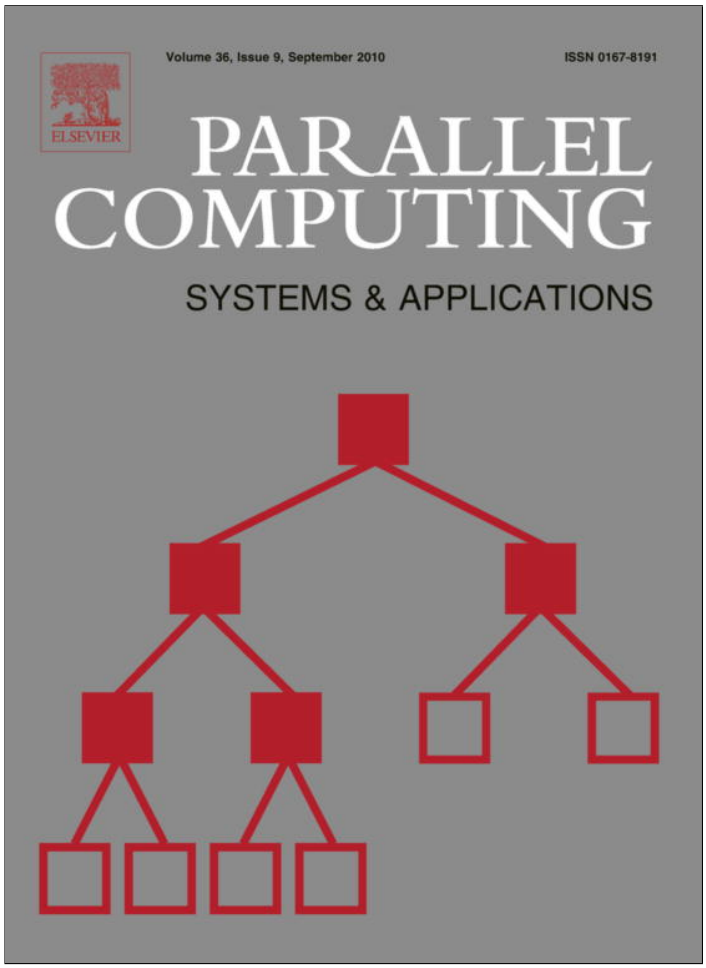

Parallel Computing 36 (2010) 495–515

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

CARP-CG: A robust and efficient parallel solver for linear systems, applied to strongly convection dominated PDEs

Dan Gordon [a,*], Rachel Gordon [b]

[a] Department of Computer Science, University of Haifa, Haifa 31905, Israel
[b] Department of Aerospace Engineering, The Technion–Israel Institute of Technology, Haifa 32000, Israel

ARTICLE INFO

Article history:
Received 7 December 2008
Received in revised form 27 January 2010
Accepted 28 May 2010
Available online 2 June 2010

Keywords:
CARP
CARP-CG
CGMN
Convection dominated
Elliptic equations
Kaczmarz
Linear systems
Parallel processing
Partial differential equations
Sparse systems

ABSTRACT

CARP-CG is a conjugate gradient (CG) acceleration of CARP, which was introduced by Gordon and Gordon as a robust block-parallel scheme for sparse linear systems. CARP performs Kaczmarz (KACZ) row projections within the blocks, and the results from the separate blocks are merged by averaging, for each component, its updated values from the different blocks. The averaging operations are equivalent to a sequence of certain KACZ row projections in some superspace (the "averaging projections"), and so CARP is equivalent to KACZ with cyclic relaxation parameters in that superspace. The CG-acceleration of CARP is based on a generalization of the (sequential) CGMN algorithm of Björck and Elfving, which accelerates KACZ by running it in a double sweep on the equations of a linear system, using a fixed relaxation parameter. CGMN is generalized to allow cyclic relaxation parameters, so the resulting method, called CGMNC, can be applied in the superspace. The averaging projections in the superspace can be done in any order, so CGMNC in the superspace can be implemented in the regular space by using CARP in a double sweep. The resulting algorithm, CARP-CG, is as robust as CARP but converges significantly faster. CARP-CG is compared with restarted GMRES, Bi-CGSTAB and CGS, with and without various preconditioners, on some stiff linear systems derived from convection dominated elliptic partial differential equations. The results indicate that CARP-CG is very robust and its runtime is very competitive with the other methods. A scaled version of CGNR was also tested, and it was as robust as CARP-CG, but slower.

1. Introduction

Many problems in scientific, engineering and biomedical applications require the solution of large sparse linear systems. In addition, the solution of a linear system is required as a repetitive inner step by many methods for solving non-linear problems, optimization problems and eigenvalue problems. Direct solvers, which are very robust, often fail to handle the huge matrices that occur in many applications. In such cases, iterative solvers are used, but they often fail on huge and ill-conditioned systems. The approach taken in many cases is to combine one of the many available solvers with a suitable preconditioner, which is sometimes tailor-fit to suit the given problem. However, solver/preconditioner combinations may be hard to find for some problems, so there is always a need for robust general-purpose methods that are applicable to a wide range of problems.

This work examines a certain parallel method, called CARP-CG, for solving large sparse linear systems arising from the discretization of elliptic convection–diffusion partial differential equations (PDEs). In this class of problems, some of the most challenging cases occur when the convection term is large, leading to large off-diagonal elements in the associated

* Corresponding author. Tel.: +972 4 8240159; fax: +972 4 8249331.
E-mail addresses: gordon@cs.haifa.ac.il (D. Gordon), rgordon@tx.technion.ac.il (R. Gordon).

0167-8191/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.parco.2010.05.004

system matrix. Such systems arise frequently in computational fluid dynamics (CFD) applications when the Péclet number is high—see Ferziger and Perić [15], and Gresho and Sani [22].

In recent years, Krylov subspace methods have emerged as the leading techniques for solving sparse linear systems. Among the better-known of these methods are the conjugate gradient (CG) [23], CGNR and CGNE [34, Section 8.3.1], Bi-CG [30,16], CGS [36], Bi-CGSTAB [38], GMRES [35], and QMR [17]. Various preconditioners can be used with these methods, but the robustness problem is still a hindrance in many applications.

CG is often used to accelerate other methods. Some of the most interesting applications of this were introduced by Björck and Elfving in a landmark paper [4]. Several works [25,6–8,1,2] combined the techniques of [4] together with Elfving's block-projection techniques [11], in order to tackle the robustness problem in a parallel setting. This approach requires the user to find blocks of independent equations (multi-coloring), and this may be problematic for unstructured grids. One of the algorithms of [4], called CGMN, is a sequential CG-acceleration of the Kaczmarz algorithm (KACZ) [24], with a fixed relaxation parameter, run in a double (forward and backward) sweep.

Until recently, CGMN has received little attention as a sequential solution method for linear systems derived from PDEs. Kamath and Weeratunga [26,27] compare CG-accelerated block-SSOR with CGMN (to which they refer as a CG-acceleration of "symmetric Kaczmarz"), using only the unity relaxation parameter. They conclude that on a single processor, CGMN is preferable, whereas the CG-accelerated block-SSOR is preferable in a parallel processing environment. Recently, we have shown that CGMN is very robust and efficient on linear systems derived from strongly convection dominated elliptic PDEs [19], including cases with discontinuous coefficients [20]. CGMN will serve as one of the two starting points for this work.

The second starting point our CARP algorithm [18], which divides the equations into blocks (which need not be disjoint). Each processor is assigned a block and, operating in parallel, the processors perform KACZ row projections on the equations of their assigned blocks. The results from the different blocks are then merged by averaging, for each component of the current iterate, its updated values from the different blocks. This mode of computation is usually called block-parallel, since the different blocks are processed in parallel. CARP avoids the need to find blocks of independent equations, and they may even overlap. Moreover, when the blocks correspond to physical subdomains, the transfer of data between processors is minimal. Numerical experiments comparing CARP to several Krylov subspace methods, with and without various preconditioners, showed that it is more robust, but it converged rather slowly on some of the test cases. The experiments were carried out on the first six problems considered in this study, all of them derived from elliptic PDEs with a very large convection term.

The theory behind CARP is that it is equivalent to KACZ in some superspace, with CARP's averaging operations being equivalent to the execution of a sequence of certain KACZ row projections in the superspace, called the "averaging projections". However, these particular row projections require a relaxation parameter of 1, whereas the regular row projections are usually done with a different relaxation parameter. This means that the superspace KACZ uses cyclic relaxation parameters (i.e., each equation is used with its own relaxation parameter).

This paper consists of three main results.

1. CGMN is generalized to allow for cyclic relaxation parameters. The resulting algorithm, called CGMNC, converges (in theory) even when the original system (1) is inconsistent and/or nonsquare.
2. In the backward sweep of CGMNC, the averaging projections are performed in reverse order. We show that executing these projections in any order achieves the same result, so the reverse order is also equivalent to CARP's averaging operations (in the regular space). Hence, CGMNC in the superspace can be implemented in the regular space by using CARP in a double sweep. The resulting algorithm, called CARP-CG, is as robust as CARP, but converges significantly faster. CARP-CG converges (in theory) even when the original linear system is inconsistent and/or nonsquare.
3. Extensive numerical experiments were done comparing CARP-CG with restarted GMRES, CGS and Bi-CGSTAB, with and without several preconditioners. The experiments were carried out on the same problems on which CGMN was tested [19], namely, linear systems derived from convection dominated elliptic PDEs using a central-difference scheme. CARP-CG converged on all the test cases, and no other algorithm/preconditioner combination achieved this level of robustness. Eight of the cases were strongly convection dominated, and on these, CARP-CG was very competitive in terms of runtime performance and robustness. The relative advantage of CARP-CG was even more pronounced in cases where the *number* of convection terms in the equations was greater than one, as was recently found in cases involving discontinuous coefficients [20]. Additionally, the residual in these eight cases decreased monotonically, whereas CGS and Bi-CGSTAB were oscillatory to some extent.

For convenience, we denote by GRS (geometric row scaling) the operation of "normalizing" the equations, i.e., dividing each equation by the L_2-norm of its coefficient. GRS is a well-known simple preconditioner which is widely used in many applications. Recently, it was shown to be generally useful for nonsymmetric problems with discontinuous coefficients and moderate convection; see [21] We also tested CGNR + GRS, and it was as robust as CARP-CG, but converged more slowly. It is quite easy to see that CGNR + GRS is equivalent to the CG-acceleration of the Cimmino algorithm [9].

CARP and CARP-CG implement a new approach to DD, which we call *component-averaged DD*, or CADD. CADD is not equivalent to any of the standard DD methods: if S is a subdomain and x_i is the value of a grid point which is external to S but borders on S, then in standard DD methods, x_i is not modified by the equations of S. In contrast, CADD "clones" x_i into S, the equations of S modify this clone, and the new value of x_i is taken as the average of all its updated values. Even with overlapping subdomains, none of the standard DD methods are equivalent to CADD (which also allows overlapping subdomains).

In this work, we are primarily interested in CARP-CG as an algebraic solver, and the PDEs are used to generate the linear systems. When evaluating solution methods for PDEs, there are several different issues to consider. There is the algebraic issue, in which we examine the relative residual and, if the exact solution of the linear system is known, also the relative error. Secondly, if the numerical method is good, one can compare the results with the analytic solution of the PDEs, when the latter is known. In some cases, the algebraic systems derived by central-differences cannot provide a good solution to the PDEs. For example, if the mesh Péclet number is large, then the exact solution of the linear system is sometimes oscillatory and may diverge widely from the analytic solution at some grid points; see [15, Chapters 3 and 4]; [22]; and [34, Section 2.2.4]. There are several approaches to this problem, one of them being the use of upwind schemes. However, these are less accurate since they are only first-order and introduce artificial diffusion—see [15]. Furthermore, since our interest here is primarily in the algebraic solution, we only consider the central-difference scheme.

The rest of this paper is organized as follows. Section 2 presents some essential background and algorithms for developing CARP-CG. Section 3 generalizes the CGMN technique and develops CARP-CG. Sections 4 and 5 describe the setup of the numerical experiments and the results. Section 6 presents some more detailed results for CARP-CG, and Section 7 compares various aspects of the tested methods. Section 8 concludes with a summary and some future research directions.

2. Background and previous relevant work

Throughout the rest of the paper, we assume that all vectors are column vectors, and we use the following notation: $\langle p,q\rangle$ denotes the scalar product of two vectors p and q, which is also p^Tq. By $\|\bullet\|$ we denote the L_2-norm of a vector. If A is an $m\times n$ matrix, we denote by a_i^* the ith row-vector of A; i.e., $a_i^* = (a_{i1},\ldots,a_{in})^T$. Similarly, we denote the jth column of A by $a_j^* = (a_{1j},\ldots,a_{mj})^T$. Unless noted otherwise, we shall assume that the matrix A of equation system (1) is of dimension $m\times n$ (m rows and n columns).

