Multilevel Approaches for FSAI Preconditioning

Andrea Franceschini [*1], Victor A. P. Magri [†1], Carlo Janna [‡1], and Massimiliano Ferronato [§1]

[1]Department of Civil and Environmental Engineering, University of Padova

Abstract

Two new approaches for the use of factorized sparse approximate inverse (FSAI) preconditioners are presented. The first one, BTFSAI, is based on a block tridiagonal factorization strategy, while the second one, DDFSAI, is built by reordering the matrix graph according to a multilevel k-way partitioning method followed by a bandwidth minimizing algorithm. These preconditioners are tested for the solution of SPD problems arising from different engineering applications. The results are evaluated in terms of performance, scalability and robustness of the preconditioner, showing that both strategies are able to converge in a smaller number of iterations and total time in comparison to the native FSAI.

I. Introduction

The solution of a large size sparse linear system of equations

$$\mathbf{A}x = b, \tag{1}$$

where $\mathbf{A}$ is a SPD matrix, requires the use of iterative methods based on Krylov subspaces. It is well known, however, that the use of these methods alone generally provide poor convergence. This can be remedied by using their preconditioned form, which requires the definition of a preconditioner, i.e., a linear transformation that approximates the action of $\mathbf{A}^{-1}$.

There are a number of ways to build and apply a preconditioner. Among the algebraic algorithms, the most popular categories are the incomplete factorizations, multigrid methods and sparse approximate inverses [Benzi, 2002; Ferronato, 2012]. Regarding its application, three possibilities may arise, namely the left, right and split preconditioning techniques, which have particular advantages between one another. Additionally, different preconditioners can be composed to form new ones, such as using domain decomposition and incomplete factorizations together or nested Krylov methods [Chan and Mathew, 1994; Mcinnes et al., 2014].

With respect to incomplete factorizations, sparse approximate inverses have the advantage of being more stable. Particularly, the factorized version of this algorithm introduces the improvement of preserving the definiteness of a given problem. Another important difference is that their application only requires a sparse matrix by vector product instead of triangular solves. This last feature is very attractive from the point of view of a parallel implementation, since that type of mathematical operation involves a high level of concurrency.

[*]andrea.franceschini@dicea.unipd.it
[†]victor.magri@dicea.unipd.it
[‡]carlo.janna@unipd.it
[§]massimiliano.ferronato@unipd.it

A necessary step for building a factorized sparse approximate inverse, FSAI, consists in selecting the sparsity pattern. Two main alternatives are available: the static and the dynamic strategy. In the former, an a priori pattern is selected upon a sparsified format of a low power of the original matrix, while in the latter, a certain number of the most significant entries minimizing the Kaporin number of the preconditioned matrix for each row are selected [Grote and Huckle, 1997; Huckle, 2003; Janna and Ferronato, 2011]. In general, the last approach is more effective than the first one as the selected entries are optimal.

In this paper, two new preconditioners built upon the dynamic FSAI algorithm are considered: the BTFSAI - Block Tridiagonal Factorized Sparse Approximate Inverse - and the DDFSAI - Domain Decomposition Factorized Sparse Approximate Inverse. The first one is based on the successive application of the FSAI algorithm on the matrix reordered in a block tridiagonal structure and the second one is based on the combination of a two level domain decomposition approach and the FSAI preconditioner, which is used for computing the approximate inverse of the inner submatrices and the Schur complement.

This work is organized as follows: in the next section, the FSAI algorithm is reviewed and the theory underlying the proposed preconditioners is presented. In the results section, these preconditioners are used to solve a set of linear systems representing real engineering problems. Here, an analysis of their performance and parallel efficiency is done. In the end, the conclusions of this work are stated.

