NASA/CR-2000-210119
ICASE Report No. 2000-24

Parallel ILU Ordering and Convergence Relationships: Numerical Experiments

David Hysom and Alex Pothen
Old Dominion University, Norfolk, Virginia

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA

Operated by Universities Space Research Association

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-97046

May 2000

DTIC QUALITY INSPECTED 2

20000512 026

PARALLEL ILU ORDERING AND CONVERGENCE RELATIONSHIPS: NUMERICAL EXPERIMENTS

DAVID HYSOM AND ALEX POTHEN*

Abstract. We recently developed a parallel algorithm for computing ILU preconditioners, which was presented at Super Computing 1999. The algorithm has been shown to be highly scalable, in terms of execution time required for preconditioner factorization and application, for problems with up to 20 million unknowns running on up to 216 processors. However, since the algorithm reorders the matrix, and it is widely known that ordering can significantly affect convergence, questions were raised concerning the quality of the computed preconditioners. In this report we present experimental results demonstrating that the orderings imposed by the algorithm do not significantly degrade convergence, as long as the number of unknowns per subdomain is not too small. We report on two model problems, Poisson's equation, and a special case of the convection-diffusion equation, which other researchers have used for ordering and convergence studies. We show that convergence behavior is fairly flat as long as subdomains contain at least 512 nodes.

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

Subject classification. Computer Science

1. Introduction. The Parallel ILU algorithm, presented at SC99 [5] [1], attempts to maximize parallelism through use of a two-phase ordering technique coupled with a *Subdomain Graph Constraint.* To describe the ordering and constraint we require a few concepts from graph theory. Given an $n \times n$ matrix A partitioned into subdomains, the graph $G(A) = (V, E)$ has vertex set V, which contains a node i for every row in the matrix, and edge set E, which contains a directed edge (i, j) for every nonzero matrix entry a_{ij}. Node i is called an *interior node*, and the associated matrix row an *interior row*, if, for every edge $(i, j) \in E, 1 \leq j \leq n$, nodes i and j are in the same subdomain. Node i is called a *boundary node*, and the associated matrix row a *boundary row*, if there is at least one edge $(i, j) \in E$ such that nodes i and j are in different subdomains. A *Subdomain Graph* $S(A) = (V^S, E^S)$ is a reduced graph of A, such that nodes in V^S correspond to subdomains, and E^S contains an edge (r, s) if subdomains r and s are connected by one or more edges in $G(A)$. An incomplete factorization of A is denoted as $F = \hat{L} + \hat{U} - I$, where $\hat{L}\hat{U} \approx A$.

During the first ordering phase Parallel ILU constructs and orders the Subdomain Graph, S(A). Since we stipulate that nodes within each subdomain be numbered contiguously, this ordering imposes a partial ordering on $G(A)$, i.e., if node i is ordered before node j in S(A), then all nodes in subdomain i are ordered before all nodes in subdomain j in $G(A)$.

During the second ordering phase nodes are ordered within individual subdomains. The ordering requires that all interior nodes be ordered before the boundary nodes in each subdomain. However, within a subdomain the relative ordering of interior nodes (and similarly boundary nodes) is not specified. A subdo-

* Old Dominion University, Norfolk, VA (email: hysom,pothen@cs.odu.edu). 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 the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-2199.

[1] Large scale results, which do not appear in the published paper, can be viewed at http://www.cs.odu.edu/~hysom, and are also being published in ICASE Report No. 2000-23 (http://www.icase.edu).

main's interior nodes may be ordered, for example, using reverse Cuthill-McKee (RCM), Nested Dissection, etc. Further, different interior orderings can be used in different subdomains.

During the next step of the algorithm the matrix is factored using a row-oriented ILU algorithm, subject to the *Subdomain Graph Constraint*, which decrees that the Subdomain Graph of A and the factor, $F = \hat{L} + \hat{U} - I$, be identical, i.e, $S(A) = G(A)$. This constraint is trivially fulfilled for ILU(0) factorization. When additional fill is permitted, however, it is possible for the subdomain graphs $S(A)$ and $S(F)$ to differ. The Subdomain Graph Constraint can result in the deletion and/or numerical alteration of entries that would otherwise be included in F.

Parallel ILU's factorization and solve stages have been shown to be highly scalable [5]. However, three questions concerning convergence behavior have been raised.

1. How do the orderings imposed by Parallel ILU affect convergence?
2. How does the Subdomain Graph Constraint affect convergence?
3. How does convergence for Parallel ILU compare with that for Block Jacobi ILU? (By Block Jacobi ILU, we refer to ILU factorization of diagonally blocked matrices.)

The first question is important since previous studies [2, 3] have shown that matrix ordering can significantly affect convergence. The second question is prompted by the intuition that "throwing out" entries in F will likely weaken the preconditioner. The final question is important since Parallel ILU has communication costs absent from Block Jacobi ILU; hence, if Parallel ILU does not require significantly fewer iterations than Block Jacobi ILU, the communication overhead may result in Parallel ILU being costlier in terms of total execution time.

We investigated these questions by conducting numerical experiments using three ILU variants: Parallel ILU(k) with and without the Subdomain Graph Constraint, and Block Jacobi ILU(k), We refer to these methods respectively as Constrained PILU(k), Unconstrained PILU(k), and BJILU(k).

The experiments used two model problems, Poisson's equation, and a convection-diffusion equation. Our scenario is that the linear systems that must be solved, of which these are representative if simplistic examples, arise during execution of Grand Challenge application codes, and there is a one-to-one mapping between subdomains and processors. Results indicate that, for realistic numbers of unknowns per subdomain (between 512 and 32K for 3D problems) the following behaviors are typical.

1. The orderings imposed by Parallel ILU affect convergence only to a relatively small degree.
2. Iteration counts are quite close for Constrained and Unconstrained PILU(k), indicating that the Subdomain Graph Constraint does not significantly affect convergence.
3. Constrained PILU(k) converges in significantly fewer iterations than does BJILU(k).