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

$$\sum_{j=1}^{n} a_{ij}x_j = b_i \text{ for } 1\leqslant i\leqslant m, \text{ or, in matrix form}: Ax = b. \tag{1}$$

One can construct from (1) two related "normal equations" systems

$$A^TAx = A^Tb, \tag{2}$$

$$AA^Ty = b, x = A^Ty. \tag{3}$$

Throughout the rest of the paper we assume that every column of A is nonzero. Such a column, if it existed, 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. Also, we can assume that every equation contains at least one nonzero coefficient.

2.1. The Kaczmarz algorithm

One of the earliest iterative algorithms, denoted as KACZ, is due to Kaczmarz [24]. KACZ starts from an arbitrary point $x^0\in\mathbb{R}^n$ as the initial iterate, and in each step, the current iterate is projected orthogonally towards a hyperplane defined by one of the equations. The hyperplanes are chosen in cyclic order. Each projection may involve a relaxation parameter $\lambda>0$ which determines the extent of the projection towards the hyperplane: the projection is either in front of the hyperplane, exactly on the hyperplane, or beyond the hyperplane, according to whether $\lambda<1$, $\lambda=1$, or $\lambda>1$, respectively. We shall henceforth assume that every relaxation parameter λ satisfies the inequalities $0<\lambda<2$. The term "cyclic relaxation parameters" means that for every $1\leqslant i\leqslant m$, there is a relaxation parameter λ_i which is always used with the projection towards the ith hyperplane. In our application of KACZ, the projections are always performed with an entire sweep (or pass) of all the equations; our formulation of KACZ incorporates this practice in the algorithm.

We shall henceforth assume that the equations are normalized, i.e., for $1\leqslant i\leqslant m$, the ith equation has been divided by $\|a_i^*\|$. We introduce three operators describing the internal Kaczmarz sweep: KSWP (the internal loop of KACZ), BKSWP is similar to KSWP, but with the equations traversed backwards, and DKSWP is KSWP followed by BKSWP.

Definition 1. Let A and b be as in (1), but after all the equations have been normalized. Let $\Lambda = (\lambda_1,\ldots,\lambda_m)$ be a vector of relaxation parameters, and $x\in\mathbb{R}^n$. Then KSWP (A,b,x,Λ) is defined as y^m, where $y^0,\ldots,y^m$ are obtained by executing the following code segment:

```
begin
  set y^0 = x.
  for i = 1,2,...,m do
    y^i = y^{i-1} + λ_i(b_i − ⟨a_{i*}, y^{i-1}⟩)a_{i*}
  enddo
end
```
(4)

BKSWP (A,b,x,Λ) is defined similarly, except that the equations are traversed in the reverse order. Finally, DKSWP (A,b,x,Λ) = BKSWP $(A,b,$KSWP $(A,b,x,\Lambda),\Lambda)$.

It is well known that KACZ is related to SOR (successive overrelaxation) [34, Section 4.1] in the following way: when SOR is applied to the normal equations system (3), with a relaxation parameter λ, the result is equivalent to KACZ with the same fixed relaxation parameter λ. For this reason, KACZ is also known as SOR-NE. Furthermore, if we use cyclic relaxation parameters as in Definition 1, we get the following formulation for KACZ:

Algorithm 1. KACZ, SOR-NE).

set $x^0 \in \mathbb{R}^n$ to an arbitrary value.
for $k = 0,1,2,\ldots$ until convergence **do**
x^{k+1} = KSWP (A,b,x^k,Λ)
enddo
end algorithm

The *symmetric* SOR (SSOR) is similar to SOR, but with the forward sweep followed by a backward sweep. SSOR applied to the normal equations (3) will be denoted by SSOR-NE. Its form is identical to Algorithm 1, except that KSWP is replaced by DKSWP.

An important result which will be required is due to Eggermont et al. [10, Theorem 1.1], according to which the following is true when KACZ is used with cyclic relaxation parameters:

- If the system (1) is consistent, then KACZ converges towards the solution.
- If the system (1) is inconsistent, then KACZ exhibits so-called "cyclic convergence", meaning that for every $1 \leqslant i \leqslant m$, if we consider just the sequence of projections towards the ith hyperplane, then this particular sequence converges.
- The above occurs with both square and nonsquare systems.

Note that according to this theorem and our formulation of KACZ with cyclic relaxation parameters, Algorithm 1 always converges (in theory), because each inner iteration always ends with the mth equation. For inconsistent systems, the convergence point will depend on the particular order with which the equations are traversed. Furthermore, a double KACZ sweep is equivalent to a regular KACZ sweep applied to a system obtained from (1) by duplicating the original equations in reverse order. Hence, by the above-mentioned result, SSOR-NE also converges when (1) is inconsistent and/or nonsquare.

2.2. Conjugate gradient and CGMN

When the system matrix A of (1) is symmetric and positive definite, one can apply the conjugate gradient (CG) algorithm, which converges in theory to a solution of (1). Björck and Elfving [4, Lemma 5.1] show that the same holds true if A is only positive semidefinite, provided the system (1) is consistent. The above lemma also provides more details on the convergence properties in such a case. CG can also be applied to the normal equations systems (2) and (3), and the resulting algorithms are known, respectively, as CGNR and CGNE [34, Section 8.3]. Note that the matrices AA^T and A^TA are always symmetric and positive semidefinite, so by the above-mentioned lemma, both CGNR and CGNE converge, in theory, when the system (1) is consistent (even if it is nonsquare).

Björck and Elfving's CGMN algorithm [4] is a CG-acceleration of KACZ obtained by running it in a double sweep, as in DKSWP. Since CGMN will be extended later, we omit the original CGMN presentation. See also [19].

2.3. The CARP algorithm

As mentioned in Section 1, CARP [18] divides the equations into blocks, and each processor performs KACZ iterations on the equations of its assigned block. The results from the separate blocks are merged by averaging, for each component, its values from the separate blocks, thus forming the next iterate. Besides PDEs, CARP is also applicable to other problems which allow domain decomposition, such as electron tomography [14]. In the presentation below, we also allow cyclic relaxation parameters on the original equations.

Formally, we proceed as follows. The (normalized) equations of (1) are divided into blocks, $B_1,\ldots,B_t$, which are not necessarily disjoint. Denote $\mathscr{B} = \{B_1,\ldots,B_t\}$. For each $1 \leqslant \ell \leqslant t$, let A^ℓ, b^ℓ denote the matrix and right-hand side, respectively, of the equations in B_ℓ. For $1 \leqslant j \leqslant n$, we denote $L_j = \{1 \leqslant \ell \leqslant t | B_\ell$ contains an equation with a nonzero coefficient of $x_j\}$, and $s_j = |L_j|$ is the number of blocks which contain at least one equation with a nonzero coefficient of x_j. Note that since A contains no column of zeros, all the s_js are positive. The next definition formalizes the component-averaging operation of CARP.

Definition 2. Let $\mathscr{B} = \{B_1,\ldots,B_t\}$ be as above. The component-averaging operator relative to $\mathscr{B}$ is a mapping $\mathrm{CA}_{\mathscr{B}} : (\mathbb{R}^n)^t \to (\mathbb{R}^n)$, defined as follows: let $y^1,\ldots,y^t \in \mathbb{R}^n$. Then $\mathrm{CA}_{\mathscr{B}}(y^1,\ldots,y^t)$ is the point in $\mathbb{R}^n$ whose jth component is given by

$$\mathrm{CA}_{\mathscr{B}}(y^1,\ldots,y^t)_j = \frac{1}{s_j}\sum_{\ell\in L_j} y_j^\ell, \tag{5}$$

where y_j^ℓ is the jth component of y^ℓ, for $1 \leqslant \ell \leqslant t$.

Given $\mathscr{B}$ and a vector of relaxation parameters $\Lambda = (\lambda_1,\ldots,\lambda_m)$, we denote by Λ_ℓ, for $1 \leqslant \ell \leqslant t$, the subsequence of Λ corresponding to the equations of B_ℓ. Let $d \geqslant 1$ be the number of sweeps in each block before the averaging operations. CARP can be expressed as follows.

Algorithm 2 (*CARP*).

set $x^0 \in \mathbb{R}^n$ to an arbitrary value.
for $k = 0,1,2,\ldots$ until convergence **do**
  for each $1 \leqslant \ell \leqslant t$ in parallel **do**
    $y^\ell = \mathrm{KSWP}^d(A^\ell, b^\ell, x^k, \Lambda_\ell)$
  enddo
  set $x^{k+1} = \mathrm{CA}_{\mathscr{B}}(y^1,\ldots,y^t)$.
enddo
end algorithm

A detailed parallel implementation of CARP is given in [18, Appendix A]. In order to explain later the development of CARP-CG, we need some details from [18]. Let $s = \sum_{j=1}^n s_j$. It is shown in [18] that CARP is equivalent to KACZ in $\mathbb{R}^s$, which is called a "superspace" of $\mathbb{R}^n$. The connection between $\mathbb{R}^n$ and $\mathbb{R}^s$ is a mapping which maps every $x \in \mathbb{R}^n$ to some $y \in \mathbb{R}^s$ such that a component x_j of x is mapped to s_j components of y, denoted as $y_{j1},\ldots,y_{js_j}$, as follows: the set of blocks $\mathscr{B} = \{B_1,\ldots,B_t\}$ is mapped to a set of blocks of equations $\mathscr{B}' = \{B_1',\ldots,B_t'\}$ in $\mathbb{R}^s$ by replacing each occurrence of a variable x_j in a block by some variable $y_{j\ell}$ according to the following rule: if $L_j = \{i_1,\ldots,i_{s_j}\}$, then x_j is replaced in B_{i_ℓ} by $y_{j\ell}$, for $1 \leqslant \ell \leqslant s_j$. The blocks of $\mathscr{B}'$ in $\mathbb{R}^s$ have no shared variables, so performing the KACZ projections in $\mathbb{R}^n$ in parallel on the blocks of $\mathscr{B}$ is equivalent to performing the corresponding projections in $\mathbb{R}^s$ sequentially (using the same relaxation parameters), i.e., as a regular KACZ sweep.

The averaging operations of CARP in $\mathbb{R}^n$ are equivalent to averaging, for each $1 \leqslant j \leqslant n$, all the components of $y \in \mathbb{R}^s$ which correspond to x_j, i.e., setting

$$y_{j1} = \cdots = y_{js_j} = \frac{1}{s_j}\sum_{\ell=1}^{s_j} y_{j\ell}.$$

It is shown in [18, Lemma 1] (the Averaging Lemma) that such averaging operations are equivalent to certain Kaczmarz row projections, done on the hyperplanes defined by so-called "averaging equations". These projections use a relaxation parameter of 1.0, which means that in $\mathbb{R}^s$, KACZ is executed with cyclic relaxation parameters (even when a fixed λ is used on the original equations). Hence, CARP is equivalent to KACZ in $\mathbb{R}^s$ with cyclic relaxation parameters. We denote by $A'y = b'$ the system of equations in $\mathbb{R}^s$ obtained from $Ax = b$ by taking the union of equations $\bigcup_{\ell=1}^t B_\ell'$, and also adding all the averaging equations.

As noted in [18], CARP converges cyclically, but since it is always executed with complete sweeps, it always converges according to [10, Theorem 1.1]. If the system (1) is inconsistent, then the convergence point will depend on the order in which the equations are taken, and also on the actual division of the equations into blocks. The Averaging Lemma and the averaging equations are presented below.

Lemma 1. The Averaging Lemma; [18, Lemma 1] *Let $1 \leqslant m \leqslant n$, $y^0 = (y_1^0,\ldots,y_n^0) \in \mathbb{R}^n$ and let $y^1 = (y_1^1,\ldots,y_n^1) \in \mathbb{R}^n$ be defined as follows: $y_i^1 = (y_1^0 + \cdots + y_m^0)/m$ for $1 \leqslant i \leqslant m$, and $y_i^1 = y_i^0$ for $m < i \leqslant n$. 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.*

The proof of the above lemma proceeds by induction on m, where for $m = 1$, it uses an orthogonal projection on the hyperplane defined by equation number $m-1$ in Table 1. For a given m, the projections are done in the order given by the table. The projections are done with relaxation parameter $\lambda = 1$.

Table 1
The averaging equations used in the proof of the Averaging Lemma.