II. FSAI Preconditioning

The FSAI preconditioner $\mathbf{M}^{-1}$ of a symmetric positive definite matrix $\mathbf{A}$ is given by

$$\mathbf{M}^{-1} = \mathbf{F}^T\mathbf{F} \approx \mathbf{A}^{-1}, \tag{2}$$

where the factor $\mathbf{F}$ is calculated by minimizing the Frobenius norm

$$||\mathbf{I} - \mathbf{FL}||_F \tag{3}$$

over the set of matrices which share the same lower triangular nonzero pattern S. In the same equation, the matrix $\mathbf{L}$ is the exact lower Cholesky factor. Note that, although this matrix is used in the equation (3), it is not required for the calculation of $\mathbf{F}$. Indeed, setting to zero the derivative of this equation with respect to the entries f_{ij} yields

$$[\mathbf{FA}]_{ij} = \begin{cases} 0, & \text{if } i \neq j \\ l_{ii}, & \text{if } i = j. \end{cases} \tag{4}$$

Substituting the matrix $\mathbf{F}$ by $\tilde{\mathbf{F}} = \mathbf{D}^{-1}\mathbf{F}$, where $\mathbf{D} = [diag(\mathbf{F})]^{-1/2}$, it is possible to state that

$$\left[\tilde{\mathbf{F}}\mathbf{A}\right]_{ij} = \delta_{ij}, \tag{5}$$

which is used to calculate the factor $\tilde{\mathbf{F}}$ and finally the matrix $\mathbf{F}$. This way, the diagonal entries of the preconditioned matrix $\mathbf{FAF}^T$ are equal to one.

A key factor which affects the performance of the FSAI preconditioner is the selection of the sparsity pattern. A number of strategies are available for this task and one of the most promising consists on the algorithm developed by Janna and Ferronato [2011] which is based on the row-wise minimization of the Kaporin conditioning number. In this strategy, the density of $\mathbf{F}$ is controlled by the following set of parameters:

1. k_{max}: maximum number of steps for adding new entries.

2. ρ_F: number of entries added to the sparsity pattern in each step.

3. ϵ_F: stopping tolerance based on the relative reduction of the Kaporin number.

In the adaptive FSAI preconditioner, a diagonal pattern is assumed as initial guess. After that, the gradient of the Kaporin number of the preconditioned matrix is computed row-wise and the indices of the ρ_F largest entries of this vector are added to the previous sparsity pattern. For each row, a linear system obtained from gathering the components of the full matrix is solved and the result is used to build the corresponding row of $\mathbf{F}$, until the relative reduction of the Kaporin number falls below ϵ_F. This process can be repeated up to a k_{max} number of times. For more details about this algorithm, the reader is referred to the paper by Janna et al. [2015].

I. Block Tridiagonal FSAI

In the Block Tridiagonal preconditioner - BTFSAI - the matrix $\mathbf{A}$ is first reordered by the reverse Cuthill-McKee algorithm [Cuthill and McKee, 1969], with the aim to reduce its bandwidth, and then it is divided into a block tridiagonal structure according to a given number of blocks. After that, the block LDU decomposition of this new matrix is calculated where the inverses of the diagonal blocks are approximated explicitly by the adaptive FSAI algorithm.

With this approach, the global matrix is subdivided into small blocks, that should be approximated with less effort. Though this method is intrinsically sequential, parallelism can be exploited for each block by the use of the FSAI algorithm.

Calling $\mathbf{A}_i$ the SPD diagonal submatrix and $\mathbf{B}_i$ the upper diagonal submatrix of the $i-th$ block, the SPD matrix $\mathbf{A}$ can be written as:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{B}_1 & & & \\ \mathbf{B}_1^T & \mathbf{A}_2 & \mathbf{B}_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \mathbf{B}_{n-2}^T & \mathbf{A}_{n-1} & \mathbf{B}_{n-1} \\ & & & \mathbf{B}_{n-1}^T & \mathbf{A}_n \end{bmatrix}, \tag{6}$$

while its block LDU decomposition reads

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}_1 & & & \\ \mathbf{B}_1^T\mathbf{S}_1^{-1} & \mathbf{I}_2 & & \\ & \ddots & \ddots & \\ & & \mathbf{B}_{n-1}^T\mathbf{S}_{n-1}^{-1} & \mathbf{I}_n \end{bmatrix} \begin{bmatrix} \mathbf{S}_1 & & & \\ & \mathbf{S}_2 & & \\ & & \ddots & \\ & & & \mathbf{S}_n \end{bmatrix} \begin{bmatrix} \mathbf{I}_1 & \mathbf{S}_1^{-1}\mathbf{B}_1 & & \\ & \mathbf{I}_2 & \mathbf{S}_2^{-1}\mathbf{B}_2 & \\ & & \ddots & \ddots \\ & & & \mathbf{I}_n \end{bmatrix} \tag{7}$$

