Comp. Appl. Math.
DOI 10.1007/s40314-016-0416-7

A cooperative conjugate gradient method for linear systems permitting efficient multi-thread implementation

Amit Bhaya[1] · **Pierre-Alexandre Bliman**[2,3] · **Guilherme Niedu**[4] · **Fernando A. Pazos**[5]

Received: 16 December 2016 / Accepted: 30 December 2016
© SBMAC - Sociedade Brasileira de Matemática Aplicada e Computacional 2017

Abstract This paper revisits, in a multi-thread context, the so-called multi-parameter or block conjugate gradient (B-CG) methods, first proposed as sequential algorithms by O'Leary and Brezinski, for the solution of the linear system $\mathbf{Ax} = \mathbf{b}$, for an n-dimensional symmetric positive definite matrix $\mathbf{A}$. Instead of the scalar parameters of the classical CG algorithm, which minimizes a scalar functional at each iteration, multiple descent and conjugate directions are updated simultaneously. Implementation involves the use of multiple threads and the algorithm is referred to as cooperative CG (CCG) to emphasize that each thread now uses information that comes from the other threads. It is shown that for a sufficiently large matrix dimension n, the use of an optimal number of threads results in a worst case flop count of $O(n^{7/3})$ in exact arithmetic. Numerical experiments on a multi-core, multi-thread computer, for synthetic and real matrices, illustrate the theoretical results.

Communicated by Eduardo Souza de Cursi.

✉ Fernando A. Pazos
quini.coppe@gmail.com

Amit Bhaya
amit@nacad.ufrj.br

Pierre-Alexandre Bliman
pierre-alexandre.bliman@inria.fr

Guilherme Niedu
guiniedu@gmail.com

1 Department of Electrical Engineering, Federal University of Rio de Janeiro, Rio de Janeiro, RJ, Brazil

2 Sorbonne Universits, Inria, UPMC Univ Paris 06, Lab. J.L. Lions UMR CNRS 7598, Paris, France

3 Escola de Matemática Aplicada, Fundação Getulio Vargas, Rio de Janeiro, RJ, Brazil

4 Petrobras, S.A., Rio de Janeiro, Brazil

5 Department of Electronics and Telecommunication Engineering, State University of Rio de Janeiro, Rio de Janeiro, RJ, Brazil

Published online: 12 January 2017

Keywords Discrete linear systems · Iterative methods · Conjugate gradient algorithm · Cooperative algorithms

Mathematics Subject Classification 65Y05

1 Introduction

The appearance of multi-core processors has motivated much recent interest in multi-thread computation, in which each thread is assigned some part of a larger computation and executes concurrently with other threads, each on its own core. All threads, however, have relatively fast access to a common memory, which is the source and destination of all data manipulated by the thread.

With the availability of ever larger on-chip memory and multi-core processors that allow multi-thread programming, it is now possible to propose a new paradigm in which each thread, with access to a common memory, computes its own estimate of the solution to the whole problem (i.e., decomposition of the problem into subproblems is avoided) and the threads exchange information amongst themselves, this being the cooperative step. The design of a cooperative algorithm has the objective of ensuring that exchanged information is used by the threads in such a way as to reduce overall convergence time.

The idea of information exchange between two iterative processes was introduced into numerical linear algebra, in the context of linear systems, long before the advent of multi-core processors by Brezinski and Redivo-Zaglia (1994) under the name of *hybrid procedures*, defined as (we quote) "a combination of two arbitrary approximate solutions with coefficients summing up to one...(so that) the combination only depends on one parameter whose value is chosen to minimize the Euclidean norm of the residual vector obtained by the hybrid procedure... The two approximate solutions which are combined in a hybrid procedure are usually obtained by two iterative methods". The objective of minimizing the residue is to accelerate convergence of the overall hybrid procedure (also see Abkowicz and Brezinski 1996; Brezinski and Chehab 1998). This idea was generalized and discussed in the context of distributed asynchronous computation in Bhaya et al. (2010). It is also worthy of note that the paradigm of cooperation between threads, thought of as an independent agents, to achieve some common objective, is also becoming popular in many areas, such as control (Kumar et al. 2005; Nedic and Ozdaglar 2010; Murray 2007).

Several iterative methods to solve the linear algebraic equation

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1}$$

where $\mathbf{A} \in \mathbb{R}^{n\times n}$ is symmetric positive definite and n is large, are well known. Solving (1) is equivalent to finding the minimizer of the strictly convex scalar function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top\mathbf{A}\mathbf{x} - \mathbf{b}^\top\mathbf{x} \tag{2}$$

since the unique minimizer of f is $\mathbf{x}^* = \mathbf{A}^{-1}\mathbf{b}$.

Conjugate direction methods, based on minimization of (2), can be regarded as being intermediate between the method of steepest descent and Newton's method. They are motivated by the desire to accelerate the typically slow convergence associated with steepest descent while avoiding the information requirements associated with the evaluation and inversion of the Hessian (Luenberger and Ye 2008, chap. 9).

The conjugate gradient algorithm (CG) is the most popular conjugate direction method. It was developed by Hestenes and Stiefel (1952). The algorithm minimizes the scalar function $f(\mathbf{x})$ along conjugate directions searched at each iteration; the convergence of the sequence of points $\mathbf{x}_k$ to the solution point $\mathbf{x}^*$ is produced after at most n iterations in exact arithmetic. The residual vector is defined as

$$\mathbf{r}_k = \mathbf{A}\mathbf{x}_k - \mathbf{b} \tag{3}$$

and, given (2), clearly $\mathbf{r}_k = \nabla f(\mathbf{x}_k)$. In the CG algorithm, the residual vector $\mathbf{r}_k$ and a direction vector $\mathbf{d}_k$ are calculated at the kth iteration, for every k.

O'Leary (1980) developed a block-CG method (B-CG) in which the conjugate directions and the residues are taken as columns of $n \times p$ matrices. The B-CG algorithm was designed to handle multiple right-hand sides which form a matrix $\mathbf{B} \in \mathbb{R}^{n\times p}$, but it is also capable of accelerating the convergence of linear systems with a single right-hand side, for example for solving systems in which several eigenvalues are widely separated from the others. Several properties observed by the vectors in the CG algorithm continue to be valid for the matrices used in the B-CG algorithm, e.g., the conjugacy property between the matrix directions. In addition, in exact arithmetic, the convergence of the B-CG algorithm to the solution matrix $\mathbf{X}^*$ occurs after at most $\lceil \frac{n}{p} \rceil$ iterations, which may involve less work than applying the CG algorithm p times. Gutknecht (2007) also analyzes block methods based on Krylov subspaces with multiple right-hand sides.

Brezinski (1999, sec. 4), and Bantegnies and Brezinski (2001) developed a block-CG algorithm called "multi-parameter CG" (MPCG), which is essentially the B-CG with a single right-hand side $\mathbf{b}$ and a single initial point $\mathbf{x}_0$. Brezinski (1999) and Bantegnies and Brezinski (2001) build on the pioneering work of O'Leary (1980), and provide some additional properties, particularly about its convergence. It should be noted that the B-CG algorithm as well as the MPCG algorithm were proposed in the context of a single processor, so issues of multi-processor implementation, speed up, and flop counts were not considered in O'Leary (1980), Brezinski (1999) and Bantegnies and Brezinski (2001).

This paper revisits Brezinski's MPCG algorithm from a multi-thread perspective, calling it, to emphasize the new context, the cooperative conjugate gradient (CCG) algorithm. The cooperation between threads resides in the fact that each thread now uses information that comes from the other threads, and, in addition, the descent and conjugate directions are updated simultaneously. The multi-thread implementation of the CCG algorithm aims to accelerate the time to convergence with respect to the B-CG and the MPCG algorithms. Preliminary versions of this paper are Bhaya et al. (2012).

In Gu et al. (2004a, b), the authors present a multi-thread CG method named "multiple search direction conjugate gradient method" (MSD-CG), and its preconditioned version (PMSD-CG). The method is midway between the CG method and the Block Jacobi method. It is based on a notion of subdomains or partitioning of the unknowns. In each iteration, there is one search direction per subdomain that is zero in the vector elements that are associated with other subdomains (Gu et al. 2004a, p. 1134). The algorithm can be executed in parallel by a multi-core processor. The problem is divided into smaller blocks, thus dividing the direction vectors and the residual vectors into smaller vectors to be calculated by each processor separately.

This paper is organized as follows. In Sect. 2, the conjugate gradient algorithm as well as some basic properties of the conjugate directions are presented. In Sect. 3, the cooperative conjugate gradient algorithm in a multi-thread context is presented. Their basic properties and the convergence rate are studied. In Sect. 4, the computational complexity of the CCG algorithm as well as the classic CG, the MPCG, and the MSD-CG are investigated. In Sect.

5, experimental results are presented. In Sect. 6, some general conclusions are mentioned. Finally, an appendix presents the proofs of theorems and lemmas.

2 Preliminaries on the classical CG algorithm

This section recalls basic results on the classical CG algorithm to motivate the presentation of the corresponding results for the cooperative CG algorithm. The reader is referred to Luenberger and Ye (2008) and Güler (2010) for all proofs and further details on the CG algorithm.

Definition 1 Given a symmetric positive definite matrix $\mathbf{A}$, two nonzero vectors $\mathbf{d}_1$ and $\mathbf{d}_2$ are said to be $\mathbf{A}$-orthogonal, or $\mathbf{A}$-conjugate, if $\mathbf{d}_1^\top \mathbf{A}\mathbf{d}_2 = 0$.

Lemma 1 *If a set of nonzero vectors* $\{\mathbf{d}_0, \ldots, \mathbf{d}_k\}$ *are* $\mathbf{A}$*-conjugate (with respect to a positive definite matrix* $\mathbf{A}$*), then these vectors are linearly independent. The solution* $\mathbf{x}^* \in \mathbb{R}^n$ *of the system* $\mathbf{A}\mathbf{x} = \mathbf{b}$ *can be expressed as a linear combination of* n $\mathbf{A}$*-conjugate vectors* $\{\mathbf{d}_0, \ldots, \mathbf{d}_{n-1}\}$.

$$\mathbf{x}^* = \alpha_0 \mathbf{d}_0 + \cdots + \alpha_{n-1}\mathbf{d}_{n-1}$$

where $\alpha_i = \frac{\mathbf{d}_i^\top \mathbf{b}}{\mathbf{d}_i^\top \mathbf{A}\mathbf{d}_i}$ *for all* $i \in \{0, \ldots, n-1\}$.

Theorem 1 *Let* $\{\mathbf{d}_0, \ldots, \mathbf{d}_{n-1}\}$ *be a set of* n $\mathbf{A}$*-conjugate nonzero vectors. For any* $\mathbf{x}_0 \in \mathbb{R}^n$*, the sequence generated according to*

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k \tag{4}$$

with

$$\alpha_k = -\frac{\mathbf{r}_k^\top \mathbf{d}_k}{\mathbf{d}_k^\top \mathbf{A}\mathbf{d}_k} \tag{5}$$

where $\mathbf{r}_k = \nabla f(\mathbf{x}_k) = \mathbf{A}\mathbf{x}_k - \mathbf{b}$*, converges to the unique solution of* $\mathbf{A}\mathbf{x} = \mathbf{b}$, $\mathbf{x}^*$ *after* n *steps, that is* $\mathbf{x}_n = \mathbf{x}^*$.

It is notable that the choice (5) ensures the convergence of the sequence (4) in at most n steps in exact arithmetic, which is known as the *finite termination property*. However, one of the most interesting and still partially understood property of the CG algorithm is that, even when implemented in finite precision arithmetic, approximate convergence to standard tolerances occurs much faster than n iterations (Meurant and Strakoš 2006). Nevertheless, we will use this "worst case" estimate of time to convergence to generate flop count estimates of the CCG algorithm in Sect. 4. Another fundamental result, the CCG analog of which is presented as Theorem 4 below, is called the expanding subspace theorem (Luenberger and Ye 2008).

Theorem 2 *Let* $\mathcal{B}_k$ *the space spanned by the set of nonzero conjugate vectors* $\{\mathbf{d}_0, \ldots, \mathbf{d}_{k-1}\}$*. The point* $\mathbf{x}_k$ *calculated by the sequence (4) with the step sizes (5) is the global minimizer of* $f(\mathbf{x})$ *on the subspace* $\mathbf{x}_0 + \mathcal{B}_k$*. Moreover, the residual vector* $\mathbf{r}_k = \nabla f(\mathbf{x}_k) = \mathbf{A}\mathbf{x}_k - \mathbf{b}$ *is orthogonal to* $\mathcal{B}_k$.

2.1 The conjugate gradient algorithm

The conjugate gradient method, developed by Hestenes and Stiefel (1952), is the particular method of conjugate directions obtained when constructing the conjugate directions by Gram–Schmidt orthogonalization, achieved at step $k+1$ on the set of the gradients $\{\mathbf{r}_0, \ldots, \mathbf{r}_k\}$. A key point here is that this construction can be carried out iteratively. The conjugate gradient algorithm is based on the minimization at each iteration of the scalar function $f(\mathbf{x})$ on conjugate directions which form a basis of the Krylov subspace $\mathcal{K}_k(\mathbf{A}, \mathbf{r}_0) := \text{span}\{\mathbf{r}_0,\ \mathbf{A}\mathbf{r}_0,\ \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{k-1}\mathbf{r}_0\}$.

