SIAM J. Sci. Comput.
Vol. 22, No. 6, pp. 2194–2215
© 2001 Society for Industrial and Applied Mathematics

A SCALABLE PARALLEL ALGORITHM FOR INCOMPLETE FACTOR PRECONDITIONING*

DAVID HYSOM† AND ALEX POTHEN†

Abstract. We describe a parallel algorithm for computing incomplete factor (ILU) preconditioners. The algorithm attains a high degree of parallelism through graph partitioning and a two-level ordering strategy. Both the subdomains and the nodes within each subdomain are ordered to preserve concurrency. We show through an algorithmic analysis and through computational results that this algorithm is scalable. Experimental results include timings on three parallel platforms for problems with up to 20 million unknowns running on up to 216 processors. The resulting preconditioned Krylov solvers have the desirable property that the number of iterations required for convergence is insensitive to the number of processors.

Key words. incomplete factorization, ILU, preconditioning, parallel preconditioning

AMS subject classifications. 65F50, 65F10, 68W10, 68R10

PII. S1064827500376193

1. Introduction.

Incomplete factorization (ILU) preconditioning is currently among the most robust techniques employed to improve the convergence of Krylov space solvers for linear systems of equations. (ILU stands for incomplete LU factorization, where L and U are the lower and upper triangular (incomplete) factors of the coefficient matrix.) However, scalable parallel algorithms for computing ILU preconditioners have not been available despite the fact that they have been used for more than twenty years [12]. We report the design, analysis, implementation, and computational evaluation of a parallel algorithm for computing ILU preconditioners.

Our parallel algorithm assumes that three requirements are satisfied.

- The adjacency graph of the coefficient matrix (or the underlying finite element or finite difference mesh) must have good edge separators, i.e., it must be possible to remove a small set of edges to divide the problem into a collection of subproblems that have roughly equal computational work requirements.
- The size of the problem must be sufficiently large relative to the number of processors so that the work required by the subgraph on each processor is suitably large to dominate the work and communications needed for the boundary nodes.
- The subdomain intersection graph (to be defined later) should have a small chromatic number. This requirement will ensure that the dependencies in factoring the boundary rows do not result in undue losses in concurrency.

An outline of the paper is as follows. In section 2, we describe the steps in the parallel algorithm for computing the ILU preconditioner in detail and provide theoretical justification. The algorithm is based on an incomplete fill path theorem; the proof and discussion of the theorem are deferred to an appendix. We also discuss

*Received by the editors August 4, 2000; accepted for publication (in revised form) December 17, 2000; published electronically April 26, 2001. This work was supported by U. S. National Science Foundation grants DMS-9807172 and ECS-9527169, by the U. S. Department of Energy under subcontract B347882 from the Lawrence Livermore Laboratory, by a GAANN fellowship from the Department of Education, and by NASA under contract NAS1-19480 while the authors were in residence at ICASE.

http://www.siam.org/journals/sisc/22-6/37619.html

†Old Dominion University, Norfolk, VA 23529-0162 and ICASE, NASA Langley Research Center, Hampton VA 23681-2199 (hysom@cs.odu.edu, pothen@cs.odu.edu).

the role that a subdomain graph constraint plays in the design of the algorithm, show that the preconditioners exist for special classes of matrices, and relate our work to earlier work on this problem. Section 3 contains an analysis that shows that the parallel algorithm is scalable for two-dimensional (2D-) and three-dimensional (3D-)model problems, when they are suitably ordered and partitioned. Section 4 contains computational results on Poisson and convection-diffusion problems. The first subsection shows that the parallel ILU algorithm is scalable on three parallel platforms; the second subsection reports convergence studies. We tabulate how the number of Krylov solver iterations and the number of entries in the preconditioner vary as a function of the preconditioner level for three variations of the algorithm. The results show that fill levels higher than one are effective in reducing the number of iterations; the number of iterations is insensitive to the number of subdomains; and the subdomain graph constraint does not affect the number of iterations while it makes possible the design of a simpler parallel algorithm.

The background needed for ILU preconditioning may be found in several books; see, e.g., [1, 15, 17, 33]. A preliminary version of this paper was presented at Supercomputing '99 and was published in the conference proceedings [18]. The algorithm has been revised, additional details have been included, and the proof of the theorem on which it is based has been added. The experimental results in section 4 are new, and most of them have been included in the technical reports [19, 20].

2. Algorithms.

In this section we discuss the Parallel ILU (PILU) algorithm and its underlying theoretical foundations.

2.1. The PILU algorithm.

Figure 2.1 describes the steps of the PILU algorithm at a high level; the algorithm is suited for implementation on both message-passing and shared-address space programming models.

The PILU algorithm consists of four major steps. In the first step, we create parallelism by dividing the problem into subproblems by means of graph partitioning. In the second step, we preserve the parallelism in the interior of the subproblems by locally scheduling the computations in each subgraph. In the third step, we preserve parallelism in the boundaries of the subproblems by globally ordering the subproblems through coloring a suitably defined graph. In the final step, we compute the preconditioner in parallel. Now we will describe the four steps in greater detail.

Step 1: Graph partitioning. In the first step of PILU, we partition the adjacency graph $G(A)$ of the coefficient matrix A into p subgraphs by removing a small set of edges that connects the subgraphs to each other. Each subgraph will be mapped to a distinct processor that will be responsible for the computations associated with the subgraph.

An example of a model five-point grid partitioned into four subgraphs is shown in Figure 2.2. For clarity, the edges corresponding to the coefficient matrix elements (within each subgraph or between subgraphs) are not shown. The edges drawn correspond to fill elements (elements that are zero in the coefficient matrix but are nonzero in the incomplete factors) that join the different subgraphs.

To state the objective function of the graph partitioning problem, we need to introduce some terminology. An edge is a *separator edge* if its endpoints belong to different subgraphs. A vertex in a subgraph is an *interior vertex* if all of its neighbors belong to that subgraph; it is a *boundary vertex* if it is adjacent to one or more vertices belonging to another subgraph. By definition, an interior vertex in a subgraph is not adjacent to a vertex (boundary or interior) in another subgraph. In Figure 2.2, the first 25 vertices are interior vertices of the subgraph S_0, and vertices numbered 26 through 36 are its

Input: A coefficient matrix, its adjacency graph, and the number of processors p.
Output: The incomplete factors of the coefficient matrix.

1. Partition the adjacency graph of the matrix into p subgraphs (subdomains), and map each subgraph to a processor. The objectives of the partitioning are that the subgraphs should have roughly equal work, and there should be few edges that join the different subgraphs.
2. On each subgraph, locally order interior nodes first, and then order boundary nodes.
3. Form the subdomain intersection graph corresponding to the partition, and compute an approximate minimum vertex coloring for it. Order subdomains according to color classes.
4. Compute the incomplete factors in parallel.
   a. Factor interior rows of each subdomain.
   b. Receive sparsity patterns and numerical values of the nonzeros of the boundary rows of lower-numbered subdomains adjacent to a subdomain (if any).
   c. Factor boundary rows in each subdomain and send the sparsity patterns and numerical values to higher-numbered neighboring subdomains (if any).

Fig. 2.1. *High level description of the PILU algorithm.*

boundary vertices. The goal of the partitioning is to keep the amount of work associated with the incomplete factorization of each subgraph roughly equal, while keeping the communication costs needed to factor the boundary rows as small as possible.

There is a difficulty with modeling the communication costs associated with the boundary rows. In order to describe this difficulty, we need to relate this cost more precisely to the separators in the graph. Define the *higher degree* of a vertex v as the number of vertices numbered higher than v in a given ordering. We assume that upward-looking, row-oriented factorization is used. At each boundary between two subgraphs, elements need to be communicated from the lower numbered subgraph to the higher numbered subgraph. The number of these elements is proportional to the sum of the higher degrees (in the filled graph $G(F)$) of the boundary vertices in the lower numbered subgraph. But unfortunately, we do not know the fill edges at this point since we have neither computed an ordering of $G(A)$ nor computed a symbolic factorization. We could approximate by considering higher degrees of the boundary vertices in the graph $G(A)$ instead of the filled graph $G(F)$, but even this requires us to order the subgraphs in the partition.

The union of the boundary vertices on all the subgraphs forms a *wide vertex separator*. This means that the shortest path from an interior vertex in any subgraph to an interior vertex in another subgraph consists of at least three edges; such a path has *length* at least three. The communication cost in the (forward and backward) triangular solution steps is proportional to the sum of the sizes of the wide vertex separators. None of the publicly available graph partitioning software has the minimization of wide separators as its objective function, but it is possible to modify existing software to optimize this objective.

