SIAM J. SCI. COMPUT. © 2005 Society for Industrial and Applied Mathematics
Vol. 27, No. 3, pp. 1092–1117

COMPONENT-AVERAGED ROW PROJECTIONS: A ROBUST, BLOCK-PARALLEL SCHEME FOR SPARSE LINEAR SYSTEMS*

DAN GORDON† AND RACHEL GORDON‡

Abstract. A new method for the parallel solution of large sparse linear systems is introduced. It proceeds by dividing the equations into blocks and operating in block-parallel iterative mode; i.e., all the blocks are processed in parallel, and the partial results are "merged" to form the next iterate. The new scheme performs Kaczmarz row projections within the blocks and merges the results by certain *component-averaging* operations—hence it is called component-averaged row projections, or CARP. The system matrix can be general, nonsymmetric, and ill-conditioned, and the division into blocks is unrestricted. For partial differential equations (PDEs), if the blocks are domain-based, then only variables at the boundaries between domains are averaged, thereby minimizing data transfer between processors. CARP is very robust; its application to test cases of linear systems derived from PDEs shows that it converges in difficult cases where state-of-the-art methods fail. It is also very memory efficient and exhibits an almost linear speedup ratio, with efficiency greater than unity in some cases. A formal proof of convergence is presented: It is shown that the component-averaging operations are equivalent to row projections in a certain superspace, so the convergence properties of CARP are identical to those of Kaczmarz's algorithm in the superspace. CARP and its convergence proof also apply to the consistent convex feasibility problem.

Key words. block-parallel, component-averaging, convex feasibility problem, domain decomposition, linear equations, parallel processing, partial differential equations, row projections, sparse linear systems, superspace

AMS subject classifications. 15A06, 65F10, 65F50, 65N12, 65Y05, 90C51

DOI. 10.1137/040609458

1. Introduction.

Iterative methods for solving large sparse linear systems of equations are advantageous over the classical direct solvers, especially for huge systems. Methods of parallelizing iterative algorithms [20, 29] divide the equations into blocks and usually fall into one of two modes of operation. In the *block-sequential* (also called block-iterative) mode, the blocks are processed sequentially, but the computations on each block's equations are done in parallel. Examples of block-sequential methods are found in [1, 6, 10]. In the second mode of operation, sometimes referred to as *block-parallel*, the blocks are assigned to different processors to be processed in parallel, and the results from all the blocks are then combined in some manner to produce the next iterate. Typical examples of block-parallel schemes are parallel versions of Bi-CGSTAB [35], RGMRES [30], conjugate gradient (CG), and CG-squared (CGS) [31]; other examples include those in [2, 4, 9, 12, 13].

Kaczmarz's row projection method (KACZ) [23] was one of the first iterative methods used for large nonsymmetric systems. Its main advantages are robustness, guaranteed convergence on consistent systems, and cyclic convergence on inconsistent systems [8, 16, 32]. KACZ was also independently discovered in the context of image reconstruction from projections, where it is called ART (algebraic reconstruction technique) [21]. Kaczmarz's algorithm, by its nature and mathematical definition, is inherently sequential since, at each iterative step, the current iterate is projected

*Received by the editors June 4, 2004; accepted for publication (in revised form) April 21, 2005; published electronically December 30, 2005.
http://www.siam.org/journals/sisc/27-3/60945.html

†Department of Computer Science, University of Haifa, Haifa 31905, Israel (gordon@cs.haifa.ac.il).

‡Department of Aerospace Engineering, The Technion–Israel Institute of Technology, Haifa 32000, Israel (rgordon@tx.technion.ac.il).

onto the hyperplane defined by the next equation. In the context of partial differential equations (PDEs), current approaches to parallelizing KACZ have taken the block-sequential approach by replacing row projections with block projections and processing the blocks sequentially; see [6, 7, 24].

In this paper we present a block-parallel method of parallelizing Kaczmarz's algorithm for PDEs without block projections. Our approach stems from domain-decomposition considerations: After partitioning the domain into subdomains and assigning subdomains to processors, each processor is assigned the block of equations whose central node lies in its subdomain(s). Starting from some initial estimate, each processor performs a sequence of row projections on the hyperplanes defined by its block's equations, as in KACZ, and the results from the various blocks are then "merged" together to form the next iterate. The merge operation takes variables, which belong to two or more blocks, and replaces their different values (in the different domains) by their average. Thus, only variables bordering subdomain boundaries are averaged, so communication between processors is limited. Domain boundaries can even overlap, which means that the blocks of equations need not be disjoint.

We call this algorithm *component-averaged row projections*, or CARP. The principle of component-averaging also appeared, in a different form, in the CAV and BICAV algorithms of Censor, Gordon, and Gordon in [11, 10]. The main idea there was to utilize the sparsity of the system matrix of image-reconstruction problems. Note that the "string-averaging" algorithm [9, 12] also performs block-parallel row projections, but the resulting vectors of the separate blocks are combined by taking their regular (weighted) average.

CARP is completely general and places no restriction on the system matrix or the selection of the blocks. A formal proof of convergence is presented. It relies on a novel result which shows that the averaging operations are equivalent to row projections in a certain superspace. From this, it follows that the convergence properties of CARP are identical to those of Kaczmarz's algorithm in the superspace, and that CARP can also be used for nonsquare systems. CARP and its convergence proof, which uses a result of Aharoni and Censor [1], are also valid for the consistent case of the convex feasibility problem.

CARP provides a very robust parallel iterative method for solving large sparse linear systems derived from PDEs. It converges in difficult cases where state-of-the-art parallel versions of RGMRES, CGS, and Bi-CGSTAB fail. Since CARP is a linear iterative method, its convergence rate is theoretically limited; see [20, sec. 3.2.5]. Hence, it is not as efficient as some of the nonlinear methods to which it is compared in this paper. However, some of the nonlinear methods fail to converge on several of the test cases. In terms of its parallel behavior, CARP exhibits an almost linear speedup ratio, as well as efficiency greater than unity on some of the test cases.

CARP is memory efficient, compared to block-based projection methods such as those in [2, 3, 6, 7] or Krylov-based methods, since each processor operates on its equations one at a time. The memory requirement for each processor is just the submatrix of its local equation coefficients and the local part of the iterated solution vector; there is no need to store any other data related to a domain's submatrices, as in other methods. Furthermore, for many problems, there is no need even to store the equation coefficients since they can be calculated "on the fly" as needed. Another advantage is that there is no need to preprocess the system matrix, so CARP is suitable for quasi-linearization methods for nonlinear problems, in which the system matrix is modified repeatedly.

The rest of the paper is organized as follows. Section 2 places our work in relation to previous work. Section 3 presents the mathematical background, the formulation of CARP, the convergence proofs, and some extensions. Section 4 presents numerical results comparing the new approach with state-of-the-art parallel techniques, and section 5 concludes with a discussion and future research directions.

2. Relation to previous work.

Block projection methods (see Elfving [17]) use block projections instead of the row projections of KACZ [23] or Cimmino's algorithm [14]. Hence, such methods require computation and storage of data related to the inverses of submatrices. This data consists of Cholesky factors of the normal equation matrices or the LU factors of the augmented system matrices. As mentioned, this gives CARP an advantage with regard to memory and applicability to nonlinear systems.

Bramley and Sameh [6], following Kamath and Sameh [24], parallelized block KACZ with CG-acceleration under five partitioning schemes on regular grids. The method was very robust, but there was no clear-cut "best" partitioning scheme: The RP9 scheme produced the fastest runtime results on most of the test problems, but it requires inverse submatrix data, and it is unsuitable for unstructured grids. The RP27 scheme does not need submatrix data, but it performed best on only one problem and it is suitable only for structured grids. The cube-4 scheme is adaptable to unstructured grids, but it also uses data related to submatrix inverses, and its runtime is the worst or second worst of the schemes tested.

In [7], the above approach was extended to include also CG-acceleration of block-Cimmino and a new projection method called V-RP. These methods, using the RP9 scheme from [6], were compared with RGMRES and CGNE, both with and without preconditioning. Arioli et al. [2] studied parallel CG-acceleration of block-Cimmino for different block partitionings. Their study was restricted to block-tridiagonal systems, though their technique may be applicable to other cases. This method also requires data related to submatrix inverses, with the above disadvantages. In [3], this research was extended to include block-Lanczos acceleration, which improved the runtime performance (over the CG-acceleration) for some ill-conditioned systems.

Three of the most successful and commonly used parallel techniques for large sparse nonsymmetric systems are RGMRES [29], CGS [31], and Bi-CGSTAB [35]. These methods, which can be used with or without preconditioners, often converge faster than the (linear) projection methods. However, they are not as robust as row projection methods and fail in some difficult cases.

As mentioned, the convergence proof of CARP relies on extending the Euclidean space into a certain superspace. Pierra [27, 28] introduced the concept of extending the Euclidean space into a product space, and this idea was used by Combettes [15] to prove the convergence of Cimmino's algorithm. In the product space extension, each coordinate is "extended" to a fixed number of coordinates in the superspace. Our technique of a superspace extension is a generalization of the product space concept, allowing extensions of only some of the coordinates, as well as different-sized extensions for different coordinates.

3. Mathematical development.

3.1. Background, motivation, and presentation of CARP.

Consider the following system of m linear equations in n variables:

$$\sum_{j=1}^{n} x_j a_j^i \;=\; b_i \quad \text{for} \quad 1 \le i \le m, \qquad \text{or, in matrix form:} \;\; Ax \;=\; b. \tag{1}$$

In KACZ, each iterate is projected onto a hyperplane (defined by one of the equations) in a cyclic manner. KACZ has been studied very extensively, both theoretically and experimentally. Its convergence with relaxation parameters, for consistent systems, has been shown by Herman, Lent, and Lutz [22] and by Trummer [33]. Tanabe [32] proved that when the system is inconsistent, it converges *cyclically*; i.e., for each hyperplane, the sequence of projections on that hyperplane converges to a limit. See also [8] for further results in that direction.

Throughout the rest of the paper we assume that every column of A is nonzero. Such a column, if it exists, can be removed as a preliminary step since it corresponds in effect to coefficients of a "fictitious" variable, i.e., one whose value can be arbitrary. For $1 \leq i \leq m$, we denote by a^i the ith row vector of the matrix A in (1) and by $\langle u, v \rangle$ the dot product of two vectors u, v. KACZ can be described as follows: Starting from an arbitrary point x^0 in the n-dimensional Euclidean space, the kth iterate x^k is projected toward the next hyperplane, and the hyperplanes are chosen in cyclic order. To simplify our notation, we denote by $i(k)$ the kth index taken cyclically from 1 to n, i.e., $i(k) = (k \bmod m) + 1$. The regular KACZ algorithm (with relaxation parameters) is the following.

ALGORITHM 1 (KACZ).