The algorithm is described as follows. Starting from any $\mathbf{x}_0 \in \mathbb{R}^n$, and choosing $\mathbf{d}_0 = \mathbf{r}_0 = \mathbf{A}\mathbf{x}_0 - \mathbf{b}$, at each iteration, calculate:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k \tag{6}$$

$$\alpha_k = -\frac{\mathbf{r}_k^\top \mathbf{d}_k}{\mathbf{d}_k^\top \mathbf{A}\mathbf{d}_k} \tag{7}$$

$$\mathbf{d}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{d}_k \tag{8}$$

$$\beta_k = -\frac{\mathbf{r}_{k+1}^\top \mathbf{A}\mathbf{d}_k}{\mathbf{d}_k^\top \mathbf{A}\mathbf{d}_k}. \tag{9}$$

To qualify as a conjugate gradient algorithm, the directions $\mathbf{d}_k$ generated at each step should be $\mathbf{A}$-conjugate, which is confirmed in the following theorem.

Theorem 3 (Conjugate gradient theorem) *The conjugate gradient algorithm* (6)–(9) *is a conjugate direction method. If it does not terminate at the step k, then:*

1. *span*$\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \ldots; \mathbf{A}^k\mathbf{r}_0\}$ = *span*$\{\mathbf{r}_0; \ldots; \mathbf{r}_k\}$ = *span*$\{\mathbf{d}_0, \ldots, \mathbf{d}_k\}$. *These subspaces have dimension* $k+1$.
2. $\mathbf{d}_k^\top \mathbf{A}\mathbf{d}_i = 0, \quad \forall i < k.$
3. $\mathbf{r}_k^\top \mathbf{r}_i = 0, \quad \forall i < k.$
4. *the point* $\mathbf{x}_{k+1}$ *is the minimizer of* $f(\mathbf{x})$ *on the affine subspace* $\mathbf{x}_0 +$ *span*$\{\mathbf{d}_0; \ldots; \mathbf{d}_k\}$
5. $\alpha_k = \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{d}_k^\top \mathbf{A}\mathbf{d}_k}.$
6. $\beta_k = \frac{\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{r}_k^\top \mathbf{r}_k}.$

For a proof of this theorem as well as further details on the contents of this section, see Luenberger and Ye (2008, chap. 9) and Güler (2010, chap. 14).

When the residue vector is zero, the optimum has been attained, showing that CG terminates in finite time, in exact arithmetic. Some other results about the convergence of the algorithm (6)–(9) can be found in Bouyouli et al. (2008). Interesting properties, both as an algorithm in exact arithmetic and as one in finite precision arithmetic can be found in Meurant and Strakoš (2006) and Greenbaum (1997).

3 The cooperative conjugate gradient method

Note that the conjugate gradient method is not parallelizable, because calculation of the new direction $\mathbf{d}_{k+1}$ requires the new residue $\mathbf{r}_{k+1}$ to have been calculated first.

However, if we suppose that p initial conditions are used, then we may ask if it is possible to initiate p CG-like computations in parallel and, in addition, share information amongst the

p processors carrying out these computations in such a way that there is an overall reduction in time of convergence? We shall refer to the p processors as *threads*, in order that the cooperative computation paradigm that we introduce below have a natural interpretation as a multi-thread cooperative algorithm.

The extension of the method (6)–(9) using p threads is defined using the following matrices:

1. $\mathbf{X} := [\mathbf{x}_1\ \mathbf{x}_2 \cdots \mathbf{x}_p] \in \mathbb{R}^{n\times p}$ is the matrix of solution estimates, in which the ith column is assigned to the ith thread.
2. $\mathbf{R} := [\mathbf{r}_1\ \mathbf{r}_2 \cdots \mathbf{r}_p] \in \mathbb{R}^{n\times p}$ is the matrix of the corresponding residues, such that $\mathbf{r}_i = \mathbf{A}\mathbf{x}_i - \mathbf{b},\ \forall i \in \{1, \ldots, p\}$.
3. $\mathbf{D} := [\mathbf{d}_1\ \mathbf{d}_2 \cdots \mathbf{d}_p] \in \mathbb{R}^{n\times p}$ is the matrix of the corresponding descent directions.

From an initial matrix $\mathbf{X}_0$, with residue $\mathbf{R}_0 = \mathbf{A}\mathbf{X}_0 - \mathbf{b}\mathbf{1}_p^\top$, and initial directions chosen as $\mathbf{D}_0 = \mathbf{R}_0$, at each step calculate:

$$\mathbf{X}_{k+1} = \mathbf{X}_k + \mathbf{D}_k \boldsymbol{\alpha}_k^\top \tag{10}$$

$$\mathbf{D}_{k+1} = \mathbf{R}_{k+1} + \mathbf{D}_k \boldsymbol{\beta}_k^\top \tag{11}$$

where,

$$\boldsymbol{\alpha}_k = -\mathbf{R}_k^\top \mathbf{D}_k (\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k)^{-1} \in \mathbb{R}^{p\times p} \tag{12}$$

$$\boldsymbol{\beta}_k = -\mathbf{R}_{k+1}^\top \mathbf{A}\mathbf{D}_k (\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k)^{-1} \in \mathbb{R}^{p\times p} \tag{13}$$

until a stopping criterion (for example $\|\mathbf{r}_i\|$ smaller than a given tolerance for some $i \in \{1, \ldots, p\}$) is met. In the following, we shall assume that $\mathbf{D}_k$ is a full column rank matrix for all iterations k. Note that if $p = 1$, the method (10)–(13) coincides with (6)–(9).

Remark 1 (a) O'Leary (1980) considers a block-CG solver which treats multiple right-hand sides at once ($\mathbf{B} \neq \mathbf{b}\mathbf{1}_p^\top$). Brezinski (1999, sec. 04) and Bantegnies and Brezinski (2001) considers only one thread $\mathbf{x}$ and the initial matrix $\mathbf{R}_0 = \mathbf{D}_0$ is a particular partition of the initial residue, such that $\mathbf{R}_0\mathbf{1}_p = \mathbf{r}_0 = \mathbf{A}\mathbf{x}_0 - \mathbf{b}$.

(b) In Sect. 3.2, the case where p does not necessarily divide n and the case where rank degeneracy may occur are both analyzed: neither case is considered in Brezinski (1999) and Bantegnies and Brezinski (2001).

(c) A preconditioned version of a multi-parameter CG algorithm (MPCG) was presented in Bridson and Greif (2006) which proposes a matrix version of the Gram–Schmidt process to find the conjugate direction matrices. However, this procedure is very expensive in computational terms. A preconditioned version of the CCG algorithm is not studied here, and will be the object of future research. It is expected that the advantages obtained by preconditioning the classical CG algorithm will also hold for the CCG algorithm.

3.1 Properties of the cooperative conjugate gradient method

All the relevant results on the CCG algorithm are collected in this subsection and the next subsection, while the proofs of the key properties are in an appendix, for this paper to be complete and self-contained. Lemmas 6, 7 and Theorems 5, 6 are new, to the best of our knowledge.

To present CCG properties that are analogous to those presented in Sect. 2.1, some notation is introduced. For any set of vectors $\mathbf{r}_i \in \mathbb{R}^n$, $i \in \{0, \ldots, k\}$, we denote, respectively, $\{\mathbf{r}_i\}_0^k$ and $[\mathbf{r}_i]_0^k$ the set of these vectors and the matrix obtained by their concatenation:

$[\mathbf{r}_i]_0^k = [\mathbf{r}_0\mathbf{r}_1 \cdots \mathbf{r}_k] \in \mathbb{R}^{n\times(k+1)}$. The notation span $[\mathbf{r}_i]_0^k$ will denote the subspace of linear combination of the columns of the matrix $[\mathbf{r}_i]_0^k$. When $\mathbb{R}^n$ is the ambient vector space, we have

$$\text{span } [\mathbf{r}_i]_0^k = \left\{\mathbf{v} \in \mathbb{R}^n \mid \exists \boldsymbol{\gamma} \in \mathbb{R}^{k+1},\ \mathbf{v} = \sum_{i=0}^{k} \gamma_i \mathbf{r}_i = [\mathbf{r}_i]_0^k \boldsymbol{\gamma}\right\}.$$

Similarly, for matrices $\mathbf{R}_i \in \mathbb{R}^{n\times p},\ i \in \{0,\ldots,k\}$, we introduce the notations $\{\mathbf{R}_i\}_0^k$ and $[\mathbf{R}_i]_0^k$, which denote, respectively, the set of these matrices and the matrix obtained as concatenation of the matrices $\mathbf{R}_0, \mathbf{R}_1, \ldots, \mathbf{R}_k$, that is $[\mathbf{R}_i]_0^k = [\mathbf{R}_0\, \mathbf{R}_1\, \cdots\, \mathbf{R}_k] \in \mathbb{R}^{n\times p(k+1)}$. In addition, we write span $[\mathbf{R}_i]_0^k$ for the subspace obtained by all possible linear combinations of the columns of $[\mathbf{R}_i]_0^k$:

$$\text{span } [\mathbf{R}_i]_0^k = \left\{\mathbf{v} \in \mathbb{R}^n \mid \exists \boldsymbol{\gamma} \in \mathbb{R}^{(k+1)p},\ \mathbf{v} = [\mathbf{R}_i]_0^k \boldsymbol{\gamma}\right\}.$$

Definition 2 Two matrices $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are called orthogonal if $\mathbf{Q}_1^\top \mathbf{Q}_2 = 0$, and they are called $\mathbf{A}$-conjugate, or simply conjugate with respect to a matrix $\mathbf{A}$ if $\mathbf{Q}_1^\top \mathbf{A}\mathbf{Q}_2 = 0$.

To prove the properties of the algorithm (10)–(13), we first substitute the equation (11) with step size (13) by the direction matrices generated by the Gram–Schmidt process, defined as follows.

The Gram–Schmidt process generates conjugate directions sequentially (Güler 2010, p. 389). It can be extended to work with matrices, where every column of a matrix generated by the process in a step is conjugate with respect to every column of the matrices generated in the former steps, in the following way:

$$\mathbf{D}_{k+1} = \mathbf{R}_{k+1} - \sum_{j=0}^{k} \mathbf{D}_j (\mathbf{D}_j^\top \mathbf{A}\mathbf{D}_j)^{-1} \mathbf{D}_j^\top \mathbf{A}\mathbf{R}_{k+1}, \quad \mathbf{D}_0 = \mathbf{R}_0. \tag{14}$$

To iterate using (14), it is assumed that all the matrices generated at each iteration have full column rank (so $\mathbf{D}_j^\top \mathbf{A}\mathbf{D}_j$ is nonsingular). It is easy to see that (14) generates matrices, such that $\mathbf{D}_j^\top \mathbf{A}\mathbf{D}_i = 0$ for all $i,\ j \in \{0,\ldots,k+1\}, i \neq j$. As in the classical case, iteration (14) is expensive in computational terms.

Theorem 4 *Let the direction matrices $\mathbf{D}_0,\ldots,\mathbf{D}_k$ be conjugate and of full column rank. The columns of $\mathbf{X}_k$ calculated as in (10), using the step size (12), minimize $f(\mathbf{x}_{i_k})$, $\forall i \in \{1,\ldots,p\}$ on the affine set $\mathbf{x}_{i_0} +$ span $[\mathbf{D}_j]_0^{k-1}$. Moreover, the columns of $\mathbf{R}_k$ are orthogonal to span $[\mathbf{D}_j]_0^{k-1}$, which means that $\mathbf{R}_k^\top \mathbf{D}_j = 0,\ \forall j < k$.*

The next lemma is easy to prove, by induction using (10)–(13), and the fact that $\mathbf{R}_{k+1} = \mathbf{A}\mathbf{X}_{k+1} - \mathbf{b}\mathbf{1}_p^\top = \mathbf{R}_k + \mathbf{A}\mathbf{D}_k\boldsymbol{\alpha}_k^\top$.

Lemma 2 *Suppose that $\mathbf{D}_0 = \mathbf{R}_0$, then*

$$\text{span } [\mathbf{R}_0\, \mathbf{R}_1 \cdots \mathbf{R}_k] = \text{span } [\mathbf{R}_0\, \mathbf{A}\mathbf{R}_0 \cdots \mathbf{A}^k\mathbf{R}_0] = \text{span } [\mathbf{D}_0\, \mathbf{D}_1 \cdots \mathbf{D}_k]. \tag{15}$$

Gutknecht (2007, sec. 8) defines block-Krylov subspace generated by matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{R}_0 \in \mathbb{R}^{n\times p}$ as

$$\mathcal{K}_k^\square(\mathbf{A}, \mathbf{R}_0) := \left\{\mathbf{X} \in \mathbb{R}^{n\times p} | \exists \gamma_0, \ldots, \gamma_k \in \mathbb{R}^{p\times p},\ \mathbf{X} = \sum_{i=0}^{k-1} \mathbf{A}^i \mathbf{R}_0 \gamma_i\right\}$$

and a block-Krylov subspace method as an iterative method which generates matrices belonging to a block-Krylov subspace $\mathbf{X}_k \in \mathbf{X}_0 + \mathcal{K}_k^{\square}(\mathbf{A}, \mathbf{R}_0)$, at each iteration.

