PARTIAL ROW PROJECTION METHODS

RANDALL BRAMLEY AND YOUNGHEE LEE
DEPARTMENT OF COMPUTER SCIENCE
INDIANA UNIVERSITY - BLOOMINGTON*

Abstract. Row projection (RP) methods are a class of algorithms for solving general linear systems. The defining characteristic of RP methods is that the matrix is partitioned into block rows, and then orthogonal projections onto the null space of the block rows are computed. Unlike most Krylov subpsace and Lanczos based methods, they are robust in theory, converging for any nonsingular system. However, three major problems still prevent their common use: computing the required orthogonal projections can be expensive for some matrix partitionings, preconditioning the RP system is difficult except for relatively ineffective preconditioners such as diagonal preconditioning from the right, and practical methods for partitioning the matrix are unknown except in special cases. This paper presents a new class of RP methods which address all three of those problems, while still converging for any nonsingular system.

Key Words. iterative methods, nonsymmetric linear systems, Kaczmarz, Cimmino, row projection

1. Introduction. Accelerated row projection (RP) algorithms for solving linear systems are a class of iterative methods which in theory converge for any nonsingular matrix. Definiteness, symmetry, or localization of the eigenvalues of the coefficient matrix are not required. In spite of this robustness, work has generally been limited to structured systems such as block tridiagonal matrices because unlike many iterative solvers, RP methods cannot be implemented by simply supplying a matrix-vector multiplication routine. This in turn is because RP methods require finding the orthogonal projection of vectors onto the null space of block rows of the matrix, which for practical problems requires accessing the actual entries in the matrix.

RP methods date back at least to the 1930's; see [3] for some of the more important bibliographic references. The basic ideas until now have all been based on the methods originally proposed by S. Kaczmarz [9] and G. Cimmino [7]. A. Björck and T. Elfving [2] proposed using CG acceleration of the basic RP schemes, and C. Kamath and A. Sameh [11] first proposed and developed methods that allow parallelism in the block tridiagonal case. They also examined various methods of computing the projections for the Kaczmarz scheme. R. Bramley and A. Sameh [4] provided a detailed analysis of the effects of different row partitionings for the seven point difference operator, and a theoretical comparison with CG on the normal equations which is summarized later in this paper.

Standard RP methods for solving $Ax = f$ begin by partitioning $A \in \Re^{N\times N}$ into m block rows: $A^T = [A_1, A_2, \ldots, A_m]$, and partitioning the vector f conformally. A *row projection* (RP) method is any algorithm which requires the computation of the orthogonal projections $P_i x = A_i(A_i^T A_i)^{-1}A_i^T x$ of a vector x onto range(A_i), $i = 1, 2, \ldots, m$. Note that the nonsingularity of A implies that A_i has full rank and so $(A_i^T A_i)^{-1}$ exists

* WORK SUPPORTED BY NSF GRANTS NSF CDA-9309746, CDA-9303189, AND ASC-9502292

for $i = 1, \ldots, m$. RP methods can also be used to solve least squares problems (in which case the block rows A_i^T need not be full rank), but this paper assumes nonsingularity of A. Two basic RP methods exist, involving the product and sum of projections respectively.

1.1. The Kaczmarz Row Projection Method. The simplest RP method is based on the method of successive projections:

$$x_{k+1} = (I - P_m)(I - P_{m-1}) \cdots (I - P_1)x_k + b_u, \tag{1}$$

where

$$\begin{aligned} b_u &= \hat{b}_m + (I - P_m)\hat{b}_{m-1} + (I - P_m)(I - P_{m-1})\hat{b}_{m-2} + \cdots \\ &\quad + (I - P_m) \cdots (I - P_2)\hat{b}_1, \end{aligned}$$

and $\hat{b}_i = A_i(A_i^T A_i)^{-1} f_i$. Because of the slow convergence of this iteration, Björck and Elfving [2] proposed symmetrizing the iteration by following a forward sweep through the rows with a backward sweep. Doing so gives the iteration

$$x_{k+1} = Q_\pi x_k + b_\pi \tag{2}$$

with

$$Q_\pi = (I - P_1)(I - P_2) \cdots (I - P_m)^2 \cdots (I - P_2)(I - P_1), \tag{3}$$

$b_\pi = T_\pi f$, and the i^{th} block column of T_π given by

$$\prod_{j=1}^{i-1}(I - P_j) \left[I + \prod_{k=i}^{m}(I - P_k) \prod_{j=m-1}^{i+1}(I - P_j) \right] (A_i^T)^\dagger. \tag{4}$$

When A is nonsingular the system $[I - Q_\pi]x = b_\pi$ is symmetric and positive definite and so the method of conjugate gradients (CG) can be applied to solve it. Note that each CG iteration will require $2m - 1$ orthogonal projections to be computed. It is possible to reduce this number to $2m - 2$, but this requires starting with the initial estimate x_0 in a certain affine set; see [4] for details.

