Science & Technology Facilities Council

Preprint

RAL-P-2016-006

On the use of iterative methods and blocking for solving sparse triangular systems in incomplete factorization preconditioning

E Chow, JA Scott

March 2016

Submitted for publication in SIAM Journal on Scientific Computing

RAL Library
STFC Rutherford Appleton Laboratory
R61
Harwell Oxford
Didcot
OX11 0QX

Tel: +44(0)1235 445384
Fax: +44(0)1235 446403
email: libraryral@stfc.ac.uk

ISSN 1361- 4762

On the use of iterative methods and blocking for solving sparse triangular systems in incomplete factorization preconditioning

Edmond Chow[1] and Jennifer Scott[2]

ABSTRACT

When using incomplete factorization preconditioners with an iterative method to solve large sparse linear systems, each application of the preconditioner involves solving two sparse triangular systems. These triangular systems are challenging to solve efficiently on computers with high levels of concurrency. On such computers, it has recently been proposed to use Jacobi iterations to solve the triangular systems from incomplete factorizations. These Jacobi iterations may not always converge, or converge quickly enough, for all problems. Thus in this paper we investigate the range of problems for which this approach is effective. We also show that by using *block* Jacobi relaxation, we can extend the range of problems for which such an approach can be effective.

Keywords: sparse linear systems, triangular solves, iterative solvers, preconditioning.

AMS(MOS) subject classifications: 65F05, 65F50

---

[1] School of Computational Science and Engineering, College of Computing, Georgia Institute of Technology, 266 Ferst Drive, Atlanta, GA 30332-0765, USA. echow@cc.gatech.edu
This material is based upon work supported by the U.S. DOE Office of Science, Office of Advanced Scientific Computing Research, Applied Mathematics program under Award Number DE-SC-0012538.

[2] Scientific Computing Department, STFC Rutherford Appleton Laboratory, Harwell Campus, Oxfordshire, OX11 0QX, UK. jennifer.scott@stfc.ac.uk
Supported by EPSRC grant EP/I013067/1.

March 16, 2016

1 Introduction

As computers increasingly rely on various forms of parallelism to obtain high performance, the design and performance optimization of large-scale numerical algorithms must focus on the efficient exploitation of parallel architectures. Unfortunately, simple algorithms frequently become remarkably complex when implemented in parallel and the most efficient parallel algorithm may not be the one that is most intuitive.

In this paper, we investigate a non-intuitive approach to solving the sparse triangular systems that arise when using incomplete factorization preconditioners with an iterative method for solving sparse linear systems. The conventional method of solving triangular systems is to use forward or backward substitution and this can be parallelized using level scheduling [2, 16, 24]. Here we investigate using an iterative method—in particular, Jacobi relaxation. The use of an iterative method is feasible when an approximate solution is acceptable, as in the case of preconditioning.

In recent previous work, this approach demonstrated significant reductions in total solution time for the preconditioned conjugate gradient (PCG) method on highly parallel architectures such as Intel Xeon Phi co-processors and graphics processing units (GPUs) [3, 7], even though additional PCG iterations may be required to achieve the requested accuracy. The improved speed for each triangular solve is because, for some problems, particularly with high levels of fill, level scheduling is unable to reveal sufficient parallelism to fully exploit the GPU hardware. On the other hand, Jacobi relaxation, which primarily relies on the sparse-matrix vector product (SpMV) operation to compute a residual vector, is highly parallel and can exploit the substantial efforts that have been invested in optimizing SpMV on various parallel architectures.

Although the iterative triangular solve approach can result in significant speedups on highly parallel architectures, the approach does not work on all problems. It is possible for the iterations on the triangular systems to converge too slowly and thus be uncompetitive with the conventional level-scheduled substitution method. The goal of this paper is to investigate the range of applicability of using iterative triangular solves for incomplete factorization preconditioning by testing it with a large set of sparse symmetric positive definite (SPD) linear systems. Further, we introduce the idea of using block Jacobi iterations, which improves the robustness of the iterative approach. Our hypothesis is that for matrices that model many types of physical systems, especially partial differential equations, an iterative approach to solving the systems involving its triangular factors can be effective. In particular, although these matrices may not be diagonally dominant and may be ill-conditioned, many have *relatively* large diagonal entries compared to the off-diagonal entries. If the incomplete factorizations of these matrices are stable, they are also likely to have relatively large diagonal entries. Systems with such matrices can typically be solved efficiently by iterative methods. If convergence is poor, a block diagonal scaling can improve diagonal dominance.

Alternatives to sparse triangular solves for incomplete factorization preconditioning have been proposed before. One major alternative is to compute and use sparse approximate inverses of the incomplete factors, so that preconditioning reduces to SpMV operations [5, 27]. Related to this is representing the inverse of a sparse triangular matrix as the product of sparse triangular factors [1, 22]. Another possible approach is to use Neumann series approximations to the inverse of the incomplete factors [5, 26]. Preconditioning is again reduced to a sequence of SpMV operations. This approach was found to be potentially competitive with other techniques for nonsymmetric problems but it lacks robustness, while for symmetric problems a number of more efficient alternatives are available [5]. The Neumann series technique is the same as the Jacobi relaxation approach investigated here if a diagonal scaling is first applied to the system in the former. Recently, based on the encouraging results in [7] for using Jacobi iterations to solve triangular systems, Bräckle and Huckle [6] have suggested the use of other stationary iterations for solving sparse triangular systems.