Section 3 contains extended results, including discussion of exceptional and 2D cases.

2. Algorithms. In this section we discuss details of the Parallel ILU algorithm, and summarize the underlying theory. Figure 2.1 shows Parallel ILU in a form suitable for message passing computational environments. Figure 2.2 shows the equivalent serial formulation that was used for this study (*equivalent* here means the factors computed by both algorithms are identical). Although not all terminology and concepts have yet been fully defined or explained, we think it useful to present the algorithms at the outset, for the reader's ready reference.

In the following subsections we discuss three notions fundamental to Parallel ILU's design: the global matrix ordering phase, which is based on ordering the subdomain graph; the local matrix ordering phase, which arises from application of the Fill Path Theorem; and the Subdomain Graph Constraint, which is based on a recent extension of the Fill Path Theorem. Each of these notions maps directly to one of the

Input: A partitioned, distributed matrix.

1. Form subdomain graph and order vertices to reduce directed path lengths, using vertex coloring.
2. On each processor, locally order interior nodes, then order boundary nodes.
3. Factor interior rows. Processors having no lower-ordered neighbors in the subdomain graph also factor boundary rows, and go to step 6.
4. Receive row structure and values of boundary rows from lower-ordered neighbors in the subdomain graph.
5. Factor boundary rows.
6. Send boundary row structures and values to higher-ordered neighbors in the subdomain graph.

FIG. 2.1. *Parallel ILU*

Input: A partitioned matrix.

1. Form subdomain graph $S(A)$ and order vertices to reduce directed path lengths using vertex coloring.
2. For each subdomain, order interior nodes, then order boundary nodes.
3. Perform ILU factorization subject to the constraint that fill entry f_{ij} is disallowed if node i is in subdomain r, node j is in subdomain s, and edge (r, s) is not in the edgeset of $S(A)$.

FIG. 2.2. *Equivalent Sequential Version of PILU*

three steps in the serial formulation (Figure 2.2). Finally, we present a scalability analysis.

2.1. Subdomain Graphs and PILU's Global Ordering Phase. The global ordering phase, which is the first step in PILU, is intended to reduce directed path lengths in the subdomain graph. Since nodes in each subdomain are ordered contiguously, $S(A)$ is considered to be a directed graph, with edges oriented from lower to higher numbered vertices. Consequently, saying "subdomain r is ordered before subdomain s" is equivalent to saying "all nodes in subdomain r are ordered, then all nodes in subdomain s are ordered."

We assume that factorization is row-oriented and upward-looking. Accordingly, the initial matrix A is mapped to subdomains (i.e., processors) by rows. Edges in $S(A)$ thus indicate data dependencies that, in parallel environments, become communication dependencies. Hence, ordering $S(A)$ to reduce directed path lengths has the effect of decreasing serial communication bottlenecks.

It is natural to ask, "how much parallelism can be gained through subdomain graph reordering?" While this is in general very difficult to answer, we can gain some intuition through analysis of simplified model problems. Consider a matrix arising from a 2nd order partial differential equation (PDE) that has been discretized on a regularly structured 2D grid using a standard 5-point stencil. Assume the grid has been partitioned into m square subgrids (m being a perfect square). 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 $m - 1$. However, regular 2D and 3D grids can easily be divided into two independent sets, commonly referred to as red-black coloring. If this is done and all red nodes are numbered, then all black nodes numbered, the longest dependency path in S will be reduced in length from $m - 1$ to 1.

In general, our approach is expected to work well with any problem that can be partitioned such that

the resulting subdomain graph has a small chromatic number.

2.2. The Fill Path Theorem and PILU's Local Ordering Phase. The local ordering phase, PILU's second step, is intended to increase the amount of communication independent processing. This step results from application of the *Fill Path Theorem* [6]. Given the graph of an initial matrix A, the theorem provides a static characterization of where fill entries arise during factorization. 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.

DEFINITION 2.1. *A* fill path *is a path joining two vertices i and j, all of whose interior vertices are numbered lower than the minimum of the numbers of i and j.* [2]

THEOREM 2.2. *Let $F = L + U - I$ be the complete factor of A. Then $f_{ij} \neq 0$ if and only if there exists a fill path joining i and j in the graph $G(A)$.*

Now consider the graph of a partitioned matrix, $G(A)$. If two interior nodes belonging to separate subdomains 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, for reasons that will shortly become apparent. If each subdomain's boundary nodes are ordered after its interior nodes, this situation cannot arise: boundary nodes in $G(A)$ remain boundary nodes in $G(F)$, and interior nodes in $G(A)$ remain interior nodes in $G(F)$. This is so since any path joining interior nodes that are assigned to separate subdomains must include at least two boundary nodes; since each boundary node will be numbered higher than (at least one of) the path's end points, no such fill path can exist.

Since, for upward-looking factorization, matrix rows are only updated from lower ordered rows, the local ordering step, in addition to preserving interior/boundary node ratios, ensures that a subdomain's interior rows can be factored independently of row updates from any other subdomain. Therefore, if all subdomains have relatively large interior/boundary node ratios and contain approximately equal numbers of nodes we expect PILU to exhibit a high degree of parallelism.

2.3. The Extended Fill Path Theorem and PILU's Subdomain Graph Constraint. PILU's Subdomain Graph Constraint arises from a recently developed theorem that extends Theorem 2.2 to provide static characterizations of fill for incomplete, structurally based factorizations (ILU(k)). Before introducing the theorem, we note that the "classic" ILU(k) algorithm operates by mimicking numeric factorization. Classic ILU(k) initially assigns all entries in A the level zero. New entries that arise during factorization are assigned a level based on the levels of the causative entries, according to the rule:

$$level(a_{ij}) = \min_{1 \leq h \leq n} \{level(a_{ih}) + level(a_{hj}) + 1\}, \quad a_{ih}, a_{hj} \in A.$$

The Extended Fill Path Theorem informs us of an intimate relationship between fill entries in Classic ILU(k) factors and path lengths in graphs.