where

$$\mathbf{S}_i = \mathbf{A}_i - \mathbf{B}_{i-1}^T\mathbf{S}_{i-1}^{-1}\mathbf{B}_{i-1}, \qquad i = 2 \ldots n, \tag{8}$$

is the $i-th$ block Schur complement and $\mathbf{S}_1 = \mathbf{A}_1$, by definition. Let $\mathbf{S}_i^{-1} = \mathbf{F}_i^T\mathbf{F}_i$ be the approximate inverse of the Schur complement computed by the adaptive FSAI algorithm and let the matrix $\mathbf{H}_i$ be equal to the product $\mathbf{F}_i\mathbf{B}_i$, it follows that the Schur complement defined on the equation (8) can be calculated recursively as $\mathbf{S}_i = \mathbf{A}_i - \mathbf{H}_{i-1}^T\mathbf{H}_{i-1}$. Using these relations, the

approximate inverse of the matrix $\mathbf{A}$ as given in the equation (7) reads

$$\mathbf{A}^{-1} \approx \mathbf{M}^{-1} = \begin{bmatrix} \mathbf{F}_1^T & -\mathbf{F}_1^T\mathbf{H}_1\mathbf{F}_2^T & & \\ & \mathbf{F}_2^T & -\mathbf{F}_2^T\mathbf{H}_2\mathbf{F}_3^T & \\ & & \ddots & \ddots \\ & & & \mathbf{F}_n^T \end{bmatrix} \begin{bmatrix} \mathbf{F}_1 & & & \\ -\mathbf{F}_2\mathbf{H}_1^T\mathbf{F}_1 & \mathbf{F}_2 & & \\ & \ddots & \ddots & \\ & & -\mathbf{F}_n\mathbf{H}_{n-1}^T\mathbf{F}_{n-1} & \mathbf{F}_n \end{bmatrix}. \quad (9)$$

The procedure to build the BTFSAI preconditioner is summarized in the algorithm 1 and its application to an arbitrary vector v is presented in the algorithm 2.

Algorithm 1 Construction of the BTFSAI preconditioner

procedure CPTBTFSAI($\mathbf{A}, n$)
  Compute $\mathbf{A}_1^{-1} \approx \mathbf{F}_1^T\mathbf{F}_1$, the approximate inverse of $\mathbf{A}_1$
  for $i = 2, \ldots, n$ **do**
    Compute $\mathbf{H}_{i-1} = \mathbf{F}_{i-1}\mathbf{B}_{i-1}$
    Compute $\mathbf{S}_i = \mathbf{A}_i - \mathbf{H}_{i-1}^T\mathbf{H}_{i-1}$
    Compute $\mathbf{S}_i^{-1} = \mathbf{F}_i^T\mathbf{F}_i$, the approximate inverse of the Schur complement
  end for
end procedure

Algorithm 2 Calculation of $u = \mathbf{M}^{-1}v$

procedure APPLYBTFSAI($\mathbf{F}_i$,$\mathbf{H}_i$,v)
  $w_1 = \mathbf{F}v_1$
  for $i = 2, \ldots, n$ **do**
    $w_i = \mathbf{F}_i\left(v_i - \mathbf{H}_{i-1}^T w_{i-1}\right)$
  end for
  $u_n = \mathbf{F}_n^T w_n$
  for $i = n-1, \ldots, 1$ **do**
    $u_i = \mathbf{F}_i^T\left(w_i - \mathbf{H}_i u_{i+1}\right)$
  end for
end procedure

The main drawback of this approach is the need of computing $n-1$ Schur complements along with their FSAI approximations. While theoretically these matrices should be SPD, the use of approximate inverses may cause the loss of this property. Indeed, for some matrices, we can have indefinite Schur complements if the approximate inverses are not accurate enough. In this case, the construction of the preconditioner should be restarted allowing for a larger number of nonzero entries per row.

II. Domain Decomposition FSAI

Roughly speaking, domain decomposition refers to a number of techniques which solve a problem defined on a large domain Ω via the solution of smaller problems defined over subdomains $\Omega_k, k = 1, \ldots, n$. Each problem can be solved independently and the solutions can be gathered to build the outcome of the original problem. A survey work on domain decomposition methods is provided by Chan and Mathew [1994]; Dolean et al. [2015].