Equation number	Equation
1	$y_1 + \cdots + y_{m-1} - (m-1)y_m = 0$
2	$y_1 + \cdots + y_{m-2} - (m-2)y_{m-1} = 0$
$\vdots$	$\vdots$
$m-2$	$y_1 + y_2 - 2y_3 = 0$
$m-1$	$y_1 - y_2 = 0$

3. Development of CARP-CG

3.1. Generalization of CGMN to cyclic relaxation parameters

We first introduce cyclic relaxation parameters $\Lambda = (\lambda_1, \ldots, \lambda_m)$ into the SSOR algorithm, and call the resulting algorithm SSORC. Note that even though our experiments used a single relaxation parameter for all the original equations, the extended theory allows us to use different parameters with different equations. This could be potentially useful for applications in which different regions are governed by different types of equations. SSORC can be applied to the normal equations, and we will call the resulting algorithm SSORC-NE.

We assume that the system (1) is normalized. Following [4], we denote by A_i the symmetric $n \times n$ matrix obtained from a_i^* as follows:

$$A_i = a_{i*}a_{i*}^T = \begin{pmatrix} a_{i1} \\ \vdots \\ a_{in} \end{pmatrix} (a_{i1}, \ldots, a_{in}).$$

Denoting $Q_i = I - \lambda_i A_i$, we obtain the following representation for the KACZ inner iteration of the normalized system (Eq. (4)):

$$y^i = Q_i y^{i-1} + \lambda_i b_i a_{i*}. \tag{6}$$

Hence, in a complete double sweep, we obtain the following representation for the SSORC-NE iteration:

$$x^{k+1} = Qx^k + Rb, \tag{7}$$

where $Q = Q_1 \cdots Q_m Q_m \cdots Q_1$ and R is the product of all the matrices multiplying b. Note that the matrices Q_i are symmetric, so Q is also symmetric. However, in SSORC-NE, the inner loop is also obtained by the operator DKSWP, so for any vector $x \in \mathbb{R}^n$, we have the identity

$$Qx + Rb = \mathrm{DKSWP}(A, b, x, \Lambda). \tag{8}$$

We now consider now the following linear system related to the iteration (7):

$$(I - Q)x = Rb. \tag{9}$$

In order to apply CG to the system (9), we need the following:

Theorem 1. *The system (9) is consistent, and the matrix $(I - Q)$ is symmetric and positive semidefinite.*

Proof. SSORC-NE is clearly KACZ with cyclic relaxation parameters, executed on a system of equations obtained from (1) by duplicating the original equations in reverse order. Hence, according to [10, Theorem 1.1], the algorithm SSORC-NE converges to a solution x^*. Therefore, x^* is a fixed point of the iteration (7), i.e., $x^* = Qx^* + Rb$. Hence, x^* is a solution to (9), which means that (9) is consistent.

The symmetry of $(I - Q)$ follows from the symmetry of Q.

To prove that $I - Q$ is positive semidefinite, note first that $A_i x = (a_{i*}a_{i*}^T)x = a_{i*}(a_{i*}^T x) = \langle a_{i*}, x\rangle a_{i*}$. This means that the operator A_i is the orthogonal projector onto a_i^*, so the spectrum of A_i is $\{0,1\}$. Now, $Q_i = I - \lambda_i A_i$, so the spectrum of Q_i is $\{1, 1 - \lambda_i\}$. Since $0 < \lambda_i < 2$, the spectral norm of Q_i satisfies $\rho(Q_i) < 1$.

We use two elementary results: for a symmetric matrix A, $\rho(A) = \|A\|$, where $\|A\|$ is the 2-norm of A [31, Prop. 1.23]; and, for any two matrices A, B, $\|AB\| \leqslant \|A\|\|B\|$ [31, Prop. 1.7]. It follows that $\rho(Q) < 1$, so the spectrum of Q is contained in $[-1, 1]$. Therefore, the spectrum of $I - Q$ is contained in $[0, 2]$, and the result follows. □

It follows from Theorem 1 and [4, Lemma 5.1] that CG can be applied to the system (9), and its theoretical convergence is guaranteed. The resulting algorithm, which we call CGMNC uses the identity (8) instead of actual matrix–vector multiplications, as explained below.

Algorithm 3 (*CGMNC*).

set $x^0 \in \mathbb{R}^n$ to an arbitrary value.
set $p^0 = r^0 = Rb - (I - Q)x^0 = Qx^0 + Rb - x^0 = \mathrm{DKSWP}(A, b, x^0, \Lambda) - x^0$.
note: the rightmost equality follows from Eq. (8).
for $k = 0, 1, 2, \ldots$ until convergence **do**
  $q^k = (I - Q)p^k = p^k - \mathrm{DKSWP}(A, 0, p^k, \Lambda)$
  note: the above equality follows from Eq. (8) with $b = 0$.
  $\alpha_k = \|r^k\|^2 / \langle p^k, q^k\rangle$

$x^{k+1} = x^k + \alpha_k p^k$
$r^{k+1} = r^k - \alpha_k q^k$
$\beta_k = \|r^{k+1}\|^2 / \|r^k\|^2$
$p^{k+1} = r^{k+1} + \beta_k p^k$

enddo
end algorithm

Based on all the above, we can state the following theorem.

Theorem 2. *Algorithm CGMNC always converges, even when the system (1) is inconsistent and/or nonsquare.*

3.2. The CARP-CG algorithm

Since CARP is KACZ in the superspace $\mathbb{R}^s$, we can apply CGMNC to the superspace equations in $\mathbb{R}^s$. Now, in the double sweep, immediately after the averaging row projections of the forward sweep, we have to perform the averaging row projections in the reverse order. The following lemma shows that such an operation is also equivalent to averaging the components, so in practice, this step can be omitted.

Lemma 2 (The Generalized Averaging Lemma). *Let $1 \leqslant m \leqslant n$, $y^0 = (y_1^0, \ldots, y_n^0) \in \mathbb{R}^n$ and let $y^1 = (y_1^1, \ldots, y_n^1) \in \mathbb{R}^n$ be defined as follows: $y_i^1 = (y_1^0 + \cdots + y_m^0)/m$ for $1 \leqslant i \leqslant m$, and $y_i^1 = y_i^0$ for $m < i \leqslant n$. Then y^1 can be obtained from y^0 by performing the sequence of $m-1$ orthogonal projections on the hyperplanes defined by the equations of Table 1 in any order.*

Proof. According to the proof of the Averaging Lemma of [18], the averaging operations are obtained by performing the orthogonal projections on the hyperplanes defined by the equations of Table 1 in the same order as given in the table. However, it is easily seen that all these hyperplanes are pairwise orthogonal, so the projections can be done in any order. □

It follows from Lemma 2 that we can replace the double sweep of CGMNC in $\mathbb{R}^s$ by a double sweep of CARP in $\mathbb{R}^n$, and, as noted, there is no need to repeat the averaging operations. However, we wish to add a further practical improvement before describing CARP-CG. Our numerical experiments showed that CARP-CG is more efficient if the backward sweep is also followed by the averaging operations. There is no formal problem with this, because the variables shared by more than one block are initially equal, so we can just assume that we start off with the averaging operations; hence, they can be repeated at the end of the back sweep. To summarize, the double sweep of CARP-CG (in $\mathbb{R}^n$) is the following: forward sweep → averaging → backward sweep → averaging.

In order to define CARP-CG conveniently, we introduce the notation DCSWP, which stands for "double CARP sweep". DCSWP is a double sweep of the CARP iteration, with component averaging between the forward and backward sweep, and also at the end of the backward sweep.

Definition 3. Let $A, b, \mathscr{B}, \Lambda, \Lambda_\ell$ $(1 \leqslant \ell \leqslant t)$ be as before, and $x \in \mathbb{R}^n$. Then DCSWP$(A, b, \mathscr{B}, x, \Lambda)$ is defined as x'', where x' and x'' are obtained by applying the following code segment:

begin
  for *each* $1 \leqslant \ell \leqslant t$ *in parallel* **do**
    y^ℓ = KSWP $(A^\ell, b^\ell, x, \Lambda_\ell)$
  enddo
  set $x' = \mathrm{CA}_{\mathscr{B}}(y^1, \ldots, y^t)$.
  for *each* $1 \leqslant \ell \leqslant t$ *in parallel* **do**
    y^ℓ = BKSWP $(A^\ell, b^\ell, x', \Lambda_\ell)$
  enddo
  set $x'' = \mathrm{CA}_{\mathscr{B}}(y^1, \ldots, y^t)$.
end

From our previous discussion, DCSWP$(A, b, \mathscr{B}, x, \Lambda)$ in $\mathbb{R}^n$ is equivalent to DKSWP (A', b', y, Λ') in $\mathbb{R}^s$, where $y \in \mathbb{R}^s$ is obtained from $x \in \mathbb{R}^n$ by replacing each x_j by $y_{j1}, \ldots, y_{js_j}$ as described in Section 2.3, and the components of Λ' are taken as identical to those of Λ for the regular equations, and 1.0 for the averaging equations. CARP-CG (in $\mathbb{R}^n$) is equivalent to CGMNC in $\mathbb{R}^s$ on the system $A'y = b'$, with the vector of relaxation parameters Λ'. So, instead of using DKSWP (A', b', y, Λ'), we use DCSWP$(A, b, \mathscr{B}, x, \Lambda)$, as follows:

Algorithm 4 (*CARP-CG*).

set $x^0 \in \mathbb{R}^n$ to an arbitrary value.
set $p^0 = r^0 = \mathrm{DCSWP}(A, b, \mathscr{B}, x^0, \lambda) - x^0$.

for $k = 0,1,2,\ldots$ until convergence **do**
$q^k = p^k - \mathrm{DCSWP}(A, 0, \mathscr{B}, p^k, \lambda)$
$\alpha_k = \|r^k\|^2 / \langle p^k, q^k \rangle$
$x^{k+1} = x^k + \alpha_k p^k$
$r^{k+1} = r^k - \alpha_k q^k$
$\beta_k = \|r^{k+1}\|^2 / \|r^k\|^2$
$p^{k+1} = r^{k+1} + \beta_k p^k$
enddo
end algorithm

The following result follows from Theorem 2 and from the fact that CARP-CG is equivalent to CGMNC in $\mathbb{R}^s$.

Theorem 3. *CARP-CG always converges, even when the system (1) is inconsistent and/or nonsquare.*

4. Setup of the numerical experiments

Tests were run on a PC Linux cluster connected with a dedicated 1 Gb/s ethernet switch. Each PC was a Pentium IV 2.8 GHz processor with 1 GB memory. Test runs were made with 1, 2, 4, 8 and 16 processors. This section describes the test problems, the nonsymmetric parallel solvers and preconditioners which were used, implementation details, and the stopping tests.

4.1. The test problems

We examined nine well-known problems derived from convection–diffusion elliptic PDEs. Problems 1–7 were collected from a variety of sources [28,29,25,6,13] and were also used as test problems in several other works [7,8,1,2,18]. Problems 3 and 7 also contain a reaction term. Problem 8 is taken from [34, Section 3.7, Problem F3D]; it is also available from the SPARSKIT library [33]. Problem 9 is identical to 8, except that the convection term is increased by two orders of magnitude. This was done in order to study how this single change affects the behavior of the algorithms. These problems appear in many fields such as CFD, heat transfer and structural mechanics, and are commonly used in a wide variety of industrial and engineering applications. In the following presentation of the test problems, we use the standard notation $\Delta u = u_{xx} + u_{yy} + u_{zz}$.

1. $\Delta u + 1000u_x = F$.
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. $\Delta u - 10^5 x^2(u_x + u_y + u_z) = F$.
5. $\Delta u - 1000(1 + x^2)u_x + 100(u_y + u_z) = F$.
6. $\Delta u - 1000[(1 - 2x)u_x + (1 - 2y)u_y + (1 - 2z)u_z] = F$.
7. $\Delta u - 1000x^2 u_x + 1000u = F$.
8. $\Delta u - \partial(10e^{xy}u)/\partial x - \partial(10e^{-xy}u)/\partial y = F$.
9. $\Delta u - \partial(1000e^{xy}u)/\partial x - \partial(1000e^{-xy}u)/\partial y = F$.

Problems 1–7 have the following analytical (preassigned) solutions:

- Problem 1: $u(x,y,z) = xyz(1 - x)(1 - y)(1 - z)$.
- Problem 2: $u(x,y,z) = x + y + z$.
- Problems 3–7: $u(x,y,z) = \exp(xyz)\sin(\pi x)\sin(\pi y)\sin(\pi z)$.