THEOREM 2.3. *Let $F = \hat{L} + \hat{U} - I$ be an incomplete factor of A, and f_{ij} a permitted 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 $G(A)$.*

Theorem 2.3 has an intuitively simple geometric interpretation. Given an initial node i in $G(A)$, construct a topological "sphere" containing all nodes that are connected to i by a path containing $k+1$ or fewer edges. Then a fill entry f_{ij} can only be permitted in an $ILU(k)$ factor if j is within the sphere.

[2] 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 when *interior* is used to refer to nodes in paths, and when to nodes in subdomains.

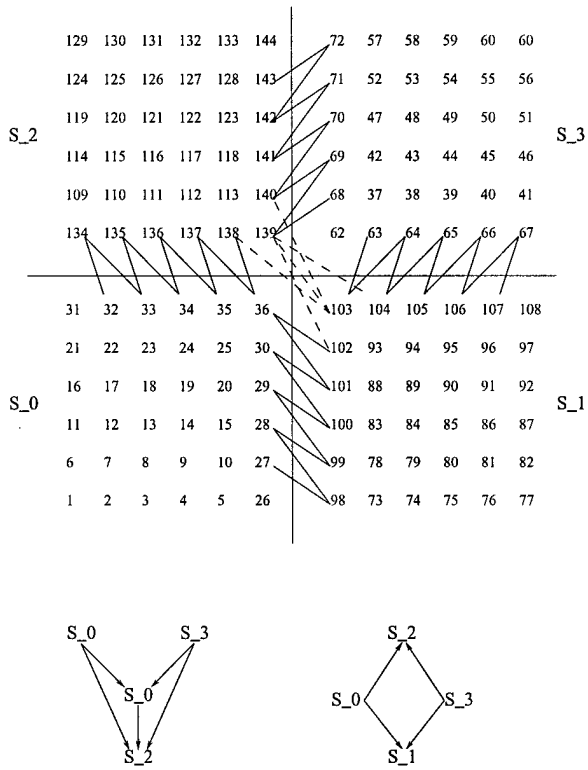

FIG. 2.3. *Illustration of the Subdomain Graph Constraint's action. The Top figure shows level 1 and 2 fill edges that cross subdomain boundaries; for clarity, all other fill is omitted. The dotted edges would alter the subdomain graph, and hence are prohibited. Bottom figure shows (left) that the subdomain graph would have a directed path of length two if the edges were not prohibited and (right) when the edges are discarded, the longest directed path has length one.*

Turning our attention to the Subdomain Graph Constraint we notice that, as a result of fill entries that arise during factorization, subdomain graph $S(F)$ can contain edges not present in subdomain graph $S(A)$. This is illustrated in Figure 2.3. The additional edges can have the effect of increasing dependency path lengths in $S(F)$. The Subdomain Graph Constraint—which states that fill entry f_{ij} is disallowed if node i is in subdomain r, node j is in subdomain s, and edge $\{r, s\}$ is not in the edgeset of $S(A)$—ensures that $S(F)$ remains identical to $S(A)$. This is particularly important since the serial communication dependencies that correspond to directed path lengths occur in both factorization and triangular solve phases.

By applying Theorem 2.3, we can gain an intuitive understanding of the number of fill entries that may be discarded on account of the Subdomain Graph Constraint. Referring again to Figure 2.3, we see that prohibited edges only arise near points where two subdomains that are not neighbors in $S(A)$ "touch corners." Theorem 2.3 tells us that these edges can only arise within a radius of $k + 1$ edges about such points. Hence, if subdomains contain relatively large numbers of nodes and k is relatively small, we expect the Subdomain Graph Constraint will prohibit only a small number of fill edges.

Several researchers have developed parallel ILU algorithms that have similarities to ours; the earliest such work of which we are aware is credited to Saad [7]. His *Distributed ILU(0) Factorization* (Section 12.6.1) is similar to ours in that interior nodes are ordered before boundary nodes, and a coloring of the subdomain graph is used to enhance concurrency in eliminating the boundary nodes. Our work, although developed independently, in effect extends Saad's ideas to show how fill effects can be included. Additionally,

we provide an analysis to show that this approach leaads to parallel algorithmisc scalability. We also report experimental results showing that the number of iterations for model problems is almost entirely independent of the number of subdomains.

2.4. Scalability Analysis. In this section we present a simplified theoretical analysis of algorithmic behavior for matrices arising from PDEs discretized on 3D grids with seven-point stencils. We assume the grid has been block-partitioned, with each subdomain consisting of a cubic subgrid of dimension $c \times c \times c$. We also assume the subdomain grid has dimensions $p^{1/3} \times p^{1/3} \times p^{1/3}$ so there are p processors total. There are thus $N = c^3 p$ total nodes in the grid, and each subdomain contains $c^3 = \frac{N}{p}$ nodes, of which at most $6c^2 = 6(\frac{N}{p})^{2/3}$ are boundary nodes.

If subdomain interior nodes are locally numbered in natural order and $k \ll c$, it can be shown that matrix rows in the factor asymptotically have k^2 (strict) upper and lower triangular entries, and the cost for factoring a row is k^4. For red-black coloring of $S(A)$, with red subdomains numbered before black subdomains, Parallel ILU simplifies to the following three stages, which are assumed non-overlapping.

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.

The cost of the first stage is bounded by the complexity of factoring all rows in a subdomain. This is $k^4 c^3 = k^4 \frac{N}{p}$.