In the symmetric case, the amount of parallelism in level scheduled sparse triangular solves can be increased by using multicolor reordering of the rows and columns of the original matrix and computing the incomplete Cholesky factors of this reordered matrix [17]. However, multicolor reorderings generally give poorer PCG convergence results compared to other orderings such as the band reducing RCM ordering [8] or profile reducing Sloan ordering [25] (see, for example, [4, 10, 11, 13, 14, 21]). For some "easy" problems, the convergence rate may be degraded by 60 to 100% but this can be compensated for by the additional parallelism in the solves. For harder problems, however, multicolor reorderings may result in no convergence.

The rest of this paper is organized as follows. In Section 2, we describe the use of Jacobi and block Jacobi relaxation for the iterative solution of triangular systems. In particular, supervariable blocking and graph partitioning-based blocking techniques are introduced. Section 3 presents our experimental study and demonstrates the potential effectiveness of Jacobi solves using a large set of test problems. Concluding remarks are made in Section 4.

2 Jacobi and block Jacobi relaxation for triangular systems

Consider a triangular system of equations

$$Ry = c, \tag{2.1}$$

where R is either upper or lower triangular. In our application in which we want to solve the system $Ax = b$, R is an incomplete factor of A and, in particular, in the symmetric case, $R = L$ or L^T, where L is an incomplete Cholesky factor of A. The Jacobi iteration for solving (2.1) is

$$y_{k+1} = y_k + D^{-1}(c - Ry_k), \tag{2.2}$$

where D is the diagonal of R. We take the initial guess

$$y_0 = D^{-1}c. \tag{2.3}$$

We will refer to the number of Jacobi iterations performed as the number of *Jacobi sweeps*. The iteration matrix

$$G = I - D^{-1}R \tag{2.4}$$

is strictly triangular with a zero diagonal. Thus the iteration is guaranteed to converge. However, in practice, because G is triangular and thus non-normal, the iteration may diverge before converging. If this initial divergence stage is long and/or causes numerical overflow, then using Jacobi iterations will not be an effective method of solution. Thus effectiveness will depend on the degree of non-normality of G. Non-normality can be measured in different ways. For triangular matrices, having only small off-diagonal entries results in a small departure from non-normality.

Block Jacobi relaxation or, equivalently, block scaling of R, can reduce non-normality. If a block structure is imposed on R then it can be trivially written as the sum

$$R = \tilde{R} + D, \tag{2.5}$$

where $\tilde{R}$ is strictly block triangular and D is now block diagonal with i-th block D_{ii}. The iteration is the same as in (2.2), where the i-th block row of the iteration matrix (2.4) now has off-diagonal blocks $G_{ij} = D_{ii}^{-1}R_{ij}$ $(j \neq i)$. If the norm of D_{ii} is large, then the norm of the off-diagonal blocks will be small, as desired.

In the rest of this section, we discuss possible techniques for determining a blocking for R such that the diagonal blocks D_{ii} have large norm. These are generally derived from a blocking for A.

Observe that, in general, we seek a blocking in which the block size is small compared to the order n of A so as to limit the cost of applying block Jacobi. With small blocks, the cost of block Jacobi is not significantly greater than for scalar Jacobi because the main cost in both is the computation of the residual $c - Ry_k$. For small blocks, one possibility to assist with efficiency is to store the inverses D_{ii}^{-1} explicitly.

2.1 Simple matrix blocking

The simplest way to impose a blocking on R is to divide its rows such that each block of rows contains similar numbers of rows. The columns are partitioned conformally, so that the diagonal blocks D_{ii} are square. To help ensure that the norms of the diagonal blocks are large, a locality-preserving ordering such as RCM or Sloan can be applied to the original matrix A, which will tend to put nonzero entries near the diagonal of A and thus of R.

2.2 Supervariable blocking and supervariable amalgamation

If a matrix has a supervariable structure, then a natural blocking is to use this structure. The set of variables that correspond to a set of columns of A with the same sparsity pattern is called a *supervariable*. Supervariables occur frequently as a result of each node or element of a partial differential equation (PDE) discretization having multiple variables or degrees of freedom associated with it. These variables are typically tightly coupled, leading to dense diagonal blocks corresponding to supervariables with large norm. A supervariable blocking orders the rows and columns corresponding to a supervariable together in the matrix. Incomplete factorization is then applied to the matrix in this ordering.

If blocks larger than the size of the supervariables are desired, then larger blocks can be formed by amalgamating adjacent supervariables. In this case, it is important that adjacent supervariables represent nearby nodes or elements in the PDE discretization. This can be accomplished with a locality-preserving ordering applied to the supervariable structure (also called the condensed structure or quotient graph) before the supervariables are amalgamated.

In this paper, we use a different, simpler approach for what we call supervariable blocking, one that treats all matrices the same way, whether or not they have supervariable structure. This approach was convenient for us when testing a large number of SPD matrices from different applications. First, the matrix is reordered symmetrically using RCM ordering so that physically nearby variables are likely to be numbered together. This also tends to preserve any existing supervariable structure, because variables belonging to a supervariable will tend to stay numbered together. Then, the supervariable structure is found using a cheap algorithm that simply compares the sparsity patterns of adjacent matrix columns. Supervariables may be amalgamated into blocks with a given minimum block size by merging adjacent supervariables.

2.3 Graph partitioning-based blocking

Graph partitioning can be used to find blockings for a matrix, where each block corresponds to a partition of the graph associated with the sparse matrix. Graph partitioning techniques minimize the number of edges shared between partitions, which corresponds to minimizing the number of nonzeros (and their magnitudes) that fall outside the diagonal blocks. Using supervariable blocking techniques, there can be a significant number of nonzero entries that do not lie within the diagonal block structure.