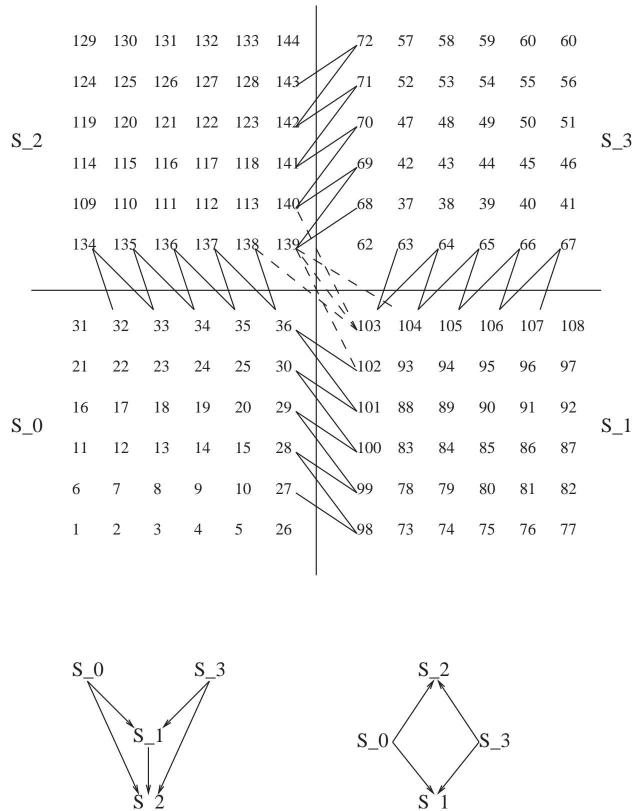

FIG. 2.2. *An example that shows the partitioning, mapping, and vertex ordering used in the PILU algorithm. The graph on the top is a regular* 12×12 *grid with a five-point stencil partitioned into four subdomains and then mapped on four processors. The subdomains are ordered by a coloring algorithm to reduce dependency path lengths. Only the level one and two fill edges that join the different subdomains are shown; all other edges are omitted for clarity. The figure on the bottom right shows the subdomain intersection graph when the subdomain graph constraint is enforced. (This prohibits fill between the boundary nodes of the subdomains* S_1 *and* S_2*, indicated by the broken edges in the top graph.) The graph on the bottom left shows the subdomain intersection graph when the subdomain graph constraint is not enforced.*

The goal of the partitioning step is to keep the amount of work associated with each subgraph roughly equal (for load balance) while making the communication costs due to the boundaries as small as possible. As the previous two paragraphs show, modeling the communication costs accurately in terms of edge and vertex separators in the initial graph $G(A)$ is difficult, but we could adopt the minimization of the wide separator sizes as a reasonable goal. This problem is NP-complete, but there exist efficient heuristic algorithms for partitioning the classes of graphs that occur in practical situations. (Among these graph classes are 2D-finite element meshes and 3D-meshes with good aspect ratios.)

Step 2: Local reordering. In the second step, in each subgraph we order the interior vertices before the boundary vertices. This ordering ensures that during the incomplete factorization, an interior vertex in one subgraph cannot be joined by a fill edge to a vertex in another subgraph, as will be shown later. Fill edges between two subgraphs can join only their boundary vertices together. Thus interior vertices corresponding to the initial graph $G(A)$ remain interior vertices in the graph of the

factor $G(F)$. The consequences of this are that the rows corresponding to the interior vertices in each subdomain of the initial problem $G(A)$ can be factored concurrently, and that communication is required only for factoring rows corresponding to the boundary rows. The reader can verify that in each subgraph in Figure 2.2 the interior nodes have been ordered before the boundary nodes.

The observation concerning fill edges in the preceding paragraph results from an application of the following *incomplete fill path theorem.* Given the adjacency graph $G(A)$ of a coefficient matrix A, the theorem provides a static characterization of where fill entries arise during an incomplete factorization $A = \hat{L}\hat{U} + E$, where $\hat{L}$ is the lower triangular incomplete factor, $\hat{U}$ is the upper triangular incomplete factor, and E is the remainder matrix. The characterization is static in that fill is completely described by the structure of the graph $G(A)$; no information from the factor is required.

We need a definition before we can state the theorem. A *fill path* is a path joining two vertices i and j, all of whose interior vertices are numbered lower than the end vertices i and j.[1]

Recall also the definition of the levels assigned to nonzeros in an incomplete factorization. To discuss the sparsity pattern of the incomplete factors, we consider the filled matrix $F = \hat{L} + \hat{U} - I$. The sparsity pattern of F is initialized to that of A. All nonzero entries in F corresponding to nonzeros in A have level zero, and zero entries have level infinity. New entries that arise during factorization are assigned a level based on the levels of the causative entries, according to the rule

$$\text{level}(f_{ij}) = \min_{1 \le h < \min\{i,j\}} \{\text{level}(f_{ih}) + \text{level}(f_{hj}) + 1\}.$$

The incomplete fill path theorem describes an intimate relationship between fill entries in ILU(k) factors and path lengths in graphs.

Theorem 2.1. *Let $F = \hat{L} + \hat{U} - I$ be the filled matrix corresponding to an incomplete factorization of A, and let f_{ij} be a nonzero entry in F. Then f_{ij} is a level k entry if and only if there exists a shortest fill path of length $k+1$ that joins i and j in $G(A)$.*

A proof and a discussion of this theorem are included in the appendix.

Now consider the adjacency graph $G(A)$ and a partition $\Pi = \{S_0, \dots, S_{p-1}\}$ of it into subgraphs (subdomains). Any path joining two interior nodes in distinct subdomains must include at least two boundary nodes, one from each of the subgraphs; since each boundary node is numbered higher than (at least one of) the path's end vertices (since these are interior nodes in the subgraph), this path cannot be a fill path. If two interior nodes belonging to separate subgraphs were connected by a fill path and the corresponding fill entry were permitted in F, the interior nodes would be transformed into boundary nodes in $G(F)$. This is undesirable for parallelism, since then there would be fewer interior nodes to be eliminated concurrently.

The local ordering step preserves interior and boundary nodes during the factorization and ensures that a subdomain's interior rows can be factored independently of row updates from any other subdomain. Therefore, when subdomains have relatively large interior/boundary node ratios, and contain approximately equal amounts of computational work, we expect PILU to exhibit a high degree of parallelism.

[1] The reader has doubtless noted that *interior* is used in a different sense here than previously. We trust it will be obvious from the context where *interior* is used to refer to nodes in paths and where it is used to refer to nodes in subgraphs.

Step 3: Global ordering. The global ordering phase is intended to preserve parallelism while factoring the rows corresponding to the boundary vertices. In order to explain the loss of concurrency that could occur during this phase of the algorithm, we need the concept of a subdomain intersection graph, which we shall call a subdomain graph for brevity.

The subdomain graph $S(G, \Pi) = (V_s, E_s)$ is computed from a graph G and its partition $\Pi = \{S_0, \ldots, S_{p-1}\}$ into subgraphs. The vertex set V_s contains a vertex corresponding to every subgraph in the partition; the edge set E_s contains edge $\{S_i, S_j\}$ if there is an edge in G with one endpoint in S_i and the other in S_j. We can compute a subdomain graph $S(A)$ corresponding to the initial graph $G(A)$ and its partition. (This graph should be denoted $S(G(A), \Pi)$, but we shall write $S(A)$ for simplicity.) We could also compute a subdomain graph $S(F)$ corresponding to the graph of the factor $G(F)$. The subdomain graph $S(A)$ corresponding to the partition of the initial graph $G(A)$ (the top graph) in Figure 2.2 is shown in the graph at the bottom right in that figure.

We impose a constraint on the fill, *the subdomain graph constraint*. The subdomain graph corresponding to $G(F)$ is restricted to be identical to the subdomain graph corresponding to $G(A)$. This prohibits some fill in the filled graph $G(F)$: if two subdomains are not joined by an edge in the original graph $G(A)$, any fill edge that joins those subdomains is not permitted in the graph of the incomplete factor $G(F)$. The description of the PILU algorithm in Figure 2.1 assumes that the subdomain graph constraint is satisfied. This constraint makes it possible to obtain scalability in the parallel ILU algorithm. Later, we discuss how the algorithm should be modified if this constraint is relaxed.

Each subdomain's nodes (in $G(A)$) are ordered contiguously. Consequently, saying "subdomain r is ordered before subdomain s" is equivalent to saying "all nodes in subdomain r are ordered, and then all nodes in subdomain s are ordered." This permits $S(A)$ to be considered as a directed graph, with edges oriented from lower to higher numbered vertices.