1.2. The Cimmino Row Projection Method. The second basic RP method can be derived as a preconditioner for the CG algorithm. Premultiply the system $Ax = b$ by $\tilde{A} = [A_1(A_1^T A_1)^{-1}, A_2(A_2^T A_2)^{-1}, \ldots, A_m(A_m^T A_m)^{-1}]$ to obtain

$$(P_1 + P_2 + \cdots + P_m)x = \tilde{A}f. \tag{5}$$

For nonsingular A this system is symmetric positive definite and so again CG acceleration can be used. The advantage of this approach is that the projections can be computed in parallel and then added. Furthermore, note that each iteration requires m projections, about half that of Kaczmarz. However, the spectrum of the Cimmino system matrix $Q_\sigma = P_1 + P_2 + \cdots + P_m$ is generally less suitable for CG acceleration. For the two-block row case ($m = 2$), the spectrum of the Kaczmarz system consists of ones and the squares of the sines of the canonical angles between the subspaces range(A_1) and range(A_2), while the eigenvalues of the Cimmino system are $1 \pm c_i$, where c_i is the cosine of the i-th canonical angle. More generally, the Cimmino spectrum is less suitable for conjugate gradient acceleration, as was shown in [4].

2. Problems with RP Methods. Because the Cimmino spectrum is generally less favorable for CG acceleration than the Kaczmarz spectrum, in the remainder of this paper we concentrate on the product form of RP algorithms. In any case, the three major difficulties of using CG-accelerated Kaczmarz methods also occur for CG-accelerated Cimmino methods.

2.1. Computing Projections. The first problem with RP methods is the computation of the action of the orthogonal projectors on vectors, in particular solving subsystems with coefficient matrix $A_i^T A_i$. Since such systems are also symmetric and positive definite, an inner CG algorithm can be used to solve those subsystems. However this has been inefficient in practice because the projections need to be computed with high accuracy, for otherwise the product of projectors will numerically have some negative eigenvalues, causing failure in the outer CG acceleration. Furthermore, $2m-1$ projections are needed for each outer CG iteration and the total number of inner CG iterations tends to grow exponentially with problem size and accuracy.

All practical approachs to date solve the subsystems by performing a QR factorization [10] of A_i or by performing a Cholesky factorization of $A_i^T A_i$ (the normal equations method). The latter method is the one we use, because the technology of computing sparse Cholesky factorization is more advanced than that of sparse QR factorization, the projections computed this way have sufficient accuracy for practical problems, and the computational cost in time and storage is less than that required by QR factorization.

The success of using the normal equations depends critically on having sparse Cholesky factors R_i of $A_i^T A_i$, a problem that occurs also when using QR factorization methods. Although sparsity can be assured by choosing row partitionings that place only a few rows in each block row, as was shown in the last Section the outer CG accelerator requires fewer iterations if the block rows can be made large. This tradeoff between memory and reduced numbers of iterations occurs in all iterative methods, but for RP methods the balance is such that they often are too expensive for many problems.

2.2. Partitioning of A. Another difficulty with RP methods is, paradoxically, the complete flexibility in choosing row partitions. Some balance needs to be made for efficiency in computing the projections, but beyond this there is no theory to guide a user in how to select partitionings from the combinatorially large number of possibilities. For centered difference operators on quasi-uniform meshes, [5] shows a large number of possible choices and gives the tradeoffs in terms of storage and parallelism possible. However, the techniques in that work do not extend to general unstructured meshes for PDE's or other problems.

2.3. Preconditioning. Even after CG acceleration, RP methods are often too slow for many practical problems. Standard Krylov subspace methods which rely on matrix-vector products with the original matrix A can be made more robust in this case by using an incomplete factorization of A as a preconditioner. Increasing the amount of fill-in (and thus storage required) in the preconditioner can improve the robustness, since in the limiting case the incomplete factorization becomes a full Gaussian elimination.

This approach cannot be taken with the RP coefficient matrices since it is impractical to actually form the RP coefficient matrix, which severely limits the type of preconditioning that can be done.

Furthermore, preconditioning the original coefficient matrix A and *then* applying RP methods leads to an equally difficult practical problem. If the preconditioner is M, computing the block rows of $M^{-1}A$ is generally more expensive than just applying a Krylov subspace method to $M^{-1}A$. Preconditioning from the right, that is, applying the RP methods to AM^{-1}, is potentially implementable. Now the projectors become

$$\tilde{P}_i = M^{-T}A_i\left(A_i^T M^{-1}M^{-T}A_i\right)^{-1}A_i^T M^{-1}.$$

However, even if $A_i^T M^{-1}M^{-T}A_i$ is formed and factored in a preprocessing step, this requires applying the full preconditioner two times for each projector. For $m = 3$ block rows, this means each matrix-vector product in the CG-accelerated RP method requires 10 solves with M^{-1} or M^{-T}, clearly an impractical approach.

