Experiments and Bounds on Block Diagonal Preconditioning

Mark Yan-Ming Chang, Martin H. Schultz

Research Report YALEU/DCS/RR-994
November 1993

This work is supported by Office of Naval Research under grant
N00014-91-J-1576

TR-994

EXPERIMENTS AND BOUNDS ON BLOCK DIAGONAL PRECONDITIONING

MARK YAN-MING CHANG* AND MARTIN H. SCHULTZ†

Abstract. We have previously shown the theorems concerning the condition number of our block diagonal preconditioning for the Laplace operator. We now extend the theorems to more general self-adjoints elliptic operators with variable coefficients.

We then show some experiments using these kinds of block diagonal preconditioning on a network of workstations.

1. Introduction. The sequential preconditioned conjugate gradient (PCG) methods are well-known as efficient ways to solve large sparse systems of linear equations [2]. These methods have been studied on variety of parallel architectures by Saad and Schultz [7].

Following the approach in our previous paper [3], we consider a class of perfectly parallelizable preconditioners and try to construct ones that have good convergence properties, rather than trying to develop good parallel versions of the best sequential preconditioners. Directly parallelizing good sequential preconditioners in terms of the number of iterations and sequential work was studyed in [1].

In a parallel computing environment with very high communication cost relative to computation, cutting down the amount of communication in the algorithm is very desirable. It is sometimes even worthwhile to do more computation in order to do less communication. This is the reason why we are investigating the perfectly parallel preconditioners.

We did some experiments using these methods on some workstations connected on a local area network. Comparing with a very good sequential modified incomplete factorization preconditioner, we achieve relatively good speedups.

Let us first show the generalized theorems.

2. Upper Bounds of Condition Number for General Elliptic Problem. We have derived bounds on the condition number of the block diagonal preconditioned systems for the discretized Laplace equation. In this section, we will extend the theorems to general elliptic equations.

Consider the second-order self-adjoint linear elliptic partial differential equation(PDE) on a unit square domain

$$(2.1) \qquad -\frac{\partial}{\partial x}[p(x,y)\frac{\partial}{\partial x}u(x,y)] - \frac{\partial}{\partial y}[r(x,y)\frac{\partial}{\partial y}u(x,y)] + s(x,y)u(x,y) = f(x,y)$$

* Department of Computer Science, Yale University, P. O. Box 208285 Yale Station, New Haven, CT 06520 E-mail: *chang-mark-yan-ming@cs.yale.edu* or *changym@cs.yale.edu.*

† Department of Computer Science, Yale University, P. O. Box 208285 Yale Station, New Haven, CT 06520 E-mail: *schultz-martin@cs.yale.edu.*

for $(x, y) \in$ interior of $\Omega = [0, 1] \times [0, 1]$, subject to the zero Dirichlet boundary conditions $u(x, y) = 0$ for $(x, y) \in$ boundary of Ω.

We discretize the square domain in both the x and y directions with step size h, where $h \equiv \frac{1}{n+1}$, and n is number of discretizations along each dimension.

The lower case u is the continuous function that we are looking for. We will use the upper case letter U to denote the discretized solution of the continuous function u.

Since rectangular domain can be scaled to the unit square, we do not lose generality by concentrating on the unit square domain.

Assume $p(x, y)$, $r(x, y)$ and $s(x, y)$ are positive and bounded in Ω:

$$0 < \underline{p} \leq p(x, y) \leq \bar{p}$$

$$0 < \underline{r} \leq r(x, y) \leq \bar{r}$$

$$0 < s(x, y) \leq \bar{s}$$

where $\underline{p}$, $\bar{p}$, $\underline{r}$, $\bar{r}$, and $\bar{s}$ are constants.

We define the forward difference operators D_x and D_y as follows:

$$D_x U_{i,j} \equiv \frac{U_{i+1,j} - U_{i,j}}{h},$$

$$D_y U_{i,j} \equiv \frac{U_{i,j+1} - U_{i,j}}{h}.$$

We define the backward difference operators D_x^- and D_y^- as:

$$D_x^- U_{i,j} \equiv \frac{U_{i,j} - U_{i-1,j}}{h},$$

$$D_y^- U_{i,j} \equiv \frac{U_{i,j} - U_{i,j-1}}{h}.$$

We use the central difference to discretize the elliptic PDE (2.1) and get

$$-D_x[p(x_i, y_j) D_x^- U_{i,j}] - D_y[r(x_i, y_j) D_y^- U_{i,j}] + s(x_i, y_j) U_{i,j} = f(x_i, y_j).$$

Let A denote the discretized elliptic operator (we normalize A by taking out the factor $\frac{1}{h^2}$):

$$\frac{1}{h^2} A U_{i,j} = -D_x[p(x_i, y_j) D_x^- U_{i,j}] - D_y[r(x_i, y_j) D_y^- U_{i,j}] + s(x_i, y_j) U_{i,j}.$$

We solve the following system of linear equations to get the discretized solution of the PDE (2.1):

$$AU = b,$$

where the entries of vector b is defined by $b_{i,j} = h^2 f(x_i, y_j)$.

We can prove the following lemma using the zero boundary condition and summation by parts [6]:

LEMMA 2.1.

$$\langle AU, U\rangle = h^2\{\langle p(x_i,y_j)D_xU, D_xU\rangle + \langle r(x_i,y_j)D_yU, D_yU\rangle + \langle s(x_i,y_j)U, U\rangle\}$$

where

$$\langle p(x_i,y_j)D_xU, D_xU\rangle \equiv \sum_{i=0}^{n}\sum_{j=0}^{n}[p(x_i,y_j)(D_xU_{i,j})^2]$$

$$\langle r(x_i,y_j)D_yU, D_yU\rangle \equiv \sum_{i=0}^{n}\sum_{j=0}^{n}[r(x_i,y_j)(D_yU_{i,j})^2]$$

$$\langle s(x_i,y_j)U, U\rangle \equiv \sum_{i=0}^{n}\sum_{j=0}^{n}[s(x_i,y_j)U_{i,j}^2]$$

The above lemma for the discretized Laplace operator L becomes

$$\langle LU, U\rangle = h^2(\langle D_xU, D_xU\rangle + \langle D_yU, D_yU\rangle).$$

The next lemma bounds the elliptic operator by the Laplace operator.

LEMMA 2.2.

Let A be the discretized elliptic operator of equation (2.1) and L be the discretized Laplace operator, then there exist constants $\underline{\mu}$ and $\bar{\mu}$ such that

$$\underline{\mu}\langle LU, U\rangle \le \langle AU, U\rangle \le \bar{\mu}\langle LU, U\rangle.$$

Proof.