In the domain decomposition FSAI preconditioner - DDFSAI - the graph of the original matrix is reordered according to the k-way partition algorithm implemented in the METIS software library [Karypis and Kumar, 1998]. With this method, the number of edges connecting a subdomain to the others is minimized, thus reducing the pieces of information that need to be communicated among different blocks. At the same time, the size of subdomains is kept to be approximately of the same order in favour of providing a good load balance among different threads, which are going to work in one or possibly more blocks. After this, each independent block of the matrix is reordered according to the reverse Cuthill-McKee algorithm in order to reduce its bandwidth.

Let $\tilde{\mathbf{A}}$ be the reordered matrix:

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_1 & \mathbf{B}_1 \\ \mathbf{B}_1^T & \mathbf{A}_2 \end{bmatrix} = \begin{bmatrix} (\mathbf{A}_1)_1 & & & & (\mathbf{B}_1)_1 \\ & (\mathbf{A}_1)_2 & & & (\mathbf{B}_1)_2 \\ & & \ddots & & \vdots \\ & & & (\mathbf{A}_1)_n & (\mathbf{B}_1)_n \\ (\mathbf{B}_1^T)_1 & (\mathbf{B}_1^T)_2 & \dots & (\mathbf{B}_1^T)_n & \mathbf{A}_2 \end{bmatrix}, \tag{10}$$

where the submatrices $(\mathbf{A}_1)_i$ represent the communications between the internal nodes of the $i-th$ subdomain for $i = 1, 2, \dots, n$; $\mathbf{A}_2$, the communications between interface nodes and $(\mathbf{B}_1)_i$, the coupling between the interface and internal nodes belonging to the $i-th$ subdomain. The block LDU decomposition of (10) is given by

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathbf{I}_1 & \\ \mathbf{B}_1^T\mathbf{A}_1^{-1} & \mathbf{I}_2 \end{bmatrix} \begin{bmatrix} \mathbf{A}_1 & \\ & \mathbf{S}_2 \end{bmatrix} \begin{bmatrix} \mathbf{I}_1 & \mathbf{A}_1^{-1}\mathbf{B}_1 \\ & \mathbf{I}_2 \end{bmatrix}, \tag{11}$$

thus, the exact inverse of $\tilde{\mathbf{A}}$ can be written as

$$\tilde{\mathbf{A}}^{-1} = \begin{bmatrix} \mathbf{I}_1 & -\mathbf{A}_1^{-1}\mathbf{B}_1 \\ & \mathbf{I}_2 \end{bmatrix} \begin{bmatrix} \mathbf{A}_1^{-1} & \\ & \mathbf{S}_2^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I}_1 & \\ -\mathbf{B}_1^T\mathbf{A}_1^{-1} & \mathbf{I}_2 \end{bmatrix}. \tag{12}$$

The computation of the inverses $\mathbf{A}_1^{-1}$ and $\mathbf{S}_2^{-1}$ is approximated by the adaptive FSAI algorithm, giving rise to the following preconditioner for $\mathbf{A}$:

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{I}_1 & -\mathbf{F}_1^T\mathbf{F}_1\mathbf{B}_1 \\ & \mathbf{I}_2 \end{bmatrix} \begin{bmatrix} \mathbf{F}_1^T\mathbf{F}_1 & \\ & \mathbf{F}_2^T\mathbf{F}_2 \end{bmatrix} \begin{bmatrix} \mathbf{I}_1 & \\ -\mathbf{B}_1^T\mathbf{F}_1^T\mathbf{F}_1 & \mathbf{I}_2 \end{bmatrix}. \tag{13}$$

Defining $\mathbf{H}_1 = \mathbf{F}_1\mathbf{B}_1$, the expression (13) can be rewritten as

$$\mathbf{M}^{-1} = \begin{bmatrix} \mathbf{F}_1^T & -\mathbf{F}_1^T\mathbf{H}_1\mathbf{F}_2^T \\ & \mathbf{F}_2^T \end{bmatrix} \begin{bmatrix} \mathbf{F}_1 & \\ -\mathbf{F}_2\mathbf{H}_1^T\mathbf{F}_1 & \mathbf{F}_2 \end{bmatrix}. \tag{14}$$

The algorithm for constructing the DDFSAI preconditioner is almost the same as 1 restricted to two blocks, with the only difference consisting in the initial ordering of the matrix, which in the current case uses the METIS library besides of the reverse Cuthill-McKee ordering. The same argument applies also to the preconditioner application that can be summarized according to the algorithm 2 restricted to two blocks only.