Initialization: $x^0 \in \mathbb{R}^n$ *is arbitrary.*

Iterative step: *Given* x^k, *compute*

$$x^{k+1} \;=\; x^k \;+\; \lambda_k \frac{b_{i(k)} - \langle a^{i(k)}, x^k \rangle}{\|a^{i(k)}\|^2} a^{i(k)} \;. \tag{2}$$

If we assume that the equations are normalized, i.e., for $1 \leq i \leq n$, the ith equation is divided by $\|a^i\|$, then (2) can be replaced by

$$x^{k+1} \;=\; x^k \;+\; \lambda_k \left(b_{i(k)} - \langle a^{i(k)}, x^k \rangle \right) a^{i(k)} \;. \tag{3}$$

It is easy to see that the sequence of points produced by KACZ is identical to the one obtained by first normalizing the equations and replacing (2) by (3).

The motivation for CARP was to parallelize KACZ for PDEs using domain decomposition, but without using block projections with their inherent limitations. This led to a block-parallel method in which the domain is divided into subdomains (possibly overlapping), which can be assigned to different processors. Each processor is assigned the block of those equations whose central node lies in its subdomain(s). Starting from some initial estimate, the following two main steps are repeated until convergence: In the first step, consecutively executed row projections are performed in every block (as in KACZ) on all the equations of the block; in fact, more than one sweep can be carried out. In the second step, every variable shared by two or more blocks is replaced by the average of its values in the separate blocks. Thus, only variables bordering a neighboring domain are averaged. The above approach is formalized as follows.

The equations of the linear system (1) are divided into blocks, $B_1, \ldots, B_t$, which are not necessarily disjoint. For each $1 \leq j \leq n$, denote by I_j the index set of the blocks which contain an equation with a nonzero coefficient of x_j; i.e., $I_j = \{1 \leq q \leq t \mid x_j$ has a nonzero coefficient in some equation of $B_q\}$. Let $s_j = |I_j|$ (the size of I_j). The following definition explains and formalizes the merge operation of CARP.

DEFINITION 1. *Let* $B = \{B_1, \ldots, B_t\}$ *be as above. The component-averaging operator relative to* B *is a mapping* $\mathrm{CA}_B : (\mathbb{R}^n)^t \to (\mathbb{R}^n)$, *defined as follows: Let*

$x^1, \dots, x^t \in \mathbb{R}^n$. Then $\mathrm{CA}_B(x^1, \dots, x^t)$ is the point in $\mathbb{R}^n$ whose jth component is given by

$$\mathrm{CA}_B(x^1, \dots, x^t)_j = \frac{1}{s_j} \sum_{q \in I_j} x_j^q ,$$

where x_j^q is the jth component of x^q, for $1 \le q \le t$.

We can now present the CARP algorithm. In the following, given a block of equations B_q and a point $x \in \mathbb{R}^n$, we use the term "KACZ sweep" to refer to the operation of successively projecting x onto the hyperplanes defined by the equations of B_q, as in KACZ (Algorithm 1); this is formalized as follows.

Definition 2. *Let B_q, $1 \le q \le t$, be a block of equations of the system* (1), *and let $\{i_1, \dots, i_r\}$ be the set of indices of B_q's equations. For $x \in \mathbb{R}^n$, we define an operator* $\mathrm{KSWP}(B_q, x) : \mathbb{R}^n \to \mathbb{R}^n$ *as follows. $x^0 = x$, and for $0 \le j < r$, define*

$$x^{j+1} = x^j + \lambda_j \frac{b_{i_j} - \langle a^{i_j}, x^j \rangle}{\|a^{i_j}\|^2} a^{i_j} ,$$

where the λ_j's are relaxation parameters. Then $\mathrm{KSWP}(B_q, x) = x^r$.

Algorithm 2 (CARP).

Initialization: *$x^0 = (x_1^0, \dots, x_n^0) \in \mathbb{R}^n$ is arbitrary.*

Iterative step: *Given x^k, do:*

1. *For every $1 \le q \le t$, in parallel, execute some finite number of KACZ sweeps on all the equations of B_q. For every $1 \le q \le t$, denote the resulting point by $\bar{x}^q$. Formally, for some $p_k \ge 1$, $\bar{x}^q = \mathrm{KSWP}^{p_k}(B_q, x^k)$.*
2. *$x^{k+1} = \mathrm{CA}_B(\bar{x}^1, \dots, \bar{x}^t)$.*

Appendix A provides a detailed implementation of CARP, allowing separate processors for the different blocks.

3.2. Convergence proofs.

The proof of convergence of CARP proceeds along the following lines. We first transform the system (1) into a system of equations in some superspace $\mathbb{R}^s$of $\mathbb{R}^n$. In $\mathbb{R}^s$, the equations belonging to different blocks do not share any common variables, so the parallel processing of the separate blocks is equivalent to regular sweeps of KACZ in $\mathbb{R}^s$. We then prove a general *averaging lemma*, according to which, operations of component averaging are equivalent to certain row projections. From this it follows that CARP is just KACZ in $\mathbb{R}^s$.

Assume w.l.o.g. that for some $1 \le r \le n$ the variables $x_1, \dots, x_r$ are exactly the ones shared by two or more blocks, i.e., $s_1, \dots, s_r \ge 2$ and $s_{r+1} = \cdots = s_n = 1$. Denote $s = \sum_{j=1}^n s_j$, and consider the space $\mathbb{R}^s$. In order to simplify our notation, we will use y instead of x for vectors and components of $\mathbb{R}^s$, and index vectors of $\mathbb{R}^s$ as follows:

$$y = \left(\underbrace{y_{1,1}, \dots, y_{1,s_1}}_{s_1 \text{ elements}}, \;\dots\;, \underbrace{y_{r,1}, \dots, y_{r,s_r}}_{s_r \text{ elements}}, \underbrace{y_{r+1}, \dots, y_n}_{n-r \text{ elements}} \right) .$$

We now define the following *expansion mapping* $E : \mathbb{R}^n \to \mathbb{R}^s$:

$$E(x_1, \dots, x_n) = (y_{1,1}, \dots, y_{1,s_1}, \;\dots\;, y_{r,1}, \dots, y_{r,s_r}, \; y_{r+1}, \dots, y_n) ,$$

where $y_{j,1} = \cdots = y_{j,s_j} = x_j$ for $1 \le j \le r$ and $y_j = x_j$ for $r < j \le n$.

Each I_j (the index set of the blocks which contain a nonzero coefficient of x_j) has s_j elements, so it can be represented as $I_j = \{q_{j,1}, q_{j,2}, \dots, q_{j,s_j}\}$. For every $1 \le j \le n$ and $1 \le \ell \le s_j$, variables of the form $y_{j,\ell}$ will be used to "represent" the "copy" of x_j in the block $B_{q_{j,\ell}}$, as follows: For $1 \le j \le r$ and $1 \le \ell \le s_j$, replace x_j by $y_{j,\ell}$ in block $B_{q_{j,\ell}}$. For $1 \le j \le r$, replace x_j by y_j in all the equations.

Let $B'_1, \dots, B'_t$ denote the blocks of equations obtained from $B_1, \dots, B_t$, respectively, by the above replacements. Clearly, in the new blocks, there are no variables belonging to two or more blocks, even if there were shared equations in the original blocks. Denote $B' = \bigcup_{q=1}^{t} B'_q$; this set of equations is defined in the space $\mathbb{R}^s$. We now present an alternative formulation of CARP in which the basic operations are performed in $\mathbb{R}^s$.

ALGORITHM 3 (alternate CARP).

```
Initialization:
  1. $x^0 = (x_1^0, \dots, x_n^0) \in \mathbb{R}^n$ is arbitrary.
  2. $y^0 = E(x^0)$ (the initial vector in $\mathbb{R}^s$).
  3. Compute the set of equations $B'$ as above.
Iterative step: Given $y^k$, do:
  1. For every $1 \le q \le t$, execute some finite number of KACZ sweeps on the
     equations of $B'_q$, and get new values for the variables of $B'_q$. Formally, for
     some $p_k \ge 1$, compute $\bar{y}^{\,q} = \mathrm{KSWP}^{\,p_k}(B'_q, y^k)$.
  2. (a) For $1 \le j \le r$, set
         $$y_{j,1}^{k+1} = \cdots = y_{j,s_j}^{k+1} = \frac{1}{s_j}\left(\sum_{\ell=1}^{s_j} \bar{y}_{j,\ell}^{\,q_{j,\ell}}\right).$$
     (b) For $r < j \le n$, $I_j = \{q_{j,1}\}$, so set $y_j^{k+1} = \bar{y}_j^{\,q_{j,1}}$.
     (c) Denote $y^{k+1} = \left(y_{1,1}^{k+1}, \dots, y_{1,s_1}^{k+1}, \dots,\ y_{r,1}^{k+1}, \dots, y_{r,s_r}^{k+1},\ y_{r+1}^{k+1}, \dots, y_n^{k+1}\right)$.
Final step (after K iterative steps): Output $x^* = (x_1^*, \dots, x_n^*)$, where
     $$x_j^* = \begin{cases} y_{j,1}^K & \text{for } 1 \le j \le r, \\ y_j^K & \text{for } r < j \le n. \end{cases}$$
```

We can see that other than the initial and final steps, the entire algorithm can be executed iteratively or in parallel in $\mathbb{R}^s$, since the blocks B'_q do not contain any common variables. It follows that iterative step 1 is one (or more) KACZ sweeps on the equations of B'. Furthermore, it is clear that the result (after some K iterative steps) of this alternative form of CARP is the same as that of the regular CARP, assuming that the same relaxation parameters and repetition factors (λ_k and p_k) are used.

The following lemma will be used to show that the averaging operations in iterative step 2 are row projections in $\mathbb{R}^s$.

LEMMA 1 (the averaging lemma). *Let $1 \le m \le n$, $y^0 = (y_1^0, \dots, y_n^0) \in \mathbb{R}^n$, and let $y^1 = (y_1^1, \dots, y_n^1) \in \mathbb{R}^n$ be defined as follows: $y_i^1 = (y_1^0 + \cdots + y_m^0)/m$ for $1 \le i \le m$, and $y_i^1 = y_i^0$ for $m < i \le n$. Then y^1 can be obtained from y^0 by performing a sequence of $m-1$ orthogonal projections on hyperplanes of $\mathbb{R}^n$, as in* KACZ.

Proof. The proof proceeds by induction on m. For $m = 1$, there is nothing to prove. For $m = 2$, project y^0 onto the plane defined by the equation $-y_1 + y_2 = 0$. The vector of coefficients is $a = (-1, 1, 0, \dots, 0)$, the square of its norm is $\|a\|^2 = 2$, and the projection $y' = (y'_1, y'_2, \dots)$ is

$$\begin{aligned} y' &= y^0 \;-\; \frac{1}{2}\langle y^0, a\rangle a \\ &= (y^0_1, y^0_2, \dots) \;-\; \frac{1}{2}(-y^0_1 + y^0_2)(-1, 1, 0, \dots, 0) \\ &= \left(\frac{1}{2}(y^0_1 + y^0_2),\ \frac{1}{2}(y^0_1 + y^0_2),\ y^0_3,\ y^0_4, \dots\right). \end{aligned}$$

In other words, for $m = 2$, we obtained $y'_1 = y'_2 = (y^0_1 + y^0_2)/2$ (and left the other components unchanged) by performing one orthogonal projection on a suitable hyperplane.

