(This is a sample cover image for this issue. The actual cover is not yet available at this time.)

Journal of Computational and Applied Mathematics 237 (2013) 182–196

Contents lists available at SciVerse ScienceDirect

Journal of Computational and Applied Mathematics

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

Robust and highly scalable parallel solution of the Helmholtz equation with large wave numbers

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 27 January 2011
Received in revised form 3 January 2012

Keywords:
CARP-CG
Helmholtz equation
High frequency
Large wave numbers
Parallel processing
Partial differential equations

ABSTRACT

Numerical solution of the Helmholtz equation is a challenging computational task, particularly when the wave number is large. For two-dimensional problems, direct methods provide satisfactory solutions, but large three-dimensional problems become unmanageable.

In this work, the block-parallel CARP-CG algorithm [Parallel Computing 36, 2010] is applied to the Helmholtz equation with large wave numbers. The effectiveness of this algorithm is shown both theoretically and practically, with numerical experiments on two- and three-dimensional domains, including heterogeneous cases, and a wide range of wave numbers. A second-order finite difference discretization scheme is used, leading to a complex, nonsymmetric and indefinite linear system.

CARP-CG is both robust and efficient on the tested problems. On a fixed grid, its scalability improves as the wave number increases. Additionally, when the number of grid points per wavelength is fixed, the number of iterations increases linearly with the wave number. Convergence rates for heterogeneous cases are similar to those of homogeneous cases. CARP-CG also outperforms, at all wave numbers, one of the leading methods, based on the shifted Laplacian preconditioner with a complex shift and solved with a multigrid.

1. Introduction

Numerical solution of the Helmholtz equation at high frequencies (large wave numbers) is a challenging computational task due to the indefiniteness of the resulting linear system. For two-dimensional problems, direct methods are known to provide satisfactory solutions, but large three-dimensional problems become unmanageable due to the large number of elements added by the fill-in. Duff et al. [1] combine direct and iterative methods with multigrid (MG) based preconditioners to produce a hybrid technique which is more efficient than either method on two-dimensional problems. MG methods are difficult to apply directly to the Helmholtz equation due to the indefiniteness of the linear system, and the fact that the coarse grids must remain sufficiently fine due to the oscillatory nature of the solution.

Bayliss et al. [2] introduced a novel approach to solving the Helmholtz equation by using a shifted Laplacian as a preconditioner. Erlangga et al. [3] introduced a complex shift into the Laplacian, and, in [4], used an MG to solve the preconditioner. This approach was parallelized by Riyanti et al. in [5], and Knibbe et al. [6] introduced the use of a GPU. Erlangga and Turkel [7] used this method with fourth-order and sixth-order discretization in two dimensions. For a recent survey on this topic and some new results, see [8]. One of the problems with this approach is that an MG may be difficult to implement on unstructured grids.

* Corresponding author.
E-mail addresses: gordon@cs.haifa.ac.il (D. Gordon), rgordon@tx.technion.ac.il (R. Gordon).

0377-0427/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.cam.2012.07.024

The following are some of the many alternative approaches. Elman et al. [9] combine an MG with GMRES smoothing to overcome the coarse grid problem. Chen and Harris [10] examine operator splitting and approximate inverse preconditioners with CGNR and GMRES solvers. Plessix and Mulder [11] introduce a separation of variables technique for solving the Helmholtz equation in heterogeneous media; however, this approach is feasible only if the wave number has such a separable form. Kim et al. [12] use a high-frequency asymptotic decomposition of the wave-field to solve high-frequency Helmholtz equations. Bollhöfer et al. [13] introduced a new approach, based on an algebraic multilevel preconditioner, and using maximum weight matchings and an inverse-based pivoting strategy. Osei-Kuffuor and Saad [14] first discretize the equation, and then obtain the preconditioner by adding the imaginary shift to the diagonal of the resulting matrix.

This work presents numerical experiments with the block-parallel CARP-CG algorithm [15], applied to the Helmholtz equation with medium and large wave numbers. CARP-CG is a conjugate-gradients (CG) acceleration of CARP (component-averaged row projections) [16], which is a domain decomposition approach to parallelizing the Kaczmarz algorithm [17]. On one processor, CARP-CG is identical to the CGMN algorithm of Björck and Elfving [18]; see also [19]. CARP-CG was shown to be a very robust and efficient parallel solver of linear systems with very large off-diagonal elements and discontinuous coefficients, obtained from strongly convection dominated elliptic PDEs in heterogeneous media [20]. It was shown in [20] that standard methods, such as GMRES and Bi-CGSTAB, with ILU(0), can handle such problems provided the system matrix has at most two large off-diagonal elements in a single row. However, when there are four or six such elements, then these methods either fail completely or take a very long time compared to CARP-CG.

CARP-CG is a simple method and easy to implement on structured and unstructured grids. We provide theoretical explanations for its ability to handle linear systems with discontinuous coefficients and very large off-diagonal elements.

Numerical experiments were carried out on two- and three-dimensional domains, with wave numbers up to $k = 600$, including heterogeneous cases. A second-order finite difference discretization scheme was used, leading to a complex, nonsymmetric and indefinite linear system. Experiments were carried out with up to 32 processors, and a fixed relaxation parameter of 1.5 was used in all cases.

The results demonstrate the robustness and runtime efficiency of CARP-CG on the tested problems. A most significant result of these experiments is that, on a fixed grid, its scalability improves as k increases: for $k = 75$, the number of iterations on 32 processors was about 58% more than required on one processor, while, for $k = 600$, only 6% more iterations were required, and there was very little variance in this result when the convergence goal or the number of grid points per wavelength was changed.

Additionally, when the number of grid points per wavelength is fixed, the number of iterations increases linearly with the wave number. Convergence rates for heterogeneous cases were similar to those of the homogeneous cases. CARP-CG on one processor (CGMN) was also compared with results from [5,6], and proved to be more efficient at all wave numbers.

The rest of this paper is organized as follows. Section 2 describes the CARP-CG algorithm and presents some points explaining its robustness. Section 3 describes the wave equation, and Section 4 explains the setup of the numerical experiments. Section 5 presents the results, and Section 6 compares CARP-CG with the results of [5,6]. Section 7 concludes with some future research directions.

2. Exposition of the CARP-CG algorithm

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 $p^T q$. By $\|\cdots\|$ 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.

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

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

One can construct from (1) the related "normal equations" system

$$AA^T y = b, \qquad x = A^T y. \tag{2}$$

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

The Kaczmarz algorithm (KACZ) [17] successively projects the current iterate onto a hyperplane defined by one of the equations, as follows. Let $1 \le i \le m$ and $x = (x_1, \ldots, x_n)$ be the current iterate. The projection of x onto the ith hyperplane yields a new iterate x' defined by

$$x' = x + \omega \frac{b_i - \langle a_{i*}, x\rangle}{\|a_{i*}\|^2} a_{i*}, \tag{3}$$

where $0 < \omega < 2$ is a relaxation parameter which may depend on i. (We use ω here because λ will be used for the wavelength.) Clearly, identical results are obtained if it is assumed that the equations have already been normalized, i.e., for $1 \le i \le m$, the ith equation has been divided by $\|a_{i*}\|$ as a preprocessing step. So, we shall assume henceforth that this has been done; note that this is also more efficient in practice.

KACZ starts from an arbitrary initial estimate and consists of successive projections onto the hyperplanes. A sequence of such projections, done in the order $1, 2, \ldots, m$, will be called a (forward) KACZ sweep. We introduce KSWP, BKSWP, and DKSWP as three operators describing the forward KACZ sweep, the backward sweep, and the double sweep.