Graph partitioning techniques are described in terms of the (edge-weighted) graph of the matrix A. Typically, nodes in the graph that are a short distance apart correspond to tightly coupled variables and should belong to the same block. In some cases, for example those from anisotropic PDE problems, the strength of the coupling is less correlated with graph distance, so the values of the matrix entries need to be considered in the blocking.

In the graph partitioning approach, the partitions are found and the matrix A is reordered such that the variables in a partition are numbered together. The ordering of the partitions or blocks themselves in the matrix is not specified by a graph partitioning, but may affect the quality of the incomplete factorization preconditioner. The block structure may be ordered using a locality-preserving ordering, which typically improves the quality of the incomplete factorization, especially if the block sizes are small. An incomplete factorization is performed on A in the resulting ordering. The same blocking is applied to the incomplete factors.

A number of algorithms for blocking have been proposed in the past, including the PABLO family of algorithms [15] and the SCPRE algorithm of Duff and Kaya [12], which is based on Tarjan's algorithm for hierarchically decomposing a digraph into its subgraphs. General purpose graph partitioners may also be used, but these are aimed at computing relatively large subdomains, and we have found them to be less effective when, as in our case, small blocks are desired. PABLO is designed for constructing blockings for block Jacobi. It is based on a greedy approach to try to optimize the strength of the connections within the blocks. While it should be possible to obtain improved blockings by employing PABLO with carefully selected parameters, it is impractical to tune the parameters for each individual problem for a large test set. Instead, we propose two straightforward approaches that can be viewed as simple variants of the PABLO strategy.

In our first approach (Approach I), each node i is initially assigned to its own set S_i. The edges (i, j) in the graph are considered in descending weight order. If edge (i, j) is the next to be considered and $i \in S_{i0}$ and $j \in S_{j0}$ (with $i0 = i$ and $j0 = j$ initially), S_{j0} is merged into S_{i0} to form a larger set, provided the size of the merged set given by $|S_{i0}| + |S_{j0}|$ (where $|S|$ denotes the number of edges in the set S) does not exceed a user-prescribed maximum. When no further merges are possible, the nodes in each set are numbered consecutively and the corresponding reordered matrix then has a block structure with the l-th block corresponding to the l-th (merged) set.

If a matrix has supervariable structure, then to avoid splitting supervariables, the graph partitioning-based blocking may be applied to the supervariable structure. In this case, the weighted graph corresponds to the quotient graph (of the block structure) where the edge weights are the Frobenius norms of the supervariable blocks.

A potential weakness of Approach I is that it does not take into account the values of the weights of the merged edges. In our second approach (Approach II) when two sets S_{i0} and S_{j0} are merged with $i \in S_{i0}$ and $j \in S_{j0}$, the edge weights are unchanged unless both nodes i and j are adjacent to the same neighbor. In this case, the new edge is given a weight equal to the sum of the weights of the edges it replaces. This corresponds to heavy edge collapsing that is widely used in graph partitioning (see, for example, [18]). For efficiency, a priority queue is used to select the next edge; the priorities are updated as the algorithm progresses.

We note that graph partitioning approaches are most effective when partitions (or blocks) are large. For small partitions, the number of edges between partitions (and outside the diagonal blocks) remains large, and no algorithm is able to substantially reduce this number. However, graph partitioning approaches can be more effective than other approaches for anisotropic problems, as mentioned above, even if block sizes are small.

3 Numerical experiments

In our numerical experiments, we use matrices from the University of Florida Sparse Matrix (UFL) Collection [9]. In each test, the matrix is first scaled on the left and right by a diagonal matrix whose j-th diagonal entry is the 2-norm of column j of A [17]. We use IC(k) to denote an incomplete Cholesky factorization of level k; IC(0) is used unless indicated otherwise. Results are shown for single solves with the lower triangular factor from the incomplete factorization, as well as for repeated solves with the same factorization within a PCG solve. For the former, the components of the right-hand side are chosen from a uniform random distribution with mean zero. For the latter, the right-hand side b is such that the exact solution of the linear system $Ax = b$ is the vector of all ones. PCG is considered to have converged when the relative residual norm has decreased below 10^{-6}.

We define the *cost estimate* of the PCG iterations to be

$$\text{cost estimate} = n_{\text{iters}} \times \left[\text{nnz}(A) + 2 \times n_{\text{sweeps}} \times (\text{nnz}(L) + \text{nnz}(D))\right] / \text{nnz}(A), \tag{3.1}$$

where n_{iters} denotes the number of PCG iterations, n_{sweeps} is the number of Jacobi sweeps per triangular solve, and nnz(A), nnz(L), and nnz(D) are the numbers of entries in A, its incomplete factor L, and the (block) diagonal matrix D, respectively. For IC(0) without blocking, the cost estimate simplifies essentially to

$$\text{cost estimate} = n_{\text{iters}} \times \left[1 + n_{\text{sweeps}}\right],$$

which shows how rapidly n_{iters} must decrease as n_{sweeps} increases in order for the cost estimate to also decrease. The cost estimate is based on the assumption that sparse matrix-vector multiplication is memory bandwidth bound and that the amount of data transfered governs the execution time. The cost estimate is the number of words read in units of the number of words in the sparse matrix A, which we call "matvec loads."

