Enlarged Krylov Subspace Conjugate Gradient Methods for Reducing Communication
Laura Grigori, Sophie Moufawad, Frédéric Nataf

To cite this version:
Laura Grigori, Sophie Moufawad, Frédéric Nataf. Enlarged Krylov Subspace Conjugate Gradient Methods for Reducing Communication. [Research Report] RR-8597, INRIA. 2014. hal-01065985

HAL Id: hal-01065985
https://inria.hal.science/hal-01065985
Submitted on 18 Sep 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Enlarged Krylov Subspace Conjugate Gradient Methods for Reducing Communication

Laura Grigori, Sophie Moufawad, Frederic Nataf
Enlarged Krylov Subspace Conjugate Gradient Methods for Reducing Communication
rapport de recherche
Inria

Laura Grigori*, Sophie Moufawad*, Frederic Nataf*

Équipes-Projets
ALPINES


Résumé : Dans cet article, nous présentons deux nouvelles méthodes itératives pour la résolution des systèmes linéaires d’équations de très grande taille en minimisant les communications. Ces deux méthodes sont basées sur l’enrichissement de l’espace de Krylov en décomposant le domaine de $A$.

Mots-clés : algèbre linéaire, méthodes itératives

* INRIA
† UPMC
Enlarged Krylov Subspace Conjugate Gradient Methods for Reducing Communication

Abstract: In this paper we introduce a new approach for reducing communication in Krylov subspace methods that consists of enlarging the Krylov subspace by a maximum of \( t \) vectors per iteration, based on the domain decomposition of the graph of \( A \). The obtained enlarged Krylov subspace \( \mathcal{K}_{t,k}(A,r_0) \) is a superset of the Krylov subspace \( \mathcal{K}_k(A,r_0) \). Thus it is possible to search for the solution of the system \( Ax = b \) in \( \mathcal{K}_{t,k}(A,r_0) \) instead of \( \mathcal{K}_k(A,r_0) \). Moreover, we show in this paper that the enlarged Krylov projection subspace methods lead to faster convergence in terms of iterations and parallelizable algorithms with less communication, with respect to Krylov methods.

In this paper we focus on Conjugate Gradient (CG) [16], a Krylov projection method for symmetric (Hermitian) positive definite matrices. We discuss two new versions of Conjugate Gradient (section 3). The first method, multiple search direction with orthogonalization CG (MSDO-CG), is an adapted version of MSD-CG [14] with the A-orthonormalization of the search directions to obtain a projection method that guarantees convergence at least as fast as CG. The second projection method that we propose here, long recurrence enlarged CG (LRE-CG), is similar to GMRES in that we build an orthonormal basis for the enlarged Krylov subspace rather than finding search directions. Then, we use the whole basis to update the solution and the residual. Both methods converge faster than CG in terms of iterations, but LRE-CG converges faster than MSDO-CG since it uses the whole basis to update the solution rather than only \( t \) search directions. And the more subdomains are introduced or the larger \( t \) is, the faster is the convergence of both methods with respect to CG in terms of iterations. For example, for \( t = 64 \) the MSDO-CG and LRE-CG methods converge in 75\% up to 98\% less iteration with respect to CG for the different test matrices. But increasing \( t \) also means increasing the memory requirements. Thus, in practice, \( t \) should be relatively small, depending on the available memory, on the size of the matrix, and on the number of iterations needed for convergence, as explained in section 4. We also present the parallel algorithms along with their expected performance based on the estimated run times, and the preconditioned versions with their convergence behavior.

Key-words: minimizing communication, linear algebra, iterative methods
1 Introduction

Krylov subspace methods are among the most practical and popular iterative methods today. They are polynomial iterative methods that aim to solve systems of linear equations \((Ax = b)\) by finding a sequence of vectors \(x_1, x_2, x_3, x_4, \ldots, x_k\) that minimizes some measure of error over the corresponding spaces

\[ x_0 + K_i(A, r_0), \quad i = 1, \ldots, k \]

where \(x_0\) is the initial iterate, \(r_0\) is the initial residual, and \(K_i(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \ldots, A^{i-1}r_0\}\) is the Krylov subspace of dimension \(i\). Conjugate Gradient (CG) \([16]\), Generalized Minimal Residual (GMRES) \([25]\), bi-Conjugate Gradient \([19, 7]\), and bi-Conjugate Gradient Sabilized \([27]\) are some of the most used Krylov subspace methods.

These methods are governed by Blas1 and Blas2 operations as dot products and sparse matrix vector multiplications. Parallelizing dot products is constrained by communication since the performed computation is negligible. If the dot products are performed by one processor, then there is a need for a communication before and after the computation. In both cases, communication is a bottleneck. This problem has been tackled by different approaches. First, block methods that solve system with multiple right-hand sides \(AX = B\) were introduced, as block CG \([22]\). Then, s-step methods that compute \(s\) basis vectors per iteration were proposed, examples are s-step CG \([28, 2]\) and s-step GMRES \([29, 6]\). Both methods, block and s-step, use Blas2 and Blas3 operations. Recently, communication avoiding methods, based on s-step methods, that aim at avoiding communication at the expense of performing some redundant flops were introduced, as CA-CG, CA-GMRES \([21, 17]\) and CA-ILU0 preconditioner \([11]\). Another approach is to hide the cost of communication by overlapping it with other computation, like pipelined CG \([5, 13]\) and pipelined GMRES \([8]\).