We assume that the statement is true for m, and prove it for $m+1$: Let $y^0 = (y^0_1, \dots, y^0_n)$. Project y^0 onto the hyperplane defined by the equation $-y_1 - y_2 - \cdots - y_m + my_{m+1} = 0$. The vector of coefficients is $a = (-1, \dots, -1, m, 0, \dots, 0)$ and the square of its norm is $\|a\|^2 = m + m^2 = m(m+1)$. The projection is the point $y' = (y'_1, \dots, y'_n)$ defined by

$$\begin{aligned} y' &= y^0 \;-\; \frac{\langle y^0, a\rangle}{m(m+1)}\, a \\ &= (y^0_1, \dots, y^0_n) \;-\; \frac{-y^0_1 - \cdots - y^0_m + my^0_{m+1}}{m(m+1)}\, (-1, \dots, -1, m, 0, \dots, 0)\,. \end{aligned}$$

For each $1 \le i \le m$, we have

$$y'_i \;=\; y^0_i \;-\; \frac{1}{m(m+1)} \left(\sum_{j=1}^{m} y^0_j \;-\; my^0_{m+1} \right),$$

and the $(m+1)$st coefficient is

$$\begin{aligned} y'_{m+1} &= y^0_{m+1} \;+\; \frac{y^0_1 + \cdots + y^0_m - my^0_{m+1}}{m(m+1)}\, m \\ &= \frac{1}{m+1}\left(y^0_1 + \cdots + y^0_m\right) \;+\; \left(1 - \frac{m}{m+1}\right) y^0_{m+1} \\ &= \frac{1}{m+1}\left(y^0_1 + \cdots + y^0_m\right) \;+\; \frac{m+1-m}{m+1}\, y^0_{m+1} \\ &= \frac{1}{m+1}\left(y^0_1 + \cdots + y^0_m + y^0_{m+1}\right). \end{aligned}$$

In other words, y'_{m+1} already has the required value. For $m+1 < i \le n$, y^0_i is unchanged, i.e., $y'_i = y^0_i$. To set the other variables to the same value, we rely on the induction hypothesis, according to which we can set

$$y''_1 \;=\; \cdots \;=\; y''_m \;=\; \frac{1}{m}\left(y'_1 + \cdots + y'_m\right)$$

by using $m-1$ orthogonal projections on hyperplanes of $\mathbb{R}^n$. Note also that by the induction hypothesis, all the other variables will remain unchanged, i.e., $y''_i = y'_i$ for $m+1 \le i \le n$.

For each $1 \le k \le m$, we have

$$\begin{aligned}
y_k'' &= \frac{1}{m}\sum_{i=1}^{m} y_i' \\
&= \frac{1}{m}\sum_{i=1}^{m}\left[y_i^0 \;-\; \frac{1}{m(m+1)}\left(\sum_{j=1}^{m} y_j^0 \;-\; m y_{m+1}^0\right)\right] \\
&= \frac{1}{m}\sum_{i=1}^{m} y_i^0 \;-\; \frac{1}{m^2(m+1)}\sum_{i=1}^{m}\left[\sum_{j=1}^{m} y_j^0 \;-\; m y_{m+1}^0\right] \\
&= \frac{1}{m}\sum_{i=1}^{m} y_i^0 \;-\; \frac{m}{m^2(m+1)}\left(\sum_{j=1}^{m} y_j^0 \;-\; m y_{m+1}^0\right) \\
&= \frac{1}{m}\sum_{i=1}^{m} y_i^0 \;-\; \frac{1}{m(m+1)}\sum_{j=1}^{m} y_j^0 \;+\; \frac{1}{m+1}\, y_{m+1}^0 \\
&= \left(\frac{1}{m} - \frac{1}{m(m+1)}\right)\sum_{i=1}^{m} y_i^0 \;+\; \frac{1}{m+1}\, y_{m+1}^0 \\
&= \frac{1}{m+1}\sum_{i=1}^{m} y_i^0 \;+\; \frac{1}{m+1}\, y_{m+1}^0 \\
&= \frac{1}{m+1}\sum_{i=1}^{m+1} y_i^0 \; .
\end{aligned}$$

This proves the induction hypothesis and the lemma. □

We shall refer to the equations required by CARP for part 2 of its iterative step as the "averaging" equations. We also denote by B'' the system of equations consisting of B' together with all the averaging equations. As a consequence of the averaging lemma, we have the following.

THEOREM 1. *If the system of equations* (1) *is consistent, then B'' is consistent and CARP converges to a solution of* (1) *for any choice of the relaxation parameters λ_k (provided $\epsilon < \lambda_k < 2-\epsilon$ for some fixed positive ϵ) and for any positive choice of the internal KACZ sweeps (p_k). If the system* (1) *is inconsistent, then CARP converges cyclically under the following two conditions:* 1. *The sweeps through the equations are cyclic, and* 2. *the relaxation parameters are cyclic; i.e., a fixed relaxation parameter is associated with every equation of B''.*

Proof. Assume that the system (1) is consistent. It follows immediately that the set of equations of B' is also consistent. In B', each of the original variables x_j belonging to two or more blocks is replaced by a set of variables of the form $y_{j,1}, \dots, y_{j,s_j}$, so the equations of B', together with the set of equations $\{y_{j,1} = y_{j,2}, \dots, y_{j,s_{j-1}} = y_{j,s_j} \mid 1 \le j \le r\}$, form a consistent set of equations. Note now that the averaging equations are mathematically equivalent to the above set of equations. Hence, the system B'' (consisting of B' together with the averaging equations) forms a consistent system of linear equations in $\mathbb{R}^s$.

Furthermore, note that the number of averaging equations is exactly equal to $s-n$, which is the number of additional coordinates we get in going from $\mathbb{R}^n$ to $\mathbb{R}^s$. This means that the number of additional equations (in going from B' to B'') is exactly equal to the number of additional variables. In particular, if the system matrix of (1) is square, then the system matrix of B'' is also square. We refer now to the results of Aharoni and Censor [1], according to which KACZ converges on a consistent system even if the projections are not performed cyclically; all that is required is that each equation should be used infinitely often. This allows us to perform, in iterative step 1 of CARP, any positive number of internal sweeps in each block. See subsection 3.4 for details of the results of [1].

Assume now that the system (1) is inconsistent. Tanabe [32] shows that KACZ converges cyclically when there are no relaxation parameters (i.e., $\lambda = 1$). The extension of this result to cyclic relaxation parameters follows from Eggermont, Herman, and Lent [16, Thm. 3.1], from which our result for inconsistent systems follows. $\square$

Note that when the averaging operations are considered as row projections, they require a relaxation parameter $\lambda = 1$. Theorem 1 allows us to use $\lambda \neq 1$ for the other equations, in an inconsistent system, provided the projections follow a cyclic pattern and the relaxation parameter associated with every equation is fixed.

The generality of [1, Thm. 1], in the consistent case, allows us to modify the blocks, the number of inner iterations, and the relaxation parameters during the course of CARP, provided the conditions for [1, Thm. 1] are observed. This could be useful in a setting in which the number of processors can vary during the course of a computation.

3.3. CARP1: Blocks of one equation. A special case of CARP, which we call CARP1, occurs when every equation is taken as a separate block. In this case, s_j is simply the number of equations in which x_j has a nonzero coefficient, and the iterative step of CARP1 is given by the equation

$$x_j^{k+1} \;=\; x_j^k \;+\; \frac{\lambda_k}{s_j}\sum_{i=1}^{m}\frac{b_i - \langle a^i, x^k\rangle}{\|a^i\|_2^2}a_j^i \;. \tag{4}$$

CARP1 can be implemented on a multiprocessor system in block-parallel mode, though the implementation blocks are not necessarily single equations—they are simply (disjoint) blocks of equations assigned to different processors for the sake of runtime efficiency. Furthermore, the implementation blocks of CARP1 are not intrinsic to the algorithm as in the general CARP. This implementation is detailed in Appendix B.

Equation (4) is similar to the well-known Cimmino algorithm [14], whose iterative step is

$$x_j^{k+1} \;=\; x_j^k \;+\; \frac{\lambda_k}{m}\sum_{i=1}^{m}\frac{b_i - \langle a^i, x^k\rangle}{\|a^i\|_2^2}a_j^i \;. \tag{5}$$

Cimmino's algorithm was the initial motivational step leading to the CAV algorithm [11]: It was observed that when system (1) is sparse, then in the sum of (5) only a "small" number of elements are nonzero, but the sum is divided by the "large" m. This led to an attempt to replace m by s_j, resulting in the iterative step of [11, eq. 1.9], which is identical to (4). Note that even though CARP1 already appeared in [11] (without any specific name), no proof of convergence was given there; naturally, the convergence properties of CARP1 follow from Theorem 1.[1]

In [11], CARP1 was examined on test cases of image reconstruction from projections. Its initial rate of convergence was identical to that of CAV, but the images obtained were somewhat inferior to those obtained with CAV. Similar comparative results were obtained in [10] with BICAV (a block-sequential version of CAV) and a block-sequential version of CARP1.

CAV made use of sparsity-oriented weights, but differently from CARP: With s_j defined as for CARP1, we define a *sparsity weight* $w_i = 1/(\sum_{j=1}^n s_j (a_j^i)^2)$ for $1 \le i \le m$. The iterative step of CAV is given by

$$x^{k+1} \;=\; x^k \;+\; \lambda_k \sum_{i=1}^{m} w_i \left(b_i - \langle a^i, x^k\rangle\right) a^i \;. \tag{6}$$

Hence, the projections in CAV are orthogonal to the hyperplanes and weighted by w_i (in addition to λ_k). CAV and CARP1 are identical only when all the s_j's are equal.

Note that in CARP1, as given by (4), x^{k+1} can be viewed as the average of certain projections of x^k, but these projections are *not*, in general, orthogonal to the hyperplanes: Assume for simplicity that the equations are normalized and that $\lambda_k = 1$; then from (4) we get

$$x_j^{k+1} \;=\; \frac{1}{m} \sum_{i=1}^{m} \left[x_j^k \;+ \frac{m}{s_j} \left(b_i - \langle a^i, x^k\rangle\right) a_j^i \right] \;. \tag{7}$$

It follows that x^{k+1} is the average of a set of vectors p^i, $1 \le i \le m$, where the jth component of p^i is the summand of (7). If we consider p^i to be a projection of x^k in some direction "toward" the hyperplane defined by the ith equation, then the *direction* of this projection is $p^i - x^k$. However, from (7), we see that the jth component of this direction vector is $m(bi - \langle a^i, x^k\rangle) a_j^i / s_j$, so the direction vector is not colinear with a^i (unless all the s_j's are equal). Hence, p^i is not, in general, an orthogonal projection of x^k in the direction of the ith hyperplane.