III. Results

The analysis of the results is performed in two stages. In the first one, the impact of the DDFSAI and BTFSAI configuration parameters on their performance is studied considering the solution

of a 3D homogeneous Poisson problem. In the second stage, both computational performance and parallel efficiency of the proposed preconditioners, considering an OpenMP implementation, are evaluated for a set of SPD problems from the University of Florida Sparse Matrix Collection [Davis and Hu, 2011]. The performance of the native FSAI algorithm as implemented in the FSAIPACK package [Janna et al., 2015] is also shown with the purpose of comparison. In all test cases, the preconditioned conjugate gradient method (PCG) is used as the linear solver; the right hand side of the linear systems is calculated as the system matrix times the unitary vector and the convergence is considered to be achieved when the relative residual is smaller than 10^{-9}. The density of a sparse approximate inverse preconditioner $\mathbf{F}$ is calculated as

$$\mu_{\mathbf{F}} = \frac{nnz\left(\mathbf{F}\right)}{nnz\left(\mathbf{A}\right)}, \tag{15}$$

with $nnz\left(\mathbf{G}\right)$ the function returning the number of non-zeros of the matrix $\mathbf{G}$. The computational performance is examined through the total number of iterations needed for convergence n_{it} and the wall clock time needed for building the preconditioner T_p and solving the linear system T_s. These data are analysed for the simulation carried out in serial using a local cluster equipped with two Intel Xeon E5-2643 processors and 256 Gbytes of RAM memory. On the other hand, the scalability tests were performed in a single node of the IBM Blue Gene/Q FERMI supercomputer at the CINECA - Centre for High Performance Computing - which is equipped with a 16-core IBM Power A2 processor at 1.6 GHz and 16 Gbytes of RAM memory with 42.6 Gbytes/s bandwidth.

I. Preconditioner Analysis

A comprehensive analysis of the preconditioners proposed in this work is performed considering the solution of a 3D Poisson test case defined on a cube. This problem is solved by the finite difference method by using a uniform grid containing 150 nodes in each direction yielding a SPD matrix containing 3,375,000 rows and 23,490,000 non-zeros.

Initially, we find the optimal configuration for the native FSAI preconditioner, providing the lowest total wall-clock time, and set this configuration as the reference one. The search for the best setup of the preconditioner is accomplished by using the guidelines suggested in [Janna and Ferronato, 2011]. The same input parameters are used for building also the BTFSAI and DDFSAI preconditioners, with the only additional information being the number of blocks or partitions. In order to see how this parameter affects the solution process, we show in the Figure 1 the number of iterations needed for convergence and the times for building these preconditioners and solving the linear systems considering a varying number of blocks between 50 and 400.

For the BTFSAI preconditioner, it is possible to note that the number of iterations needed for convergence decreases with respect to the reference case up to 200 blocks, approximately, then it starts to increase again. This behaviour is probably caused by the quality of the approximated Schur complements, which decreases as the number of blocks gets higher. Considering the DDFSAI preconditioner, we see that the number of iterations grows moderately with respect to both FSAI and BTFSAI preconditioners, but remains approximately constant changing the number of blocks, thus showing a very stable behaviour with n. In this problem, we see an increase in both of the times for computing and applying the preconditioner if compared to the reference case, however this 3D Poisson problem is so simple that a wall-clock time comparison is not significant.

