A PRIORI SPARSITY PATTERNS FOR PARALLEL SPARSE APPROXIMATE INVERSE PRECONDITIONERS*

EDMOND CHOW†

Abstract. Parallel algorithms for computing sparse approximations to the inverse of a sparse matrix either use a prescribed sparsity pattern for the approximate inverse, or attempt to generate a good pattern as part of the algorithm. This paper demonstrates that for PDE problems, the patterns of powers of sparsified matrices (PSM's) can be used *a priori* as effective approximate inverse patterns, and that the additional effort of *adaptive* sparsity pattern calculations may not be required. PSM patterns are related to various other approximate inverse sparsity patterns through matrix graph theory and heuristics about the PDE's Green's function. A parallel implementation shows that PSM-patterned approximate inverses are significantly faster to construct than approximate inverses constructed adaptively, while often giving preconditioners of comparable quality.

Key words. preconditioned iterative methods, sparse approximate inverses, graph theory, parallel computing

AMS subject classifications. 65F10, 65F35, 65F50, 65Y05

1. Introduction.

A *sparse approximate inverse* approximates the inverse of a (usually sparse) matrix A by a sparse matrix M. This can be accomplished, for example in the *least-squares* method, by minimizing the matrix residual norm[1]

$$\|I - AM\|_F^2 \tag{1.1}$$

with the constraint that M is sparse. In general, the degrees of freedom of this problem are the nonzero values in M as well as their locations. A minimization that considers all these variables simultaneously, however, is very complex, and thus a simple approach is to prescribe the set of nonzeros, or the sparsity pattern $\mathcal{S}$, of M before performing the minimization. The objective function (1.1) can then be decoupled as the sum of squares of the 2-norms of the n individual columns, i.e.,

$$\|e_j - Am_j\|_2^2, \quad j = 1, 2, \ldots, n \tag{1.2}$$

in which e_j and m_j are the jth columns of the identity matrix and of the matrix M, respectively. Each least-squares matrix is small, having a number of columns equal to the number of nonzeros in its corresponding m_j. If A is nonsingular, then the least-squares matrices have full rank.

Thus the approximate inverse can be constructed by solving n least-squares problems in parallel. However, sparse approximate inverses are attractive for parallel preconditioning primarily because the preconditioning operation is a sparse matrix by vector product. The cost of constructing the approximate inverse for a large matrix is usually so high, especially with the adaptive pattern selection strategies described below, that they are only competitive if they are constructed in parallel.

For diagonally dominant A, the entries in A^{-1} decay rapidly away from the diagonal [17], and a banded pattern for M will produce a good approximate inverse.

* This work was performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract W-7405-ENG-48.

† Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, L-560, Box 808, Livermore, CA 94551 (`echow@llnl.gov`).

[1] The form for the right approximate inverse is used here, which is notationally slightly clearer. If the matrix is distributed over parallel processors by rows, a left approximate inverse can be computed row-wise.

In a general setting, without application-specific information, it is not clear how best to choose a sparsity pattern for M. Algorithms have been developed that first compute an approximate inverse with an initial pattern $\mathcal{S}$; then $\mathcal{S}$ is updated and a new minimization problem is solved either exactly or inexactly. This process is repeated until a threshold on the residual norm has been satisfied, or a maximum number of nonzeros has been reached [12, 15, 22]. We refer to these as *adaptive* procedures.

One such procedure is to use an iterative method, such as minimal residual, starting with a sparse initial guess [12] to approximately minimize (1.2), i.e., find sparse approximate solutions to

$$Am_j = e_j, \quad j = 1, 2, \ldots, n. \tag{1.3}$$

Suppose a sparse initial guess for m_j is used. The first few iterates will be sparse. To maintain sparsity, a strategy to drop small elements is usually used, either in the search direction or the iterates. No prescribed pattern $\mathcal{S}$ is necessary since the sparsity pattern emerges automatically. For this method to be efficient, *sparse-sparse* operations must be used: the product of a sparse matrix by a sparse vector with p nonzeros only involves p columns of the sparse matrix.

Another adaptive procedure, called SPAI [15, 22], uses a numerical test to determine which nonzero locations should be added to the current sparsity pattern. For the jth column, the numerical test for adding a nonzero in location k has the form

$$\frac{(r^T A e_k)^2}{\|Ae_k\|_2^2} > \textit{tolerance} \tag{1.4}$$

where $r = e_j - Am_j$ is the residual for a given sparsity pattern of column j, and m_j is the current approximation. The test (1.4) is a lower bound on the improvement in the square of the residual norm when the pattern of m_j is augmented. Entry k is added if the tolerance is satisfied, or if the left-hand side of (1.4) is large compared to its values for other k. The cost of performing this test is a sparse dot product between r and column k of A for each location to test. Interprocessor communication is needed to test a k corresponding to a column not on a local processor.

These adaptive algorithms utilize the additional degrees of freedom in minimizing (1.1) afforded by the locations of the nonzeros in M and have allowed much more general problems to be solved than before. Adaptive methods, however, tend to be very expensive. Thus, this paper focuses on the problem of selecting $\mathcal{S}$ in a preprocessing step so that a sparse approximate inverse can be computed immediately by minimizing (1.1). Section 2 first examines the sparsity patterns that are produced by both non-adaptive and adaptive schemes. Section 3 tests the idea of using the patterns of powers of sparsified matrices (PSM's) as *a priori* sparsity patterns for sparse approximate inverses. Numerical tests are presented in Section 4, with comparisons to both sequential and parallel versions of some current methods. Finally, Section 5 draws some conclusions.

2. Graph interpretations of approximate inverse sparsity patterns.

2.1. Use of the pattern of A and variants. The *structure* of a sparse matrix A of order n is the directed graph $G(A)$, whose vertices are the integers $1, \ldots, n$, and whose edges $(i \rightarrow j)$ correspond to nonzero off-diagonal entries in A. (This notation usually implies matrices with all nonzero diagonal entries.) A *subset* of $G(A)$ is a directed graph with the same vertices, but with a subset of the edges in $G(A)$. The graph $G(A)$ is a representation of the sparsity pattern of A, and when it is clear from the context, we will not distinguish between them.

The *structure* of a vector x of order n is the subset of $\{1, \ldots, n\}$ that corresponds to the nonzero entries in x. When there is an associated matrix of order n, we often refer to the structure of x as a subset of the vertices of the associated matrix. Notice then, that the structure of column j of a matrix A is the set of vertices in $G(A)$ that have edges pointing to vertex j plus vertex j itself. The structure of row j is vertex j plus the set of vertices pointed to by vertex j.

The inverse of a matrix shows how each unknown in a linear system depends on the other unknowns. The structure of the matrix A shows only the immediate dependencies. This suggests that in the structure of A^{-1} there is an edge $(i \to j)$ whenever there is a directed path from vertex i to vertex j in $G(A)$ [21] (if A is nonsingular, and ignoring coincidental cancellation). This structure is called the *transitive closure* of $G(A)$, and is denoted $G^*(A)$. For an irreducible matrix, this result says that the inverse is a full matrix, but it does suggest the possibility of *truncating* the transitive closure process to approximate the inverse by a sparse matrix.

