfied. Then a second run of DACG to the final DACG tolerance is carried on, using $\tilde{v}_1^{(0)}, \tilde{v}_2^{(0)}, \ldots, \tilde{v}_m^{(0)}$ as the starting points and also using them for updating the RFSAI preconditioner. Clearly this DACG step will be accelerated by the non-projected spectral preconditioner $\widehat{P}_j^{(0)}$. The output of this second run will be the sequence of vectors $\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_m$ which will be in their turn the starting points of the subsequent Newton scheme, while the set $\{\tilde{v}_1, \tilde{v}_2, \ldots, \tilde{v}_m, \tilde{v}_{m+1}^{(0)}, \ldots, \tilde{v}_{m+\text{win}}^{(0)}\}$ will be used for the preconditioner updating.

These steps are summarized in Algorithm 2 where we only underline the variations with respect to Algorithm 1.

Algorithm 2 Two-stage DACG-Newton with spectral preconditioner.

1. INPUT: (in addition to that of Algorithm 1) tolerance for the first DACG stage $\mu(\geq \tau)$;

(STEPS 3. and 4. in Algorithm 1 are substituted with the following ones)

3.a $V^{(0)} := [\,]$, $V := [\,]$.
3.b FOR $j := 1$ TO $m + \text{win}$

(a) Choose x_0 such that ${V^{(0)}}^\top x_0 = 0$.
(b) Compute $\tilde{v}_j^{(0)}$ by DACG with initial vector x_0, preconditioner P_{RFSAI} and tolerance μ.
(c) Set $V^{(0)} := [V^{(0)}\ \tilde{v}_j^{(0)}]$

END FOR
4.a FOR $j := 1$ TO m

(a) Compute $\tilde{v}_j$ by DACG with initial guess $\tilde{v}_j^{(0)}$, preconditioner $\widehat{P}_0^{(j)}$ (4) and tolerance τ.
(b) Set $V := [V\ \tilde{v}_j]$

END FOR
4.b $V := [V\ \tilde{v}_{m+1}^{(0)} \ldots \tilde{v}_{m+\text{win}}^{(0)}]$.

2.2. BFGS low-rank update of given preconditioners

In [2] the sequence of correction equations $J_k^{(j)} s_k = -r_k$ is preconditioned by means of a sequence of low-rank updates of a given approximate inverse of A. A limited variant is also defined which fix the maximum number, $k_{\max}$, of rank two corrections allowed. The BFGS approach and the spectral techniques described in the previous sections can be combined giving raise to a spectral-BFGS preconditioner for the Inexact Newton method, which is defined as follows for a given eigenpair j:

$$\widehat{P}_0^{(j)} = P_{RFSAI} - W\left(W^\top A V_j\right)^{-1} W^\top, \qquad \text{with} \qquad W = P_{RFSAI} A V_j - V_j \tag{5}$$

$$\widehat{P}_{k+1}^{(j)} = -\frac{s_k s_k^\top}{s_k^\top r_k} + \left(I - \frac{s_k r_k^\top}{s_k^\top r_k}\right) \widehat{P}_k^{(j)} \left(I - \frac{r_k s_k^\top}{s_k^\top r_k}\right) \qquad k = 0, \ldots, \tag{6}$$

$$P_{k+1}^{(j)} = (I - Q_{k+1}^{(j)} {Q_{k+1}^{(j)}}^\top) \widehat{P}_{k+1}^{(j)} (I - Q_{k+1}^{(j)} {Q_{k+1}^{(j)}}^\top).$$

3. Parallel implementation

3.1. Choice and construction of the initial preconditioner

The classical 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.

The drawback of the FSAI approach is due to (a) 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, (b) a fixed sparsity pattern, based on small powers of A, can hardly capture all the most important nonzeros in A^{-1}.