Definition 1. Let A and b be as in (1) (after normalization), $\Omega = (\omega_1, \ldots, \omega_m)$ a vector of relaxation parameters, and $x \in \mathbb{R}^n$. Then KSWP(A, b, x, Ω) is the following operator.

operator KSWP(A, b, x, Ω)
  set $y^0 = x$
  for $i = 1$ **to** m **do**
    $\left\{y^i = y^{i-1} + \omega_i \left(b_i - \langle a_{i*}, y^{i-1}\rangle\right) a_{i*}\right\}$ (4)
  return y^m
end operator

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

It is well known that KACZ is actually SOR (successive over-relaxation) applied to the normal equations system (2), and their respective relaxation parameters are identical, so KACZ is also known as SOR-NE. Using cyclic relaxation parameters as in Definition 1, we call the result SORC-NE.

Algorithm 1 (*KACZ, SORC-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$(A, b, x^k, \Omega)\}$

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 (2) will be denoted by SSOR-NE, and, with cyclic relaxation parameters, we call it SSORC-NE. Its form is identical to Algorithm 1, except that KSWP is replaced by DKSWP.

2.2. The CARP algorithm

CARP [16] 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. The KACZ iterations can use cyclic relaxation parameters. Normally, the blocks of equations are chosen by partitioning the domain into subdomains, and each block contains the equations of a single subdomain. Each block of equations is assigned to a single processor, so communications between processors are limited to the boundaries between subdomains. Thus, CARP is actually a type of domain decomposition (DD) technique. The difference between CARP and standard DD methods is that, due to the averaging operations, boundary values of a subdomain are changed by the internal computations in a neighboring subdomain.

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 $1 \le \ell \le t$, let A^ℓ, b^ℓ denote the matrix and the right-hand side, respectively, of the equations in B_ℓ. For $1 \le j \le n$, we denote $L_j = \{1 \le \ell \le t \mid B_\ell$ 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_j 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 $1 \le \ell \le t$.

Given $\mathscr{B}$ and a vector of relaxation parameters $\Omega = (\omega_1, \ldots, \omega_m)$, we denote by Ω_ℓ, for $1 \le \ell \le t$, the subsequence of Ω corresponding to the equations of B_ℓ. Let $d \ge 1$ 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 $1 \le \ell \le t$ in parallel **do**
$\{y^\ell = \text{KSWP}^d(A^\ell, b^\ell, x^k, \Omega_\ell)\}$
set $x^{k+1} = \text{CA}_{\mathscr{B}}(y^1, \ldots, y^t)$
}
end algorithm

A detailed parallel implementation of CARP is given in [16, Appendix A].

Some additional details will be needed to explain CARP-CG. Let $s = \sum_{j=1}^{n} s_j$. It is shown in [16] 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 $1 \le \ell \le s_j$. 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 $1 \le j \le n$, all the components of $y \in \mathbb{R}^s$ which correspond to x_j, i.e., replacing y by $y' \in \mathbb{R}^s$, where

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

Furthermore, it is shown in [15, Lemma 2] (the Generalized Averaging Lemma) that such averaging operations are equivalent to KACZ row projections in $\mathbb{R}^s$ onto the hyperplanes defined by certain 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 ω is used on the original equations). 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. If A is square, then so is A'.

As noted in [16], CARP converges cyclically, but since it is always executed with complete sweeps, it always converges according to [21, Theorem 1.1]. If 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.

2.3. The CGMNC algorithm

When the system matrix A of (1) is symmetric and positive definite, one can apply the Conjugate Gradient algorithm (CG), which converges in theory to a solution of (1). Björck and Elfving [18, Lemma 5.1] show that the same holds true if A is only positive semidefinite, provided the system (1) is consistent. The CGMN algorithm [18] is a CG acceleration of KACZ obtained by running it in a double sweep, as in DKSWP; see also [19].

In [15], CGMN is extended to CGMNC, which allows cyclic relaxation parameters. This is done by showing that a complete double sweep of SSORC-NE has the following iterative form:

$$x^{k+1} = Qx^k + Rb. \tag{6}$$

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

$$Qx + Rb = \text{DKSWP}(A, b, x, \Omega). \tag{7}$$

We now consider the following linear system related to the iteration (6):

$$(I - Q)x = Rb. \tag{8}$$

It is shown in [15, Theorem 1] that system (8) is consistent, and that $(I - Q)$ is symmetric and positive semidefinite. Hence, from [18, Lemma 5.1], it follows that CG can be applied to (8), and its theoretical convergence is guaranteed. The resulting algorithm, which we call CGMNC, uses the identity (7) instead of actual matrix-vector multiplications, as follows.

- If the initial estimate for CG is x^0, then the initial residual is
$$r^0 = Rb - (I - Q)x^0 = Qx^0 + Rb - x^0 = \text{DKSWP}(A, b, x^0, \Omega) - x^0.$$
- At each iteration, if p is vector for which we want to calculate $(I - Q)p$, we use the identity
$$(I - Q)p = p - \text{DKSWP}(A, 0, p, \Omega),$$
i.e., we use Eq. (7) with $b = 0$.

It follows from [15, Theorem 1] that CGMNC always converges, even when system (1) is inconsistent and/or non-square.

2.4. The CARP-CG algorithm

Since CARP in $\mathbb{R}^n$ is equivalent to KACZ in $\mathbb{R}^s$, we can apply CGMNC to the system $A'y = b'$ in $\mathbb{R}^s$. It follows that we can replace the double sweep of CGMNC in $\mathbb{R}^s$ by a double sweep of CARP in $\mathbb{R}^n$. Furthermore, experiments indicate that it 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 is the following sequence of operations: (forward sweep, averaging, backward sweep, averaging). These operations are formalized by the following operator DCSWP (double CARP sweep).

Definition 3. Let A, b, $\mathscr{B}$, Ω, Ω_ℓ $(1 \leq \ell \leq t)$ be as before, and let $x \in \mathbb{R}^n$. Then DCSWP(A, b, $\mathscr{B}$, x, Ω) is the following operator.

operator DCSWP(A, b, $\mathscr{B}$, x, Ω)
  foreach $1 \leq \ell \leq t$ in parallel **do**
    $\{y^\ell = \text{KSWP}(A^\ell, b^\ell, x, \Omega_\ell)\}$
  set $x' = \text{CA}_{\mathscr{B}}(y^1, \ldots, y^t)$
  foreach $1 \leq \ell \leq t$ in parallel **do**
    $\{y^\ell = \text{BKSWP}(A^\ell, b^\ell, x', \Omega_\ell)\}$
  return $\text{CA}_{\mathscr{B}}(y^1, \ldots, y^t)$
end operator

Based on all the above, DCSWP(A, b, $\mathscr{B}$, x, Ω) in $\mathbb{R}^n$ is equivalent to DKSWP(A', b', y, Ω') in $\mathbb{R}^s$, where A', b', y are obtained from A, b, x, $\mathscr{B}$, x as described earlier; Ω' is the same as Ω for the regular equations, and with a relaxation parameter of 1.0 for the averaging equations. CARP-CG (in $\mathbb{R}^n$) is defined as the equivalent of CGMNC in $\mathbb{R}^s$ on the system $A'y = b'$ with Ω'. This means that, instead of running CGMNC in $\mathbb{R}^s$ with DKSWP(A', b', y, Ω') and DKSWP(A', 0, y, Ω'), we use DCSWP(A, b, $\mathscr{B}$, x, Ω) and DCSWP(A, 0, $\mathscr{B}$, x, Ω) in $\mathbb{R}^n$ to obtain the following.