II. Numerical Results

The performance and scalability of the preconditioners BTFSAI and DDFSAI are analysed considering a set of matrices representing different engineering applications, such as structural mechanics,

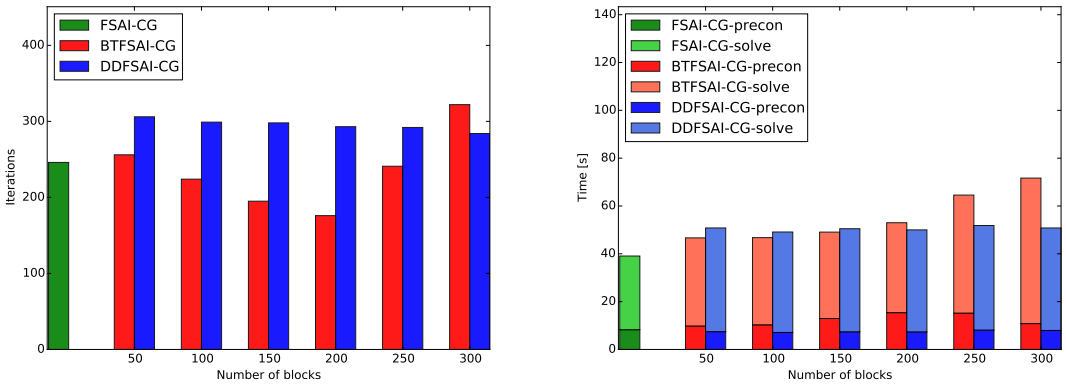

Figure 1: *Number of iterations needed for convergence as a function of the number of blocks used, left. Wall clock time needed for computing the preconditioner and solving the linear system again as a function of the number of blocks used, right. These figures show the results obtained with DDFSAI and BTFSAI.*

geomechanics, electromagnetism and multiphase flow. Table 1 lists the details about the linear problems solved.

Table 1: *Test problems solved*

Name	Rows	Nonzeros	Description
Ecology2	999,999	4,995,991	2D ecology problem
Tmt-sym	726,713	5,080,961	Electromagnetic problem
Cylinder	372,960	16,473,150	3D geomechanical problem
Hook1498	1,498,023	59,374,451	3D structural problem

Similarly as Figure 1, we show in Figure 2 how the performance of both preconditioners vary according to the number of blocks or partitions used in their construction. Except for the last test case, the BTFSAI preconditioner always shows a gain in performance with respect to the FSAI preconditioner, though with a performance that is sensitive to the number of selected blocks. It is possible to note that there is a problem dependent optimal number of blocks for building this preconditioner. More blocks involve the approximation of too many Schur complements with a performance degradation. On the other hand, a few blocks do not exhibit thoroughly the block tridiagonal structure of the matrix, but still can perform better than the pure FSAI. The DDFSAI preconditioner can also decrease the solution time and the total number of iterations, although this effect is not so pronounced as in the BTFSAI case. Its behaviour, however, is always very stable with n. When the time needed for building a good preconditioner is relatively large, like in the Cylinder case, the use of DDFSAI is more pronounced since the inverse of the Schur complement can be calculated with a low accuracy showing no significant loss of effectiveness.

