Parallel Multilevel Block ILU Preconditioning Techniques for Large Sparse Linear Systems [1]

Chi Shen, Jun Zhang, Kai Wang

Laboratory for High Performance Scientific Computing
and Laboratory for High Performance Scientific Computing
and Computer Simulation,
Department of Computer Science, University of Kentucky,
773 Anderson Hall, Lexington, KY 40506-0046, USA

Abstract

We present a class of parallel preconditioning strategies built on a multilevel block incomplete LU (ILU) factorization technique to solve large sparse linear systems on distributed memory parallel computers. The preconditioners are constructed by using the concept of block independent sets. Two algorithms for constructing block independent sets of a distributed sparse matrix are proposed. We compare a few implementations of the parallel multilevel ILU preconditioners with different block independent set algorithms and different coarse level solution strategies. We also use some diagonal thresholding and perturbation strategies to enhance factorization stability. Numerical experiments indicate that our parallel multilevel block ILU preconditioners are robust and efficient.

Key words: Parallel multilevel preconditioning, sparse matrices, Schur complement techniques

Email addresses: cshen@csr.uky.edu (Chi Shen), jzhang@cs.uky.edu (Jun Zhang), kwang0@cs.uky.edu (Kai Wang).

[1] The research work was supported in part by the U.S. National Science Foundation under grants CCR-9902022, CCR-9988165, CCR-0092532, and ACI-0202934, by the U.S. Department of Energy Office of Science under grant DE-FG02-02ER45961, by the Japanese Research Organization for Information Science & Technology, and by the University of Kentucky Research Committee.

1 Introduction

For solving large sparse linear systems, multilevel preconditioning techniques may take advantage of the fact that different parts of the error spectrum can be treated independently on different level scales. Several multilevel block ILU preconditioners (BILUM and BILUTM [10,11]) have been developed and tested to show promising convergence results and scalability for solving certain matrix problems. The parallel implementations of the multilevel preconditioning methods are a nontrivial issue. The difficulty associated with these methods is related to computing the block independent sets from a distributed sparse matrix.

This paper presents a class of truly multilevel block ILU preconditioners (PBILUM) based on building block independent sets from a distributed sparse matrix. Two algorithms are designed to search for block independent sets. One is based on a parallelized block independent set search and the other is based on a sequential block independent set search analogous to that used in BILUM. However, the construction of the global block independent set in both algorithms is carried out in parallel.

In order to enhance the robustness and stability of the parallel block ILU factorization, we implement some diagonal thresholding and perturbation strategies in PBILUM both for the block independent set construction at each level and for the local block Schur complement ILU factorization at the coarsest level [12,14]. The experimental results show that these strategies help improve the robustness of the multilevel ILU factorization.

This paper is organized as follows. In the next section, we outline the general framework for constructing two level block ILU preconditioning techniques. In Section 3, we discuss useful strategies to implement PBILUM with two different block independent set search algorithms. The diagonal thresholding and perturbation strategies are given in Section 4. Numerical experiment section (Section 5) contains a comparison of different variants of the PBILUM preconditioners for solving various distributed linear systems. Concluding remarks are given in Section 6.

2 Two Level Block ILU Preconditioner

PBILUM is built on the framework of the two level block ILU techniques PBILU2 described in [13]. For completeness, we give a short overview of the main steps in constructing PBILU2.

2.1 Distributed matrix based on block independent set

Consider a sparse linear system $Ax = b$, where A is a nonsingular sparse matrix of order n. First we use a simple approach to find independent blocks in a sparse matrix. This approach is similar to that used in [10,11]. The only difference between them is that we use some strategies for avoiding some rows with small diagonal elements to be included in the block independent set, which will be discussed in Section 4.

Assume that a block independent set with a uniform block size k has been found, and the coefficient matrix A is permuted into a block form. Such a reordering leads to a block system:

$$\left(\begin{array}{ccc|c} B_1 & & & F_1 \\ & B_2 & & F_2 \\ & & \ddots & \vdots \\ & & B_m & F_m \\ \hline E_1 & E_2 & \cdots & C_1 \\ & & & C_2 \\ & & & \vdots \\ & & & C_m \end{array}\right) \left(\begin{array}{c} u_1 \\ u_2 \\ \vdots \\ u_m \\ \hline y_1 \\ y_2 \\ \vdots \\ y_m \end{array}\right) = \left(\begin{array}{c} f_1 \\ f_2 \\ \vdots \\ f_m \\ \hline g_1 \\ g_2 \\ \vdots \\ g_m \end{array}\right),$$

where m is the number of processors used in the computation. At the same time, the global vector of unknowns x is split into two subvectors $(u, y)^T$,where $u = (u_1, \ldots, u_m)^T$, $y = (y_1, \ldots, y_m)^T$, and vector b is also split conformally into subvectors f and g.