Arioli and Duff [1] proposed using right preconditioning with a diagonal matrix D. By choosing a partitioning into two block rows and making D have small diagonal components for the columns of A that have nonzeros in both A_1^T and A_2^T, they were able to reduce the number of CG iterations required. However, this leads to computing projections using submatrices that are ill-conditioned. Because they used direct methods for computing the orthogonal projections, this ill-conditioning could be ignored for some problems. However, this form of preconditioning is limited to reducing the dependencies between two block rows, and does not seem to be extensible to a practical method that can help precondition within a block row, or for problems with more than two block rows.

3. A Partial RP Method. The advantages of RP methods can be retained and the three problems addressed by introducing a new "partial" RP algorithm. This derivation considers only the $m = 2$ case. Consider the normal equations

$$\begin{aligned} AA^Ty &= \begin{bmatrix} A_1^TA_1 & A_1^TA_2 \\ A_2^TA_1 & A_2^TA_2 \end{bmatrix}\begin{pmatrix} u \\ w \end{pmatrix} = b && (6)\\ x &= A^Ty = A_1u + A_2w && (7)\end{aligned}$$

The Schur complement of the system has the form

$$\begin{aligned} [A_2^TA_2 - A_2^TA_1(A_1^TA_1)^{-1}A_1^TA_2]w &= \\ [A_2^T(I-P_1)A_2]w &= b_2 - A_2^TA_1(A_1^TA_1)^{-1}b_1 \end{aligned} \tag{8}$$

Note that $A_1(A_1^TA_1)^{-1} = (A_1^T)^\dagger$ is the Moore-Penrose pseudoinverse of A_1^T, so

$$\begin{aligned} x = A^Ty &= A_1u + A_2w \\ &= A_1(A_1^TA_1)^{-1}[b_1 - A_1^TA_2w] + A_2w \\ &= (A_1^T)^\dagger b_1 + (I-P_1)A_2w. \end{aligned} \tag{9}$$

Furthermore, the conjugate gradient algorithm can be applied to solve system (8), since the nonsingularity of A implies that the coefficient matrix is symmetric and positive definite. The vector $\hat{b}_1 = (A_1^T)^\dagger b_1$ is the minimum norm solution to the first block row of equations $A_1^T x = b_1$, and the second component $(I - P_1)A_2 w$ is the correction vector in null(A_1^T) to assure that the second block row $A_2^T x = b_2$ is satisfied. By initializing $x_0 = \hat{b}_1$ in the CG algorithm, w can be eliminated and x_k can be directly updated on each iteration k, giving:

Unpreconditioned Partial Row Projection Algorithm

Initialize: *Form and Cholesky factor $A_1^T A_1$ to get R_1.*

$$
\begin{aligned}
x_0 &= A_1 \, (R_1^T R_1)^{-1} \, b_1 \\
r_0 &= b_2 - A_2^T x_0 \\
d_0 &= (I - P_1) \, A_2 \, r_0 \\
\rho &= r_0^T r_0, \quad k = 1
\end{aligned}
$$

while *($\rho > \epsilon$ and $k < maxits$)*

$$
\begin{aligned}
s_k &= A_2^T d_{k-1} \\
\alpha &= \rho \, / \, d_{k-1}^T d_{k-1} \\
x_k &= x_{k-1} + \alpha \, d_{k-1} \\
r_k &= r_{k-1} - \alpha \, s_k \\
\gamma &= r_k^T r_k \\
\beta &= \gamma \, / \, \rho, \; \rho = \gamma. \\
d_k &= (I - P_1) A_2 \, r_k + \beta \, d_{k-1} \\
k &= k + 1
\end{aligned}
$$

endwhile

PROPOSITION 3.1. *For any nonsingular A, the unpreconditioned partial row projection algorithm gives iterates x_k such that $A_1^T x_k = b_1$ for all k. Furthermore, the update vectors d_k all satisfy $A_1^T d_k = 0$.*

Proof. Cholesky factorization gives $R_1^T R_1 = A_1^T A_1$, and so $A_1^T x_0 = A_1^T A_1 (R_1^T R_1)^{-1} b_1 = b_1$. Furthermore, $I - P_1$ is the orthogonal projector onto null(A_1^T), so $A_1^T(I - P_1) = 0$. That implies that $A_1^T d_0 = A_1^T (I - P_1) A_2 r_0 = 0$.