3.1 Examples

Figure 3.1 presents results for the matrix Hook_1498. The matrix is preordered using RCM. The left figure shows the residual norm history for a solve with the lower triangular IC(0) factor using the Jacobi method. The residual norm is relative to the norm of the right-hand side vector. Zero sweeps corresponds to using the initial guess (2.3). The middle figure reports the number of PCG iterations required for convergence when 0 to 6 Jacobi sweeps are used to solve with the IC(0) factors. At 6 or more sweeps, the PCG iterations converge in 804 iterations, the same number as if exact triangular solves are used, corresponding to the conventional approach. The right figure plots the cost estimate. The convex shape of this curve shows that the optimum number of Jacobi sweeps for this problem is approximately 3. With this number of sweeps, the residual norm for a triangular solve is reduced by about 1.5 orders of magnitude (left figure). Once the PCG iteration count stops decreasing, the cost estimate must increase with an increasing number of Jacobi sweeps.

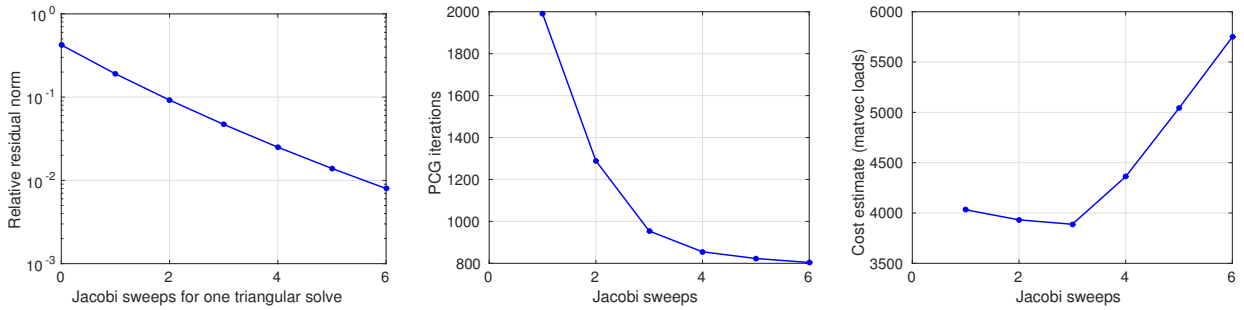

Figure 3.1: For Hook_1498, the effects of increasing the number of Jacobi sweeps.

Figure 3.2 shows the corresponding results for the matrix Geo_1438 (again, preordered using RCM). The behavior here is very different from that observed for the previous matrix. In particular, the cost estimate curve (right figure) is not convex but increases starting from zero Jacobi sweeps. There is no initial decrease in the cost estimate because the number of PCG iterations does not decrease fast enough with increasing numbers of Jacobi sweeps (middle figure). This is related to the slower convergence of the relative residual norm when Jacobi sweeps are applied to the triangular factors (left figure): the reduction after 3 sweeps is significantly less than an order of magnitude.

The behaviors illustrated by the above two matrices are common, but other behaviors are also possible. For example, the cost estimate can initially increase and then decrease below its initial value, before increasing again. Note that the cost estimate pessimistically assumes no cache effects, and actual timing curves may differ in shape depending on the hardware and the quality of the software implementation.

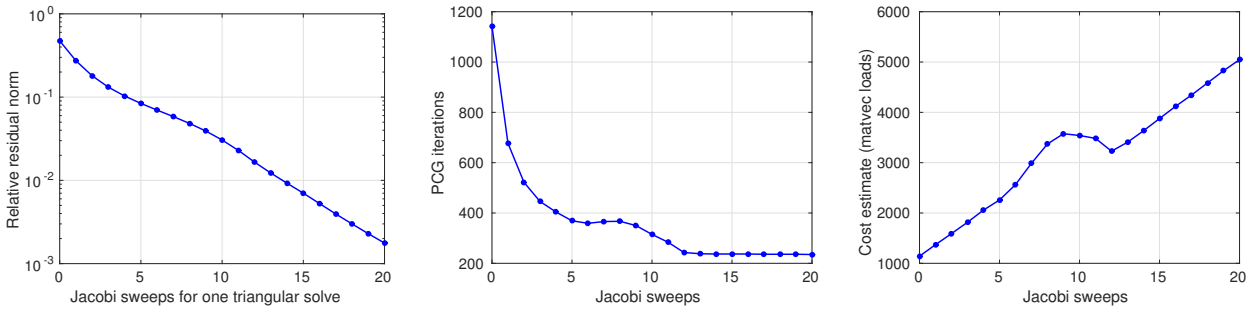

Figure 3.2: For Geo_1438, the effects of increasing the number of Jacobi sweeps.

In the previous two examples, the number of Jacobi sweeps needed to give PCG convergence that is essentially the same as that using exact triangular solves is small (6 and 12 sweeps, respectively). Moreover, for any number of sweeps, PCG eventually converged. For other problems, with a given fixed number of Jacobi sweeps, the relative residual norm may diverge and a large number of sweeps may be needed to give convergence. This is the case for the test matrix bcsstk24. Although this example is relatively small (of order 3562), it is challenging to solve: the matrix has large off-diagonal entries even after scaling, both diagonal preconditioning and symmetric Gauss-Seidel preconditioning fail to give convergence within a number of steps equal to the dimension of the matrix, and the level 0 incomplete Cholesky factorization breaks down (that is, a zero or negative pivot is encountered during the factorization). However, the level 1 incomplete Cholesky factorization exists and provides a good preconditioner, giving convergence in 56 PCG iterations. Thus we use IC(1) preconditioning for this matrix.