For deriving the global Schur complement matrix in parallel, the submatrix E need to be partitioned in column.

$$E = \left(M_1 \; M_2 \; \cdots \; M_m \right) = \left(E_1 \; E_2 \; \cdots \; E_m \right)^T.$$

Where M_i has the same number of columns as B_i. All submatrices B_i, F_i, E_i, M_i, C_i and the subvectors f_i and g_i are assigned to the same processor i. u_i and y_i are the local part of the unknown vectors.

2.2 Derivation of Schur complement techniques in parallel

We now consider a block LU factorization in the form of

$$\begin{pmatrix} B & F \\ E & C \end{pmatrix} = \begin{pmatrix} I & 0 \\ EB^{-1} & I \end{pmatrix} \begin{pmatrix} B & F \\ 0 & S \end{pmatrix}, \tag{1}$$

where S is the global Schur complement:

$$S = (C_1, C_2, ..., C_m)^T - \sum_{i=1}^{m} M_i B_i^{-1} F_i. \tag{2}$$

On the ith processor, a local submatrix A_i is formed and looks like

$$A_i = \begin{pmatrix} B_i & F_i \\ M_i & \bar{C}_i \end{pmatrix} = \left(\begin{array}{c|c} B_i & F_i \\ \hline & 0 \\ & \vdots \\ M_i & C_i \\ & \vdots \\ & 0 \end{array}\right). \tag{3}$$

We perform a restricted Gaussian elimination on the local matrix (3). After this is done, we obtain a new reduced submatrix $\tilde{C}_i$ and it forms a piece of the global Schur complement matrix. If the factorization procedure is exact, $S = \tilde{C}_1 + \tilde{C}_2 + ... + \tilde{C}_m$. Each submatrix $\tilde{C}_i$ is partitioned into m parts (using the same partitionings as the original submatrix C), and the corresponding parts are scattered to relevant processors. After receiving and summing all of those parts of the submatrices, the "local" Schur complement matrix S_i is formed. Here "local" means some rows of the global Schur complement that are held in a given processor. Note that $S_i \neq \tilde{C}_i$.

3 Parallel Multilevel Preconditioner

In algebraic multilevel preconditioning techniques, the reduced systems are recursively constructed as the Schur complement matrices. The structure of the multilevel preconditioning matrices is depicted in Fig. 1.

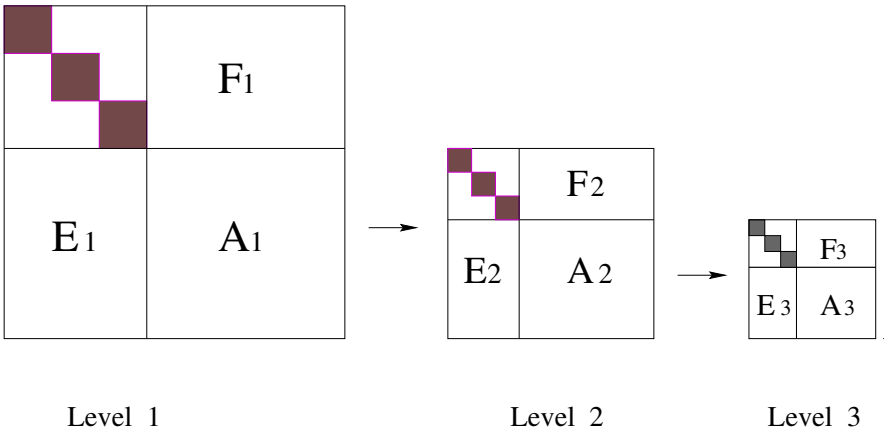

Fig. 1. Structure of the multilevel preconditioning matrices.

Algorithm 3.1. PBILUM Factorization.

1. **If** not at the last level, **Then**
2. Find a global block independent set.
3. Permute the local matrix in each processor.
4. Generate the local reduced matrix for the next level.
5. **Else**
6. Do ILUT factorization on the local matrix.
7. **EndIf**

This process is similar to the BILUTM factorization [11]. The parallel implementation of this generic process is described in the following algorithm:

The parallel implementations of Steps 4 and 6 have already been discussed in Section 2.2 and in [13]. One of the most important steps in Algorithm 3.1 is to develop strategies for constructing block independent set efficiently. Two approaches to accomplishing this task are discussed in the following.