According to these definitions, the method (10) with step size (12) is a block-Krylov subspace method. Note, however, that the spaces defined in (15) are not block-Krylov subspaces.

Lemma 3 *The matrices generated by* (14) *are the same as those generated by* (11) *with step size* (13).

The following useful lemmas are proved in O'Leary (1980), Brezinski (1999) and Bantegnies and Brezinski (2001).

Lemma 4 (Orthogonality properties)

$$\mathbf{R}_k^\top \mathbf{A}\mathbf{D}_k = \mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k \tag{16}$$

$$\mathbf{R}_k^\top \mathbf{R}_k = \mathbf{R}_k^\top \mathbf{D}_k. \tag{17}$$

Lemma 5 (Formulas for matrix step sizes)

$$\boldsymbol{\alpha}_k^\top = -(\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k)^{-1}\mathbf{R}_k^\top \mathbf{R}_k \tag{18}$$

$$\boldsymbol{\beta}_k^\top = -(\mathbf{R}_k^\top \mathbf{R}_k)^{-1}\mathbf{R}_{k+1}^\top \mathbf{R}_{k+1}. \tag{19}$$

Theorem 4, and Lemmas 2 and 3 indicate that, as long as the residue matrix $\mathbf{R}_k$ is full rank, the algorithm CCG behaves essentially as does CG, providing p different estimates at iteration k, each of them being optimal in an affine set constructed from one of the p initial conditions and the common vector space obtained from the columns of the direction matrices $\mathbf{D}_i$, $i \in \{0, \ldots, k-1\}$. This vector space, span $[\mathbf{D}_i]_0^k$, has dimension $(k+1)p$: each iteration involves the cancellation of p directions. Notice that different columns of the matrices $\mathbf{D}_k$ are not necessarily $\mathbf{A}$-orthogonal (in other words, $\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k$ is not necessarily diagonal), but, when $\mathbf{R}_k$ is full rank, they constitute a set of p independent vectors. The statements as well as the proofs of these theorems and lemmas (all in the appendix) have been inspired by the corresponding ones for the conventional CG algorithm given in Luenberger and Ye (2008, p. 270) and Güler (2010, pp. 390–391).

3.2 Convergence of the cooperative conjugate gradient method

To prove the convergence of the algorithm (10)–(13), first consider the case in which, if rank $\mathbf{D}_k = p$, then rank $\mathbf{D}_{k+1} = p$, at least until $\mathbf{D}_{k+1} = \mathbf{R}_{k+1} = \mathbf{0}$, in which case, the algorithm, in exact arithmetic, terminates.

Lemma 6 *If rank $\mathbf{D}_0 = p$ and the algorithm* (10)–(13) *does not terminate at the iteration k, which implies that $\mathbf{R}_k$ and $\mathbf{D}_k$ are different from zero, then rank $[\mathbf{D}_i]_0^{k-1} = pk \le n$, which means that, before convergence, all the column vectors are linearly independent.*

Remark 2 If rank $[\mathbf{D}_i]_0^k = n < p(k+1)$, then $\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_\ell \neq 0,\ \forall \ell < k$, which implies that the conjugacy property is no longer satisfied. In fact, only the $n - pk$ first columns of the matrix $\mathbf{D}_k$ continue to be conjugate to the former matrices, that is $\mathbf{D}_\ell^\top \mathbf{A}[\mathbf{d}_{i_k}]_1^{n-pk} = 0,\ \forall \ell < k$.

The same happens with the matrix $\mathbf{R}_k$, where $\mathbf{R}_k^\top \mathbf{D}_\ell \neq 0,\ \forall \ell < k$, only the first $n - pk$ columns continue to be orthogonal to span $[\mathbf{D}_i]_0^{k-1}$, that is $\mathbf{D}_\ell^\top [\mathbf{r}_{i_k}]_1^{n-pk} = 0,\ \forall \ell < k$.

The following theorem affirms that the iteration that follows the satisfaction of this condition (i.e., iteration $k+1$, such that $p(k+1) > n$), convergence to the solution occurs. We first consider the simpler case where p divides n.

Theorem 5 *If rank* $\mathbf{R}_0 = p$*, then all of the threads in the cooperative conjugate gradient method* (10)–(13) *converge to the solution* $\mathbf{x}^*$ *in at most* $k^* = \frac{n}{p}$ *iterations, which means that* $\mathbf{R}_{k^*} = \mathbf{D}_{k^*} = 0$ *and* $\mathbf{X}_{k^*} = \mathbf{x}^*\mathbf{1}_p^\top$.

Note that Lemma 6 indicates that if each matrix $\mathbf{D}_i$ generated at each iteration has a rank p, then all the columns of $[\mathbf{D}_0 \cdots \mathbf{D}_k]$ are linearly independent. Unfortunately, even if all columns of the matrices $\mathbf{R}_{k+1}$ and $\mathbf{D}_k$ are linearly independent, this does not guarantee that the columns of $\mathbf{D}_{k+1}$, calculated by (11), also have linearly independent columns.

When the columns of $\mathbf{D}_k$ are linearly dependent, it is enough to eliminate columns (threads) in such a way that $\mathbf{D}_k$ continues to have full column rank, choosing any full-rank subset of columns (so that $\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k$ continues to be nonsingular). The linear dependence of the columns of $\mathbf{D}_k$ is known as rank degeneracy [or deflation, which is the term used in Gutknecht (2007, sec. 8)]. Note that the term rank degeneracy includes the case when p does not divide n and thus $pk^* > n$, as pointed out in Remark 2.

O'Leary also considers the possibility of "deleting the zero or redundant column j of $\mathbf{D}_k$ and the corresponding columns of $\mathbf{X}_k$ and $\mathbf{R}_k$, and continuing the algorithm with $p - 1$ vectors…The resulting sequences retain all of the properties necessary to guarantee convergence" (O'Leary 1980, p. 301).

To consider the case where rank degeneracy occurs, denote as p_k the number of threads, such that rank $\mathbf{D}_k = p_k$. We assume that $p_0 = p$, and thus $p_k \le p$ for all $k > 0$.

Note that the best case is $p_k = p$ for all $k > 0$ until convergence occurs (there is no rank degeneracy). The worst case is $p_k = 1$, $\forall k > 0$ until convergence occurs.

Lemma 7 *There exists a finite natural number* k^**, such that*

$$\min_{k^* \in \mathbb{N}} \text{rank } [\mathbf{D}_i]_0^{k^*-1} = n.$$

The following theorem states conditions for convergence of the CCG algorithm in the general case where rank degeneracy may occur.

Theorem 6 *If rank* $\mathbf{R}_0 = p > 1$*, all the threads that converge in the conjugate gradient method* (10)–(13) *converge to* $\mathbf{x}^*$ *in* $\lceil \frac{n}{p} \rceil \le k^* \le n - p + 1$ *iterations, i.e.,* $\mathbf{x}_{i_{k^*}} = \mathbf{x}^*$ *and* $\mathbf{r}_{i_{k^*}} = \mathbf{d}_{i_{k^*}} = \mathbf{0}$ *for all* $i \in \{1, \ldots, p_{k^*-1}\}$.

The proof of this theorem is similar to that of Theorem 5 for all the threads that are not eliminated at any iteration by rank degeneracy, i.e., $\mathbf{x}_{i_k}$, $\forall i \in \{1, \ldots, p_{k^*-1}\}$. Note that, although rank degeneracy may occur, for all $i, j \in \{0, \ldots, k^* - 1\}$, $i \ne j$, $\mathbf{D}_i \in \mathbb{R}^{n \times p_i}$, $\mathbf{D}_j \in \mathbb{R}^{n \times p_j}$, the conjugacy property $\mathbf{D}_i^\top \mathbf{A}\mathbf{D}_j = 0 \in \mathbb{R}^{p_i \times p_j}$ continues to be valid, as well as the orthogonality property $\mathbf{R}_i^\top \mathbf{D}_j = 0 \in \mathbb{R}^{p_i \times p_j}$, $\forall j < i$.

The following lemma, proved in Bantegnies and Brezinski (2001, property 11) and O'Leary (1980, theorem 5), gives the rate of convergence of the CCG algorithm.

Lemma 8 *Let* $\kappa = \lambda_n / \lambda_1$ *be the condition number of A:*

$$\|\mathbf{x}_{i_{p_k}} - \mathbf{x}^*\|_\mathbf{A} = \frac{2}{\sqrt{\lambda_1}} \sum_{i=1}^{p} \|\mathbf{r}_{i_0}\| \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k. \tag{20}$$

Next, we present an example of the execution of the CCG algorithm for a dense randomly generated matrix and using six threads. In this example, no rank degeneracy occurs, so that the columns of the matrix $[\mathbf{D}_i]_0^k$ are linearly independent until the iteration k^*.

Example 1 Let $\mathbf{A} \in \mathbb{R}^{50\times 50}$ be a randomly generated symmetric positive definite matrix, $\mathbf{b} \neq \mathbf{0} \in \mathbb{R}^{50}$, and six initial conditions $\mathbf{X}_0 \in \mathbb{R}^{50\times 6}$ are also randomly chosen; thus, we have $p = 6$ threads. Table 1 reports the rank of the matrices $[\mathbf{D}_k]$ and $[\mathbf{D}_i]_0^k$ and the norm of the residual vector of every thread at each iteration k. Note that, in this example, the direction vectors are linearly independent ($[\mathbf{D}_i]_0^k$ has full column rank and hence no rank degeneracy occurs). The convergence is produced at an iteration $k^* = 9$, where the norm of the residual vectors are lower than 10^{-3}.

Bantegnies and Brezinski (2001, p. 12) affirms that, when $pk^* > n$, rank degeneracy always occurs in the last iteration, then a "breakdown occurs… and no general conclusion can be deduced". In fact, Algorithm 1 shows pseudocode for the cooperative conjugate gradient algorithm in the case when rank degeneracy may occur. In Algorithm 1, $\mathbf{X}|_{j\in J}$ refers to the matrix $\mathbf{X}$ from which the columns specified in the set J have been removed.

Algorithm 1 Pseudocode for the cooperative conjugate gradient algorithm, where $\mathbf{X}|_{j\in J}$ refers to the matrix $\mathbf{X}$ from which the columns specified in the set J have been removed.

Choose $\mathbf{X}_0 \in \mathbb{R}^{n\times p}$
$\mathbf{R}_0 := \mathbf{A}\mathbf{X}_0 - \mathbf{b}^{\mathsf{T}}\mathbf{1}_p$
$\mathbf{D}_0 := \mathbf{R}_0$
$p_{-1} := p$
$p_0 :=$ rank $\mathbf{R}_0$ ▷ p_0 is the initial value of the rank
$k := 0$
while $p_k > 0$, **do**
  if $p_k < p_{k-1}$ **then** ▷ $p_{k-1} - p_k$ threads are suppressed
    choose $J \subset \{1, .., p_{k-1}\}$, such that $\mathbf{D}_k|_{j\in J} \in \mathbb{R}^{n\times p_k}$ and rank $\mathbf{D}_k|_{j\in J} = p_k$
    $\mathbf{X}_k \leftarrow \mathbf{X}_k|_{j\in J}$ ▷ $\mathbf{X}_k \in \mathbb{R}^{n\times p_k}$
    $\mathbf{R}_k \leftarrow \mathbf{R}_k|_{j\in J}$ ▷ $\mathbf{R}_k \in \mathbb{R}^{n\times p_k}$
    $\mathbf{D}_k \leftarrow \mathbf{D}_k|_{j\in J}$ ▷ $\mathbf{D}_k \in \mathbb{R}^{n\times p_k}$
  end if
  $\boldsymbol{\alpha}_k := -\mathbf{R}_k^{\mathsf{T}}\mathbf{D}_k(\mathbf{D}_k^{\mathsf{T}}\mathbf{A}\mathbf{D}_k)^{-1}$ ▷ $\boldsymbol{\alpha}_k \in \mathbb{R}^{p_k\times p_k}$
  $\mathbf{X}_{k+1} := \mathbf{X}_k + \mathbf{D}_k\boldsymbol{\alpha}_k^{\mathsf{T}}$ ▷ $\mathbf{X}_{k+1} \in \mathbb{R}^{n\times p_k}$
  $\mathbf{R}_{k+1} := \mathbf{R}_k + \mathbf{A}\mathbf{D}_k\boldsymbol{\alpha}_k^{\mathsf{T}}$ ▷ $\mathbf{R}_{k+1} \in \mathbb{R}^{n\times p_k}$
  $\boldsymbol{\beta}_k := -\mathbf{R}_{k+1}^{\mathsf{T}}\mathbf{A}\mathbf{D}_k(\mathbf{D}_k^{\mathsf{T}}A\mathbf{D}_k)^{-1}$ ▷ $\boldsymbol{\beta}_k \in \mathbb{R}^{p_k\times p_k}$
  $\mathbf{D}_{k+1} := \mathbf{R}_{k+1} + \mathbf{D}_k\boldsymbol{\beta}_k^{\mathsf{T}}$ ▷ $\mathbf{D}_{k+1} \in \mathbb{R}^{n\times p_k}$
  $p_{k+1} :=$ rank $\mathbf{D}_{k+1}$
  $k \leftarrow k + 1$
end while