Algorithm 3 (*CARP-CG*)**.**

set $x^0 \in \mathbb{R}^n$ to an arbitrary value
set $p^0 = r^0 = \text{DCSWP}(A, b, \mathscr{B}, x^0, \Omega) - x^0$
for $k = 0, 1, 2, \ldots$ until convergence **do**

$$
\begin{aligned}
q^k &= p^k - \text{DCSWP}(A, 0, \mathscr{B}, p^k, \Omega)\\
\alpha_k &= \|r^k\|^2 / \langle p^k, q^k \rangle\\
x^{k+1} &= x^k + \alpha_k p^k\\
r^{k+1} &= r^k - \alpha_k q^k\\
\beta_k &= \|r^{k+1}\|^2 / \|r^k\|^2\\
p^{k+1} &= r^{k+1} + \beta_k p^k
\end{aligned}
$$

end do
end algorithm

According to [15, Theorem 3], CARP-CG always converges, even when system (1) is inconsistent and/or non-square.

2.5. On the effectiveness of CARP-CG

The effectiveness of CARP-CG follows from two inherent features of KACZ, which will be explained below, and from the CG acceleration.

As noted previously, KACZ is SOR on the normal equations system (2), but this is done after the equations have been normalized. Therefore, we can assume that this normalization has already been done as a preliminary step; it is also more efficient to do so in practice. Note that KACZ is a geometric algorithm in the following sense: the iterates depend only on the hyperplanes defined by the equations and not on any particular algebraic representation of these hyperplanes. For this reason, we call the normalization GRS (geometric row scaling).

This inherent use of GRS by KACZ is the reason that CARP-CG can handle discontinuous coefficients effectively. GRS is a diagonal scaling of A, and the positive effect of such scalings, when dealing with discontinuous coefficients, has been known for a long time; see, for example, [22]. More recently, an extensive study in [23] has shown that GRS significantly improves the convergence properties of Bi-CGSTAB [24] and restarted GMRES [25] (both of them with and without the ILU(0) preconditioner) on nonsymmetric linear systems with discontinuous coefficients. However, for these algorithm/preconditioner combinations, these results held only for small- to moderate-sized convection terms; for large convection terms, [20] shows that CARP-CG is preferable.

An examination of GRS in [23,20] shows that it "pushes" heavy concentrations of eigenvalues around the origin away from the origin. It is known that such concentrations are detrimental to convergence. These results explain why CARP-CG

can handle the problem of discontinuous coefficients. Note that GRS has a certain "equalization" effect on the coefficients of different equations (but not on different-sized coefficients in the same equation).

Consider again the fact that KACZ is SOR on the normal equation system $AA^Ty = b$ ($x = A^Ty$). Using AA^T is generally considered detrimental because its condition number is the square of the condition number of A. However, a fact that seems to have been overlooked is that, if A is normalized first (as is inherent in KACZ), then the diagonal elements of AA^T are equal to 1, and all the off-diagonal elements are <1, provided no two rows of A are colinear [20, Theorem 3.1]. In other words, the two simple operations of first applying GRS, and then using AA^T, form a very simple means of dealing with systems with large off-diagonal elements, including cases of discontinuous coefficients.

Additionally, as the diagonal elements of A decrease, the sum of the off-diagonal elements of AA^T also decreases (and the diagonal remains 1). To see this, consider, a simplified case of a 5-point stencil matrix A obtained with second-order finite difference scheme on a regular grid. Assume that GRS has been applied to A, and denote by a and d the absolute values of the off-diagonal and the diagonal elements of A, respectively. In AA^T, the diagonal element is $4a^2 + d^2 = 1$. This gives us the relation $a = \sqrt{1 - d^2}/2$. We denote by S the sum of the absolute values of the off-diagonal elements of AA^T (in general position, i.e., without the extremal rows).

To evaluate S, we consider a 5-point cross of grid points C, which corresponds to some equation i in the linear system. Consider now all possible placements of another cross which shares at least one grid point with C. Each nonzero off-diagonal element in row i of AA^T is obtained by taking such a cross and adding the products of the corresponding values at the common grid points (i.e., computing the dot product of the rows). There are three types of such crosses, each type containing four crosses: type 1 consists of crosses which contain the midpoint of C, type 2 consists of crosses which share exactly two non-central grid points of C, and type 3 consists of crosses which share exactly one grid point with C. It is easy to see from this that $S = 12a^2 + 8|ad|$. Substituting $a = \sqrt{1 - d^2}/2$ into S, we get S as a function of d: $S(d) = 3\sqrt{1 - d^2} + 4d\sqrt{1 - d^2}$. Therefore, $\lim_{d\to 0} S'(d) = 4 > 0$, so $S(d)$ decreases as $d \to 0$.

3. The wave equation

Assume we are given a domain D in several dimensions. We use the following notation.

- c is the speed of sound (in m/s); c may vary in D.
- f is the frequency of the waves in Hertz.
- $\lambda = c/f$ is the wavelength (in meters).
- The (dimensional) wave number is defined as $k = 2\pi f/c$, from which we have $\lambda = 2\pi/k$.

The wave equation in D is $-(\Delta + k^2)u = g$, where k is the wave number, and g is a prescribed right-hand side. We use the following notation in the discretization of the problem.

- If D is a square measuring $L \times L$, then one can define the non-dimensional wave number as $\bar{k} = kL$. Generally, the term "large wave number" refers to large values of $\bar{k}$. We shall use k instead of $\bar{k}$ when the meaning is clear from the context.
- h (in meters) denotes the mesh size; we consider only uniform meshes.
- From the above, we obtain the number of grid points per wavelength as $N_g = c/(fh) = 2\pi/(kh)$.

In our experiments, we discretized the equations by using a second-order finite difference scheme. This yields a system of equations $Ax = b$, which, depending on the boundary conditions, may contain complex elements. Boundary conditions vary with the problems. In some cases, an impact on one side is modeled by a discontinuity on that side. In order to model unbounded domains, it is necessary to prevent reflecting waves from the boundaries. In our examples, we used the first-order absorbing boundary condition (the Sommerfeld radiation condition):

$$\partial u/\partial \vec{n} - iku = 0, \tag{9}$$

where $\vec{n}$ is the unit vector pointing outwards from the domain, and $i = \sqrt{-1}$.

4. Setup of the numerical experiments

Tests were run on a Supermicro cluster consisting of 12 nodes connected by an Infiniband network. Each node consists of two Intel Xeon E5520 quad CPUs running at 2.27 GHz, so the cluster can provide a total of 96 cores. The two CPUs share 8 GB of memory, and each has its own 8 MB cache. The cluster runs under Debian Linux, and message passing used the MPICH2 public domain MPI package. The rest of this section describes the implementation details and the approach we used to solve the complex-valued equations.

4.1. Implementation details

Our implementation of the data structures used by CARP-CG is similar to that of the AZTEC numerical software package [26]. 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

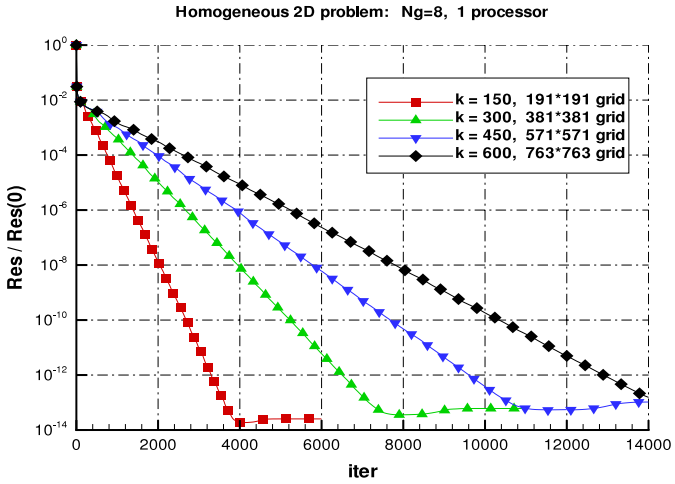

Fig. 1. Problem 1: convergence results for one processor, for different values of k.

submatrix and subvectors used by the processor. The local submatrix in each processor is stored in the sparse matrix format called DMSR (distributed modified sparse row) [26], which is a generalization of the MSR format [27, Section 3.4]. Additionally, every processor stores the "clones" of the relevant external grid points.