The expression for the right-hand side function F in Problems 1–7 is computed analytically, using the preassigned solution. Similarly to [34, Section 3.7, Problem F3D], Problems 8 and 9 are meant to test just algebraic convergence, so the right-hand side was created artificially by first computing A, and then computing $b = Av$, where $v = (1,1,\ldots,1)^T$. All the problems were solved on the unit cube domain $[0,1] \times [0,1] \times [0,1]$. In Problems 1–7, Dirichlet boundary conditions were used, as determined by the preassigned solutions. For Problems 8 and 9, the boundary conditions were taken as $u = 0$. The problems were discretized using a grid size of $80 \times 80 \times 80$, resulting in 512,000 equations, obtained by using a seven-point centered difference scheme.

In [20], on data with discontinuous coefficients, it was found that the performance of GMRES and Bi-CGSTAB, with and without ILU (0), depends very significantly on the *number* of large convection terms in the equation: if there is only one such term, as in Problems 1 and 7, then some of these methods perform quite well. However, if there are two or three such terms, then these methods perform very poorly, while CGMN and CGNR + GRS perform quite well. We therefore tested three variants of Problems 1, 5 and 7, obtained by changing the number of large convection terms from one to two:

- Problem 1A: $\Delta u + 1000(u_x + u_y) = F$.
- Problem 5A: $\Delta u - 1000(1 + x^2)u_x + 1000u_y + 100u_z = F$.
- Problem 7A: $\Delta u - 1000x^2(u_x + u_y) + 1000u = F$.

Problems 3 and 7 are indefinite, with eigenvalues on both sides of the imaginary line. This explains, in part, the particular difficulty of most methods on these problems. Another property of interest is whether the symmetric part of A, $(A + A^T)/2$, is positive definite. This interest is due to Elman [12], who showed that this property implies the convergence of many restarted algorithms. Problem 1 (before the scaling of the equations) and Problem 8 satisfy this property.

The performance of CARP-CG depends somewhat on the subdivision of the domain into blocks. We denote by "$a \times b \times c$-subdivision" the subdivision obtained by partitioning the domain into a segments in the x direction, b in the y direction, and c in the z direction. With 16 processors, we obviously have $abc = 16$. We experimented with all possible subdivisions in the x,y,z directions, e.g., $1 \times 1 \times 16$, $4 \times 1 \times 4$, and so on.

4.2. Parallel methods and preconditioners

CARP-CG is proposed as a general-purpose solver, so we compared it with other general-purpose, state-of-the-art methods, even though better solvers exist for each particular problem. We used four parallel Krylov-based iterative nonsymmetric solvers: CGNR [34, Section 8.3], restarted GMRES [35], CGS [36] and Bi-CGSTAB [38]. CGNR was actually CGNR + GRS, which, as noted earlier, is also a CG-acceleration of the Cimmino algorithm [9]. We also tried QMR [17], but its performance was quite poor on our test problems, so we left it out. As mentioned, all the equations were normalized before applying the various solvers. Restarted GMRES, CGS and Bi-CGSTAB were tested without preconditioners and also with the following preconditioners: ILUT [35], Neumann and Least Squares. We also tried the Jacobi and the symmetric Gauss–Seidel preconditioners but they failed in all the cases. For details of these preconditioners, see for example [34, Chapters 10 and 12]. In all, CARP-CG was compared to 13 different combinations of algorithms and preconditioners. The comparisons used the AZTEC software library [37], which is designed for the parallel solution of large sparse systems of linear equations. Where necessary, AZTEC uses the additive Schwarz DD method for parallelization.

In all cases, the initial estimate was taken as $x^0 = 0$. Restarted GMRES was run with Krylov subspace size $k = 10$; larger values of k did not improve the robustness of RGMRES on our test problems, until k became a significant fraction of the problem size (with a large increase of the required storage). The Neumann and Least Squares polynomial preconditioners were run with AZTEC's default polynomial order 3. Among the various least squares preconditioners, AZTEC uses the one based on [32]. For ILUT's parameters, we used AZTEC's default values: drop tolerance = 0 (meaning that no elements were dropped), and fill-in = 1.0 (i.e., no additional elements). These default values produced very good results, so we did not experiment with them. No coarse-grid correction was applied to ILUT.

A comment on our choice of preconditioners: although the above preconditioners were originally designed for sequential processing, they have proved to be very effective in a parallel setting on the tested problems. This is borne out by fact that for every problem, the same algorithm/preconditioner combinations that worked well sequentially on a single processor (see [19]) also worked well in the parallel setting, and vice versa. Furthermore, the relative performance of the various algorithms and preconditioners was very similar in both settings. 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

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. For the structured three-dimensional grids that we used, the resulting coefficient matrix is a 7-point stencil matrix.

Our implementation of the data structures used by CARP-CG 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: a list of neighboring processors with which it communicates, a list of local nodes that are coupled with external nodes, and a local representation of the submatrix and subvectors used by the processor. The local submatrix in each processor was stored in the sparse matrix format called DMSR (distributed modified sparse row) [37], which is a generalization of the MSR format [34, Section 3.4]. Additionally, every processor stores the "clones" of the relevant external grid points.

AZTEC uses the BLAS numerical routines [5]. It is written in C, and compiled with the GNU compiler gcc with optimization parameter -O (which produced the best results). CARP-CG and CGNR were written in Fortran (without multithreading) and compiled with g77, with the same optimization. In our implementation, all matrix–vector and scalar products were coded inline; this is one contributing factor to the efficiency of our code. Both AZTEC and our implementations of CARP-CG and CGNR use the MPI (message passing interface) library for exchanging messages between the processors. The public domain MPICH package was used for MPI. CARP-CG was implemented with the MPI Cartesian topology routines for inter-processor communications. These routines are designed for 6-way nearest neighbor communications for domain-based data.

4.4. Stopping tests

There are several stopping criteria that one may apply to iterative systems. Our stopping criterion was to use the relative residual: $\|b - Ax\|/\|b - Ax^0\| < 10^{-8}$. In some of the cases, this was not attainable. Since this stopping criterion depends on the scaling of the equations, we first normalized the equations (for all the tested methods) by dividing each equation by the L_2-norm of its coefficients. 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.

One should note that the test for numerical breakdown in AZTEC uses the machine precision constant `DBL_EPSILON`, and this may result in a premature notice of numerical breakdown in some cases. To get around this problem, we suggest to multiply the variable `brkdown_tol` in AZTEC's Bi-CGSTAB by some small number, e.g., 10^{-10}. (`brkdown_tol` is normally set equal to `DBL_EPSILON`, which is approximately 2.22×10^{-16} on our machine).

5. Runtime results and discussion

In this section, we present the runtime results for Problems 1–9 on 16 processors, with 512,000 equations. For each problem, the plot for CARP-CG was obtained with the optimal subdivision scheme and the optimal λ for that problem. Figs. 1–9 present a comparison of the relative residual versus the execution time for the nine test problems. In the figures, RGMRES denotes restarted GMRES. Since $x^0 = 0$, the relative residual is $\|b - Ax\|/\|b\|$. The setup time for preconditioners was added to the time of the first iteration.

The following plots are not shown, either because their convergence was too slow to be practical, or because they were too oscillatory:

Problem 1: CGS + Neumann and Bi-CGSTAB.
Problem 5: CGS + ILUT.
Problem 6: CGS + Neumann.
Problem 7: CGS + ILUT.
Problem 8: CGS + ILUT/Neumann/Least Squares.

Of the tested methods, only CGNR + GRS and CARP-CG converged on Problem 3, so we also run the original CARP algorithm [18] for comparison. The improvement of CARP-CG over CARP is very significant. Also, on Problem 3, CARP-CG's initial rate of convergence is much better than that of CGNR + GRS, but eventually CGNR + GRS exhibits slightly better convergence. Problems 3 and 7 are hard for solvers such as GMRES because they are 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 [1,7,8].

In terms of robustness, we can see that CARP-CG and CGNR + GRS converged on all the problems—a property not shared by the other methods. On Problems 1–7 and 9, CARP-CG's initial rate of convergence was very good as compared to the other methods. Also on Problems 1–7 and 9, CARP-CG was monotonic, whereas CGS and Bi-CGSTAB were oscillatory to some extent. Some very slight deviations from monotonicity appeared in CARP-CG's behavior on Problem 8 (this is not seen in the plot). On Problems 1–6 and 9, the convergence rate of CARP-CG was among the best of all the other methods. On Problem 7, CARP-CG and CGNR + GRS seem to stagnate, but they actually continued to improve at a very gradual rate. On Problem 8, GMRES and CARP-CG performed quite similarly, but both were worse than 10 other algorithm/preconditioner combinations (not all of them are shown in the figure).

Figs. 8 and 9 demonstrate that CARP-CG is particularly useful when the convection terms are very large. One should note that CARP-CG and CGNR + GRS required more iterations than the other leading methods in order to achieve a given value for the relative residuals. As a result, they are more sensitive than the other methods to the communications speed. We expect that with faster communications, CARP-CG and CGNR + GRS will perform even better.

A natural question that arises in view of the above figures is whether the relative performance shown in the figures persists when the convergence goal is raised to machine precision. Except for Problems 3 and 7, most of the methods that converged to 10^{-8} also converged to 10^{-13}, so this value was chosen as the convergence goal. Table 2 shows the time required to achieve this goal, for the remaining problems and their variants; related problems (1 and 1A, 5 and 5A, 8 and 9) are shown in proximity. Timings for CGS are not shown, because these were never better than those of Bi-CGSTAB. As to preconditioner results, only those of ILUT are shown. We can see that the line trends shown in the convergence plots continue beyond 10^{-8}; the only exception is that in Problem 6, Bi-CGSTAB + ILUT stagnated at about 1.8×10^{-12}. In Problem 5, RGMRES + ILUT overtakes CARP-CG, and in Problem 9, RGMRES overtakes CGNR + GRS. More detailed results appear in Appendix A.

The results for Problems 1A and 5A indicate that the addition of a second large convection term into the equation is detrimental to RGMRES+ILUT and Bi-CGSTAB+ILUT. This is consistent with the results obtained in [20, §5] for discontinuous coefficients. Table 3 below shows that the results on Problem 7A are consistent with those of Problems 1A and 5A: Bi-CGSTAB+ILUT performed worse than CARP-CG and CGNR+GRS on this variant. We can also see that Problem 7A is much more difficult than Problem 7 for CARP-CG and CGNR+GRS in terms of the relative residual achieved, and especially in terms of the time required.

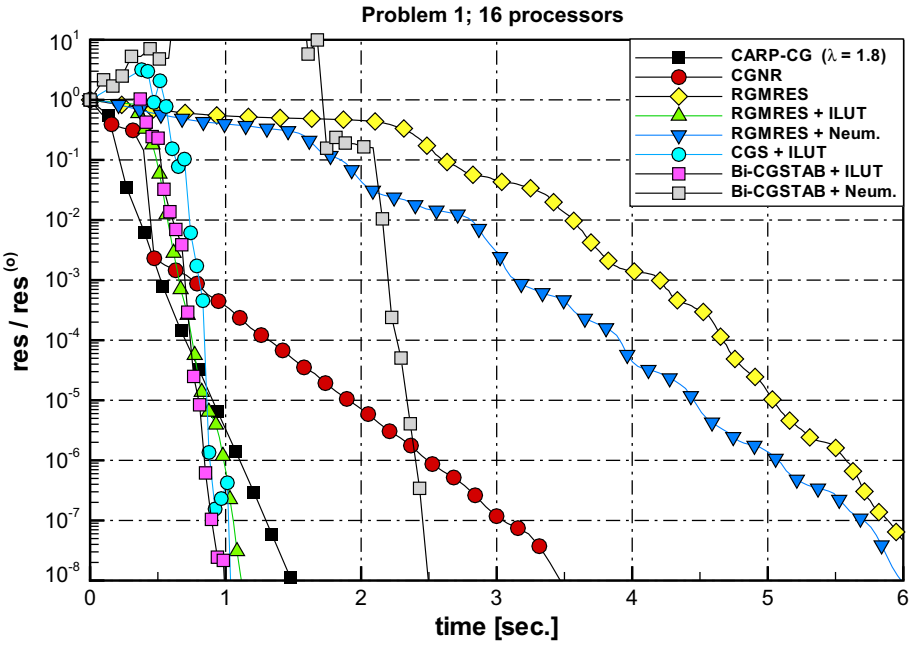

Fig. 1. Convergence results for Problem 1.