Following the developments in [5], we propose an implicit enlargement of the sparsity pattern using a banded target matrix B: 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 compute explicitly the preconditioned matrix $S = G_{out}AG_{out}^T$ and then evaluate a second FSAI factor G_{in} for S. Thus the final preconditioner can be written as $P_{RFSAI} = G_{out}^T G_{in}^T G_{in} G_{out}$. This RFSAI – recursive FSAI – procedure can be iterated a number of times to yield a preconditioner written as a product of several triangular factors.

We denote as FSAIout the procedure which computes the G_{out} factor by minimizing $\|B-GL\|$ where B is an arbitrary banded matrix. 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.

The second preconditioner factor, G_{in}, is the result of the FSAI procedure applied to the whole product matrix $G_{out}AG_{out}^T$, with parameters δ_{in}, d_{in} and ε_{in}. The steps to obtain the final RFSAI preconditioner are summarized in Algorithm 3.

Algorithm 3 RFSAI computation

INPUT: `nband`, $\delta_{out}, \varepsilon_{out}, d_{out}, \delta_{in}, \varepsilon_{in}, d_{in}$

Compute the first lower triangular factor: $G_{out} = \text{FSAIout}(A, \texttt{nband}, \delta_{out}, d_{out}, \varepsilon_{out})$

Compute the product: $A^{(1)} = G_{out}AG_{out}^T$

Compute the second lower triangular factor: $G_{in} = \text{FSAI}(A^{(1)}, \delta_{in}, d_{in}, \varepsilon_{in})$

We have developed a parallel code which implements the construction, and application inside parallel PCG, of the RFSAI algorithm along with the spectral-BFGS updates. 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 which are all stored in static data structures in CSR format.

Parallelization of the FSAI preconditioner, which is the basis of the parallel RFSAI construction, has been performed and tested e.g. in [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 code makes also use of an optimized parallel matrix-vector product which has been developed in [9] showing its effectiveness up to 1024 processors.

3.2. Parallel application of the spectral preconditioner

At every PCG inner iteration the application of the spectral preconditioner is made by multiplying matrix $\widehat{P}_0^{(j)} = P_{RFSAI} - W\left(W^\top AV_j\right)^{-1}W^\top$ by the residual vector r. The communication cost in the spectral update is essentially within the product $W^\top r$ which needs a collective communication at the end. Small matrix $(W^\top AV_j)^{-1}$ is replicated in all processors and its application is communication-free as well as multiplication of the resulting vector by W.

4. Numerical Results

We have tested the resulting parallel code onto a number of large scale SPD matrices arising from 3D FE discretization of realistic fluid flow and geomechanical models. In detail:

1. `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, the computational grid is characterized by highly irregularly shaped elements.
2. `CUBE-`k : 3D homogeneous elasticity problem. The matrices `CUBE-`k have been obtained through the FE discretization of an elasticity problem on a cube with linear tetrahedra. The material property is constant on the domain and the Poisson ratio is assumed equal to 0.3. The sequence of matrices is generated by subsequent refinement of the original grid (that is each tetrahedron is divided in 8 smallest ones).

All 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 elements for each matrix are provided in **Table 1**.

Table 1. Size n and number of nonzeros `nnz` of the test matrices.

name	n	`nnz`
`EMILIA-923`	923 136	41 005 206
`Cube105`k	105 597	4 088 979
`Cube739`k	739 167	29 657 925
`Cube5317`k	5 317 443	222 615 369

4.1. Machine characteristics

All tests have been performed on the new HPC Cluster Marconi at the CINECA Centre, on both the A1 version (1512 nodes, 2×18-cores Intel Xeon E5-2697 v4 (Broadwell) at 2.30 GHz) and the more recent A2 update (with 3600 nodes and 1×68-cores Intel Xeon 7250 CPU (Knigths Landing) at 1.4GHz). The Broadwell nodes have 128 Gb memory each, while in the A2 system the RAM is subdivided into 16GB of MDRAM and 96GB of DDR4. The Marconi Network type is: new Intel Omnipath, 100 Gb/s. (MARCONI is the largest Omnipath cluster of the world).