CARP-CG is now written entirely in C and compiled with the GNU C compiler, with optimization parameter -O. In our implementation, all matrix-vector and scalar products were coded inline; this is one contributing factor to the efficiency of our code. 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.

The initial estimate was taken in all cases as $x^0 = 0$, and iterations were stopped when the relative residual, $\|b - Ax\|/\|b - Ax^0\|$, fell below some given threshold value. Since this stopping criterion depends on the scaling of the equations, we first normalized the equations by dividing each equation by the L_2-norm of its coefficients; as mentioned earlier, this normalization is also part of KACZ.

4.2. Solving complex-valued equations

There are several known methods for solving complex equations; see [28]. According to this paper, and as confirmed by our experiments, the following approach is the most efficient. A complex system of equations $Ax = b$ can be broken down as follows. Let $A = B + iC$, $b = c + id$, and $x = y + iz$, where B, C, c, d, z, y are all real-valued. This yields the system $(B + iC)(y + iz) = c + id$, which is equivalent to the two real-valued systems $By - Cz = c$, $Cy + Bz = d$. These two $n \times 2n$ systems can be solved by "merging" them into one real-valued $2n \times 2n$ system by alternately taking one equation from the first system and one from the second system. This gives us the system

$$\begin{pmatrix} b_{11}, -c_{11}, \ldots, b_{1n}, -c_{1n} \\ c_{11}, \quad b_{11}, \ldots, c_{1n}, \quad b_{1n} \\ \vdots \\ b_{n1}, -c_{n1}, \ldots, b_{nn}, -c_{nn} \\ c_{n1}, \quad b_{n1}, \ldots, c_{nn}, \quad b_{nn} \end{pmatrix} \begin{pmatrix} y_1 \\ z_1 \\ \vdots \\ y_n \\ z_n \end{pmatrix} = \begin{pmatrix} c_1 \\ d_1 \\ \vdots \\ c_n \\ d_n \end{pmatrix}. \tag{10}$$

Note that when solving system (10) with CARP-CG, our first step is to normalize this system (and not the original $Ax = b$). In our experiments, we found that the difference between solving the original system and solving (10) with CARP-CG was quite significant: solving (10) was about twice as fast as solving the original system.

5. Runtime results and discussion

5.1. Problem 1

This problem is similar to [3, Section 6.2]. The equation $\Delta u + k^2 u = 0$ is defined on the unit square $[0, 1] \times [0, 1]$. Boundary conditions were chosen as follows: on the side $y = 0$, $u(0.5, 0) = 1$, and $u(x, 0) = 0$ for $x \neq 0$, i.e., there is a discontinuity at the midpoint. On the other three sides, the boundary conditions were chosen as first-order absorbing boundary condition (9). The number of grid points per wavelength was taken as $N_g = 6, 8, 10$, with $k = 75, 150, 300, 450$ and 600. Tests were run on one to 32 processors.

Fig. 1 shows the convergence plots of the relative residual, for a fixed number of grid points per wavelength, for one processor, and for k varying from 150 to 600. Naturally, when N_g is fixed, the grid sizes increase with k, so more iterations are needed for higher values of k.

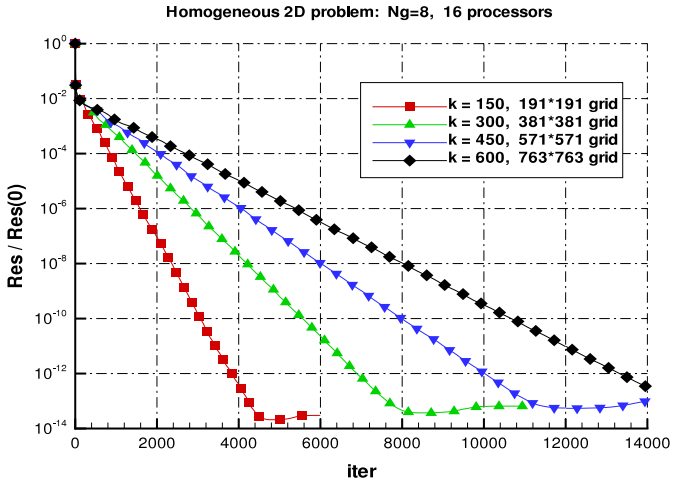

Fig. 2. Problem 1: convergence results for 16 processors, for different values of k.

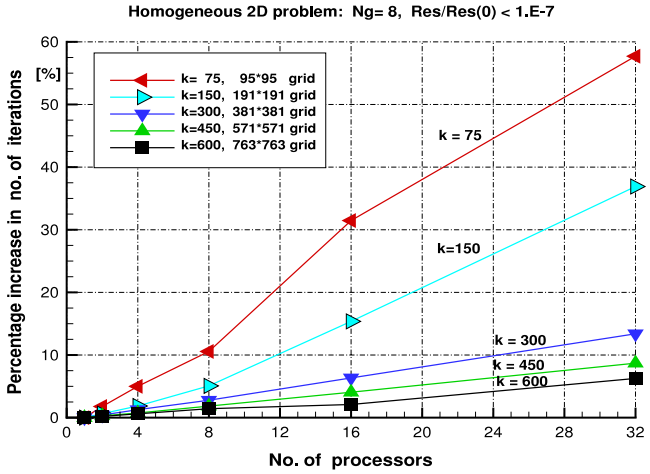

Fig. 3. Problem 1: percentage increase in the number of iterations required for convergence to 10^{-7}, as a function of the number of processors, for different values of k.

Fig. 2 shows the results for 16 processors. Note that the convergence plots for one and 16 processors are very similar to each other. Both figures show that, below 10^{-13}, the relative residual starts to deteriorate due to the accumulation of roundoff error.

Fig. 3 shows how the scalability of CARP-CG improves with higher values of k. This is shown by considering the percentage increase in the number of iterations required for convergence to a relative residual $<10^{-7}$ versus the number of processors, for different values of k. For $k = 75$, this percentage is 58%, while, for $k = 600$, it is approximately 6%.

Fig. 4 shows the number of iterations required for convergence (to a given relative residual) as a function of k, for a fixed value of N_g, on one and 32 processors. The figure shows that the number of iterations increases (almost) linearly with k. Note also that the distance between the two lines is almost constant as k increases; this is another indication that the percentage difference between them decreases with increasing k.

5.2. Problem 2

This problem is also based on a problem from Erlangga et al. [3, Section 6.3]. It is identical to Problem 1, but with a heterogeneous domain, resulting in so-called discontinuous coefficients. The unit square is divided into three parts, each with a different value of k: $k = 300$ for $0 < y \leq 1/3$, $k = 450$ for $1/3 < y \leq 2/3$, and $k = 600$ for $2/3 < y < 1$. The domain was discretized by a grid measuring 763×763, which corresponds to $N_g = 8$ for $k = 600$ and $N_g = 16$ for $k = 300$.

Fig. 5 shows the convergence plots on the heterogeneous domain, together with the convergence plots of two homogeneous cases with wave numbers equal to the two extreme values of the heterogeneous case, for one processor.

Fig. 6 shows the results for 16 processors. The most interesting point of these plots is that the convergence of the heterogeneous case is very similar to that of the homogeneous case with the lower-valued wave number ($k = 300$). Also, there is very little difference between the results for one and 16 processors.

Fig. 7 shows the percentage increase in the number of iterations required for convergence, as a function of the number of processors. Similarly to the previous results, the plot of the heterogeneous case is similar to that of the lower-valued homogeneous case.