3.1 Sequential block independent set search

One of the simplest ideas is the sequential search, utilizing the greedy algorithm developed in [10]. Once a distributed matrix is formed, each processor holds several rows of it. The local matrix structure (graph) is sent to one processor. We can perform independent block search in this processor. This procedure is similar to that used in the two level case. The main disadvantage of this algorithm is that it may be expensive to implement, since it may cause other processors idle. However the algorithm can find a larger block independent set, compared to that found by the parallel search algorithm, to be discussed later. This may lead to a smaller system at the coarsest level and the preconditioning quality approaches that of the (sequential) BILUTM.

3.2 Parallel block independent set search

Once the global Schur complement matrix is formed in parallel, each processor holds several rows of it. For the sake of reducing communications between processors, instead of sending those rows back to one processor, we perform block independent set searching in the local submatrices, i.e., in the block diagonal matrix of the global Schur complement. See the Fig. 2.

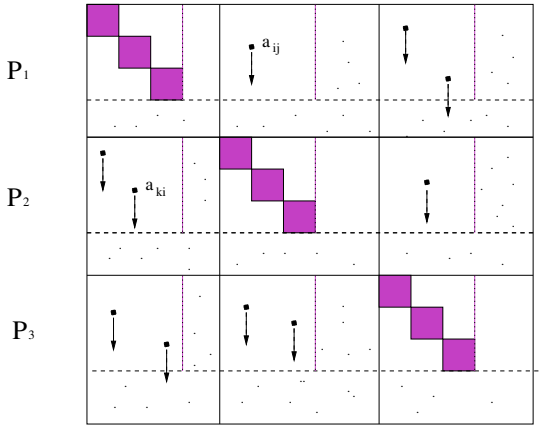

Fig. 2. Parallel search of local block independent sets.

Nevertheless the union of these local block independent sets may not form a global block independent set. Some nodes in a local block independent set may be coupled to the nodes in other local block independent sets in other processors. In Fig. 2, the solid dots with arrows represent those offdiagonal nonzero entries that destroy the global independence of certain local independent blocks. In order to form a global block independent set, those nodes should be removed from the local block independent set.

Node moving approach. With all permutation information, each processor removes those coupled nodes independently. After this is done, the union of all distributed local block independent sets forms a global block independent set. There is a potential double dropping problem. E.g., in Fig. 2, suppose the entry a_{ij} in the processor 1 and the entry a_{ki} in the processor 2 are both nonzero. Independently, the processors 1 and 2 will remove the nodes i and k from their local block independent sets respectively. Although this double dropping problem does not affect the global independence of the global block independent set, the size of it is unnecessarily reduced. The reason is that *either* removing the node i from the processor 1 *or* removing the node k from the processor 2 is sufficient to guarantee the global independence of the local block independent sets. In this paper, this simple approach is used in our experimental tests. A generic algorithm for parallel block independent set search is described in Algorithm 3.2.

This algorithm has many advantages compared with the sequential one. The

Algorithm 3.2 Parallel Block Independent Set Search.

0. In the ith processor, do:
1. Use sequential algorithm to find a local block independent set.
2. Find the local permutation array $\tilde{P}_i$.
3. Exchange its permutation array $\tilde{P}_i$ with other processors.
4. Remove (or drop) those nodes (or entries) which are not globally independent.
5. Find the new local permutation array P_i.

Algorithm 3.3 One Step of the PBILUM Preconditioner.

Forward elimination step:

1. For $l = 1, \ldots, (L-1)$,
2. Compute $\tilde{g}_{l,i} = g_{l,i} - E_{l,i}\,(B_{l,1}^{-1} f_{l,1}, B_{l,2}^{-1} f_{l,2}, \ldots, B_{l,m}^{-1} f_{l,m})^T$.
3. Permute the right hand side $\tilde{g}_{l,i}$ to $(f_{l+1,i}, g_{l+1,i})^T$.

Coarsest level solution:

4. Solve $S_{L,i}\, y_L = \tilde{g}_{L,i}$ iteratively.

Backward solution step:

5. For $l = (L-1), 1, -1$,
6. Perform matrix-vector product: $\tilde{f}_{l,i} = F_{l,i}\, y_l$.
7. Solve $(L_{B_{l,i}} U_{B_{l,i}})\, u_{l,i} = f_{l,i} - \tilde{f}_{l,i}$.
8. Permute the vector $(u_{l,i}, y_{l,i})^T$ back to $y_{l-1,i}$.

communication cost is low in the construction phase. When we perform forward and backward procedures in multilevel preconditioning steps, the permutations of some vectors at each level are local. However, since we only perform block independent set search in the local matrix, the sizes of the block independent set may be smaller than that constructed by the sequential algorithm.