Figure 3.3 (left) shows the relative residual norm for solving with the lower triangular Cholesky factor of bcsstk24 using the Jacobi method. As the iteration matrix for this factor is highly non-normal, we observe a large divergence in the residual norm before convergence sets in. Using fewer than 160 Jacobi sweeps with IC(1) does not serve as a useful preconditioner. Figure 3.3 (right) shows the relative residual norm if *block* Jacobi is used with the same incomplete factor, using supervariable blocking with amalgamation and block sizes of 6, 12, and 24. (Note that bcsstk24 has a supervariable structure with block size of 6.) Here, there is still an initial divergence in the relative residual norm for a small number of Jacobi sweeps, but it is much smaller. The relative residual can be reduced by 1.5 orders of magnitude in approximately 20 sweeps, depending on the block size.

Figure 3.4 shows PCG iteration counts and the cost estimate for different numbers of block Jacobi sweeps. This example demonstrates the effectiveness of block Jacobi compared to scalar Jacobi iterations.

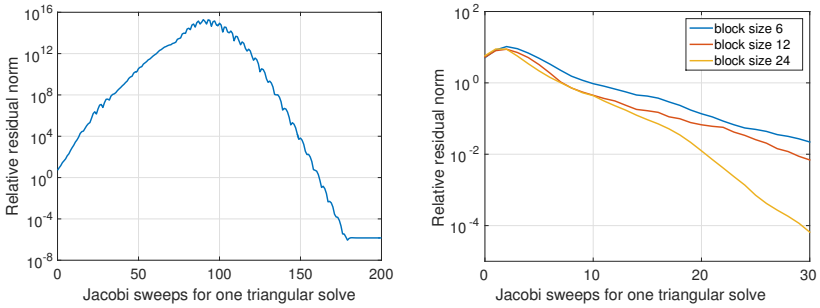

Figure 3.3: For bcsstk24, the relative residual norm for a triangular solve with the lower triangular IC(1) factor, without blocking (left) and with supervariable blocking and amalgamation (right).

3.2 Comprehensive results for Jacobi and block Jacobi iterations

This section presents results for a large number of test problems. We chose all real, SPD matrices from the UFL Collection that are of order at least 1000 and are not diagonal. (Three matrices from the af_shell set were removed due to their similarity to af_shell7, which was retained.) This gives a set of 171 matrices. From this set, we chose two subsets: those that can be solved by PCG with IC(0) preconditioning (using exact triangular solves) within 3000 iterations and those can can be solved in the same way using IC(1) preconditioning. This results in a test set of 73 matrices for IC(0) and a set of 86 matrices for IC(1).

Our goal is to find what fraction of these matrices can be solved with PCG when the sparse triangular solves are replaced by Jacobi and block Jacobi iterations. Based on the results in the previous subsection, we use the following simple rule: for a solve with a lower triangular incomplete factor (with a random right-hand side), the iterative (block) Jacobi method can be used to apply the preconditioner if it reduces the residual norm by two orders of magnitude (0.01) within 30 sweeps. Both scalar and block Jacobi are used. In the

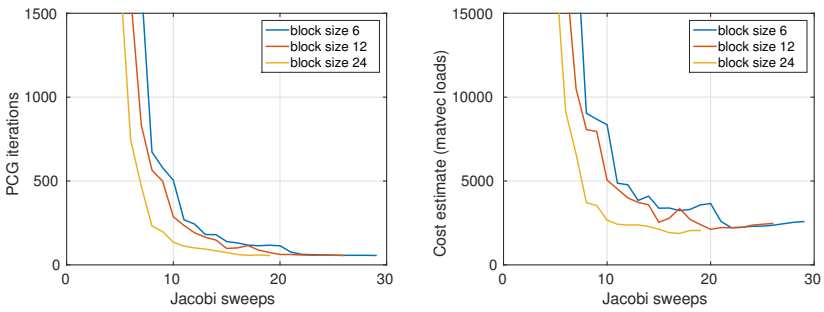

Figure 3.4: For bcsstk24, the effect of increasing the number of Jacobi sweeps on PCG iterations (left) and the cost estimate (right).

latter case, we use supervariable amalgamation with a maximum block size of 12. This block size was chosen because many PDE systems have 3, 4, or 6 degrees of freedom per node.

Table 3.1 summarizes our findings. In the case of IC(0), 74% of the problems that can be solved by IC(0)-PCG can also be solved when the triangular solves are replaced by Jacobi iterations. The fraction increases to 93% when block Jacobi is used. For IC(1), the fractions are somewhat worse: 60% and 81%, respectively. These results show that for a significant fraction of our test problems, the exact sparse triangular solves can be replaced by iterative solves, with a likely reduction in execution time on highly parallel hardware.

Table 3.1: Number of problems that can be solved when exact triangular solves in PCG are replaced by Jacobi and block Jacobi triangular solves.

	IC(0)	IC(1)
Total	73	86
Num. solved using Jacobi triangular solves	54 (74%)	52 (60%)
Num. solved using block Jacobi triangular solves	68 (93%)	70 (81%)

For the problems that successfully used Jacobi and block Jacobi, Figures 3.5 and 3.6 show how many Jacobi and block Jacobi sweeps, respectively, are needed in the PCG solves such that the number of PCG iterations using iterative triangular solves is the same as the number of PCG iterations using exact triangular solves. The figures are histograms showing the frequency of the number of sweeps required. For the scalar Jacobi case, a maximum of 40 Jacobi sweeps are needed, but the majority of problems require fewer than 10 sweeps. For block Jacobi, the number of sweeps required is generally lower, as expected. Note that these numbers are pessimistic; the optimal number of sweeps in terms of total computational cost may be much lower, as shown in Figure 3.1 (middle and right), since the best computational time is generally not achieved by trying to match the number of PCG iterations with exact triangular solves.