3.4. Applicability of CARP to the convex feasibility problem. Aharoni and Censor [1] proved a general theorem regarding orthogonal projections onto convex sets. This result, together with the averaging lemma, led to the proof of Theorem 1. We shall show that Theorem 1 also applies to the consistent case of the convex feasibility problem (CFP), from which it follows that CARP is also applicable to the CFP.

Some preliminary definitions are needed: Let $I = \{1, 2, \dots, m\}$ be a set of integers. A function $w : I \to \mathbb{R}$ is called a *weight function* if $w(i) \ge 0$ for all $i \in I$ and $\sum_{i \in I} w(i) = 1$. An infinite sequence of weight functions $\{w_k\}_{k \ge 0}$ is called *fair* if for every $i \in I$, $w_k(i) > 0$ infinitely often; i.e., the inequality holds for infinitely many values of k.

Let $\{Q_i \mid i \in I\}$ be a finite family of closed convex sets in $\mathbb{R}^n$ such that $Q = \bigcap_{i \in I} Q_i \ne \emptyset$, and let $P_i : \mathbb{R}^n \to Q_i$ be defined as the *orthogonal projection* onto Q_i; i.e., for $x \in \mathbb{R}^n$, $P_i(x) = \arg\min\{\|x - y\| \mid y \in Q_i\}$. Now, given a weight function w, we define $P_w : \mathbb{R}^n \to \mathbb{R}^n$ by $P_w(x) = \sum_{i \in I} w(i) P_i(x)$.

[1]While revising this paper, the authors learned that Censor, Elfving, and Herman independently proved that CARP1, with weights, converges for both consistent and inconsistent systems.

Consider now the *block-iterative projections* (BIP) algorithm [1, Alg. 1], which starts from an arbitrary point $x^0 \in \mathbb{R}^n$ and whose iterative step is given by

$$x^{k+1} \;=\; x^k \;+\; \lambda_k \left(P_{w_k}(x^k) - x^k\right), \tag{8}$$

where $\{w_k\}_{k\geq 0}$ is a fair sequence of weight functions and $\{\lambda_k\}_{k\geq 0}$ is a sequence of user-determined relaxation parameters. We can see from (6) and (8) that when the convex sets Q_i are hyperplanes, then CAV and BIP have the same form, though the sparsity weights of CAV do not necessarily satisfy the conditions for the weights of BIP.

In [1, Thm. 1], it is shown that if for some $\epsilon > 0$, $\epsilon < \lambda_k < 2 - \epsilon$ for all k, then the sequence of points $\{x^k\}_{k\geq 0}$ generated by the BIP algorithm converges to a point $x^* \in \widehat{Q}$, where $\widehat{Q} = \bigcap_{i\in\widehat{I}} Q_i$ and $\widehat{I} = \{i \in I \mid \sum_{k=0}^{\ell} w_k(i) \underset{\ell\to\infty}{\longrightarrow} \infty\}$. (If $\widehat{I} = \emptyset$, then $\widehat{Q}$ is defined as $\mathbb{R}^n$.) The choices of blocks and weight functions determine the behavior of BIP—from sequential to block-sequential and to simultaneous; see [1] for details.

To apply CARP to the CFP, we divide the set of projections (onto the convex sets) into blocks, which need not be disjoint. Starting from an arbitrary initial point, we perform, for each block of projections, a sequence of orthogonal projections onto the convex sets of the block. This is done in parallel on all the blocks. The results from the separate blocks are then merged by component-averaging, as in the CARP algorithm for linear equations, to form the next iterate. The variables that are averaged are only those which are affected by projections belonging to two or more blocks.

The convergence proof for the consistent case is the same as in Theorem 1. This follows from the simple observation that, in the superspace, the hyperplanes defined by the averaging equations are also convex sets. It is easily seen that if the original convex sets have a nonempty intersection, then in the superspace, the convex sets and the hyperplanes defined by the averaging equations also have a nonempty intersection.

4. Numerical results and discussion.

Tests were run on two different types of parallel machines: A shared memory SGI origin 2000, and a distributed machine—a PC cluster running under Linux. The PC cluster consists of 16 single Pentium IV, 2.8GHz processors with 1GB memory each, connected with a dedicated 1Gb/sec. ethernet switch. Results were obtained on both machines on six test problems from the literature. The problems were run on the SGI and the PC cluster with 1, 2, 4, 8, and 16 processors, and also with 32 and 64 processors on the SGI machine. This section describes the test problems, the nonsymmetric solvers to which CARP was compared, numerical considerations and implementation, the stopping tests, and the numerical results and discussion.

4.1. The test problems.

We examined six PDE problems proposed by Bramley and Sameh [7]. These problems were collected from a variety of sources [18, 24, 25, 26] and were later used as test problems in several other works [2, 3]. The problems are typical of many fields, such as computational fluid dynamics (CFD), heat transfer, and structural mechanics, and are commonly used in a wide variety of industrial and engineering applications.

Each problem has an assigned analytic solution which is used to determine the boundary conditions. Furthermore, the analytic solution enables the calculation of the norm of the error. In the following, we use the common notation $\Delta u = u_{xx} + u_{yy} + u_{zz}$. Our test problems and their predetermined solutions are the following.

1. $\Delta u + 1000 u_x = F$.
2. $\Delta u + 1000 \exp(xyz)(u_x + u_y - u_z) = F$.

3. $\Delta u + 100xu_x - yu_y + zu_z + 100(x+y+z)u/xyz = F$.
4. $\Delta u - 10^5x^2(u_x + u_y + u_z) = F$.
5. $\Delta u - 1000(1+x^2)u_x + 100(u_y + u_z) = F$.
6. $\Delta u - 1000\,[(1-2x)u_x + (1-2y)u_y + (1-2z)u_z] = F$.

The analytical (preassigned) solutions for the above problems are the following:

Problem 1: $u(x,y,z) = xyz(1-x)(1-y)(1-z)$.

Problem 2: $u(x,y,z) = x+y+z$.

Problems 3–6: $u(x,y,z) = \exp(xyz)\sin(\pi x)\sin(\pi y)\sin(\pi z)$.

The expression for the right-hand-side function F in all six problems is computed analytically for each case, using the preassigned solution.

The test problems were solved on the unit cube domain ($[0,1]\times[0,1]\times[0,1]$) using a seven-point centered difference scheme. The equations were discretized using the same number of grid points in each direction. Results are shown for problems of size $40\times40\times40 = 64{,}000$ equations and of size $80\times80\times80 = 512{,}000$ equations.

4.2. Parallel methods used for comparison. The common wisdom today is that there is no "best" solver for *all* problems. Each class or type of problem has its own most effective solver. Since we propose CARP as a general-purpose solver, we chose to compare it with other general-purpose, state-of-the-art methods, even though better solvers exist for each particular problem. Our methods of choice were three parallel Krylov-based iterative nonsymmetric solvers: RGMRES [30], CGS [31], and Bi-CGSTAB [35]. Each of these methods was used by itself and with the following preconditioners: ILUT, Jacobi, Neumann, and least-squares. We also tried the symmetric Gauss–Seidel preconditioner, but it failed completely in all cases. The comparisons used the AZTEC software library [34], which is designed for the parallel solution of large sparse linear systems of equations.

RGMRES(k) was run with Krylov subspace size $k = 10$; larger values of k do not improve the robustness of RGMRES(k) until k becomes a significant fraction of the problem size (with a large increase of the required storage). The Jacobi, Neumann, and least-squares polynomial preconditioners were run with AZTEC's default polynomial order parameter of 3.

The ILUT preconditioner [29] depends on two parameters which strongly affect its behavior: the *drop tolerance* (the value below which elements are taken as zero), and the *fill-in*, which controls the maximum number of nonzeros allowed in each column/row of the incomplete LU factors. Similarly to the other preconditioners, we used the default AZTEC values for ILUT's parameters: drop tolerance = 0 and fill-in = 1.0 (i.e., no additional elements). These default values produced excellent results, so we found no need to experiment with them. Moreover, a thorough study of ILUT should also consider recent improvements, such as reorderings, scalings, and multilevel extensions, all of which go beyond the scope of the present study. Note that ILUT is essentially sequential, and its parallel implementation in AZTEC is based on using it separately on the submatrices of the subdomains.

4.3. Implementation details. In order to implement CARP efficiently on the given problems, we used the domain-decomposition approach and mapped different domains to different processors. Although other approaches are feasible in theory, this approach limits interprocessor communications because only variables bordering a neighboring domain need to be averaged (and hence exchanged between processors).

Both AZTEC and our implementation of CARP used the message passing interface (MPI) library for exchanging messages between the processors. We used SGI's proprietary implementation of MPI on the SGI machine and the public domain

MPICH routines on the PC cluster. CARP was implemented with the MPI Cartesian topology routines for interprocessor communications; these routines are designed for six-way nearest neighbor communications for domain-based data.

Numerical PDE approximations on three-dimensional grids (structured and unstructured) exhibit spatial locality, since each equation centered about a grid point involves only its neighboring grid points. This feature enables domain-decomposition approaches to parallel solvers, in which the coefficient matrix is distributed among the processors with local indexing of submatrices. Besides the memory savings, this allows efficient basic matrix-vector operations. For structured three-dimensional grids, the resulting coefficient matrix is a seven-point stencil matrix.

Our implementation of the data structures used by CARP is similar to that of AZTEC. It is aimed at facilitating the implementation of the communication tasks and gaining efficiency during the iterative procedure. In a preprocessing stage, the following information is saved in each processor:

1. a list of neighboring processors with which it communicates,
2. a list of local nodes that are coupled with external nodes,
3. local representation of the submatrix used by each processor.

The local submatrix in each processor was stored in the sparse matrix format called the "distributed modified sparse row" (DMSR) format; see [34]. This format, which is a generalization of the MSR format [29], stores only the nonzero elements of the coefficient matrix.

In CARP, one can perform any finite number of inner KACZ iterations inside a block before averaging the boundary values that are common to neighboring blocks. Such inner iterations are fast relative to the outer iterations since they do not use interprocessor communications. This allows us to choose the number of inner iterations so as to minimize the overall computation time. Naturally, this choice depends on the specific problem and the machine architecture. Note that when implementing CARP, the residual can be computed efficiently as part of the iterative procedure; see (3).

4.4. Stopping tests. There are several stopping criteria which one may apply to iterative systems. Our stopping criterion was to use the L_2 norm of the residual: $||b - Ax||_2 < \tau$, where τ is a preassigned tolerance. Since this stopping criterion depends on the scaling of the equations, we first normalized the equations by dividing each equation by the norm of its coefficient vector. For problems 1, 2, 4, 5, and 6, τ was defined by setting $\tau^2 = 10^{-11}$ ($\tau \approx 3.16\times10^{-5}$). For problem 3, τ was taken as $\tau = 2.3\times10^{-3}$. The reason for this difference is that in problem 3, CARP converged well until this residual value was reached, and then continued to converge too slowly to be practically significant. All the other methods failed to converge on problem 3.