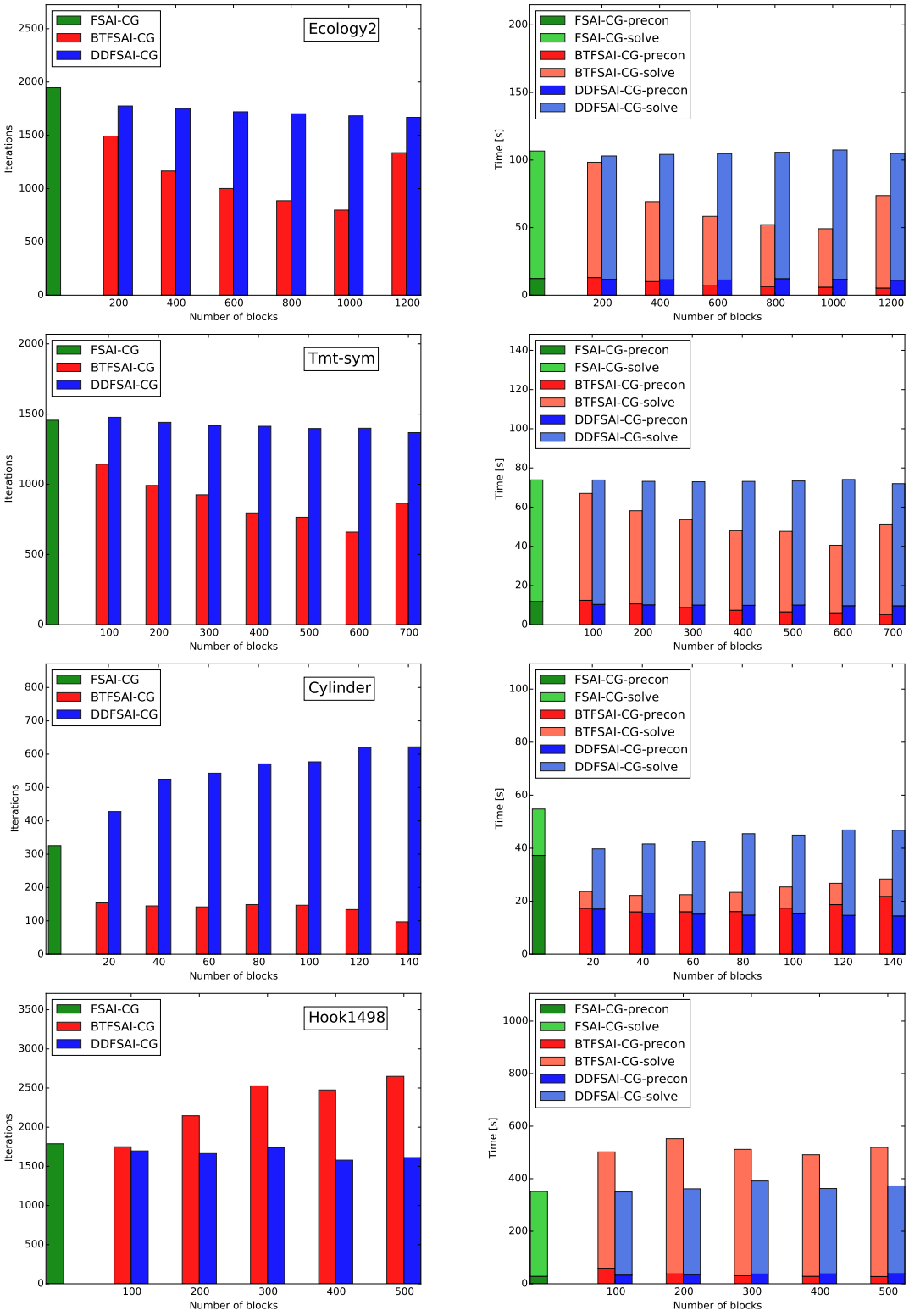

Figure 2: *Convergence results for the matrices Ecology2, Tmt-sym, Cylinder and Hook1498, respectively*

The densities of the optimal preconditioners in the sense of execution time for each test case are given in table 2. It is interesting to note that the values for BTFSAI and DDFSAI are smaller than the one for FSAI, except for the last test case, meaning that those preconditioners can achieve convergence with a smaller storage need and hence generally exhibit a better quality in the selection of nonzeroes. In particular, BTFSAI can allow for a significant saving of storage amount, especially when a denser native FSAI is needed.

Table 2: *Densities of the optimal preconditioners with respect to execution time.*

Test Case	**FSAI**	**BTFSAI**(n)	**DDFSAI**(n)
Ecology2	2.40	1.58(1000)	2.34(400)
Tmt-sym	1.66	1.36(100)	1.60(500)
Cylinder	0.47	0.29(40)	0.30(200)
Hook1498	0.40	0.66(400)	0.45(400)

The strong scalability profiles of DDFSAI and BTFSAI for the first three test cases are given in Figure 3. For each preconditioner, we consider two values for the number of blocks, i.e., a small value and the one giving the best total time of solution as shown in table 2. We show the scalability profiles up to 64 threads since the IBM Blue Gene/Q is a computer designed for parallel computations supporting this number of threads with negligible overhead. According to the figure on the left, DDFSAI shows a good scalability close to the ideal one for the test cases *ecology2* and *tmt-sym* in both configurations, i.e., with many partitions and just a few. The scalability for the *cylinder* is a little bit worse, which is probably caused by the small size of this linear system. Regarding the figure on the right, we note that the scalability of BTFSAI is not as good. This is because the algorithm involves more synchronization points than DDFSAI, thus decreasing the level of concurrency especially when increasing n. However, the BTFSAI can still provide a good scalability as can be seen for the *tmt-sym* matrix.