In this paper we introduce a new approach that consists of enlarging the Krylov subspace by a maximum of \(t\) vectors per iteration. First, the input matrix is partitioned into \(t\) sub-domains by using a graph partitioning algorithm. At the beginning of the iterative method, the residual is split into \(t\) vectors corresponding to the \(t\) sub-domains. Then, the obtained \(t\) vectors are multiplied by \(A\) at each iteration to generate \(t\) new basis vectors. The obtained enlarged Krylov subspace \(K_{t,k}(A, r_0)\) is a superset of the Krylov subspace \(K_k(A, r_0)\). Thus it is possible to search for the solution of the system \(Ax = b\) in \(K_{t,k}(A, r_0)\) instead of \(K_k(A, r_0)\). Moreover, we show in this paper that the enlarged Krylov projection subspace methods lead to faster convergence in terms of iterations and parallelizable algorithms with less communication, with respect to Krylov methods.

In this paper we focus on Conjugate Gradient (CG) \([16]\), a Krylov projection method for symmetric (Hermitian) positive definite matrices, which was introduced by Hestenes and Stiefel in 1952 (section 2.4). After giving a brief overview of related existing CG methods (section 2) such as block-CG \([22]\), coop-CG \([1]\), and MSD-CG \([14]\), we discuss two new versions of Conjugate Gradient (section 3). The first method, multiple search direction with orthogonalization CG (MSDO-CG), is an adapted version of MSD-CG \([14]\). MSD-CG has the same structure as the classical conjugate gradient method where first \(t\) new search directions are defined on the \(t\) subdomains, then the \(t\) step lengths are obtained by solving a \(t \times t\) system, and finally the solution and the residual are updated. But unlike CG, the search directions are not A-orthogonal. Thus, in MSD-CG we A-orthonormalize the search directions, to obtain a projection method that guarantees convergence at least as fast as CG. The idea of using more than one search direction was also exploited in Rixen’s thesis \([23]\) for two subdomains in the context of domain decomposition methods, and further developed in \([10]\). The second method that we propose here, long recurrence enlarged CG (LRE-CG), is similar to GMRES in that we build an orthonormal basis for the enlarged Krylov subspace rather than finding search directions. Then, we use the whole basis to update the solution and the residual. We show that this method is a projection method and hence should converge at least as fast as CG. We compare the convergence behavior of both methods using different A-orthonormalization and orthonormalization methods and then we compare the most stable versions with CG and other related methods (section 4).
We have tested our methods on matrices arising from the discretization of 2D poisson equations (\textsc{Poisson2D}), 3D elasticity equations (\textsc{Elasticity3D}), and 2D and 3D convection-diffusion equations such as \textsc{Nit2D}, \textsc{Sky2D}, \textsc{Sky3D}, and \textsc{Ani3D} as discussed in section 4. Both methods converge faster than CG in terms of iterations, but LRE-CG converges faster than MSDO-CG since it uses the whole basis to update the solution rather than only $t$ search directions. And the more subdomains are introduced or the larger $t$ is, the faster is the convergence of both methods with respect to CG in terms of iterations. For example, for $t = 64$ the MSDO-CG and LRE-CG methods converge in 75\% to 82\% less iteration with respect to CG for the matrices \textsc{Nit2D}, \textsc{Poisson2D}, and \textsc{Elasticity3D}, and 95\% to 98\% less iteration with respect to CG for the matrices \textsc{Sky2D}, \textsc{Sky3D}, and \textsc{Ani3D}. But increasing $t$ also means increasing the memory requirements. Thus, in practice, $t$ should be relatively small, depending on the available memory, on the size of the matrix, and on the number of iterations needed for convergence, as explained in section 4. We present the parallel algorithms along with their expected performance based on the estimated run times in section 5. And in section 6 we introduce the preconditioned version with its convergence behavior.

2 Overview of Existing Conjugate Gradient (CG) Methods

The Krylov projection methods find a sequence of approximate solutions $x_k$ ($k > 0$) of the system $Ax = b$, and are defined by the following two conditions:

1. Subspace condition: $x_k \in x_0 + K_k(A,r_0)$

2. Petrov-Galerkin condition: $r_k \perp \mathcal{L}_k$

\[ \iff (r_k)^t y = 0, \quad \forall \ y \in \mathcal{L}_k \]

where $x_0$ is the initial iterate, $r_0$ is the initial residual, $K_k(A,r_0) = \text{span}\{r_0, Ar_0, A^2r_0, ..., A^{k-1}r_0\}$ is the Krylov subspace of dimension $k$, and $\mathcal{L}_k$ is a well-defined subspace of dimension $k$. The classical conjugate gradient is a Krylov projection method where $\mathcal{L}_k = K_k(A,r_0)$.

In this section we briefly introduce the conjugate gradient versions related to our MSDO-CG and LRE-CG versions, starting with the 1952 Hestenes and Stiefel version (section 2.1). Since then, many different versions of CG have been introduced (refer to [2] for a historical overview of CG till 1976). In 1980 O’Leary introduced a Block CG version [22] that solves a system with multiple right-hand sides $AX = B$ (section 2.2). The cooperative-CG [11] which was recently introduced, solves the system $AX = b$ by starting with $t$ distinct initial guesses. This is equivalent to solving the system $AX = b + \mathbb{1}$ (algorithmically very similar to Block CG) where $\mathbb{1}$ is a vector of ones of size $t$. The authors also present a parallel implementation that needs 2 to 3 synchronizations per iteration. This method has faster convergence than CG (section 2.3). The multiple search directions CG (MSD-CG) [14] solves $Ax = b$ by decomposing $A$’s domain into $t$ subdomains and defining a search direction on each of the $t$ subdomains. Then $x_k = x_{k-1} + P_k\alpha_k$, where $P_k$ is a matrix containing all the $t$ search directions and $\alpha_k$ is a vector of size $t$ (section 2.4). Unlike CG, block CG and coop CG, MSD-CG does not have the $A$-orthogonality condition of the search directions, i.e. $P_k^tAP_k$ is not equal to zero for all $i$ not equal to $k$. Hence it is not a projection method. This causes MSD-CG to have slower convergence than CG, and in some cases not to converge at all. That is why in multiple search directions with orthogonalization CG (MSDO-CG), after defining a search direction on each of the $t$ subdomains, we $A$-orthonormalize the search directions and this leads to better convergence than CG (section 5.2).