In order to limit the time taken by the methods implemented in AZTEC, the maximum number of iterations was set to 5000. The AZTEC library has several other built-in stopping criteria: numerical breakdown, numerical loss of precision, and numerical ill-conditioning.

4.5. Results and discussion.

4.5.1. Main results. In all our runs the initial estimate x^0 was taken as the zero vector. Table 1 presents a summary of the convergence status of all the methods and preconditioners that were examined, for the six test problems, for 512,000 equations on 16 processors. In the case of convergence failure, it also indicates the reason for the failure. The results shown in the table are also valid for four processors with 64,000 and 512,000 equations, unless noted otherwise.

Table 1
Convergence status of the six problems on 16 *processors with* 512,000 *equations.*

Method	Problem number					
	1	2	3	4	5	6
CARP	**conv**	**conv**	**conv**	**conv**	**conv**	**conv**
RGMRES	**conv**	**conv**	stgn	**conv**	**conv**	**conv**
RGMRES + ILUT	**conv**	stgn^4	stgn	nbdn^{5,6}	**conv**	stgn
RGMRES + Jacobi	mxit	mxit	divg^3	mxit	mxit	mxit
RGMRES + Neum.	**conv**	**conv**	stgn	**conv**	**conv**	**conv**
RGMRES + LSQ	**conv**	**conv**	stgn	**conv**	**conv**	mxit^1
CGS	divg^1	divg^1	divg	divg	divg	divg^1
CGS + ILUT	**conv**	divg	divg	divg	**conv**	**conv**^{2}
CGS + Jacobi	divg	divg	divg	divg	divg	divg
CGS + Neum.	**conv**	divg^1	divg	nbdn^1	divg^3	**conv**^{4}
CGS + LSQ	divg^1	divg^1	divg	nbdn^2	divg	divg^1
Bi-CGSTAB	nbdn	nbdn	nbdn	nbdn	nbdn	nbdn
Bi-CGSTAB + ILUT	**conv**	divg	divg	divg^4	**conv**	**conv**^{3}
Bi-CGSTAB + Jacobi	nbdn	nbdn	divg^3	nbdn	nbdn	nbdn
Bi-CGSTAB + Neum.	**conv**^{3}	nbdn	nbdn	nbdn	nbdn	nbdn
Bi-CGSTAB + LSQ	nbdn	nbdn	nbdn	nbdn	nbdn	nbdn

Notation: **conv** – convergence, divg – divergence, mxit – max. iter. reached, nbdn – num. breakdown, stgn – stagnation.

Notes: Status also valid for 4 proc. with 64,000 and 512,000 eqns., unless noted: ^{1} **conv** with 64,000 eqns. ^{2} divg with 64,000 eqns. ^{3} nbdn with 64,000 eqns. ^{4} nbdn with 512,000 eqns. ^{5} stgn with 64,000 eqns. ^{6} stgn with 512,000 eqns.

Table 1 demonstrates the robustness of CARP—it converged in all cases, including problem 3, in which all the other methods failed. However, it should be noted that in problem 3, the residual decreased very slowly after it was reduced to 3×10^{-3}. Of the other methods tested, RGMRES is generally the most successful; it converged by itself (and with some of the preconditioners) in all the problems except problem 3.

Figures 1–6 present a comparison of the (L_2 norm of the) residual versus the execution time for the six test problems, for 64,000 equations, as obtained on the SGI machine with four processors. For each problem, results are presented for all the methods that converged (except for problems 1 and 2 with CGS and CGS+LSQ, which converged extremely slowly). The CARP results are for the optimal relaxation parameters. These figures indicate that there is no "best" solver for all cases; however, RGMRES, either by itself or with ILUT, is among the fastest methods, while ILUT is the most successful preconditioner.

We also tried out the following parallel methods:

- CARP1 performed poorly in all cases except number 3.
- CAV [11], although successful in problems of image reconstruction from projections, performed poorly in all cases except number 3, where it was identical to CARP1.
- String averaging [9, 12] also performs block-parallel row projections, but, as mentioned in the introduction, the results are merged by taking the weighted average of the separate results. This method performed poorly in all six problems, indicating the effectiveness of componentwise averaging in CARP.

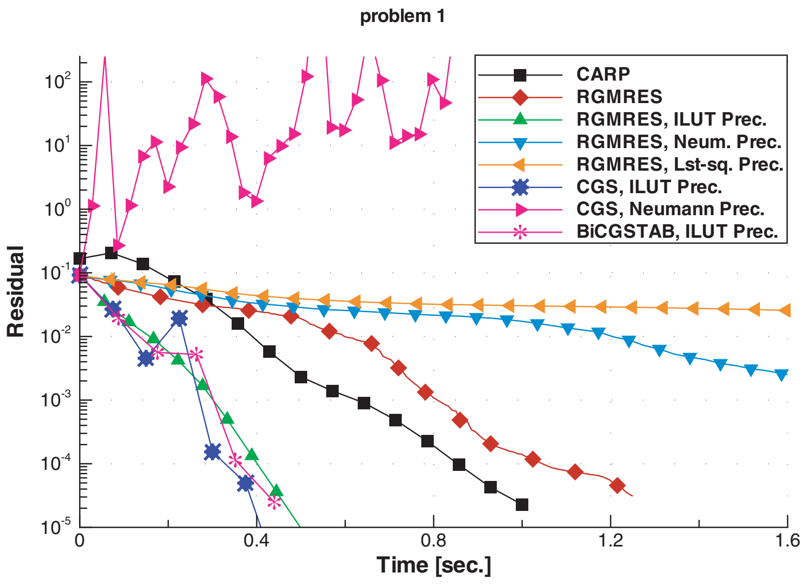

FIG. 1. *Execution times for problem* 1.

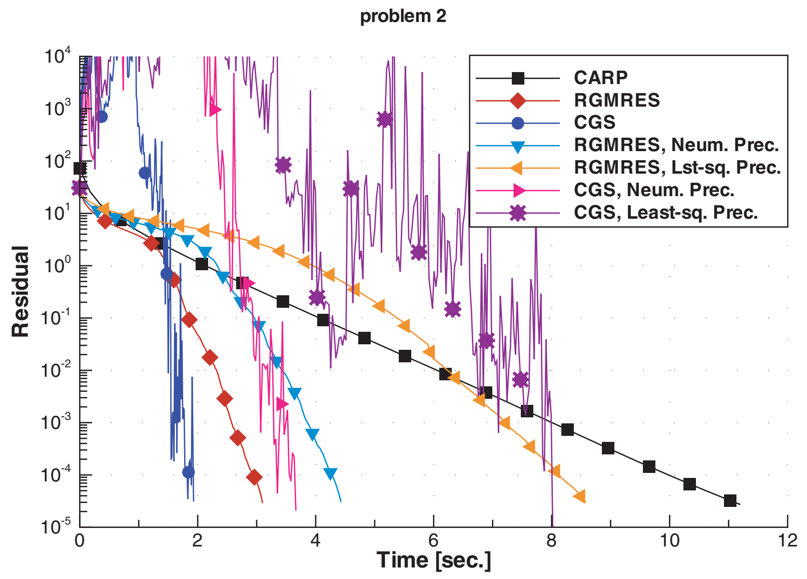

FIG. 2. *Execution times for problem* 2.

The convergence graphs for problem 3 compare CARP, CARP1, KACZ, and CAV. It can be seen that CARP's initial rate of convergence is very fast compared to the others, until a residual of approximately 10^{-2} is reached. From then on, CARP1, KACZ, and CAV exhibit a better rate of convergence. From this we can surmise that a hybrid algorithm, consisting initially of CARP and then CARP1 or KACZ or CAV, will function in this problem better than any single method. Problem 3 is hard

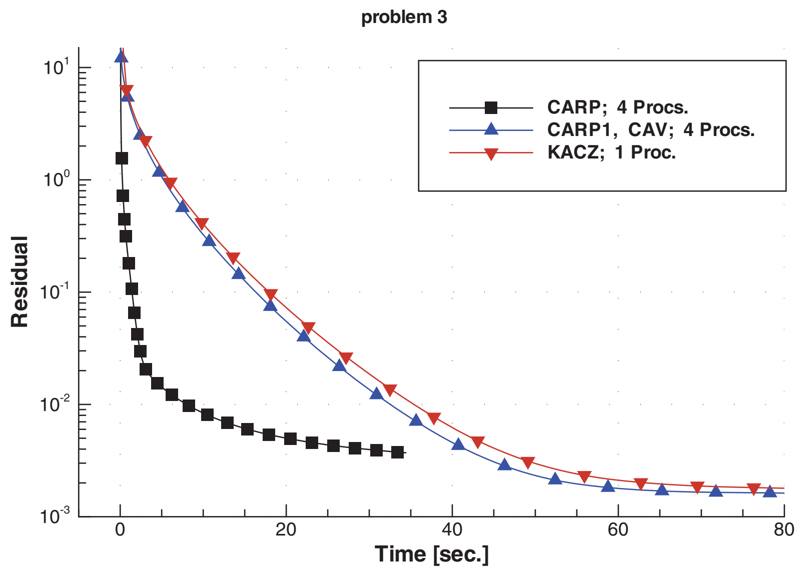

FIG. 3. *Execution times for problem 3.*

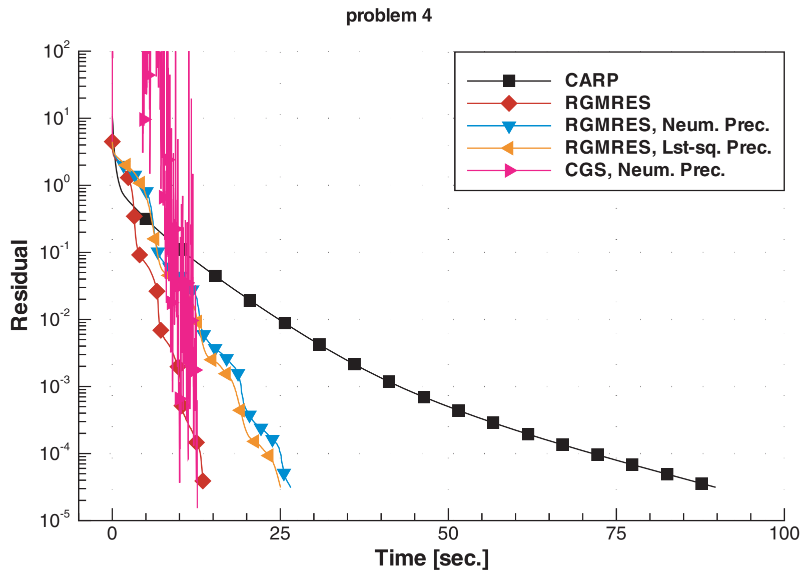

FIG. 4. *Execution times for problem 4.*

for solvers such as GMRES because it is indefinite, with eigenvalues surrounding the origin. It is well known that the eigenvalue distribution and the conditioning of the matrix of eigenvectors are more important for the convergence of GMRES than the conditioning of the system matrix. See also [2, 6, 7].