Edges in $S(F)$ indicate data dependencies in factoring the boundary rows of the subdomains. If an edge in $S(F)$ joins r and s and subdomain r is ordered before subdomain s, then updates from the boundary rows of r have to be applied to the boundary rows of s before the factorization of the latter rows can be completed. It follows that ordering $S(F)$ so as to reduce directed path lengths reduces serial bottlenecks in factoring the boundary rows. If we impose the subdomain graph constraint, these observations apply to the subdomain graph $S(A)$ as well since then $S(A)$ is identical with $S(F)$.

We reduce directed path lengths in $S(A)$ by coloring the vertices of the subdomain graph with few colors using a heuristic algorithm for graph coloring, and then by numbering the subdomains by color classes. The boundary rows of all subdomains corresponding to the first color can be factored concurrently without updates from any other subdomains. These subdomains update the boundary rows of higher numbered subdomains adjacent to them. After the updates, the subdomains that correspond to the second color can factor their boundary rows. This process continues by color classes until all subdomains have factored their boundary rows. The number of steps it takes to factor the boundary rows is equal to the number of colors it takes to color the subdomain graph.

In Figure 2.2, let p_i denote the processor that computes the subgraph S_i. Then p_0 computes the boundary rows of S_0 and sends them to processors p_1 and p_2. Similarly,

p_3 computes the boundary rows of subgraph S_3 and sends them to p_1 and p_2. The latter processors first apply these updates and then compute their boundary rows.

How much parallelism can be gained through subdomain graph reordering? We can gain some intuition through analysis of simplified model problems, although we cannot answer this question a priori for general problems and all possible partitions. Consider a matrix arising from a second order PDE that has been discretized on a regularly structured 2D grid using a standard five-point stencil. Assume that the grid is naturally ordered and that it has been partitioned into square subgrids and mapped into a square grid of p processors. In the worst case, the associated subdomain graph, which itself has the appearance of a regular 2D grid, can have a dependency path of length $2(\sqrt{p} - 1)$. Similarly, a regularly structured 3D grid discretized with a seven-point stencil that is naturally ordered and then mapped on a cube containing p processors can have a dependency path length of $3(\sqrt[3]{p} - 1)$. However, regular 2D grids with the five-point stencil and regular 3D grids with the seven-point stencil are bipartite graphs and can be colored with two colors. If all subdomains of the first color class are numbered first, and then all subdomains of the second color class are numbered, the longest dependency path in S will be reduced to one. This discussion shows that coloring the subdomain graph is an important step in obtaining a scalable parallel algorithm.

Step 4: Preconditioner computation. Now that the subdomains and the nodes in each subdomain have been ordered, the preconditioner can be computed. We employ an upward-looking, row oriented factorization algorithm. The interior of each subdomain can be computed concurrently by the processors, and the boundary nodes can be computed in increasing order of the color classes. Either a level-based ILU(k) or a numerical threshold based ILUT(τ, p) algorithm may be employed on each subdomain. Different incomplete factorization algorithms could be employed in different subdomains when appropriate, as in multiphysics problems. Different fill levels could be employed for the interior nodes in a subdomain and for the boundary nodes to reduce communication and synchronization costs.

2.2. Relaxing the subdomain graph constraint. Now we consider how the subdomain graph constraint might be relaxed. Given a graph $G(A)$ and a partition of it into subgraphs, we color the subdomain graph $S(A)$ and order its subdomains as before. Then we compute the graph $G(F)$ of an incomplete factor and its subdomain graph $S(F)$. To do this, we need to discover the dependencies in $S(F)$, but initially we have only the dependencies in $S(A)$ available. This has to be done in several rounds, because fill edges could create additional dependencies between the boundary rows of subdomains, which in turn might lead to further dependences. The number of rounds needed is the length of a longest dependency path in the subdomain graph $G(F)$, and this could be $\Omega(p)$. This discussion applies when an ILU(k) algorithm is employed, with symbolic factorization preceding numerical factorization. If ILUT were to be employed, then symbolic factorization and numerical factorization must be interleaved, as would be done in a sequential algorithm.

We can then color the vertices of $S(F)$ to compute a schedule for factoring the boundary rows of the subdomains. For achieving concurrency in this step the subdomain graph $S(F)$ should have a small chromatic number (independent of the number of vertices in $G(A)$). Note that the description of the PILU algorithm in Figure 2.1 needs to be modified to reflect this discussion when the subdomain graph constraint is relaxed.

The graph $G(F)$ in Figure 2.2 indicates the fill edges that join S_1 to S_2 as broken

lines. The corresponding subdomain intersection graph $S(F)$ is shown on the lower left. The edge between S_1 and S_2 necessitates three colors to color $S(F)$: the subdomains S_0 and S_3 form one color class; S_1 by itself constitutes the second color class; and S_2 by itself makes up the third color class. Thus three steps are needed for the computation of the boundary rows of the preconditioner when the subdomain graph constraint is relaxed. Note that the processor responsible for the subdomain S_2 can begin computing its boundary rows when it receives an update from either S_0 or S_3, but that it cannot complete its computation until it has received the update from the subdomain S_1.

Theorem 2.1 has an intuitively simple geometric interpretation. Given an initial node i in $G(A)$, construct a topological "sphere" containing all nodes that are at a distance less than or equal to $k + 1$ edges. Then a fill entry f_{ij} is admissible in an $ILU(k)$ factor only if j is within the sphere. Note that all such nodes j do not cause fill edges since there needs to be a fill path joining i and j. By applying Theorem 2.1, we can gain an intuitive understanding of the fill entries that may be discarded on account of the subdomain graph constraint. Referring again to Figure 2.2, we see that prohibited edges arise when two nonadjacent subdomains in $G(A)$ have nodes that are joined by a fill path of length less than $k + 1$. No level zero edge is discarded by the constraint.

2.3. Existence of PILU preconditioners. The existence of preconditioners computed from the PILU algorithm can be proven for some classes of problems.

Meijerink and van der Vorst [28] proved that if A is an M-matrix, then ILU factors exist for any predetermined sparsity pattern, and Manteuffel [27] extended this result to H-matrices with positive diagonal elements. These results immediately show that PILU preconditioners with sparsity patterns based on level values exist for these classes of matrices. This is true even when different level values are used for the various subdomains and boundaries.

Incomplete Cholesky (IC) preconditioners for symmetric problems could be computed with our parallel algorithmic framework using preconditioners proposed by Jones and Plassmann [21] and by Lin and Moré [23] on each subdomain and on the boundaries. The sparsity patterns of these preconditioners are determined by the numerical values in the matrix and by memory constraints. Lin and Moré have proved that these preconditioners exist for M- and H-matrices. Parallel IC preconditioners also can be shown to exist for M- and H-matrices. If the subdomain graph constraint is not enforced, then the preconditioner computed in parallel corresponds to a preconditioner computed by the serial algorithm from a reordered matrix. If the constraint is enforced, some specified fill elements are dropped from the Schur complement; it can be shown that the resulting Schur complement matrix is componentwise larger than the former and hence still an M-matrix.

2.4. Relation to earlier work. We now briefly discuss earlier parallel ILU algorithms that are related to the PILU algorithm proposed here. Earlier attempts at parallel algorithms for preconditioning (including approaches other than incomplete factorization) are surveyed in [6, 12, 34]; orderings suitable for parallel incomplete factorizations have been studied inter alios in [4, 11, 13]. The surveys also describe the alternate approximate inverse approach to preconditioning.

Saad [33, section 12.6.1] discusses a distributed ILU(0) algorithm that has the features of graph partitioning, elimination of interior nodes in a subdomain before boundary nodes, and coloring the subdomains to process the boundary nodes in parallel. Only level 0 preconditioners are discussed there, so that fill between subdomains, or

within each subdomain, do not need to be considered. No implementations or results were reported, although Saad has informed us recently of a technical report [24] that includes an implementation and results. Our work, done independently, shows how fill levels higher than zero can be accommodated within this algorithmic framework. We also analyze our algorithm for scalability and provide computational results on the performance of PILU preconditioners. Our results show that fill levels higher than zero are indeed necessary to obtain parallel codes with scalability and good performance.

Karypis and Kumar [22] have described a parallel ILUT implementation based on graph partitioning. Their algorithm does not include a symbolic factorization, and they discover the sparsity patterns and the values of the boundary rows after the numerical computation of the interior rows in each subdomain. The factorization of the boundary rows is done iteratively, as in the discussion given above, where we show how the subdomain graph constraint might be relaxed. The partially filled graph of the boundary rows after the interior rows are eliminated is formed, and this graph is colored to compute a schedule for computing the boundary rows. Since fill edges in the boundary rows are discovered as these rows are being factored, this approach could lead to long dependency paths that are $\Theta(p)$. The number of boundary rows is $\Omega(N^{1/2})$ for 2D meshes, and $\Omega(N^{2/3})$ for 3D meshes with good aspect ratios. If the cost of factoring and communicating a boundary row is proportional to the number of rows, then this phase of their algorithm could cost $\Omega(p\sqrt{N})$, severely limiting the scalability of the algorithm (cf. the discussion in section 3).

