Procedia Computer Science

Volume 51, 2015, Pages 266–275

ICCS 2015 International Conference On Computational Science

A novel Factorized Sparse Approximate Inverse preconditioner with supernodes

Massimiliano Ferronato[1], Carlo Janna[1], and Giuseppe Gambolati[1]

Department ICEA, University of Padova, Padova, Italy
massimiliano.ferronato@unipd.it

Abstract

Krylov methods preconditioned by Factorized Sparse Approximate Inverses (FSAI) are an efficient approach for the solution of symmetric positive definite linear systems on massively parallel computers. However, FSAI often suffers from a high set-up cost, especially in ill-conditioned problems. In this communication we propose a novel algorithm for the FSAI computation that makes use of the concept of supernode borrowed from sparse LU factorizations and direct methods.

Keywords: linear systems, preconditioned iterative methods, approximate inverse, parallel computing

1 Introduction

Current computer simulations, especially related to earth models, may easily require the computation of several millions or even billions of unknowns, and the efficient solution to the sequences of sparse linear systems of equations

$$A\boldsymbol{x} = \boldsymbol{b} \tag{1}$$

may represent one of the most, and often the most, expensive tasks. Roughly speaking, two main classes of algorithms are available, namely direct [3] and iterative methods [14]. The former are based on the sparse Gaussian elimination procedure and obtain the solution $\mathbf{x}$ in (1) with a number of operations that can be predicted a priori. The latter start from a tentative guess solution and improve it through a problem- and algorithm-dependent number of iterations. The flop count of direct methods is typically much higher than that of efficient iterative methods, especially in sparse three dimensional problems. However, direct methods are still competitive because of the use of the so-called supernodes, i.e., clusters of unknowns that are grouped and processed together with cache efficient dense linear algebra kernels. By distinction, iterative methods typically use an indirect memory indexing that prevents an intensive use of the processor cache. Recently, the supernodes have already been successfully used with iterative methods in the field of incomplete LU preconditioning [6, 15].

In recent years the growing diffusion of parallel computers has fostered the development of algorithms able to take advantage of the potential offered by multiple processors. Iterative

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2015

doi:10.1016/j.procs.2015.05.238

methods are in principle much more attractive than direct methods because of their intrinsic better parallelism. The bottleneck, however, is generally the preconditioner, that can be expensive in terms of both computation and application to a vector. Approximate inverses can address the demand for efficient parallel preconditioner [1, 5, 10, 8]. They can be computed in several ways, e.g., through a bi-orthogonalization process or a Frobenius norm minimization, and are applied to a vector with a matrix-by-vector multiplication. Even though approximate inverses can be effective in a large number of problems, with ill-conditioned matrices their computation can be very time consuming and may represent a barrier to their diffusion in industrial applications.

In the present communication, we address such a drawback by introducing the concept of supernodes to accelerate the set-up stage of the Factorized Sparse Approximate Inverse (FSAI) preconditioner [13] for symmetric positive definite (SPD) problems. Identifying a supernodal structure in the system matrix A may help reduce significantly both the cost for computing FSAI and the iteration count to converge, with an improvement of the overall FSAI performance.

2 The FSAI preconditioner

The classical FSAI preconditioner M^{-1} for an SPD matrix A reads:

$$M^{-1} = G^T G \simeq A^{-1} \tag{2}$$

where G is computed minimizing the Frobenius norm

$$\|I - GL\|_F \tag{3}$$

over the set $\mathcal{W}_\mathcal{S}$ of matrices having a prescribed lower triangular non-zero pattern $\mathcal{S}$. The matrix L in (3) is the exact lower triangular factor of A and actually is not required to get G. In fact, differentiating (3) with respect to the G entries g_{ij} and setting to zero give:

$$[GA]_{ij} = [L^T]_{ij} \qquad \forall\ (i,j) \in \mathcal{S} \tag{4}$$

Since L^T is upper triangular and $\mathcal{S}$ is a lower triangular pattern, the matrix equation (4) can be rewritten as:

$$[GA]_{ij} = \begin{cases} 0 & i \neq j, \quad (i,j) \in \mathcal{S} \\ l_{ii} & i = j \end{cases} \tag{5}$$