Tables 2 and 3 present various convergence measures obtained with CARP for 64,000 and 512,000 equations. Denoting by u and x the true solution and current

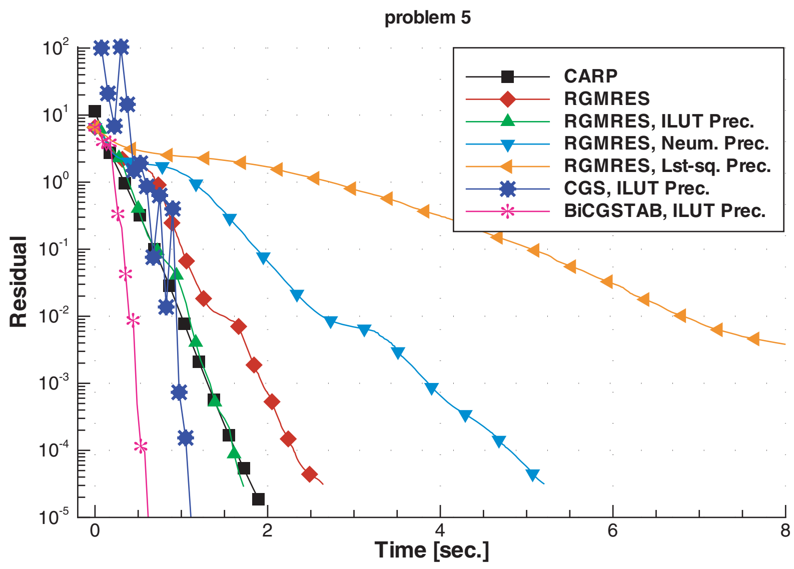

FIG. 5. *Execution times for problem* 5.

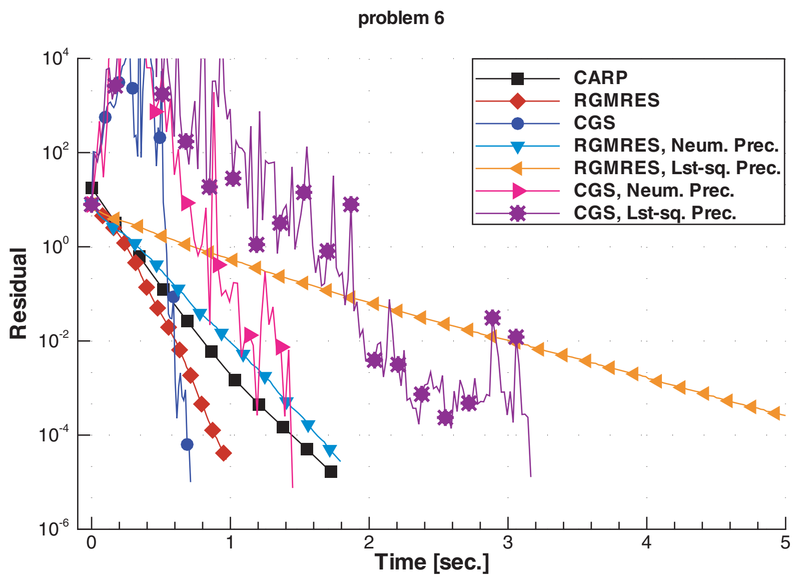

FIG. 6. *Execution times for problem* 6.

estimate, respectively, we computed the following convergence measures:

- The L_2 norm of the residual: RES$_2 = \|b - Ax\|_2$.
- The relative (or scaled) L_2 norm of the residual: REL-RES$_2 = \|b - Ax\|_2/\|b\|_2$ (the initial estimate is taken as zero).
- The L_∞ norm of the residual: RES$_\infty = \|b - Ax\|_\infty$.
- The relative L_2 norm of the error: REL-ERR$_2 = \|u - x\|_2/\|u\|_2$.
- The L_∞ norm of the error: ERR$_\infty = \|u - x\|_\infty$.

TABLE 2
Various convergence measures for four processors with 64,000 *equations.*

Problem:	1	2	3	4	5	6
RES$_2$	2.28E-5	2.72E-5	2.70E-3	3.13E-5	1.85E-5	1.66E-5
REL-RES$_2$	1.37E-4	3.78E-7	8.71E-5	2.87E-6	1.63E-6	9.29E-7
RES$_\infty$	2.28E-6	1.81E-6	1.68E-4	1.21E-6	1.96E-6	1.98E-6
REL-ERR$_2$	8.19E-5	7.82E-7	8.30E-3	1.72E-3	1.16E-3	9.39E-4
ERR$_\infty$	9.60E-6	2.24E-5	5.60E-2	7.80E-3	1.38E-3	2.19E-3

TABLE 3
Various convergence measures for four processors with 512,000 *equations.*

Problem:	1	2	3	4	5	6
RES$_2$	2.64E-5	2.93E-5	2.70E-3	3.16E-5	3.08E-5	2.80E-5
REL-RES$_2$	1.34E-4	2.10E-7	1.47E-4	2.06E-6	2.16E-6	1.37E-6
RES$_\infty$	1.23E-6	1.98E-6	6.51E-5	1.09E-6	1.15E-6	1.36E-6
REL-ERR$_2$	9.56E-5	6.83E-7	8.93E-3	3.96E-4	2.96E-4	2.40E-4
ERR$_\infty$	9.93E-6	2.74E-5	6.98E-2	2.05E-3	3.33E-4	3.05E-4

As can be seen from these tables, there is a wide variance between the measures achieved for the different problems, even though the convergence criterion for all of them (except problem 3) was taken as a fixed value for RES$_2$. For example, the achieved values of REL-RES$_2$ and REL-ERR$_2$ for problem 2 are much better than for the other problems. Also, REL-RES$_2$ for problem 3 is similar to that of problem 1, even though they differ greatly in RES$_2$. The consequence of the above is that there can be no absolute convergence criterion suitable for all cases.

Figures 7 and 8 present the speedup graphs of CARP for problems 1–6 with 512,000 equations, on the PC cluster and the SGI machine, respectively. The figures also include the plot corresponding to an efficiency of 1. It can be seen that all plots exhibit an almost linear behavior, and in some cases, the efficiency is greater than 1. This “super speedup” phenomenon is well known; it is due to the fact that in some cases, the total number of iterations required by CARP decreases as the number of blocks increases. A secondary contributing factor could be due to better utilization of memory cache when a problem of a certain size is divided among more processors.

4.5.2. Parametric studies.

Relaxation parameter $\boldsymbol{\lambda}$. Although the general theory allows us to vary λ during the iterative process, we chose a fixed value in our tests. Varying the value of λ during the iterations is a topic for further study. CARP's rate of convergence depends strongly on the choice of λ, and there is an optimal value for λ for each case, which we determined by numerical experimentation. This optimal value depends mainly on the problem being solved, and to a small extent, on the number of equations and the number of processors. Table 4 presents the optimal (fixed) relaxation parameters used for CARP, for the six test problems, with four processors.

Partitioning scheme. The initial strategy for choosing the partitioning scheme is to minimize the number of shared variables, but depending on the problem at hand, other partitioning schemes may yield better runtime results. The runtime and number of iterations required for convergence of CARP depend, for some problems, on the domain decomposition. Problems 1 and 5 are symmetric with respect to y and z but asymmetric with respect to x. Thus, a decomposition of the computational domain

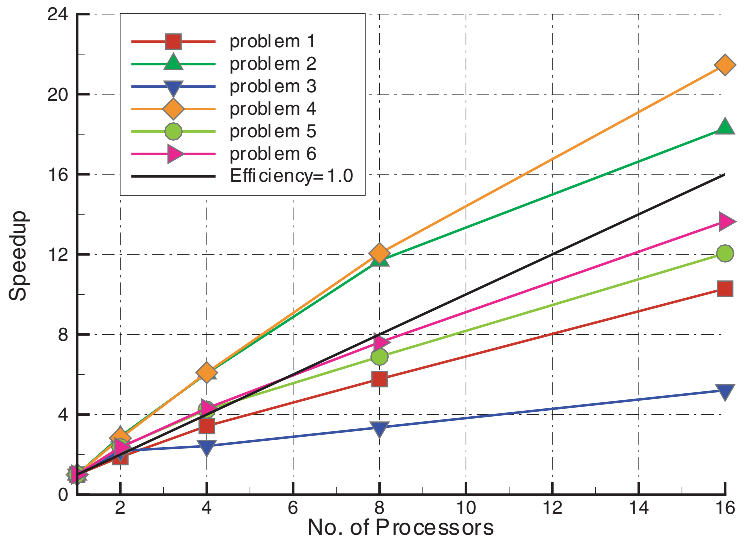

Fig. 7. *Speedup of CARP on the Linux cluster.*

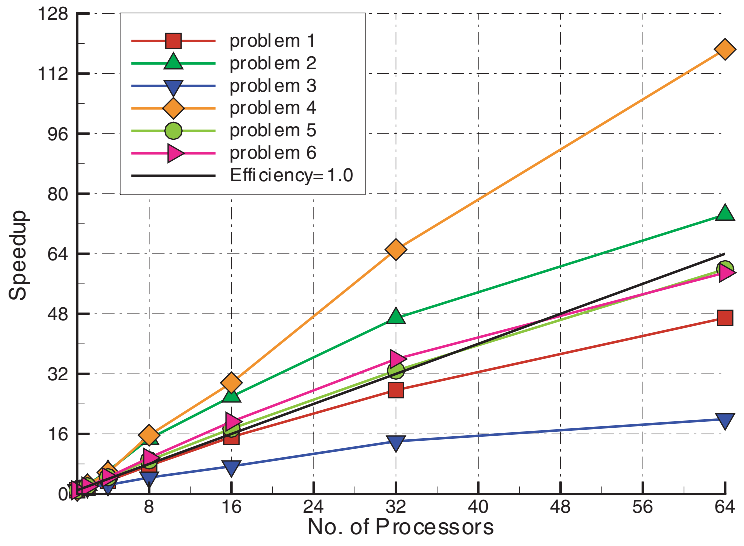

Fig. 8. *Speedup of CARP on the SGI machine.*

Table 4
Optimal relaxation parameter λ for CARP, for problems 1–6, with four processors.

Problem:	1	2	3	4	5	6
64,000 eqs.	1.90	1.90	1.60	1.50	1.85	1.35
512,000 eqs.	1.94	1.75	1.60	1.40	1.90	1.50

into four subdomains using a $4\times1\times1$ division in the x, y, z directions, respectively, requires considerably more iterations than a decomposition into $1\times4\times1$ or $1\times1\times4$ subdomains.