The cost for the second stage is the cost of sending structural and numerical values from the upper-triangular portions of boundary rows to neighboring processors. Assuming a standard, non-contentious communication model wherein α and β represent message startup and cost-per-word respectively, and with time for each operation normalized to unity, the cost for this step is bounded by $6(\alpha + k^2\beta c^2) = 6(\alpha + k^2\beta(\frac{N}{p})^{2/3})$.

For the third step, the cost of factoring a boundary row can be shown asymptotically identical to the cost of factoring an interior row, so the cost here is $6k^4c^2 = 6k^4(\frac{N}{p})^{2/3}$.

Speedup can then be expressed as

$$\text{speedup} = \frac{k^4 N}{k^4 \frac{N}{p} + 6(\alpha + k^2\beta(\frac{N}{p})^{2/3}) + 6k^4(\frac{N}{p})^{2/3}}.$$

The numerator represents cost for uni-processor (sequential) execution, and the three denominator terms represent the costs for the corresponding stages of the simplified algorithm for parallel execution.

3. Results. We report on model problems identical to those used by Benzi et. al, in "Numerical Experiments with Parallel Orderings for ILU Preconditioners." [2].

Problem 1. Poissons's equation,

$$\Delta u = g.$$

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

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

Homogeneous boundary conditions were used for both problems. Derivative terms were discretized on the unit square or cube, using central differencing (5-point and 7-point stencils) on regularly spaced $n_x \times n_y \times n_z$

grids ($n_z = 1$ for 2D). Problem 2 was run using coefficients $\varepsilon = 1/100$ and $\varepsilon = 1/500$. 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.

Both problems were solved using Krylov subspace methods as implemented in the PETSc software library [1]. Problem 1 was solved using Conjugate Gradient [4], and Problem 2 was solved using Bi-CGSTAB [8]. PETSc's default convergence criteria was used, 10^5 reduction in the residual of the preconditioned system. We used our own codes for problem generation, partitioning, ordering, and symbolic factorization.

Since we are here concerned with convergence behavior rather than parallel efficiency, scalability, etc., the experiments were performed in a sequential environment.

3.1. Fill Count Comparisons. Intuitively one expects, especially for diagonally dominant matrices, that larger amounts of fill in preconditioners will reduce the number of iterations required for convergence. As explained in Section 2, the Subdomain Graph Constraint implies that fill edges arising during factorization will be dropped if they join subdomains that are not neighbors in the Subdomain Graph, $S(A)$. Block Jacobi ILU(k) can be considered as resulting from a more severe constraint, one that requires the dropping of all edges that join subdomains. Both constraints have the effect of reducing permitted fill.

For constant-size problems the number of dropped edges (or, contrariwise, the amount of permitted fill) is a function of three components: the factorization level, k; the subdomain size(s); and the discretization stencil. While the numerics 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 7-point stencil.

Table 4.1 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 more pronounced for the latter. For example, less than 5% of fill is discarded from Unconstrained PILU(k) factors when subdomains contain upwards of 512 nodes, 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).

Table 4.2 shows similar data for problems discretized on a 256×256 grid using a standard 5-point stencil. Note that, for both 2D and 3D problems, when there is a single subdomain the factors returned by the three algorithms are identical.

3.2. Convergence Studies. Tables 4.3 and 4.4 show iterations required for convergence for various partitionings and fill levels. The raw data in these tables can be interpreted in various ways; we begin by discussing two we think particularly significant.

First, by scanning vertically one can see how changing the partitioning, hence, matrix ordering, affects convergence. When there is a single subdomain there is no global or local reordering, and our iteration counts are in close agreement with those reported by Benzi, et. al., [2] for natural ordering (NO). Our data shows that, unless subdomain sizes are small (64 or fewer nodes) and the factorization level high, PILU's reordering does not significantly influence convergence. For example (see Table 4.3), 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), and 512 subdomains necessitated only two additional iterations. We conclude that, for subdomain sizes that are likely to arise in real-world applications and systems similar to our model problems, the reorderings imposed by PILU will likely have little effect on convergence.

For an intuitive understanding of convergence behavior, note that, if one performs Parallel ILU on a matrix where each node is considered an entire subdomain, the imposed ordering degenerates to a red-black ordering on $G(A)$. Since at the other extreme a single subdomain corresponds to a natural ordering on $G(A)$, we can consider that orderings imposed by Parallel ILU lie on a continuum bounded by Natural and Red-Black.

Second, scanning the data horizontally permits evaluation of the Subdomain Graph Constraint's effects. Again, unless subdomains are small and the factorization level 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 (again, see Table 4.3). Solution with Unconstrained PILU(k) required 25 iterations while Constrained PILU(k) required 26.

There are two approaches for comparing Parallel ILU(k) and Block Jacobi ILU(k) preconditioning. Horizontal scanning of the data in the tables permits comparison for a given partitioning and factorization level. Doing so for Table 4.3 reveals that, with a single exception (Problem 2, $\epsilon = 1/500$, ILU(0) with 32,768 nodes per subdomain), PILU(k) preconditioning is more effective than BJILU(k) for all 3D trials. The results from the 2D runs, Table 4.4, are less consistent. While PILU(k) is more effective than BJILU(k) for Poisson's equation, BJILU(k) is sometimes more effective in the convection-diffusion problems.

The second approach is to examine iteration counts as a function of preconditioner size. This comparison involves data from both the fill and iteration tables, and is best illustrated graphically. Plots of this data appear in Figures 4.1 through 4.6. As in the first approach, PILU(k) preconditioning is almost always more effective than BJILU(k) for 3D problems, but only sometimes so for 2D.

4. Conclusions. The results presented herein indicate that, at least for simple problems, if the number of nodes per subdomain is not too small or the factorization level too high, the orderings imposed by Parallel ILU(k) have little effect on convergence. These results are for systems wherein subdomain interior nodes are naturally ordered; experiments with other interior orderings have yet to be conducted. The results also show that fill discarded on account of the Subdomain Graph Constraint has little adverse effect on convergence. Finally, the data suggests that Parallel ILU(k) can provide considerably stronger preconditioning than Block Jacobi ILU(k) for 3D problems.