where $[\cdot]_{ij}$ is the entry in row i and column j of the matrix between square brackets, and l_{ii} is the i-th diagonal element of L. L is unknown, so l_{ii} in (5) is replaced by 1. The matrix $\widetilde{G}$ computed by solving:

$$[\widetilde{G}A]_{ij} = \delta_{ij} \tag{6}$$

with δ_{ij} the Kronecker delta, is scaled as:

$$G = D\widetilde{G}, \qquad D = [\mathrm{diag}(\widetilde{G})]^{-1/2} \tag{7}$$

thus obtaining the matrix G used in the FSAI definition (2). The scaling (7) ensures that the diagonal entries of the preconditioned matrix GAG^T are unitary, and its Kaporin condition number is minimum over all matrices $G \in \mathcal{W}_\mathcal{S}$ [11, 12]. The Kaporin number of an SPD matrix is defined as the ratio between the arithmetic and geometric mean of its eigenvalues and gives a measure of the number of iterations required by the Preconditioned Conjugate Gradient (PCG) method to converge. The FSAI preconditioner is very robust as it can be computed

for any choice of the non-zero pattern $\mathcal{S}$ and the resulting preconditioned matrix is SPD by construction.

The main computational cost in the FSAI set-up is the solution of the sequence of n small dense linear systems, n being the size of A, arising from the component-wise equation (5). In particular, it depends on the non-zero pattern $\mathcal{S}$ that can be selected either statically, i.e., prescribed a priori, or dynamically, i.e., generated during the computation of G. In this work we introduce the use of supernodes in the static FSAI computation.

2.1 Supernodes

The effective a priori selection of $\mathcal{S}$ usually is not trivial. Inspection of the Neumann power series of A^{-1} suggests the use of non-zero patterns of small powers of A, recalling that for FSAI only the lower triangular pattern is needed. This idea may be effectively combined with both prefiltration [2] and recursion [7].

Recalling equation (5), it can be observed that the i-th row of $\widetilde{G}$ is computed by solving a dense $m_i \times m_i$ system, with m_i the number of non-zeroes assigned to row i. As a major consequence, the cost for computing G is asimptotically proportional to $\sum_{i=1}^{n} m_i^3$ and grows very quickly with m_i. Supernodes aim at reducing such a cost by aggregating the solution of the systems required by different rows in one dense system only. In the sequel, we will often use the term "node" to denote a row of A, as it is typically done in graph theory.

Suppose that $\mathcal{S}$ is given and denote by $\mathcal{P}_i$ the set of column indices belonging to the i-th row of $\mathcal{S}$:

$$\mathcal{P}_i = \{j : (i,j) \in \mathcal{S}\} \tag{8}$$

$A[\mathcal{P}_i, \mathcal{P}_i]$ is the submatrix of A consisting of the elements with row and column indices in $\mathcal{P}_i$. Denote by $\widetilde{\boldsymbol{g}}_i$ and $\boldsymbol{g}_i$ the dense vectors collecting the non-zero entries prescribed in the i-th row of $\widetilde{G}$ and G, respectively. With this notation the component-wise equation (6) is equivalent to:

$$A[\mathcal{P}_i, \mathcal{P}_i]\widetilde{\boldsymbol{g}}_i = \boldsymbol{e}_{m_i} \qquad\qquad i = 1, \ldots, n \tag{9}$$

where $\boldsymbol{e}_{m_i} = [0, 0, \ldots, 1]^T$ is the m_i-th vector of the canonical basis of $\mathbb{R}^{m_i}$. The vector $\boldsymbol{g}_i$ is $\widetilde{\boldsymbol{g}}_i$ scaled by the square root of its last entry:

$$\boldsymbol{g}_i = \frac{\widetilde{\boldsymbol{g}}_i}{\sqrt{\widetilde{g}_{i,m_i}}} \tag{10}$$