A heuristic that is often employed is that vertices closer to vertex j along directed paths are more important, and should be retained in an approximate inverse sparsity pattern. This idea is supported by the decay in the elements observed by Tang [30] in the discrete Green's function for many problems.

These sparsity patterns were first used by Benson and Frederickson [4] in the symmetric case, who also defined matrices with these patterns to be *q-local matrices*. Given a graph $G(A)$ of a structurally symmetric matrix A with a full diagonal, the structure of the jth column of a q-local matrix consists of vertex j and its qth level nearest-neighbors in $G(A)$. A 0-local matrix is a diagonal matrix, while a 1-local matrix has the same sparsity pattern as A.

The sparsity pattern of A is the most common *a priori* pattern used to approximate A^{-1}. It gives good results for many problems, but can usually be improved, or fails for many other problems. One improvement is to use higher levels of q. Unfortunately, the storage for these preconditioners grows very quickly when q is increased, and even $q = 2$ is impractical in many cases [20].

Huckle [24] proposed similar patterns which may be more effective when A is nonsymmetric. These include the patterns corresponding to the graphs $G((A^TA)^kA^T)$ and $G((I + |A| + |A^T|)^kA^T)$ for small integers $k \geq 0$. The density of the former, in particular, grows very quickly with increasing k. Primarily, these patterns are useful as envelope patterns from which the adaptive SPAI algorithm can select its pattern. This gives an upper bound on the interprocessor communication required by a parallel implementation [23].

Cosgrove and Díaz [14] proposed augmenting the pattern of A without going to the full 2-local matrix. They suggested adding nonzeros to m_j in a way that minimizes the number of new rows introduced into the jth least-squares matrix (in expression (1.2)). The augmented structure is determined only from the structure of A. Kolotilina and Yeremin [28] proposed similar heuristics for augmenting the sparsity pattern for *factorized* sparse approximate inverses.

Sparsification. Instead of augmenting the pattern of A, it is also possible to diminish the pattern of A when A is relatively full. This can be accomplished by *sparsification* (dropping small elements in the matrix A) and using the resulting pattern. This was introduced by Kolotilina [26] for computing sparse approximate inverses for *dense* matrices (see also [10, 32]), and Kaporin [25] for sparse matrices and factorized approximate inverses.

Sparsification can be combined with the use of higher level neighbors. Tang [30] showed that sparsifying a matrix prior to applying the adaptive SPAI algorithm is effective for anisotropic problems. The observation is that the storage and therefore operation count required for preconditioners produced this way are much smaller. This technique can generate patterns that are those of powers of sparsified matrices.

The idea of explicitly combining sparsification with the use of higher level neighbors was used by Alléon et al. [1], who attributes the technique to Cosnuau [16]. For approximating the inverse of dense matrices in electromagnetics, however, their tests showed that higher levels were not warranted. Tang and Wan [31] also used a sparsification before applying a q-local matrix pattern, for $q > 1$, for approximate inverses used as multigrid smoothers. They showed that the sparsification does not cause a deterioration in convergence rate for their problems. Both the work by Alléon et al. and Tang and Wan represent the first uses of PSM patterns.

Instead of applying the sparsification to A, it is also appropriate in some cases to apply the sparsification to the sparse approximate inverse after it has been computed [27, 31]. This is useful to reduce the cost of using the approximate inverse when it is relatively full.

2.2. Insights from adaptive schemes. Adaptive schemes can generate patterns that are very different from the pattern of A, for example, the generated patterns can be much sparser than A. Nevertheless, the patterns produced by adaptive schemes can be interpreted using the graph of A.

Consider first the approximate minimization method described in Section 1. The following algorithm finds a sparse approximate solution to $Am = e_j$ via sparse-sparse minimal residual iterations. A dropping strategy for elements in the search direction r is encapsulated by the function "drop" in step 4, which may depend on the current pattern of m.

ALGORITHM 2.1. *Sparse approximate solution to* $Am = e_j$

1. $m :=$ *sparse initial guess*
2. $r := e_j - Am$
3. *Loop until* $\|r\| < \mathrm{tol}$ *or reached max. iterations*
4. $\quad d := \mathrm{drop}(r)$
5. $\quad q := Ad$
6. $\quad \alpha := \frac{(r,q)}{(q,q)}$
7. $\quad m := m + \alpha d$
8. $\quad r := r - \alpha q$
9. *EndLoop*

The elements in r not already in m are candidates for new elements in m. The vector r is generated essentially by the product Am and thus the structure of r is the set of vertices that have edges pointing to vertices in the structure of m. If the initial guess for m consists of a single nonzero element at location j, then the structure of m grows outward from vertex j in $G(A)$ with each iteration of the above algorithm.

If the search direction in the iterative method is $A^T r$ instead of r, then the kth entry in the search direction is $r^T A e_k$. A dropping strategy based on the size of the these entries is similar to one based on the test (1.4) which attempts to minimize the updated residual norm. In this method, the candidates for new elements are the first and second level neighbors of the vertices in m (for nonsymmetric A, the directions of the edges are important).

This is exactly the graph interpretation for the SPAI algorithm (see Huckle [24]). When computing column j of M, vertices far from vertex j will not enter into the pattern, at least initially. Algebraically, this means that the nonzero locations of r and Ae_k do not intersect, and the value of the test is zero. An efficient implementation of SPAI uses these graph ideas to narrow down the indices k that need to be checked.

An early parallel implementation of SPAI [18] tested only the first level neighbors of a vertex, rather than both the first and second levels. This is a good approximation in many cases. This implementation also assumed A is structurally symmetric, so that one-sided interprocessor communication is not necessary. A more recent parallel implementation of SPAI [2, 3] implements the algorithm exactly. This code implements one-sided communication with the Message Passing Interface (MPI), and uses dynamic load balancing in case some processors finish computing their rows earlier than others.

3. Patterns of powers of sparsified matrices.

3.1. Graph interpretation. In Section 2, we observed that prescribed sparsity patterns or patterns generated by the adaptive methods are generally *subsets* of the pattern of low powers of A (given that A has a full diagonal) and typically increase in accuracy with higher powers. Clearly all these methods are related to the Neumann series or characteristic polynomial for A [24].

The structure of column j of the approximate inverse of A is a subset of the vertices in the level sets (with directed edges) about vertex j in $G(A)$. Good vertices to choose are those in the level sets in the neighborhood of vertex j, but the algorithms differ in how these vertices are selected.

For convection-dominated and anisotropic problems, upstream vertices or vertices in the preferred directions will have a greater influence than others on column j of the inverse. Figure 3.1 shows the discrete Green's function for a point on a PDE with convection. The nonsymmetry of the function shows that upstream nonzeros in a row or column of the exact inverse are greater in magnitude than others. Without additional physical information such as the direction of flow, however, it is often possible to use *sparsification* to identify the preferred or upstream directions.