Now suppose the result holds for x_k and d_k. From $A_1^T d_k = 0$, $A_1^T x_{k+1} = A_1^T(x_k + \alpha d_k) = A_1^T x_k = b_1$. Also, $A_1^T d_{k+1} = A_1^T[(I - P_1)A_2 r_{k+1} + \beta d_k] = \beta A_1^T d_k = 0$, completing the proof by induction. □

Proposition 3.1 shows that the first block of equations is always satisfied. Since the partitioning between the two block rows is arbitrary above, equations which are particularly important can be placed into the first block row; for example, the constraint equations in a Lagrange multiplier system for constrained optimization. Furthermore, when the projections with $I - P_1$ are computed inexactly, perhaps with an inner iteration, the size of $A_1^T x_k - b_1$ can be used to monitor if and when it is necessary to restart the CG iteration.

One problem with iteratively solving systems by means of a derived system like (8) is that the size of the residual norm being updated may differ greatly in size from the residual of the original system. The next proposition shows that the residual norm being tracked by the partial row projection method is the *same* as that of the original system.

PROPOSITION 3.2. *In the unpreconditioned partial row projection algorithm,* $\|r_k\| = \|Ax_k - b\|$ *for all* k*, where the norm used is the 2-norm.*

Proof. The proof uses Proposition 3.1. Let $t_k = b - Ax_k$ be the residual of the original system. Since the first block of equations is satisfied for $k = 0$,

$$
\begin{aligned}
\|r_0\|^2 &= \|b_2 - A_2^T x_0\|^2 \\
&= \|b_2 - A_2^T x_0\|^2 + \|b_1 - A_1^T x_0\|^2 \\
&= \|t_0\|^2
\end{aligned} \tag{10}
$$

Now suppose that $\|r_k\| = \|t_k\|$. Then

$$
\begin{aligned}
\|t_{k+1}\|^2 &= \|b - Ax_{k+1}\|^2 \\
&= \|b_2 - A_2^T x_{k+1}\|^2 + \|b_1 - A_1^T x_{k+1}\|^2 \\
&= \|b_2 - A_2^T x_{k+1}\|^2 \\
&= \|b_2 - A_2^T x_k - \alpha A_2^T d_k\|^2 \\
&= \|r_k - \alpha s_k\|^2 \\
&= \|r_{k+1}\|^2
\end{aligned} \tag{11}
$$

The result follows by induction. □

4. Selecting Partitionings. In [5], criterion were listed for practical partitionings of the matrix A into block rows. The most important were to select partitionings that allowed parallelism in computing the orthogonal projections, and for which computing and storing the required Cholesky factors would involve an amount of storage that is linear in the number n of unknowns. The basic idea here is to partition the graph of the matrix into p subdomains using standard domain decomposition techniques. The partitioning is done by means of curves (in 2D) or manifolds (in 3D) that cut edges in the computational graph. Temporarily suppose that each node of the graph that has an edge cut is placed into block row A_1^T, and the remaining nodes are placed into block row A_2^T. If no node has an edge cut by more than one partitioning line, then

1. The normal equations $A_1^T A_1$ will be block diagonal, with the number of diagonal blocks equal to the number p of partitioning lines.
2. The trailing principal submatrix corresponding to the non-interface nodes will be block diagonal, with $p + 1$ diagonal blocks.

However, standard domain decomposition methods typically give partitioning lines that intersect, and some nodes will have edges cut by multiple partitioning lines. To allow parallelism in computing the needed projections, nodes corresponding to intersections of the partitioning lines in the graph are placed in a separate block row of A_2^T. Those are nodes which have edges that cross two or more partitioning cut–lines, and are called *cross–interface* nodes. Those which cross only a single partition cut line are called

Node Type	Block Row	Number of Subblocks	Number of Rows
Interface	A_1^T	12	1682
Non-Interface	A_2^T	8	18012
Cross–Interface	A_2^T	1	590

TABLE 1
Breakdown of Rows for $n = 20284$ Matrix

interface nodes. Figure 1 shows a uniform 12 x 12 mesh partitioned by a horizontal and vertical line into four subdomains. Figure 2 shows the domain with interface and cross–interface nodes colored black indicating their assignment to the rows of A_1^T. Since this would give just one block in the projector P_1, the central cross–interface nodes are placed as a final block row in A_2^T, as indicated in Figure 3. Note that now the interface nodes are decoupled, allowing four–way parallelism in computing P_1. Figure 4 shows the resulting nonzero structure of the matrix, with partition lines corresponding to the different blocks of A_1 and A_2.