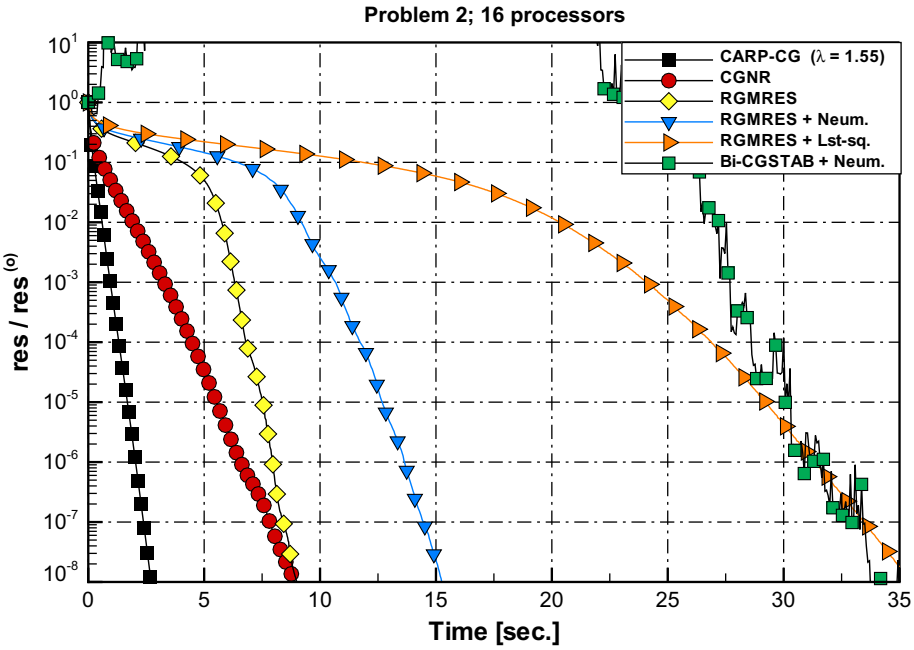

Fig. 2. Convergence results for Problem 2.

6. Additional results for CARP-CG

6.1. Relative error results

As noted in Section 1, we are interested in three types of error measures, abbreviated as follows.

- rel-res denotes the relative residual, as before.
- err-alg denotes the relative algebraic error, obtained as follows: for Problems 1–7, instead of solving the regular system $Ax = b$, we use the preassigned analytic solution u to define $b' = Au$. Then we solve the system $Ay = b'$. If y^k is the kth iterate, then err-alg is defined as $\|y^k - u\|/\|u\|$. Problems 8 and 9 are by definition algebraic problems, and we use $u = (1,\ldots,1)^T$.

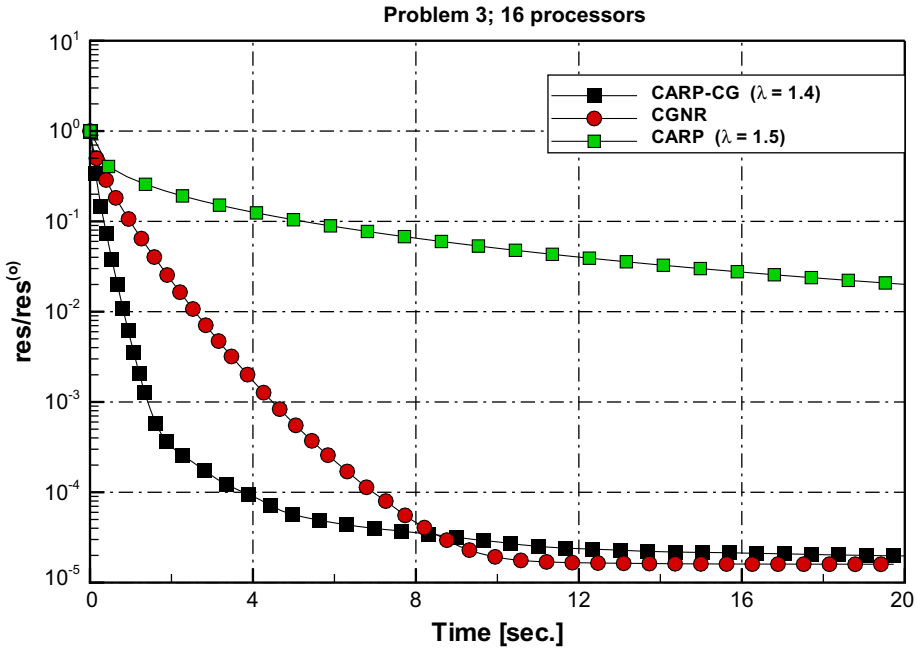

Fig. 3. Convergence results for Problem 3.

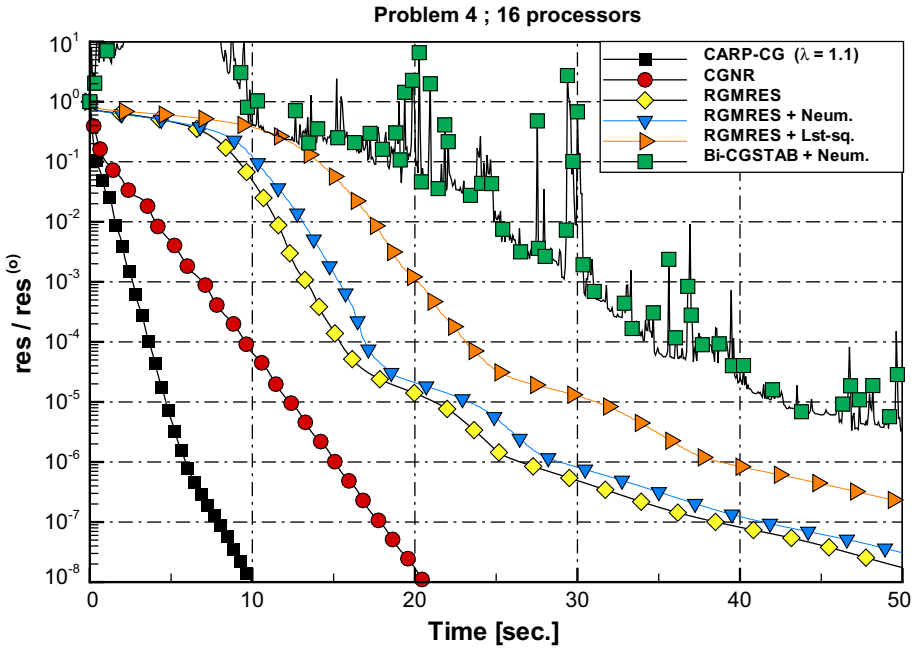

Fig. 4. Convergence results for Problem 4.

- For Problems 1–7, err-an denotes the relative error w.r.t. the analytic solution u, defined as $\|x^k - u\|/\|u\|$, where x^k is the kth iterate.

Table 4 shows the error results obtained for CARP-CG on the nine problems.
Several conclusions can be drawn from Table 4:

1. In Problems 1 and 2, all three measures are very good, indicating that CARP-CG performs very well on the algebraic system, and that the discretization scheme is very appropriate.
2. Problem 3: both err-alg and err-an are poor, though they might be good enough for some applications. For example, in solving non-linear systems by quasi-linearization, the linear system is repeatedly updated, and such accuracy might be sufficient. It should be mentioned that on this problem, CGNR + GRS achieves a relative error (err-alg) that is better by about one order of magnitude, in about the same time.

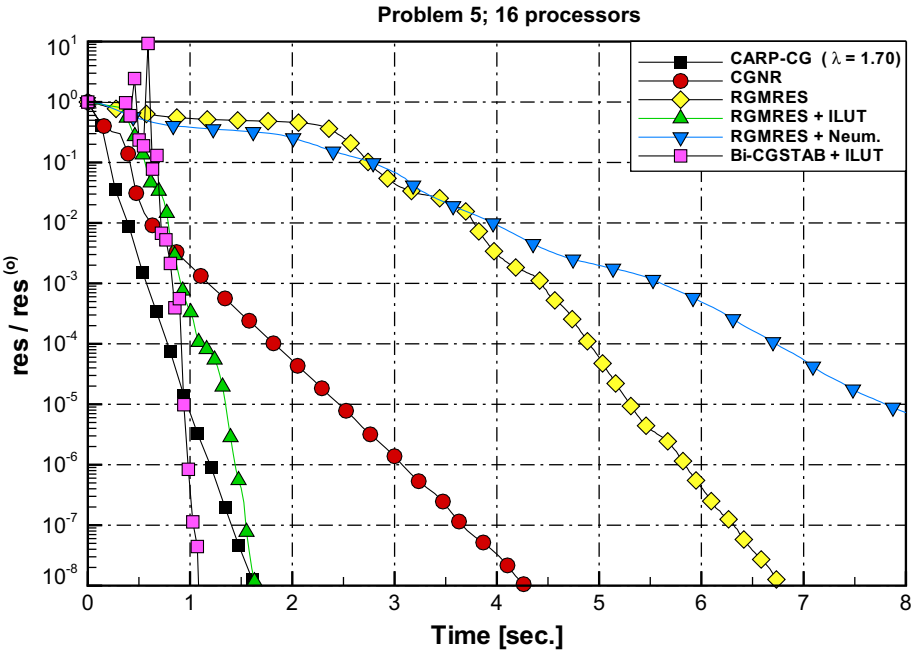

Fig. 5. Convergence results for Problem 5.

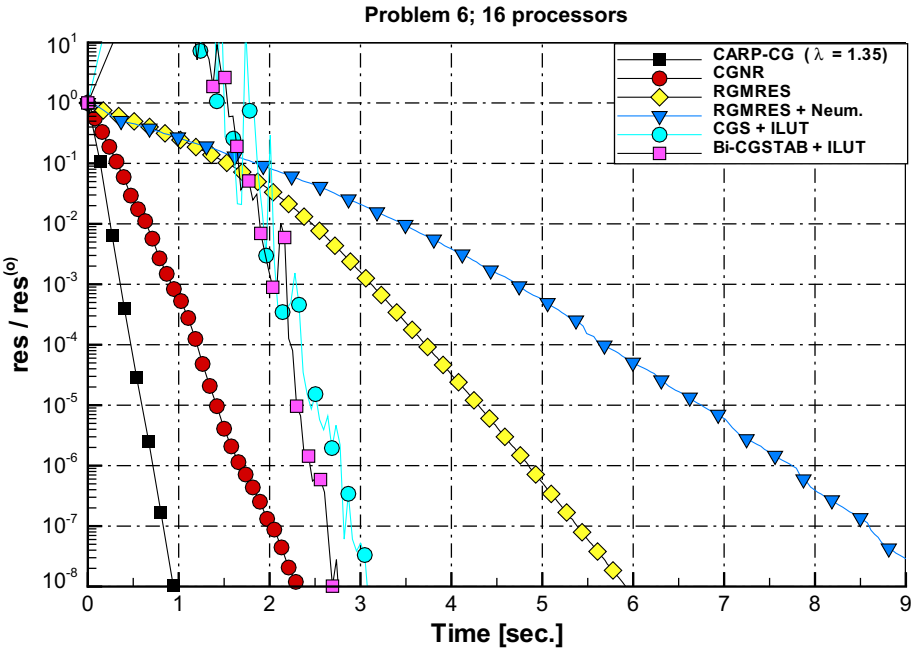

Fig. 6. Convergence results for Problem 6.

3. Problems 4, 5 and 6: CARP-CG is a very good solver of the algebraic systems, but the error compared to the analytic pre-assigned solution is much worse (though it might be good enough for some applications). This means that the discretization scheme does not represent the physical model well. Other discretizations, such as higher order schemes, might be better for these problems.
4. Problem 7: here we see quite a large discrepancy between the relative residual and the relative errors. This discrepancy should be studied further, and CARP-CG should be used on this problem with caution. Note that on this problem, Bi-CGSTAB with ILUT, when given sufficient time, achieved rel-res = 1.52×10^{-6} and err-alg = 1.71×10^{-4}.
5. Problems 8 and 9: CARP-CG achieved very good solutions to these algebraic problems. However, other methods are much more efficient on Problem 8 (with the small convection), while CARP-CG is very efficient on Problem 9 (with very large convection).

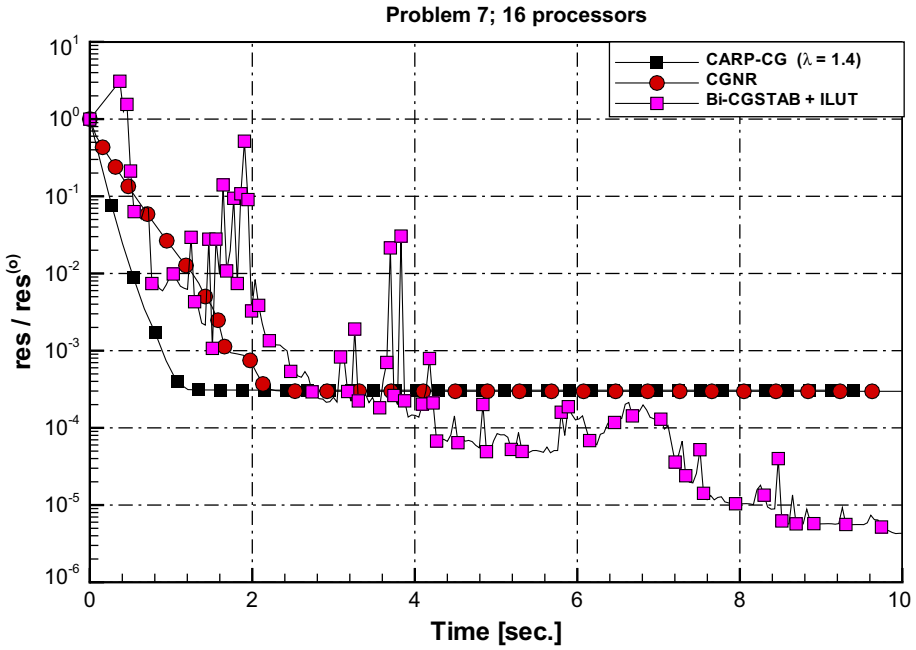

Fig. 7. Convergence results for Problem 7.