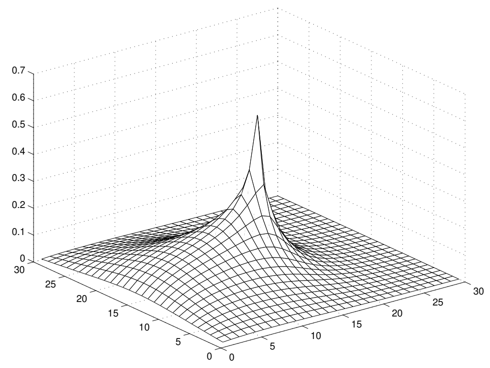
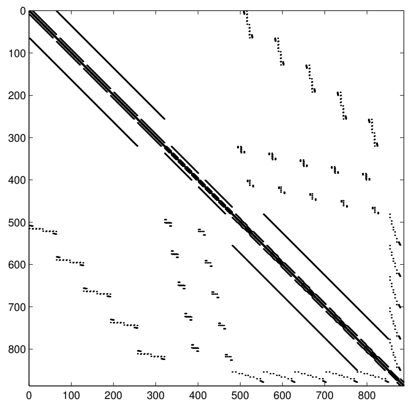
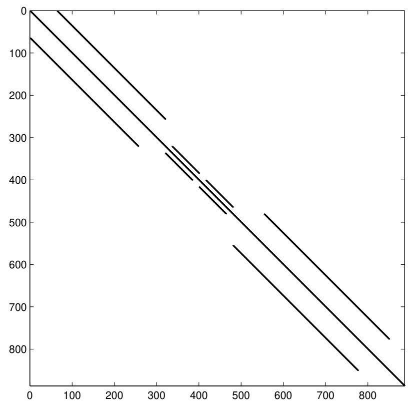


FIG. 3.1. *Green's function for a point on a PDE with convection.*

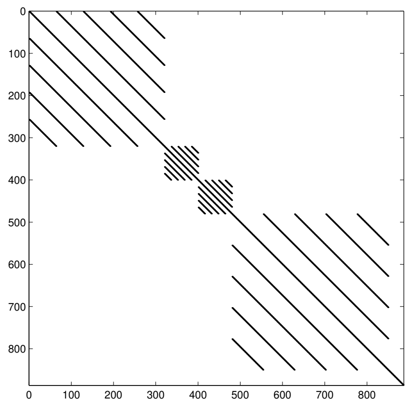
We examined the sparsity patterns produced by the adaptive algorithms and tried to determine if they could have been generated by simpler graph algorithms. For some simple examples, it turned out that the structures produced are exactly or very close to the transitive closures of a *subset* of $G(A)$, i.e., of the structure of a sparsified matrix. In Figure 3.2 we show the structures of several matrices: (a) ORSIRR2 from the Harwell-Boeing collection, (b) A_0, a sparsification of the original matrix, (c) the transitive closure $G^*(A_0)$, and (d) the structure produced by the SPAI algorithm. This latter figure was selected from [2] which shows it as an example of an effective sparse approximate inverse pattern for this problem. (There are, however, some bothersome features of this example: the approximate inverse is four independent diagonal blocks.) Note that we can approximate the adaptively generated pattern (d) very well by the pattern (c) generated using the transitive closure.



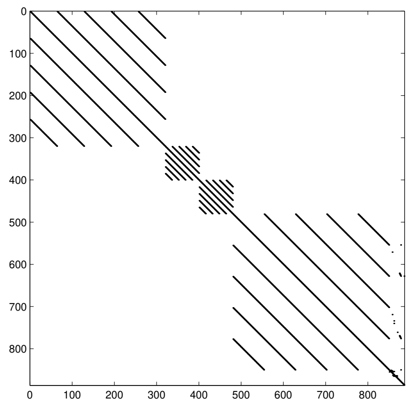
(a) ORSIRR2



(b) Sparsified ORSIRR2



(c) Transitive closure of (b)



(d) Pattern from SPAI

FIG. 3.2. *The adaptively generated pattern (d) can be approximated by the transitive closure (c).*

3.2. How to sparsify. The simplest method to sparsify a matrix is to retain only those entries in a matrix greater than a global threshold, *thresh*. In the example of Figure 3.2, *thresh* $= 10^3$ was used. It was important, however, to make sure that the diagonal elements were retained, otherwise a structurally singular matrix would have resulted in this case. In general, the diagonal should always be retained.

One strategy for choosing a threshold is to choose one that retains, for example, one-third of the original nonzeros in a matrix. Fewer nonzeros should be retained if powers of this sparsified matrix have numbers of nonzeros that grow too quickly. This is how thresholds were chosen for the small problems tested in Section 4. The number of levels used may be increased until a preconditioner reaches a target number of nonzeros. The best choices for these parameters will be problem-dependent. For special problems, this strategy may not be effective, for example, when a matrix contains only a few unique values.

When a matrix is to be sparsified using a global threshold, how the matrix is scaled becomes important. It is often the case that a matrix contains many different types of equations and variables that are not scaled the same way. For example, consider the matrix

$$\begin{pmatrix} Z^2 & Z & 0 \\ Z & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

which has its first row and column scaled by a large number, Z. If a threshold Z is chosen, and if the diagonal of the matrix is retained, the third row of the sparsified matrix has become independent of the other rows. We thus apply the thresholding to a matrix that has been symmetrically scaled so that it has all ones on its diagonal, e.g., for the above matrix, the scaled matrix is

$$\begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

A threshold less than or equal to 1 will guarantee that the diagonal is retained. The scaling also makes it easier to choose a threshold. This method of scaling is not foolproof, but does avoid some simple problems.

In the graph of the sparsified matrix A_0, each vertex should have some connections to other vertices. This can be accomplished by sparsifying the matrix A such that one retains at least a fixed number of edges to or from each vertex, for example, the ones corresponding to the largest matrix values. Given a parameter *lfil*, this can be implemented for column j by selecting the diagonal (jth) element plus the *lfil*-1 largest off-diagonal elements in column j of the original matrix. This guarantees that there are *lfil*-1 vertices with edges into vertex j. Applied row-wise, this guarantees *lfil*-1 vertices with edges emanating from vertex j. Again, we choose explicitly to keep the diagonal of the matrix; thus each column (or row) has at least *lfil* nonzeros. Choosing *lfil* may be simpler and more meaningful than choosing a threshold on the matrix values. Different values of *lfil* may be used for different vertices, depending on the vertex's initial degree (number of incident edges).

Let $\mathcal{S}_0$ denote the structure of a matrix A_0 that has been sparsified from A. Let $\mathcal{S}_i, i \geq 1$ denote the structure (with the same set of vertices) that has an edge whenever there is a path of distance i or less in $\mathcal{S}_0$. The structure $\mathcal{S}_{i-1}$ is a subset of $\mathcal{S}_i$. In matrix form, $\mathcal{S}_i = G(A_0^{i+1})$, ignoring coincidental cancellation. These are called *level set expansions* of a sparsified matrix or patterns of powers of a sparsified matrix (PSM patterns). Heuristic 3 tested by Alléon et al. [1] is equivalent to $\mathcal{S}_i$ using a variable *lfil* at each level to perform sparsification.