By using lemma 2.1, and the fact that $s(x,y)$ is positive, we have

$$\begin{aligned}
&\frac{1}{h^2}\langle AU, U\rangle \\
=\ & \langle p(x_i,y_j)D_xU, D_xU\rangle + \langle r(x_i,y_j)D_yU, D_yU\rangle + \langle s(x_i,y_j)U, U\rangle \\
=\ & \sum_{i=0}^{n}\sum_{j=0}^{n}[p(x_i,y_j)(D_xU_{i,j})^2] + \sum_{i=0}^{n}\sum_{j=0}^{n}[r(x_i,y_j)(D_yU_{i,j})^2] + \sum_{i=0}^{n}\sum_{j=0}^{n}[s(x_i,y_j)U_{i,j}^2] \\
\ge\ & \sum_{i=0}^{n}\sum_{j=0}^{n}[\underline{p}(D_xU_{i,j})^2] + \sum_{i=0}^{n}\sum_{j=0}^{n}[\underline{r}(D_yU_{i,j})^2] \\
\ge\ & \min(\underline{p},\underline{r})\{\sum_{i=0}^{n}\sum_{j=0}^{n}[(D_xU_{i,j})^2] + \sum_{i=0}^{n}\sum_{j=0}^{n}[(D_yU_{i,j})^2]\} \\
=\ & \underline{\mu}\frac{1}{h^2}\langle LU, U\rangle
\end{aligned}$$

where $\underline{\mu} = \min(\underline{p}, \underline{r})$.

This proves the lower bound of the lemma, and the upper bound is proved in a similar manner:

$$
\begin{aligned}
(2.2) \qquad & \frac{1}{h^2}\langle AU, U\rangle \\
= \; & \sum_{i=0}^{n}\sum_{j=0}^{n}[p(x_i,y_j)(D_xU_{i,j})^2] + \sum_{i=0}^{n}\sum_{j=0}^{n}[r(x_i,y_j)(D_yU_{i,j})^2] + \sum_{i=0}^{n}\sum_{j=0}^{n}[s(x_i,y_j)U_{i,j}^2] \\
\le \; & \sum_{i=0}^{n}\sum_{j=0}^{n}[\bar{p}(D_xU_{i,j})^2] + \sum_{i=0}^{n}\sum_{j=0}^{n}[\bar{r}(D_yU_{i,j})^2] + \sum_{i=0}^{n}\sum_{j=0}^{n}[\bar{s}U_{i,j}^2] \\
\le \; & \max(\bar{p},\bar{r})\{\sum_{i=0}^{n}\sum_{j=0}^{n}[(D_xU_{i,j})^2] + \sum_{i=0}^{n}\sum_{j=0}^{n}[(D_yU_{i,j})^2]\} + \bar{s}\sum_{i=0}^{n}\sum_{j=0}^{n}U_{i,j}^2 \\
= \; & \max(\bar{p},\bar{r})\frac{1}{h^2}\langle LU, U\rangle + \bar{s}\sum_{i=0}^{n}\sum_{j=0}^{n}U_{i,j}^2
\end{aligned}
$$

Apply the lemma that bounds the term $U_{i,j}^2$ from [3] we have

$$
\begin{aligned}
\bar{s}\sum_{i=0}^{n}\sum_{j=0}^{n}U_{i,j}^2 \; \le \; & \bar{s}\sum_{i=0}^{n}\sum_{j=0}^{n}\{\frac{h}{8}\langle D_xU_{*,y_j}, D_xU_{*,y_j}\rangle + \frac{h}{8}\langle D_yU_{x_i,*}, D_yU_{x_i,*}\rangle\} \\
= \; & \bar{s}\sum_{j=0}^{n}\{\frac{h}{8}(n+1)\langle D_xU_{*,y_j}, D_xU_{*,y_j}\rangle\} + \bar{s}\sum_{i=0}^{n}\{\frac{h}{8}(n+1)\langle D_yU_{x_i,*}, D_yU_{x_i,*}\rangle\} \\
= \; & \bar{s}\frac{1}{8}\langle D_xU, D_xU\rangle + \bar{s}\frac{1}{8}\langle D_yU, D_yU\rangle \\
= \; & \bar{s}\frac{1}{8}\frac{1}{h^2}\langle LU, U\rangle
\end{aligned}
$$

Substitute this result back into (2.2) to get the upper bound:

$$
\begin{aligned}
\langle AU, U\rangle \; & \le \; \max(\bar{p},\bar{r})\langle LU, U\rangle + \bar{s}\frac{1}{8}\langle LU, U\rangle \\
& \le \; \bar{\mu}\langle LU, U\rangle
\end{aligned}
$$

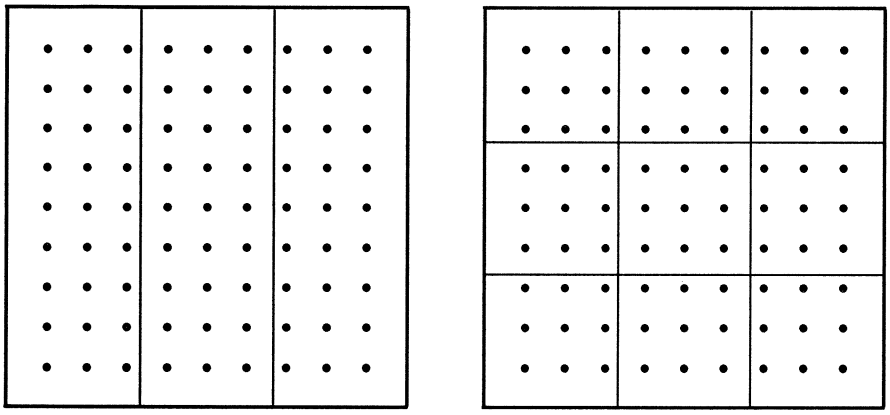
where $\bar{\mu} = \max(\bar{p}, \bar{r}, \bar{s}\frac{1}{8})$. □

Now let us consider cutting the problem domain by lines as show in Figure 1. Similar to the way we obtain a block preconditioning matrix M for the Laplace matrix L, we can obtain a block preconditioning matrix B for the elliptic matrix A by ignoring the connection across the cutting lines.

The cutting lines divide the big domain into many smaller subdomains. In each subdomain, we can apply lemma 2.2, and this allows us to prove the following lemma:

LEMMA 2.3. *There exist constants $\underline{\nu}$ and $\bar{\nu}$ such that*

$$\underline{\nu}\langle MU, U\rangle \le \langle BU, U\rangle \le \bar{\nu}\langle MU, U\rangle.$$



cut in one dimension cut in both dimensions

FIG. 1. *Cutting two dimensional grid with lines.*

Now we are ready for the generalized theorem.