In general A_1^T will have many fewer rows than A_2^T, consisting only of *interface* nodes. For 2D problems with $\mathcal{O}(N^2)$ nodes, A_1^T will have only $\mathcal{O}(N)$ rows and the number of cross–interface nodes will be $\mathcal{O}(1)$, while for 3D problems with $\mathcal{O}(N^3)$ nodes, A_1^T will have only $\mathcal{O}(N^2)$ rows and the cross–interface nodes will be $\mathcal{O}(N)$.

This simple modification of a domain decomposition based partitioning of the computational graph allows effective parallelism in both the multiplications with P_1 and A_2^T. Furthermore, even when using a compressed row storage scheme for A_2^T, multiplying a vector by A_2 can be done in parallel across all the subblocks except for the last cross–interface block. Finally, the reduced dimensionality of the set of interface nodes implies that a Cholesky factorization of the normal equations of A_1 can be used to compute the orthogonal projections at a cost in storage that grows sublinearly with the number of unknowns for 2D problems and linearly for 3D problems.

This modified graph partitioning approach can be used to enhance parallelism in any iterative method that requires both matrix–vector and transposed matrix–vector products, where the matrix is stored in a row or column based data structure, without requiring the use of an auxillary data structure. In particular it can be used for CG on the normal equations.

5. Preconditioning Partial RP Systems. A key advantage of the partial row projection system is that general preconditioning of the coefficient matrix is practical. In particular, if A_2^T corresponds to the majority of the matrix (which occurs if domain decomposition is used and A_1^T corresponds to the interface nodes), then we can precondition A_2^T. Note that if the preconditioner used is the Cholesky factor R_2 of $A_2^T A_2$, the resulting system has the same eigenvalues as the Kaczmarz system with two block rows. However, the advantage of partial RP is that more general (and robust) preconditioning can be performed. Using a two-sided version with a square, nonsingular preconditioner M for A_2^T, the preconditioned CG algorithm becomes

Preconditioned Partial Row Projection Algorithm

Initialize: *Form and Cholesky factor* $A_1^T A_1$ *to get* R_1.

$$x_0 = A_1 \, (R_1^T R_1)^{-1} \, b_1$$
$$r_0 = b_2 - A_2^T x_0$$
$$z_0 = M^{-T} r_0$$
$$d_0 = (I - P_1) \, A_2 \, M^{-1} z_0$$
$$\rho = z_0^T z_0, \;\; k = 1$$

while *(*$\rho > \epsilon$ *and* $k < maxits$*)*

$$s_k = A_2^T d_{k-1}$$
$$\alpha = \rho \, / \, d_{k-1}^T d_{k-1}$$
$$x_k = x_{k-1} + \alpha \, d_{k-1}$$
$$r_k = r_{k-1} - \alpha \, s_k$$
$$z_k = M^{-T} r_k$$
$$\gamma = z_k^T z_k$$
$$\beta = \gamma \, / \, \rho, \; \rho = \gamma.$$
$$d_k = (I - P_1) A_2 \, M^{-1} z + \beta \, d_{k-1}$$
$$k = k + 1$$

endwhile

Two-sided preconditioning is used here because essentially only the matrix A_2 is visible. Attempting to form a preconditioner for the entire coefficient matrix $A_2^T(I - P_1)A_2$ is not practical, since in general it will be a large dense matrix even when A is sparse. If the domain decomposition method of section 4 is used to partition the matrix, then ignoring the interface edges of A_2 gives a block diagonal preconditioner which can be applied in parallel.

PROPOSITION 5.1. *The preconditioned partial row projection algorithm gives iterates* x_k *which satisfy* $A_1^T x_k = b_1$ *for all* k. The proof is identical to that of Proposition 3.1.

6. Testing Results. Testing has been carried out to address the following questions:

1. How does the partial row projection (PRP) method compare with existing RP methods such as CG accelerated Kaczmarz and Cimmino methods, in robustness, memory usage, efficiency, and parallelism?
2. What are the effects of increasing the number of partition sets in the domain decomposition phase?
3. The proposed method uses only the diagonal blocks and ignores the interface and cross–interface edges in defining the preconditioner for A_2^T. What is the effect of discarding those off–block diagonal elements on the quality of preconditioning?

NOTE: these testing results are preliminary in this version of the paper, and further tests are being carried out.

Tests were carried out on a 12-processor SGI Power Challenge, with 75 MHz R8000 chips. Each chip has 16 kbytes of primary and 4 Mbytes of secondary data cache, and global memory is 2 Gbytes. The SGI system is at the Computer Science Department at Indiana University, and was running version 6.1 of the IRIX64 operating system.