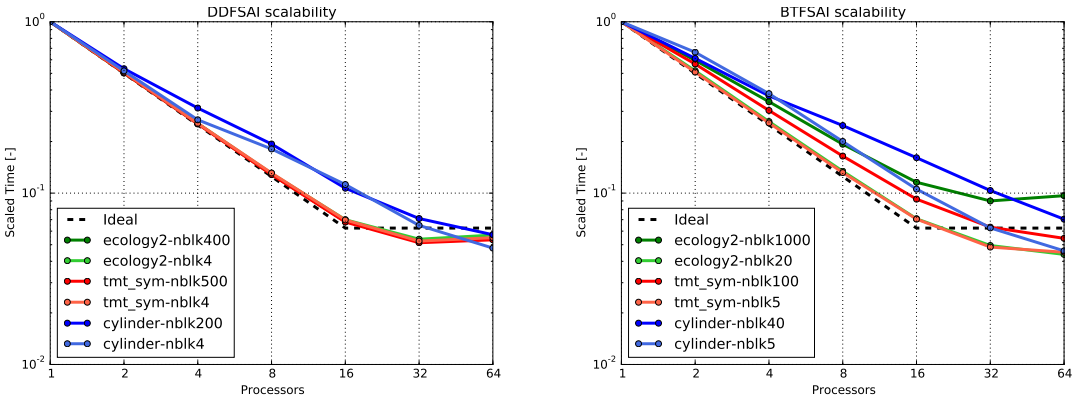

Figure 3: *Scalability profiles of DDFSAI and BTFSAI, respectively.*

IV. Conclusions

In this work two new multilevel preconditioners based on the FSAI algorithm were discussed, the DDFSAI and BTFSAI algorithms. They were applied in the serial and parallel solution of SPD linear problems arising from different engineering applications in order to test their performance and scalability. The results show that the BTFSAI technique is very promising in terms of decreasing the total number of iterations and time for achieving the solution. Further, with the DDFSAI technique, the computational pain is smaller, but stability and scalability are generally better. We remark that these strategies are under development and further computational improvements are being addressed, for instance a hybrid MPI/OpenMP implementation.

References

Michele Benzi. Preconditioning techniques for large linear systems: A survey. *Journal of Computational Physics*, 182(2):418 – 477, 2002.

Tony F. Chan and Tarek P. Mathew. Domain decomposition algorithms. *Acta Numerica*, 1994.

E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference*, ACM '69, pages 157–172, New York, NY, USA, 1969. ACM.

Timothy A. Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1):1:1–1:25, December 2011.

Victorita Dolean, Pierre Jolivet, and Frédéric Nataf. *An Introduction to Domain Decomposition Methods: Algorithms, Theory and Parallel Implementation*. SIAM, 2015.

Massimiliano Ferronato. Preconditioning for sparse linear systems at the dawn of the 21st century: History, current developments, and future perspectives. *ISRN Applied Mathematics*, 2012.

Marcus J. Grote and Thomas Huckle. Parallel preconditioning with sparse approximate inverses. *SIAM Journal of Scientific Computing*, 18(3):838–853, May 1997.

Thomas Huckle. Factorized sparse approximate inverses for preconditioning. *J. Supercomput.*, 25 (2):109–117, June 2003.

Carlo Janna and Massimilano Ferronato. Adaptive pattern research for block fsai preconditioning. *SIAM J. Sci. Comput.*, 33, 2011.

Carlo Janna, Massimiliano Ferronato, Flavio Sartoretto, and Giuseppe Gambolati. FSAIPACK: A software package for high-performance factored sparse approximate inverse preconditioning. *ACM Trans. Math. Softw.*, 41(2):10:1–10:26, February 2015.

George Karypis and Vipin Kumar. Multilevel k-way partitioning scheme for irregular graphs. *J. Parallel Distrib. Comput.*, 48(1):96–129, January 1998.

Lois Curfman Mcinnes, Barry Smith, Hong Zhang, and Richard Tran Mills. Hierarchical krylov and nested krylov methods for extreme-scale computing. *Parallel Computing*, January 2014.