THEOREM 2.4.

Let A be the discretized elliptic operator of equation (2.1), and L be the discretized Laplace operator. If we cut q vertical and/or horizontal lines through the $n \times n$ grid to obtain block diagonal preconditioners B for A, and M for L, then the condition number is bounded above as:

$$cond(B^{-1}A) \leq O(qn + q + 1).$$

Proof.

$$\frac{\langle AU, U\rangle}{\langle BU, U\rangle} = \frac{\langle AU, U\rangle}{\langle LU, U\rangle} * \frac{\langle LU, U\rangle}{\langle MU, U\rangle} * \frac{\langle MU, U\rangle}{\langle BU, U\rangle}.$$

Apply lemma 2.2 and lemma 2.3 we get

$$(\underline{\mu}/\bar{\nu}) \min(\frac{\langle LU, U\rangle}{\langle MU, U\rangle}) \leq \frac{\langle AU, U\rangle}{\langle BU, U\rangle} \leq (\bar{\mu}/\underline{\nu}) \max(\frac{\langle LU, U\rangle}{\langle MU, U\rangle})$$

Since the matrix $(B^{-1}A)$ is similar to a symmetric matrix, we can apply Rayleigh quotient theorem [9, p312] and get:

$$cond(B^{-1}A) \leq \frac{\bar{\mu}\bar{\nu}}{\underline{\mu}\underline{\nu}} cond(M^{-1}L).$$

Apply the bound we derived for the Laplace operator, we get our result

$$cond(B^{-1}A) \leq \frac{\bar{\mu}\bar{\nu}}{\underline{\mu}\underline{\nu}}(qn + q + 1) = O(qn + q + 1).$$

□

3. Test Problem. We experiment with elliptic PDE's in both two and three dimensions. The two dimensional problem is defined in equation (2.1):

$$-\frac{\partial}{\partial x}[p(x,y)\frac{\partial}{\partial x}u(x,y)] - \frac{\partial}{\partial y}[r(x,y)\frac{\partial}{\partial y}u(x,y)] + s(x,y)u(x,y) = f(x,y)$$

The variable coefficients are defined as:

$$p(x,y) = e^{-xy}$$

$$r(x,y) = e^{+xy}$$

$$s(x,y) = 2 + \frac{1}{1+x+y}.$$

The three dimensional problem is defined by:

$$-\frac{\partial}{\partial x}[p(x,y,z)\frac{\partial}{\partial x}u(x,y,z)] - \frac{\partial}{\partial y}[q(x,y,z)\frac{\partial}{\partial y}u(x,y,z)] - \frac{\partial}{\partial y}[r(x,y,z)\frac{\partial}{\partial y}u(x,y,z)]$$

$$+w(x,y,z)u(x,y,z) = f(x,y,z)$$

The variable coefficients are defined as:

$$p(x,y,z) = e^{xyz}$$

$$q(x,y,z) = e^{xyz}$$

$$r(x,y,z) = e^{xyz}$$

$$w(x,y,z) = 40 + \frac{1}{1+x+y+z}.$$

Different right hand sides were tried. The relative performance between different methods seem to stay the same. The tables we show here uses the 1 vector as the right hand side. Zero initial guess is used.

We use central difference to discretize the PDE to get a system of linear equations. This linear system is solved using our block diagonal preconditioned conjugate gradient methods.

4. Implementations. The parallel preconditioned conjugate gradient methods are implemented in C-Linda [4]. We can run our program anywhere that C-Linda is running. This includes network of workstations.

The sequential program that we are comparing with is the modified incomplete factorization preconditioned conjugate gradient method, MIC/PCG written in FORTRAN. We run the sequential program with varying fill in level, and pick the best run time.

In our block diagonal preconditioning, we need to solve the local block in each processor. We call various existing solvers to solve the local system. The solvers used are SMPAK [8] and ESSL [5]. We also tried MIC routines on the local system.

4.1. Communication Costs. We pick the perfect parallelizable preconditioners for our parallel PCG methods, so there is no communication needed for the purpose of preconditioning. But there are two parts of the parallel PCG needing communication, namely the matrix vector multiply and the dot product.

The communication pattern in the matrix vector multiply depends on the structure of the sparse matrix. Each processor holds a block of rows of the matrix and part of the vector. We do some preprocessing to find out exactly who needs which part of the vector before starting the PCG loop. In the PCG loop, each processor will put out only the vector elements that are needed by some other processor into one tuple. Each processor will read in the relevant tuples and get the needed remote vector elements. These remote elements will be appended to the local vector, and the sparse matrix index will be modified to point to the local copies of the remote vectors. This index change is done only once before the iteration stage. After this change, the sequential matrix vector multiply routine can be called and no extra indexing is done for the previously remote elements.

For the parallel dot product, each processor first computes a partial sum locally, then the partial sums are added together. There are many communication patterns to use for the global sum, including trees, linear array, etc. We did not see a big difference in timing for small number of processors. We settled with binary tree sum.

5. Modify the Diagonal. Analogous to the method of modifying the incomplete factorization, we can add the cut off points in our block diagonal preconditioner back to the diagonal. Actually, we can add a fraction of the off-diagonal elements back. It turns out that the factor of 1 is the best when we solve the block diagonal preconditioner exactly. Adding the cut off points back to the diagonal makes the preconditioner have the same row sum as the original matrix. This gives the preconditioner more power without changing the matrix structure.

fraction	iteration	fraction	iteration	fraction	iteration
1.00	14	.99	40	1.01	47
-0.10	65	.90	52	1.10	503
-0.20	65	.80	57	1.20	844
-0.30	66	.70	59	1.30	764
-1.00	61	.20	62	1.70	1132
-1.10	68	.00	64	2.00	1342

TABLE 1

parallel Smpak/PCG, Laplace on 240×240 *grid, with* 1×4 *cut, reduce 2-norm of the residual to 10e-7, CG iteration=947*

Table 1 shows the iteration numbers for adding different fraction of the off diagonal back to the diagonal. We can see that factor of 1 is far better, and it gives an iteration number of 14. Not adding back to the diagonal will require 64 iterations.

We don't know a theorem showing why adding off-diagonal elements back to the diagonal creates a better preconditioner. But the following two theorems show that

there are more eigenvalues equaling to 1 if we do add the off-diagonal elements back.

THEOREM 5.1.