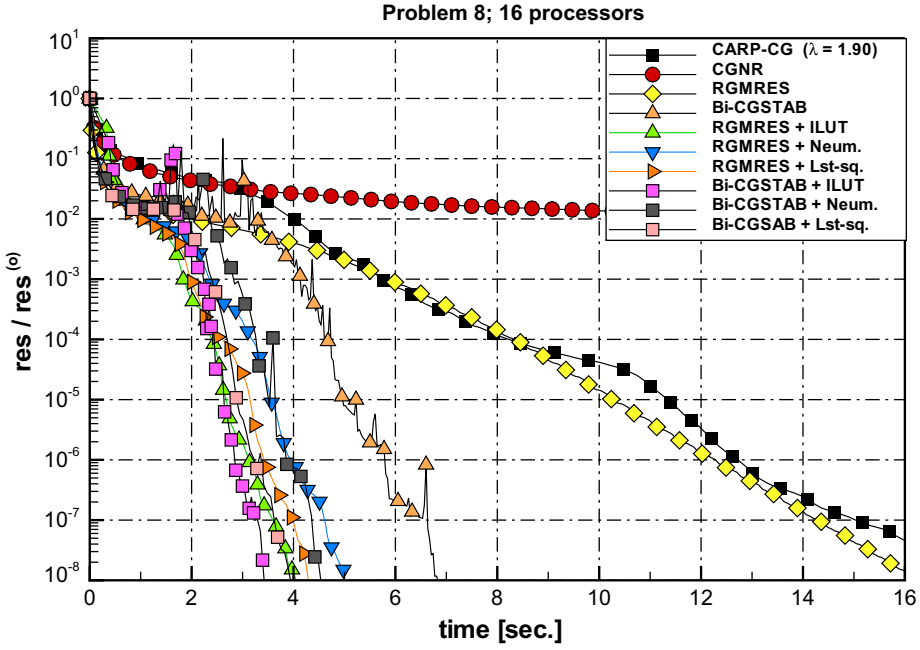

Fig. 8. Convergence results for Problem 8.

6.2. Convergence of CARP-CG as a function of λ

Fig. 10 shows how the convergence rate of CARP-CG depends on the choice of the relaxation parameter λ, for various partitionings of the equations into 16 blocks, for Problems 2, 4, 8 and 9. The plots for Problems 2, 4 and 9 are representative of all the cases, except for Problem 8. The plot for one processor is also shown for comparison. The number of iterations were the number required to achieve a relative residual below 10^{-7}.

The following observations can be made from Fig. 10 (and from similar plots for the other five problems):

1. The optimal λ does not depend strongly on the partitioning or the number of processors.
2. In some cases, the choice of a $1 \times 1 \times 16$ partitioning appears to be the worst, and the dependence of the convergence rate on λ with this partitioning is the most erratic.

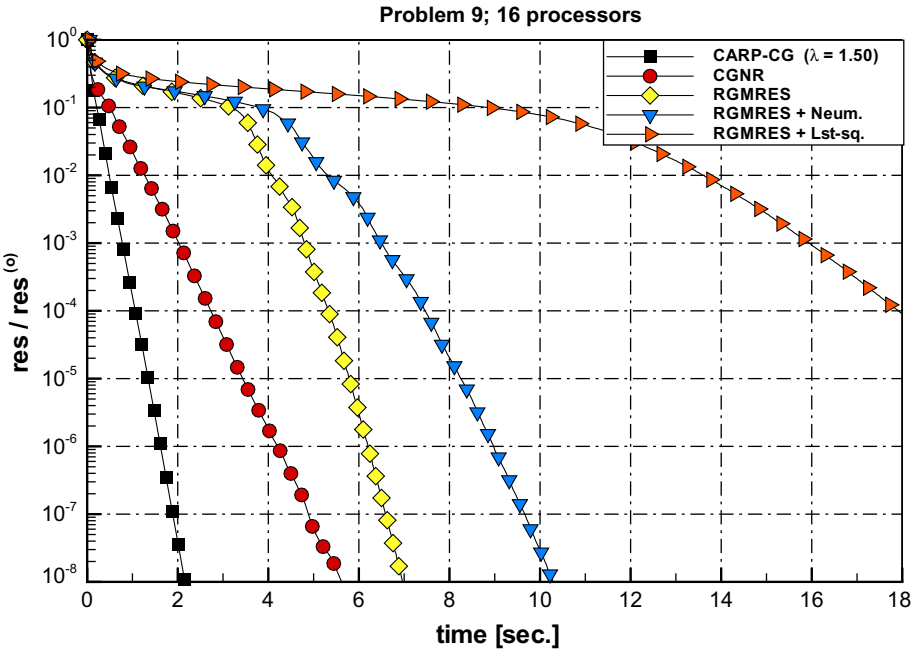

Fig. 9. Convergence results for Problem 9.

Table 2
Time to reach a relative residual of 10^{-13}. Dashes indicate no convergence and minimal times are shown in boldface.

Method	Problem								
	1	1A	2	4	5	5A	6	8	9
CARP-CG	2.48	**4.42**	**4.40**	**20.4**	2.56	**2.65**	**1.54**	29.9	**3.49**
CGNR + GRS	6.04	9.94	13.7	34.6	7.30	11.6	6.72	79.9	9.00
RGMRES	7.80	8.90	11.2	143.2	9.90	9.5	8.90	25.5	8.77
RGMRES + ILUT	1.55	–	–	–	2.33	–	–	5.84	–
Bi-CGSTAB	–	–	–	–	–	–	–	9.35	–
Bi-CGSTAB + ILUT	**1.20**	–	–	–	**1.29**	–	–	**4.79**	–

Table 3
Relative residual achieved and time on Problems 7 and 7A.

	Method					
	CARP-CG		CGNR + GRS		Bi-CGSTAB + ILUT	
Problem 7	3×10^{-4}	2	3×10^{-4}	2	5×10^{-6}	9
Problem 7A	4.8×10^{-4}	54	4.8×10^{-4}	82	1.2×10^{-3}	56

Table 4
Relative residual and errors of CARP-CG on the nine test problems.

Problem	Iteration	rel-res	err-alg	err-an
1	210	1.51E−15	1.03E−15	7.47E−16
2	380	7.57E−15	1.40E−15	1.34E−15
3	1000	1.54E−05	3.58E−03	3.50E−03
4	1750	3.80E−14	3.30E−14	4.00E−04
5	220	1.50E−14	1.95E−15	2.97E−04
6	130	7.53E−15	1.18E−15	2.40E−04
7	100	3.10E−04	6.66E−01	6.66E−01
8	2500	4.74E−14	8.31E−14	n/a
9	300	7.17E−15	2.75E−15	n/a

3. Except for case 8, the dependence is a convex function, except for the $1 \times 1 \times 16$ partitioning in some cases. Hence, finding the optimal λ is simple, and it can be easily automated.
4. Except for case 8, the function slopes on either side of the optimal λ are very mild. This means that it is not necessary to find the optimal λ with high precision. In case 8, the slopes are relatively sharper, especially after the optimum.
5. Except for case 8, the number of iterations required by CARP-CG to achieve the prescribed relative residual does not depend strongly on the number of processors. This indicates the good scalability of CARP-CG for Problems 1–7 and 9.

Table 5 shows the number of iterations required by CARP-CG for Problems 1–9, on 1, 2, 4, 8 and 16 processors. We can see that in most cases, doubling the number of processors causes only a small change in the number of iterations; i.e., CARP-CG exhibits good scalability. The only exception is Problem 8 (which does not have large off-diagonal elements). Also shown in the table are the optimal values of λ and the optimal partitioning in each case. These figures were obtained for a relative residual below 10^{-7}, except where noted otherwise.

7. Computational comparisons

In this section, we will compare the various algorithms in terms of the number of operations they require to achieve certain convergence goals, the times per iteration, and attempt to explain the efficiency of CARP-CG. Table 6 shows the number

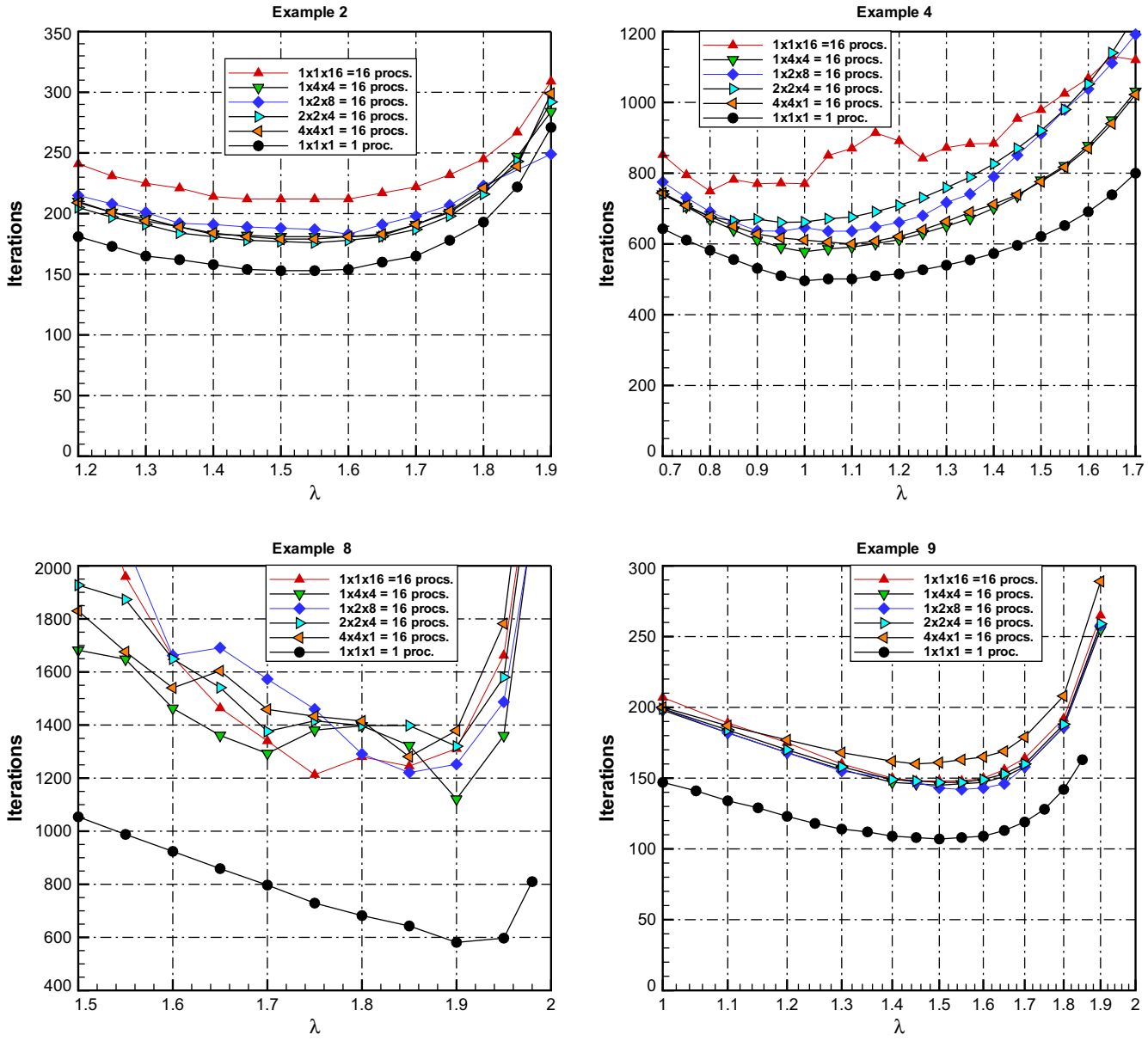

Fig. 10. Number of iterations to reach rel-res < 10^{-7}, as a function of λ.