Three problems were tested, with various partitionings. The first problem (CFD), comes from a computational fluid dynamics problem and has $n = 20284$ unknowns. The problem involves a highly refined finite element mesh for the backward facing step problem, and solves for steady state flow with Reynold's number of 100. PDE1 and PDE4 are systems that occur when using seven–point centered differences in solving PDE's on the unit cube. PDE1 solves

$$u_{xx} + u_{yy} + u_{zz} + 1000xu_x = F \tag{12}$$

with predetermined solution vector $u = xyz(1-x)(1-y)(1-z)$, and PDE4 solves

$$(e^{xyz}u_x)_x + (e^{-xyz}u_y)_y + (e^{-xyz}u_z)_z - 250(x+y+z)u_x - 250[(x+y+z)u]_x - \frac{u}{1+x+y+z} = F \tag{13}$$

with predetermined solution vector $u = \exp(xyz)\sin(\pi x)\sin(\pi y)\sin(\pi z)$. Both problems were tested with two domain partitionings: the first uses a single plane to bisect the unit cube in each coordinate direction, while the second uses two planes in each direction. Both PDE1 and PDE2 were discretized to have $n = 110592$. Although PDE1 and PDE2 give a regularly structured matrix, the codes use a general compressed sparse row matrix data structure and do not take advantage of the seven diagonal structure.

The methods tested were the partial row projection (PRP) method with no preconditioning (NoPC), ILU(0), and ILU(2) preconditioners. These were compared with the original CG-accelerated Kaczmarz method (KACZ) using both the modified domain decomposition (DD) method described earlier, and with the original RP-9 partitioning [4]. The stopping test was $\|r_k\| < 10^{-6}\|r_0\|$, where $r_k = b - Ax_k$. A maximum of 5000 iterations was allowed, which was reached in only one case: PRP-NoPC applied to problem CFD.

Table 2 shows the memory required to solve the problems in millions of integer words, and Table 3 shows the additional storage as a multiple of that needed to represent A. The domain decomposition matrices require entirely too much memory for the original CG–accelerated KACZ method, and this is why relatively small problems were used. PRP has successfully solved problems with one million unknowns. KACZ requires forming and factoring the normal equations for entire subdomains, which is why for the same partitioning as PRP, it requires much more storage.

Table 4 shows the numbers of iterations required to solve the problem. Using preconditioning does bring down the numbers of iterations by as much as a factor of four. Interestingly, increasing the number of levels of fill in the preconditioner to two does not have as significant of an effect. This is possibly because the preconditioning is being applied to the diagonal blocks of A_2 alone, ignoring the contributions from interface nodes. Increasing the degree of approximation of something that is already an approximation does not help much here. The results using the original Kaczmarz method bear this out; the numbers of iterations required there are about a third of those required for PRP-ILU(2). Recall that the original Kaczmarz method could be seen as the case where optimal preconditioning is applied to A_2^T, in the sense that premultiplying by the Cholesky factor makes the rows of A_2^T orthonormal. Table 5 shows the elapsed time for the solution, including forming Cholesky factors, preconditioners, and iterating. The results show that the "optimality" of KACZ is in terms of iteration counts only, since it requires up to 44 times longer to solve the problem, and in the best case (CFD-ILU(2)) it is still twice as slow. This is partly because of the large amounts of data involved, and the computations require solving large triangular systems corresponding to the Cholesky factor of A_2^T, which inhibits pipelining. Note that for the second partitioning method (problems PDE1(2) and PDE4(2)), KACZ performs significantly better. Since the second partitioning creates 27 instead of 8 subdomains, the Cholesky factors are more numerous but smaller, giving iterations that are faster by a factor of four. Also note from Table 4 that PRP-NoPC fails for problem CFD with a maximum number of iterations reached. However, further testing has shown that it does converge when 30,000 iterations are allowed. Although that is an excessive number, even then PRP-NoPC required less *time* to solve the problem than KACZ with the same partitioning. Table 6 shows the number of blocks in A_1 and A_2 and the time required when using eight processors. Note that A_2 has one more block than the number of subdomains induced by the partitioning, because of the cross-interface block. The number of parallel tasks is also one less for the transposed matrix–vector products than for the matrix–vector products because of the cross–interface blocks. The speedups ranged from 4.7 to 6.8, compared to single processor times. Note that this is a fair definition of speedups since PRP also gave the best single processor times. The parallel efficiency of up to 85% is surprising, since the numbers of parallel tasks in the different phases are not multiples of the number of processors used.

7. Summary and Conclusions. A new class of iterative algorithms called partial row projection methods was introduced, analyzed, and tested. This class addresses the three major roadblocks that have until now prevented row projection methods from being used as practical solvers. First, the choice of row partitionings can be naturally defined using a simple modification of standard graph partitioning methods; this modification has broader possibilities since it allows parallelism in both matrix–vector and transposed matrix–vector products. Secondly, both the storage and computational cost of computing the projections can be controlled. Thirdly, standard preconditioners such as ILU can be applied to the system, greatly enhancing convergence. All of these desirable properties are achieved while still retaining the robustness and parallelism of RP methods. Testing shows that PRP takes significantly less time than the original CG-accelerated Kaczmarz method required.