For all these results, we note that incomplete Cholesky may not be the best preconditioner among all preconditioners available; the results only show the potential efficacy of the iterative triangular solves if incomplete Cholesky were used as the preconditioner.

3.3 Results with graph partitioning-based blocking

In this section, we test block Jacobi triangular solves with matrices that have been partitioned using the graph partitioning techniques discussed in Section 2.3. We again use real, non-diagonal, SPD matrices of order at least 1000 from the UFL Collection. For these tests, however, we use IC(1) preconditioning with a limit of 800 iterations and we additionally use Manteuffel shifting [20] if the incomplete Cholesky factor does not exist. Thus the IC(1) factorization of $\bar{A} = Q^T SASQ + \alpha I$ is computed, where S is a diagonal

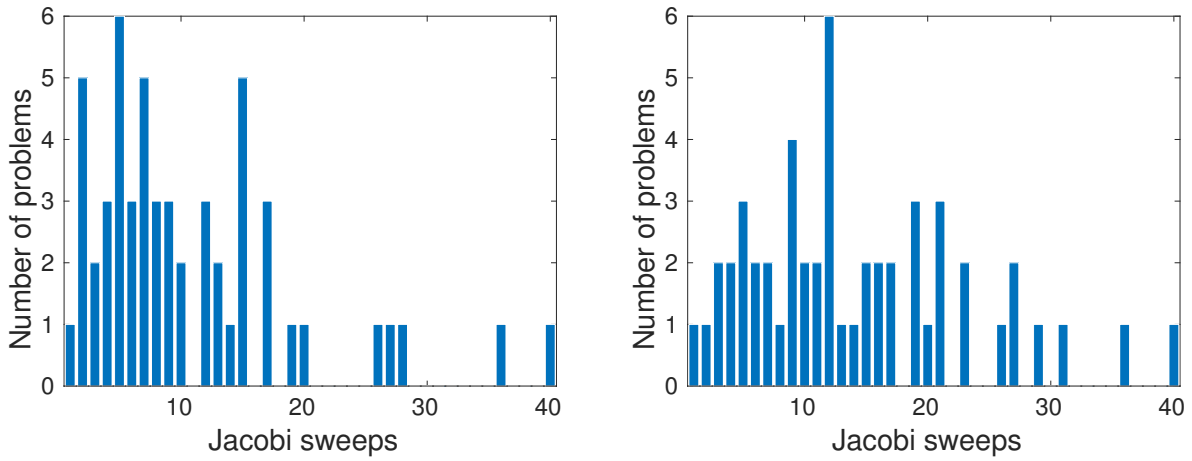

Figure 3.5: Number of sweeps for Jacobi triangular solves for IC(0) and IC(1).

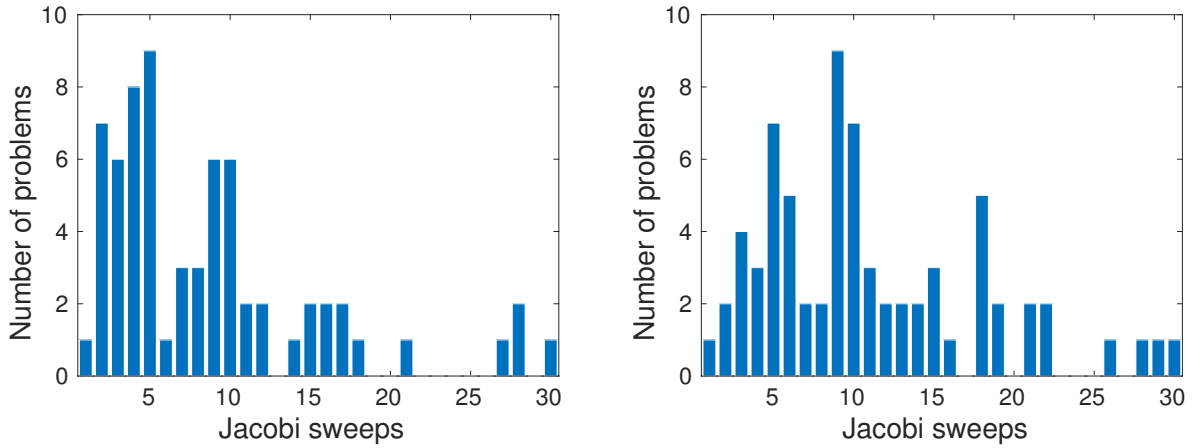

Figure 3.6: Number of sweeps for block Jacobi triangular solves for IC(0) and IC(1), using supervariable amalgamation and a maximum block size of 12.

scaling matrix, Q is a permutation matrix and $\alpha \geq 0$ is chosen as small as possible to avoid (almost) zero or negative pivots while maintaining stability [19]. The test set comprises 129 matrices. For each problem, instead of testing every possible number of Jacobi sweeps, we chose the number of Jacobi sweeps (called the *chosen number* of sweeps) to be the number of sweeps required to reduce the residual norm in an iterative solve with the lower triangular incomplete factor by two orders of magnitude (0.01).

For 65 of the 129 problems, the chosen number of sweeps is less than 50 for the scalar Jacobi method. For the other 64 "harder" problems, we investigate the effect of graph partitioning-based blocking. In Figure 3.7 we report the chosen number of sweeps when Approaches I and II of Section 2.3 are applied to these problems. Here the maximum block size is 10. For Approach I, 76% of the problems require 15 or fewer sweeps and only 4 need more than 30 sweeps. For Approach II, the corresponding statistics are 86% and 2 problems. This clearly illustrates the potential benefits of graph-based blocking.