Inner iterations. Table 5 shows comparisons of the time and number of iterations required for convergence with one and four inner iterations, with different subdivisions, for problems 1 and 5 with 64,000 equations. In Tables 5 and 6, the number of iterations is the number of outer iterations, i.e., not counting the inner ones. The advantage of the $1\times4\times1$ and $1\times1\times4$ schemes over the $4\times1\times1$ scheme can be attributed to the fact that in the first two schemes, a relatively large number of "almost" parallel row vectors are placed in the same subdomain. Table 5 also shows that in some cases, one inner iteration is better than four, and in other cases, the opposite is true. For symmetric problems, there is no direction preference. The domain should be divided into (approximately) the same number of subdivisions in all directions to minimize the number of boundary nodes and reduce the communication time.

TABLE 5
Time and number of iterations required for different subdivision schemes for problems 1 *and* 5 *with* 64,000 *equations.*

Subdivision	Problem 1				Problem 5			
	1 inner it.		4 inner it.		1 inner it.		4 inner it.	
scheme	time	# it.	time	# it.	time	# it.	time	# it.
$4\times1\times1$	3.26	400	12.35	640	8.95	1050	17.11	880
$1\times4\times1$	**0.99**	**140**	1.28	70	3.42	480	**1.96**	**110**
$1\times1\times4$	**0.99**	**140**	1.28	70	3.56	500	2.07	110

Figures 9 and 10 present a more detailed view of the dependence of the number of iterations (required for convergence) on the relaxation parameter λ and the partitioning scheme for problems 1 and 6, respectively, for 512,000 equations. In both figures, the number of inner iterations was one, unless noted otherwise. Note in particular the shape of the graph for problem 1: It decreases linearly until the minimum and then "shoots" upward superlinearly. Figure 10 is typical for problems 2–6.

Table 6 summarizes the optimal values of the following parameters for the six problems, with 512,000 equations: the number of (outer) iterations, the relaxation parameter λ, the number of inner iterations, and the partitioning scheme.

Communication time. Table 7 provides the communication time required by CARP on problem 1 with 512,000 equations for the SGI machine and Linux cluster. This table emphasizes how the partitioning scheme affects the time required for communications: The $1\times1\times16$ scheme requires some 96,000 transfer edges (between adjacent grid points divided by a boundary), while the $1\times4\times4$ scheme requires only about 38,000 transfer edges. Also, the net computation time (total time minus communication time) for each machine is almost independent of the partitioning scheme. It follows that most of the time difference for this case is due to the communications overhead.

5. Conclusions and further research.

CARP is a new block-parallel method for solving large sparse nonsymmetric, general linear systems of equations on parallel machines. It uses single-row projections within each block, as in Kaczmarz's algorithm [23], followed by merging of the block results using certain component-averaging operations. CARP is also applicable in principle to nonsquare systems,

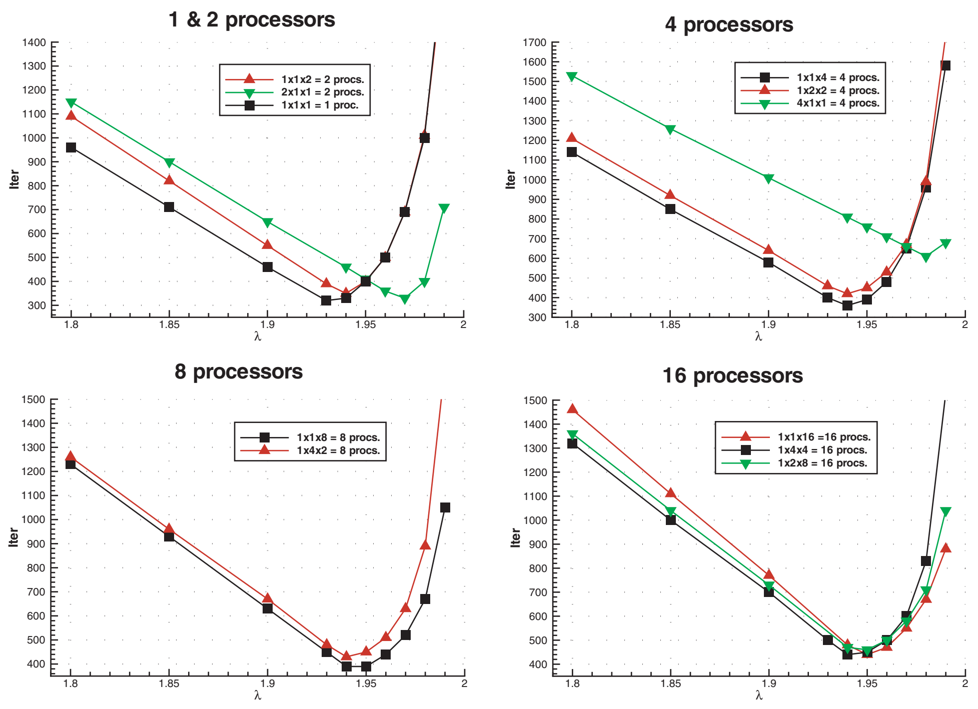

FIG. 9. *Number of iterations as a function of* λ *for problem* 1, 512,000 *equations.*

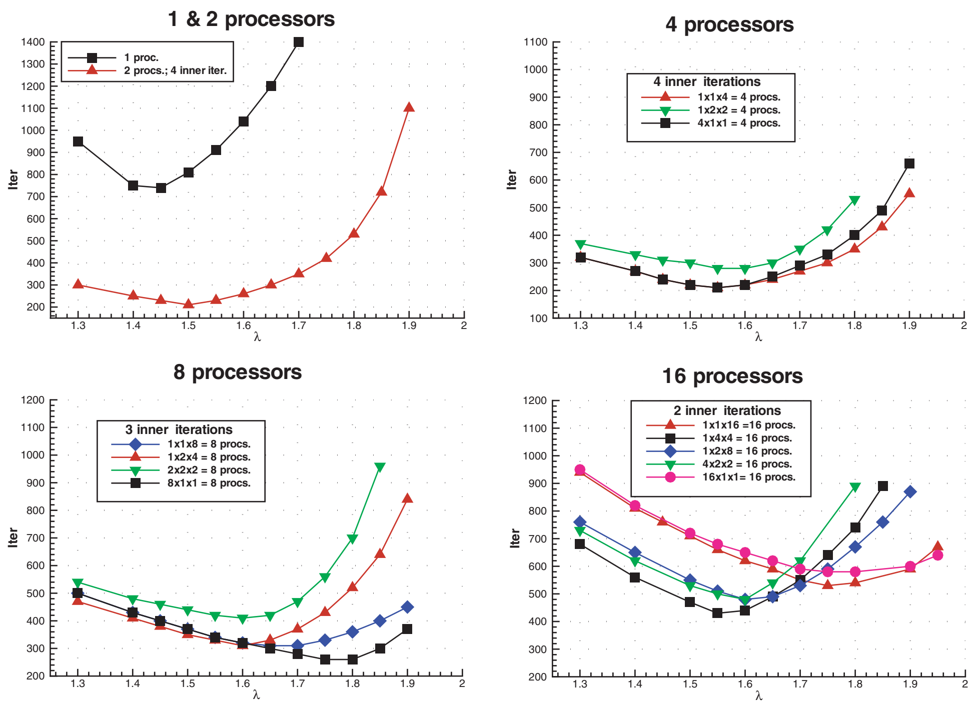

FIG. 10. *Number of iterations as a function of* λ *for problem* 6, 512,000 *equations.*

Table 6
Convergence details for CARP on the six problems with 512,000 *equations.*

No. of proc.		Problem no. 1	2	3	4	5	6
1	No. of iter.	330	6770	4200	59,600	1000	740
	λ	1.93	1.60	1.60	1.25	1.90	1.45
	Inner iter.	n/a	n/a	n/a	n/a	n/a	n/a
	Partition	1×1×1	1×1×1	1×1×1	1×1×1	1×1×1	1×1×1
2	No. of iter.	350	1,590	980	11,460	360	210
	λ	1.94	1.65	1.60	1.40	1.90	1.50
	Inner iter.	1	4	5	5	3	4
	Partition	1×1×2	1×2×1	2×1×1	2×1×1	1×2×1	1×2×1
4	No. of iter.	360	1,440	2,700	10,230	360	210
	λ	1.94	1.75	1.60	1.40	1.90	1.55
	Inner iter.	1	4	5	5	3	4
	Partition	1×1×4	1×2×2	2×2×1	4×1×1	1×1×4	1×1×4
8	No. of iter.	390	1,840	4,790	10,020	550	260
	λ	1.94	1.85	1.60	1.50	1.90	1.75
	Inner iter.	1	3	5	5	2	3
	Partition	1×1×8	1×4×2	2×2×2	2×4×1	1×1×8	8×1×1
16	No. of iter.	440	3,030	7,530	13,560	630	430
	λ	1.94	1.80	1.60	1.50	1.90	1.55
	Inner iter.	1	2	5	4	2	2
	Partition	1×4×4	2×4×2	2×4×2	1×4×4	1×2×8	1×4×4

Table 7
Communication time for problem 1 *with* 512,000 *equations.*

Configuration	SGI Total time	SGI Comm. time	SGI Comm. perc.	Linux cluster Total time	Linux cluster Comm. time	Linux cluster Comm. perc.
4 proc., 1×1×4	38.67	4.52	11.7%	19.50	3.31	17%
16 proc., 1×1×16	14.35	6.53	45.4%	8.49	3.50	41.2%
16 proc., 1×4×4	9.69	2.17	22.4%	6.24	1.52	24.4%

converges cyclically on inconsistent systems, and is applicable to the consistent case of the convex feasibility problem. A formal proof of convergence shows that CARP is equivalent to Kaczmarz's algorithm in a certain superspace. The new method is robust, memory efficient, and simple to program. It is also suitable for quasi-linearization approaches for nonlinear systems since there is no need to compute and store data related to submatrix inverses after every update of the system matrix.

CARP was compared with other state-of-the-art nonsymmetric solvers, with and without preconditioners, on six well-known test problems and on two different types of parallel machines. Test runs indicate that it is extremely robust compared to these methods, but since CARP is a linear method, its runtime, compared to nonlinear methods, is somewhat limited on some of the problems. On the other hand, CARP exhibits a linear speedup ratio, with efficiency sometimes greater than unity. These results indicate that CARP deserves a place in any robust iterative solution package for parallel machines.

One should note that the optimal performance of CARP depends on the choice of the relaxation parameter and the number of inner iterations in each block, and these are not known a priori. However, the rate of convergence behaves as a simple function of the relaxation parameters (just one local extremum). Hence, an adaptive search scheme can handle this problem: The processors are set to work for a small number of iterations on different values of these parameters, and the initial rates of convergence are compared; the search space is then narrowed by using the most promising parameter values.