For any row i, the most expensive operation is the solution of the system (9). If two rows, say i_1 and i_2, have a similar pattern, solving an enlarged system obtained from merging $\mathcal{P}_{i_1}$ and $\mathcal{P}_{i_2}$ might be less expensive than solving two smaller systems. Let us clarify this idea using the example of Figure 1. Assume $i_1 < i_2$ and $\widetilde{\mathcal{P}} = \mathcal{P}_{i_1} \bigcup \mathcal{P}_{i_2}$, with m_{i_1}, m_{i_2} and $\widetilde{m}$ the cardinalities of $|\mathcal{P}_{i_1}|$, $|\mathcal{P}_{i_2}|$ and $|\widetilde{\mathcal{P}}|$, respectively. The vectors $\widetilde{\boldsymbol{g}}_{i_1}$ and $\widetilde{\boldsymbol{g}}_{i_2}$ can be found simultaneously by solving the multiple right-hand side system:

$$A[\widetilde{\mathcal{P}}, \widetilde{\mathcal{P}}] \left\{\widetilde{\boldsymbol{g}}_{i_1}\ \widetilde{\boldsymbol{g}}_{i_2}\right\} = \{\boldsymbol{b}_1\ \boldsymbol{b}_2\} \tag{11}$$

with $\boldsymbol{b}_1, \boldsymbol{b}_2 \in \mathbb{R}^{\widetilde{m}}$. Finding the explicit expression for $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ is easy. Define the two sets:

$$\widetilde{\mathcal{P}}_1 = \{j : j \in \widetilde{\mathcal{P}} \text{ and } j \leq i_1\} \qquad \widetilde{\mathcal{P}}_2 = \widetilde{\mathcal{P}} \setminus \widetilde{\mathcal{P}}_1 \tag{12}$$

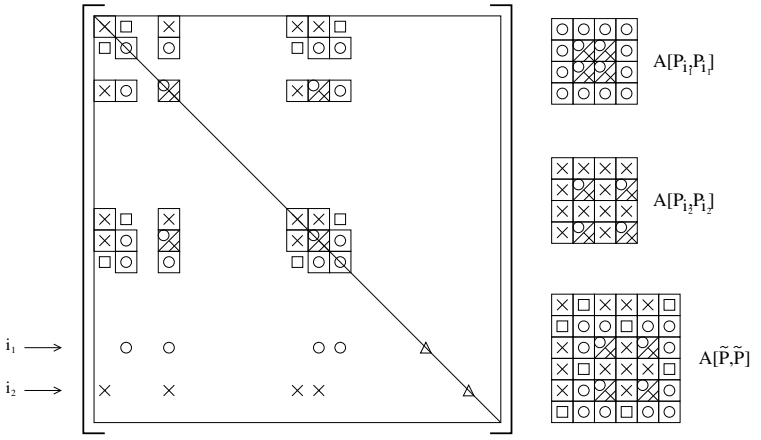

Figure 1: Schematic representation of the merging process of two rows, i_1 and i_2. The entries of A corresponding to $\mathcal{P}_{i_1}$ and $\mathcal{P}_{i_2}$ are denoted by circles and crosses, respectively. Squares denote the A entries that do not belong to $\mathcal{P}_{i_1}$ or $\mathcal{P}_{i_2}$ but are collected in $A[\widetilde{\mathcal{P}}, \widetilde{\mathcal{P}}]$.

with cardinalities $\widetilde{m}_1$ and $\widetilde{m}_2$, respectively, and write (11) in the block form:

$$\begin{bmatrix} A[\widetilde{\mathcal{P}}_1, \widetilde{\mathcal{P}}_1] & A[\widetilde{\mathcal{P}}_1, \widetilde{\mathcal{P}}_2] \\ A[\widetilde{\mathcal{P}}_2, \widetilde{\mathcal{P}}_1] & A[\widetilde{\mathcal{P}}_2, \widetilde{\mathcal{P}}_2] \end{bmatrix} \begin{Bmatrix} \widetilde{\boldsymbol{g}}_{i_1,1} & \widetilde{\boldsymbol{g}}_{i_2,1} \\ \widetilde{\boldsymbol{g}}_{i_1,2} & \widetilde{\boldsymbol{g}}_{i_2,2} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{b}_{1,1} & \boldsymbol{b}_{2,1} \\ \boldsymbol{b}_{1,2} & \boldsymbol{b}_{2,2} \end{Bmatrix} \tag{13}$$