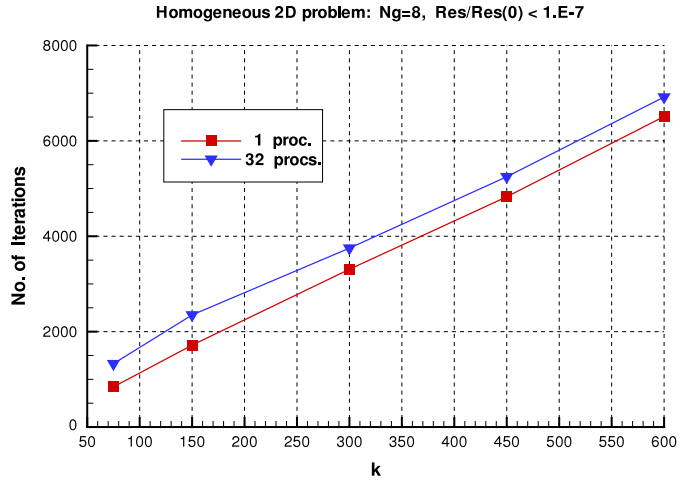

Fig. 4. Problem 1: number of iterations for convergence to 10^{-7}, as a function of k, for a constant number of grid points per wavelength ($N_g = 8$), on 1 and 32 processors.

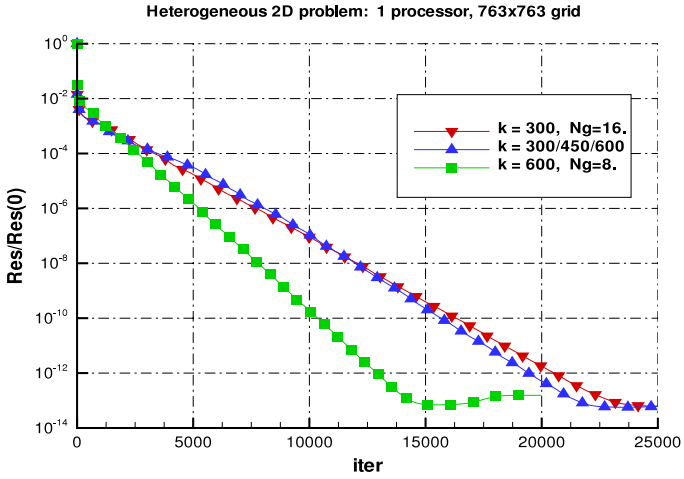

Fig. 5. Problem 2: convergence plots of the heterogeneous case and two related homogeneous cases, on one processor.

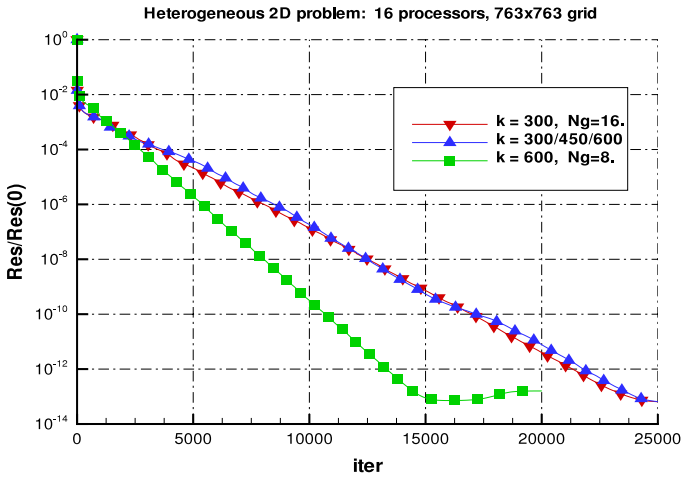

Fig. 6. Problem 2: convergence plots of the heterogeneous case and two related homogeneous cases, on 16 processors.

5.3. Problem 3

Problem 3 is based on the well-known Marmousi model, created in 1988 by the Institut Français du Petróle, and first studied in 1990 at the Workshop for Practical Aspects of Data Inversion [29]. This model was studied by many researchers; see, for example, Bollhöfer et al. [13].

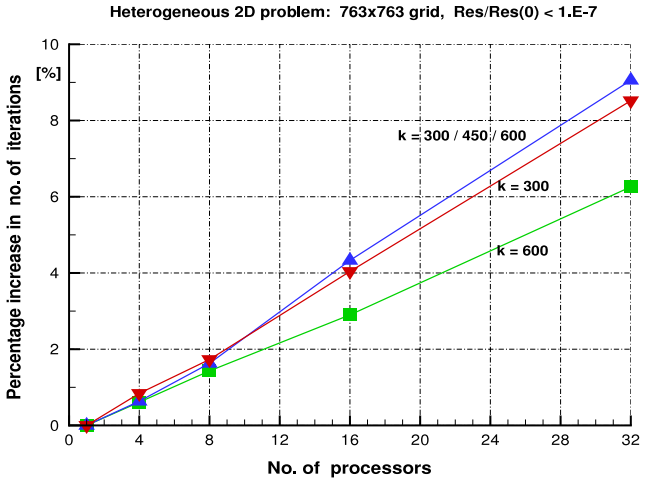

Fig. 7. Problem 2: percentage increase in the number of iterations as a function of the number of processors, for the heterogeneous case and two related homogeneous cases.

Table 1
Number of iterations and time (s) to reach a relative residual $<10^{-7}$ for the three Marmousi grid sizes, with $N_g \geq 7.5$, for 1–32 processors.

Grid, frequency and minimum N_g	Number of processors									
	1	2	4	8	12*	16*	20	24	28	32
751×201 $f = 25, N_g \geq 7.5$	4278 (97)	4327 (63)	4455 (35)	4548 (22)	4755 (18)	4936 (20)	5228 (19)	5629 (19)	5533 (18)	5803 (18)
1501×401 $f = 50, N_g \geq 7.5$	8540 (786)	8544 (490)	8608 (262)	8726 (144)	8895 (106)	9062 (107)	9250 (99)	9326 (85)	9613 (82)	9795 (77)
2001×534 $f = 65, N_g \geq 7.7$	11,301 (1868)	11,343 (1162)	11,405 (627)	11,552 (334)	11,737 (237)	11,924 (253)	12,028 (211)	12,176 (190)	12,424 (169)	12,635 (158)

* See text for explanation of the irregularity between 12 and 16 nodes.

The domain D corresponds to a 6000 m $\times$ 1600 m vertical slice of the Earth's subsurface, with the x-axis as horizontal and the y-axis pointing downwards. A point disturbance source is located at the center of the upper boundary. The wave equation is $\Delta u + k^2 u = g$, where $g(x, y) = \delta(x - (x_{\min} + x_{\max})/2)\delta(y)$. The domain is highly heterogeneous, with velocity c ranging from 1500 to 4450 m/s. This problem is discretized with a regular grid, with the Sommerfeld radiation condition (Eq. (9)) at the boundary. Three grid sizes, available with the original Marmousi data, were tested: 751×201, 1501×401, and 2001×534. Each grid was tested with a frequency f calculated to yield at least 7.5 grid points per wavelength.

Table 1 shows the number of iterations and time (in seconds) required for convergence to a relative residual $<10^{-7}$, for the three grid sizes, for one to 32 processors. The table also shows the frequency used with each grid and the minimal value of N_g, which is kept approximately constant. When running a program on more than 12 processors on the cluster, some of the nodes utilize both of their CPUs, and these compete with each other for communications. Thus, speedup results show a slight drop when going from 12 to 16 processors.

Fig. 8 shows the speedup plots for the three grid sizes, based on Table 1. The above-mentioned problem when going from 12 to 16 processors is clearly seen.

Table 2 shows the number of iterations and runtimes (in seconds) for the mid-sized grid of 1501×401, for $10 \leq f \leq 60$, for 1–32 processors. The most notable feature seen from this table is that, on a fixed grid, while CARP-CG performs poorly for $f = 10$, it improves significantly as the frequencies get higher. This improvement is twofold.