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:

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

4.2. Scalability results on matrix `EMILIA-923`

We now report the results of two runs in the computation of the 10 leftmost eigenpairs of matrix `EMILIA` $-$ `923` for two different values of parameter win. The parameters selected and the number of iterations (which do not change with the number of processors) are summarized in **Table 2**. Run # 1, with win $= 0$, provides the smallest number of total iterations thus the related combination of parameters has been chosen to perform the parallel scalability analysis reported in **Table 3**. In this Table we specified the number of processors p, as well the configuration used to get such number of processors, namely the number of nodes and CPUs per node used.

Table 2. Parameters and number of Matrix-Vector Products (MVP) of DACG-Newton in eigensolving matrix `EMILIA-923` .

						MVP			
Run	win	$l_{\max}$	$k_{\max}$	μ	τ	DACG 1	DACG 2	Newton	overall
# 1	0	5	20	0.1	0.02	5229	675	4680	10584
# 2	2	5	20	0.1	0.02	6613	580	4481	11574

Table 3. Timings and scalability for Run #1 with matrix `EMILIA-923` on the A2 partition.

p	nodes	CPUs	elapsed time						
			T_{prec}	T_{DACG}	$T_{lowrank}$	T_{Newton}	T_{tot}	$S_p^{(8)}$	$E_p^{(8)}$
8	8	1	87.50	348.50	0.38	296.47	732.85	8.0	1.00
32	8	4	31.05	127.07	0.11	109.54	267.76	21.9	0.68
128	32	4	9.43	53.07	0.04	46.12	108.66	54.0	0.42
256	32	8	5.43	35.92	0.03	31.90	73.28	80.0	0.31
512	32	16	3.70	27.75	0.04	25.74	57.24	102.4	0.20
512	64	8	3.38	28.35	0.03	26.54	58.29	102.6	0.20

4.3. Scalability results on matrices `CUBE-`k

Matrices `CUBE-`k , albeit arising from FE discretization of a PDE on unstructured meshes, have been recursively generated by halving the side lengths of the elements. This results in a sequence of nested grid having roughly 8 times more nodes and hence a sequence of matrices having 8 times more rows and nonzeros. It is possible then to perform a sort of weak scalability on this problems, by taking into account that also the condition number of such matrices increases proportionally with their size. Hence, even fixing for all matrices the same parameters for the initial RFSAI preconditioner, the number of iterations increases with increasing dimension. To obtain a sound measure of the scalability performance of our code we analyzed:

1. The overall CPU time in computing the initial RFSAI preconditioner;
2. The average CPU time needed for a single Newton iteration.

Table 4. Parameters and number of Matrix-Vector Products (MVP) of DACG-Newton in eigensolving matrices CUBE-k .

DACG-Newton				RFSAI		MVP		
win	$l_{\max}$	$k_{\max}$	μ, τ	$\delta_{out}, \varepsilon_{out}, d_{out}$	$\delta_{in}, \varepsilon_{in}, d_{in}$	105k	739k	5317k
3	5	20	0.1, 0.02	0.05, 0.05, 4	0.1, 0.1, 2	3342	8574	19263

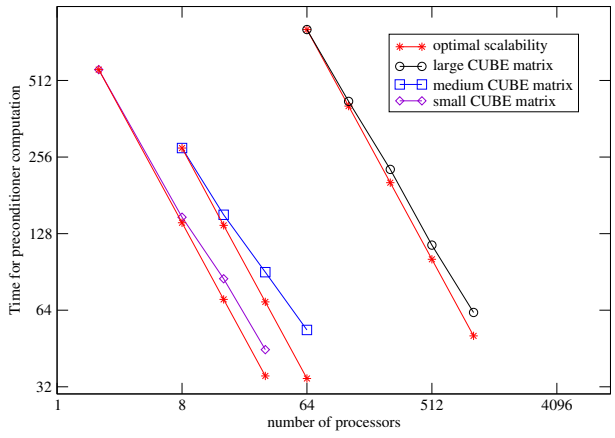
Table 5. Timings and relative speedups and efficiencies for the matrices CUBE-k .