Let A be the discretized elliptic operator of equation (2.1). We cut q vertical lines through the $n \times n$ grid to obtain the block diagonal preconditioner B for A. Assume each subgrid has at least 3 columns after the cuts, then the number of eigenvalues equaling 1 for the preconditioned system $B^{-1}A$ is:

$$n^2 - 2qn.$$

Proof.

We write out the matrix A using natural ordering in block form as:

$$A \equiv \begin{bmatrix} A_1 & C_{1,2} & & & & & \\ C_{1,2}^T & A_2 & C_{2,3} & & & & \\ & C_{2,3}^T & A_3 & C_{3,4} & & & \\ & & C_{3,4}^T & A_4 & C_{4,5} & & \\ & & & C_{4,5}^T & \ddots & & \\ & & & & & A_q & C_{q,q+1} \\ & & & & & C_{q,q+1}^T & A_{q+1} \end{bmatrix}.$$

The block diagonal preconditioner can be obtained by dropping the off-diagonal blocks from the A matrix:

$$B \equiv \begin{bmatrix} A_1 & 0 & & & & & \\ 0 & A_2 & 0 & & & & \\ & 0 & A_3 & 0 & & & \\ & & 0 & A_4 & 0 & & \\ & & & 0 & \ddots & & \\ & & & & & A_q & 0 \\ & & & & & 0 & A_{q+1} \end{bmatrix}.$$

The preconditioned system will look like:

$B^{-1}A = I + R = I + B^{-1}C$

$$I + B^{-1} \begin{bmatrix} 0 & C_{1,2} & & & & & \\ C_{1,2}^T & 0 & C_{2,3} & & & & \\ & C_{2,3}^T & 0 & C_{3,4} & & & \\ & & C_{3,4}^T & 0 & C_{4,5} & & \\ & & & C_{4,5}^T & \ddots & & \\ & & & & & 0 & C_{q,q+1} \\ & & & & & C_{q,q+1}^T & 0 \end{bmatrix}.$$

Zooming in on the off-diagonal blocks for q=2, we see:

$$R = B^{-1} \left[\begin{array}{c|c|c} \begin{matrix} & 0 & \\ & & \\ \begin{bmatrix} 0 & 0 & D_1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & & \\ & & \\ & 0 & \end{matrix} & \begin{matrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ D_1 & 0 & 0 \end{bmatrix} & & \\ & 0 & \\ & & \begin{bmatrix} 0 & 0 & D_2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix} & \begin{matrix} & 0 & \\ & & \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ D_2 & 0 & 0 \end{bmatrix} & & \\ & & \\ & 0 & \end{matrix} \end{array}\right].$$

It is easy to imagine the matrices for q greater than 2. Each of the nonzero blocks D_1 and D_2 are $n \times n$ full rank diagonal matrices. They appear in different block rows of C. Therefore C has $2qn$ linearly independent columns, and $rand(C)$ is $2qn$.

Note, $rand(R) = rand(B^{-1}C) = rand(C)$, so R has $2qn$ nonzero eigenvalues and $n^2 - 2qn$ zero eigenvalues. Therefore $B^{-1}A$ has $n^2 - 2qn$ eigenvalues equaling 1. □

THEOREM 5.2.

Let A be the discretized elliptic operator of equation (2.1). We cut q vertical lines through the $n \times n$ grid to obtain the block diagonal preconditioner B for A. Assume each subgrid has at least 3 columns after the cuts. If we add the elements of $A - B$ back to the diagonals of B to obtain B_d, then the number of eigenvalues equaling 1 for the preconditioned system $B_d^{-1}A$ is:

$$n^2 - qn.$$

Proof. Let us first look at the matrices for $q = 2$ in detail. The matrix A using natural ordering in block form is:

$$A = \left[\begin{array}{c|c|c} \begin{matrix} & A_1 & \\ & & \\ \begin{bmatrix} 0 & 0 & D_1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & & \\ & & \\ & 0 & \end{matrix} & \begin{matrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ D_1 & 0 & 0 \end{bmatrix} & & \\ & A_2 & \\ & & \begin{bmatrix} 0 & 0 & D_2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{matrix} & \begin{matrix} & 0 & \\ & & \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ D_2 & 0 & 0 \end{bmatrix} & & \\ & & \\ & A_3 & \end{matrix} \end{array}\right].$$

Adding the off-diagonal elements to the main diagonal, we get the preconditioner:

$$B_d = \begin{bmatrix} B_1 & 0 & 0 \\ 0 & B_2 & 0 \\ 0 & 0 & B_3 \end{bmatrix} =$$

$$\left[\begin{array}{c|c|c} A_1 + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & D_1 \end{bmatrix} & \begin{matrix} 0 \\ \\ \end{matrix} & \begin{matrix} 0 \\ \\ \end{matrix} \\ \hline \begin{matrix} 0 \\ \\ \end{matrix} & A_2 + \begin{bmatrix} D_1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & D_2 \end{bmatrix} & \begin{matrix} 0 \\ \\ \end{matrix} \\ \hline \begin{matrix} 0 \\ \\ \end{matrix} & \begin{matrix} 0 \\ \\ \end{matrix} & A_3 + \begin{bmatrix} D_2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array}\right].$$

$$B_d^{-1}A = I + R_d = I + B_d^{-1}C_d$$

$$I + B_d^{-1}\left[\begin{array}{c|c|c} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -D_1 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ D_1 & 0 & 0 \end{bmatrix} & \begin{matrix} 0 \\ \\ \end{matrix} \\ \hline \begin{bmatrix} 0 & 0 & D_1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} -D_1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -D_2 \end{bmatrix} & \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ D_2 & 0 & 0 \end{bmatrix} \\ \hline \begin{matrix} 0 \\ \\ \end{matrix} & \begin{bmatrix} 0 & 0 & D_2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} -D_2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{array}\right],$$

where D_1 and D_2 are full rank diagonal matrices. It is easy to imagine the matrices for q greater than 2.

Note along each vertical line in the matrix C_d, the n point-columns on left of the line is linearly dependent on the n point-columns on right of the line. Also, each cluster of $2n$ point-columns appears in different rows from the clusters around other lines. Thus the matrix C_d has total of qn linearly independent columns. Since $rand(R_d) = rand(B_d^{-1}C_d) = rand(C_d)$, then R_d has qn nonzero eigenvalues and $n^2 - qn$ zero eigenvalues. Therefore $B_d^{-1}A$ has $n^2 - qn$ eigenvalues equaling 1. □

There are two ways to bound the number of iterations for a PCG method. One way is to use the condition number of the preconditioned system. Another is to use the eigenvalue distribution of the preconditioned system. Basicly, one PCG iteration can take care one cluster of eigenvalues. If the eigenvalues are clustered in a few groups, then only a few iterations are needed to make the process converge. The previous two theorems do not prove the convergence behavior of our block-diagonal PCG, but it does show that if we add the off-diagonal elements back to the diagonal in the preconditioner, there will be more eigenvalues equaling 1. This is a good sign by itself.