We mention that it is also possible to perform sparsification on $\mathcal{S}_i$ after every level set expansion. We denote this variant by $\mathcal{S}_i^*$. For this variant, values need to be computed, and we propose the following, which stresses the larger elements in A. If "drop" denotes a sparsification process, then we can define $A_i = \text{drop}(A_{i-1}A_0), i \geq 1$, and $\mathcal{S}_i^* = G(A_i)$. We define $\mathcal{S}_0^* = \mathcal{S}_0$ and note that $\mathcal{S}_{i-1}^*$ is not generally a subset of $\mathcal{S}_i^*$. More complicated strategies are possible; the thresholds can be different for each level i. Note that determining $\mathcal{S}_i^*$ is much more difficult than determining $\mathcal{S}_i$ since values need to be computed.

3.3. Factorized forms of the approximate inverse. Sparse approximate inverses for the Cholesky or LU factors of A are often used. The analogue of the least-squares method (minimization of (1.1)) here is the factorized sparse approximate inverse (FSAI) technique of Kolotilina and Yeremin [28], implemented in parallel by Field [19]. If the normal equations method is used to solve the least-squares systems, the Cholesky or LU factors are not required to compute the approximate inverse. This means, however, that the adaptive pattern selection schemes cannot be used, since the matrix whose inverse is being approximated is not available. *A priori* sparsity patterns must be used instead.

Given $A = LU$, a factorized sparse approximate inverse approximates U^{-1} and L^{-1} by sparse matrices G and H, respectively, so that

$$A^{-1} \approx GH.$$

The patterns for G and H should be chosen such that the pattern of GH is close in some sense to good patterns for approximating A^{-1}. Supposing that $\mathcal{S}$ is a good pattern for A^{-1}, then the upper and lower triangular parts of $\mathcal{S}$ are good patterns for G and H, since the pattern of GH includes the pattern $\mathcal{S}$. These patterns will be tested in Section 4.

It may also be possible to use the patterns of the powers of the exact or approximate L and U if they are known. These L and U factors are not discretizations of PDE's, but their inverses are often banded with elements decaying rapidly away from the main diagonal. This technique may be appropriate if approximate L and U are available, for example from a very sparse incomplete LU factorization.

As opposed to the inverses of irreducible matrices, the inverses of Cholesky or LU factors are often sparse. An ordering should thus be applied to A that gives factors whose inverses can be well approximated by sparse matrices. Experimentally, fewer nonzeros in the exact inverse factors translates into lower construction cost and better performance for factorized approximate inverses computed by an incomplete biconjugation process [7, 8]. The transitive closure can be used to compute the number of nonzeros in the exact inverse of a Cholesky factor, based on the height of all the nodes in the elimination tree. This has lead to reordering strategies that approximately minimize the height of the elimination tree and thus the number of nonzeros in the inverse factors, and allows some prediction of how well these approximate inverses might perform on a given problem [7, 8].

3.4. Approximate inverse of a Schur complement. To determine a good pattern for a Schur complement matrix, we notice that

$$A^{-1} = \begin{pmatrix} B & F \\ E & C \end{pmatrix}^{-1} = \begin{pmatrix} B^{-1} + B^{-1}FS^{-1}EB^{-1} & -B^{-1}FS^{-1} \\ -S^{-1}EB^{-1} & S^{-1} \end{pmatrix}$$

where $S = C - EB^{-1}F$ is the Schur complement. Thus a good sparsity pattern for S^{-1} can be determined from a good sparsity pattern for A^{-1}; it is simply the (2,2) block of a good sparsity pattern for A^{-1}. B should be of small order compared to the global matrix or else the method will be overly costly. In a code, it may be possible to compute the approximate inverse of A and extract the approximation to S^{-1}, or compute a partial approximate inverse, i.e., those rows or columns of the approximate inverse that correspond to S^{-1} [11]. Again, S should be of almost the same order as A.

3.5. Parallel computation. Computation of $\mathcal{S}_i$ is equivalent to structural sparse matrix-matrix products of sparsified matrices. The computation can also be viewed as n level set expansions, one for each row or column, which can be performed in parallel. For vertices that are near other vertices on a different processor, some communication will be necessary. Communication can be reduced by partitioning the graph of the sparsified matrix among the processors such that the number of edge-cuts is reduced.

Unfortunately, in general, one-sided communication is required to compute sparse approximate inverses. Processors need to request rows from other processors, and a processor cannot predict which rows it will need to send. One-sided communication may be implemented in MPI by having each processor occasionally probe for messages from other processors. The latency between probes is a critical performance factor here. In a multithreaded environment, it is possible to dedicate some threads on a local processing node to servicing requests for rows (server threads), while the remaining threads compute each row and make requests for rows when necessary (worker threads).

Consider a matrix A and an approximate inverse M to be computed that are partitioned the same way by rows across several processors. Algorithm 3.1 describes one organization of the parallel computation. Each processor computes a level set expansion for all of its rows before before continuing on to the next level. At each level, the requests and replies to and from a processor are coalesced, allowing fewer and larger messages to be used. Like in [3], external rows of A are cached on a processor in case they are needed to compute other rows. There is no communication during the numerical phase when the values of M are being computed. This algorithm was implemented using occasional probing for one-sided communication.

ALGORITHM 3.1. *Parallel level set expansions for computing $\mathcal{S}_i$*

Communicate rows
Initialize the set of vertices $\mathcal{V}$ to empty
Sparsify all the rows on the local processor
Merge the structures of all the locally sparsified rows into $\mathcal{V}$
For level $= 0, \ldots, i-1$
    For nonlocal $k \in \mathcal{V}$, request and receive row k
    Sparsify received rows
    Merge structures of new sparsified rows into $\mathcal{V}$
EndFor
For nonlocal $k \in \mathcal{V}$, request and receive row k
Compute structure of each row
For each local row j
    Initialize $\mathcal{V}_j$ to a single entry in location j
    For level $= 0, \ldots, i$
        For new $k \in \mathcal{V}_j$, merge sparsified structure of row k into $\mathcal{V}_j$
    EndFor
EndFor
Compute values of each row

16. For each local row j, find $m_j = \arg\min \|e_j^T - m_j^T A\|_2$, where m_j has the pattern $\mathcal{V}_j$

We also implemented a second parallel code, which has the following features:

- multithreaded, to take advantage of multiple processors per shared-memory node on symmetric multiprocessor computers
- uses server and worker threads to more easily implement one-sided communication
- uses a simpler algorithm than Algorithm 3.1: computes each row and performs all the associated communications before continuing on to the next row; when multiple threads are used, this avoids worker threads needing to synchronize and coordinate which rows to request from other nodes; smaller messages are used, but communication is also spread over the entire execution time of the algorithm
- scalable with its use of memory, but is thus also slower than the first version which used direct-address tables (traded memory for faster computation)

Timings for this second code will be reported in the next section. Some limited timings for the first code will also be shown. We are also working on a factorized implementation for symmetric matrices, which will guarantee that the preconditioner is also symmetric. This implementation makes a simple change in step 16 of Algorithm 3.1, and does not require one-sided communication when the full matrix is stored.