Recently Magolu monga Made and van der Vorst [25, 26] have reported variations of a parallel algorithm for computing ILU preconditioners. They partition the mesh, linearly order the subdomains, and then permit fill in the interior and the boundaries of the subdomains. The boundary nodes are classified with respect to the number of subdomains they are adjacent to, and are eliminated in increasing order of this number. Since the subdomains are linearly ordered, a "burn from both ends" ordering is employed to eliminate the subdomains. Our approaches are similar, except that we additionally order the subdomains by means of a coloring to reduce dependency path lengths to obtain a scalable algorithm. They have provided an analysis of the condition number of the preconditioned matrices for a class of 2D second order elliptic boundary value problems. They permit high levels of fill (four or greater) as we do, and show that the increased fill permitted across the boundaries enables the condition number of the preconditioned matrix to be insensitive to the number of subdomains (except when the latter gets too great). We have worked independently of each other.

A different approach, based on partitioning the mesh into rectangular strips and then computing the preconditioner in parallel steps in which a "wavefront" of the mesh is computed at each step by the processors, was proposed by Bastian and Horton [3] and was implemented for shared memory multiprocessors recently by Vuik, van Nooyen, and Wesseling [36]. This approach has less parallelism than the one considered here.

3. Performance analysis. In this section we present simplified theoretical analyses of algorithmic behavior for matrices arising from PDEs discretized on 2D grids with five-point stencils and 3D grids with seven-point stencils. Since our arguments are structural in nature, we assume ILU(k) is the factorization method used. After a word about nomenclature, we begin with the 2D case.

The word *grid* refers to the grid (mesh) of unknowns for regular 2D and 3D grids with five- and seven-point stencils, respectively; this is identical to the adjacency graph $G(A)$ of the coefficient matrix of these problems. We use the terms *eliminating*

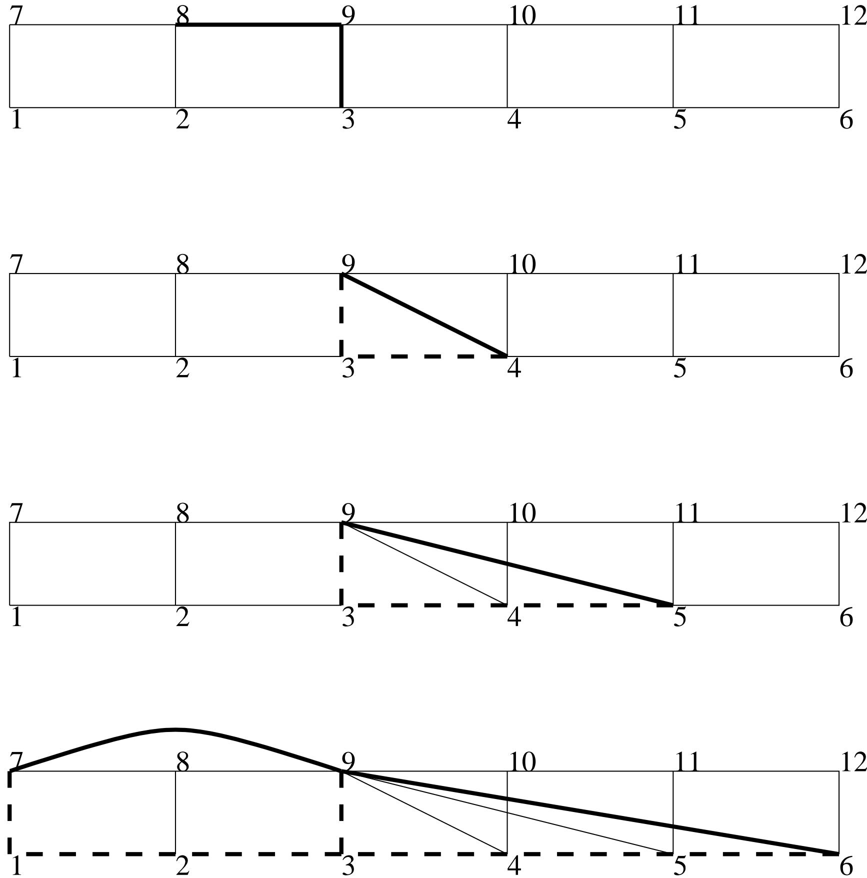

Fig. 3.1. *Counting lower triangular fill edges in a naturally ordered grid. We count the number of edges incident on vertex* 9*. Considering the graphs from top to bottom, we find that there are two level* 0 *edges; there is one level* 1 *edge, due to fill path* 9, 3, 4*; there is one level* 2 *edge due to fill path* 9, 3, 4, 5*; there are two level* 3 *edges, due to fill paths* 9, 3, 4, 5, 6 *and* 9, 3, 2, 1, 7*. We can generalize that two additional fill edges are created for every level greater than three, except near the boundaries. We conclude that asymptotically there are* $2k$ *lower triangular edges incident on a vertex in a level* k *factorization. Since the mesh corresponds to a structurally symmetric problem, there are* $2k$ *upper triangular edges incident on a vertex as well.*

a node and *factoring a row* synonymously.

We assume the grid has been block-partitioned, with each subdomain consisting of a square subgrid of dimension $c \times c$. We also assume the subdomain grid has dimensions $\sqrt{p} \times \sqrt{p}$, so there are p processors in total. There are thus $N = c^2 p$ nodes in the grid, and subdomains have at most $4c = 4\sqrt{\frac{N}{p}}$ boundary nodes.

If subdomain interior nodes are locally numbered in natural order and $k \ll c$, each row in the factor F asymptotically has $2k$ (strict) upper triangular and $2k$ (strict) lower triangular nonzero entries. The justification for this statement arises from a consideration of the incomplete fill path theorem; the intuition is illustrated in Figure 3.1.

Assuming that the classical ILU(k) algorithm is used for symbolic factorization, both symbolic and numeric factorization of row j entails $4k^2$ arithmetic operations. This is because for each lower triangular entry f_{ji} in matrix row j, factorization requires an arithmetic operation with each upper triangular entry in row i.

A red-black ordering of the subdomain graph gives an optimal bipartite division.

If red subdomains are numbered before black subdomains, our algorithm simplifies to the following three stages.

1. Red processors eliminate all nodes; black processors eliminate interior nodes.
2. Red processors send boundary-row structure and values to black processors.
3. Black processors eliminate boundary nodes.

If these stages are nonoverlapping, the cost of the first stage is bounded by the cost of eliminating all nodes in a subdomain. This cost is $4k^2c^2 = \frac{4k^2N}{p}$.

The cost for the second stage is the cost of sending structural and numerical values from the upper-triangular portions of the boundary rows to neighboring processors. If $k \ll c$, the incomplete fill path theorem can be used to show that, asymptotically, a processor only needs to forward values from c rows to each neighbor. We assume a standard, noncontentious communication model wherein α and β represent message startup and per-word-transfer times, respectively. We measure these times in nondimensional units of flops by dividing them by the time it takes to execute one flop. The time for an arithmetic operation is thus normalized to unity. Then the cost for the second step is $4(\alpha + 2k\beta c) = 4(\alpha + 2k\beta\sqrt{\frac{N}{p}})$.

Since the cost of factoring a boundary row can be shown to be asymptotically identical to that for factoring an interior row, the cost for eliminating the $4c$ boundary nodes is $(4k^2)(4c) = 16k^2\sqrt{\frac{N}{p}}$. Speedup can then be expressed as

$$\text{speedup} = \frac{4k^2N}{\frac{4k^2N}{p} + 4(\alpha + 2k\beta\sqrt{\frac{N}{p}}) + 16k^2\sqrt{\frac{N}{p}}}.$$

The numerator represents the cost for sequential execution, and the three terms in the denominator represent the costs for the three stages (arithmetic for interior nodes, communication costs, and arithmetic for the boundary nodes) of the parallel algorithm.

Three implications from this equation are in order. First, for a fixed problem size and number of processors, the parallel computational cost (the first and third terms in the denominator) is proportional to k^2, while the communication cost (the second term in the denominator) is proportional to k. This explains the increase in efficiency with level that we have observed. Second, if the ratio N/p is large enough, the first term in the denominator will become preeminent, and efficiency will approach 100%. Third, if we wish to increase the number of processors p by some factor while maintaining a constant efficiency, we need only increase the size of the problem N by the same factor. This shows that our algorithm is scalable. This observation is not true for a direct factorization of the coefficient matrix, where the dependencies created by the additional fill cause loss in concurrency.