Note that in this paper we use matlab notation for matrices and vectors. For example, given a vector $p$ of size $n \times 1$ and a set of indices $\delta, p(\delta)$ is the vector formed by the subset of the entries of $p$ whose indices belong to $\delta$. For a matrix $A$, $A(\delta, \cdot)$ is a submatrix formed by the subset of the rows of $A$ whose indices belong to $\delta$. Similarly, $A(:, \alpha)$, is a submatrix formed by the subset of the columns of $A$ whose indices belong to $\alpha$. And $A(\alpha, \beta) = [A(\alpha, \cdot)](:, \beta)$, is formed by the $\beta$ columns of the submatrix $A(\alpha, \cdot)$.
2.1 Conjugate Gradient (CG) Method

Conjugate Gradient [16] is an iterative Krylov projection method for symmetric (Hermitian) positive definite (SPD) matrices of the form

\[
\begin{align*}
Ax &= b, \\
A &= A', \\
x^tAx &> 0, \forall x \neq 0.
\end{align*}
\]

Given an initial guess or iterate \(x_0\), at the \(k^{th}\) iteration CG finds the new approximate solution \(x_k = x_{k-1} + \alpha_k p_k\) that minimizes \(\phi(x) = \frac{1}{2} (x)^t A x - b^t x\) over the corresponding space \(x_0 + K_k(A, r_0)\), where \(k > 0\), \(p_k \in K_k(A, r_0)\) is the \(k^{th}\) search direction, and \(\alpha_k\) is the step along the search direction.

Theorem 2.1. The Petrov-Galerkin condition \((r_k)^t y = 0\), for all \(y \in K_k\) implies the A-orthogonality of the search directions \(p_i^t Ap_j = 0\), for all \(i \neq j\) and \(i, j \leq k\).

Proof. By definition, \(p_i \in K_i\) and \(K_i \subset K_{i+1}\). Thus \(p_i \in K_{i+c}\) for \(c \geq 0\). By the Petrov-Galerkin condition \(r_{k-1} p_i = 0\) and \(r_{i-1}^t p_i = 0\) for \(i \leq k - 1\). Thus, \(r_{k-1}^t p_i = r_{k-1}^t p_i - \alpha p_i^t A p_i = 0\) for \(i \leq k - 1\). This implies that \(p_i^t A p_i = 0\) for \(i \leq k - 1\) since \(\alpha \neq 0\). Therefore, the A-orthogonality of the search directions.

This theorem means that the A-orthogonality of the search directions has to be ensured or else the Petrov-Galerkin condition won’t be respected. On the other hand, the search direction \(p_k \in K_k\) is chosen according to the following recursion relation:

\[
\begin{align*}
p_1 &= r_0, \\
p_k &= r_{k-1} + \beta_k p_{k-1}
\end{align*}
\]

where \(p_1\) is set equal to \(r_0\) since the initial residual is equal to negative the gradient \(\triangledown \phi(x_0)\) which is the steepest descent from \(x_0\). But \(p_k\) is not set to \(r_{k-1}\), the steepest descent from \(x_{k-1}\) for \(k > 1\), since the residuals are not A-orthogonal. It can be shown that the search directions defined in (2) are A-orthogonal i.e. \(p_i^t A p_i = 0\) for all \(i \leq k - 1\). For \(i < k - 1\), we have

\[
p_i^t A p_i = r_{k-1}^t A p_i + \beta_i p_{k-1}^t A p_i = \beta_i p_{k-1}^t A p_i
\]

since \(r_{k-1} A p_i = 0\) by Petrov-Galerkin condition. In addition, \(r_{i-1}^t p_i = r_{i-2}^t p_i - \alpha p_{i-1}^t A p_i = 0\) with \(r_{k-2}^t p_i = 0\) since \(i \leq k - 2\). Thus, \(p_{k-1}^t A p_i = 0\) for \(i < k - 1\).

As for \(i = k - 1\), \(r_{k-1}^t A p_{k-1} \neq 0\) and \(p_{k-1}^t A p_{k-1} \neq 0\) for \(p_{k-1} \neq 0\). Thus, \(\beta_k = \frac{(r_{k-1})^t A p_{k-1}}{(p_{k-1})^t A p_{k-1}}\) is chosen so that \(p_{k} A p_{k-1} = 0\).

At each iteration, the step \(\alpha_k = \frac{(r_{k-1})^t A (r_{k-1})}{(p_k)^t A p_k}\) is chosen such that,

\[
\phi(x_k) = \min \{ \phi(x_{k-1} + \alpha p_k), \forall \alpha \in \mathbb{R} \}.
\]

Using the definition of \(\alpha_k, \beta_k = \frac{(r_{k-1})^t A p_{k-1}}{(p_{k-1})^t A p_{k-1}} = \frac{|r_{k-1}|^2}{|r_{k-2}|^2}.