4. Numerical tests.

4.1. Preconditioning quality. First we test the quality of sparsity patterns generated by powers of sparsified matrices on small problems from the Harwell-Boeing collection. In particular, we chose problems that were tested with SPAI [22] in order to make comparisons. We performed tests in exactly the same conditions: we solve the same linear systems using GMRES(20) to a relative residual tolerance of 10^{-8} with a zero initial guess. We report the number of GMRES steps needed for convergence, or indicate no convergence using the symbol †. Right preconditioning was used.

In Tables 4.1 to 4.5, we show test results for $\mathcal{S}_i$ for both unfactored (column 2) and factored (column 3) forms of the approximate inverse. We compare the results to the least-squares (LS) method using the pattern of the original matrix A, and FSAI, the least-squares method for the (nonsymmetric) factored form [28], again using the pattern of the original matrix A. For the unfactored form, we also display the result of the SPAI method reported in [22], using their choice of parameters. Adaptive methods for factored forms are also available [5, 6, 29] but were not tested here. Global thresholds (shown for each table) on a scaled matrix were used to perform these sparsifications. In the tables, we also show the number of nonzeros nnz in the unfactored preconditioners (the entry for LS/FSAI is the number of nonzeros in A).

The results show that preconditioners of almost the same quality as the adaptive SPAI can be achieved using the $\mathcal{S}_i$ patterns. In some cases, even better preconditioners can result, sometimes with even less storage (Table 4.1). The results also show that using the pattern of A does not generally give as good a preconditioner for these problems.

SHERMAN2 is a relatively hard problem for sparse approximate inverses. The result reported in [22] shows that SPAI could reduce the residual norm by 10^{-5} (the target was 10^{-8}) with a preconditioner with 26327 nonzeros in 7 steps. The results are similar with $\mathcal{S}_i$ patterns, but the full residual norm reduction can be achieved with an approximate inverse that is denser. Note that in this case, the sparsification

TABLE 4.1
Iteration counts for ORSIRR2, n = 886, thresh = 0.1.

Pattern	unfactored	factored	nnz
LS/FSAI	335	383	5970
$\mathcal{S}_0$	315	376	2352
$\mathcal{S}_1$	128	261	3530
$\mathcal{S}_2$	91	205	4388
$\mathcal{S}_3$	76	73	4896
$\mathcal{S}_4$	76	73	5036
SPAI	84		5318

TABLE 4.2
Iteration counts for SHERMAN1, n = 1000, thresh = 0.2.

Pattern	unfactored	factored	nnz
LS/FSAI	145	456	3750
$\mathcal{S}_0$	144	†	2004
$\mathcal{S}_1$	111	109	2954
$\mathcal{S}_2$	108	98	3872
$\mathcal{S}_3$	107	94	4690
$\mathcal{S}_4$	105	94	5392
SPAI	89		5025

threshold was applied to the original matrix rather than to the diagonally scaled matrix, although the matrix has values over 27 orders of magnitude; the diagonal scaling is not foolproof. Factorized approximate inverses were not effective for this problem with these patterns.

SAYLR4 is a relatively hard problem for GMRES. Grote and Huckle [22] report that SPAI could not solve the problem with GMRES, but could with BiCGSTAB. This is also true for $\mathcal{S}_i$ patterns; the results in Table 4.5 are with BiCGSTAB.

There are, of course, many problems that are difficult for PSM-patterned approximate inverses. These include NNC666 and GRE1107 from the Harwell-Boeing collection, and FIDAP problems from Navier-Stokes simulations. These problems, however, can be solved using adaptive methods [12]. These problems pose difficulties for PSM patterns because the Green's function heuristic is invalid; the problems either are not PDE problems, or have been modified (e.g., the FIDAP problems used a penalty formulation).

In Tables 4.6 and 4.7, we show test results for $\mathcal{S}_i^*$ for the unfactored form of the approximate inverse. $\mathcal{S}_0^*$ and the LS patterns were the same for these problems. Since $\mathcal{S}_i^*$ is not generally a superset of $\mathcal{S}_{i-1}^*$, there is no guarantee that $\mathcal{S}_i^*$ is a better pattern than $\mathcal{S}_{i-1}^*$ in terms of the norm of $R = I - AM$. To show this, we also display these matrix residual norms. The parameter *lfil* (shown for each table) was used to sparsify these matrices after each level set expansion. Again, the results show that the *a priori* methods can approach the quality of the adaptive methods very closely.

4.2. Parallel timing results. The results above show that the preconditioning quality for PDE problems is not significantly degraded by using the non-adaptive schemes based on powers of sparsified matrices. In this section we illustrate the main advantage of these preconditioners: their very low construction costs compared to the adaptive schemes.

In this and Section 4.3 we show results using the multithreaded version of our code, ParaSAILS (parallel sparse approximate inverse, least squares). Like the parallel version of SPAI [3] with which we make comparisons, ParaSAILS is implemented

TABLE 4.3
Iteration counts for SHERMAN2, n = 1080, thresh = 1.

Pattern	unfactored	factored	nnz
LS/FSAI	†	†	23094
$\mathcal{S}_0$	†	†	12354
$\mathcal{S}_1$	244	†	48502
$\mathcal{S}_2$	7	†	118254
SPAI	†		26327

TABLE 4.4
Iteration counts for PORES3, n = 532, thresh = 0.001.

Pattern	unfactored	factored	nnz
LS/FSAI	†	†	3474
$\mathcal{S}_0$	†	†	1999
$\mathcal{S}_1$	634	495	4700
$\mathcal{S}_2$	256	257	8779
$\mathcal{S}_3$	216	177	13630
$\mathcal{S}_4$	148	86	18820
SPAI	599		16745

as a preconditioner object in the ISIS++ solver library [13]. Both these codes generate a sparse approximate inverse partitioned across processors by rows; thus left preconditioning is used. In all the codes, the least-squares problems that arose were solved using LAPACK routines for QR decomposition. For problems with relatively full approximate inverses, solving these least-squares problems takes the majority of the computing time.

Tests were run on multiple nodes of an IBM RS/6000 SP supercomputer at the Lawrence Livermore National Laboratory. Each node contains four 332 MHz PowerPC 604e CPU's. Timings were performed using user-space mode, which is much more efficient than internet-protocol mode. However, nonthreaded codes can only use one processor per node in user-space mode. We tested SPAI with one processor per node, and ParaSAILS with up to four processors per node. The iterative solver and matrix-vector product codes were also nonthreaded, and used only one processor per node.