Then, we can set:

$$\begin{aligned} \boldsymbol{b}_{1,1} &= \boldsymbol{e}_{\widetilde{m}_1} & \boldsymbol{b}_{2,1} &= \boldsymbol{0} \\ \boldsymbol{b}_{1,2} &= H_{21} L_{11}^{-1} \boldsymbol{e}_{\widetilde{m}_1} & \boldsymbol{b}_{2,2} &= \boldsymbol{e}_{\widetilde{m}_2} \end{aligned} \tag{14}$$

where $\boldsymbol{e}_{\widetilde{m}_1}$ and $\boldsymbol{e}_{\widetilde{m}_2}$ are the $\widetilde{m}_1$-th and the $\widetilde{m}_2$-th vectors of the canonical basis of $\mathbb{R}^{\widetilde{m}_1}$ and $\mathbb{R}^{\widetilde{m}_2}$, respectively, and H_{21} and L_{11}^{-1} are part of the block factorization of $A[\widetilde{\mathcal{P}}, \widetilde{\mathcal{P}}]$:

$$\begin{bmatrix} A[\widetilde{\mathcal{P}}_1, \widetilde{\mathcal{P}}_1] & A[\widetilde{\mathcal{P}}_1, \widetilde{\mathcal{P}}_2] \\ A[\widetilde{\mathcal{P}}_2, \widetilde{\mathcal{P}}_1] & A[\widetilde{\mathcal{P}}_2, \widetilde{\mathcal{P}}_2] \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ H_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & H_{21}^T \\ 0 & L_{22}^T \end{bmatrix} \tag{15}$$

Note that the computation of $\boldsymbol{b}_{1,2}$ is relatively inexpensive since $\boldsymbol{e}_{\widetilde{m}_1}$ has only one non-zero component. This procedure can be generalized to merge an arbitrary number of rows.

It is now necessary to define a rule for deciding when it is convenient to aggregate two or more rows into a supernode. The computational burden to gather and solve from scratch a linear system of size m with l right-hand sides is:

$$c(m, l) = \sum_{i=0}^{3} a_i m^i + l \sum_{i=0}^{2} b_i m^i \tag{16}$$

where the coefficients a_i and b_i depend on the hardware. Their values can be found by performing a least square regression on an ensemble of test runs. A supernode $\widetilde{i}$ merging i_1 and i_2 is formed if $c(\widetilde{m}, 2) < c(m_1, 1) + c(m_2, 1)$, otherwise i_1 and i_2 are computed separately. A similar argument is used if we want to merge the node i_k to the supernode $\widetilde{i}$ that already aggregates $\widetilde{l}$ nodes. Let m_k and $\widetilde{m}$ denote the cardinality of $\mathcal{P}_{i_k}$ and $\widetilde{\mathcal{P}}$, respectively, and h the number

of mismatches, that is the number of column indices lying in $\mathcal{P}_{i_k}$ but not in $\widetilde{\mathcal{P}}$. The node i_k is included into the supernode $\widetilde{i}$ if:

$$c(\widetilde{m}+h,\widetilde{l}+1) < c(\widetilde{m},\widetilde{l}) + c(m_k,1) \tag{17}$$

Condition (17) reduces the FSAI setup time and generally accelerates the PCG convergence. In fact, the use of supernodes enlarges the non-zero pattern $\mathcal{S}$, i.e., $\mathcal{S} \subseteq \widetilde{\mathcal{S}}$, and the new preconditioner minimizes the Kaporin condition number over a wider set. As a drawback, the cost per iteration may grow because of the larger density. Hence, condition (17) can be relaxed by introducing the parameter α:

$$c(\widetilde{m}+h,\widetilde{l}+1) < \alpha\left[c(\widetilde{m},\widetilde{l}) + c(m_k,1)\right] \tag{18}$$

Finally, we define the score function:

$$s_\alpha(\widetilde{m},\widetilde{l},m_k,h) = \alpha\left[c(\widetilde{m},\widetilde{l}) + c(m_k,1)\right] - c(\widetilde{m}+h,\widetilde{l}+1) \tag{19}$$

A supernode is generated if $s_\alpha > 0$. The larger s_α, the more cost effective the merging.