RR n° 8597
2.2 Block Conjugate Gradient (B-CG) Method

In 1980 O’Leary introduced a Block CG version [22] that solves an SPD system with multiple right-hand sides.

\[ \begin{align*}
AX &= B, \\
A &= A^t, \\
x^tAx &> 0, \forall x \neq 0
\end{align*} \tag{4} \]

where \( A \) is an \( n \times n \) matrix, \( X \in \mathbb{R}^{n \times t} \) is a block vector, and \( B \) is a block vector of size \( n \times t \) containing the multiple right-hand sides.

Starting with an initial guess \( X_0 \in \mathbb{R}^{n \times t} \), initial residual \( R_0 = B - AX_0 \), and \( P_1 = R_0 \gamma_1 \), with \( \gamma_1 \) a \( t \times t \) full rank freely chosen matrix, the B-CG searches for an approximate solution \( X_{k+1} \in X_0 + \mathcal{K}_{k+1}(A, R_0) \) where \( \mathcal{K}_{k+1}(A, R_0) = \text{block span} \{ R_0, AR_0, A^2R_0, ..., A^kR_0 \} \) is the block Krylov subspace. Every \( n \times t \) block \( Z \in \mathcal{K}_{k+1}(A, R_0) \) is defined as \( Z = \sum_{i=1}^{k} A^i R_0 \zeta_i \), where \( \zeta_i \) is a \( t \times t \) matrix. By the Petrov-Galerkin condition we have that:

\[ R_{k+1} = R_k - AP_{k+1} \alpha_{k+1} \in \mathcal{K}_{k+2}(A, R_0) \]

where

\[ P_{k+1} = (P_{k+1}^t AP_{k+1})^{-1} \gamma_k (R_k^t R_k) \]

\[ \alpha_{k+1} = \gamma_k (R_k^t R_k) \]

\[ \beta_{k+2} = \gamma_k (R_k^t R_k) \]

Note that \( \alpha_{k+1} \) is chosen such that \( \phi(X_{k+1}) = \min \{ \phi(X_k + P_{k+1} \alpha) \} \) for all \( \alpha \in \mathbb{R}^{n \times t} \). As for \( \beta_{k+1} \), it is chosen to ensure the A-orthogonality of the \( P_{k+1} \) and \( P_k \). \( (P_{k+1}^t)^t AP_i = 0 \), whereas \( \gamma_k \) is a \( t \times t \) full rank matrix that can be chosen freely to decrease roundoff errors in the implementation. Moreover, the search direction \( P_{k+1} \in \mathcal{K}_{k+1}(A, R_0) \) of the block conjugate gradient method is A-conjugate, \( (P_{k+1})^t AY = 0 \), for all \( Y \in \mathcal{K}_k(A, R_0) \). This leads to the A-orthogonality of the search direction \( \implies (P_{k+1})^t AP_i = 0 \), for all \( i < k + 1 \). We present the Block-CG algorithm in Appendix A (Algorithm 2).

2.3 Cooperative Conjugate Gradient (coop-CG) Method

Recently, in 2012, Bhaya et al. presented a new version of conjugate gradient which is similar in structure to the Block conjugate gradient method. The coop-CG [1] solves the system \( AX = b \) by starting with \( t \) different initial guesses and solving the same system \( t \) times in parallel, where \( t \) threads/agents cooperate to find the solution. This is equivalent to solving the system \( AX = b * \text{ones}(1, t) \) where \( X_0 \) is a block-vector containing the \( t \) initial guesses, \( R_0 = AX_0 - b * 1 \) is the block residual, \( P_1 = R_0 \) is the initial block search direction. Then the derivations and the algorithm of the coop-CG (Algorithm ??, Appendix A) are the same as the Block-CG with \( \gamma_k = I \).

2.4 Multiple search direction Conjugate Gradient (MSD-CG) Method

The multiple search directions CG (MSD-CG), introduced by Gu et al. [14], solves the system \( AX=b \), and starts by having a decomposed domain and by defining at each iteration \( k \) a search direction \( p_i^k \) on each of the \( t \) subdomains \( (\delta_i, i = 1, 2, ..., t) \) such that \( p_i^k(\delta_j) = 0 \) for all \( j \neq i \). Then, the approximate solution at the \( k^{th} \) iteration is defined as \( x_k = x_{k-1} + P_k \alpha_k \) where \( P_k = [p_1^k, p_2^k, p_3^k, ..., p_t^k] \) is a matrix containing all the \( k^{th} \) search directions and \( \alpha_k \) is a vector of size \( t \).
Given an initial guess $x_0$, the residual is defined as $r_k = b - Ax_k$ for $k \geq 0$. The first set of domain search directions is defined by the initial residual $r_0$, such that $p_1^k(\delta_i) = r_0(\delta_i)$ for $i = 1, 2, \ldots, t$ and zero otherwise. Then, for $k > 1$ the domain search directions are defined as follows, $p_i^k = T_i(r_{k-1}) + \beta^k_i p_{i-1}^{k-1}$ for $i = 1, 2, \ldots, t$ where $\beta_i^k$ is a scalar and $T_i$ is an operator that projects a vector onto the subdomain $\delta_i$ ($[T_i(x)](\delta_j) = 0$ for $j \neq i$ and $[T_i(x)](\delta_i) = x(\delta_i)$). The search directions block has the following sparsity pattern for all $k$,

$$
P_k = \begin{pmatrix}
0 & \cdots & 0 & \cdots & 0 \\
0 & \cdots & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & \cdots & 0 \\
0 & \cdots & 0 & \cdots & 0 
\end{pmatrix}_{n \times t}.
$$