4 Estimates of operation counts and speedup of the CCG algorithm

This section estimates the operation counts and speedup of the CCG algorithm, when using the CCG algorithm with p threads, instead of CG and MPCG algorithms. The expected speedup is due to the parallelism inherent in a multi-thread implementation. To calculate these estimates for the cooperative conjugate gradient algorithm (10)–(13), we make the following assumptions:

1. the computations are carried out in exact arithmetic;
2. the only floating point operations that are counted are multiplication and division and both operations take the same amount of time;

Table 1 Example of the execution of the CCG algorithm with a randomly generated matrix $\mathbf{A} \in \mathbb{R}^{50\times 50}$ from the initial conditions that are the columns of a randomly generated matrix $\mathbf{X}_0 \in \mathbb{R}^{50\times 6}$, which implies the use of $p = 6$ threads

k	Rank $[\mathbf{D}_k]$	Rank $[\mathbf{D}_i]_0^k$	$\Vert\mathbf{r}_{1_k}\Vert$	$\Vert\mathbf{r}_{2_k}\Vert$	$\Vert\mathbf{r}_{3_k}\Vert$	$\Vert\mathbf{r}_{4_k}\Vert$	$\Vert\mathbf{r}_{5_k}\Vert$	$\Vert\mathbf{r}_{6_k}\Vert$
0	6	6	3.80×10^5	4.16×10^5	3.83×10^5	3.44×10^5	3.82×10^5	3.08×10^5
1	6	12	1.04×10^5	0.88×10^5	1.11×10^5	1.09×10^5	0.97×10^5	0.93×10^5
2	6	18	4.43×10^4	4.19×10^4	4.18×10^4	3.04×10^4	3.98×10^4	4.44×10^4
3	6	24	1.28×10^4	2.02×10^4	2.15×10^4	1.78×10^4	1.85×10^4	2.39×10^4
4	6	30	5.94×10^3	8.69×10^3	7.48×10^3	12.52×10^3	14.48×10^3	10.80×10^3
5	6	36	3.18×10^3	3.11×10^3	4.04×10^3	5.57×10^3	6.00×10^3	4.60×10^3
6	6	42	1.50×10^3	2.96×10^3	3.31×10^3	3.60×10^3	2.70×10^3	4.42×10^3
7	6	48	1.16×10^3	4.08×10^3	3.56×10^3	4.41×10^3	2.42×10^3	5.58×10^3
8	6	50	252.87	227.31	293.49	250.26	641.00	363.04
9	6	50	1.09×10^{-4}	1.13×10^{-4}	1.43×10^{-4}	1.36×10^{-4}	2.67×10^{-4}	0.45×10^{-4}

The right-hand side $\mathbf{b}$ is also random. The norm of the residual vector of each thread and the rank of the matrices $[\mathbf{D}_k]$ and $[\mathbf{D}_i]_0^k$ at each iteration k are reported

3. the worst case of finite termination in $\lceil \frac{n}{p} \rceil$ steps occurs, where n is the dimension of the matrix $\mathbf{A}$;
4. all threads are computationally identical: i.e., all floating point operations are executed in the same amount of time on each thread,
5. rank degeneracy does not occur, i.e., $p_k = p$ for all $k \in \{0, \ldots, k^* - 1\}$.

Rewriting Eqs. (10)–(13) to make the calculations in (10), (11) explicit, we assume that each iteration of the algorithm requires calculating the following recursions:

$$
\begin{aligned}
\mathbf{X}_{k+1} &= \mathbf{X}_k - \mathbf{D}_k(\mathbf{R}_k^\top \mathbf{D}_k(\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k)^{-1})^\top \\
\mathbf{R}_{k+1} &= \mathbf{R}_k - \mathbf{A}\mathbf{D}_k(\mathbf{R}_k^\top \mathbf{D}_k(\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k)^{-1})^\top \\
\mathbf{D}_{k+1} &= \mathbf{R}_{k+1} - \mathbf{D}_k(\mathbf{R}_{k+1}^\top \mathbf{A}\mathbf{D}_k(\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k)^{-1})^\top.
\end{aligned} \tag{21}
$$

Table 2, based on the iteration defined in (21), shows the number of floating point operations realized per iteration by each processor in a multi-thread implementation. Total computation time can be assumed to be proportional to the total number of floating point operations required to satisfy the stopping criterion, neglecting the time spent on communication.

In Table 2, the first column indicates the task carried out at each stage by every thread. The double lines, separating the first row from the second and the second from the third, indicate the necessity of a phase of information exchange: every thread at that stage needs to know results from other threads. The second column, labelled composite result, contains the information that is available by pooling the partial results from each thread and the third column gives the dimension of this composite result. The last three columns contain the number of operations carried out by the ith thread.

As indicated by the last line of Table 2, a total of $n^2 + 6np + \frac{p(p+1)(2p+1)}{3} - 2p$ multiplications per processor are needed to complete an iteration. In addition, $p(p+1)$ divisions are carried out per iteration. Since, generically speaking, the algorithm ends in at most $\frac{n}{p}$ iterations, an estimate of the worst case multi-thread execution time is given by the following result.

Theorem 7 (Worst case multi-thread CCG flop count) *The worst case multi-thread execution of CCG using p agents for a linear system* (1) *of size n requires*

$$
N_{\mathrm{CCG}}(p) = \frac{n^3}{p} + 6n^2 + \frac{2}{3}np^2 + 2np - \frac{2}{3}n \tag{22}
$$

multiplications and divisions carried out in parallel and synchronously, by each processor.

Note that (22) for $p = 1$ gives the worst case flop count for the CG algorithm:

$$
N_{\mathrm{CG}} = N_{\mathrm{CCG}}(1) = n^3 + 6n^2 + 2n \tag{23}
$$

multiplications and divisions.

Theorem 7 has the following important corollaries.

Corollary 1 (Multi-thread gain) *For problems of size n at least equal to* 8*, it is always beneficial to use $p \leq n$ processors rather than a single one. In other words, when $n \geq 8$,*

$$
\forall\, 1 \leq p \leq n, \quad N_{\mathrm{CCG}}(1) \geq N_{\mathrm{CCG}}(p) \tag{24}
$$

Table 2 Operations and corresponding number of floating point operations in the iteration k executed by each processor in a multi-thread implementation

Operation of proc. i[a]	Result[b]	Dimension of the result	Products	Additions	Divisions
$\mathbf{Ad}_i$	$\mathbf{AD}$	$n \times p$	n^2	$n(n-1)$	0
$\mathbf{d}_i^\top \mathbf{AD}$	$\mathbf{D}^\top \mathbf{AD}$	$p \times p$	np	$p(n-1)$	0
$\mathbf{r}_i^\top \mathbf{D}$	$\mathbf{R}^\top \mathbf{D}$	$p \times p$	np	$p(n-1)$	0
$\boldsymbol{\alpha}_i \mid \boldsymbol{\alpha}_i(\mathbf{D}^\top \mathbf{AD}) = \mathbf{r}_i^\top \mathbf{D}$	$\boldsymbol{\alpha} := \mathbf{R}^\top \mathbf{D}(\mathbf{D}^\top \mathbf{AD})^{-1}$	$p \times p$	$\frac{p(p+1)(2p+1)}{6} - p$		$\frac{p(p+1)}{2}$ [c]
$\mathbf{r}_i := \mathbf{r}_i - \mathbf{AD}\boldsymbol{\alpha}_i^\top$	$\mathbf{R} := \mathbf{R} - \mathbf{AD}\alpha^\top$	$n \times p$	np	np	0
$\mathbf{x}_i := \mathbf{x}_i - \mathbf{D}\boldsymbol{\alpha}_i^\top$	$\mathbf{X} := \mathbf{X} - \mathbf{D}\boldsymbol{\alpha}^\top$	$n \times p$	np	np	0
$\mathbf{r}_i^\top \mathbf{AD}$	$\mathbf{R}^\top \mathbf{AD}$	$p \times p$	np	$p(n-1)$	0
$\boldsymbol{\beta}_i \mid \boldsymbol{\beta}_i(\mathbf{D}^\top \mathbf{AD}) = \mathbf{r}_i^\top \mathbf{AD}$	$\boldsymbol{\beta} := \mathbf{R}^\top \mathbf{AD}(\mathbf{D}^\top \mathbf{AD})^{-1}$	$p \times p$	$\frac{p(p+1)(2p+1)}{6} - p$		$\frac{p(p+1)}{2}$ [c]
$\mathbf{d}_i := \mathbf{r}_i - \mathbf{D}\boldsymbol{\beta}_i^\top$	$\mathbf{D} := \mathbf{R} - \mathbf{D}\boldsymbol{\beta}^\top$	$n \times p$	np	np	0
Total number of products per iteration			$n^2 + 6np + \frac{p(p+1)(2p+1)}{3} - 2p$		
Total number of additions per iteration			$n^2 - n + 6np + \frac{p(p+1)(2p+1)}{3} - 5p$		
Total number of divisions per iteration			$p(p+1)$		

The double line indicates a stage at which communication between all threads occurs, which means that every thread i needs to know results from other threads

[a] Operation realized by the ith thread, for all $i \in \{1, \ldots, p\}$

[b] Composite result of the operations realized by all the p threads at the same time

[c] These numbers of floating points operations are needed to realize a Gaussian elimination by LU factorization (Strang 1988, p. 15)

Proof

$$
\begin{aligned}
N_{\mathrm{CCG}}(1) - N_{\mathrm{CCG}}(n) &= n^3 + 6n^2 + 2n - \left(\frac{2}{3}n^3 + 9n^2 - \frac{2}{3}n\right) \\
&= \frac{1}{3}(n^3 - 9n^2 + 8n) \\
&= \frac{1}{3}n(n-1)(n-8)
\end{aligned}
$$

Moreover,

$$
\left.\frac{\mathrm{d}N_{\mathrm{CCG}}}{\mathrm{d}p}\right|_{p=1} = \frac{1}{3}n\left(-3n^2 + 10\right)
$$

which is negative for $n \geq 2$, while

$$
\left.\frac{\mathrm{d}N_{\mathrm{CCG}}}{\mathrm{d}p}\right|_{p=n} = n\left(\frac{4}{3}n + 1\right) > 0
$$

The convexity of $N_{\mathrm{CCG}}(p)$ as a function of p then yields the conclusion that $N_{\mathrm{CCG}}(p) \leq N_{\mathrm{CCG}}(1)$ for any $1 \leq p \leq n$. □

Corollary 2 (Optimal multi-thread gain for CCG) *For any size n of the problem, there exists a unique optimal number p^* of processors minimizing $N_{\mathrm{CCG}}(p)$. Moreover, when $n \to +\infty$,*

$$
p^* \approx \left(\frac{3}{4}\right)^{\frac{1}{3}} n^{\frac{2}{3}} \tag{25a}
$$

$$
N_{\mathrm{CCG}}(p^*) \approx \left(\left(\frac{4}{3}\right)^{\frac{1}{3}} + \frac{2}{3}\left(\frac{3}{4}\right)^{\frac{2}{3}}\right) n^{2+\frac{1}{3}} \approx 1.651 n^{2+\frac{1}{3}} \tag{25b}
$$

Proof

$$
\frac{\mathrm{d}N_{\mathrm{CCG}}(p)}{\mathrm{d}p} = -\frac{n^3}{p^2} + \frac{4}{3}np + 2n.
$$

There exists a unique p^*, such that $dN_{\mathrm{CCG}}/dp = 0$. For this value, one has $n^2 = (p^*)^2(\frac{4}{3}p^* + 2)$, which yields the asymptotic behavior given in (25a). The value in (25b) is directly deduced, by substituting (25a) in (22). □

Corollary 1 implies that for $n > 8$, for every choice of the number of threads p: $N_{\mathrm{CCG}}(p) < N_{\mathrm{CG}}$, and thus, the CCG algorithm can be expected to converge in less time than the CG algorithm.

The important conclusion of Corollary 2 is that, in the asymptotic limit, as n becomes large, implying that the optimal p^* also increases according to (25a), solution of $\mathbf{Ax} = \mathbf{b}$ is possible, by the multi-thread method proposed in this paper, with a cost of $O(n^{2+\frac{1}{3}})$ floating point operations, showing a clear advantage over the classical result of $O(n^3)$ for Gaussian elimination.

4.1 Worst case operation count for other block-CG algorithms

The multi-parameter conjugate gradient algorithm proposed by Bantegnies and Brezinski (2001) and Brezinski (1999, sec. 4), carried out using only one thread, calculates a total number of scalar products and scalar divisions given by

$$N_{\text{MPCG}}(p) = pN_{\text{CCG}}(p) = n^3 + 6n^2p + \frac{2}{3}np^3 + 2np^2 - \frac{2}{3}np \tag{26}$$

so that with $p > 1$, $N_{\text{CCG}} < N_{MPCG}$, which means that the CCG algorithm implemented in a multi-thread context is faster than the MPCG algorithm. The fact that a lower number of iterations than the worst case n/p are expected in practice does not modify the analysis, because for the same number of iterations to reach convergence for both the algorithms, the total number of floating point operations executed by the CCG algorithm is always smaller than those executed by the MPCG algorithm.