- For a fixed number of processors, the number of iterations and the timings improve significantly as f increases.
- The parallel scalability improves as f increases, i.e., the percentage increase in the number of iterations as the number of processors increases improves with higher values of f. This point is elucidated in Fig. 9.

5.4. Problem 4

Problem 4 is a three-dimensional (3D) heterogeneous problem analogous to Problem 2. It bears some similarity to a problem considered by Bollhöfer et al. in [13, Section 4.2]. The domain $D = (0, 1)^3$ is divided into three equal layers, each with a constant wave number. The wave number of the first layer is denoted by k_{ref}, the wave number of the second layer is $k = 1.2 \times k_{\text{ref}}$, and, for the third layer, $k = 1.5 \times k_{\text{ref}}$. The partitioning planes between the layers are equidistant and orthogonal to the z-axis. A point source is located at $(0.5, 0, 0.5)$, so the wave equation is $\Delta u + k^2 u = g$, where $g(x, y, z) = \delta(x - 0.5)\delta(y)\delta(z - 0.5)$. The Sommerfeld radiation condition (Eq. (9)) is imposed on the boundary.

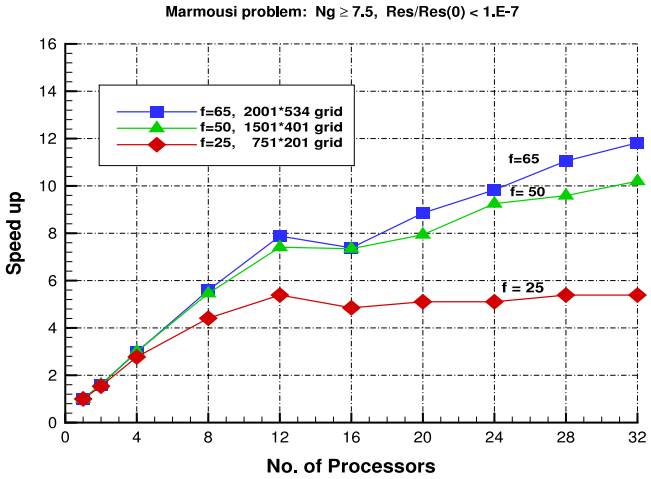

Fig. 8. Problem 3: speedup of the Marmousi problem for the three grid sizes, for $N_g \geq 7.5$. See text for explanation of the drop between 12 and 16 processors.

Table 2
Problem 3: number of iterations and time (s) to reach a relative residual $<10^{-7}$ for the mid-sized 1501 × 401 Marmousi grid, for different frequencies, on 1–32 processors.

Frequency and minimum N_g	Number of processors					
	1	2	4	8	16	32
$f = 10$ $N_g \geq 37.6$	27,106 (2492)	27,204 (1555)	27,878 (848)	29,304 (483)	32,018 (393)	38,618 (305)
$f = 20$ $N_g \geq 18.8$	15,498 (1425)	15,607 (893)	15,850 (482)	16,265 (267)	17,201 (202)	19,333 (151)
$f = 30$ $N_g \geq 12.5$	11,413 (1050)	11,431 (649)	11,551 (352)	11,847 (196)	12,384 (155)	13,712 (104)
$f = 40$ $N_g \geq 9.4$	9610 (884)	9655 (549)	9717 (296)	9927 (163)	10,294 (128)	11,293 (89)
$f = 50$ $N_g \geq 7.5$	8540 (786)	8544 (490)	8608 (262)	8726 (144)	9062 (107)	9795 (77)
$f = 60$ $N_g \geq 6.3$	7989 (753)	8012 (463)	8078 (244)	8236 (137)	8515 (105)	9072 (72)

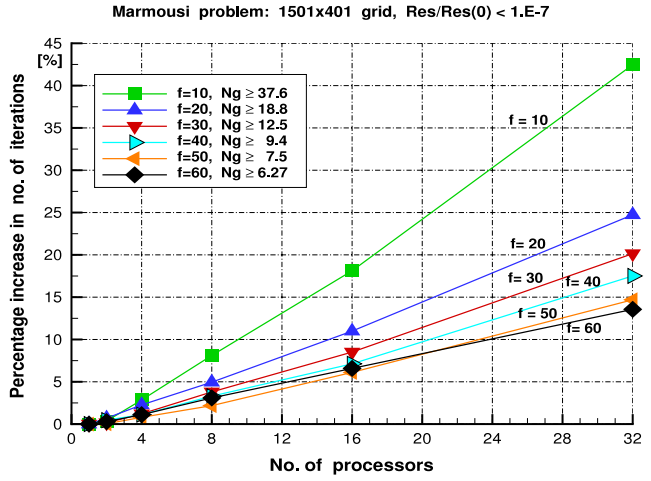

Fig. 9. Problem 3: percentage increase in the number of iterations as a function of the number of processors, for different frequencies.

Similarly to the problem in [13], this problem was tested for $k_{\text{ref}} = 60$, on a 95 × 95 × 95 grid. Thus, the three wave numbers of the subdomains are 60, 72, and 90, and the corresponding values of N_g are 9.95, 8.29, and 6.63. Figs. 10 and 11 show the convergence plots of the relative residual, for one processor and for 16 processors. Also shown for comparison are the convergence plots for two homogeneous domains with wave numbers equal to the extreme values of the heterogeneous

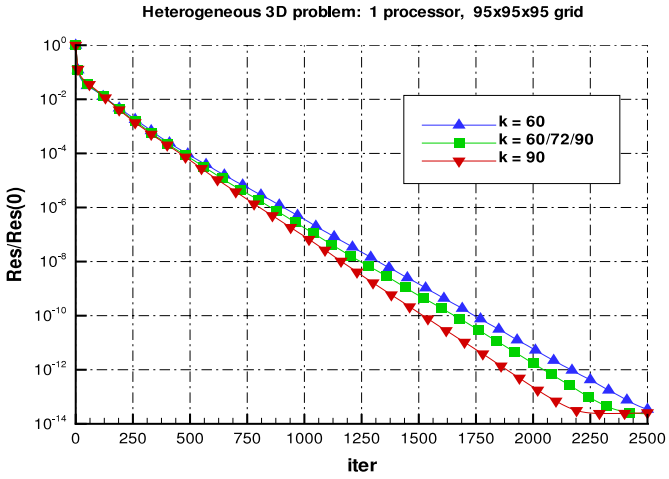

Fig. 10. Problem 4: convergence plot of the 3D heterogeneous problem, compared with two homogeneous cases, for one processor.

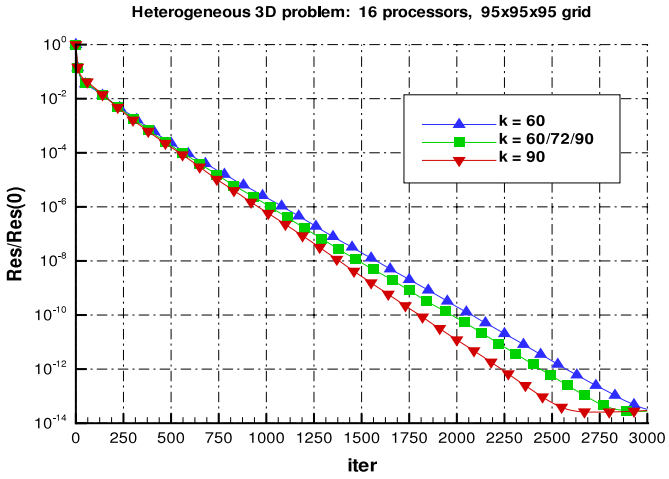

Fig. 11. Problem 4: convergence plot of the 3D heterogeneous problem, compared with two homogeneous cases, for 16 processors.

case. Note that, in both figures, the convergence plot of the heterogeneous case lies between the two other plots, and closer to the lower-valued case.