Currently, work is in progress to speed up CARP's runtime efficiency by incorporating CG acceleration schemes, with very good results [19]. This is done by applying the techniques of [5] to the Kaczmarz scheme in the superspace. Another potential avenue of research would be to replace Kaczmarz in each block by other solution methods, such as CGNR or CGNE, on the block's submatrix. The solutions from the blocks can then be merged as in CARP. Such a scheme would require a different convergence proof. Another potential for speedup could be the implementation of communication hiding, i.e., information transmission between processors done in parallel with computations. Future research on CARP will also study its applicability to other areas of scientific computing.

Appendix A. Implementation details for CARP.

1. The equations of (1) are divided into blocks $B_1, \dots, B_t$, which are not necessarily disjoint. A processor P_q is assigned to B_q for $1 \le q \le t$.
2. The (normalized) system (1) and the s_j's are sent to the processors, but each processor retains only the equations belonging to its block. For $1 \le q \le t$, denote $J_q = \{j \mid \text{in some equation of } B_q, \text{ the coefficient of } x_j \neq 0\}$. Processor P_q retains only those s_j's for which $j \in J_q$.
3. Every processor P_q, for every $j \in J_q$, determines if it is the "main owner" of x_j. For every j, there should be exactly one main owner of x_j, even if x_j has nonzero coefficients in several blocks. The main owner of x_j will need access to and from any other processor P_r for which $j \in J_r$.
4. The initial estimate $x^0 = (x_1^0, \dots, x_n^0)$ is sent to the processors, but each processor P_q holds only the variables x_j^0 for $j \in J_q$.
5. Every processor P_q, for $1 \le q \le t$, sets the iteration index $k = 1$ and repeats the following steps until convergence:
    (a) Perform some finite number of KACZ sweeps on the equations of B_q and denote the result by $\bar{x}^q$; i.e., for some $p_k \ge 1$, $\bar{x}^q = \text{KSWP}^{p_k}(B_q, x^k)$.
    (b) For every $j \in J_q$: If P_q is not the main owner of x_j, send $\bar{x}_j^q$ to the main owner of x_j; if P_q is the main owner of x_j, receive the values $\bar{x}_j^r$ from all other processors P_r for which $j \in J_r$.
    (c) For every $j \in J_q$, if P_q is the main owner of x_j, compute the jth component of the next iterate:

$$x_j^{k+1} = \begin{cases} \bar{x}_j^q & \text{if } s_j = 1, \\ \dfrac{1}{s_j} \displaystyle\sum_{j \in J_r} \bar{x}_j^r & \text{if } s_j > 1. \end{cases}$$

    Note that the sum includes the case $j \in J_q$.
    (d) For every $j \in J_q$, if P_q is the main owner of x_j, send x_j^{k+1} to all other processors P_r for which $j \in J_r$.
    (e) Advance the iteration index k by 1.

6. The last iterate is sent by the processors to the master processor.

Appendix B. Block-parallel implementation of CARP1.

1. The equations of (1) are partitioned into disjoint blocks by partitioning the index set $\{1, 2, \dots, m\}$ into disjoint sets $I_1, \dots, I_t$. A processor P_q is assigned to the equations of I_q for $1 \le q \le t$.
2. The (normalized) system (1) and the s_j's are sent to the processors, but each processor retains only the equations belonging to its block. For $1 \le q \le t$, denote $J_q = \{j \mid \text{in some equation of } I_q, \text{ the coefficient of } x_j \neq 0\}$. Processor P_q retains only those s_j's for which $j \in J_q$.
3. Every processor P_q, for every $j \in J_q$, determines if it is the "main owner" of x_j. For every j, there should be exactly one main owner of x_j, even if x_j has nonzero coefficients in several blocks. The main owner of x_j will need access to and from any other processor P_r for which $j \in J_r$.
4. The initial estimate $x^0 = (x_1^0, \dots, x_n^0)$ is sent to the processors, but each processor P_q holds only the variables x_j^0 for $j \in J_q$.
5. Every processor sets the iteration index $k = 0$.
6. Every processor P_q, for $1 \le q \le t$, repeats the following steps until convergence:
   (a) For every $j \in J_q$, compute the value
   $$d_{q,j} \;=\; \sum_{i \in I_q} \left(b_i - \langle a^i, x^k \rangle \right) a_j^i \;.$$
   (b) For every $j \in J_q$: If P_q is not the main owner of x_j, send $d_{q,j}$ to the main owner of x_j; if P_q is the main owner of x_j, receive the values $d_{r,j}$ from all other processors P_r for which $j \in J_r$.
   (c) For every $j \in J_q$, if P_q is the main owner of x_j, compute the jth component of the next iterate:
   $$x_j^{k+1} \;=\; x_j^k \;+\; \frac{\lambda_k}{s_j} \sum_{j \in J_r} d_{r,j} \;.$$
   Note that the sum includes the case $j \in J_q$.
   (d) For every $j \in J_q$, if P_q is the main owner of x_j, send x_j^{k+1} to all other processors P_r for which $j \in J_r$.
   (e) Advance the iteration index k by 1.
7. The last iterate is sent by the processors to the master processor.

Acknowledgments. The authors are indebted to the anonymous reviewers, whose instructive and detailed comments led to a much improved presentation. Thanks are also due to Y. Censor for his helpful comments.

REFERENCES

[1] R. Aharoni and Y. Censor, *Block-iterative projection methods for parallel computation of solutions to convex feasibility problems*, Linear Algebra Appl., 120 (1989), pp. 165–175.

[2] M. Arioli, I. Duff, J. Noailles, and D. Ruiz, *A block projection method for sparse matrices*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 47–70.

[3] M. Arioli, I. S. Duff, D. Ruiz, and M. Sadkane, *Block Lanczos techniques for accelerating the block Cimmino method*, SIAM J. Sci. Comput., 16 (1995), pp. 1478–1511.

[4] Z.-Z. Bai, V. Migallón, J. Penadés, and D. B. Szyld, *Block and asynchronous two-stage methods for mildly nonlinear systems*, Numer. Math., 82 (1999), pp. 1–20.

[5] Å. Björck and T. Elfving, *Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations*, BIT, 19 (1979), pp. 145–163.

[6] R. Bramley and A. Sameh, *Domain decomposition for parallel row projection algorithms*, Appl. Numer. Math., 8 (1991), pp. 303–315.

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

[8] Y. Censor, P. P. B. Eggermont, and D. Gordon, *Strong underrelaxation in Kaczmarz's method for inconsistent systems*, Numer. Math., 41 (1983), pp. 83–92.

[9] Y. Censor, T. Elfving, and G. T. Herman, *Averaging strings of sequential iterations for convex feasibility problems*, in Inherently Parallel Algorithms in Feasibility and Optimization and Their Applications, D. Butnariu, Y. Censor, and S. Reich, eds., Elsevier, Amsterdam, The Netherlands, 2001, pp. 101–114.

[10] Y. Censor, D. Gordon, and R. Gordon, *BICAV: A block-iterative parallel algorithm for sparse systems with pixel-dependent weighting*, IEEE Trans. Medical Imaging, 20 (2001), pp. 1050–1060.

[11] Y. Censor, D. Gordon, and R. Gordon, *Component averaging: An efficient iterative parallel algorithm for large and sparse unstructured problems*, Parallel Comput., 27 (2001), pp. 777–808.

[12] Y. Censor and E. Tom, *Convergence of string-averaging projection schemes for inconsistent convex feasibility problems*, Optim. Methods Software, 18 (2003), pp. 543–554.

[13] Y. Chen, *An accelerated block-parallel Newton method via overlapped partitioning*, in Proceedings of the 15th International Conference on Domain Decomposition Methods, Lecture Notes in Comput. Sci. 40, R. Kornhuber et al., eds., Springer, Berlin, 2003, pp. 547–554.

[14] G. Cimmino, *Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari*, Ricerca Sci. XVI, Ser. II, Anno IX, 1 (1938), pp. 326–333.

[15] P. L. Combettes, *Inconsistent signal feasibility problems: Least-squares solutions in a product space*, IEEE Trans. Signal Process., SP-42 (1994), pp. 2955–2966.

[16] P. P. B. Eggermont, G. T. Herman, and A. Lent, *Iterative algorithms for large partitioned linear systems, with applications to image reconstruction*, Linear Algebra Appl., 40 (1981), pp. 37–67.

[17] T. Elfving, *Block-iterative methods for consistent and inconsistent linear equations*, Numer. Math., 35 (1980), pp. 1–12.

[18] H. Elman and G. Golub, *Iterative methods for cyclically reduced non-self-adjoint linear systems*, Math. Comput., 54 (1990), pp. 671–700.

[19] D. Gordon and R. Gordon, *CG Acceleration of CARP: Robust and Efficient Solution of Sparse Linear Systems*, in preparation.

[20] W. Hackbusch, *Iterative Solution of Large Sparse Systems of Equations*, Springer-Verlag, New York, 1994.

[21] G. T. Herman, *Image Reconstruction from Projections: The Fundamentals of Computerized Tomography*, Academic Press, New York, 1980.

[22] G. T. Herman, A. Lent, and P. H. Lutz, *Relaxation methods for image reconstruction*, Comm. ACM, 21 (1978), pp. 152–158.

[23] S. Kaczmarz, *Angenäherte Auflösung von Systemen linearer Gleichungen*, Bull. Acad. Polon. Sci. Lett., A35 (1937), pp. 355–357.

[24] C. Kamath and A. Sameh, *A projection method for solving non-symmetric linear systems on multi processors*, Parallel Comput., 9 (1988), pp. 291–312.

[25] D. Kincaid and D. Young, *Adapting iterative algorithms developed for symmetric systems to non-symmetric systems*, in Elliptic Problem Solvers, M. Schultz, ed., Academic Press, New York, 1981, pp. 353–359.

[26] D. Kuck, E. Davidson, D. Lawrie, and A. Sameh, *Parallel super-computing today and the Cedar approach*, Science, 231 (1986), pp. 967–974.

[27] G. Pierra, *Méthodes de décomposition et croisement d'algorithmes pour des problèmes d'optimisation*, Ph.D. thesis, University of Grenoble, Grenoble, France, 1976.

[28] G. Pierra, *Decomposition through formalization in a product space*, Math. Programming, 28 (1984), pp. 96–115.

[29] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003.

[30] Y. Saad and M. H. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 856–869.

[31] P. Sonneveld, *CGS, a fast Lanczos-type solver for nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 36–52.

[32] K. Tanabe, *Projection method for solving a singular system of linear equations and its applications*, Numer. Math., 17 (1971), pp. 203–214.

[33] M. R. Trummer, *Reconstructing pictures from projections: On the convergence of the ART algorithm with relaxation*, Computing, 26 (1981), pp. 189–195.

[34] R. S. Tuminaro, M. Herous, S. A. Hutchinson, and J. N. Shadid, *AZTEC User's Guide*, Tech. Report SAND99-8801J, Sandia National Laboratories, Albuquerque, NM, 1999.

[35] H. A. van der Vorst, *BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems*, SIAM J. Sci. Stat. Comput., 13 (1992), pp. 631–644.