6. Timing results. We present here the run times on RS6000 workstations using Network Linda V2.5.2. The workstations are connected via an Allnode switch from

IBM/Endicott. We have a network of RS6000/340's and RS6000/560's at Yale University computer science department.

We also did experiments on the IBM SP1 computer located at Maui High Performance Computing Center (MHPCC). The SP1 has thirty two RS6000/370 nodes, and a fast switch called TB0 connecting all of the nodes. The C-Linda used on the SP1 is a beta test version.

The time are in seconds, and they are truncated to keep only one or two digits after the decimal point.

The unknowns are ordered in the x direction first, then the y direction. The z direction is ordered last when there is such a dimension. When we cut the square domain for the 2D case in different ways along the x and y directions, we obtain different bandwidth in the preconditioning matrix. Similarly for the 3D case, we cut the cube domain by planes and obtain different block-diagonal preconditioners.

We use some shorthands to refer to the methods we use. Here are the explanations of these shorthands:

MIC/PCG Sequential preconditioned conjugate gradient (PCG) method with modified incomplete Cholesky (MIC) factorization as preconditioner.

MIC/ParPCG Parallel PCG with MIC factorization of the block-diagonal matrix as the preconditioner.

MIC/ParPCG2 Similar to MIC/ParPCG, the difference is that a fraction of the off-diagonal elements are added back to the diagonal.

Smpak/ParPCG2 Similar to MIC/ParPCG2, the difference is that the block-diagonal matrix is factored and solved completely as a preconditioner by calling routines in the SMPAK package.

BandESSL/ParPCG2 Similar to Smpak/ParPCG2, the difference is that the preconditioning solver is a banded solver from the ESSL library.

MICPCG/ParPCG2 Similar to Smpak/ParPCG2, the difference is that the precondition solver is the sequential MICPCG.

The columns of data in each table represent the following:

p the number of processors.

cut the way the square or cube domain is cut. 2x3 means that the square domain is cut into 2 pieces in the x direction and 3 pieces in the y direction. 4x2x1 means that the cube is cut into 4 pieces in the x direction, 2 pieces in the y direction, and no cut in the z direction.

itr or **iter**, the number of iterations.

pre the preconditioning time.

ord the ordering time for preconditioning.

fac the factorization time for preconditioning.

fraction the fraction constant used to multiply the off-diagonal element before adding the result to the diagonal in the preconditioner.

sov the forward/backward solve time for preconditioning.

dot the dot product time.

mvp the matrix vector multiply time.

cmp the computation time.
cmm the communication time.
tot the total time for a group of operations.
spup the speedup with respect to the 1 processor time in the same table.
sp same as **spup**.
sp2 the speedup with respect to MIC/PCG time on 1 processor.
sp3 the scaled speedup: assume the sequential time is p multiplied by the parallel computation time.

			Dot			MVP			SAXPY			CG	
p	cut	iter	cmm	tot	spup	cmm	tot	spup	cmm	tot	spup	tot	spup
1	1x1	2040	0.00	74.53	1.00	0.00	403.47	1.00	0	142.97	1.00	622.8	1.00
2	1x2	2041	13.97	51.55	1.44	7.40	207.87	1.94	0	73.40	1.94	347.9	1.79
2	2x1	2040	13.76	51.39	1.45	7.49	207.86	1.94	0	72.76	1.96	347.7	1.79
3	1x3	2040	14.08	45.65	1.63	11.81	146.40	2.75	0	51.20	2.79	268.8	2.31
3	3x1	2040	24.43	49.34	1.51	12.02	145.19	2.77	0	47.94	2.98	265.2	2.34
4	1x4	2040	27.05	46.56	1.60	23.08	124.54	3.23	0	38.89	3.67	227.5	2.73
4	2x2	2040	15.17	38.32	1.94	19.83	120.17	3.35	0	41.09	3.47	231.7	2.68
4	4x1	2039	40.01	59.20	1.25	21.57	121.57	3.31	0	39.00	3.66	237.7	2.62
5	1x5	2040	30.44	45.41	1.64	22.32	103.56	3.89	0	29.40	4.86	218.0	2.85
5	5x1	2039	30.30	45.27	1.64	24.77	105.09	3.83	0	29.11	4.91	215.4	2.89
6	1x6	2039	54.97	67.55	1.10	14.41	84.83	4.75	0	23.90	5.98	195.6	3.18
6	2x3	2041	28.98	41.59	1.79	28.67	98.05	4.11	0	25.92	5.51	193.9	3.21
6	3x2	2040	35.10	47.64	1.56	19.71	86.66	4.65	0	23.96	5.96	184.7	3.37
6	6x1	2040	44.64	57.18	1.30	26.38	93.47	4.31	0	23.87	5.98	200.1	3.11

TABLE 2

Parallel CG on 420 x 420 grid, on RS6000/560, run time in seconds. reduce 2-norm of the residual to 10e-7

In Table 2, the time spent in different part of the CG method are shown. For the dot-product and matrix vector multiply, there are both the communication time and the computation time. For the SAXPY operations, there is no communication. The speedups are shown for each group of operations and for the CG method as a whole. The table shows that both MVP and SAXPY are quite parallelizable, while dot product has too big a communication cost as we increase the number of processors, no matter which kind of global sum we used.

Table 3 shows the sequential MIC/PCG time with different levels of fill in. For fill in level 3, we have the best total run time. We will use this time to compute one of the speedups for the parallel timings.

In Tables 4, 5, and 6, the cut-off points are added back to the diagonal of the preconditioner. This decreases the iteration number required without introducing any extra fill in.

fillin	iteration	ordering	factor	solvs	dot prod	mvp	saxpy	total time
0	122	0.56	0.87	25.20	4.73	24.34	8.55	64.44
1	84	0.79	1.09	20.80	3.26	16.77	5.98	48.84
2	73	1.04	1.37	21.31	2.83	14.62	5.14	46.45
3	61	1.70	2.08	22.49	2.38	12.26	4.29	45.31
4	52	2.49	2.86	23.71	2.02	10.48	3.68	45.35
5	47	3.52	3.87	25.43	1.83	9.49	3.31	47.56
6	42	4.67	4.91	26.02	1.64	8.50	2.94	48.79

TABLE 3

sequential 420 x 420 grid MIC/PCG on RS6000/560, run time in seconds. reduce 2-norm of the residual to 10e-7

Table 4 shows the parallel PCG using MIC on the local preconditioner. Note, this is not the parallel implementation of the sequential MIC/PCG method. We believe that a direct implementation of the MIC/PCG method in parallel will require too much communication on a slow network for workstations.