We emphasize that, in practice, specially when sparse matrices are used, the number of iterations to attain a specified error tolerance is usually much lower than $\lceil \frac{n}{p} \rceil$ and, furthermore, this behavior is also observed with the MPCG and the B-CG algorithms (Meurant and Strakoš 2006), just as is the case with the classical CG algorithm.

The multiple search direction conjugate gradient algorithm (MSD-CG) (Gu et al. 2004a, b), also implemented in a multi-thread context, performs a number of scalar products per thread and per iteration equal to $\frac{n^2}{p} + 3\frac{n}{p} + 3n + \frac{p(p+1)(2p+1)}{3} - 2p$; a number of additions equal to $\frac{n^2}{p} + 3\frac{n}{p} + n + np + \frac{p(p+1)(2p+1)}{3} - 3p - 2$, and a number of scalar divisions equal to $p(p+1)$. Hence, assuming again that the time expended to calculate a scalar product is equal to the time expended to calculate a division, and neglecting the time used to calculate additions, the number of floating point operations per iteration performed by the MSD-CG algorithm is given by the following:

$$\frac{n^2}{p} + 3n + 3\frac{n}{p} + \frac{p(p+1)(2p+1)}{3} + p(p-1). \tag{27}$$

However, it is not proved that the MSD-CG algorithm converges in a finite number of iterations (Gu et al. 2004a, p. 1140), and, indeed, this is not expected to occur, essentially because the linear independence of the columns of $[\mathbf{D}_i]_0^k$ is lost. The only available result is that the convergence rate is at least as fast as that of the steepest descent method (Gu et al. 2004b, p. 1294). Another drawback of the MSD-CG method is that communication between all the threads is required after every operation.

5 Computational experiments

This section reports on experimental results obtained with both randomly generated linear systems of low dimension, as well as larger dimensional matrices that arise in real applications.

5.1 Experiments on dense randomly generated matrices of low dimension

For the experiments reported in this section, a suite of randomly generated symmetric positive definite matrices of small dimensions will be used for illustrative purposes. The dimensions of the matrices are 50, 100, 200, 300, and 1000, respectively, and they have condition numbers

Table 3 Properties of randomly generated s.p.d. test matrices: size, condition number

Matrix	Dimension	Condition no.
A1	50	10^3
A2	50	10^4
A3	50	10^5
B1	100	10^3
B2	100	10^4
B3	100	10^5
C1	200	10^3
C2	200	10^4
C3	200	10^5
D1	300	10^3
D2	300	10^4
D3	300	10^5
E3	1000	10^5

of 10^3, 10^4 and 10^5. The right-hand side $\mathbf{b}$ is also randomly chosen. Table 3 shows the matrix label, its dimension, and condition number. In all the cases, the CCG algorithm was tested from 20 different initial points per thread (i.e., 20 executions of the CCG algorithm from every initial point per thread). All these initial points are localized on a hypersphere of norm 1 centred on the known solution point, that is $\|\mathbf{x}_{0_i} - \mathbf{x}^*\| = 1,\ \forall i \in \{1, \ldots, 20\}$. The matrices tested, as well as the right-hand sides and the initial points are available on request.

The stopping criterion adopted is that at least one thread has a norm of its residue smaller than 10^{-8}. We test the CG algorithm (one thread), the CCG algorithm with two and three threads and, for comparative purposes, the MSD-CG algorithm was also implemented.

The 20 initial points tested in the MSD-CG (as well as the CG algorithm) are the same used by the first thread in the CCG algorithm. The partition into subdomains of the vector $\mathbf{d}_k$ which results in the matrix $\mathbf{D}_k$ was made according to the criterion given in Gu et al. (2004a, p. 1136). Two and three threads were tested. The stopping criterion used was $\|\mathbf{r}_k\| < 10^{-8}$.

Table 4 reports the mean number of iterations from every initial point and the mean number of floating point operations (proportional to the time of convergence, neglecting the communication time and the time expended to calculate additions) realized in each case.

5.2 Experiments on dense randomly generated matrices of larger dimension

This section reports on a suite of numerical experiments carried out on a set of random symmetric positive definite matrices of dimensions varying from 1000 to 25,000, the latter being the largest dimension that could be accommodated in the fast access RAM memory of the multi-core processor. The random symmetric matrices were generated using a C translation of Shilon's MATLAB code (Shilon 2006), which produces a matrix distribution uniform over the manifold of orthogonal matrices with respect to the induced $\mathbb{R}^{n^2}$ Lebesgue measure. The right-hand sides and initial conditions were also randomly generated, with all entries uniformly distributed on the interval $[-10, 10]$. In this section, the matrices used were dense and the use of preconditioners was not investigated.

Table 4 Mean number of iterations (it.) and mean number of floating point operations (flop) for every matrix tested

Matrix	CG		CCG				MSD-CG			
	$p = 1$		$p = 2$		$p = 3$		$p = 2$		$p = 3$	
	It.	Flop	It.	Flop	It.	Flop	It.	Flop	It.	Flop
A1	48.65	136,320	25.95	80,750	17.45	**59,923**	159.1	236,580	160.15	170,930
A2	52	145,704	27.35	85,113	21.5	**73,831**	405.35	602,760	443.1	472,940
A3	50.8	142,340	32.1	97,875	19.9	**68,337**	216.9	322,530	235.25	251,090
B1	64.85	687,540	42.25	507,343	33.75	**399,400**	158.1	863,540	155.7	586,570
B2	74.85	793,560	48.7	546,020	34.4	**407,090**	263.8	1.44×10^6	279.85	1.05×10^6
B3	90.1	955,240	90.4	963,270	53.75	**627,110**	268.55	1.46×10^6	300.85	1.13×10^6
C1	106.95	4.40×10^6	71.55	3.03×10^6	56.35	$\mathbf{2.45} \times 10^6$	240.05	5.02×10^6	257.4	3.64×10^6
C2	117.5	4.84×10^6	81.2	3.44×10^6	61.85	$\mathbf{2.69} \times 10^6$	402.9	8.42×10^6	407.9	5.78×10^6
C3	117.25	4.83×10^6	79.2	3.35×10^6	66.7	$\mathbf{2.90} \times 10^6$	414	8.65×10^6	446.35	6.32×10^6
D1	89.5	8.21×10^6	67	6.27×10^6	57	5.44×10^6	157.75	7.31×10^6	155.4	$\mathbf{4.85} \times 10^6$
D2	179.45	16.47×10^6	112.95	10.57×10^6	85.85	$\mathbf{8.19} \times 10^6$	437.4	20.27×10^6	465.7	14.54×10^6
D3	152	13.95×10^6	107.85	10.09×10^6	86.26	$\mathbf{8.23} \times 10^6$	576.55	26.73×10^6	607.5	18.97×10^6
E3	224.75	2.26×10^8	168.25	1.70×10^8	139.7	1.42×10^8	406.35	2.05×10^8	409.25	$\mathbf{1.38} \times 10^8$

$\mathbf{b} \neq \mathbf{0}$. In the CCG algorithm $flop = it.\left(n^2 + 6np + \frac{p(p+1)(2p+1)}{3} + p(p-1)\right)$. Stopping criteria $\|\mathbf{r}_i\| < 10^{-8}$ for some $i \in \{1, \ldots, p\}$. In the MSD-CG algorithm, $Flop = It.\left(\frac{n^2}{p} + 3n + 3\frac{n}{p} + \frac{p(2p+1)(p+1)}{3} + p(p-1)\right)$. Stopping criteria $\|\mathbf{r}\| < 10^{-8}$. The algorithms assume that rank degeneracy may occur (so p may be nonconstant). Trajectories were initiated at 20 different initial points per thread, all of them with norm one, surrounding the solution point. In the MSD-CG algorithm, each initial thread is a partition of every initial point. For each matrix, the floating point operation count in boldface occurs in the column corresponding to the algorithm that minimized this number

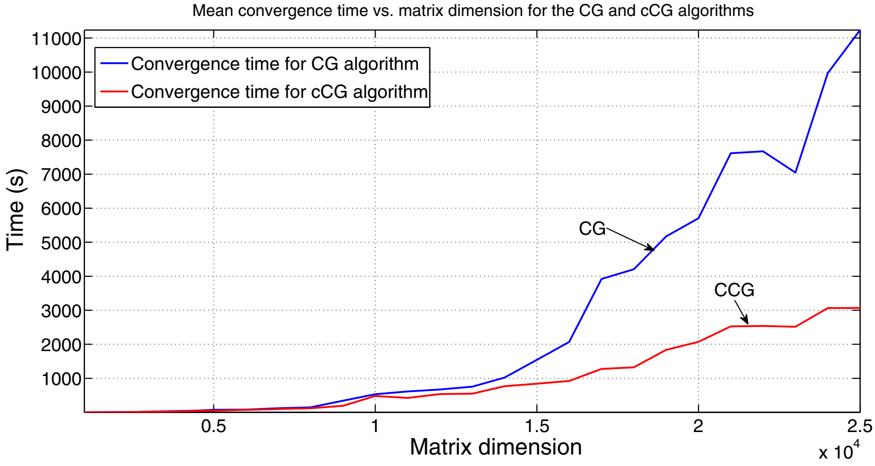

Fig. 1 Mean time to convergence for random test matrices of dimensions varying from 1000 to 25,000, condition number equal to 10^6, for 3 thread CCG and standard CG algorithms

The matrices used, the right-hand sides and initial conditions, as well as the numerical results of the tests (omitted here for lack of space) are available on request.

To evaluate the performance of the algorithm proposed in this paper, a program was written in language C. The compiler used was the GNU Compiler Collection (GCC), running under Linux Ubuntu 10.0.4. For the Linear Algebra calculations, we used the Linear Algebra Package (LAPACK) and the Basic Linear Algebra Subprograms (BLAS). Finally, to parallelize the program, we used the Open Multi Processing (OMP) API. The processor used was an Intel Core2Quad CPU Q8200 running at 2.33 MHz with four cores.

5.2.1 Experimental evaluation of speedup

The results of the cooperative 3 thread CCG, in comparison with standard CG, with a tolerance of 10^{-3}, and matrices with different sizes, but all with the same condition number of 10^6, are shown in Fig. 1. Multiple tests were performed, using different randomly generated initial conditions (20 different initial conditions for the small matrices and 10 for the bigger ones). Figure 1 shows the mean values computed for these tests. The *iteration speedup* of CCG in comparison with CG is defined as the mean number of iterations from every initial point that CG took to converge divided by the mean number of iterations that CCG took to converge, i.e.:

$$\text{Iteration speedup}(p) = \frac{\text{mean number of iterations CG}}{\text{mean number of iterations CCG}(p)}$$

Similarly, the *time speedup* (classical speedup) is the ratio of the time to convergence, that is, the mean time taken by the CG algorithm to run the main loop until convergence divided by the mean time taken by the CCG algorithm from every initial point, i.e.:

$$\text{Time speedup}(p) = \frac{\text{mean time of convergence CG}}{\text{mean time of convergence CCG}(p)}$$

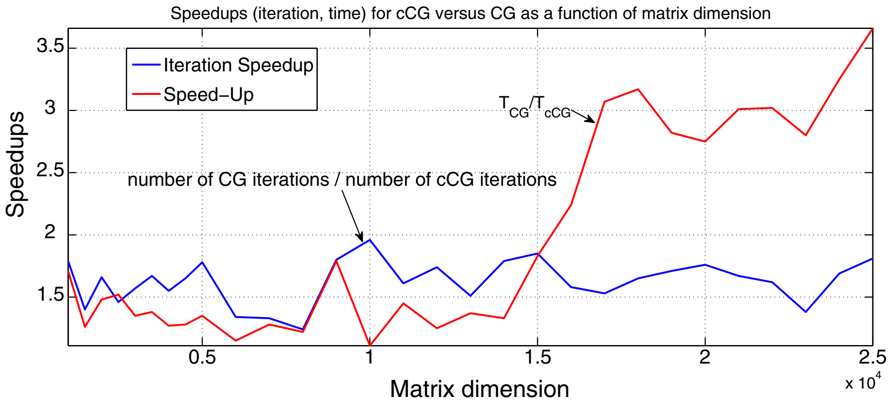

Fig. 2 Average speedups of Cooperative 3-thread CCG over classic CG for random test matrices of dimensions varying from 1000 to 25,000, condition number equal to 10^6

The experimental speedups for each dimension are shown in Fig. 2. The iteration and classical speedups seem to be roughly equal up to a certain size of matrix ($n = 16{,}000$); however, above this dimension, there is an increasing trend for both speedups.

The numerical results obtained show that CCG, using 3 threads, leads to an improvement in comparison with the usual CG algorithm. The average iteration speedup and the classical speedup of CCG are, respectively, 1.62 and 1.94, indicating that CCG converges almost twice as fast as CG for dense matrices with reasonably well-separated eigenvalues.