The first problem we tested is a finite element model of three concentric spherical shells with different material properties. The matrix has order $n = 16881$ and has $nnz = 1134441$ nonzeros. The SPAI algorithm using the default parameters (target residual norm $\|e_j - Am_j\|_2$ for each row < 0.4) produced a much sparser preconditioner, with 171996 nonzeros, and solved the problem using GMRES(50) to a tolerance of 10^{-6} in 324 steps. For comparison purposes, we chose parameters for ParaSAILS that gave a similar number of nonzeros in the preconditioner. In particular, $\mathcal{S}_3$ with an *lfil* parameter of 3 gave a preconditioner with 179550 nonzeros, and solved the problem in 331 steps. Figure 4.1 shows the two resulting sparsity patterns. Table 4.8 reports, for various numbers of nodes (npes), the wall-clock times for the preconditioner setup phase (Precon), the iterative solve phase (Solve), and the total time (Total). The time for constructing the preconditioner in each code includes the time for determining the sparsity pattern. Due to the relatively small size of this (and the next) problem, only one worker and one server thread was used per node (i.e., two processors per node) in the ParaSAILS runs; one processor was used in the SPAI runs.

For comparison, in Table 4.9 we show the results using the first (nonthreaded, occasional MPI probes for one-sided communication) version of the code. This code

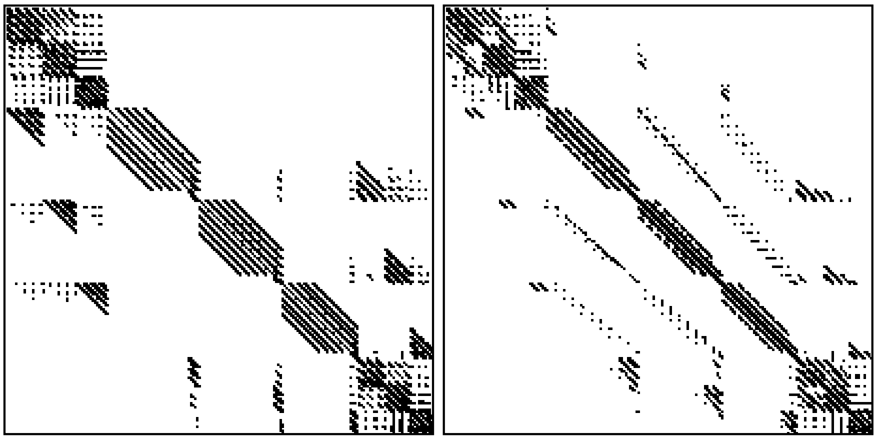
TABLE 4.5
Iteration counts for SAYLR4, n = 3564, thresh = 0.001.

Pattern	unfactored	factored	nnz
LS/FSAI	†	†	22316
$\mathcal{S}_0$	†	†	10168
$\mathcal{S}_1$	†	†	16854
$\mathcal{S}_3$	423	406	31748
$\mathcal{S}_5$	248	423	48432
$\mathcal{S}_7$	180	181	65766
$\mathcal{S}_9$	115	186	83046
$\mathcal{S}_{11}$	86	130	100176
SPAI	67		84800

TABLE 4.6
Iteration counts for SHERMAN3, n = 5005, lfil = 20.

	steps	nnz	$\|R\|_F$
LS	†	20033	17.3620
$\mathcal{S}_1^*$	†	54061	12.1730
$\mathcal{S}_2^*$	483	59615	10.6537
$\mathcal{S}_3^*$	420	60094	12.3150
$\mathcal{S}_4^*$	272	60395	13.6814
SPAI	264	48480	

is much faster because it uses direct-address arrays to quickly merge the sparsity patterns of rows; direct-address arrays have length the global size of the matrix and are not scalable.



(a) ParaSAILS (b) SPAI

FIG. 4.1. *Structure of the sparse approximate inverses for the concentric shells problem.*

We tested a second, larger problem which models the work-hardening of metal by squeezing it to make it pancake-like ($n = 49600, nnz = 3153942$). In this particular example, the pattern of a sparsified matrix ($\mathcal{S}_0$) with an *lfil* parameter of 3 lead to a good preconditioner. ParaSAILS produced a preconditioner with 141848 nonzeros and solved the problem in 142 steps; SPAI produced a preconditioner with 120192 nonzeros

TABLE 4.7
Iteration counts for SHERMAN4, $n = 1104$, lfil = 15.

	steps	nnz	$\|R\|_F$
LS	199	3786	6.2503
$\mathcal{S}_1^*$	94	8432	4.5807
$\mathcal{S}_2^*$	78	8745	4.4539
$\mathcal{S}_3^*$	110	8752	4.8623
$\mathcal{S}_4^*$	139	8778	5.9313
SPAI	86	9276	

TABLE 4.8
Timings for concentric shells problem.

	ParaSAILS			SPAI		
npes	Precon	Solve	Total	Precon	Solve	Total
1	28.14	76.51	104.65	215.42	74.72	290.14
2	16.09	40.87	56.96	166.06	38.64	204.70
4	11.90	25.48	37.38	88.65	22.84	111.49
8	8.61	16.10	24.71	65.47	14.21	79.68
16	4.98	10.74	15.72	46.32	9.34	55.66
32	3.84	7.77	11.61	25.85	6.77	32.62
64	3.30	6.04	9.34	21.40	5.43	26.83

and solved the problem in 139 steps. Results for varying numbers of processing nodes are shown in Table 4.10. The results show that the non-adaptive ParaSAILS algorithm implemented here is many times faster than the adaptive SPAI algorithm.

4.3. Implementation scalability. In this section, we experimentally investigate the implementation scalability of constructing sparse approximate inverses with ParaSAILS. Let $T(n,p)$ be the time to construct an approximate inverse of order n on a parallel computer using p processors. We define the *scaled efficiency* to be

$$E(n,p) \equiv T(n,1)/T(pn,p).$$

If $E(n,p) = 1$ for all n and p, then the implementation is *perfectly scalable*, i.e., one could double the size of the problem and the number of processors without increasing the execution time. However, as long as $E(n,p)$ is bounded away from zero for a fixed n as p is increased, we say that the implementation is *scalable*.

We consider the 3-D constant coefficient PDE

$$\begin{aligned} au_{xx} + bu_{yy} + cu_{zz} = 1 \quad &\text{in} \quad \Omega = (0,1)^3 \\ u = 0 \quad &\text{on} \quad \partial\Omega \end{aligned}$$

discretized using standard finite differences on a uniform mesh, with the anisotropic parameters $a = 0.1$, $b = 1$, and $c = 10$. This problem has been used to test the scalability of multigrid solvers [9]. The problems are a constant size per compute node, from 10^3 to 60^3 local problem sizes. Node topologies of 1^3 to 5^3 were used. Thus the largest problem was over a cube with $(60 \times 5)^3 = 27,000,000$ unknowns. Each node used all four processors (4 worker threads, 1 server thread) for this problem in the preconditioner construction phase (i.e., 500 processors in the largest configuration).

A threshold for ParaSAILS was chosen so that only the nonzeros along the strongest (z) direction are retained, and the $\mathcal{S}_3$ pattern was used. Although this is a symmetric problem, the preconditioner is not symmetric, and we used GMRES(50) as the iterative solver with a zero initial guess. The convergence tolerance was 10^{-6}.

TABLE 4.9
Concentric shells problem: Timings for preconditioner setup using direct addressing.

npes	Precon
1	12.59
2	8.83
4	6.77
8	4.95
16	3.83
32	2.57
64	1.18

TABLE 4.10
Timings for work-hardening problem.