Our preconditioner depends on the way we cut the domain. We first derive a block diagonal matrix by ignoring the connections along the cutting lines. We then use MIC on this block diagonal matrix and the modified incomplete fatorization is used as the parallel preconditioner. For two processors, there are two ways to cut the domain into two pieces, namely 1x2 and 2x1. Since we order the unknowns along the x direction first, the 1x2 cutting gives us a longer bandwidth in the resulting matrix than the 2x1 cutting. For six processors, there are four ways to cut the domain.

For a given cut of the domain, we run our program with different levels of fill in, and choose the best time to put in the table.

In addition to the fill in level, the way which we cut the domain affects the power of the parallel preconditioner. On the one hand, we want to cut more in the x direction so we have a smaller bandwidth for the preconditioner. On the other hand, we want to keep the maximum number of cuts in any direction small to have faster convergence according to our theorem. There is a best way to cut the domain for a given number of processors.

In Table 5, the block diagonal preconditioner is solved completely by calling minimum degree sparse linear system solver called SMPAK [8]. We have three speedups in the table: sp, sp2 and sp3. The sp is computed against the sequential direct solver of SMPAK, while the sp2 is computed against the sequential MIC/PCG time. We call sp3 the scaled speedup, computed by assuming the sequential time is p times of the parallel computation time, where p is the number of processors used. The scaled speedup does not require us to obtain a sequential time for the same problem size, so we could report the result for a much bigger problem. A bigger problem will show a even better scaled speedup.

In Table 6, we use the banded linear system solver from ESSL [5] to factor and solve the block diagonal preconditioner. Since banded solvers in general need more memory than sparse solvers, we can not solve the given problem on one or two processors. In

p	itr	fillin	cut	pre	ord	fac	slv	dot	mvp	cmp	cmm	tot	spup
1	61	3	1x1	26.2	1.7	2.0	22.4	2.38	12.26	45.3	0.00	45.3	1.00
2	182	1	1x2	23.3	0.4	0.5	22.3	4.87	18.66	51.4	3.27	54.7	0.82
2	134	2	2x1	20.5	0.5	0.6	19.3	3.52	13.66	41.2	2.34	43.5	1.04
3	269	1	1x3	22.8	0.2	0.3	22.1	5.73	19.37	52.1	6.48	58.6	0.77
3	174	2	3x1	17.5	0.3	0.4	16.7	4.23	12.52	36.0	4.29	40.3	1.12
4	353	1	1x4	22.3	0.1	0.2	21.9	8.12	21.54	50.3	11.42	61.7	0.73
4	213	2	2x2	15.7	0.2	0.3	15.1	6.43	13.64	32.5	7.31	39.8	1.13
4	206	2	4x1	15.2	0.2	0.3	14.6	4.97	12.43	31.4	6.88	38.3	1.18
5	423	1	1x5	21.0	0.1	0.2	20.6	9.56	21.42	47.2	18.44	65.6	0.69
5	245	1	5x1	12.3	0.1	0.2	12.0	5.55	12.31	27.4	10.72	38.1	1.18
6	487	1	1x6	20.2	0.1	0.1	19.9	14.13	20.86	47.9	20.14	68.0	0.66
6	304	2	2x3	14.7	0.1	0.2	14.3	8.99	13.40	31.2	12.58	43.8	1.03
6	226	3	3x2	14.7	0.2	0.3	14.1	5.61	9.61	26.4	8.96	35.3	1.28
6	251	2	6x1	12.3	0.1	0.2	11.9	5.90	11.55	25.3	11.41	36.7	1.23

TABLE 4

MIC/ParPCG2 on 420 x 420 grid, on RS600/560, run time in seconds. reduce 2-norm of the non-preconditioned residual to 10e-7

fact we can not solve the problem on six processors if we order the unknowns along the longer dimension first. The table shows all the ways we could solve the problem up to six processors. The speedup sp2 and sp3 are computed using the same method as in Table 5. The total run time on six processors using a small bandwidth preconditioner is the best time among all the timings we got.

Table 7 shows the parallel MIC/ParPCG2 time for 420x420 2D general elliptic PDE. Note in the fraction column, the best fraction to use depends on the way the domain was cut.

Table 8 shows the time of parallel ParPCG2 using sequential MICPCG as preconditioner for 36x36x36 3D Laplace PDE. The fillin is for the sequential MICPCG. The iteration number refers the parallel iteration. Note all of the parallel iteration converged immediately. This is very special for the Laplace operator, and one should not expect such a nice global convergence in other problems.

Table 9 shows the parallel MIC/ParPCG2 time for 36x36x36 3D general elliptic PDE.

Table 10 shows the parallel MIC/ParPCG2 time for 60x60x60 3D general elliptic PDE.

Table 14 shows the parallel MIC/ParPCG2 time for 48x48x48 3D general elliptic PDE.

Table 15 shows the parallel MIC/ParPCG2 time for 96x96x96 3D general elliptic PDE.

The extrapolated sequential times in tables 15 and 16 assume the run time is a constant times $n^{3.5}\log_{10} n$, and the constant is determined by using the run time on a

p	itr	cut	pre	ord	fac	slv	dot	mvp	cmp	cmm	tot	sp	sp2	sp3
1	1	1x1	116.7	8.0	105.3	3.3	0.05	0.39	117.3	0.00	117.3	1.00	0.26	1.00
2	45	1x2	75.7	3.4	36.7	35.4	1.33	4.56	82.7	0.86	83.5	1.40	0.54	1.98
2	41	2x1	59.5	3.3	26.9	29.2	1.26	4.16	65.9	0.84	66.7	1.75	0.67	1.97
3	59	1x3	50.4	2.3	19.1	28.9	2.01	4.42	56.7	2.03	58.8	1.99	0.77	2.89
3	54	3x1	40.0	2.3	13.7	23.9	1.55	3.93	46.0	1.37	47.4	2.47	0.95	2.91
4	72	1x4	31.3	1.8	7.8	21.6	1.85	4.20	37.0	2.52	39.5	2.96	1.14	3.74
4	61	2x2	33.5	1.6	11.4	20.3	1.79	3.48	38.4	2.05	40.5	2.89	1.11	3.79
4	63	4x1	31.7	1.6	9.6	20.3	1.49	3.49	36.7	2.15	38.9	3.01	1.16	3.77
5	81	1x5	27.0	1.4	5.9	19.6	2.92	3.88	32.3	3.32	35.7	3.28	1.26	4.52
5	71	5x1	21.9	1.2	4.6	16.0	2.10	3.32	26.5	3.05	29.6	3.96	1.53	4.47
6	93	1x6	22.2	1.2	3.6	17.4	3.60	3.16	27.0	4.24	31.3	3.74	1.44	5.17
6	77	2x3	24.5	1.1	6.7	16.6	2.44	4.09	28.6	3.54	32.2	3.64	1.40	5.32
6	72	3x2	20.0	1.1	4.9	14.0	1.70	3.76	23.8	3.23	27.0	4.34	1.67	5.28
6	78	6x1	20.7	1.0	4.4	15.2	2.00	4.02	24.9	3.79	28.7	4.08	1.57	5.20

TABLE 5

Smpak/ParPcg2 on 420 x 420 grid, on RS600/560, run time in seconds. reduce 2-norm of the non-preconditioned residual to 10e-7

smaller problem that fits on one node.