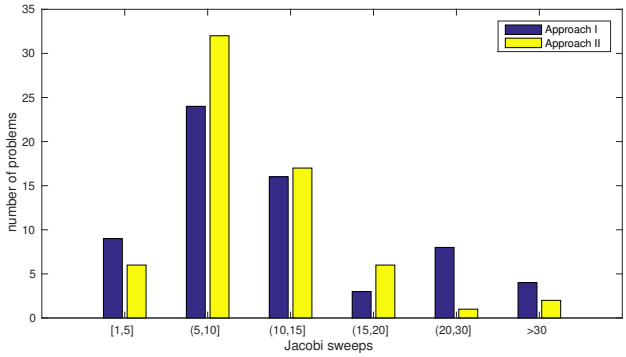

Figure 3.7: The chosen number of Jacobi sweeps for Approaches I and II for the 64 harder problems. For these problems, the chosen number of sweeps is more than 50 when no blocking is used.

In Table 3.2, we present results for PCG solves for a subset of the harder problems. For comparison, we include results using supervariable amalgamation, Approach I, and Approach II. Note that graph-based blocking corresponds to reordering the matrix A, and thus the resulting preconditioners are different for the different blocking strategies. In addition, different shifts α may be needed for the different graph-based blocking strategies and block sizes. For example, for bcsstk24, no shift is needed to compute the IC(1) factorization of A, but if Approach I or Approach II is used with a block size of 10 (respectively, 20), a shift of 0.0005 (respectively, 0.001) is used to prevent break down of the factorization. Since the factorizations are different, the cost estimates (3.1) in the tables in this subsection are in units of words.

The results in Table 3.2 show that Approach I and Approach II lead to orderings that often increase the iteration count compared to the locality preserving ordering used by supervariable amalgamation, but fewer Jacobi sweeps are needed for each PCG solve. The cost estimates for the different blockings are not significantly different.

Finally, in Table 3.3 we present some results for Approach II applied to the quotient graph (that is, the graph is first compressed using supervariables before Approach II is applied). Here we set the maximum block size to 30. We see that, in terms of the cost estimate, there is no consistent advantage in using supervariables and in our experiments we found the number of sweeps was generally smaller if supervariables were not used. For problems bcsstk10 and msc10848, a larger shift is needed if supervariables are not used and this leads to an increase in the iteration count.

Table 3.2: The effect of different blocking strategies on the number of Jacobi sweeps, the number of PCG iterations and the cost estimate. `bsize` denotes the block size. The figures in parentheses are the number of PCG iterations if exact solves are used with the corresponding blocking. A dash (–) denotes that PCG was not run.

Problem	`bsize`	Supervariable amalgamation			Approach I			Approach II		
		sweeps	iterations	cost estimate	sweeps	iterations	cost estimate	sweeps	iterations	cost estimate
cfd1	10	300	– (251)	–	30	239 (238)	3.38×10^{10}	11	275 (266)	1.57×10^{10}
	20	200	– (251)	–	50	284 (235)	6.68×10^{10}	8	266 (266)	1.12×10^{10}
	30	200	– (251)	–	50	262 (239)	6.26×10^{10}	7	282 (278)	1.04×10^{10}
ex13	10	61	16 (15)	1.03×10^{8}	35	29 (29)	1.81×10^{8}	10	27 (27)	4.67×10^{7}
	20	25	16 (15)	4.31×10^{7}	9	28 (29)	5.03×10^{7}	6	26 (26)	2.57×10^{7}
	30	21	15 (15)	3.41×10^{7}	8	28 (28)	4.45×10^{7}	6	21 (21)	2.05×10^{7}
shipsec8	10	40	120 (120)	5.19×10^{10}	10	202 (202)	3.84×10^{10}	11	153 (153)	3.17×10^{10}
	20	30	118 (120)	3.85×10^{10}	9	205 (205)	3.63×10^{10}	9	165 (165)	2.87×10^{10}
	30	30	119 (120)	3.88×10^{10}	9	197 (197)	3.54×10^{10}	8	131 (131)	2.04×10^{10}
bcsstk24	10	45	28 (26)	2.84×10^{8}	13	55 (55)	2.63×10^{8}	20	49 (49)	3.47×10^{8}
	20	41	27 (26)	2.50×10^{8}	9	93 (93)	3.22×10^{8}	9	61 (64)	2.04×10^{8}
	30	32	27 (26)	1.96×10^{8}	10	70 (70)	2.70×10^{8}	9	64 (63)	2.14×10^{8}

Table 3.3: Results with and without the exploitation of supervariables in Approach II. The block size is 30. α denotes the global shift used in the construction of the incomplete factorization and *supmax* is the size of the largest supervariable.

Problem	*supmax*	Not using supervariables				Using supervariables			
		α	sweeps	iterations	cost estimate	α	sweeps	iterations	cost estimate
bcsstk10	3	3.2×10^{-2}	5	47	1.03×10^{7}	2.5×10^{-4}	6	21	4.46×10^{6}
gyro_m	3	0.0	6	41	3.21×10^{8}	0.0	9	46	5.14×10^{8}
nasa2146	4	0.0	4	9	1.37×10^{7}	0.0	8	15	1.45×10^{7}
s1rmq4m1	6	0.0	7	81	4.04×10^{8}	0.0	9	75	3.13×10^{7}
msc10848	21	1.0×10^{-3}	8	35	1.28×10^{9}	1.56×10^{-5}	25	31	2.46×10^{9}

4 Concluding remarks

The main goal of this paper was to understand the applicability of Jacobi and block Jacobi iterations for solving the sparse triangular systems arising from incomplete Cholesky preconditioning. For a diverse set of SPD test problems using IC(0) and IC(1) PCG, Jacobi iterations were shown to be effective for a large majority of the problems; by using blocks and block Jacobi, robustness was enhanced. Although it is outside the scope of this study to perform and measure timings for parallel computations, previous studies [3, 7] suggest that Jacobi iterations can be much faster than level-scheduled triangular solves on highly parallel computers.