p	Cube105k	$S_p^{(2)}$	$E_p^{(2)}$	Cube739k	$S_p^{(8)}$	$E_p^{(8)}$	Cube5317k	$S_p^{(64)}$	$E_p^{(64)}$
2	659.51	2.0	–						
4	343.49	3.8	0.96						
8	178.05	7.4	0.93	1262.72	8.0	–			
16	85.08	15.5	0.97	656.67	15.4	0.96			
32	44.78	29.5	0.92	368.65	27.4	0.86			
64				198.28	50.9	0.80	1922.12	64.0	–
128				130.14	77.6	0.61	1064.17	115.6	0.90
256							659.81	186.4	0.73
512							382.13	321.9	0.63
1024							284.43	432.5	0.42

We first report in **Table 4** the parameters employed and the overall number of iterations displayed by the DACG-Newton method. Next in **Table 5** we report for each run the overall timings, the relative speedups and efficiencies for the three matrices CUBE−k.

The scalability analysis for the RFSAI computation is displayed in **Figure 1**, from which, by comparing the curves with the theoretical scalability, we can appreciate the almost perfect scalability of this task at least for the largest matrix: Cube5317k .

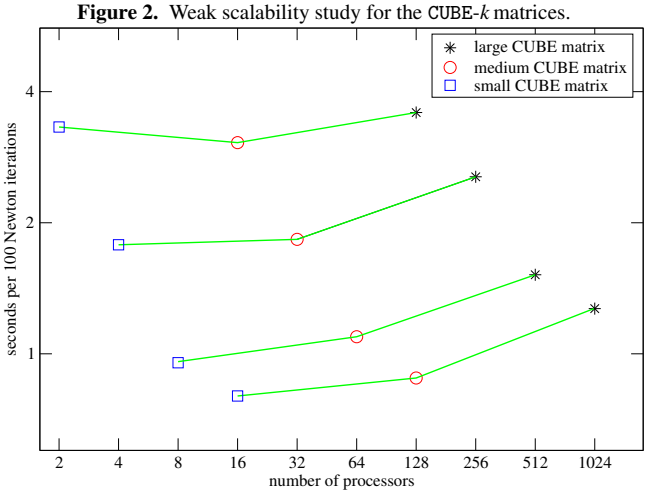
Figure 1. Timings for computing the RFSAI preconditioner for different CUBE-k matrices and number of processors.



A weak scalability analysis for the Newton step is carried on in **Figure 2**, where the average CPU time per Newton iteration (in seconds per 100 iteration) is plotted against the number of processors for the three different matrices CUBE-k . Weak scalability is not completely satisfactory as we would expect almost straight lines. There is an (obvious) loss of perfect scalability from matrix Cube739k to matrix Cube5317k due to the fact

that the frequent scalar products inside the PCG algorithm have a deeper impact on the overall cost due to the increasing number of processor involved.

Figure 2. Weak scalability study for the CUBE-k matrices.



We report some further results on two matrices of the CUBE-k class to evidence the importance of the low-rank acceleration of a given preconditioner and the influence of low-rank preconditioners on the sequential and parallel efficiency. With the same initial preconditioner as that employed in the previous section we run the DACG-Newton code either with the BFGS acceleration only or with no low-rank updates. The results obtained using 128 processors for the Newton step only are summarized in **Table 6** where we see that the spectral plus BFGS update is mandatory for the largest Cube5317k matrix. As for the Cube739k matrix, with no acceleration, no convergence is observed while the spectral acceleration provides a halving of the Newton iterations with respect to using the BFGS update only.