Fig. 12 shows the scalability of CARP-CG for the 3D heterogeneous problem and the two related homogeneous problems. The line plots show the percentage increase in the number of iterations as a function of the number of processors. Note the sudden jump in the percentage increase when going from one to two processors. After this jump, the percentage increase is quite regular.

In order to test the applicability of CARP-CG to 3D homogeneous and heterogeneous problems with even higher values of k, some experiments were run on one processor. The tested values of k were 60, 90, and 145. Table 3 shows the number of iterations and runtimes required to achieve various convergence goals, for five related homogeneous cases and one heterogeneous case. Columns 2, 3 and 6 show the results for $N_g \approx 6.5$. Columns 4–7 show the results on a fixed grid of 151^3. Two points can be noted.

- On the fixed grid of 151^3, and for each convergence goal, the efficiency of CARP-CG improves with increasing wave numbers.
- The number of iterations (and time) of the heterogeneous case is approximately the average of the homogeneous cases of $k = 60$ and $k = 90$ (with the same 151^3 grid).

6. Comparisons with the shifted Laplacian approach

In this section, we compare the results using CARP-CG on one processor (CGMN) with results in two recent papers using the shifted Laplacian preconditioner with an imaginary shift, solved by a multigrid.

Our first comparison is with Riyanti et al. [5]. The example is similar to Problem 4: a 3D cube of volume 304^3 m^3 is partitioned into three layers with a different value of k in each layer. The three values, k_1, k_2, k_3, are determined by fixing k_2, the value of the central layer, and taking $k_1 = 1.25k_2$ and $k_3 = 1.5k_2$. Sommerfeld boundary conditions, Eq. (9), are imposed

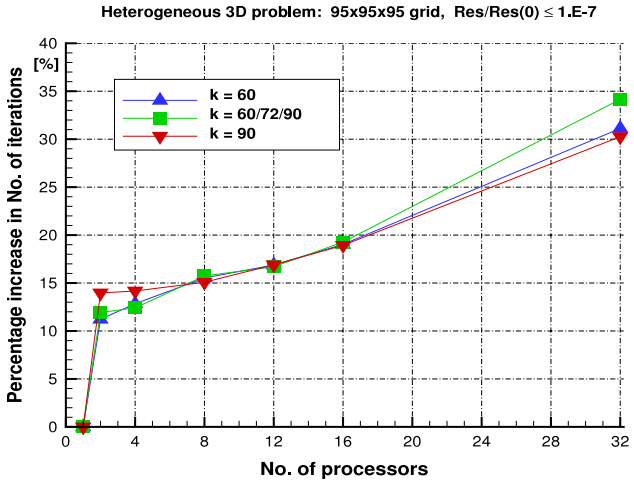

Fig. 12. Problem 4: percentage increase in the number of iterations as a function of the number of processors, for one heterogeneous 3D problem and two related homogeneous problems.

Table 3
Problem 4 (3D): number of iterations and runtimes (in seconds) on one processor to achieve four convergence goals, for $N_g \geq 6.5$, for five homogeneous cases and one heterogeneous case.

k: Grid: N_g:	60 63^3 6.5	90 95^3 6.6	60 151^3 15.7	90 151^3 10.5	145 151^3 6.5	60/90/145 151^3 6.5–15.7
10^{-4}	319 (16)	456 (77)	902 (606)	735 (493)	693 (465)	795 (533)
10^{-7}	666 (33)	988 (168)	2213 (1488)	1748 (1173)	1526 (1024)	2011 (1347)
10^{-10}	1021 (51)	1519 (258)	3546 (2384)	2810 (1886)	2395 (1607)	3222 (2159)
10^{-13}	1376 (69)	2067 (351)	4880 (3275)	3879 (2603)	3291 (2209)	4421 (2967)

Table 4
Comparison with results in Riyanti et al. [5, Table 3], showing number of iterations, runtimes (in seconds), and runtime ratios (corrected to account for the difference between processors).

Grid	16^3	32^3	48^3	64^3	80^3
$k_1\|k_2\|k_3$	15|10|12.5	30|20|25	45|30|37.5	60|40|50	75|50|62.5
[5, Table 3]	16 (6.11)	23 (23.3)	31 (65.1)	39 (136)	52 (320)
CARP-CG (CGMN)	149 **(0.16)**	284 **(1.95)**	401 **(8.92)**	530 **(27.4)**	653 **(64.7)**
CARP-CG/[5]*	5.2%	16.8%	27.4%	40%	40%

* Runtime ratios corrected for the difference between processors.

on all boundaries. We compare the results from using CARP-CG on one processor (CGMN) with the best runtime results of [5, Table 3]. Five grid sizes, from 16^3 to 80^3, are used. The grid sizes and values of k are chosen so as to maintain an approximately fixed value of $N_g \approx 10$. Table 4 shows the number of iterations and runtimes (in seconds) of these comparisons. The runtimes, in parentheses, are shown to three significant digits.

The tests of [5, Table 3] were performed on an SGI Altix (on a single processor), so, in order to reach a true evaluation, we also ran our tests on a single processor of a similar SGI machine (a 64-bit Intel Itanium 2 running at 1.5 GHz). The timings indicated that the Itanium was approximately half as fast as a single core of the Supermicro cluster. Table 4 also shows the ratio of the CARP-CG time to the timings of [5, Table 3], but corrected so as to account for the difference between the processors. We can see that CARP-CG takes approximately 40% of the time given in [5] for large wave numbers, and much less for small wave numbers.

Next, we compare a two-dimensional problem, called MP1, from Knibbe et al. [6, Section 4]. The domain D is the unit square, and the equation is $-\Delta u - k^2 u = \delta(x - 0.5)(y - 0.5)$, i.e., there is a disturbance at the center of the square. Values of k range from 40 to 640, and the grid sizes are chosen so as to maintain an approximately fixed value of $N_g \approx 10$. First-order absorbing boundary conditions are imposed on all sides. The purpose of [6] was to implement the shifted Laplacian

Table 5
Comparison with Knibbe et al. [6, Table 7, CPU results], showing number of iterations, runtimes (in seconds), and runtime ratios (corrected to account for the difference between processors).

Grid	64^2	128^2	256^2	512^2	1024^2
k	40	80	160	320	640
[6, Table 7]	12 (0.3)	21 (1.9)	40 (15.1)	77 (115)	151 (895)
CARP-CG (CGMN)	218 **(0.12)**	410 **(0.93)**	737 **(6.85)**	1368 **(52.5)**	2541 **(395)**
CARP-CG/[6]*	59%	72%	67%	67%	65%

* Runtime ratios corrected for the difference between processors.

preconditioner (with an imaginary shift and solved by an MG), using a GPU to accelerate some of the computations, and to compare the results with a pure CPU implementation. However, tests in [6] were ran only to a relative residual $<10^{-3}$ in order to allow Bi-CGSTAB to converge on the CPU and the GPU; this fact indicates a possible convergence problem. Tests were also made in [6] with the IDR(s) algorithm [30], but we consider only the Bi-CGSTAB timings since they were better than those with IDR(s).

Table 5 shows the number of iterations and runtimes (in seconds) of CARP-CG on a single processor (CGMN) and the CPU results of [6, Table 7]. The processor used in [6] was an AMD Phenom 9850 quad-core running at 2.5 GHz, with 8 GB memory. In order to reach a true evaluation, we compared the Passmark rankings [31] of the two processors. The Xeon E5520 (of our Supermicro cluster) is approximately 46.8% faster than the AMD Phenom 9850 (4357 CPU Mark for the Xeon and 2968 for the Phenom). The runtime ratios in Table 5 are corrected so as to account for these differences.