Table 5
Number of iterations of CARP-CG to achieve a relative residual below 10^{-7} (except as noted), for varying numbers of processors. Also shown are the optimal values of λ and the partitioning at which the optimum was obtained.

	Problem								
	1	2	3[a]	4	5	6	7[b]	8	9
1 Processors	77	155	116	497	82	59	52	581	123
	1.75	1.55	160	1.00	1.75	1.30	1.70	1.90	1.50
	$1 \times 1 \times 1$	$1 \times 1 \times 1$	$1 \times 1 \times 1$	$1 \times 1 \times 1$	$1 \times 1 \times 1$	$1 \times 1 \times 1$	$1 \times 1 \times 1$	$1 \times 1 \times 1$	$1 \times 1 \times 1$
2 Processors	94	159	262	540	99	59	63	847	133
	1.80	1.55	1.40	1.00	1.75	1.35	1.70	1.90	1.50
	$1 \times 1 \times 2$	$1 \times 1 \times 2$	$1 \times 2 \times 1$	$1 \times 1 \times 2$	$1 \times 1 \times 2$	$1 \times 1 \times 2$	$2 \times 1 \times 1$	$1 \times 1 \times 2$	$1 \times 1 \times 2$
4 Processors	90	164	266	545	104	59	56	991	133
	1.80	1.55	1.40	1.00	1.75	1.35	1.60	1.90	1.50
	$1 \times 1 \times 4$	$1 \times 2 \times 2$	$2 \times 2 \times 1$	$1 \times 2 \times 2$	$1 \times 2 \times 2$	$1 \times 2 \times 2$	$4 \times 1 \times 1$	$1 \times 2 \times 2$	$1 \times 2 \times 2$
8 Processors	106	168	334	576	104	60	69	1088	138
	1.75	1.55	1.40	1.00	1.75	1.35	1.55	1.90	1.50
	$1 \times 1 \times 8$	$2 \times 2 \times 2$	$2 \times 4 \times 1$	$1 \times 2 \times 4$	$1 \times 2 \times 4$	$2 \times 2 \times 2$	$8 \times 1 \times 1$	$1 \times 2 \times 4$	$1 \times 1 \times 8$
16 Processors	97	176	282	578	105	62	77	1121	142
	1.80	1.55	1.40	1.00	1.75	1.35	1.40	1.90	1.50
	$1 \times 1 \times 16$	$2 \times 2 \times 4$	$1 \times 1 \times 16$	$1 \times 4 \times 4$	$1 \times 4 \times 4$	$2 \times 2 \times 4$	$1 \times 1 \times 16$	$1 \times 4 \times 4$	$1 \times 2 \times 8$

[a] rel-res $< 10^{-4}$.
[b] rel-res $< 5 \times 10^{-4}$.

of iterations required by CGNR + GRS, GMRES and Bi-CGSTAB to reach the same relative residuals as CARP-CG, as shown in Table 5, for one and 16 processors. Since the unpreconditioned algorithms are inherently parallel, the number of processors had no effect on the number of iterations. Some differences appeared when ILUT was used, due to AZTEC's parallel implementation of ILUT, which is actually a distributed variant of ILUT. More detailed results appear in Appendix A. In order to compare the algorithms, we first need to estimate the number of operations in each iteration.

In most unpreconditioned algorithms, one generally counts the number of matvec operations, the number of inner products of full-length vectors, and the number of operations of the type $\alpha x + y$, where α is a constant; see [3, Table 2.1]. For example, every iteration of Bi-CGSTAB consists of two matvecs, four inner products, and six $\alpha x + y$ operations. However, the sequential KACZ does not fit into this scheme, so we will simply count the number of multiplications (the number of additions is similar). Assuming a k-point stencil matrix, it is easy to see that a double KACZ sweep requires $(4k + 2)n$ multiplications. For CARP-CG, we also need to consider the fact that the regular CG requires two inner products, three $\alpha x + y$ operations, and one matvec. However, in CARP-CG, the matvec is replaced by the double KACZ sweep, so CARP-CG requires in total $(4k + 7)n$ multiplications per iteration. By comparison, Bi-CGSTAB requires $(2k + 10)n$ multiplications. Iteration i of GMRES requires one matvec, $i + 1$ inner products, and $i + 1$ operations of type $\alpha x + y$. With a restart value of 10, each cycle of 10 iterations requires 65 inner products and the same number of $\alpha x + y$ operations. This gives us an average of $(k + 13)n$ multiplications per iteration. On our problems, $k = 7$, so we get $35n$, $20n$ and $24n$ operations for CARP-CG, GMRES and Bi-CGSTAB, respectively.

Using Tables 5 and 6, we can multiply the number of iterations by the number of operations per iteration to find the total number of operations to reach convergence of 10^{-7}. On one processor, and with the exception of Problem 8, the number of operations required by CARP-CG to reach convergence of 10^{-7} is between 41% and 69% of the minimum of the unpreconditioned GMRES and Bi-CGSTAB. This explains the fact that CARP-CG is more efficient on these problems in terms of the number of operations, but we can see from the convergence plots that the time ratios are even more in favor of CARP-CG than suggested by these results.

Table 6
Number of iterations to achieve a relative residual below 10^{-7} (except as noted), for 1 and 16 processors. With ILUT, the number of iterations varies from one to 16 processors. Dashes indicate no convergence.

Method	Problem								
	1	2	3[a]	4	5	6	7[b]	8	9
CGNR + GRS	387	993	882	2264	464	257	261	6655	620
RGMRES	278	397	–	1815	298	254	–	674	311
RGMRES + ILUT	22–29	–	–	–	48–56	–	–	109–127	–
Bi-CGSTAB	904	–	–	–	–	–	–	183	–
Bi-CGSTAB + ILUT	13–14	–	–	–	16–17	53–55	48–51	55–67	–

[a] rel-res $< 10^{-4}$.
[b] rel-res $< 5 \times 10^{-4}$.

Table 7
Time per iteration of the various algorithms, their ratios relative to CARP-CG, and the iteration ratio = (time/iter for 1 processor)/(time/iter for 16 processor).

Method	1 Processor		16 Processors		Iteration ratio
	Time/iter	Ratio	Time/iter	Ratio	
CARP-CG	0.0978	1.00	1.34E−02	1.00	7.30
CGNR + GRS	0.0683	0.70	7.89E−03	0.59	8.66
RGMRES	0.1490	1.52	2.12E−02	1.58	7.03
RGMRES + ILUT	0.3250	3.32	2.60E−02	1.94	12.50
RGMRES + Neum	0.3059	3.13	3.91E−02	2.91	7.82
RGMRES + LSQ	0.3024	3.09	4.08E−02	3.04	7.41
CGS	n/a	n/a	n/a	n/a	n/a
CGS + ILUT	0.4842	4.95	4.52E−02	3.37	10.71
CGS + Neum	0.4304	4.40	6.91E−02	5.15	6.23
CGS + LSQ	0.4127	4.22	6.95E−02	5.18	5.94
Bi-CGSTAB	0.1477	1.51	3.44E−02	2.56	4.29
Bi-CGSTAB + ILUT	0.4800	4.90	4.38E−02	3.26	10.96
Bi-CGSTAB + Neum	0.4091	4.18	6.86E−02	5.11	5.96
Bi-CGSTAB + LSQ	0.4077	4.17	6.77E−02	5.04	6.02

In order to provide additional information, we present in Table 7 the actual time for one iteration, and the ratio of the time to that of CARP-CG, for one and 16 processors. The last column of the table is the time per iteration for one processor, divided by the time per iteration for 16 processors; we call this the "iteration ratio" for convenience. For the algorithms that are inherently parallel, this ratio is also the speedup (for the other algorithms, the speedup is different since the number of iterations varies with the number of processors).

Table 7 shows that CARP-CG and CGNR + GRS take less time per iteration than the other methods. In part, this is due to the efficiency of the implementations—all matrix and vector calculations in CARP-CG and CGNR + GRS were coded inline. It is also interesting to note that although CGNR + GRS is faster than CARP-CG per single iteration, CARP-CG converges faster (see Figs. 1–9). CGS without a preconditioner did not converge on any of the test cases, so there is no relevant data for it. As for the other algorithms, their runtime ratios over CARP-CG range from 1.58 (for restarted GMRES) to 5.18 (for CGS + ILUT).

We will concentrate on the relation between CARP-CG and the unpreconditioned GMRES and Bi-CGSTAB. First, we can see from Table 7 that on one processor, their ratios are approximately 1.5, and for GMRES, this does not change much for 16 processors. However, the ratio for Bi-CGSTAB goes up to 2.56 for 16 processors, and no doubt, this can be attributed to its rather poor speedup of 4.29. The runtime efficiency of CARP-CG can be explained as follows.

Among the factors influencing the actual runtime performance, memory cache utilization is of the utmost importance. In this respect, CARP-CG is especially efficient: consider the computations done in the inner loop of Definition 1 (Eq. (4)). Except for the initialization of CGMNC, this equation is executed with $b_i = 0$, and it is done in two steps. First, we compute $c = \lambda_i \langle a_i^*, y^{i-1} \rangle$ (c is some variable); then, we set $y^i = y^{i-1} - c a_i^*$. After the first step, the vectors y^{i-1} and a_i^* are already in cache and ready for the second step (in the DMSR format, the nonzero elements of a_i^*, except for a_{ii}, occupy adjacent memory locations, and these are followed by the nonzero elements of a_{i+1}^*). Now consider also what happens at the next step of the iteration, $i + 1$: y^i is already in cache, and with very high probability, the nonzero elements of a_{i+1}^* will also be in cache. Furthermore, on one processor, the forward sweep is followed immediately by the back sweep, which operates on the same vector used by forward sweep. Hence, the double Kaczmarz sweep utilizes the cache extremely well. Of the $(4k + 7)n$ multiplications required by CGMNC, $(4k + 2)n$ are due to the double Kaczmarz sweep, so for $k = 7$, about 86% of the computations are done on cached data. On several processors, the Kaczmarz iterations are performed internally in each processor and do not require any inter-processor communications.

8. Conclusions and further research

This paper introduced the block-parallel algorithm CARP-CG for solving large sparse linear systems. CARP-CG, which is a CG-acceleration of CARP [18], requires an extension of the sequential CGMN algorithm of Björck and Elfving [4] in order to allow cyclic relaxation parameters, as required by the CARP theory. CARP-CG uses only row projections and component-averaging operations and thus avoids the need to find independent sets of equations as in the parallel block-projection methods of previous approaches. This property makes it a convenient algorithm to use with unstructured grids. Another property is the guaranteed theoretical convergence of CARP-CG, even on inconsistent and/or nonsquare systems of equations.

CARP-CG was compared with restarted GMRES, CGS and Bi-CGSTAB, which were used with and without preconditioners. The comparisons were made on nine well-known linear systems derived from convection–diffusion elliptic PDEs. CARP-CG converged in all the test cases, while none of the other algorithm/preconditioner combinations achieved this level of robustness. CGNR + GRS (which is actually a CG-acceleration of the Cimmino algorithm) was also tested, and although it was very robust, it was less efficient than CARP-CG.

Eight of the problems were strongly convection dominated, and on these problems, the runtime of CARP-CG was very competitive with respect to the other algorithm/preconditioner combinations. Also, on these eight problems, CARP-CG'S initial rate of convergence was very fast, and the achieved relative residual decreased monotonically. These properties indicate that CARP-CG is potentially useful in methods for solving non-linear convection dominated problems in which solving a linear system is a frequent intermediate step. Such problems occur frequently in CFD applications when the Péclet number is large. CARP-CG scaled well on these eight problems, in the sense that the number of iterations did not increase much with the number of processors.

To a very small extent, the performance of CARP-CG depends on the partitioning of the problem into subdomains. CARP-CG also requires a relaxation parameter λ to achieve optimal results. In most cases, small deviations from the optimal λ have a very small effect on the rate of convergence. Furthermore, the dependence of the convergence rate on the choice of λ is, in most cases, a convex function, so the optimal λ can be determined quite efficiently and even automatically. CARP-CG is more memory efficient than GMRES because there is no need to store the Krylov subspace basis. Furthermore, CGS and Bi-CGSTAB are oscillatory to some extent, whereas CARP-CG is monotonic (except for a very slight deviation in the one test case which was not strongly convection dominated).

Another factor to note is that CARP-CG and CGNR + GRS require more iterations to achieve certain prescribed convergence criteria than the other leading methods/preconditioners. Thus, their rate of convergence on a parallel system, relative to other methods, depends on the communications speed; on a high-bandwidth parallel system (or on a single processor) they

Table 8
Results for 1 and 16 processors. Top: no. of iterations; bottom: runtime in sec.