As for $\alpha_k = (P_k^tAP_k)^{-1}P_k^tr_{k-1}$, it is chosen such that it minimizes $\phi(x_k) = \min\phi(x_{k-1} + P_k\alpha), \forall \alpha \in \mathbb{R}^t$. Unlike CG, block CG and coop CG, MSD-CG does not have the A-orthogonality condition of the search directions, i.e. $P_k^tAP_k$ is not equal to zero for all $i$ not equal to $k$. Thus $\beta_k^i = (P_{k-1}^tAP_{k-1})^{-1}P_{k-1}^tP_k$ is chosen so that the global search direction $p_k^i = \sum_{i=1}^{t} p_k^i$ is A-orthogonal to the previous domain search direction $p_i^{k-1}$, i.e. $(p_k^i)^tAP_{k-1} = 0$, for $i = 1, 2, \ldots, t$. As for the convergence, it is shown that the rate of convergence of MSD-CG is at least as fast as that of the steepest descent method. Yet, steepest descent is known for its slow “zig-zagging” convergence. This causes the MSD-CG to have slower convergence than CG, and in some cases it does not converge at all with respect to the given stopping criteria as shown in section 4.

3 The New Conjugate Gradients

We will introduce two new conjugate gradient methods, MSDO-CG and LRE-CG, which are based on replacing the Krylov subspace $K_k$ with a larger subspace leading to better convergence. Thus we will first introduce the new enlarged Krylov subspace and its properties in the context of conjugate gradient methods in section 3.1. Then in section 3.2 and section 3.3, we introduce the multiple search direction conjugate gradient with orthogonalization (MSDO-CG) and the long recurrence enlarged conjugate gradient (LRE-CG).

As previously mentioned, MSDO-CG is an adapted version of MSD-CG, where the $t$ newly defined search directions are A-orthonormalized against previous search directions and against each others. This A-orthonormalization guarantees a convergence behavior at least as good as CG. As for the LRE-CG, at each iteration, $t$ new basis vectors are computed for the enlarged Krylov subspace. Then, rather than having short recurrences, $x_k$ is defined by all the basis vectors as in GMRES, where the basis vectors are orthonormalized. Both methods, MSD-CG and LRE-CG, require saving at most $tk$ vectors versus one search direction in CG. Yet LRE-CG converges faster than MSDO-CG (section 4) at the expense of solving growing systems of size $tk$. Several remedies to this problem are discussed in section 3.3.

3.1 The Enlarged Krylov Subspace

The enlarged Krylov subspace and methods are based on a partition of the unknowns, or alternatively the rows of the $n \times n$ matrix $A$. Assume that the index domain $\delta = \{1, 2, \ldots, n\}$ is divided into $t$ distinct
Theorem 3.3. In theorem 3.3, we do not use the direct sum $\oplus$ or $\otimes$ used in section 3.3. Theorem 3.3 is that we can search for an approximate solution $x_0$ for the block containing the $t$ vectors obtained from $T(r_0)$. Note that $R_0 = \{ T_1(r_0), T_2(r_0), \ldots, T_t(r_0) \}$ is a set of vectors. But $R_0 = [ T_1(r_0) T_2(r_0) \ldots T_t(r_0) ]$, where the brackets denote a matrix format.

**Definition 3.1.** Let
\[
\mathcal{K}_{t,k} = \text{span}(T(r_0), A^T(r_0), A^2T(r_0), \ldots, A^{k-1}T(r_0))
\]
\[
= \text{span} \{ T_1(r_0), T_2(r_0), \ldots, T_t(r_0), AT_1(r_0), AT_2(r_0), \ldots, AT_t(r_0), \ldots, A^{k-1}T_1(r_0), \ldots, A^{k-1}T_t(r_0) \}
\]
be an enlarged Krylov subspace of dimension $k \leq z \leq tk$ generated by the matrix $A$ and the vector $r_0$, and associated to a given partition defined by $\delta_i$ for $i = 1, 2, \ldots, t$.

The enlarged Krylov subspaces $\mathcal{K}_{t,k}(A, r_0)$ are increasing subspaces, yet bounded. We denote by $k_{\text{max}}$ the upper bound $k$ for which the dimension of the enlarged Krylov subspace $\mathcal{K}_{t,k}(A, r_0)$ stops increasing. For simplicity, we will denote the enlarged Krylov subspace generated by $A$ and $r_0$, $\mathcal{K}_{t,k}(A, r_0)$, by $\mathcal{K}_{t,k}$, and the Krylov subspace generated by $A$ and $r_0$, $\mathcal{K}_{k}(A, r_0)$ by $\mathcal{K}_k$.

**Theorem 3.2.** The Krylov subspace $\mathcal{K}_k$ is a subset of the enlarged Krylov subspace $\mathcal{K}_{t,k}$ ($K_k \subset \mathcal{K}_{t,k}$).

**Proof.** Let $y \in \mathcal{K}_k$ where $\mathcal{K}_k = \text{span}\{ r_0, Ar_0, \ldots, A^{k-1}r_0 \}$. Then
\[
y = \sum_{j=0}^{k-1} a_j A^j r_0 = \sum_{j=0}^{k-1} a_j A^j R_0 \ast \mathbb{1}_t = \sum_{j=0}^{k-1} \sum_{i=1}^{t} a_j A^j T_i(r_0) \in \mathcal{K}_{t,k}
\]
since $r_0 = R_0 \ast \mathbb{1}_t = [ T_1(r_0) T_2(r_0) \ldots T_t(r_0) ] \ast \mathbb{1}_t$. \hfill \Box

Krylov subspace methods search for an approximate solution $x_0 \in x_0 + \mathcal{K}_k$. A corollary of theorem 3.2 is that we can search for an approximate solution $x_0$ in $x_0 + \mathcal{K}_{t,k}$ instead, since $\mathcal{K}_k \subset \mathcal{K}_{t,k}$.