5.2.2 Verifying the flop count estimates

Figure 3 shows the mean time spent per iteration in seconds (points plotted as squares), versus matrix dimension, as well as the parabola fitted to this data, using least squares. Using the result from the last row of Table 2 and multiplying it by the mean time per scalar multiplication, we obtain the parabola (dash-dotted line in Fig. 3) expected in theory. To estimate the time per scalar multiplication, we divided the experimentally obtained mean total time spent on each iteration and divided it by the number of scalar multiplications performed in each iteration. This was done for each matrix dimension. Since the multi-core threads being used for all experiments are identical, each of these divisions should generate the same value of time taken to carry out each scalar multiplication, regardless of matrix dimension. It was observed that these divisions produced a data set which has a mean value of 8.10 ns per scalar multiplication, with a standard deviation of 1.01 ns, showing that the estimate is reasonable. From Eq. (22), substituting $p = 3$, neglecting lower order terms, and multiplying it by the estimated mean time per scalar multiplication (8.10 ns), the number of matrix multiplications per iteration, $N_{\mathrm{CCG}}(p)$, $p = 3$, is a cubic polynomial in n. Thus, the logarithm of the dimension (n) of the problem versus the logarithm of time needed to convergence is expected to be a straight line of slope 3. Figure 4 shows this straight line, fitted to the data (squares) by least squares. Its slope (2.542) is fairly close to 3, and data seem to follow a linear trend. The deviation of the slope from the ideal value has several probable causes, the first one being that the exact exponent of 3 is a result of a worst case analysis of CG in exact arithmetic. It is known that CG usually converges, to a reasonable tolerance, in much less than n iterations (Meurant and Strakoš 2006).

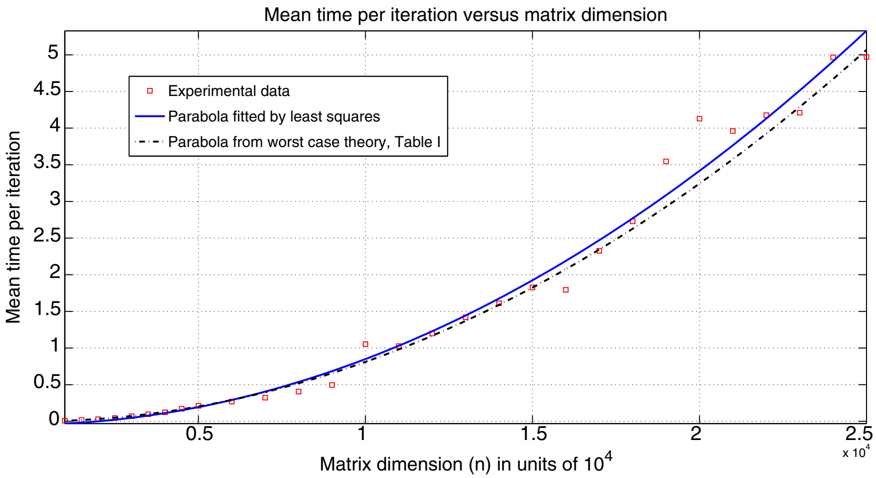

Fig. 3 Mean time per iteration versus problem dimension

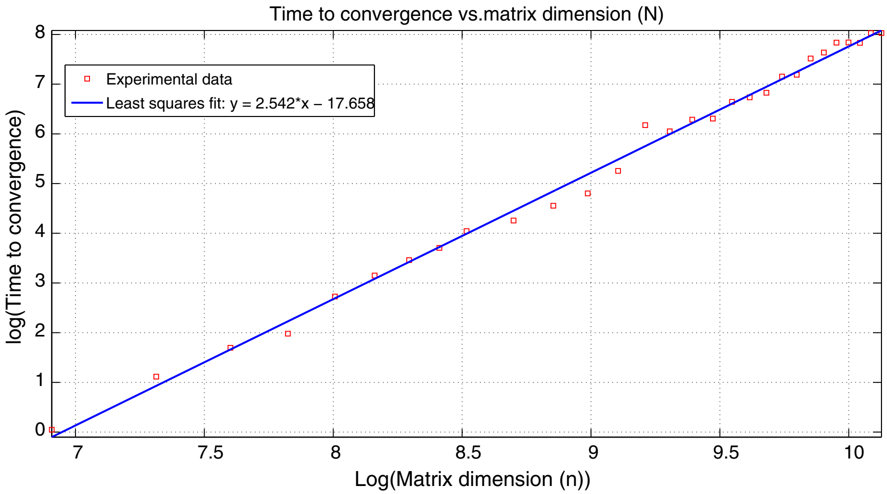

Fig. 4 Log–log plot of mean time to convergence versus problem dimension

Similarly, the logarithm of the number of iterations needed to convergence versus the logarithm of the dimension of the problem should also follow a linear trend. Since the number of iterations is expected to be $n/3$, the slope of this line should be 1. This log–log plot is shown in Fig. 5, in which the straight line was fitted by least squares to the original data (red squares). The slope (0.501) of the fitted line is smaller than 1, but is seen to fit the data well (small residuals). The fact that both slopes are smaller than their expected values indicates that the CCG algorithm is converging faster than the worst case estimate. Another reason is that a fairly coarse tolerance of 10^{-3} is used, and experiments reported show that decreasing the tolerance favors the CCG algorithm even more.

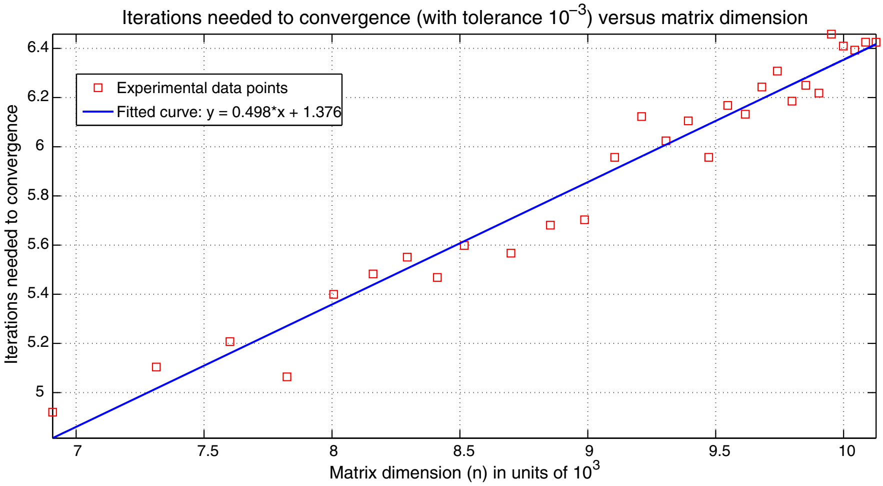

Fig. 5 Iterations needed to convergence versus problem dimension

5.3 Experiments on sparse matrices arising from real applications

In this section, the results of tests carried out with a suite of sparse symmetric positive definite matrices which arise from real applications are reported. The matrices chosen were taken from Davis and Hu (2009) and their characteristics are shown in Table 5. The right-hand side **b** was randomly chosen as well as the initial conditions, with all entries uniformly distributed on the interval $[-10, 10]$. The tests were performed from five different initial points per thread. The stopping criterion was that at least one thread has a norm of its residual vector lower than 10^{-8}.

Table 5 Sparse symmetric positive definite matrices extracted from Davis and Hu (2009)

Name	Dimension n	Nonzeros	Cond. number
HB\bcsstk14	1806	63,454	1.31e10
HB\bcsstk18	11,948	149,090	6.486e11
Lourakis\bundle1	10,581	770,811	1.3306e4
TKK\cbuckle	13,681	676,515	8.0476e7
JGD-Trefethen\Trefethen-20000b	19,999	554,435	–
JGD-Trefethen\Trefethen-20000	20,000	554,466	–
MathWorks\Kuu	7102	340,200	3.2553e4
Pothen\bodyy4	17,546	121,550	1.016e3
Pothen\bodyy5	18,589	128,853	9.9769e3
Pothen\bodyy6	19,366	134,208	9.7989e4
UTEP\Dubcova1	16,129	253,009	2.6247e3
HB\gr-30-30	900	7744	377.23

Dimension, number of entries different to zero, and approximate condition number. The dash means that the condition number was not specified or calculated

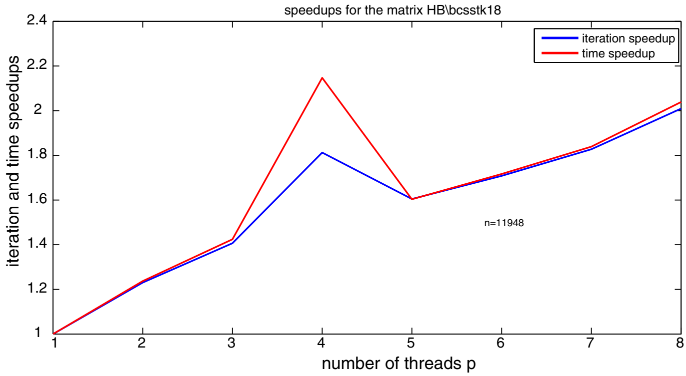

Fig. 6 Iteration speedup and time speedup for the matrix HB\bcsstk18 vs. number of threads p

The right-hand sides used, the initial conditions as well as the numerical results of the tests are available on request.

The code for the CCG algorithm was written in language C, with compiler GNU Compiler Collection (GCC), running under Linux Ubuntu 10.0.4. We also used the Linear Algebra Package (LAPACK) and the Basic Linear Algebra Subprograms (BLAS). Finally, to parallelize the program, we used the Open Multi Processing (OMP) API. The processor used was an Intel Core i7-4770 3.4 GHz.

The tests were performed varying the number of threads from 1 (classical CG) to 8 and reporting the mean number of iterations to reach the stopping criterion and the mean time of convergence of the five executions performed from each one of the different initial points.

For illustrative purposes, Figs. 6, 7 and 8 show the speedups for a number of threads varying from 1 to 8 with respect to the classical CG algorithm for the matrices bcsstk18, cbuckle, and gr-30-30, respectively.

Note that if the CCG algorithm and the CG algorithm converge in the worst case, the numbers of iterations to converge are $\lceil \frac{n}{p} \rceil$ and n, respectively, and hence, for $n \gg p$, the expected iteration speedup is equal to p (a straight line of slope 1). Similarly, the expected time speedup in the worst case is given by the time to convergence of the CG algorithm, which is proportional to N_{CG} given by (23) divided by the time to convergence taken by the CCG algorithm, which is proportional to $N_{\mathrm{CCG}}(p)$ given by (22); for large n, this expected time speedup in the worst case is also approximately equal to p:

$$\text{Expected worst case time speedup} = \frac{n^3 + 6n^2 + 2n}{\frac{n^3}{p} + 6n^2 + \frac{2}{3}np^2 + 2np - \frac{2}{3}n}.$$

Figure 9 shows the iteration speedup and the time speedup for each matrix reported in Table 5 and for each thread tested from 1 to 8. Figure 9 shows that for all the matrices tested, the greater the number of threads, the greater the speedups, exactly as expected when a small number of threads is used and as it happens in the worst case. For some matrices, an increase in the number of threads does not increase the speedup significantly, as observed with the matrices bodyy4, bodyy5, and bodyy6, whereas with other matrices, the increase of the speedup with the number of threads is much greater. Further research is needed to explain these results and correlate them to, for example, the eigenvalue distribution of the matrices.

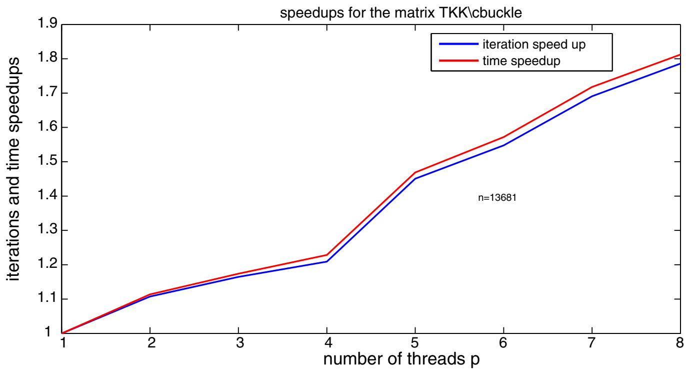

Fig. 7 Iteration speedup and time speedup for the matrix TKK\cbuckle vs. number of threads p

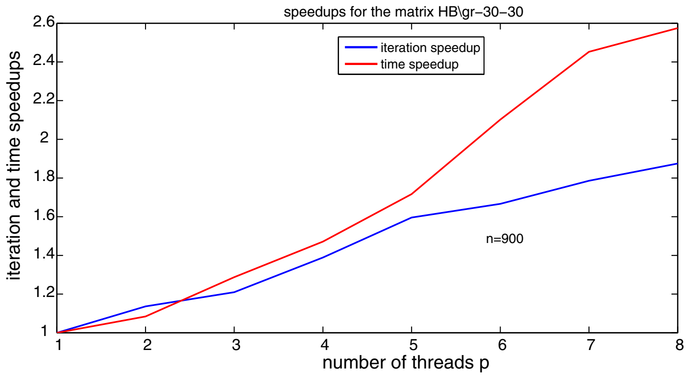

Fig. 8 Iteration speedup and time speedup for the matrix HB\gr-30-30 vs. number of threads p

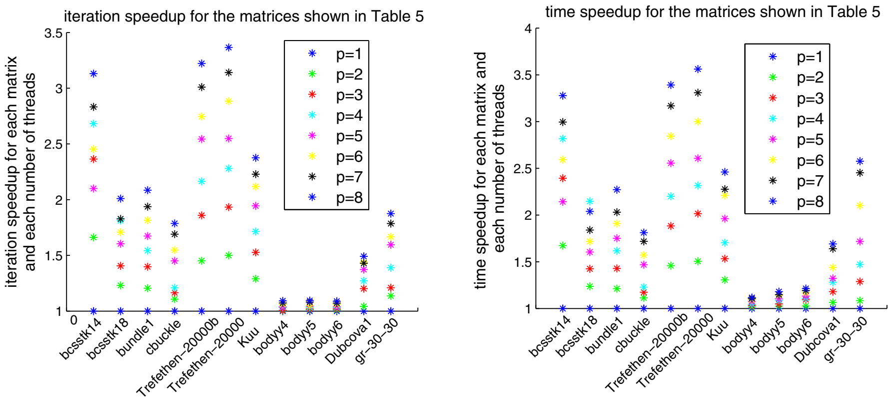

Fig. 9 Iteration speedup (*left*) and time speedup (*right*) for each matrix reported in Table 5 and for each number of threads used by the CCG algorithm

Figure 10 shows the average iteration and average time speedups for the 12 matrices reported in Table 5. Figure 10 confirms the trend of speedup increasing with the number of threads.