3.3 Induced parallel multilevel preconditioner

As discussed in [13], with the main factorization shown in (1) and (2), the preconditioning step $LUe = r$ can be implemented in three steps (with the subvector notations):

$$\begin{cases} \tilde{g} = g - E\, B^{-1}\, f, \\ S\, y = \tilde{g}, \\ B\, u = f - F\, y. \end{cases} \tag{4}$$

Thus the parallel multilevel preconditioning procedure can be written as follows in Algorithm 3.3, based on the multilevel structure in Fig (1) and the preconditioning step (4).

Here i denotes the processor number and L denotes the number of total levels. Based on the Algorithm 3.2, permutation Step 3 and Step 8 are performed locally and independently by each processor without any communication. We denote this version of the parallel multilevel preconditioner as PBILUM_P. Analogously, the multilevel preconditioner associated with the sequential search algorithm is denoted as PBILUM_S. Even though constructing a block independent set in PBILUM_S is conceptually easier than that in the parallel one, the preconditioning application phase is much more complicated to implement. In PBILUM_S, the permutations associated with the block independent set ordering are not local. Moreover, unlike in PBILUM_P, the nodes of PBILUM_S in each processor change from level to level. This may increase communications. However, the block independent sets constructed in PBILUM_S are larger and have better quality, and PBILUM_S achieves higher degree of load balancing at each level.

3.4 Multilevel Schur complement preconditioning strategy

If the Algorithm 3.3 includes an iteration procedure at a certain level or at all levels, it may lead to a class of more accurate preconditioning algorithms, which involves a Krylov iteration step. The multilevel preconditioner solves the given Schur complement matrix using the lower level parts of the PBILUM as the preconditioner. As it is discussed in [9,15,16], it can be beneficial to find more accurate solutions to the coarse level systems. However the amount of the computations required to carry out each preconditioning step is increased. In stead of employing Krylov iterations at every levels, we only have it for the first level Schur complement system. This preconditioning strategy will be abbreviated as P_SchPre and S_SchPre for PBILUM_P and PBILUM_S algorithms respectively, following the notations of [15].

In a multilevel preconditioner, let A_1 be the first exact reduced system. Since there is a Krylov iteration at this level, a matrix-vector multiplication and a preconditioning step are performed. Thus we need A_1 in one form or another. To avoid using extra memory, one strategy proposed in [9,15] is considered. The idea comes from the expansion $A_1 = C - EB^{-1}F$. In processor i, submatrix $A_{1,i} = C_i - E_i B_i^{-1} F_i$. For B_i^{-1}, it can be replaced by a solution with $(L_{B_i} U_{B_i})$, which comes from the restricted Gaussian elimination. If an inexact factorization is performed, we get an approximate matrix $\tilde{A}_1$ to A_1. Instead of performing explicit matrix-vector product $\tilde{A}_{1,i} y$ in processor i, we perform this operation by computing $x_i = (L_{B_{1,i}} U_{B_{1,i}})^{-1} F_{1,i}\, y$, followed by computing $C_{1,i}\, y - E_{1,i}\, x$ $(= \tilde{A}_{1,i}\, y)$. Here $x = (x_1, x_2, \ldots, x_m)^T$. This strategy can avoid storing the reduced system matrix.

4 Diagonal Thresholding and Perturbation

An unstable ILU factorization may be resulted from factoring a matrix with small or zero diagonals [1,4]. In order to improve stability, it is beneficial to only include those nodes with large diagonal entries in the block independent set [12,14]. There are a few heuristic strategies to implement the diagonal thresholding strategy [3,9,12,14]. A diagonal dominance measure array of the rows is computed and is used to determine the qualification of each row to be included in the block independent set. To account for the vast difference in the matrix properties, we give a carefully designed strategy to compute the relative diagonal dominance measure of each row and a diagonal thresholding criterion β. They are used to determine when it is acceptable to include a node in the block independent sets. The measure $\omega(i)$ for the ith row is computed as $\omega(i) = |a_{ii}|/\max|a_{ij}|$, and $\beta = \min(\omega_{\text{ave}}, (\omega_{\min} + \omega_{\max})/2)$. The computation cost is $O(nnz)$, where "nnz" is the number of nonzeros of the given matrix.

We also employ a diagonal perturbation strategy on the Schur complement matrix at the coarsest level before performing the block ILU factorization. Since this is the last level, it may have some zero or small diagonal entries. We remedy this kind of diagonal entries with a perturbation strategy based on a given criterion and the diagonal dominance measure of the rows.