The conclusions from these comparisons are that, using one processor on the tested problems, and at all wave numbers, CARP-CG (CGMN) outperforms methods based on the shifted Laplacian preconditioner solved with an MG. Furthermore, on the tested 3D problem [5], CARP-CG performed much better on the *smaller* wave numbers.

7. Conclusions and further research

This paper demonstrated the usefulness of the block-parallel CARP-CG algorithm for solving two- and three-dimensional Helmholtz problems in both homogeneous and heterogeneous media. CARP-CG is highly scalable for large wave numbers, and, for a fixed number of grid points per wavelength, the number of iterations increases linearly with the wave number. Comparisons with methods using the shifted Laplacian preconditioner with an imaginary shift, and solved by a multigrid, indicate that CARP-CG performs better at all tested wave numbers, and, on a 3D problem, it performs much better at *smaller* wave numbers.

CARP-CG is a simple algorithm, suitable for structured and unstructured grids. The dependency of CARP-CG on the relaxation parameter is very mild; in fact, a single value (1.5) was used in all the experiments. It is a robust general-purpose method, particularly useful for problems in which the system matrix contains very large off-diagonal elements, and possibly also discontinuous coefficients.

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

- Application of CARP-CG to the Helmholtz equation using fourth-order and sixth-order difference schemes. Recent work in this direction appears in [32,33].
- Application of CARP-CG to other difficult problems, such as circuit design.
- Application of CARP-CG as a solver of intermediate linear systems in various nonlinear problems. For example, quasi-linear solution methods in nonlinear CFD problems, and eigenvalue problems.

Acknowledgments

The authors wish to thank Olaf Schenk for his useful comments and for the Marmousi data set. Thanks are also due to the anonymous reviewers for their helpful comments.

References

[1] I. Duff, S. Gratton, X. Pinel, X. Vasseur, Multigrid based preconditioners for the numerical solution of two-dimensional heterogeneous problems in geophysics, International Journal of Computer Mathematics 84 (8) (2007) 1167–1181.
[2] A. Bayliss, C.I. Goldstein, E. Turkel, An iterative method for the Helmholtz equation, Journal of Computational Physics 49 (1983) 443–457.
[3] Y.A. Erlangga, C. Vuik, C.W. Oosterlee, On a class of preconditioners for solving the Helmholtz equation, Applied Numerical Mathematics 50 (2004) 409–425.
[4] Y.A. Erlangga, C.W. Oosterlee, C. Vuik, A novel multigrid based preconditioner for heterogeneous Helmholtz problems, SIAM Journal on Scientific Computing 27 (4) (2006) 1471–1492.
[5] C.D. Riyanti, A. Kononov, Y.A. Erlangga, C. Vuik, C.W. Oosterlee, R.-E. Plessix, W.A. Mulder, A parallel multigrid-based preconditioner for the 3D heterogeneous high-frequency Helmholtz equation, Journal of Computational Physics 224 (2007) 431–448.

[6] H. Knibbe, C.W. Oosterlee, C. Vuik, GPU implementation of a Helmholtz Krylov solver preconditioned by a shifted Laplace multigrid method, Journal of Computational and Applied Mathematics 236 (2011) 281–293.
[7] Y.A. Erlangga, E. Turkel, Iterative schemes for high order compact discretizations to the exterior Helmholtz equation, ESAIM: Mathematical Modelling and Numerical Analysis 46 (3) (2012) 647–660.
[8] Y.A. Erlangga, Advances in iterative methods and preconditioners for the Helmholtz equation, Archives of Computational Methods in Engineering 15 (2008) 37–66.
[9] H.C. Elman, O.G. Ernst, D.P. O'Leary, A multigrid method enhanced by Krylov subspace iteration for discrete Helmholtz equations, SIAM Journal on Scientific Computing 23 (4) (2001) 1291–1315.
[10] K. Chen, P.J. Harris, Efficient preconditioners for iterative solution of the boundary element equations for the three-dimensional Helmholtz equation, Applied Numerical Mathematics 36 (2001) 475–489.
[11] R.-E. Plessix, W.A. Mulder, Separation-of-variables as a preconditioner for an iterative Helmholtz solver, Applied Numerical Mathematics 44 (2003) 385–400.
[12] S. Kim, C.-S. Shin, J.B. Keller, High-frequency asymptotics for the numerical solution of the Helmholtz equation, Applied Mathematics Letters 18 (2005) 797–804.
[13] M. Bollhöfer, M.J. Grote, O. Schenk, Algebraic multilevel preconditioner for the Helmholtz equation in heterogeneous media, SIAM Journal on Scientific Computing 31 (5) (2009) 3781–3805.
[14] D. Osei-Kuffuor, Y. Saad, Preconditioning Helmholtz linear systems, Applied Numerical Mathematics 60 (2010) 420–431.
[15] D. Gordon, R. Gordon, CARP-CG: a robust and efficient parallel solver for linear systems, applied to strongly convection-dominated PDEs, Parallel Computing 36 (9) (2010) 495–515.
[16] 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.
[17] S. Kaczmarz, Angenäherte Auflösung von Systemen linearer Gleichungen, Bulletin de l'Académie Polonaise des Sciences et Lettres A35 (1937) 355–357.
[18] Å. Björck, T. Elfving, Accelerated projection methods for computing pseudoinverse solutions of systems of linear equations, BIT 19 (1979) 145–163.
[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] P.P.B. Eggermont, G.T. Herman, A. Lent, Iterative algorithms for large partitioned linear systems, with applications to image reconstruction, Linear Algebra and its Applications 40 (1981) 37–67.
[22] O.B. Widlund, On the effects of scaling of the Peaceman–Rachford method, Mathematics of Computation 25 (113) (1971) 33–41.
[23] D. Gordon, R. Gordon, Row scaling as a preconditioner for some nonsymmetric linear systems with discontinuous coefficients, Journal of Computational and Applied Mathematics 234 (12) (2010) 3480–3495.
[24] 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 and Statistical Computing 13 (1992) 631–644.
[25] Y. Saad, M.H. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM Journal on Scientific and Statistical Computing 7 (1986) 856–869.
[26] R.S. Tuminaro, M.A. Heroux, S.A. Hutchinson, J.N. Shadid, AZTEC user's guide, Tech. Rep. SAND99-8801J, Sandia National Laboratories, Albuquerque, New Mexico, 1999.
[27] Y. Saad, Iterative Methods for Sparse Linear Systems, second ed., SIAM, Philadelphia, PA, 2003.
[28] D. Day, M.A. Heroux, Solving complex-valued linear systems via equivalent real formulations, SIAM Journal on Scientific Computing 23 (2) (2001) 480–498.
[29] The Marmousi Model, Workshop on Practical Aspects of Seismic Data Inversion, in: Proceedings of 52nd Annual Meeting of the European Association of Geophysicists & Engineers, 1990.
[30] P. Sonneveld, M.B. van Gijzen, IDR(s): a family of simple and fast algorithms for solving large nonsymmetric systems of linear equations, SIAM Journal on Scientific Computing 31 (2008) 1035–1062.
[31] Passmark CPU Benchmark, http://www.cpubenchmark.net/cpu_list.php.
[32] D. Gordon, R. Gordon, Parallel solution of high frequency Helmholtz equations using high order finite difference schemes, Applied Mathematics & Computation 218 (21) (2012) 10737–10754.
[33] E. Turkel, D. Gordon, R. Gordon, S. Tsynkov, Compact 2D and 3D sixth order schemes for the Helmholtz equation with variable wave number, Tech. Rep., Dept. of Computer Science, University of Haifa, Israel. http://cs.haifa.ac.il/~gordon/sixth_order.pdf, Aug. 2012.