In theorem 3.3, we do not use the direct sum $\oplus$ since it is not guaranteed that the intersection of the two subspaces, $\mathcal{K}_{t,k}$ and $\text{span}\{ A^kT_1(r_0), A^kT_2(r_0), \ldots, A^kT_t(r_0) \}$, is empty.

**Theorem 3.3.** By definition 3.1, the enlarged Krylov subspace
\[
\mathcal{K}_{t,k+1} = \mathcal{K}_{t,k} + \text{span}\{ A^kT_1(r_0), A^kT_2(r_0), \ldots, A^kT_t(r_0) \}
\]
If $A^kT_v(r_0) \in \mathcal{K}_{t,k}$ for all $1 \leq v \leq t$, then $A^{k+q}T_i(r_0) \in \mathcal{K}_{t,k}$ for some $1 \leq i \leq t$ and for some $q > 0$.

**Proof.** We prove this by induction.

**Base Case:**
Given that $A^kT_v(r_0) \in \mathcal{K}_{t,k}$ for all $1 \leq v \leq t$, we show that $A^{k+1}T_i(r_0) \in \mathcal{K}_{t,k}$, where $1 \leq i \leq t$. If $A^kT_i(r_0) = \sum_{u=0}^{k-1} \sum_{v=1}^{t} a_{u,v} A^u T_v(r_0)$ since $A^kT_i(r_0) \in \mathcal{K}_{t,k}$, then
\[
A^{k+1}T_i(r_0) = \sum_{u=0}^{k-1} \sum_{v=1}^{t} a_{u,v} A^{u+1} T_v(r_0) = \sum_{u=0}^{k-2} \sum_{v=1}^{t} a_{u,v} A^{u+1} T_v(r_0) + \sum_{v=1}^{t} a_{k-1,v} A^k T_v(r_0)
\]
\[
= \sum_{u=0}^{k-2} \sum_{v=1}^{t} a_{u,v} A^{u+1} T_v(r_0) + \sum_{v=1}^{t} a_{k-1,v} \left( \sum_{u=0}^{k-1} \sum_{y=1}^{t} \beta_{u,y} A^u T_y(r_0) \right)
\]
\[
= \sum_{u=0}^{k-1} \sum_{v=1}^{t} \gamma_{u,v} A^u T_v(r_0) \in \mathcal{K}_{t,k}
\]
Assume true for $q$
Assume that $A^{k+q}T_i(r_0) \in K_{t,k}$ where $1 \leq i \leq t$, that is $A^{k+q}T_i(r_0) = \sum_{u=0}^{k-1} \sum_{v=1}^{t} \alpha_{u,v}A^uT_v(r_0)$
Prove true for $q+1$
Show that $A^{k+q+1}T_i(r_0) \in K_{t,k}$
\[
A^{k+q+1}T_i(r_0) = \sum_{u=0}^{k-1} \sum_{v=1}^{t} \alpha_{u,v}A^{u+1}T_v(r_0) = \sum_{u=0}^{k-2} \sum_{v=1}^{t} \alpha_{u,v}A^{u+1}T_v(r_0) + \sum_{v=1}^{t} \alpha_{k-1,v}A^kT_v(r_0)
\]
\[
= \sum_{u=0}^{k-2} \sum_{v=1}^{t} \alpha_{u,v}A^{u+1}T_v(r_0) + \sum_{u=0}^{k-1} \sum_{y=1}^{t} \beta_{u,y}A^uT_y(r_0)
\]
\[
= \sum_{u=0}^{k-1} \sum_{v=1}^{t} \gamma_{u,v}A^uT_v(r_0) \in K_{t,k}
\]

Given that $K_{t,k} \neq K_{t,k-1}$, then a corollary of Theorem 3.3 is that $K_{t,k} = K_{t,k+q}$ for all $q > 0$, where $k_{\max} = k$ is the upper bound for which the dimension of the enlarged Krylov subspace stops increasing. Assume that $A^{1}T_1(r_0) \in K_{t,k}$ for all $1 \leq i \leq t$, then by Theorem 3.3 $A^{k+q}T_i(r_0) \in K_{t,k}$ for all $q > 0$ and for some $1 \leq i \leq t$. Then for all $1 \leq i \leq t$ and for all $q > 0$, $A^{k+q}T_i(r_0) \in K_{t,k}$. Thus no new vector is added to the basis of $K_{t,k+q}$ for all $q > 0$ and $K_{t,k} = K_{t,k+q}$. Moreover, since $K_{t,k} \neq K_{t,k-1}$ then $k_{\max} = k$, the upper bound for which the dimension of enlarged Krylov subspace $K_{t,k}(A, r_0)$ stops increasing.

**Theorem 3.4.** If $A^{k}T_i(r_0) \in K_{t,k} + \text{span}\{A^{k}T_1(r_0), \ldots, A^{k}T_{i-1}(r_0), A^{k}T_{i+1}(r_0), \ldots, A^{k}T_t(r_0)\}$, then $A^{k+q}T_i(r_0) \in K_{t,k+q} + \text{span}\{A^{k+q}T_1(r_0), \ldots, A^{k+q}T_{i-1}(r_0), A^{k+q}T_{i+1}(r_0), \ldots, A^{k+q}T_t(r_0)\}$ for all $1 \leq i \leq t$ and $q > 0$.