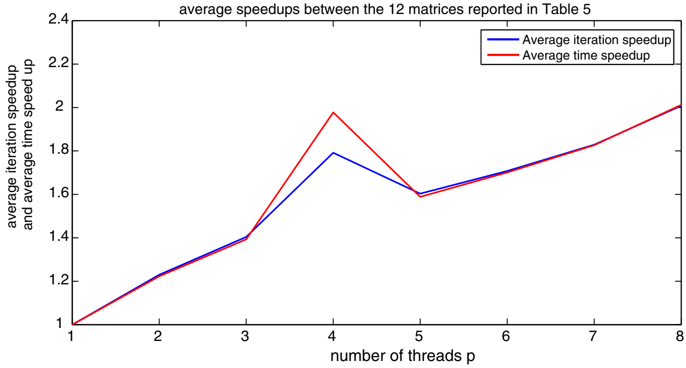

Fig. 10 Average iteration speedup and time speedup for the 12 matrices reported in Table 5 vs. number of threads p

6 Conclusions

This paper revisited some existing block and multiparameter CG algorithms in the new context of multi-thread computing, proposing a cooperative conjugate gradient (CCG) method for linear systems with symmetric positive definite coefficient matrices. This CCG method permits efficient implementation on a multi-core computer and experimental results bear out the main theoretical properties, namely, that speedups close to the theoretical value of p, when a p-core computer is used, are possible, when the matrix dimension is suitably large.

The experimental results were carried out with dense randomly generated matrices as well as with matrices arising from real applications, which are typically sparse and sometimes ill-conditioned. In all the cases, the increase of the speedups with the number of threads was observed, although the results are less significant for some matrices than for others, which is a topic requiring further investigation.

The comparison with the other multi-thread block-CG method presented in the literature, the MSD-CG (Gu et al. 2004a, b) showed that the CCG algorithm converges faster than the MSD-CG (with the same number of threads), in almost all the cases. The tests with large matrices, either dense and randomly generated or sparse arising from real applications, show that the CCG algorithm is faster than the classic CG and that the speedup increases with the number of threads.

The use of processors with a larger number of threads should also permit further exploration of the notable theoretical result of Corollary 2 that, in the asymptotic limit, as n becomes large, implying that the optimal number of threads p^* also increases according to (25a), solution of $\mathbf{Ax} = \mathbf{b}$ is possible by the method proposed here with a worst case cost of $O(n^{2+\frac{1}{3}})$ floating point operations.

Compliance with ethical standards

Funding A. Bhaya's work was supported by a BPP grant, G. Niedu, and F. Pazos were at UFRJ and supported by DS and PNPD fellowships, respectively, all of them from the National Counsel of Technological and Scientific Development (CNPq), while this paper was being written.

Appendix: Proofs of results in Sects. 3.1 and 3.2

Proof of Theorem 4 For all $i \in \{1, \ldots, p\}$, denoting

$$h_i(\boldsymbol{\gamma}_{i_0}, \boldsymbol{\gamma}_{i_1}, \ldots, \boldsymbol{\gamma}_{i_{k-1}}) := f(\mathbf{x}_{i_0} + \mathbf{D}_0\boldsymbol{\gamma}_{i_0}^{\mathsf{T}} + \mathbf{D}_1\boldsymbol{\gamma}_{i_1}^{\mathsf{T}} + \cdots + \mathbf{D}_{k-1}\boldsymbol{\gamma}_{i_{k-1}}^{\mathsf{T}}) \in \mathbb{R}$$

where $\boldsymbol{\gamma}_{i_\ell} \in \mathbb{R}^{1\times p}$, $\ell \in \{0, \ldots, k-1\}$ are row vectors; the coefficients $\boldsymbol{\gamma}_{i_\ell}$ that minimize the scalar function $f(\mathbf{x})$ on the affine set $\mathbf{x}_{i_0} + \operatorname{span}\,[\mathbf{D}_j]_0^{k-1}$ are given by the following:

$$\begin{aligned}\forall \ell < k : \frac{\partial h_i}{\partial \boldsymbol{\gamma}_{i_\ell}}^{\mathsf{T}} &= \nabla^{\mathsf{T}} f(\mathbf{x}_{i_0} + \mathbf{D}_0\boldsymbol{\gamma}_{i_0}^{\mathsf{T}} + \mathbf{D}_1\boldsymbol{\gamma}_{i_1}^{\mathsf{T}} + \cdots + \mathbf{D}_{k-1}\boldsymbol{\gamma}_{i_{k-1}}^{\mathsf{T}})\mathbf{D}_\ell \\ &= (\mathbf{r}_{i_0} + \mathbf{A}\mathbf{D}_0\boldsymbol{\gamma}_{i_0}^{\mathsf{T}} + \mathbf{A}\mathbf{D}_1\boldsymbol{\gamma}_{i_1}^{\mathsf{T}} + \cdots + \mathbf{A}\mathbf{D}_{k-1}\boldsymbol{\gamma}_{i_{k-1}}^{\mathsf{T}})^{\mathsf{T}}\mathbf{D}_\ell \\ &= \mathbf{r}_{i_0}^{\mathsf{T}}\mathbf{D}_\ell + \boldsymbol{\gamma}_{i_\ell}\mathbf{D}_\ell^{\mathsf{T}}\mathbf{A}\mathbf{D}_\ell = \mathbf{0}\end{aligned}$$

which implies that $\boldsymbol{\gamma}_{i_\ell} = -\mathbf{r}_{i_0}^{\mathsf{T}}\mathbf{D}_\ell(\mathbf{D}_\ell^{\mathsf{T}}\mathbf{A}\mathbf{D}_\ell)^{-1} \in \mathbb{R}^{1\times p}$; and considering all the row vectors $i \in \{1, \ldots, p\}$:

$$\gamma_\ell = -\mathbf{R}_0^{\mathsf{T}}\mathbf{D}_\ell(\mathbf{D}_\ell^{\mathsf{T}}\mathbf{A}\mathbf{D}_\ell)^{-1} \in \mathbb{R}^{p\times p}. \tag{28}$$

For all $\mathbf{x} \in \mathbf{x}_{i_0} + \operatorname{span}\,[\mathbf{D}_j]_0^{\ell-1}$: $\mathbf{x} = \mathbf{x}_{i_0} + \mathbf{D}_0\boldsymbol{\delta}_0^{\mathsf{T}} + \cdots + \mathbf{D}_{\ell-1}\boldsymbol{\delta}_{\ell-1}^{\mathsf{T}}$, where $\boldsymbol{\delta}_j \in \mathbb{R}^{1\times p}$, $\forall j \in \{0, \ldots, \ell-1\}$. Thus

$$\nabla f(\mathbf{x}) = \mathbf{r}_{i_0} + \mathbf{A}\mathbf{D}_0\boldsymbol{\delta}_0^{\mathsf{T}} + \cdots + \mathbf{A}\mathbf{D}_{\ell-1}\boldsymbol{\delta}_{\ell-1}^{\mathsf{T}}, \quad \text{which implies } \nabla^{\mathsf{T}} f(\mathbf{x})\mathbf{D}_\ell = \mathbf{r}_{i_0}^{\mathsf{T}}\mathbf{D}_\ell$$

and this is valid for all $\mathbf{x} \in \mathbf{x}_{i_0} + \operatorname{span}\,[\mathbf{D}_j]_0^{\ell-1}$, and hence, it is valid for $\mathbf{x}_{i_\ell} = \mathbf{x}_{i_0} + \mathbf{D}_0\boldsymbol{\alpha}_{i_0}^{\mathsf{T}} + \cdots + \mathbf{D}_{\ell-1}\boldsymbol{\alpha}_{i_{\ell-1}}^{\mathsf{T}}$, where $\boldsymbol{\alpha}_{i_j}^{\mathsf{T}}$ is the ith column of the $\boldsymbol{\alpha}_j^{\mathsf{T}}$ matrix (12) for all $j \in \{0, \ldots, \ell-1\}$.

Therefore, $\nabla^{\mathsf{T}} f(\mathbf{x}_{i_\ell})\mathbf{D}_\ell = \mathbf{r}_{i_\ell}^{\mathsf{T}}\mathbf{D}_\ell = \mathbf{r}_{i_0}^{\mathsf{T}}\mathbf{D}_\ell$, and considering all the row vectors $\mathbf{r}_{i_\ell}^{\mathsf{T}}$, $i \in \{1, \ldots, p\}$:

$$\mathbf{R}_\ell^{\mathsf{T}}\mathbf{D}_\ell = \mathbf{R}_0^{\mathsf{T}}\mathbf{D}_\ell$$

and substituting in (28):

$$\gamma_\ell = -\mathbf{R}_\ell^{\mathsf{T}}\mathbf{D}_\ell(\mathbf{D}_\ell^{\mathsf{T}}\mathbf{A}\mathbf{D}_\ell)^{-1} = \boldsymbol{\alpha}_\ell \quad \forall \ell < k \tag{29}$$

which proves that the step size (12) minimizes $f(\mathbf{x}_{i_k})$ on the affine set $\mathbf{x}_{i_0} + \operatorname{span}\,[\mathbf{D}_j]_0^{k-1}$ for all $i \in \{1, \ldots, p\}$.

Note also that, by (10) $\mathbf{x}_{i_k} = \mathbf{x}_{i_0} + \mathbf{D}_0\boldsymbol{\alpha}_{i_0}^{\mathsf{T}} + \mathbf{D}_1\boldsymbol{\alpha}_{i_1}^{\mathsf{T}} + \cdots + \mathbf{D}_{k-1}\boldsymbol{\alpha}_{i_{k-1}}^{\mathsf{T}}$, hence, for all $\ell < k$:

$$\begin{aligned}\nabla^{\mathsf{T}} f(\mathbf{x}_{i_k})\mathbf{D}_\ell &= \nabla^{\mathsf{T}} f(\mathbf{x}_{i_0} + \mathbf{D}_0\boldsymbol{\alpha}_{i_0}^{\mathsf{T}} + \mathbf{D}_1\boldsymbol{\alpha}_{i_1}^{\mathsf{T}} + \cdots + \mathbf{D}_{k-1}\boldsymbol{\alpha}_{i_{k-1}}^{\mathsf{T}})\mathbf{D}_\ell \\ &= (\mathbf{r}_{i_0} + \mathbf{A}\mathbf{D}_0\boldsymbol{\alpha}_{i_0}^{\mathsf{T}} + \mathbf{A}\mathbf{D}_1\boldsymbol{\alpha}_{i_1}^{\mathsf{T}} + \cdots + \mathbf{A}\mathbf{D}_{k-1}\boldsymbol{\alpha}_{i_{k-1}}^{\mathsf{T}})^{\mathsf{T}}\mathbf{D}_\ell = \mathbf{r}_{i_k}^{\mathsf{T}}\mathbf{D}_\ell = \mathbf{0}\end{aligned}$$

and considering all the row vectors $i \in \{1, \ldots, p\}$:

$$\mathbf{R}_k^{\mathsf{T}}\mathbf{D}_\ell = \mathbf{0} \quad \forall \ell < k. \tag{30}$$

□

Proof of Lemma 3 By Theorem 4, if $\mathbf{R}_k$ is orthogonal to $\operatorname{span}\,[\mathbf{D}_i]_0^{k-1}$, then this is orthogonal to $\operatorname{span}\,[\mathbf{R}_i]_0^{k-1}$, which means that for all $j < k$: $\mathbf{R}_k^{\mathsf{T}}\mathbf{R}_j = 0$.