5 Numerical Experiments

In the numerical experiments, we compare performance of different variants of the PBILUM preconditioners. All matrices are considered general sparse and the right-hand side is generated by assuming that the solution is a vector of all ones and the initial guess is a vector of all zeros. We use a flexible variant of restarted GMRES(30) in a parallel version which is in P_SPARSLIB [7]. The preconditioner for the coarsest level system is the block Jacobi. The (outer iteration) computations are terminated when the 2-norm of the residual vector is reduced by a factor of 10^8. The inner iterations on the coarsest level satisfies convergence test if the 2-norm of the residual vector is reduced by a factor of 10^2 or the maximum number of 5 iterations is reached.

For all PBILUM variants, the first level block independent set from the original matrix is constructed and the matrix is permuted in a single processor before it is distributed to the other processors. The CPU time for this step is included in the preconditioner construction time. The computations are carried out on a 32 processor (750MHz) subcomplex of an HP superdome (supercluster) at the University of Kentucky. The code is written in Fortran 90 in double precision.

In all tables containing numerical results, "$bsize$" and "n_c" stand for independent block size and the size of the coarsest system respectively. "iter" is the number of the preconditioned iterations. "T_{total}" is the sum of the iteration time and preconditioner construction time. "sparsity" stands for the sparsity ratio of the memory used by the preconditioner with respect to that used by the original matrix. "τ"and "p" are the dropping tolerance and fill-in parameters respectively used in the incomplete Gaussian elimination. "α" is used in the diagonal perturbation algorithm. "ε" is a dropping tolerance used for sparsifing the reduced system for the next level use. Unless specified, $\varepsilon = 10\,\tau$.

5.1 2D and 3D convection diffusion problems

The 2D convection diffusion problems arising from discretizing the following partial differential equation

$$\Delta u + \mathrm{Re}\,(\exp(xy-1)u_x - \exp(-xy)u_y) = 0,$$

defined on a unit square, and the 3D problems from

$$\Delta u + \mathrm{Re}\,(x(x-1)(1-2y)(1-2z)u_x + y(y-1)(1-2z)(1-2x)u_y$$

$$+z(z-1)(1-2x)(1-2y)u_z) = 0,$$

defined on a unit cube. Here Re is the Reynolds number, which represents the strength of the convection against diffusion. For the 2D problem, the discretization scheme is the 9-point fourth compact scheme and the resulting matrices are called the 9-POINT matrices. For the 3D problem, the discretization scheme is either the 7-point central difference scheme or the 19-point fourth order compact scheme, the resulting matrices are referred to as the 7-POINT or 19-POINT matrices, respectively.

We first generate a series of 2D 9-POINT matrices with $\mathrm{Re} = 1.0$, by increasing the problem size from $n = 200^2$ to $n = 1000^2$. 6 processors are used to solve them. The test results are listed in Table 1.

In our experiments, we find that the preconditioner with the Schur complement preconditioning strategy achieves better convergence rate than the other strategies. However, since it involves many matrix-vector products, each iteration needs more CPU time. Comparing the 2-level and the 4-level PBILUM preconditioners, since the size of the last reduced system for the 4-level one is much smaller than that for the 2-level one, as the problem size increases, the 4-level preconditioners converge faster and use less CPU time and memory than the 2-level preconditioner. Usually, the sparsity ratio of the multilevel

Table 1
9-POINT matrices with Re = 1.0 ($bsize = 100$, $\tau = 10^{-3}$, $p = 50$).

problem size	preconditioner	level	iter	T_{total}	sparsity	n_c
40,000	PBILUM_S	2	25	3.10	3.44	7100
	PBILUM_S	4	27	3.13	3.26	1500
	S_SchPre	4	12	3.50	3.26	1500
	PBILUM_P	4	26	2.45	3.26	2849
	P_SchPre	4	12	2.98	3.26	2849
360,000	PBILUM_S	2	94	77.50	3.60	63800
	PBILUM_S	4	98	68.51	3.29	15000
	S_SchPre	4	27	64.10	3.29	15000
	PBILUM_P	4	98	66.10	3.30	20001
	P_SchPre	4	27	60.30	3.30	20001
1,000,000	PBILUM_S	2	166	392.89	3.63	171900
	PBILUM_S	4	175	356.20	3.35	41900
	S_SchPre	4	49	296.90	3.35	41900
	PBILUM_P	4	176	324.00	3.35	46685
	P_SchPre	4	47	266.94	3.35	46685

preconditioners is smaller than that of the two level preconditioner due to its larger dropping tolerance $\varepsilon > \tau$.

Then we test a 3D 7-POINT convection diffusion problem. We fixed the problem size as $n = 1,000,000$, $nnz = 6,490,000$, $bsize = 100$, $\tau = 10^{-2}$, $p = 20$. In the Fig. 3, we see that the 4-level PBILUM performs better than the 2-level counterpart. Using 32 processors, PBILUM_S is faster than PBILUM_P since the former one generates a smaller last level reduced system due to its ability to construct larger block independent sets at all levels.