REFERENCES

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

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

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

[4] M. R. Hestenes and E. L. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–436.

[5] D. Hysom and A. Pothen, *Efficient parallel computation of ILU(k) preconditioners*, SC99, ACM, November 1999. published on CDROM, ISBN #1-58113-091-0, ACM Order #415990, IEEE Computer Society Press Order # RS00197.

[6] D. J. Rose and R. E. Tarjan, *Algorithmic aspects of vertex elimination on directed graphs*, SIAM J. Appl. Math., 23 (1978), pp. 176–197.

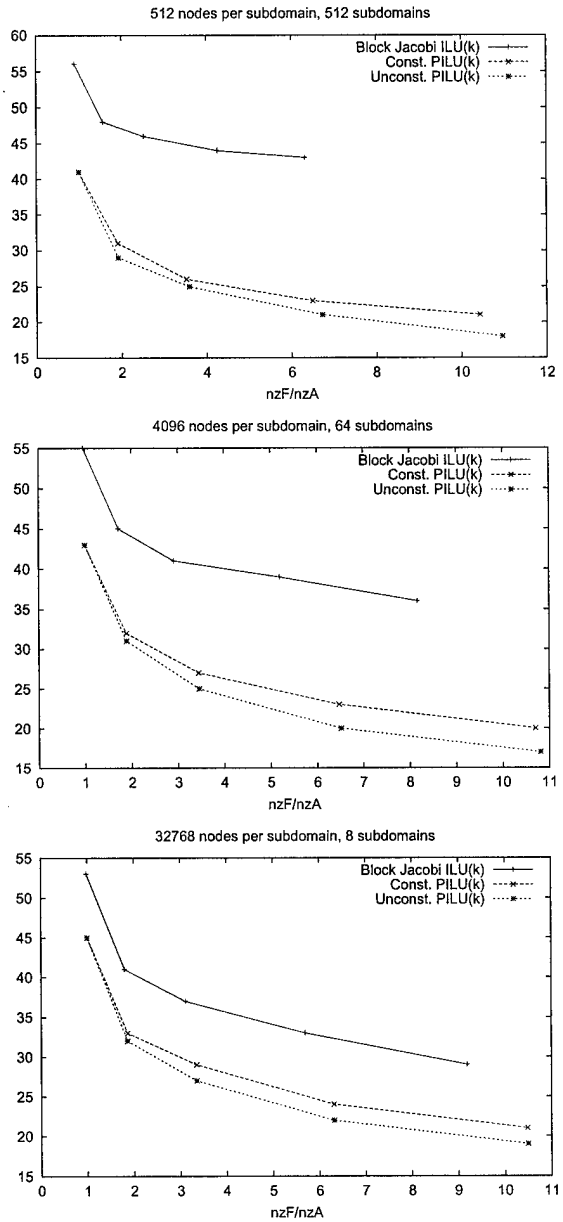

FIG. 4.1. *Convergence comparison as a function of preconditioner size for Poisson's equation on* $64 \times 64 \times 64$ *grid. Data points are for levels 0 through 4.*

[7] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, PWS Publishing Company, 20 Park Plaza, Boston, MA 02116, 1996. ISBN 0-534-94776-X (hardcover).

[8] H. A. VAN DER VORST, *Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 631–634.

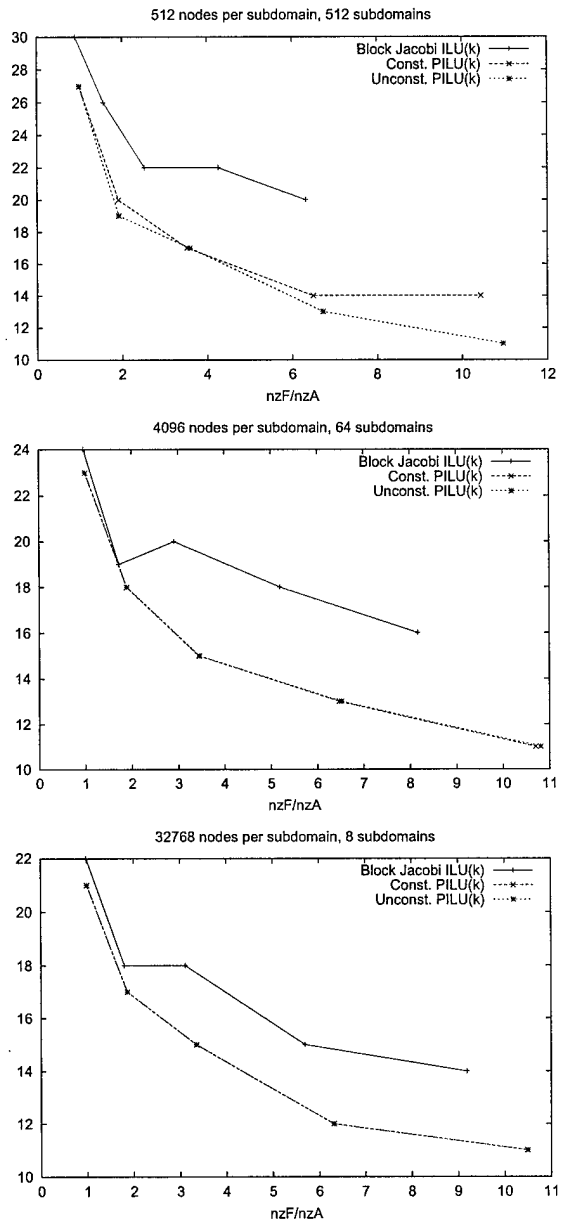

FIG. 4.2. *Convergence comparison as a function of preconditioner size for convection-diffusion problem,* $\epsilon = 1/100$ *on* $64 \times 64 \times 64$ *grid. Data points are for levels 0 through 4. Data points for Constrained and Unconstrained PILU(k) are indistinguishable in the second and third graphs.*