	ParaSAILS			SPAI		
npes	Precon	Solve	Total	Precon	Solve	Total
1	9.50	99.87	109.37	60.90	93.74	154.64
2	5.36	50.65	56.01	53.87	49.80	103.67
4	4.06	26.50	30.56	39.78	25.80	65.58
8	3.16	15.18	18.34	33.03	15.26	48.29
16	2.76	9.59	12.35	30.82	9.97	40.79
32	1.83	5.87	7.70	18.59	6.21	24.80
64	1.32	3.91	5.23	12.75	4.14	16.89

Table 4.11 shows the results, including wall-clock times for constructing the preconditioner and the iterative solve phase, the number of iterations required for convergence, the average time for one iteration in the solve phase, and E_p and E_s, the scaled efficiencies for constructing the preconditioner and for one step in the solve phase. Figures 4.2 and 4.3 graph the scaled efficiencies E_p and E_s, respectively. The implementation seems scalable for all values of p that may be encountered. For comparison, in Table 4.12, we show results for SPAI using a 40^3 local problem size. Larger problems led to excessive preconditioner construction times. Again, one processor per node was used for SPAI.

5. Conclusions. This paper demonstrates the effectiveness of patterns of powers of sparsified matrices for sparse approximate inverses for PDE problems. As opposed to many existing methods for prescribing sparsity patterns, PSM patterns use both the values and structure of the original matrix, and very sparse patterns can be produced. PSM patterns allow simpler direct methods of constructing sparse approximate inverse preconditioners to be used, with comparable preconditioning quality to adaptive methods, but with significantly less computational cost. The numerical tests show that the additional effort of adaptive sparsity pattern calculations is not always required.

Acknowledgments. The author is indebted to Wei-Pai Tang, who was one of the first to use sparsification for computing approximate inverse sparsity patterns. John Gilbert was instrumental in directing attention to the transitive closure of a matrix, and motivating the possibility of finding good patterns *a priori*. Michele Benzi made helpful comments and directed the author to [24]. The author is also grateful for the ongoing support of Robert Clay, Andrew Cleary, Robert Falgout, Esmond Ng, Ivan Otero, Yousef Saad, and Alan Williams, and finally for the cogent comments of the anonymous referees.

TABLE 4.11
Timings, iteration counts, and efficiencies for ParaSAILS.

$10 \times 10 \times 10$ *local problem size*							
npes	N	Precon	Solve	Iter	Solve/Iter	E_p	E_s
1	1000	0.09	0.06	13	0.0048	1.000	1.000
8	8000	0.78	0.25	26	0.0098	0.116	0.494
27	27000	1.39	0.69	40	0.0174	0.065	0.279
64	64000	1.79	1.07	54	0.0199	0.050	0.244
125	125000	10.42	1.73	68	0.0255	0.009	0.190

$20 \times 20 \times 20$ *local problem size*							
npes	N	Precon	Solve	Iter	Solve/Iter	E_p	E_s
1	8000	0.49	0.99	26	0.0379	1.000	1.000
8	64000	2.61	3.28	54	0.0607	0.189	0.625
27	216000	4.27	5.71	81	0.0705	0.116	0.538
64	512000	5.49	8.20	113	0.0726	0.090	0.523
125	1000000	10.81	11.51	144	0.0799	0.046	0.475

$30 \times 30 \times 30$ *local problem size*							
npes	N	Precon	Solve	Iter	Solve/Iter	E_p	E_s
1	27000	1.73	7.76	40	0.1941	1.000	1.000
8	216000	5.47	18.97	81	0.2342	0.317	0.829
27	729000	8.00	32.98	125	0.2639	0.216	0.735
64	1728000	10.87	50.10	179	0.2799	0.159	0.693
125	3375000	17.96	66.31	233	0.2846	0.096	0.682

$40 \times 40 \times 40$ *local problem size*							
npes	N	Precon	Solve	Iter	Solve/Iter	E_p	E_s
1	64000	4.21	28.32	54	0.5244	1.000	1.000
8	512000	9.31	65.44	113	0.5791	0.452	0.906
27	1728000	14.23	109.34	179	0.6108	0.296	0.859
64	4096000	17.82	162.07	250	0.6483	0.236	0.809
125	8000000	25.07	232.64	350	0.6647	0.168	0.789

$50 \times 50 \times 50$ *local problem size*							
npes	N	Precon	Solve	Iter	Solve/Iter	E_p	E_s
1	125000	8.28	65.02	68	0.9561	1.000	1.000
8	1000000	14.67	164.83	144	1.1447	0.564	0.835
27	3375000	24.02	277.94	233	1.1929	0.345	0.802
64	8000000	29.21	436.84	350	1.2481	0.283	0.766
125	15625000	32.13	596.05	476	1.2522	0.258	0.764

$60 \times 60 \times 60$ *local problem size*							
npes	N	Precon	Solve	Iter	Solve/Iter	E_p	E_s
1	216000	14.49	136.10	81	1.6803	1.000	1.000
8	1728000	24.05	345.94	179	1.9326	0.603	0.869
27	5832000	39.56	601.90	291	2.0684	0.366	0.812
64	13824000	49.21	970.46	456	2.1282	0.294	0.790
125	27000000	53.83	1371.59	636	2.1566	0.269	0.779

TABLE 4.12
Timings, iteration counts, and efficiencies for SPAI.

$40 \times 40 \times 40$ *local problem size*							
npes	N	Precon	Solve	Iter	Solve/Iter	E_p	E_s
1	64000	24.50	31.27	67	0.4668	1.000	1.000
8	512000	139.23	77.25	142	0.5440	0.176	0.858
27	1728000	580.23	132.39	230	0.5756	0.042	0.811
64	4096000	1731.26	204.92	347	0.5906	0.014	0.790

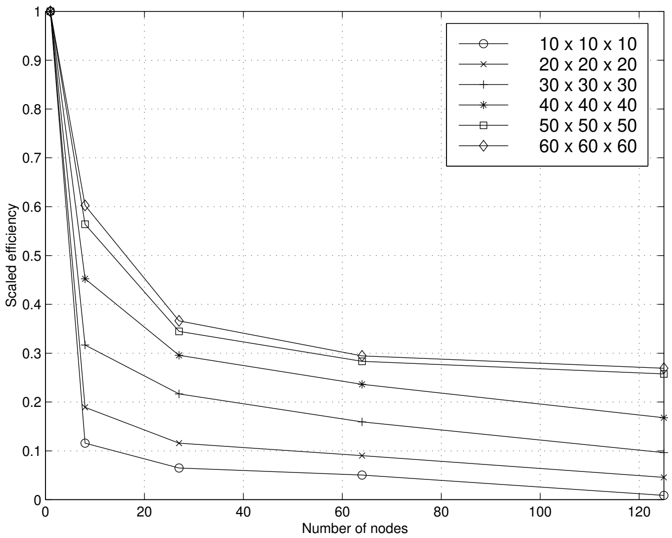


Fig. 4.2. *Implementation scalability of ParaSAILS preconditioner construction phase.*