For the 3D case we assume partitioning into cubic subgrids of dimension $c \times c \times c$ and a subdomain grid of dimension $p^{1/3} \times p^{1/3} \times p^{1/3}$, which gives $N = c^3p$. Subdomains have at most $6c^2$ boundary nodes. A development similar to that above shows that, asymptotically, matrix rows in the factor F have $2k^2$ (strict) upper and lower triangular entries, so the cost for factoring a row is $4k^4$. Speedup for this case can then be expressed as

$$\begin{aligned}\text{speedup} &= \frac{4k^4N}{\frac{4k^4N}{p} + 6(\alpha + 2k^2\beta(\frac{N}{p})^{1/3}) + 24k^4(\frac{N}{p})^{1/3}}\\ &= \frac{2k^4N}{\frac{2k^4N}{p} + 3(\alpha + 2k^2\beta(\frac{N}{p})^{1/3}) + 12k^4(\frac{N}{p})^{1/3}}.\end{aligned}$$

4. Results. Results in this section are based on the following model problems.

Problem 1. Poisson's equation in two or three dimensions:

$$\Delta u = g.$$

Problem 2. Convection-diffusion equation with convection in the xy plane:

$$-\varepsilon \triangle u + \frac{\partial}{\partial x} e^{xy} u + \frac{\partial}{\partial y} e^{-xy} u = g.$$

Homogeneous boundary conditions were used for both problems. Derivative terms were discretized on the unit square or cube, using 3-point central differencing on regularly spaced $n_x \times n_y \times n_z$ grids ($n_z = 1$ for 2D). The values for ε in Problem 2 were set to 1/500 and 1/1000. The problem becomes increasingly unsymmetric, and more difficult to solve accurately as ε decreases. The right-hand sides of the resulting systems, $Ax = b$, were artificially generated as $b = A\hat{e}$, where $\hat{e}$ is the all-ones vector.

ILU(k) preconditioning is amenable to performance analysis since the nonzero structures of ILU(k) preconditioners are identical for *any* PDE that has been discretized on a 2D or 3D grid with a given stencil. The structure depends on the grid and the stencil only and is not affected by numerical values if pivoting is not needed for numerical stability. Identical structures imply identical symbolic factorization costs, as well as identical flop counts during the numerical factorization and solve phases. In parallel contexts, communication patterns and costs are also identical. While preconditioner effectiveness—the number of iterations until the stopping criteria is reached—differs with the numerics of the particular problem being modeled, the parallelism available in the preconditioner does not.

The structure of ILUT preconditioners, on the other hand, is a function of the grid, the stencil, and the numerics. Changing the problem, particularly for non-diagonally dominant cases, can alter the preconditioner structure, even when the grid and stencil remain the same.

We report our performance evaluation for ILU(k) preconditioners, although the parallel algorithmic framework proposed here could just as easily work with ILUT(τ, p). We have compared the performance of ILU(k) with ILUT in an earlier report [18]. We report there that for Problem 2 with $\varepsilon = 1/500$, ILUT(0.001, 10) incurred more fill than ILU(5) on a 2D domain for grid sizes up to 400×400; for 3D domains and grid sizes up to $64 \times 64 \times 64$, the same ILUT preconditioner incurred fill between ILU(2) and ILU(3).

In addition to demonstrating that our algorithm can provide high degrees of parallelism, we address several other issues. We study the influence of the subdomain graph constraint on the fill permitted in the preconditioner and on the convergence of preconditioned Krylov space solvers. We also report convergence results as a function of the number of nonzeros in the preconditioner.

4.1. Parallel performance. We now report timing and scalability results for preconditioner factorization and application on three parallel platforms:

- an SGI Origin2000 at NASA Ames Research Center (AMES);
- the Coral PC Beowulf cluster at ICASE, NASA Langley Research Center;
- a Sun HPC 10000 Starfire server at Old Dominion University (ODU).

Table 4.1

Time (sec.) required for incomplete (symbolic and numeric) factorization for a 3D *scaled problem;* 91, 125 *unknowns per processor, seven-point stencil, ILU*(2) *factorization on interior nodes, and ILU*(1) *factorization on boundary nodes. Dashes (-) for Beowulf and HPC* 10000 *indicate that the machines have insufficient cpus to perform the runs.*

Procs	Origin2000 AMES	Beowulf (ICASE)	HPC 10000 (ODU)
1	2.04	2.27	2.13
8	2.44	3.11	2.43
27	2.96	4.06	2.97
64	3.11	4.64	-
125	3.18	-	-
216	3.32	-	-

Both problems were solved using Krylov subspace methods as implemented in the PETSc [2] software library. Problem 1 was solved using the conjugate gradient method, and Problem 2 was solved using Bi-CGSTAB [35]. PETSc's default convergence criterion was used, which is five orders of magnitude (10^5) reduction in the residual of the preconditioned system. We used our own codes for problem generation, partitioning, ordering, and symbolic factorization.

Table 4.1 shows incomplete factorization timings for a 3D memory-scaled problem with approximately 91, 125 unknowns per processor. As the number of processors increases, so does the size of the problem. The coefficient matrix of the problem factored on 216 processors has about 19.7 million rows. ILU(2) was employed for the interior nodes, and ILU(1) was employed for the boundary nodes. Reading down any of the columns shows that performance is highly scalable, e.g., for the SGI Origin2000, factorization for 216 processors and 19.7 million unknowns required only 62% longer than the serial case. Scanning horizontally indicates that performance was similar across all platforms, e.g., execution time differed by less than a factor of two between the fastest (Origin2000) and slowest (Beowulf) platforms.

Table 4.2 shows similar data and trends for the triangular solves for the scaled problem. Scalability for the solves was not quite as good as for factorization; e.g., the solve with 216 processors took about 2.5 times longer than the serial case. This is expected due to the lower computation cost relative to communication and synchronization costs in triangular solution.

We observed that the timings for identical repeated runs on the HPC 10000 and SGI typically varied by 50% or more, while repeated runs on the Beowulf were remarkably consistent.

Table 4.3 shows speedup for a constant-sized problem of 1.7 million unknowns. There is a clear correlation between performance and subdomain interior/boundary node ratios; this ratio needs to be reasonably large for good performance.

The performances reported in these tables are applicable to any PDE that has been discretized with a seven-point central difference stencil since the sparsity pattern of the symbolic factor depends on the grid and the stencil only.

4.2. Convergence studies. Our approach for designing parallel ILU algorithms reorders the coefficient matrices whose incomplete factorization is being computed. This reordering could have a significant influence on the effectiveness of the ILU preconditioners. Accordingly, in this section we report the number of iterations of a preconditioned Krylov space solver needed to reduce the residual by a factor of 10^5.

We compare three different algorithms.

TABLE 4.2

Time (sec.) to compute triangular solves for 3D *scaled problem;* 91,125 *unknowns per processor, seven-point stencil, ILU*(2) *factorization on interior nodes, ILU*(1) *factorization on boundary nodes. Dashes (-) for Beowulf and HPC* 10000 *indicate that the machines have insufficient cpus to perform the runs.*

Procs	Origin2000 (AMES)	Beowulf (ICASE)	HPC 10000 (ODU)
1	.182	.187	.289
8	.431	.359	.515
27	.405	.508	.629
64	.472	.556	-
125	.610	-	-
216	.646	-	-

TABLE 4.3

Speedup for 3D *constant-size problem; the grid was* $120 \times 120 \times 120$ *for a total of approximately* 1.7 *million unknowns; data is for ILU*(0) *factorization performed on the SGI Origin*2000*; "I/B ratio" is the ratio of interior to boundary nodes in each subdomain.*

Procs	Unknowns/ Processor	I/B ratio	Time (sec.)	Efficiency (%)
8	216,000	9.3	2.000	100
27	64,000	6.0	0.846	70
64	27,000	4.3	.408	62
125	13,824	3.4	.307	42

- Constrained PILU(k) is the parallel ILU(k) algorithm with the subdomain graph constraint enforced.
- In unconstrained PILU(k), the subdomain graph constraint is dropped, and all fill edges up to level k between the boundary nodes of different subdomains are permitted, even when such edges join two nonadjacent subdomains of the initial subdomain graph $S(A)$.
- In block Jacobi ILU(k) (BJILU(k)), all fill edges joining two different subdomains are excluded.

Intuitively, one expects, especially for diagonally dominant matrices, that larger amounts of fill in preconditioners will reduce the number of iterations required for convergence.

4.2.1. Fill count comparisons. For a given problem, the number of permitted fill edges is a function of three components: the factorization level, k; the subdomain size(s); and the discretization stencil. While the numerical values of the coefficients of a particular PDE influence convergence, they do not affect fill counts. Therefore, our first set of results consists of fill count comparisons for problems discretized on a $64 \times 64 \times 64$ grid using a standard, seven-point stencil.