TABLE 4.1

Fill comparisons for $64 \times 64 \times 64$ grid. 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 Unconstrained PILU(k). These columns are indicative of the amount of fill that is dropped due to subdomain graph constraints.

nodes per	subdomain		**nzF/nzA**			**constraint effects (%)**	
subdomain	count	level	uncon	const	bj	const	bj
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

TABLE 4.2

Fill comparisons for 256×256 *grid. The columns headed "nzF/nzA" show preconditioner size relative to problem size, 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 Unconstrained PILU(k). These columns are indicative of the amount of fill that is dropped due to subdomain graph constraints.*

nodes per	subdomain		**nzF/nzA**			**constraint effects (%)**	
subdomain	count	level	uncon	const	bj	const	bj
65,536	1	0	1.00	1.00	1.00	100.00	100.00
		1	1.40	1.40	1.40	100.00	100.00
		2	1.79	1.79	1.79	100.00	100.00
		3	2.59	2.59	2.59	100.00	100.00
		4	3.37	3.37	3.37	100.00	100.00
		5	4.16	4.16	4.16	100.00	100.00
		6	4.94	4.94	4.94	100.00	100.00
16,384	4	0	1.00	1.00	1.00	100.00	99.69
		1	1.40	1.40	1.39	100.00	99.11
		2	1.82	1.82	1.78	100.00	98.13
		3	2.63	2.63	2.56	100.00	97.42
		4	3.45	3.45	3.34	100.00	96.71
		5	4.28	4.28	4.11	99.99	95.99
		6	5.11	5.11	4.87	99.99	95.28
1,024	64	0	1.00	1.00	0.98	100.00	97.81
		1	1.42	1.42	1.35	99.98	95.24
		2	1.88	1.88	1.72	99.92	91.46
		3	2.73	2.73	2.44	99.87	89.08
		4	3.61	3.60	3.13	99.83	86.70
		5	4.50	4.49	3.80	99.78	84.39
		6	5.41	5.39	4.44	99.74	82.14
64	1,024	0	1.00	1.00	0.90	100.00	90.28
		1	1.49	1.48	1.21	99.60	81.46
		2	2.08	2.05	1.47	98.59	70.74
		3	3.02	2.96	1.96	97.76	64.70
		4	3.97	3.85	2.35	96.96	59.28
		5	4.89	4.70	2.66	96.09	54.38
		6	5.77	5.49	2.88	95.15	49.90
16	4,096	0	1.00	1.00	0.80	100.00	80.25
		1	1.55	1.52	1.03	98.43	66.43
		2	2.24	2.12	1.18	94.58	52.53
		3	3.06	2.78	1.40	90.87	45.87
		4	3.74	3.14	1.48	84.07	39.60
		5	4.26	3.30	1.48	77.48	34.72
		6	4.64	3.38	1.48	72.76	31.89

TABLE 4.3

Iteration comparisons for $64 \times 64 \times 64$ grid. The starred entries (*) indicate that, since there is a single subdomain, the factor is structurally and numerically identical to Unconstrained PILU(k). Dashed entries (-) indicate the solutions either diverged or failed to converge after 200 iterations.

nodes per	subdomain		Problem 1			Problem 2, $\epsilon = 1/100$			Problem 2, $\epsilon = 1/500$		
subdomain	count	level	uncon	const	bj	uncon	const	bj	uncon	const	bj
262,144	1	0	43	*	*	20	*	*	19	*	*
		1	29	*	*	16	*	*	16	*	*
		2	24	*	*	13	*	*	8	*	*
		3	19	*	*	11	*	*	8	*	*
		4	16	*	*	9	*	*	6	*	*
32,768	8	0	45	45	53	21	21	22	32	32	26
		1	32	33	41	17	17	18	14	14	19
		2	27	29	37	15	15	18	11	11	17
		3	22	24	33	12	12	15	8	8	13
		4	19	21	29	11	11	14	7	7	13
4,096	64	0	43	43	55	23	23	24	33	33	49
		1	31	32	45	18	18	19	15	15	21
		2	25	27	41	15	15	20	12	11	22
		3	20	23	39	13	13	18	9	9	16
		4	17	20	36	11	11	16	8	8	19
512	512	0	41	41	56	27	27	30	28	28	67
		1	29	31	48	19	20	26	18	16	29
		2	25	26	46	17	17	22	11	12	36
		3	21	23	44	13	14	22	11	11	31
		4	18	21	43	11	14	20	9	12	34
64	4,096	0	43	43	64	32	32	41	28	28	-
		1	30	33	60	21	23	36	17	18	124
		2	26	30	58	17	20	37	13	15	115
		3	21	28	58	14	18	37	12	17	127
		4	17	28	58	11	18	35	10	17	132
8	32,768	0	46	46	83	40	40	74	43	43	-
		1	32	41	82	27	36	74	24	46	-
		2	25	40	82	15	37	70	11	45	-
		3	19	40	82	10	36	74	5	44	-
		4	16	40	82	8	36	74	4	45	-

TABLE 4.4

Iteration comparisons for 256 × 256 grid. The starred entries (*) indicate that, since there is a single subdomain, the factor is structurally and numerically identical to Unconstrained PILU(k). Dashed entries (-) indicate the solution diverged.