To find the supernodal structure a systematic comparison of all the nodes in the adjacency graph is necessary. For this task an efficient gready strategy based on level set traversals can be employed. Level sets of a graph are defined recursively:

- *Basis*: level 0 is a simple set of nodes;
- *Recursion*: level $k+1$ includes all the neighbouring nodes of level k (adj(k)) not belonging to level $k-1$.

Our procedure works as follows. Let $\mathcal{S}$ and $\mathcal{Q} = \emptyset$ be the initial static pattern and list of supernodes, respectively. Level 0 is represented by node n only. Visit all the nodes in the adjacency graph following the level set hierarchy and inspecting the elements within a level from the last to the first with respect to the original ordering. Level 0 is trivially processed by adding node n to the list of supernodes $\mathcal{Q}$. Level k is processed by exploring its nodes in reverse order. Each node i is compared to the supernodes listed in $\mathcal{Q}$ to determine the maximum s_α value. If the maximum $s_\alpha > 0$ the current node i is merged, otherwise i is pushed into the head of $\mathcal{Q}$ as a new supernode. The traversal is carried out in reverse order because it is most likely that lower rows include upper rows as $\mathcal{S}$ is lower triangular. A drawback of this procedure can occur when the list of supernodes $\mathcal{Q}$ becomes too long and comparing a new node with all the supernodes may be too expensive. It can be observed that a node in a level generally shares more column indices with the nodes from adjacent levels than those from far levels. Hence, it is likely for a node to achieve the best score with a newly generated supernode. As a major consequence, we can consider in our comparison the last l_{max} supernodes only. A reasonable value for l_{max} is not overly difficult to set. With $l_{max} = 30$, the cost of the above procedure is negligible with respect to the FSAI set-up time, while larger l_{max} values slow down the procedure without producing less supernodes.

3 Numerical results

The FSAI computational performance using supernodes is investigated in a set of large size test problems arising from different applications. All the test matrices are publicly available

Name	Size	# of non-zeroes	Problem description
Sebe	742,793	37,138,461	3D pressure-temperature in porous media
Emilia_923	923,136	40,373,538	3D geomechanical reservoir simulation
Audikw_1	943,695	77,651,847	3D automotive structural problem
Serena	1,391,349	64,531,701	3D structural mechanics
Bump_2911	2,911,419	130,378,257	3D geomechanical reservoir simulation
Queen_4147	4,147,110	329,499,288	3D structural problem

Table 1: Size, number of non-zeroes and brief description of the test matrices.

a_0	a_1	a_2	a_3
$0.527655 \cdot 10^{-5}$	$0.132448 \cdot 10^{-5}$	$0.131749 \cdot 10^{-7}$	$0.230335 \cdot 10^{-9}$
b_0	b_1	b_2	
$0.153699 \cdot 10^{-5}$	$0.618331 \cdot 10^{-7}$	$0.317156 \cdot 10^{-8}$	

Table 2: Cost model parameters, according to equation (16), for the IBM-BG/Q FERMI supercomputer.

from the University of Florida Sparse Matrix Collection [4] and are summarized in Table 1. In all test cases the Preconditioned Conjugate Gradient (PCG) solver is used with the right-hand side computed so that the solution is the unitary vector. The iterations are completed when the relative residual is smaller than 10^{-10}. The computational performance is evaluated in terms of the number of iterations n_{iter}, the wall clock time in seconds T_p and T_s needed for the preconditioner computation and the PCG convergence, respectively, with the total time $T_t = T_p + T_s$. The algorithm is coded in Fortran90 with OpenMP directives to exploit shared memory architectures. All tests are performed on the IBM-BG/Q FERMI at the CINECA Centre for High Performance Computing, equipped with IBM PowerA2 processors at 1.6 GHz with 10,240 nodes, 163,840 computing cores, and 1 Gbyte/core of RAM. For each test case a parallel run using 16 cores is performed. The cost model parameters a_i and b_i (equation (16)) for the IBM-BG/Q FERMI supercomputer are given in Table 2.