This paper did not address how to choose the optimum number of Jacobi sweeps to use in the PCG solver. A fixed number of sweeps is desirable, as the preconditioner is then a fixed operator and a "flexible" solver is not needed. However, in practice, the number of sweeps could be adjusted dynamically and the iterations restarted based on the convergence of PCG. If a flexible solver is used, e.g, FGMRES [23] in the nonsymmetric case, then a different number of Jacobi sweeps based on the residual norm reduction for each approximate triangular solve could be used.

References

[1] F. L. Alvarado and R. Schreiber, *Optimal parallel solution of sparse triangular systems*, SIAM J. on Scientific Computing, 14 (1993), pp. 446–460.

[2] E. C. Anderson and Y. Saad, *Solving sparse triangular systems on parallel computers*, International J. of High Speed Computing, 1 (1989), pp. 73–96.

[3] H. Anzt, E. Chow, and J. Dongarra, *Iterative sparse triangular solves for preconditioning*, Lecture Notes in Computer Science, 9233 (2015), pp. 650–661.

[4] M. Benzi, W. D. Joubert, and G. Mateescu, *Numerical experiments with parallel orderings for ILU preconditioners*, Electronic Transactions on Numerical Analysis, 8 (1999), pp. 88–114.

[5] M. Benzi and M. Tůma, *A comparative study of sparse approximate inverse preconditioners*, Applied Numerical Mathematics, 30 (1999), pp. 305–340.

[6] J. Bräckle and T. K. Huckle, *Incomplete sparse approximations of matrices, inverses of matrices, and their factorizations*, (2015). Submitted manuscript.

[7] E. Chow and A. Patel, *Fine-grained parallel incomplete LU factorization*, SIAM J. on Scientific Computing, 37 (2015), pp. C169–C193.

[8] E. H. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings 24^{th} National Conference of the ACM, ACM Press, 1969, pp. 157–172.

[9] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Transactions on Mathematical Software, 38 (2011). Article 1, 25 pages.

[10] S. Doi, *On parallelism and convergence of incomplete LU factorizations*, Applied Numerical Mathematics, 7 (1991), pp. 417–436.

[11] S. Doi and T. Washio, *Ordering strategies and related techniques to overcome the trade-off between parallelism and convergence in incomplete factorizations*, Parallel Computing, 25 (1999), pp. 1995–2014.

[12] I. S. Duff and K. Kaya, *Preconditioners based on strong subgraphs*, Electronic Transactions on Numerical Analysis, 40 (2013), pp. 225–248.

[13] I. S. Duff and G. A. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT Numerical Mathematics, 29 (1989), pp. 635–657.

[14] H. C. Elman and E. Agrón, *Ordering techniques for the preconditioned conjugate-gradient method on parallel computers*, Computer Physics Communications, 53 (1989), pp. 253–269.

[15] D. Fritzsche, A. Frommer, and D. B. Szyld, *Extensions of certain graph-based algorithms for preconditioning*, SIAM J. on Scientific Computing, 29 (2007), pp. 2144–2161.

[16] S. W. Hammond and R. Schreiber, *Efficient ICCG on a shared memory multiprocessor*, International J. High Speed Computing, 4 (1992), pp. 1–21.

[17] M. T. Jones and P. E. Plassmann, *Scalable iterative solution of sparse linear-systems*, Parallel Computing, 20 (1994), pp. 753–773.

[18] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. on Scientific Computing, 20 (1998), pp. 359–392.

[19] C.-J. Lin and J. J. Moré, *Incomplete Cholesky factorizations with limited memory*, SIAM J. on Scientific Computing, 21 (1999), pp. 24–45.

[20] T. A. Manteuffel, *An incomplete factorization technique for positive definite linear systems*, Mathematics of Computation, 34 (1980), pp. 473–497.

[21] E. L. Poole and J. M. Ortega, *Multicolor ICCG methods for vector computers*, SIAM J. on Numerical Analysis, 24 (1987), pp. 1394–1417.

[22] A. Pothen and F. L. Alvarado, *A fast reordering algorithm for parallel sparse triangular solution*, SIAM J. on Scientific Computing, 13 (1992), pp. 645–653.

[23] Y. Saad, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. on Scientific and Statistical Computing, 14 (1993), pp. 461–469.

[24] J. H. Saltz, *Aggregation methods for solving sparse triangular systems on multiprocessors*, SIAM J. on Scientific and Statistical Computing, 11 (1990), pp. 123–144.

[25] S. W. Sloan, *An algorithm for profile and wavefront reduction of sparse matrices*, International J. of Numerical Methods in Engineering, 23 (1986), pp. 239–251.

[26] H. A. Van der Vorst, *A vectorizable variant of some ICCG methods*, SIAM J. on Scientific Computing, 3 (1982), pp. 350–356.

[27] A. C. N. van Duin, *Scalable parallel preconditioning with the sparse approximate inverse of triangular matrices*, SIAM J. on Matrix Analysis and Applications, 20 (1999), pp. 987–1006.