nodes per	subdomain		Problem 1			Problem 2, $\epsilon = 1/100$			Problem 2, $\epsilon = 1/500$		
subdomain	count	level	uncon	const	bj	uncon	const	bj	uncon	const	bj
65,536	1	0	109	*	*	94	*	*	9	*	*
		1	67	*	*	55	*	*	11	*	*
		2	55	*	*	44	*	*	11	*	*
		3	40	*	*	30	*	*	8	*	*
		4	34	*	*	22	*	*	6	*	*
		5	29	*	*	18	*	*	6	*	*
		6	24	*	*	14	*	*	5	*	*
16,384	4	0	110	110	125	95	95	95	16	16	13
		1	69	69	85	56	56	56	14	14	13
		2	56	56	74	46	46	46	14	14	12
		3	47	47	60	31	31	31	11	11	11
		4	39	39	54	24	24	24	9	9	9
		5	33	35	54	19	19	19	8	8	8
		6	29	31	50	16	16	16	7	7	8
1,024	64	0	112	112	132	97	97	97	24	24	20
		1	72	73	98	57	57	57	20	21	19
		2	59	60	89	49	50	49	19	19	19
		3	46	47	78	35	35	35	17	17	18
		4	38	40	73	27	27	27	16	16	17
		5	33	35	71	23	-	23	15	15	17
		6	31	34	70	20	20	20	14	14	15
64	1,024	0	120	120	158	111	111	111	59	59	71
		1	81	85	137	69	72	69	49	51	64
		2	66	73	132	58	60	58	41	44	63
		3	52	63	128	45	51	45	37	39	-
		4	43	57	126	38	45	38	34	36	57
		5	37	54	127	32	43	32	33	36	61
		6	34	52	126	29	39	29	29	33	61
16	4,096	0	130	130	193	-	-	-	-	-	-
		1	90	101	182	82	93	82	70	76	-
		2	67	88	179	-	77	-	49	62	-
		3	49	84	178	41	68	41	32	-	132
		4	40	84	177	30	68	30	21	62	129
		5	35	84	177	22	68	22	13	62	129
		6	29	84	177	17	68	17	7	62	129

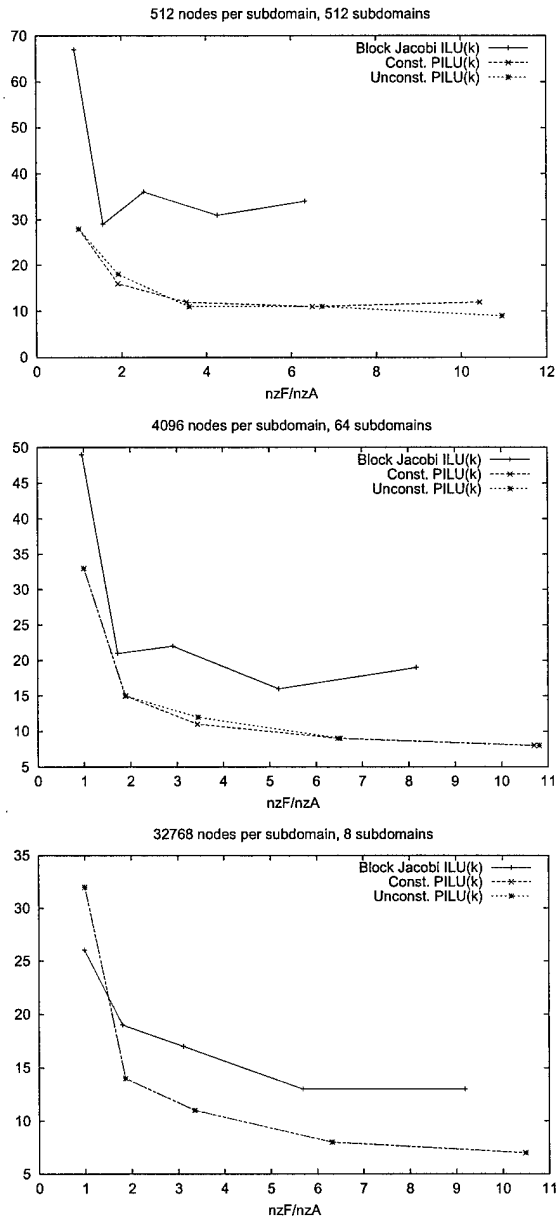

FIG. 4.3. *Convergence comparison as a function of preconditioner size for convection-diffusion problem,* $\epsilon = 1/500$ *on* $64 \times 64 \times 64$ *grid. Data points are for levels 0 through 4. Data points for Constrained and Unconstrained PILU(k) are indistinguishable in the third graph.*

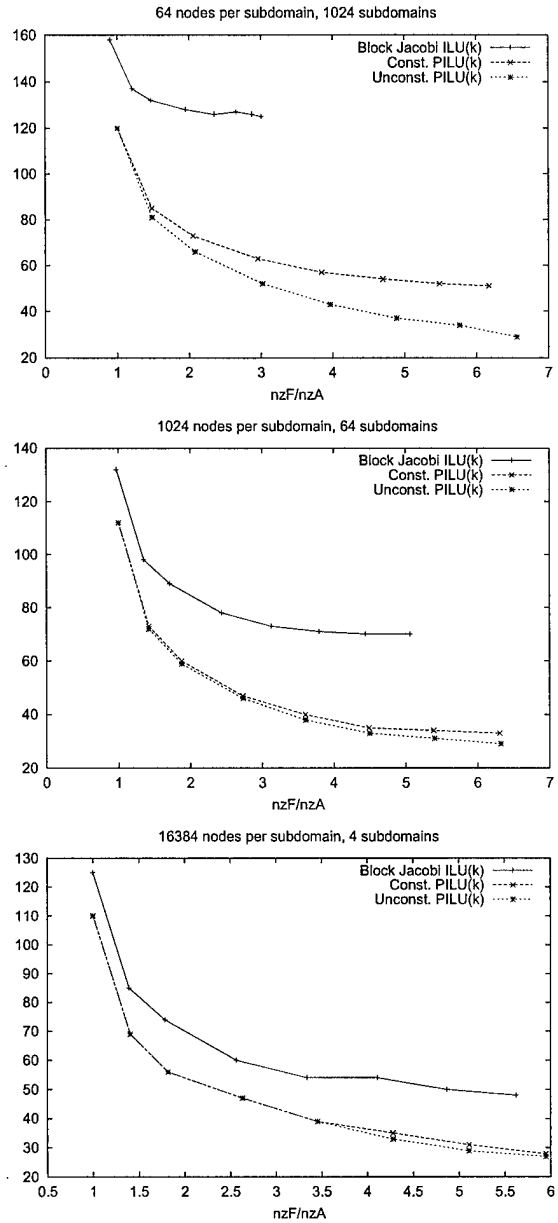

FIG. 4.4. *Convergence as a function of preconditioner size, for Poisson's equation on* 256×256 *grid. Data points are for levels 0 through 7.*