Let us consider the SPD matrix EMILIA_923. Table 3 shows the outcome of a set of simulations performed with this test case. The user-specified parameters κ, τ_1, and τ_2 required to set the static pattern have the following meaning:

- κ is the power of the lower triangular part of A used as reference pattern;
- τ_1 is the pre-filtration tolerance used to sparsify A before computing the pattern of A^κ;
- τ_2 is the post-filtration tolerance used to sparsify G after its computation.

For more details on the meaning and the suggested range of the user-specified parameters, see Janna et al. [9].

As expected, the use of supernodes decreases the cost for computing the preconditioner, but the most significant impact is on the PCG acceleration due to the increase of the preconditioner density μ defined as the ratio between the number of non-zeroes of G and A:

$$\mu = \frac{\mathrm{nnz}(G)}{\mathrm{nnz}(A)} \tag{20}$$

Table 3 provides also the average size $dim_{\tilde{i}}$ of the supernodes, denoting the average number of rows collapsed to the same sparsity pattern. In the computation of the score s_α we have set

Parameters			FSAI				FSAI with supernodes				
κ	τ_1	τ_2	n_{iter}	T_p	T_s	T_t	$dim_{\tilde{i}}$	n_{iter}	T_p	T_s	T_t
1	10^{-3}	0.01	4758	5.1	200.7	205.8	10.3	3189	4.1	154.1	158.2
2	10^{-3}	0.03	3143	22.7	146.3	169.0	6.4	2164	16.0	103.7	119.7
3	10^{-2}	0.05	2315	11.2	97.8	109.0	5.6	1720	9.9	75.9	85.8
4	10^{-2}	0.05	1749	34.9	77.5	112.4	6.1	1372	29.4	62.9	92.3
5	10^{-1}	0.03	1899	4.6	76.0	80.6	3.2	1616	2.8	66.9	69.7

Table 3: `EMILIA_923` test case: FSAI computational performance with the standard pattern (on the left) and the supernode approach (on the right) in a set of simulations. $dim_{\tilde{i}}$ is the average size of the supernodes.

Test case	S	$\max R_t$	$\min R_t$	R_i	R_p	R_s	R_t	R_μ
`Sebe`	62.5%	1.85	0.83	1.06	1.19	0.97	1.09	0.81
`Emilia_923`	93.7%	2.70	0.98	1.27	1.49	1.20	1.32	0.78
`Audikw_1`	57.8%	1.56	0.83	1.01	1.12	0.95	1.03	0.70
`Serena`	72.0%	1.82	0.95	1.03	1.20	1.01	1.08	0.85
`Bump_2911`	80.0%	1.75	0.88	1.41	1.19	1.25	1.23	0.63
`Queen_4147`	79.4%	1.92	0.92	1.23	1.12	1.14	1.12	0.58

Table 4: Summary of the performance obtained with and without using supernodes.

$\alpha = 1$. Usually, the larger $dim_{\tilde{i}}$, the smaller the iteration count. In the previous example the overall gain in terms of total wall clock time can be up to 25%.

The supernode efficiency is problem-dependent. An extensive numerical experimentation has been performed on the set of matrices of Table 1 varying for each test case the user-specified parameters κ, τ_1 and τ_2. The total wall clock time obtained using FSAI with and without supernodes is shown in Figure 2. In each diagram the experiments are reported according to the ascending order of T_t. The numerical results show that supernodes are generally helpful. In particular, the supernodes help reduce the variability range of the PCG performance with κ, τ_1 and τ_2, thus decreasing the solver sensitivity to the user-defined parameters.

The global outcome of the numerical experiments for the different test cases is summarized in Table 4. The meaning of the symbols is as follows:

- S: percentage of experiments where the supernodes prove effective in increasing the computational efficiency;
- $\max R_t$: maximal T_t ratio without and with supernodes, i.e., a measure of the best computational gain obtained from the tests;
- $\min R_t$: minimal T_t ratio without and with supernodes, i.e., the most unfavourable case;
- R_i: average n_{iter} ratio without and with supernodes;
- R_x, with $x = p, s, t$: average T_x ratio without and with supernodes;
- R_μ: average μ ratio without and with supernodes.