**Proof.** If $A^{k}T_i(r_0) \in K_{t,k} + \text{span}\{A^{k}T_1(r_0), \ldots, A^{k}T_{i-1}(r_0), A^{k}T_{i+1}(r_0), \ldots, A^{k}T_t(r_0)\}$, then $A^{k}T_i(r_0) = \sum_{u=0}^{k-1} \sum_{v=1}^{t} \alpha_{u,v}A^uT_v(r_0) + \sum_{v=1}^{t} \alpha_{k,v}A^kT_v(r_0)$. Thus,
\[
A^{k+q}T_i(r_0) = \sum_{u=0}^{k-1} \sum_{v=1}^{t} \alpha_{u,v}A^{u+q}T_v(r_0) + \sum_{v=1}^{t} \alpha_{k,v}A^{k+q}T_v(r_0)
\]
\[
\in K_{t,k+q} + \text{span}\{A^{k+q}T_1(r_0), \ldots, A^{k+q}T_{i-1}(r_0), A^{k+q}T_{i+1}(r_0), \ldots, A^{k+q}T_t(r_0)\}
\]

A corollary of Theorem 3.4 is that if $t-i_k$ vectors of the form $A^yT_y(r_0)$ with $y = i_k + 1, \ldots, t$ belong to the subspace $K_{t,k} + \text{span}\{A^yT_1(r_0), A^yT_2(r_0), \ldots, A^yT_{i_k}(r_0)\}$, then the $t-i_k$ vectors of the form $A^{k+q}T_y(r_0)$ belong to the subspace $K_{t,k+q} + \text{span}\{A^{k+q}T_1(r_0), A^{k+q}T_2(r_0), \ldots, A^{k+q}T_{i_k}(r_0)\}$.

**Theorem 3.5.** Let $k_{\max}$ be the smallest integer such that $K_{t,k_{\max}} = K_{t,k_{\max}+q}$ for all $q > 0$. Then, for all $k < k_{\max}$ the dimension of the enlarged Krylov subspaces $K_{t,k}$ and $K_{t,k+1}$ is strictly increasing by some number $i_k$ and $i_{k+1}$ respectively, where $1 \leq i_{k+1} \leq i_k \leq t$.

**Proof.** By definition of $k_{\max}$, we have that for all $q > 0$
\[
K_{t,1} \subseteq \ldots \subseteq K_{t,k_{\max}+1} \subseteq K_{t,k_{\max}+q} = K_{t,k_{\max}+q}
\]
Then for all $k < k_{\max}$, the dimension of the enlarged Krylov subspaces $K_{t,k}$ is strictly increasing by some number $i_k \neq 0$ with respect to the dimension of $K_{t,k-1}$.

If the $t$ new vectors are linearly independent and none of them belongs to $K_{t,k-1}$, then the $t$ vectors are added to the basis of $K_{t,k}$ and $\text{dim}(K_{t,k}) = \text{dim}(K_{t,k-1}) + t$, where $i_k = t$ and $\text{dim}(K)$ is the dimension of a subspace. In case the $t$ new vectors are linearly dependent and none of them belongs to $K_{t,k-1}$, then only one vector is added to the basis of $K_{t,k}$ and $\text{dim}(K_{t,k}) = \text{dim}(K_{t,k-1}) + 1$, that
is \( i_k = 1 \). There are many other cases where \( 1 < t - i_k < t \) of the \( t \) vectors belong to \( \mathcal{K}_{t,k-1} \) or are linearly dependent on the other \( i_k \) vectors and \( \mathcal{K}_{t,k-1} \). Then \( i_k \) vectors are added to the basis of \( \mathcal{K}_{t,k} \) and \( \dim(\mathcal{K}_{t,k}) = (\dim(\mathcal{K}_{t,k-1}) + i_k \), where \( 1 < i_k < t \).

In general, \( \dim(\mathcal{K}_{t,k}) = \dim(\mathcal{K}_{t,k-1}) + i_k \), where \( 1 \leq i_k \leq t \). Similarly, \( \dim(\mathcal{K}_{t,k+1}) = \dim(\mathcal{K}_{t,k}) + i_{k+1} \), where \( 1 \leq i_{k+1} \leq t \). Moreover, in \( \mathcal{K}_{t,k} \)'s basis we added \( i_k \) new vectors of the form \( A^{k-1}T_i(r_0) \) while the other \( t - i_k \) either belong to \( \mathcal{K}_{t,k-1} \) or are linearly dependent on the \( i_k \) vectors and \( \mathcal{K}_{t,k-1} \). In both cases, the \( t - i_k \) vectors of the form \( A^{k-1}T_i(r_0) \) belong to the subspace \( \mathcal{K}_{t,k-1} + \text{span}\{A^{k-1}T_1(r_0), \ldots, A^{k-1}T_{i_k}(r_0)\} \). Then by Theorem 3.4 and its corollary, the \( t - i_k \) vectors of the form \( A^{t_0}T_i(r_0) \) belong to the subspace \( \mathcal{K}_{t,k+q} + \text{span}\{A^{t_0}T_1(r_0), A^{t_0}T_2(r_0), \ldots, A^{t_0}T_{i_k}(r_0)\} \) for \( q > 0 \). Therefore, we have at least \( t - i_k \) linearly dependent vectors added to \( \mathcal{K}_{t,k+1} \), hence \( i_{k+1} \) can never be greater than \( i_k \).

**Theorem 3.6.** Let \( p_{\max} \) and \( k_{\max} \) be such that \( \mathcal{K}_{p_{\max}} = \mathcal{K}_{p_{\max}+q} \) and \( \mathcal{K}_{t,k_{\max}} = \mathcal{K}_{t,k_{\max}+q} \) for \( q > 0 \). Then \( k_{\max} \leq p_{\max} \).