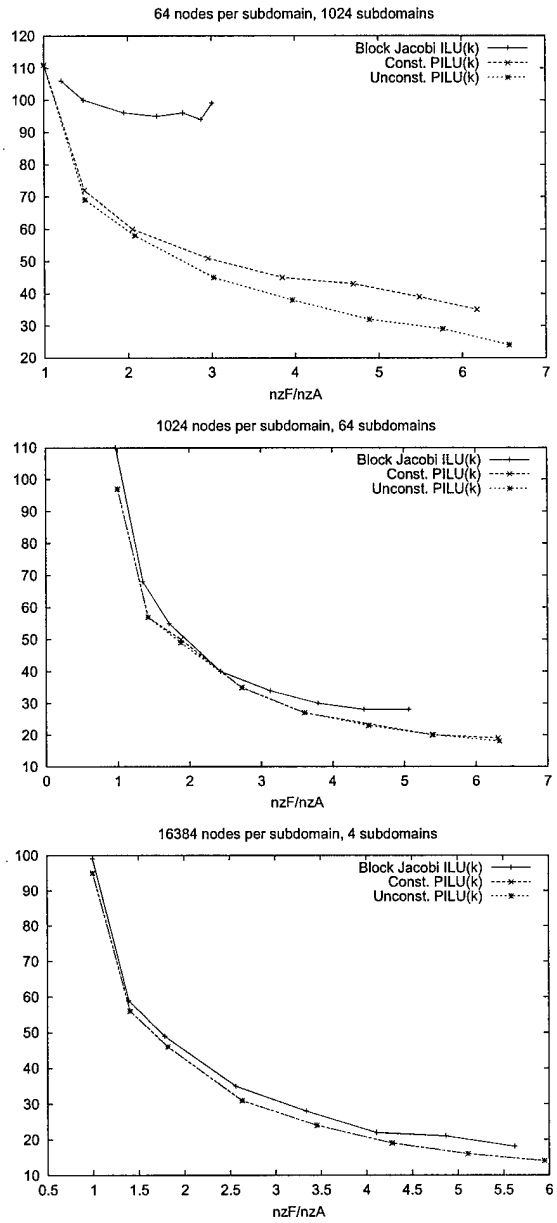

FIG. 4.5. *Convergence comparison as a function of preconditioner size for convection-diffusion problem,* $\epsilon = 1/100$ *on* 256×256 *grid. Data points are for levels 0 through 7 (some data points omitted due to convergence failure). Data points for Constrained and Unconstrained PILU(k) are nearly indistinguishable in the second and third graphs.*

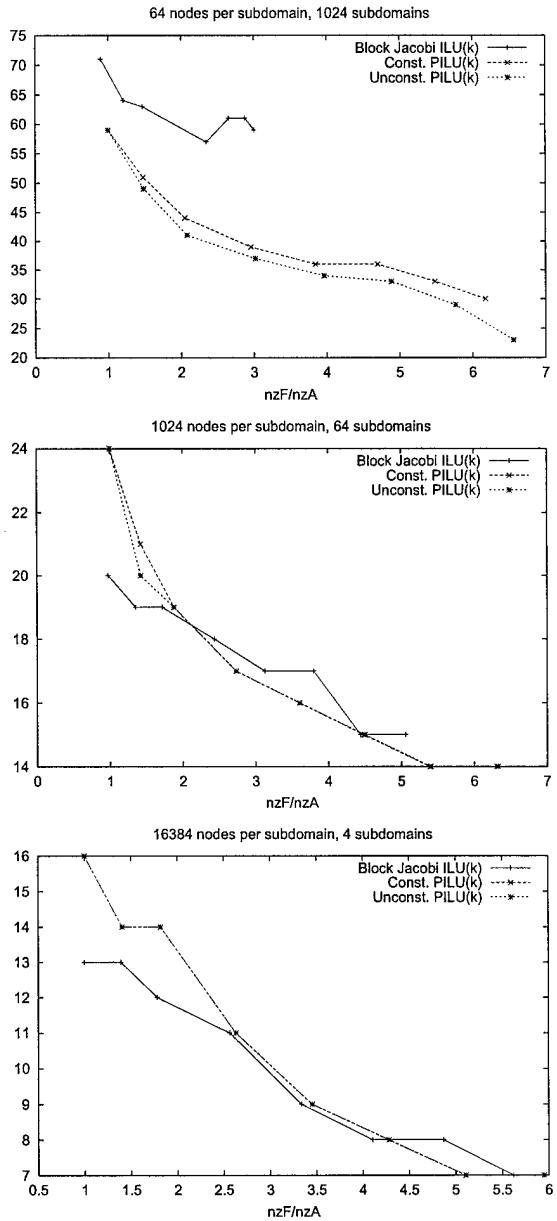

FIG. 4.6. *Convergence comparison as a function of preconditioner size for convection-diffusion problem,* $\epsilon = 1/500$ *on* 256×256 *grid. Data points are for levels 0 through 7. (some data points omitted due to convergence failure). Data points for Constrained and Unconstrained PILU(k) are nearly indistinguishable in the second and third graphs.*