Table 4 shows that supernodes yield a performance improvement in the majority of experiments. `Emilia_923` is the most favourable test case with the supernodes allowing for a best computational gain of almost 3, and with most of the other matrices the best gain is not far

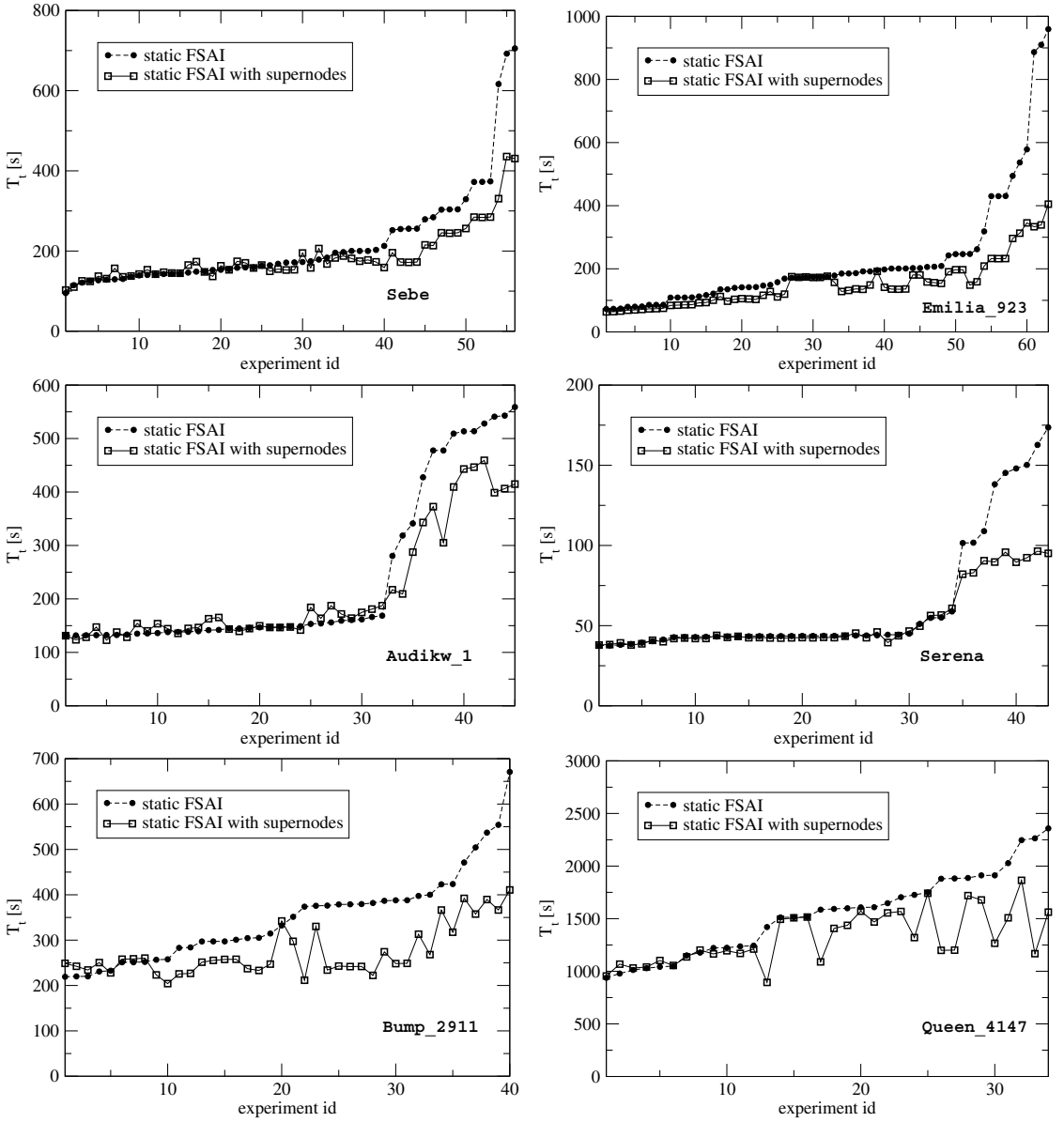

Figure 2: Static FSAI: total wall clock time [s] with and without supernodes. For each experiment id different user-specified parameters have been used within a plausible range.