Even when preconditioned, the PRP iteration allows certain equations in the linear system to be kept satisfied in exact arithmetic. This allows a user to keep particularly important equations such as constraints satisfied on every iteration, and provides a useful computational check on the accumulation of errors in finite precision arithmetic.

REFERENCES

[1] M. Arioli, I. Duff, J. Noailles, and D. Ruiz, *A block projection method for general sparse matrices*, SIAM Journal of Scientific and Statistical Computing, 13 (1992), pp. 47–70.

[2] A. Björck and T. Elfving, *Accelerated projection methods for computing pseudo-inverse solutions of systems of linear equations*, BIT, 19 (1979), pp. 145–163.

[3] R. Bramley, *Row Projection Methods for Linear Systems*, PhD thesis, University of Illinois Urbana-Champaign, 1989. Also available as Tech. Rep. 881, Center for Supercomputing Research and Development, University of Illinois – Urbana.

[4] R. Bramley and A. Sameh, *Row projection methods for large nonsymmetric linear systems*, SIAM J. Sci. Stat. Comp., 13 (1992), pp. 168–193.

[5] ——, *Domain decomposition for parallel row projection algorithms*, Applied Numerical Mathematics, 8 (November 1991), pp. 303–315.

[6] G. Cimmino, *Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari*, Ric. Sci. Progr. tecn. econom. naz., 9 (1939), pp. 326–333.

[7] S. Kaczmarz, *Angenäherte auflösung von systemen linearer gleichungen*, Bull. intern. Acad. polonaise Sci. lettres (Cracouie); Class sci. math. natur.: Seira A. Sci. Math., (1939), pp. 355–357.

[8] C. Kamath, *Solution of nonsymmetric systems of equations on a multiprocessor*, Tech. Rep. 591, Center for Supercomputing Research and Development, University of Illinois, Urbana, Illinois, 1986.

[9] C. Kamath and A. Sameh, *A projection method for solving nonsymmetric linear systems on multiprocessors*, Parallel Computing, 9 (1988), pp. 291–312.