Table 4.4 shows fill count comparisons between unconstrained PILU(k), constrained PILU(k), and block Jacobi ILU(k) for various partitionings and factorization levels. The data shows that more fill is discarded as the factorization level increases, and as subdomain size (the number of nodes in each subdomain) decreases. These two effects hold for both constrained PILU(k) and block Jacobi ILU(k) but are much more pronounced for the latter. For example, less than 5% of fill is discarded from unconstrained PILU(k) factors when subdomains contain at least 512 nodes (so that the

TABLE 4.4

Fill comparisons for the $64 \times 64 \times 64$ grid. U denotes unconstrained, C denotes constrained, and B denotes block Jacobi $ILU(k)$ preconditioners. The columns headed "nzF/nzA" show the ratio of the number of nonzeros in the preconditioner to the number of nonzeros in the original problem and are indicative of storage requirements. The columns headed "constraint effects" present another view of the same data: here, the percentage of nonzeros in the constrained $PILU(k)$ and block Jacobi $ILU(k)$ factors are shown relative to that for the unconstrained $PILU(k)$. These columns show the amount of fill dropped due to the subdomain graph constraint.

Nodes per subdom.	Subdom. count	Level	**nzF/nzA** U	C	B	**Constraint effects (%)** C	B
262,144	1	0	1.00	1.00	1.00	100.00	100.00
		1	1.84	1.84	1.84	100.00	100.00
		2	3.22	3.22	3.22	100.00	100.00
		3	5.96	5.96	5.96	100.00	100.00
		4	9.73	9.73	9.73	100.00	100.00
32,768	8	0	1.00	1.00	0.99	100.00	98.64
		1	1.87	1.87	1.80	99.99	96.53
		2	3.36	3.35	3.12	99.96	92.91
		3	6.32	6.32	5.70	99.92	90.13
		4	10.50	10.49	9.19	99.89	87.56
4,096	64	0	1.00	1.00	0.96	100.00	95.93
		1	1.89	1.89	1.72	99.90	91.24
		2	3.45	3.44	2.91	99.62	84.36
		3	6.51	6.47	5.19	99.34	79.72
		4	10.81	10.70	8.17	99.06	75.61
512	512	0	1.00	1.00	0.90	100.00	90.50
		1	1.92	1.91	1.57	99.46	81.62
		2	3.59	3.52	2.53	98.05	70.35
		3	6.72	6.50	4.27	96.62	63.47
		4	10.96	10.43	6.32	95.20	57.69
64	4,096	0	1.00	1.00	0.80	100.00	79.64
		1	1.97	1.92	1.29	97.58	65.15
		2	3.73	3.42	1.86	91.67	49.79
		3	6.60	5.64	2.71	85.37	41.04
		4	10.01	7.76	3.35	77.56	33.45
8	32,768	0	1.00	1.00	0.58	100.00	57.92
		1	2.05	1.85	0.80	90.07	38.81
		2	3.98	2.55	0.87	64.14	21.84
		3	6.15	2.89	0.90	46.95	14.72
		4	7.40	2.90	0.90	39.26	12.23

subgraphs on each processor are not too small), but up to 42% is discarded from block Jacobi factors. Thus, one might tentatively speculate that, for a given subdomain size and level, PILU(k) will provide more effective preconditioning than BJILU(k). We have observed similar behavior for 2D problems also. For both 2D and 3D problems, when there is a single subdomain the factors returned by the three algorithms are identical. For the single subdomain case, the ordering we have used corresponds to the natural ordering for these model problems.

An important observation to make in Table 4.4 is how the sizes (number of nonzeros) of the preconditioners depend on levels of fill. For the 3D problems considered here (cube with 64 points on each side, seven-point stencil), a level one preconditioner typically requires twice as much storage as the coefficient matrix A; when the level is two, this ratio is about three; when the level is three, it is about six; and when the level is four, it is about ten. For 2D problems (square grid with 256 points on a side,

Table 4.5

Iteration comparisons for the $64 \times 64 \times 64$ grid. U denotes unconstrained, C denotes constrained, and B denotes block Jacobi ILU(k) preconditioners. The starred entries (*) indicate that, since there is a single subdomain, the factor is structurally and numerically identical to the unconstrained PILU(k). Dashed entries (-) indicate the solutions either diverged or failed to converge after 200 iterations. For Problem 2, when $\varepsilon = 1/500$ the level zero preconditioners did not reduce the relative error in the solution by a factor of 10^5 at termination; when $\varepsilon = 1/1000$, the level one preconditioners did not do so either.

Nodes per subdom.	Subdom. count	Level	Problem 1 U	Problem 1 C	Problem 1 B	Problem 2 $\epsilon = 1/500$ U	Problem 2 $\epsilon = 1/500$ C	Problem 2 $\epsilon = 1/500$ B	Problem 2 $\epsilon = 1/1000$ U	Problem 2 $\epsilon = 1/1000$ C	Problem 2 $\epsilon = 1/1000$ B
262,144	1	0	43	*	*	19	*	*	-	*	*
		1	29	*	*	16	*	*	30	*	*
		2	24	*	*	8	*	*	32	*	*
		3	19	*	*	8	*	*	14	*	*
		4	16	*	*	6	*	*	8	*	*
32,768	8	0	45	45	53	32	32	26	-	-	-
		1	32	33	41	14	14	19	38	39	41
		2	27	29	37	11	11	17	38	38	66
		3	22	24	33	8	8	13	16	15	21
		4	19	21	29	7	7	13	10	11	18
4,096	64	0	43	43	55	33	33	49	-	-	-
		1	31	32	45	15	15	21	42	41	46
		2	25	27	41	12	11	22	24	28	78
		3	20	23	39	9	9	16	18	17	28
		4	17	20	36	8	8	19	11	12	27
512	512	0	41	41	56	28	28	67	-	-	-
		1	29	31	48	18	16	29	39	40	111
		2	25	26	46	11	12	36	21	21	106
		3	21	23	44	11	11	31	20	21	110
		4	18	21	43	9	12	34	13	14	70
64	4,096	0	43	43	64	28	28	-	63	63	-
		1	30	33	60	17	18	124	55	56	-
		2	26	30	58	13	15	115	25	28	-
		3	21	28	58	12	17	127	24	36	-
		4	17	28	58	10	17	132	11	27	-
8	32,768	0	46	46	83	43	43	-	83	83	-
		1	32	41	82	24	46	-	152	-	-
		2	25	40	82	11	45	-	13	115	-
		3	19	40	82	5	44	-	7	107	-
		4	16	40	82	4	45	-	6	111	-

five-point stencil), the growth of fill with level is slower; the ratios are about 1.4 for level one, 1.8 for level two, 2.6 for level three, 3.5 for level four, 4.3 for level five, and 5.4 for level six.

In parallel computation fill levels higher than those employed in sequential computing are feasible since modern multiprocessors are either clusters or have virtual shared memory, and these have memory sizes that increase with the number of processors. Another point to note is that the added memory requirement for these level values is not as prohibitive as it is for a complete factorization. Hence it is practical to trade-off increased storage in preconditioners for reducing the number of iterations in the solver.

4.2.2. Convergence of preconditioned iterative solvers. The fill results in the previous subsection are not influenced by the actual numerical values of the

nonzero coefficients; however, the convergence of preconditioned Krylov space solvers is influenced by the numerical values. Accordingly, Table 4.5 shows iterations required for convergence for various partitionings and fill levels for the three variant algorithms that we consider. The data in these tables can be interpreted in various ways; we begin by discussing two ways that we think are primarily significant.

First, by scanning vertically one can see how changing the number of subdomains, and hence, matrix ordering, affects convergence. The basis for comparison is the iteration count when there is a single subdomain. The partitioning and ordering for these cases is identical to, and our data in close agreement with, that reported by Benzi, Joubert, and Mateescu [4] for natural ordering. (They report results for Problem 2 with $\varepsilon = 1/500$ but not for $\varepsilon = 1/1000$.)

A pleasing property of both the constrained and unconstrained PILU algorithms is that the number of iterations increases only mildly when we increase the number of subdomains from one to 512 for these problems. This insensitivity to the number of subdomains when the number of nodes per subdomain is not too small confirms that the PILU algorithms enjoy the property of parallel algorithmic scalability. For example, Poisson's equation (Problem 1) preconditioned with a level two factorization and a single subdomain required 24 iterations. Preconditioning with the same level, constrained PILU(k) on 512 subdomains needed only two more iterations. Similar results are observed for the convection-diffusion problems also. This property is a consequence of the fill between the subdomains that is included in the PILU algorithm. Similar results have been reported in [26, 36], and the first paper includes a condition number analysis supporting this observation.