from 2. Where supernodes are not convenient, the efficiency loss is limited to a small percentage, say 10-15% in the worst cases. Also note that on the average the iteration count and the time for computing the preconditioner is generally reduced with supernodes, while the time for iterating may increase because of a denser FSAI preconditioner.

4 Conclusions

The technique of supernodes is widely and successfully used in the parallel solution of sparse linear systems by direct methods. The concept is here developed for the FSAI preconditioning of SPD systems. In the present work, we have introduced the supernodes in static FSAI and performed a numerical experimentation of their performance in a set of test problems.

The supernodal structure is generated evaluating the cost for computing each row of G. A set of rows are merged into a supernode if the cost for solving a larger multiple right-hand side system is lower than that required for several smaller systems. This procedure is helpful for reducing the FSAI set-up time, improving the preconditioner quality and decreasing the iteration count for the PCG convergence. The use of supernodes is cost effective in 75% of the runs carried out, allowing for a PCG scheme that may be up to about three times faster than the standard FSAI. In cases where supernodes are not convenient, the total wall-clock time increase is limited to 10-15% only. Moreover, the use of supernodes reduce to some extent the solver sensitivity to the user-specified parameters, thus improving the overall robustness of the final PCG scheme.

Currently, the supernodal concept is going to be extended to two FSAI variants: (i) for non-symmetric and/or symmetric indefinite matrices, and (ii) using a dynamic computation of the FSAI non-zero pattern, that is generally more efficient than static FSAI especially in ill-conditioned problems.

References

References

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

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

[3] T. A. Davis. *Direct Methods for Sparse Linear Systems*. Philadelphia (PA), SIAM, 2006.

[4] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38 (2011), pp. 1–25.

[5] M. Ferronato. Preconditioning of sparse linear systems at the dawn of the 21st century: history, current developments, and future perspectives. *ISRN Appl. Math.*, 2012 (2012), article ID 127647, doi: 10.5402/2012/127647.

[6] A. Gupta and T. George. Adaptive techniques for improving the performance of incomplete factorization preconditioning. *SIAM J. Sci. Comput.*, 32 (2010), pp. 84–110.

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

[8] C. Janna, M. Ferronato and G. Gambolati. Enhanced block FSAI preconditioning using domain decomposition techniques. *SIAM J. Sci. Comput.*, 35 (2013), pp. S229–S249.

[9] C. Janna, M. Ferronato, F. Sartoretto and G. Gambolati. FSAIPACK: a software package for high performance FSAI preconditioning. *ACM Trans. Math. Softw.*, 41 (2015), pages to be defined.

[10] Z. Jia and B. Zhu. A power sparse approximate inverse preconditioning procedure for large sparse linear systems. *Numer. Lin. Alg. Appl.*, 16 (2009), pp. 259–299.

[11] I. E. Kaporin. A preconditioned conjugate gradient method for solving discrete analogs of differential problems. *Diff. Equat.*, 26 (1990), pp. 897–906.

[12] I. E. Kaporin. New convergence results and preconditioning strategies for the conjugate gradient method. *Numer. Lin. Alg. Appl.*, 1 (1994), pp. 179–210.

[13] L. Y. Kolotilina and A. Y. Yeremin. Factorized sparse approximate inverse preconditioning. I. Theory. *SIAM J. Mat. Anal. Appl.*, 14 (1993), pp. 45–58.

[14] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Philadelphia (PA), SIAM, 2003.

[15] N. Vannieuwenhoven and K. Meerbergen. IMF: An incomplete multifrontal LU-factorization for element-structured sparse linear systems. *SIAM J. Sci. Comput.*, 35 (2013), pp. A270–A293.