Table 16 shows the parallel MIC/ParPCG2 time for 144x144x144 3D general elliptic PDE.

Table 17 shows the parallel MIC/ParPCG2 time for 420x420 2D general elliptic PDE.

Table 18 shows the parallel MIC/ParPCG2 time for 420x420 2D general elliptic PDE on SP1 using ethernet or switch.

Table 19 shows the parallel MIC/ParPCG2 time for 1320x1320 2D general elliptic PDE.

The Cray XMP time in table 19 is extrapolated as follows. We have run times on

p	itr	cut	pre	ord	fac	slv	dot	mvp	cmp	cmm	tot	sp2	sp3
3	55	3x1	62.8	0	16.3	46.5	1.47	4.10	68.6	1.57	70.2	0.64	2.93
4	61	2x2	81.5	0	25.8	55.7	1.43	3.64	86.6	2.16	88.8	0.51	3.90
4	63	4x1	35.0	0	6.0	29.0	1.76	3.62	39.9	2.33	42.2	1.07	3.78
5	72	5x1	24.6	0	3.0	21.6	1.78	3.38	29.7	2.82	32.5	1.39	4.56
6	77	2x3	63.8	0	17.1	46.6	1.86	3.66	68.0	3.47	71.5	0.63	5.70
6	72	3x2	38.4	0	8.1	30.2	2.07	3.38	42.1	3.20	45.3	1.00	5.57
6	78	6x1	17.8	0	1.8	15.9	2.29	3.48	21.8	3.77	25.6	1.76	5.10

TABLE 6

BandESSL/ParPcg2 on 420 x 420 grid, on RS600/560, run time in seconds. reduce 2-norm of the non-preconditioned residual to 10e-7

proc	iter	fillin	cut	fraction	comp	comm	total	speedup
1	61	3	1x1	not appl	45.3	0.00	45.3	1.00
2	110	2	2x1	0.97	34.1	2.21	36.3	1.24
3	140	1	3x1	0.97	27.2	3.45	30.6	1.48
4	148	1	4x1	0.97	21.0	5.15	26.2	1.72
5	156	1	5x1	0.96	18.3	7.33	25.6	1.76
6	160	1	3x2	0.95	15.2	7.00	22.2	2.04

TABLE 7

Parallel MIC/ParPCG2 on RS6000/560 for 420x420 PDE. reduce 2-norm of the residual to 10e-7

p	itr	pre itr	fill	cut	fraction	comp	comm	total	spup
1	32	0	1	1x1x1	not appl	12.88	0.00	12.88	1.00
2	1	31	1	1x1x2	1.00	6.00	0.26	6.27	2.05
4	1	29	1	2x2x1	1.00	2.78	0.44	3.22	4.00
8	1	25	1	2x2x2	1.00	1.24	0.46	1.71	7.53

TABLE 8

Parallel MICPCG/ParPCG2 on RS6000/340 for 36x36x36 Laplace. reduce 2-norm of the residual to 10e-7

a smaller problem on both the Cray and the IBM, and Cray XMP is about 1.5 times faster. We use this factor to approximate the Cray run time on this bigger problem.

The extrapolated sequential time in tables 19 assumes the run time is a constant times $n^{2.5}\log_{10} n$, and the constant is determined by using the run time on a smaller problem that fits on one node.

7. Concluding Remarks. We have generalized our block diagonal precondition theorem to more general elliptic PDEs. We have implemented our block diagonal PCG methods on a network of workstations.

The speedup with respect to the sequential MIC/PCG is not very close to linear. This shows that parallelizing sparse linear system solver in an environment with high

proc	iter	fillin	cut	fraction	comp	comm	total	speedup
1	25	0	1x1x1	not appl	8.30	0.00	8.30	1.00
2	34	0	1x1x2	0.80	5.51	0.98	6.49	1.27
3	35	0	1x1x3	0.70	4.28	1.55	5.84	1.42
4	36	0	4x1x1	0.76	3.20	2.16	5.36	1.54
5	39	0	5x1x1	0.70	3.20	2.39	5.59	1.48
6	40	0	1x1x6	0.80	2.19	3.22	5.38	1.54
7	42	0	1x1x7	0.70	1.92	3.62	5.54	1.49
8	39	0	2x2x2	0.70	1.78	3.32	5.10	1.62

TABLE 9

Parallel MIC/ParPCG2 on RS6000/340 for 36x36x36 PDE. reduce 2-norm of the residual to 10e-7

proc	iter	fillin	cut	fraction	comp	comm	total	sp3
5	50	0	1x1x5	0.80	14.96	6.66	21.62	3.45
6	54	0	1x6x1	0.80	13.65	7.05	20.71	3.95
7	56	0	1x1x7	0.76	12.44	7.26	19.70	4.42
8	53	0	2x2x2	0.80	9.65	6.89	16.54	4.66

TABLE 10

Parallel MIC/ParPCG2 on RS6000/340 for 60x60x60 PDE. reduce 2-norm of the residual to 10e-7

	iter # for different fillin				
fraction	fill=0	fill=1	fill=2	fill=3	fill=4
0	38	32	30	30	28
1	37	39	36	33	32
.30	34	31	30	30	29
.40	32	31	30	30	29
.60	31	32	30	30	29
.68	31	32	31	30	30
.69	31	32	31	29	29
.70	31	32	31	29	29
.75	31	32	31	29	29
.76	31	33	31	29	29
.79	31	33	31	29	29
.80	31	33	31	30	29
.90	33	34	32	30	29
cost per iter	1	1.03	1.09	1.20	1.40

TABLE 11

MIC/ParPCG2, general elliptic PDE on $24 \times 24 \times 24$ *grid, with* $2 \times 2 \times 2$ *cut, reduce 2-norm of the residual to 10e-7.*

communication cost is a very challenging job.