We test another set of the 3D 19-POINT matrices with Re = 0.0, using the same parameters as in the previous test. In this test, we increase both the problem size and the number of processors. The problem size is defined as $n = nx^3$, where nx is the number of unknowns in one coordinate direction. For $nx = 20$, where the problem size is 20^3, we use 4 processors. For $nx = 40$, the problem size is 40^3, we use 8 processors. For $nx = 60, 80, 90, 100$, we use $16, 24, 28, 32$ processors, respectively. Due to the configuration of the test problems, we are unable to generate a set of matrices so that submatrices of a fixed size can be allocated to each processor. We are only interested in comparing the relative performance of our preconditioners.

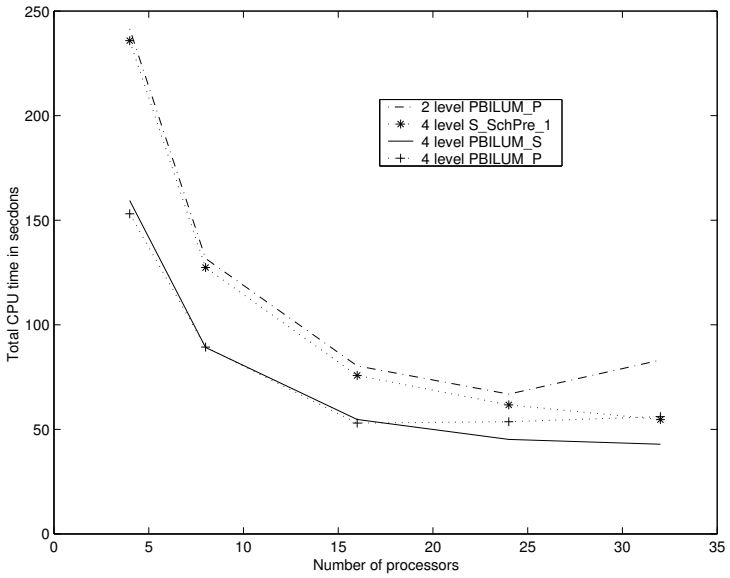

Fig. 3. Comparison of parallel iteration time for different variants of the PBILUM preconditioners for solving a 7-POINT matrix with Re = 1000.0 and $n = 1,000,000$.

The Fig. 4 shows that the total CPU time increases as the problem size and the number of processors increases. Again, even though the 2-level preconditioner has almost the same number of iterations as the 4-level preconditioners. The 2-level one takes more CPU time as it has a larger coarsest level system to solve. We confirm that the 4-level PBILUM_S and PBILUM_P achieve better scalability than the others.

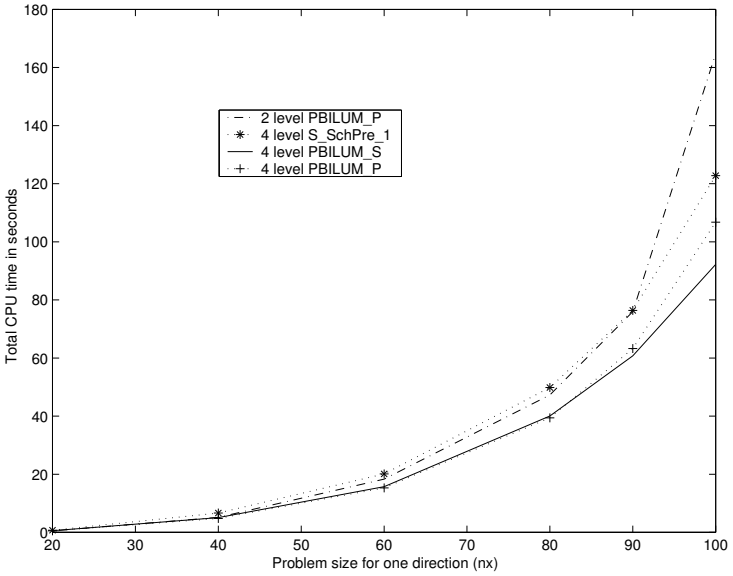

Fig. 4. Comparison of parallel iteration time for different variants of the PBILUM preconditioners for solving a series of 19-POINT matrices with Re = 0.0.