Table 6. Newton iterations for the two largest matrices with different low-rank acceleration parameters.

win	$l_{\max}$	$k_{\max}$	Cube739k	Cube5317k
3	5	20	2931	7119
–	–	20	5457	†
–	–	–	†	†

Table 7. Timings and relative speedups and efficiencies for the matrix Cube739k (Newton phase).

p	Cube739k	$S_p^{(8)}$	$E_p^{(8)}$	Cube739k	$S_p^{(8)}$	$E_p^{(8)}$
	BFGS only acceleration			spectral+BFGS acceleration		
8	295.88	8.0	–	167.16	8.0	–
16	163.13	14.5	0.91	92.82	14.4	0.90
32	96.05	24.6	0.77	55.66	24.0	0.75
64	61.74	38.3	0.60	33.25	40.2	0.63
128	41.14	57.5	0.45	26.81	49.9	0.39

Regarding scalability we compare the spectral-BFGS preconditioner with the BFGS preconditioner for the `Cube739`k matrix once again for the Newton phase only obtaining the CPU times reported in **Table 7**. It can be observed that the spectral acceleration provides a great reduction of the CPU time, irrespective of the number of processors, while it only slightly worsens the parallel scalability of the Newton phase.

5. Conclusion and future work

We have developed and tested a pure-MPI parallel preconditioned Newton-based iterative algorithm to compute the leftmost eigenpairs of large and sparse SPD matrices. To accelerate the inner PCG solver we have proposed a preconditioner based on low-rank corrections of a recursive FSAI initial preconditioner. The results on the `Marconi` supercomputer show the efficiency of the low-rank update and the good parallel scalability of the overall code. In particular the computation of the initial RFSAI preconditioner achieves almost optimal scalability. The not completely satisfactory scalability of the iterative phase is due to the communications needed by the sparse matrix-vector products and by the scalar products to be performed both in the PCG solver and in the low-rank acceleration phase. We plan to supply to these weaknesses both algorithmically, by rewriting the BFGS update in a compact form as proposed in [10] so as to reduce the number of scalar products, and by writing a MPI-Cuda version of the code, following e.g. the works in [7,8] where a GPU-based implementation of the PCG solver as well as optimized routines for the matrix-vector and the scalar product are developed.

Acknowledgements. This research have been carried on under the ISCRA C project *Low-rank acceleration of parallel preconditioners* and under the Italian GNCS project *Numerical Methods for Large Scale Constrained Optimization Problems & Applications*.

References

[1] Bergamaschi L, Martínez A. Two-stage spectral preconditioners for iterative eigensolvers. *Numer. Lin. Alg. Appl.* 2017; **24**:1–14.

[2] Bergamaschi L, Martínez A. Efficiently preconditioned inexact Newton methods for large symmetric eigenvalue problems. *Optimization Methods & Software* 2015; **30**:301–322.

[3] Bergamaschi L, Gambolati G, Pini G. Asymptotic convergence of conjugate gradient methods for the partial symmetric eigenproblem. *Numer. Lin. Alg. Appl.* 1997; **4**(2):69–84.

[4] Freitag MA, Spence A. Shift-invert Arnoldi's method with preconditioned iterative solves. *SIAM J. Matrix Anal. Appl.* 2009; **31**(3):942–969.

[5] Bergamaschi L, Martínez A. Banded target matrices and recursive FSAI for parallel preconditioning. *Numerical Algorithms* 2012; **61**(2):223–241.

[6] Bergamaschi L, Martínez A. Parallel inexact constraint preconditioners for saddle point problems. *Euro-Par 2011, Bordeaux (France)*, *Lecture Notes in Computer Sciences*, vol. 6853, Part II, E Jeannot RN, Roman J (eds.), Springer: Heidelberg, 2011; 78–89.

[7] Greathouse JL, Daga M. Efficient sparse matrix-vector multiplication on GPUs using the CSR storage format. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, IEEE Press: Piscataway, NJ, USA, 2014; 769–780.

[8] Helfenstein R, Koko J. Parallel preconditioned conjugate gradient algorithm on GPU. *Journal of Computational and Applied Mathematics* 2012; **236**(15):3584 – 3590.

[9] Martínez A, Bergamaschi L, Caliari M, Vianello M. A massively parallel exponential integrator for advection-diffusion models. *J. Comput. Appl. Math.* 2009; **231**(1):82–91.

[10] Nocedal J, Wright SJ, *Numerical optimization*, Springer Series in Operations Research, Springer-Verlag, New York, 1999.