However, we do have very good scaled speedup. This shows the feasibility of solving very large sparse linear systems on a network of workstations without paying too much to the communication cost. Our approach is one way to aggregate all of the memories from different workstations and to solve a much bigger problem that does not fit on one node.

REFERENCES

[1] D. BAXTER, J. SALTZ, M. SCHULTZ, S. EISENSTAT, AND K. CROWLEY, *An experimental study of methods for parallel preconditioned keylov methods*, Department of Computer Science YALEU/DCS/RR-629, Yale University, June 1988.

[2] R. CHANDRA, *Conjugate Gradient Methods for Partial Differential Equations*, PhD thesis, Department of Computer Science, Yale University, 1978. Also available as Technical Report 129.

[3] M. Y.-M. CHANG AND M. H. SCHULTZ, *Bounds on block diagonal preconditioning*, Department of

fraction	iter # for different fillin				
	fill=0	fill=1	fill=2	fill=3	fill=4
0	56	47	42	38	37
1	53	53	50	46	43
-1	75	63	55	49	45
.60	41	40	39	38	38
.67	40	40	39	38	38
.68	39	40	39	38	38
.70	39	40	39	38	38
.71	39	40	39	38	38
.72	39	41	40	38	38
.75	40	41	40	38	37
cost per iter	1	1.06	1.17	1.45	1.88

TABLE 12

MIC/ParPCG2, general elliptic PDE on $36 \times 36 \times 36$ *grid, with* $2 \times 2 \times 2$ *cut, reduce 2-norm of the residual to 10e-7.*

Computer Science YALEU/DCS/RR-729, Yale University, August 1989.

[4] S. C. D. GELERNTER, N. CARRIERO AND S. CHANG, *Parallel programming in linda*, in Proc. Int. Conf. Parallel Processing, August 1985.

[5] IBM, *Engineering and Scientific Subroutine Libarary Guide and Reference Release 3*, IBM, November 1988.

[6] M. LEES, *Discrete methods for nonlinear two-point boundary value problems*, in Numerical Solution of Partial Differential Equations, J. H. Bramble, ed., Academic Press, 1966, pp. 59–72.

[7] Y. SAAD AND M. SCHULTZ, *Parallel implementations of preconditioned conjugate gradient methods*, Department of Computer Science YALEU/DCS/TR-425, Yale University, October 1985.

[8] SCA, *Smpak user's guide*, tech. report, Scientific Computing Associates, Inc., January 1990.

[9] G. W. STEWART, *Introduction to Matrix Computations*, Academic Press, New York, 1973.

fraction	iter # for different fillin				
	fill=0	fill=1	fill=2	fill=3	fill=4
0	94	78	69	60	55
1	81	80	75	69	65
.70	58	54	52	52	51
.78	54	55	53	52	51
.79	53	55	54	52	51
.80	53	55	54	52	51
.83	53	55	54	52	51
.84	53	56	55	53	52
.85	54	56	55	53	52
cost per iter	1	1.11	1.32	1.68	2.38

TABLE 13

MIC/ParPCG2, general elliptic PDE on $60 \times 60 \times 60$ *grid, with* $2 \times 2 \times 2$ *cut, reduce 2-norm of the residual to 10e-7.*

proc	iter	fillin	cut	fraction	comp	comm	total	speedup
1	30	0	1x1x1	0.70	12.93	0.0	12.93	1
8	47	0	2x2x2	0.71	2.54	2.78	5.32	2.43
27	55	0	3x3x3	0.71	1.10	4.82	5.92	2.18

TABLE 14

Parallel MIC/ParPCG2 on SP1 for 48x48x48 PDE. reduce 2-norm of the residual to 10e-7

proc	iter	fillin	cut	fraction	comp	comm	total	sp	sp3
1	extrapolated time using $O(n^{3.5}\log_{10} n)$						172.47	1	1
8	91	0	2x2x2	0.71	37.58	10.64	48.23	3.57	6.23
27	85	1	3x3x3	0.71	13.26	13.25	26.51	6.50	13.50

TABLE 15

Parallel MIC/ParPCG2 on SP1 for 96x96x96 PDE. reduce 2-norm of the residual to 10e-7

proc	iter	fillin	cut	fraction	comp	comm	total	sp	sp3
1	extrapolated time using $O(n^{3.5}\log_{10} n)$						776.24	1	1
27	125	1	3x3x3	0.71	60.75	37.69	98.44	7.88	16.66

TABLE 16

Parallel MIC/ParPCG2 on SP1 for 144x144x144 PDE. reduce 2-norm of the residual to 10e-7

proc	iter	fillin	cut	fraction	comp	comm	total	speedup
1	61	3	1x1	not appl	45.73	0.0	45.73	1
2	110	2	2x1	0.97	33.87	1.15	35.02	1.30
3	140	1	3x1	0.97	26.74	2.24	28.99	1.57
4	148	1	4x1	0.97	20.64	3.81	24.45	1.87
5	156	1	5x1	0.96	17.76	4.70	22.47	2.03
6	164	1	3x2	0.97	15.31	5.13	20.44	2.23
9	181	1	3x3	0.95	11.85	6.95	18.81	2.43
16	212	0	4x4	0.96	7.35	9.02	16.38	2.79
24	220	0	6x4	0.96	5.39	10.03	15.43	2.96

TABLE 17

Parallel MIC/ParPCG2 on SP1 for 420x420 PDE. reduce 2-norm of the residual to 10e-7

network	proc	iter	fillin	cut	fraction	comp	comm	total	sp
	1	61	3	1x1	not appl	45.73	0.0	45.73	1
ethernet	24	224	0	6x4	0.97	4.98	259.48	264.47	0.17
switch	24	224	0	6x4	0.97	5.70	10.55	16.26	2.81

TABLE 18

Parallel MIC/ParPCG2 on SP1 using ethernet or switch for 420x420 PDE. reduce 2-norm of the residual to 10e-7

proc	iter	fillin	cut	fraction	comp	comm	total	sp	sp3
1	extrapolated time using $O(n^{2.5}\log_{10} n)$						952.59	1	1
16	393	1	4x4	0.98	138.38	32.94	171.32	5.56	12.92
24	420	0	6x4	0.99	91.25	36.09	127.34	7.48	17.19
30	440	0	6x5	0.99	76.16	37.18	113.34	8.40	20.15
1	extrapolated time on Cray XMP						635.06	1.5	1.5

TABLE 19

Parallel MIC/ParPCG2 on SP1 for 1320x1320 PDE. reduce 2-norm of the residual to 10e-7