Increasing the level of fill generally has the beneficial effect of reducing the number of iterations needed; this influence is largest for the worse-conditioned convection-diffusion problem with $\varepsilon = 1/1000$. For this problem, level zero preconditioners do not converge for reasonable subdomain sizes. Also, even though level one preconditioners require fewer iteration numbers than level two preconditioners in some cases, when the PETSc solvers terminate because the residual norms are reduced by 10^5, the relative errors are larger than 10^{-5} for the former preconditioners. The relative errors are also large for the convection-diffusion problem with $\varepsilon = 1/500$ when the level is set to zero.

Second, scanning the data in Table 4.5 horizontally permits evaluation of the subdomain graph constraint's effects. Again, unless subdomains are small and the factorization level is high; constrained and unconstrained PILU(k) show very similar behavior. Consider, for example, Poisson's equation (Problem 1) preconditioned with a level two factorization and 512 subdomains. The solution with unconstrained PILU(k) required 25 iterations while constrained PILU(k) required 26.

We also see that PILU(k) preconditioning is more effective than BJILU(k) for all 3D trials. (Recall that the single apparent exception, Problem 2, $\varepsilon = 1/500$, ILU(0) with 32,768 nodes per subdomain, has large relative errors at termination.) Again, the extremes of convergence behavior are seen for Problem 2 with $\varepsilon = 1/1000$. Here, with level one preconditioners, BJILU(k) suffers large relative errors at termination while the other two algorithms do not, when the number of subdomains is 64 or fewer.

On 2D domains, while PILU(k) is more effective than BJILU(k) for Poisson's equation, BJILU(k) is sometimes more effective in the convection-diffusion problems.

We also examine iteration counts as a function of preconditioner size graphically. A plot of this data appears in Figure 4.1. In these figures the performance of the

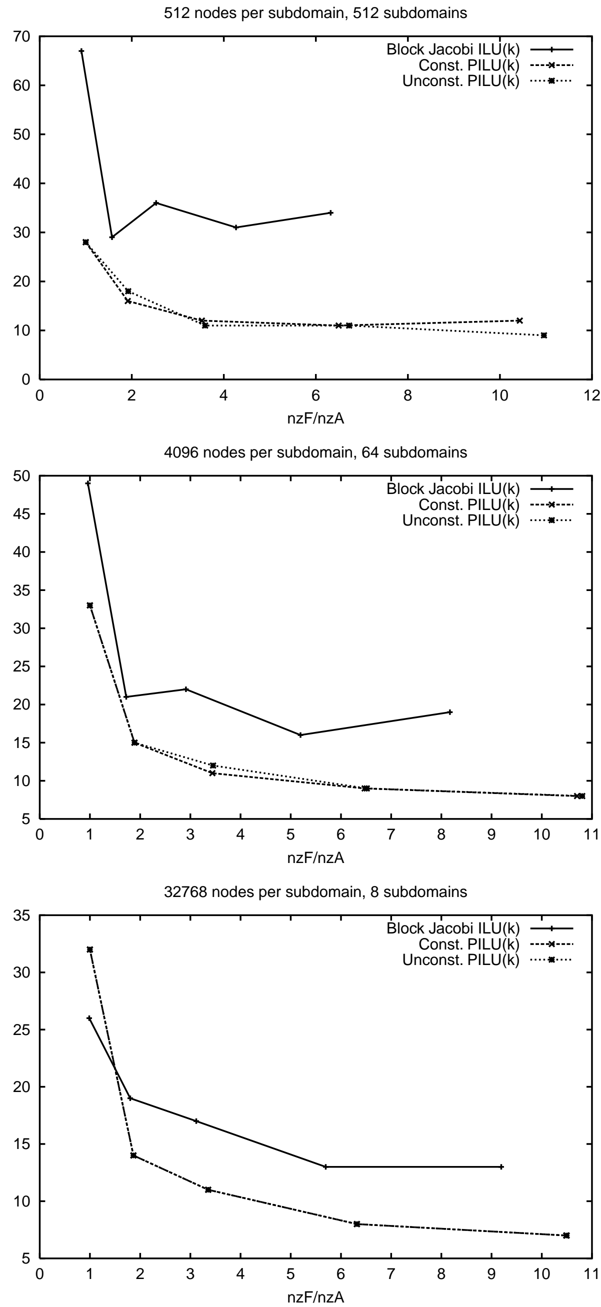

Fig. 4.1. *Convergence comparison as a function of preconditioner size for the convection-diffusion problem, $\epsilon = 1/500$ on the $64 \times 64 \times 64$ grid. Data points are for levels* 0 *through* 4. *Data points for constrained and unconstrained PILU(k) are indistinguishable in the third graph.*

constrained and unconstrained PILU algorithms is often indistinguishable. We find again that PILU(k) preconditioning is more effective than BJILU(k) for 3D problems for a given preconditioner size; however, this conclusion does not always hold for 2D

problems, especially for lower fill levels. As the number of vertices in the subdomains increases, higher fill levels become more effective in reducing the number of iterations needed for convergence. We find that fill levels as high as four to six can be the most effective when the subdomains are sufficiently large. Fill levels higher than these do not seem to be merited by these problems, even for the difficult convection-diffusion problems with $\varepsilon = 1/1000$, where a level four preconditioner reduces the number of iterations below ten.

5. Conclusions. We have designed and implemented a PILU algorithm, a scalable parallel algorithm for computing ILU preconditioners that creates concurrency by means of graph partitioning. The theoretical basis of the algorithm is the incomplete fill path theorem that statically characterizes fill elements in an incomplete factorization in terms of paths in the adjacency graph of the initial coefficient matrix. To obtain a scalable parallel algorithm, we employ a subdomain graph constraint that excludes fill between subgraphs that are not adjacent in the adjacency graph of the initial matrix. We show that the PILU algorithm is scalable by an analysis for 2D- and 3D-model problems and by computational results from parallel implementations on three parallel computing platforms.

We also study the convergence behavior of preconditioned Krylov solvers with preconditioners computed by the PILU algorithm. The results show that fill levels higher than one are effective in reducing the number of iterations, that the number of iterations is insensitive to the number of subdomains, and that the subdomain graph constraint does not affect the number of iterations needed for convergence while it makes possible the design of a scalable parallel algorithm.

Appendix. Proof of the incomplete fill path theorem.

THEOREM A.1. *Let $F = \hat{L} + \hat{U} - I$ be the filled matrix corresponding to an incomplete factorization of A, and let f_{ij} be a nonzero entry in F. Then f_{ij} is a level k entry if and only if there exists a shortest fill path of length $k+1$ that joins i and j in $G(A)$.*

Proof. If there is a shortest fill path of length $k+1$ joining i and j, we prove that the edge exists by induction on the length of the fill path.

Define a *chord* of a path to be an edge that joins two nonconsecutive vertices on the path. The fill path joining i and j is chordless, since a chord would lead to a shorter fill path.

The base case $k = 0$ is immediate, since a fill path of length one in the graph $G(A)$ is an edge $\{i, j\}$ in $G(A)$ that corresponds to an original nonzero in A.

Now assume that the result is true for all lengths less than $k + 1$. Let h denote the highest numbered interior vertex on the fill path joining i and j.

We claim that the (i, h) section of this path is a shortest fill path in $G(A)$ joining i and h. This section is a fill path by the choice of h since all intermediate vertices on this section are numbered lower than h. If there were a fill path joining i and h that is shorter than the (i, h) section, then we would be able to concatenate it with the (h, j) section to form a shorter (i, j) fill path. Hence the (i, h) section is a shortest fill path joining i and h. Similarly, the (h, j) section of this path is the shortest fill path joining h and j.

Each of these sections has fewer than $k + 1$ edges, and hence the inductive hypothesis applies. Denote the number of edges in the (i, h) $((h, j))$ section of this path by k_1 (k_2), where $k_1 + k_2 = k + 1$. By the inductive hypothesis, the edge $\{i, h\}$ is a fill edge of level $k_1 - 1$, and the edge $\{h, j\}$ is a fill edge of level $k_2 - 1$. Now by the sum rule for updating fill levels, when the vertex h is eliminated, we have a fill edge

$\{i,j\}$ of level

$$(k_1 - 1) + (k_2 - 1) + 1 = (k_1 + k_2) - 1 = (k + 1) - 1 = k.$$

Now we prove the converse. Suppose that $\{i,j\}$ is a fill edge of level k; we show that there is a fill path in $G(A)$ of length $k+1$ edges by induction on the level k.