5.2 FIDAP matrices

This set of the test matrices were extracted from the test problems provided in the FIDAP package [5]. As many of these matrices have small or zero diagonals, they are difficult to solve using standard ILU preconditioners. We solve 28 of the largest FIDAP matrices ($n > 2000$). We use the diagonal perturbation strategy with the parameter α for the last reduced system. The range of the α values is from 10^{-2} to 10^{-4}. Before the preconditioner construction, most of the FIDAP matrices are scaled. We first make the 2-norm of each column of a matrix to be unity, then we scale the rows so that the 2-norm of each row is unity. Compared with the previous tested problems, the FIDAP matrices are small. We only use 2 processors in our test and GMRES(50) is used. Since the test results for PBILUM_P and PBILUM_S are almost the same, we only list the test results of some larger FIDAP matrices for PBILUM_S in Table 2.

Table 2
Solving the FIDAP matrices using PBILUM_S.

Matrices	n	nnz	level	iter	sparsity	$bsize$	τ	p
FIDAP008	3,096	106,302	3	17	3.50	100	10^{-8}	300
FIDAP009	3,363	99,397	4	22	4.46	100	10^{-10}	300
FIDAP014	3,251	66,647	3	24	4.00	50	10^{-8}	100
FIDAP015	6,867	9,6421	2	9	3.50	100	10^{-12}	200
FIDAP018	5,773	69,335	2	8	3.92	100	10^{-8}	300
FIDAP019	12,005	259,863	4	5	3.60	100	10^{-10}	400
FIDAP031	3,909	115,299	2	32	3.45	200	10^{-8}	400
FIDAP035	19,716	218,308	2	23	3.41	100	10^{-8}	200
FIDAP036	3,079	53,851	4	18	2.73	100	10^{-4}	50
FIDAP040	7,740	456,226	2	57	2.40	500	10^{-5}	100
FIDAPM11	22,294	623,554	4	181	8.99	100	10^{-10}	200
FIDAPM15	9,287	98,519	4	89	2.92	100	10^{-5}	30
FIDAPM29	13,668	186,294	2	82	5.40	500	10^{-5}	70
FIDAPM37	9,152	765,944	2	87	3.77	500	10^{-5}	800

The robustness of our parallel preconditioners is impressive, since the existing best result that we are aware of is that 27 of the largest (not scaled) FIDAP matrices are solved by a sequential multilevel preconditioner with a dual reordering strategy [14].

5.3 Flame matrix

We generate a sparse matrix from the 2D numerical simulation of an axisymmetric laminar diffusion flame [2]. It is the Jacobi matrix associated with the nonlinear Newton iteration and has $n = 21,060$ and $nnz = 382,664$. The matrix has a 4-by-4 block structure, corresponding to the number of coupled governing PDEs. The 2D nonuniform tensor product grid is covered with a mesh of 65×81. Mixed 9-point central and upwind difference schemes are used so that each row of the matrix has at most 36 nonzeros. The matrix is scaled as we did for the FIDAP matrices.

We solve the Flame matrix using 4 processors to show the performance difference of PBILUM with different levels. The sparsity ratio for PBILUM with different levels is almost the same, which is around 2.5. In this test, we use $\varepsilon = \tau$.

Except for the 2-level case, we find that the 4-level and the 8-level implementations of both the PBILUM_S and PBILUM_P preconditioners achieve almost the same results. However, for the 6-level case, PBILUM_P has almost the same results as for the 8-level case, thus it converges faster than the 6-level PBILUM_S. For simplicity, we only list the test results of PBILUM_S in Fig. 5. we can see that PBILUM with more levels converges faster than that with fewer levels.

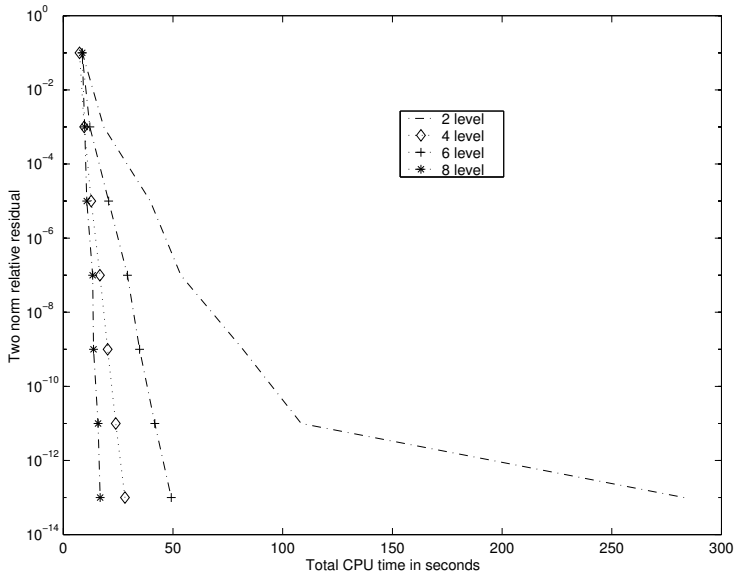

Fig. 5. Comparison of the performance of PBILUM_S with different levels for solving the Flame matrix ($bsize = 100$, $\tau = 10^{-6}, p = 40$, $\alpha = 10^{-4}$).