Problem	rel-res	CARP-CG	CGNR + GRS	RGMRES	RGMRES + ILUT	Bi-CGSTAB	Bi-CGSTAB + ILUT
1	1.E−7	77–97	387	278	22–29	904	13–14
		7.5–1.30	26.4–3.05	41.4–5.9	15.6–1.06	133.5–31.1	14.7–0.94
	1.E−10	111–143	474	330	31–39	–	17–17
		10.8–1.92	32.3–3.74	49.1–7.0	18.5–1.32		16.6–1.07
	1.E−13	149–185	765	369	40–48	–	21–20
		14.5–2.48	52.2–6.04	55.0–7.8	21.5–1.55		18.5–1.20
2	1.E−7	155–176	993	397	–	–	–
		15.1–2.36	67.8–7.84	59.1–8.4			
	1.E−10	220–251	1389	462	–	–	–
		21.5–3.37	94.8–10.9	68.8–9.81			
	1.E−13	285–329	1734	528	–	–	–
		37.8–4.4	118–13.7	78.6–11.2			
4	1.E−7	497–578	2264	1815	–	–	–
		48.6–7.8	155–17.8	270–38.5			
	1.E−10	873–1049	3307	4213	–	–	–
		85.4–14.1	226–26.1	628–89.5			
	1.E−13	1324–1521	4389	6741	–	–	–
		130–20.4	300–34.6	1004–143.2			
5	1.E−7	82–105	464	298	56–48	–	16–17
		8.0–1.41	31.7–3.66	44.4–6.3	26.7–1.55		16.1–1.07
	1.E−10	120–148	702	359	69–61	–	19–19
		11.7–1.98	47.9–5.54	53.5–7.6	30.9–1.89		17.5–1.16
	1.E−13	157–191	925	467	80–78	–	21–22
		15.3–2.56	63.1–7.3	69.6–9.9	34.5–2.33		18.5–1.29
6	1.E−7	59–62	257	254	–	–	54–55
		5.77–0.83	17.5–2.03	37.8–5.39			33.4–2.73
	1.E−10	80–89	368	334	–	–	(no −61
		7.8–1.19	25.1–2.91	49.7–7.09			conv) - 3.0
	1.E−13	117	484	418	–	–	–
		11.4–1.57	33.0–6.72	62.3–8.9			
8	1.E−7	581–1121	6655	674	109–127	183	55–67
		56.8–15.0	454–52.5	100–14.3	43.9–3.61	27.0–6.29	34.8–3.26
	1.E−10	797–1653	8430	944	150–179	228	66–83
		78.0–22.2	576–66.5	141–20.0	57.9–4.96	33.7–7.84	40.1–3.96
	1.E−13	951–2230	10,130[a]	1201	194–213	272	83–102
		93.0–29.9	692–79.9	179–25.5	72.2–5.84	40.2–9.35	48.3–4.79
9	1.E−7	123–142	620	311	–	–	–
		12.0–1.91	42.3–4.89	46.3–6.61			
	1.E−10	179–204	887	363	–	–	–
		17.5–2.74	60.5–7.0	54.0–7.7			
	1.E−13	229–260	1143	413	–	–	–
		22.4–3.49	78.0–9.0	61.5–8.77			

[a] CGNR + GRS on Problem 8 converged only to 1.08E−13.

will perform relatively better than on a low-bandwidth system. Of course, CARP-CG on a single processor and with a fixed relaxation parameter is simply the CGMN algorithm [4]. Similar experiments to the ones described in this paper were carried out with CGMN on a single processor, with very good results [19]. Note that although CGNR is not generally considered useful (see [34, Section 8.3.1]), CGNR + GRS performed quite well on the eight convection dominated problems. It is therefore a reasonable second choice, particularly in view of the fact that it is inherently parallel and parameter-free.

Our results raise the question of why CARP-CG (and to a lesser extent, CGNR + GRS) are particularly efficient on the strongly convection dominated cases. The system matrices of these cases are problematic because they have very large off-diagonal elements. However, CARP-CG and CGNR + GRS first scale the equations by the L_2-norm of the coefficients, so the diagonal elements of AA^T are larger than the off-diagonal ones; see [20, Theorem 1].

Several research directions and further applications are suggested by this work:

- CARP-CG can be viewed as a preconditioner for CG. However, it uses unpreconditioned CG, and this raises the question of whether it can benefit from preconditioned CG. As noted (but not recommended) by one of the reviewers, a polynomial preconditioner can be easily worked into the CARP-CG code. There is a strong motivation for this research, especially for indefinite problems, such as Problems 3, 7 and 7A. Although CARP-CG performed quite well on these problems relatively to the other methods, the results still leave a lot to be desired in terms of the residual achieved and the rate of convergence.
- Application of CARP-CG to other problems requiring massive parallel computations, such as computerized tomography and electron tomography [14].
- Application of CARP-CG as a solver of intermediate linear systems in various applications. For example, quasi-linear solution methods in non-linear CFD problems, and eigenvalue problems.
- Research of the component-averaging domain decomposition (CADD) method, such as replacing KACZ with some other algorithm. Main issues of research include conditions for convergence, symmetrizability, and performance compared to other DD methods.
- Potential applications of CARP-CG to least squares problems.

Acknowledgments

Many thanks are due to the anonymous reviewers for their careful reading of the manuscript and their many useful and instructive comments.

Appendix A. Detailed runtime results

Table 8 presents data about the number of iterations and the runtimes, on one and 16 processors, required to reach three convergence goals: 10^{-7}, 10^{-10}, 10^{-13}. In places where the numbers vary according to the number of processors, both numbers are provided. On Problem 6, Bi-CGSTAB + ILUT on one processor could reach only an accuracy of 1.2×10^{-9}, but on 16 processors, it reached the convergence goal of 10^{-10}. This is probably due to the difference between ILUT (on one processor) and the distributed version of ILUT on 16 processors.

References

[1] M. Arioli, I.S. Duff, J. Noailles, D. Ruiz, A block projection method for sparse matrices, SIAM Journal on Scientific & Statistical Computing 13 (1992) 47–70.
[2] M. Arioli, I.S. Duff, D. Ruiz, M. Sadkane, Block Lanczos techniques for accelerating the block Cimmino method, SIAM Journal on Scientific & Statistical Computing 16 (1995) 1478–1511.
[3] R. Barrett, M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H.A. van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, second ed., SIAM, Philadelphia, PA, 1994. Available from: <http://netlib2.cs.utk.edu/linalg/html_templates/Templates.html>.
[4] Å. Björck, T. Elfving, Accelerated projection methods for computing pseudo-inverse solutions of systems of linear equations, BIT 19 (1979) 145–163.
[5] BLAS – Basic Linear Algebra Subprograms. Available from: <http://www.netlib.org/blas>.
[6] R. Bramley, H.-C. Chen, U. Meier, A. Sameh, On some parallel preconditioned CG schemes, in: O. Axelsson, L.Yu. Kolotilina (Eds.), Proceedings of the International Conference on Preconditioned Conjugate Gradient Methods, Nijmegen, The Netherlands, June 1989, Lecture Notes in Mathematics, vol. 1457, Springer, Berlin, 1990, pp. 17–27.
[7] R. Bramley, A. Sameh, Domain decomposition for parallel row projection algorithms, Applied Numerical Mathematics 8 (1991) 303–315.
[8] R. Bramley, A. Sameh, Row projection methods for large nonsymmetric linear systems, SIAM Journal on Scientific & Statistical Computing 13 (1992) 168–193.
[9] G. Cimmino, Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari, La Ricerca Scientifica XVI, Series II, Anno IX 1 (1938) 326–333.
[10] P.P.B. Eggermont, G.T. Herman, A. Lent, Iterative algorithms for large partitioned linear systems, with applications to image reconstruction, Linear Algebra & Its Applications 40 (1981) 37–67.
[11] T. Elfving, Block-iterative methods for consistent and inconsistent linear equations, Numerische Mathematik 35 (1980) 1–12.
[12] H. Elman, Iterative methods for large, sparse, nonsymmetric systems of linear equations. Technical Report 229, Department of Computer Science, Yale University, New Haven, CT, USA, 1982.
[13] H. Elman, G. Golub, Iterative methods for cyclically reduced non-self-adjoint linear systems, Mathematics of Computation 54 (190) (1990) 671–700.
[14] J.-J. Fernández, D. Gordon, R. Gordon, Efficient parallel implementation of iterative reconstruction algorithms for electron tomography, Journal of Parallel & Distributed Computing 68 (5) (2008) 626–640.

[15] J.H. Ferziger, M. Perić, Computational Methods for Fluid Dynamics, third ed., Springer, Berlin, 2002.
[16] R. Fletcher, Conjugate gradient methods for indefinite systems, in: G.A. Watson (Ed.), Proceedings of the Dundee Biennial Conference on Numerical Analysis, 1975, Lecture Notes in Mathematics, vol. 506, Springer, Berlin, 1976, pp. 73–89.
[17] R. Freund, N. Nachtigal, QMR: a quasi-minimal residual method for non-Hermitian linear systems, Numerische Mathematik 60 (1991) 315–339.
[18] D. Gordon, R. Gordon, Component-averaged row projections: a robust, block-parallel scheme for sparse linear systems, SIAM Journal on Scientific Computing 27 (2005) 1092–1117.
[19] D. Gordon, R. Gordon, CGMN revisited: robust and efficient solution of stiff linear systems derived from elliptic partial differential equations, ACM Transactions on Mathematical Software 35 (3) (2008) 18:1–18:27.
[20] D. Gordon, R. Gordon, Solution methods for linear systems with large off-diagonal elements and discontinuous coefficients, Computer Modeling in Engineering & Sciences 53 (1) (2009) 23–45.
[21] D. Gordon, R. Gordon, Geometric scaling: a simple and effective preconditioner for some linear systems with discontinuous coefficients, Journal of Computational & Applied Mathematics, in press. doi:10.1016/j.cam.2010.05.021.
[22] P. Gresho, R. Sani, Incompressible Flow and the Finite Element Method: Advection-Diffusion and Isothermal Laminar Flow, Wiley, Chichester, 1998.
[23] M.R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, Journal of Research of the National Bureau of Standards 49 (1952) 409–436.
[24] S. Kaczmarz, Angenäherte Auflösung von Systemen linearer Gleichungen, Bulletin de l'Académie Polonaise des Sciences et Lettres A35 (1937) 355–357.
[25] C. Kamath, A. Sameh, A projection method for solving non-symmetric linear systems on multi-processors, Parallel Computing 9 (3) (1989) 291–312.
[26] C. Kamath, S. Weeratunga, Implementation of two projection methods on a shared memory multiprocessor: DEC VAX 6240, Parallel Computing 16 (1990) 375–382.
[27] C. Kamath, S. Weeratunga, Projection methods for the numerical solution of non-self-adjoint elliptic partial differential equations, Numerical Methods for Partial Differential Equations 8 (1992) 59–76.
[28] D. Kincaid, D. Young, Adapting iterative algorithms developed for symmetric systems to non-symmetric systems, in: M. Schultz (Ed.), Elliptic Problem Solvers, Academic Press, New York, 1981, pp. 353–359.
[29] D. Kuck, E. Davidson, D. Lawrie, A. Sameh, Parallel super-computing today and the cedar approach, Science 231 (1986) 967–974.
[30] C. Lanczos, Solution of systems of linear equations by minimized iterations, Journal of Research of the National Bureau of Standards 49 (1952) 33–53.
[31] G. Meurant, Computer Solution of Large Linear Systems, Elsevier, Amsterdam, 1999.
[32] Y. Saad, Practical use of polynomial preconditionings for the conjugate gradient method, SIAM Journal on Scientific & Statistical Computing 6 (1985) 865–881.
[33] Y. Saad, SPARSKIT: a basic tool kit for sparse matrix computations. Technical Report RIACS-90-20, Research Institute for Advanced Computer Science, NASA Ames Research Center, Moffet Field, CA, 1990.
[34] Y. Saad, Iterative Methods for Sparse Linear Systems, second ed., SIAM, Philadelphia, PA, 2003.
[35] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM Journal on Scientific & Statistical Computing 7 (1986) 856–869.
[36] P. Sonneveld, CGS: a fast Lanczos-type solver for nonsymmetric linear systems, SIAM Journal on Scientific & Statistical Computing 10 (1989) 36–52.
[37] R.S. Tuminaro, M.A. Heroux, S.A. Hutchinson, J.N. Shadid, AZTEC user's guide. Technical Report SAND99-8801J, Sandia National Laboratories, Albuquerque, New Mexico, 1999.
[38] H.A. van der Vorst, Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, SIAM Journal on Scientific & Statistical Computing 13 (1992) 631–644.