The base case $k = 0$ is immediate, since the edge $\{i,j\}$ constitutes a trivial fill path of length one. Assume that the result is true for all fill levels less than k. Let h be a vertex whose elimination creates the fill edge $\{i,j\}$ of level k. Let the edge $\{i,h\}$ have level k_1, and let the edge $\{h,j\}$ have level k_2; by the sum rule for computing levels, we have that $k_1 + k_2 + 1 = k$. By the inductive hypothesis, there is a shortest fill path of length $k_1 + 1$ joining i and h, and such a path of length $k_2 + 1$ joining h and j. Concatenating these paths, we find a fill path joining i and j of length

$$(k_1 + 1) + (k_2 + 1) = k_1 + k_2 + 2 = k + 1.$$

We need to prove that the (i,j) fill path in the previous paragraph is a shortest fill path between i and j. Consider the elimination of any another vertex g that causes the fill edge $\{i,j\}$. By the choice of the vertex h, if the level of the edge $\{i,g\}$ is k_1' and that of $\{g,j\}$ is k_2', then $k_1' + k_2' + 1 \geq k$. The inductive hypothesis applies to the (i,g) and (g,j) sections, and hence the sum of their lengths is at least $k+1$.

This completes the proof. ∎

This result is a generalization of the following theorem that characterizes fill in complete factorizations for direct methods, due to Rose and Tarjan [30].

THEOREM A.2. *Let $F = L + U - I$ be the filled matrix corresponding to the complete factorization of A. Then $f_{ij} \neq 0$ if and only if there exists a fill path joining i and j in the graph $G(A)$.*

Here we associate level values with each fill edge and relate it to the length of shortest fill paths. The incomplete fill path theorem enables new algorithms for incomplete symbolic factorization that are more efficient than the conventional algorithm that simulates numerical factorization. We have described these algorithms in an earlier work [29] and the report is in preparation.

D'Azevedo, Forsyth, and Tang [9] have defined the (sum) level of a fill edge $\{i,j\}$ using the length criterion employed here, and hence they were aware of this result. However, the theorem is neither stated nor proved in their paper. Definitions of level that compute levels of fill nonzeros by rules other than by summing the levels of the causative pairs of nonzeros have been used in the literature. The "maximum" rule defines the level of a fill nonzero to be the minimum over all causative pairs of the maximum value of the levels of the causative entries:

$$\text{level}(f_{ij}) = \min_{1 \leq h \leq \min\{i,j\}} \max\{\text{level}(f_{ih}), \text{level}(f_{hj})\} + 1.$$

A variant of the incomplete fill path theorem can be proved for this case, but it is not as simple or elegant as the one for the "sum" rule. Further discussion of these issues will be deferred to a future report.

Acknowledgments. We thank Dr. Edmond Chow of CASC, Lawrence Livermore National Laboratory, and Professor Michele Benzi of Emory University for helpful discussions.

REFERENCES

[1] O. Axelsson, *Iterative Solution Methods*, Cambridge University Press, Cambridge, UK, 1994.

[2] S. Balay, W. D. Gropp, L. Curfman McInnes, and B. F. Smith, *PETSc home page*, http://www.mcs.anl.gov/petsc, 1999.

[3] P. Bastian and G. Horton, *Parallelization of robust multigrid methods: ILU factorization and frequency decomposition method*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1457–1470.

[4] M. Benzi, W. Joubert, and G. Mateescu, *Numerical experiments with parallel orderings for ILU preconditioners*, Electron. Trans. Numer. Anal., 8 (1999), pp. 88–114.

[5] A. M. Bruaset and H. P. Langtangen, *Object-oriented design of preconditioned iterative methods in Diffpack*, ACM Trans. Math. Software, 23 (1997), pp. 50–80.

[6] T. F. Chan and H. A. van der Vorst, *Approximate and incomplete factorizations*, in Parallel Numerical Algorithms, ICASE/LaRC Interdiscip. Ser. Sci. Engrg. 4, D. E. Keyes, A. Sameh, and V. Venkatakrishnan, eds., Kluwer Academic, Dordecht, The Netherlands, 1997, pp. 167–202.

[7] E. Chow and M. A. Heroux, *An object-oriented framework for block preconditioning*, ACM Trans. Math. Software, 24 (1998), pp. 159–183.

[8] E. Chow and Y. Saad, *Experimental study of ILU preconditioners of indefinite matrices*, J. Comput. Appl. Math, 86 (1997), pp. 387–414.

[9] E. F. D'Azevedo, P. A. Forsyth, and W.-P. Tang, *Ordering methods for preconditioned conjugate gradient methods applied to unstructured grid problems*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 944–961.

[10] S. Doi and A. Lichnewsky, *A Graph-Theory Approach for Analyzing the Effects of Ordering on ILU Preconditioning*, Tech. report 1452, Institut National de Recherche in Informatique et en Automatique, Rocqhencourt, BP105-78153, Le Chesnay Cedex, France, 1991.

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

[12] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst, *Numerical Linear Algebra for High Performance Computers*, SIAM, Philadelphia, 1998.

[13] I. S. Duff and G. A. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1983), pp. 635–657.

[14] V. Eijkhout, *Analysis of parallel incomplete point factorizations*, Linear Algebra Appl., 154–156 (1991), pp. 723–740.

[15] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, 1997.

[16] G. Haase, *Parallel incomplete Cholesky preconditioners based on the nonoverlapping data distribution*, Parallel Comput., 24 (1998), pp. 1685–1703.

[17] W. Hackbusch and G. Wittum, eds., *Incomplete Decomposition (ILU): Algorithms, Theory, and Applications*, Notes Numer. Fluid Mech. 41, Vieweg, Braunschweig, Wiesbaden, 1993.

[18] D. Hysom and A. Pothen, *Efficient parallel computation of ILU(k) preconditioners*, in Proceedings of Supercomputing 99, ACM, New York, 1999, published on CDROM.

[19] D. Hysom and A. Pothen, *Efficient Parallel Computation of ILU(k) Preconditioners*, Tech. report 2000-23, ICASE, NASA Langley Research Center, Hampton, VA, 2000.

[20] D. Hysom and A. Pothen, *Parallel ILU Ordering and Convergence Relationships: Numerical Experiments*, Tech. report 2000-24, ICASE, NASA Langley Research Center, Hampton, VA, 2000.

[21] M. T. Jones and P. E. Plassmann, *An improved incomplete Cholesky factorization*, ACM Trans. Math. Software, 21 (1995), pp. 5–17.

[22] G. Karypis and V. Kumar, *Parallel threshold-based ILU factorization*, in Proceedings of the ACM Conference on Supercomputing, San Jose, CA, 1997, CD-ROM, ACM, New York, 1997.

[23] C.-J. Lin and J. J. Moré, *Incomplete Cholesky factorizations with limited memory*, SIAM J. Sci. Comput., 21 (1999), pp. 24–45.

[24] S. Ma and Y. Saad, *Distributed ILU*(0) *and SOR Preconditioners for Unstructured Sparse Linear Systems*, Tech. report 94-27, Army High Performance Computing Research Center, University of Minnesota, Minneapolis, MN, 1994.

[25] M. Magolu monga Made and H. A. van der Vorst, *Parallel incomplete factorizations with pseudo-overlapped subdomains*, Parallel Comput., to appear.

[26] M. Magolu monga Made and H. A. van der Vorst, *Spectral analysis of parallel incomplete factorizations with implicit pseudo-overlap*, Numer. Linear Algebra Appl., to appear.

[27] T. A. Manteuffel, *An incomplete factorization technique for positive definite linear systems*, Math. Comput., 34 (1980), pp. 307–327.
[28] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear equation systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comput., 31 (1977), pp. 148–162.
[29] A. Pothen and D. Hysom, *Fast algorithms for incomplete factorization*, in Proceedings of the Copper Mountain Conference on Iterative Methods, Copper Mountain, CO, 1998, report in preparation.
[30] D. J. Rose and R. E. Tarjan, *Algorithmic aspects of vertex elimination on directed graphs*, SIAM J. Appl. Math., 34 (1978), pp. 176–197.
[31] Y. Saad, *Sparskit, version* 2, http://www-users.cs.umn.edu/ saad/software.html, 1990, 1994.
[32] Y. Saad, *ILUT: A dual-threshold incomplete LU factorization*, Numer. Linear Algebra Appl., 1 (1994), pp. 387–402.
[33] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing, Boston, 1996.
[34] H. A. van der Vorst, *High performance preconditioning*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1174–1185.
[35] H. A. van der Vorst, *Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 13 (1992), pp. 631–634.
[36] C. Vuik, R. R. P. van Nooyen, and P. Wesseling, *Parallelism in ILU-preconditioned GMRES*, Parallel Comput., (1998), pp. 1927–1946.