By (10): $\forall j < k : \mathbf{X}_{j+1} = \mathbf{X}_j + \mathbf{D}_j\boldsymbol{\alpha}_j^{\mathsf{T}}$; hence, $\mathbf{D}_j\boldsymbol{\alpha}_j^{\mathsf{T}} = \mathbf{X}_{j+1} - \mathbf{X}_j$ which implies $\mathbf{R}_k^{\mathsf{T}}\mathbf{A}\mathbf{D}_j\boldsymbol{\alpha}_j^{\mathsf{T}} = \mathbf{R}_k^{\mathsf{T}}(\mathbf{R}_{j+1} - \mathbf{R}_j)$

Supposing rank $\mathbf{D}_j = p$, $\boldsymbol{\alpha}_j$ is nonsingular, thus $\forall j < k-1: \ \mathbf{R}_k^\top \mathbf{A}\mathbf{D}_j = 0$
For $j = k-1: \ \mathbf{R}_k^\top \mathbf{A}\mathbf{D}_{k-1}\alpha_{k-1}^\top = \mathbf{R}_k^\top \mathbf{R}_k \neq 0 \ \Rightarrow \ \boldsymbol{\alpha}_{k-1}^\top = (\mathbf{R}_k^\top \mathbf{A}\mathbf{D}_{k-1})^{-1}\mathbf{R}_k^\top \mathbf{R}_k$
Using this result in (14):

$$\begin{aligned}\mathbf{D}_{k+1} &= \mathbf{R}_{k+1} - \sum_{j=0}^{k} \mathbf{D}_j(\mathbf{D}_j^\top \mathbf{A}\mathbf{D}_j)^{-1}\mathbf{D}_j^\top \mathbf{A}\mathbf{R}_{k+1} \\ &= \mathbf{R}_{k+1} - \mathbf{D}_k(\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k)^{-1}\mathbf{D}_k^\top \mathbf{A}\mathbf{R}_{k+1} \\ &= \mathbf{R}_{k+1} - \mathbf{D}_k(\mathbf{R}_{k+1}^\top \mathbf{A}\mathbf{D}_k(\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k)^{-1})^\top = \mathbf{R}_{k+1} + \mathbf{D}_k\boldsymbol{\beta}_k^\top \end{aligned} \tag{31}$$

which coincides with (11) with step size (13), thus proving that the matrices generated by this method are also conjugate. □

Proof of Lemma 6 By induction. Since the columns of $\mathbf{D}_0$ are linearly independent by hypotheses, supposing rank $[\mathbf{D}_i]_0^{k-1} = pk$, it is enough to prove that the columns of $[\mathbf{D}_i]_0^k$ also are linearly independent if $p(k+1) \le n$.

By Eq. (11): $\mathbf{D}_{k+1} = \mathbf{R}_{k+1} + \mathbf{D}_k\boldsymbol{\beta}_k^\top$.

Evidently, $\{\mathbf{d}_{i_k}\}_1^p \subset$ span $[\mathbf{D}_i]_0^k$. Using the proof of the Lemma 3, $\{\mathbf{r}_{i_{k+1}}\}_1^p \subset$ span $[\mathbf{A}^i\mathbf{R}_0]_0^{k+1} =$ span $[\mathbf{D}_i]_0^{k+1}$ and $\{\mathbf{r}_{i_{k+1}}\}_1^p \not\subset$span $[\mathbf{A}^i\mathbf{R}_0]_i^k =$ span $[\mathbf{D}_i]_0^k$, because $\mathbf{R}_{k+1}$ is orthogonal to span $[\mathbf{D}_i]_0^k$. Hence, the columns of $\mathbf{R}_{k+1}$ can be expressed neither as a linear combination of $[\mathbf{D}_0 \cdots \mathbf{D}_k]$, and, from (11), nor as a linear combination of the columns of $\mathbf{D}_{k+1}$, which means that the columns of $[\mathbf{D}_0 \cdots \mathbf{D}_k \mathbf{D}_{k+1}] = [\mathbf{D}_i]_0^{k+1}$ are linearly independent.

Note that this linear independence persists until an iteration k, such that $p(k+1) \ge n$, and in this case, $[\mathbf{D}_i]_0^k \in \mathbb{R}^{n \times p(k+1)}$ and rank $[\mathbf{D}_i]_0^k = n \le p(k+1)$. □

Proof of Theorem 5 By Lemma 6 $[\mathbf{D}_0 \cdots \mathbf{D}_{k^*-1}] \in \mathbb{R}^{n \times pk^*}$, and rank $[\mathbf{D}_0 \cdots \mathbf{D}_{k^*-1}] = n = pk^*$. Hence, every vector $\mathbf{x}^* - \mathbf{x}_{i_0}$, $\forall i \in \{1, \dots, p\}$ can be expressed as a linear combination of a base of the subspace span $[\mathbf{D}_i]_0^{k^*-1}$:

$$\mathbf{x}^* - \mathbf{x}_{i_0} = \mathbf{D}_0\boldsymbol{\gamma}_{i_0}^\top + \mathbf{D}_1\boldsymbol{\gamma}_{i_1}^\top + \cdots + \mathbf{D}_{k^*-2}\boldsymbol{\gamma}_{i_{k^*-2}}^\top + \mathbf{D}_{k^*-1}\boldsymbol{\gamma}_{i_{k^*-1}}^\top \tag{32}$$

where $\boldsymbol{\gamma}_{i_k} \in \mathbb{R}^{1\times p}$, $k \in \{0, \dots, k^*-1\}$ are the coefficients of the linear combination of $\mathbf{x}^* - \mathbf{x}_{i_0}$.

Thus, for all $k \in \{0, \dots, k^*-1\}$, $i \in \{1, \dots, p\}$:

$$\mathbf{D}_k^\top \mathbf{A}(\mathbf{x}^* - \mathbf{x}_{i_0}) = \mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k\boldsymbol{\gamma}_{i_k}^\top \in \mathbb{R}^p \quad \Rightarrow \quad \boldsymbol{\gamma}_{i_k}^\top = (\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k)^{-1}\mathbf{D}_k^\top \mathbf{A}(\mathbf{x}^* - \mathbf{x}_{i_0}).$$

Following the sequence (10), from $\mathbf{x}_{i_0}$ to $\mathbf{x}_{i_k}$ for all $i \in \{1, \dots, p\}$:

$$\begin{aligned}\mathbf{x}_{i_k} - \mathbf{x}_{i_0} &= \mathbf{D}_0\boldsymbol{\alpha}_{i_0}^\top + \mathbf{D}_1\boldsymbol{\alpha}_{i_1}^\top + \cdots + \mathbf{D}_{k-1}\boldsymbol{\alpha}_{i_{k-1}}^\top \\ &\Rightarrow \mathbf{D}_k^\top \mathbf{A}(\mathbf{x}_{i_k} - \mathbf{x}_{i_0}) = 0 \\ &\Rightarrow \mathbf{D}_k^\top \mathbf{A}\mathbf{x}_{i_k} = \mathbf{D}_k^\top \mathbf{A}\mathbf{x}_{i_0}\end{aligned}$$

where Lemma 3 was used and $\boldsymbol{\alpha}_{i_j}$ is the ith row of the $\boldsymbol{\alpha}_j$ matrix calculated as in (12). Substituting in the former equation:

$$\boldsymbol{\gamma}_{i_k}^\top = (\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k)^{-1}\mathbf{D}_k^\top \mathbf{A}(\mathbf{x}^* - \mathbf{x}_{i_k}) = -(\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k)^{-1}\mathbf{D}_k^\top \mathbf{r}_{i_k}$$

and considering all the p rows:

$$\Gamma_k^\top := [\boldsymbol{\gamma}_{i_k}^\top]_1^p = -(\mathbf{D}_k^\top \mathbf{A}\mathbf{D}_k)^{-1}\mathbf{D}_k^\top \mathbf{R}_k$$

which coincides with (12). Hence, the coefficients of the linear combination (32) are the step sizes $\boldsymbol{\alpha}_k$, and the sequence

$$\mathbf{x}_{i_{k^*}} = \mathbf{x}_{i_0} + \mathbf{D}_0 \boldsymbol{\alpha}_{i_0}^{\mathsf{T}} + \cdots + \mathbf{D}_{k^*-1} \boldsymbol{\alpha}_{i_{k^*-1}}^{\mathsf{T}} \tag{33}$$

for all $i \in \{1, \ldots, p\}$ is equal to $\mathbf{x}^*$, thus proving that all the p threads converge in k^* iterations. □

Proof of Lemma 7 By Lemma 6, for all $k > 0$ until convergence, if $\mathbf{D}_k \neq 0$, then $\{\mathbf{d}_{i_k}\}_1^{p_k} \not\subset \text{span}\ [\mathbf{D}_i]_0^{k-1}$ and rank $[\mathbf{d}_{i_k}]_1^{p_k} = p_k \geq 1$. Therefore, the matrix $[\mathbf{D}_0 \cdots \mathbf{D}_{k^*-1}]$, where k^* is chosen such that min rank $[\mathbf{D}_0 \cdots \mathbf{D}_{k^*-1}] \geq n$, has a finite number of columns.

Finally, we can choose $p_{k^*-1} = n - \sum_{k=0}^{k^*-2} p_k$, that is, we eliminate columns of $\mathbf{D}_{k^*-1}$ in such a way to have a number of columns enough to complete n linearly independent columns, i.e., rank $[\mathbf{D}_i]_0^{k^*-1} =$ rank $\mathbf{D}_0 + \cdots +$ rank $\mathbf{D}_{k^*-1} = p_0 + \cdots + p_{k^*-1} = n$. □

Note that in the best case, $p_k = p$, $\forall k \in \{0, \ldots, k^* - 1\}$, which implies that $k^* = \lceil \frac{n}{p} \rceil$. In the general case, $\lceil \frac{n}{p} \rceil \leq k^* \leq n - p + 1$.

References

Abkowicz A, Brezinski C (1996) Acceleration properties of the hybrid procedure for solving linear systems. Appl Math 4(23):417–432

Bantegnies, F., Brezinski, C.: The multiparameter conjugate gradient algorithm. Technical Report 429, Laboratoire d'Analyse Numérique et d'Optimisation. Université des Sciences et Technologies de Lille, Lille, France (2001)

Bhaya A, Bliman PA, Pazos F (2010) Cooperative parallel asynchronous computation of the solution of symmetric linear systems. In: Proceedings of the 49th IEEE conference on decision and control, Atlanta, USA

Bhaya A, Bliman PA, Niedu G, Pazos F (2012) A cooperative conjugate gradient method for linear systems permitting multithread implementation of low complexity. In: Proceedings of the 51st IEEE conference on decision and control, Maui, Hawaii, USA

Bouyouli R, Meurant G, Smoch L, Sadok H (2008) New results on the convergence of the conjugate gradient method. In: Numerical linear algebra with applications, pp 1–12

Brezinski C (1999) Multiparameter descent methods. Linear Algebra Appl 296:113–141

Brezinski C, Chehab JP (1998) Nonlinear hybrid procedures and fixed point iterations. Numer Funct Anal Optim 19:465–487

Brezinski C, Redivo-Zaglia M (1994) Hybrid procedures for solving linear systems. Numer Math 67:1–19

Bridson R, Greif C (2006) A multipreconditioned conjugate gradient algorithm. SIAM J Matrix Anal Appl 27(4):1056–1068

Davis T, Hu Y (2009) Sparse matrix collection. University of Florida, USA. http://www.cise.ufl.edu/research/sparse/matrices/. Accessed 16 Dec 2016

Greenbaum A (1997) Iterative methods for solving linear systems. SIAM, Philadelphia

Gu T, Liu X, Mo Z, Chi X (2004a) Multiple search direction conjugate gradient method 1: methods and their propositions. Int J Comput Math 81(9):1133–1143

Gu T, Liu X, Mo Z, Chi X (2004b) Multiple search direction conjugate gradient method 2: theory and numerical experiments. Int J Comput Math 81(10):1289–1307

Güler O (2010) Foundations of optimization. Graduate texts in mathematics. Springer, New York

Gutknecht MH (2007) Block Krylov space methods for linear systems with multiple right-hand sides: an introduction. In: Siddiqi IDAH, Christensen O (eds) Modern mathematical models, methods and algorithms for real world systems. Anamaya Publishers, New Delhi, pp 420–447

Hestenes MR, Stiefel E (1952) Methods of conjugate gradients for solving linear systems. J Res Natl Bur Stand 49:409–436

Kumar V, Leonard N, Morse AS (eds) (2005) 2003 Block Island workshop on cooperative control, lecture notes in control and information sciences, vol 309. Springer

Luenberger DG, Ye Y (2008) Linear and nonlinear programming, 3rd edn. Springer, New York

Meurant G, Strakoš Z (2006) The Lanczos and conjugate gradient algorithms in finite precision arithmetic. Acta Numer 15:471–542

Murray RM (2007) Recent research in cooperative control of multi-vehicle systems. J Guid Control Dyn 129(5):571–583

Nedic A, Ozdaglar A (2010) Convex optimization in signal processing and communications, chap. Cooperative distributed multi-agent optimization. Cambridge University Press, Cambridge

O'Leary DP (1980) The block conjugate gradient algorithm and related methods. Linear Algebra Appl 29:293–322

Shilon O (2006) RandOrthMat.m: MATLAB code to generate a random $n \times n$ orthogonal real matrix. http://www.mathworks.com/matlabcentral/profile/authors/870484-ofek-shilon. Accessed 16 Dec 2016

Strang G (1988) Linear algebra and its applications. Harcourt Brace Jovanovich, San Diego