6 Concluding Remarks

We developed a class of truly parallel multilevel block incomplete LU preconditioning techniques (PBILUM) based on two algorithms for constructing block independent sets from a distributed sparse matrix. Our experimental results showed that the multilevel PBILUM is more robust and converges faster than the 2-level PBILUM for solving large scale problems. Thus, the efforts put in developing the more complicated truly parallel multilevel preconditioners are well paid.

In particular, we showed that the algorithm to search the block independent sets in parallel is faster than the one to search block independent sets in a serial mode. For very large scale problems in which each processor has a large local subproblem, PBILUM_P is shown to be better than PBILUM_S. However, when the number of processors is large and the local subproblem is small, PBILUM_P tends to generate a larger last level reduced system and takes longer time in the solution process. Even in these cases, their total computational costs are comparable, as PBILUM_P takes much less CPU time in the preconditioner construction phase.

Our PBILUM preconditioners are not optimal, similar to all existing preconditioners aiming at solving general sparse linear systems. There is much room for improvements. If we can have better and more inexpensive strategies to avoid the double dropping problem in a distributed environment, we will have larger block independent sets at all levels. This gain may be significant as we see that the inner iteration process to solve the last level reduced system of large size may slow down the overall computational process. Thus a new approach to these problems is under way.

References

[1] E. Chow and Y. Saad. Experimental study of ILU preconditioners for indefinite matrices. *J. Comput. Appl. Math.*, 86(2):387–414, 1997.

[2] C. C. Douglas, A. Ern, and M. D. Smooke. Multigrid solution of flame sheet problems on serial and parallel computers. *Parallel Algorithms and Appl.*, 10:225–236, 1997.

[3] I. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Anal. Appl.*, 22(4):973–996, 2001.

[4] H. C. Elman. A stability analysis of incomplete LU factorization. *Math. Comp.*, 47(175):191–217, 1986.

[5] M. Engelman. FIDAP: Examples Manual, Revision 6.0. Technical report, Fluid Dynamics International, Evanston, IL, 1991.

[6] G. Karypis and V. Kumar. Parallel threshold-based ILU factorization. Technical Report 96-061, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1996.

[7] Y. Saad. Parallel sparse matrix library (P_SPARSLIB): The iterative solvers module. In *Advances in Numerical Methods for Large Sparse Sets of Linear Equations*, volume Number 10, Matrix Analysis and Parallel Computing, PCG 94, pages 263–276, Yokohama, Japan, 1994. Keio University.

[8] Y. Saad and M. Sosonkina. Distributed Schur complement techniques for general sparse linear systems. *SIAM J. Sci. Comput.*, 21(4):1337–1356, 1999.

[9] Y. Saad and B. Suchomel. ARMS: an algebraic recursive multilevel solver for general sparse linear systems. *Numer. Linear Alg. Appl.*, 9(5):359–378, 2002.

[10] Y. Saad and J. Zhang. BILUM: block versions of multielimination and multilevel ILU preconditioner for general sparse linear systems. *SIAM J. Sci. Comput.*, 20(6):2103–2121, 1999.

[11] Y. Saad and J. Zhang. BILUTM: a domain-based multilevel block ILUT preconditioner for general sparse matrices. *SIAM J. Matrix Anal. Appl.*, 21(1):279–299, 1999.

[12] Y. Saad and J. Zhang. Diagonal threshold techniques in robust multi-level ILU preconditioners for general sparse linear systems. *Numer. Linear Algebra Appl.*, 6(4):257–280, 1999.

[13] C. Shen and J. Zhang. Parallel two level block ILU preconditioning techniques for solving large sparse linear systems. *Paral. Comput.*, 28(10):1451–1475, 2002.

[14] J. Zhang. A multilevel dual reordering strategy for robust incomplete LU factorization of indefinite matrices. *SIAM J. Matrix Anal. Appl.*, 22(3):925–947, 2000.

[15] J. Zhang. On preconditioning Schur complement and Schur complement preconditioning. *Electron. Trans. Numer. Anal.*, 10:115–130, 2000.

[16] J. Zhang. A class of multilevel recursive incomplete LU preconditioning techniques. *Korean J. Comput. Appl. Math.*, 8(2):213–234, 2001.