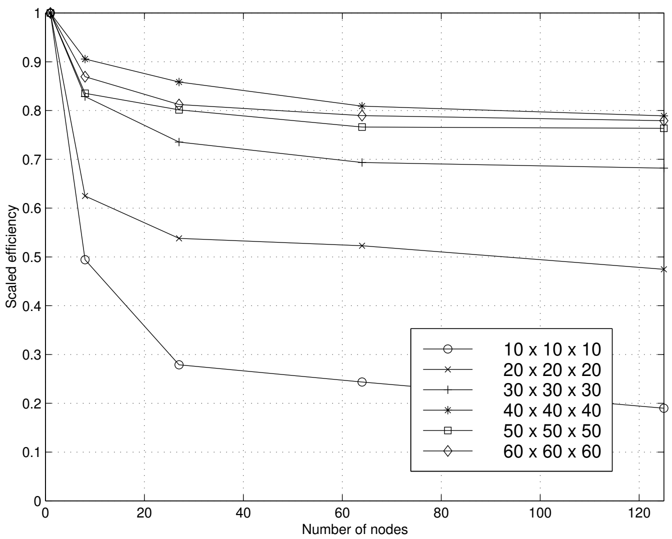


Fig. 4.3. *Implementation scalability of one step of iterative solution.*

REFERENCES

[1] G. Alléon, M. Benzi, and L. Giraud, *Sparse approximate inverse preconditioning for dense linear systems arising in computational electromagnetics*, Numerical Algorithms, 16 (1997), pp. 1–15.

[2] S. T. Barnard, L. M. Bernardo, and H. D. Simon, *An MPI implementation of the SPAI preconditioner on the T3E*, Tech. Report LBNL-40794, Lawrence Berkeley National Laboratory, Berkeley, CA, 1997.

[3] S. T. Barnard and R. Clay, *A portable MPI implementation of the SPAI preconditioner in ISIS++*, in Proc. Eighth SIAM Conference on Parallel Processing for Scientific Computing, Minneapolis, MN, March, 1997.

[4] M. W. Benson and P. O. Frederickson, *Iterative solution of large sparse linear systems arising in certain multidimensional approximation problems*, Utilitas Math., 22 (1982), pp. 127–140.

[5] M. Benzi, J. Marín, and M. Tůma, *A two-level parallel preconditioner based on sparse approximate inverses*, in Iterative Methods in Scientific Computation II, IMACS, 1994, pp. 1–11.

[6] M. Benzi and M. Tůma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., 19 (1998), pp. 968–994.

[7] ———, *Orderings for factorized sparse approximate inverse preconditioners*, SIAM J. Sci. Comput., to appear (1999).

[8] R. Bridson and W.-P. Tang, *An ordering method for a factorized approximate inverse preconditioner*, SIAM J. Sci. Comput., submitted (1998).

[9] P. N. Brown, R. D. Falgout, and J. E. Jones, *Semicoarsening multigrid on distributed memory machines*, SIAM J. Sci. Comput., (to appear).

[10] K. Chen, *On a class of preconditioning methods for dense linear systems from boundary elements*, SIAM J. Sci. Comput., 20 (1998), pp. 684–698.

[11] E. Chow and Y. Saad, *Approximate inverse techniques for block-partitioned matrices*, SIAM J. Sci. Comput., 18 (1997), pp. 1657–1675.

[12] ———, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., 19 (1998), pp. 995–1023.

[13] R. L. Clay, K. D. Mish, and A. B. Williams, *ISIS++: Iterative scalable implicit solver (in C++)*, Tech. Report SAND97-8535, Sandia National Laboratories, Livermore, CA, 1997.

[14] J. D. F. Cosgrove and J. C. Díaz, *Structural properties of the graph of augmented sparse approximate inverses*, in Proc. 1990 Symposium on Applied Computing, H. Berghel, J. Talburt, and D. Roach, eds., Fayetteville, AR, 1990, IEEE Computer Press, Los Alamitos, CA, pp. 131–136.

[15] J. D. F. Cosgrove, J. C. Díaz, and A. Griewank, *Approximate inverse preconditioning for sparse linear systems*, Intl. J. Comp. Math., 44 (1992), pp. 91–110.

[16] A. Cosnuau, *Etude d'un préconditionneur pour les matrices complexes dense issues des équations de Maxwell en formulation intégrale*, Tech. Report 142 328.96/DI/MT, ONERA, France, 1996.

[17] S. Demko, W. F. Moss, and P. W. Smith, *Decay rates for inverses of band matrices*, Math. Comp., 43 (1984), pp. 491–499.

[18] V. Deshpande, M. J. Grote, P. Messmer, and W. Sawyer, *Parallel implementation of a sparse approximate inverse preconditioner*, in Parallel Algorithms for Irregularly Structured Problems (Proc. IRREGULAR '96), A. Ferreira, J. Rolim, Y. Saad, and T. Yang, eds., Santa Barbara, CA, 1996, pp. 63–74.

[19] M. R. Field, *An efficient parallel preconditioner for the conjugate gradient algorithm*, Tech. Report HDL-TR-97-175, Hitachi Dublin Laboratory, Trinity College, Dublin, 1997.

[20] ———, *Improving the performance of parallel factorised sparse approximate inverse preconditioners*, Tech. Report HDL-TR-98-199, Hitachi Dublin Laboratory, Trinity College, Dublin, 1998.

[21] J. R. Gilbert, *Predicting structure in sparse matrix computations*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 62–79.

[22] M. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., 18 (1997), pp. 838–853.

[23] T. Huckle, *PVM-implementation of sparse approximate inverse preconditioners for solving large sparse linear equations*, in Lecture Notes in Computer Science, vol. 1156, Parallel Virtual Machine–EuroPVM'96, Springer, 1996, pp. 166–173.

[24] ———, *Approximate sparsity patterns for the inverse of a matrix and preconditioning*, in Prelim. Proc. IMACS World Congress 1997, R. Weiss and W. Schönauer, eds., Berlin, 1997.

[25] I. E. Kaporin, *A preconditioned conjugate gradient method for solving discrete analogs of*

differential problems, Differential Equations, 26 (1990), pp. 897–906.

[26] L. Yu. Kolotilina, *Explicit preconditioning of systems of linear algebraic equations with dense matrices*, J. Soviet Math., 43 (1988), pp. 2566–2573.

[27] L. Yu. Kolotilina, A. A. Nikishin, and A. Yu. Yeremin, *Factorized sparse approximate inverse preconditionings. IV: Simple approaches to rising efficiency*, tech. report, 1998.

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

[29] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Co., Boston, MA, 1996.

[30] W.-P. Tang, *Towards an effective sparse approximate inverse preconditioner*, SIAM J. Matrix Anal. Appl., to appear (1998).

[31] W.-P. Tang and W. L. Wan, *Sparse approximate inverse smoother for multi-grid*, Tech. Report CAM 98-18, Department of Mathematics, University of California, Los Angeles, CA, 1998.

[32] S. A. Vavasis, *Preconditioning for boundary integral equations*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 905–925.