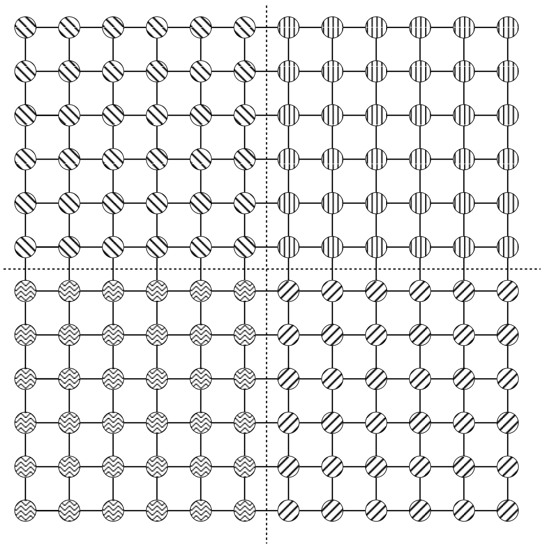

FIG. 1. *Decomposition of* 12×12 *mesh with two partitioning (dotted) lines*

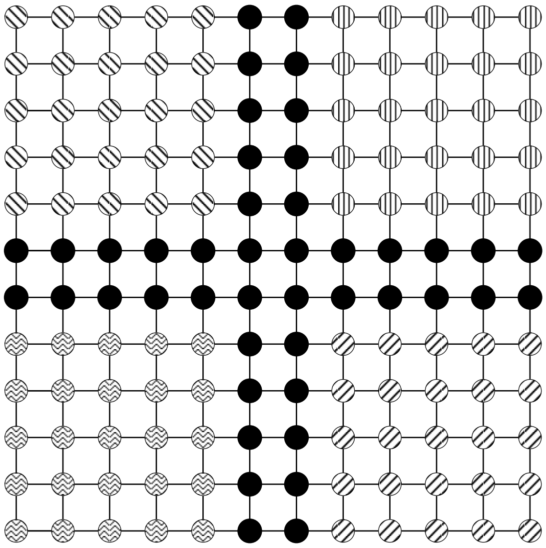

FIG. 2. *Assignment of interface nodes (black) to* A_1

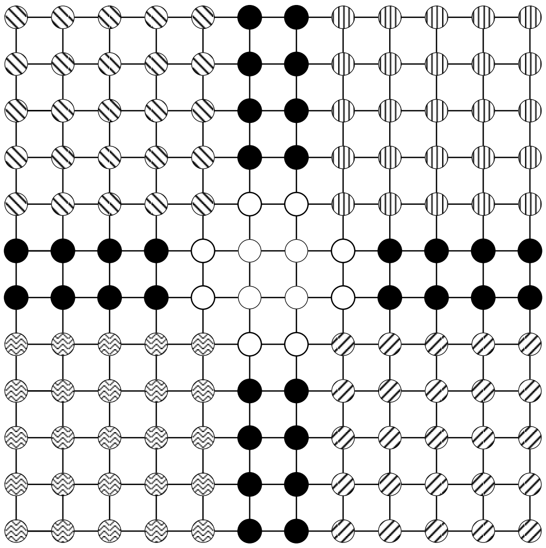

FIG. 3. *Decoupling of interface nodes into four subblocks by reassigning nodes corresponding to partition line intersections to* A_2

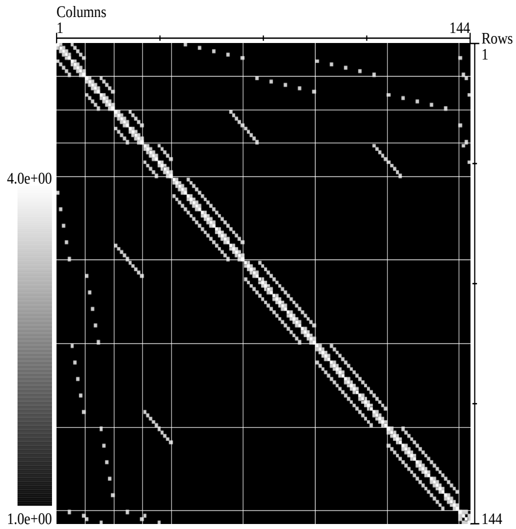

FIG. 4. *Matrix nonzero structure corresponding to decoupled interface nodes; last row of* A_2^T *corresponds to white nodes in Figure 3*

	KACZ RP-9	KACZ DD	PRP-NoPC	PRP-ILU(0)	PRP-ILU(2)
PDE1(1)	12.31	120.27	4.96	6.95	10.96
PDE1(2)		48.32	5.21	7.00	10.39
PDE4(1)	12.31	120.27	4.96	6.95	10.96
PDE4(2)		48.32	5.21	7.00	10.39
CFD		6.54	1.68	2.83	3.61

TABLE 2
Total memory usage in integer Mwords

	KACZ RP-9	KACZ DD	PRP-NoPC	PRP-ILU(0)	PRP-ILU(2)
PDE1(1)	3.84	51.18	0.62	1.49	3.25
PDE1(2)		19.63	0.73	1.52	3.00
PDE4(1)	3.84	51.18	0.62	1.49	3.25
PDE4(2)		19.63	0.73	1.52	3.00
CFD		3.80	0.10	0.98	1.57

TABLE 3
Additional storage required as a multiple of that required for A

	KACZ RP-9	KACZ DD	PRP-NoPC	PRP-ILU(0)	PRP-ILU(2)
PDE1(1)	22	52	340	196	151
PDE1(2)		69	335	222	177
PDE4(1)	51	51	884	182	169
PDE4(2)		66	840	200	194
CFD		1536	MX	3680	3433

TABLE 4
Number of iterations required

	KACZ RP-9	KACZ DD	PRP-NoPC	PRP-ILU(0)	PRP-ILU(2)
PDE1(1)	3.59466E+03	8.65438E+03	2.06203E+02	2.05510E+02	2.67465E+02
PDE1(2)		2.80583E+03	2.07738E+02	2.17571E+02	2.66986E+02
PDE4(1)	3.67396E+03	8.62586E+03	4.80386E+02	1.93121E+02	2.93962E+02
PDE4(2)		2.76514E+03	4.84132E+02	1.97546E+02	2.88854E+02
CFD		2.70134E+03	7.15072E+02	1.18491E+03	1.48228E+03

TABLE 5
Total execution time in seconds

	A_1^T block	A_2^T block	PRP-NoPC	PRP-ILU(0)	PRP-ILU(2)
PDE1(1)	12	9	4.01895E+01	4.13579E+01	5.58675E+01
PDE1(2)	54	28	3.47119E+01	3.40292E+01	3.98970E+01
PDE4(1)	12	9	9.69896E+01	3.95226E+01	6.27103E+01
PDE4(2)	54	28	8.21345E+01	3.10645E+01	4.28983E+01
CFD	11	9	1.24730E+02	2.21376E+02	2.77686E+02

TABLE 6
Number of parallel tasks and total execution time on 8 processors