**Proof.** Let \( \mathcal{K}_{p_{\max}} = \mathcal{K}_{p_{\max}+q} \) and \( A^{p_{\max}+q-1}r_0 \in \mathcal{K}_{p_{\max}+q} \) where \( q > 0 \). Then \( \sum_{j=1}^{p_{\max}} \alpha_j A^{j-1}T_1(r_0) = \sum_{j=1}^{p_{\max}} \sum_{i=1}^{t} \alpha_j A^{j-1}T_i(r_0) = \sum_{j=1}^{p_{\max}+q} \sum_{i=1}^{t} \alpha_j A^{j-1}T_i(r_0) \).

Suppose that \( A^{p_{\max}+q-1}T_1(r_0) \notin \mathcal{K}_{p_{\max}} \) for all \( 1 \leq i \leq t \). Then \( \sum_{j=1}^{p_{\max}+q} \sum_{i=1}^{t} \alpha_j A^{j-1}T_i(r_0) \). We may assume that there exists at least one \( \alpha_{j,i} \neq 0 \) for \( j > p_{\max} \), then this leads to a contradiction. This implies that \( \mathcal{K}_{p_{\max}+q-1}T_1(r_0) \notin \mathcal{K}_{p_{\max}} \) for all \( 1 \leq i \leq t \).

Thus by definition of the \( T() \) operator and since \( \mathcal{K}_p \) is a subset of \( \mathcal{K}_{t,p} \), if \( k_{p_{\max}} = k_{p_{\max}+q} \), then \( \mathcal{K}_{p_{\max}} = \mathcal{K}_{t,p_{\max}} \). However, if \( k_{t,k_{\max}} = k_{t,k_{\max}+q} \) this does not imply that \( k_{p_{\max}} = k_{p_{\max}+q} \). Therefore, since \( \mathcal{K}_{t,k} \) is a much larger subspace than \( \mathcal{K}_k \), it is possible to reach stagnation earlier. Therefore \( k_{\max} \leq p_{\max} \).

**Theorem 3.7.** The solution of the system \( Ax = b \) belongs to the subspace \( x_0 + \mathcal{K}_{t,k_{\max}} \), where \( \mathcal{K}_{t,k_{\max}+q} = \mathcal{K}_{t,k_{\max}} \) for \( q > 0 \).

**Proof.** The solution \( x_{sol} \in x_0 + \mathcal{K}_{p_{\max}} \), where \( \mathcal{K}_{p_{\max}} = \text{span}\{r_0, A_1r_0, \ldots, A^{p_{\max}-1}r_0\} \) and \( \mathcal{K}_{p_{\max}} = \mathcal{K}_{p_{\max}+q} \) for \( q > 0 \). Since \( \mathcal{K}_{p_{\max}} \subseteq \mathcal{K}_{t,p_{\max}} \), the solution \( x_{sol} \in x_0 + \mathcal{K}_{t,p_{\max}} \), where \( p_{\max} \geq p_{\max} \) by Theorem 3.6.

Suppose that \( x_{sol} \in x_0 + \mathcal{K}_{t,p_{\max}} \), but \( x_{sol} \notin x_0 + \mathcal{K}_{t,k_{\max}} \). This implies that \( \mathcal{K}_{t,k_{\max}} \notin \mathcal{K}_{t,p_{\max}} \). However, by definition of \( k_{\max} \) and since \( k_{\max} \leq p_{\max} \), we have that \( \mathcal{K}_{t,k_{\max}} \notin \mathcal{K}_{t,p_{\max}} \). This is a contradiction.

### 3.1.1 Krylov projection methods

The Krylov projection methods find a sequence of approximate solutions \( x_k (k > 0) \) of the system \( Ax = b \) from the subspace \( x_0 + \mathcal{K}_k \subseteq \mathbb{R}^n \) (or \( \subseteq \mathbb{C}^n \)) by imposing the Petrov-Galerkin constraint on the \( k \)th residual \( r_k = b - Ax_k \), that is \( r_k \) is orthogonal to some well-defined subspace of dimension \( k \).

We define our new enlarged Krylov projection methods based on CG by the subspace \( \mathcal{K}_{t,k} \) and the following two conditions:

1. Subspace condition: \( x_k \in x_0 + \mathcal{K}_{t,k} \)
2. Orthogonality condition: \( r_k \perp \mathcal{K}_{t,k} \) \[ \iff (r_k)^T y = 0, \text{ for all } y \in \mathcal{K}_{t,k} \]

where \( \mathcal{K}_{t,k} \) is a well-defined subspace of dimension \( k \leq z \leq tk \).
3.1.2 The minimization property

The new enlarged CG methods find the new approximate solution by minimizing the function \( \phi(x) \) over the subspace \( x_0 + K_{t,k} \).

**Theorem 3.8.** If \( r_k \perp K_{t,k} \), then \( \phi(x_k) = \min \{ \phi(x), \forall x \in x_0 + K_{t,k} \} \).

**Proof.** By the Petrov-Galerkin condition we have that \( r_k \perp K_{t,k} \)

\[
(r_k)^T y = 0, \quad \forall y \in K_{t,k}
\]

\[
(b - Ax_k)^T y = 0, \quad \forall y \in K_{t,k}
\]

\[
b^T y - (x_k)^T Ay = 0, \quad \forall y \in K_{t,k}
\]

Let \( y = x_k - x_0 \in K_{t,k} \)

\[
\Rightarrow (x_k)^T (x_k - x_0) - b^T (x_k - x_0) = 0
\]

\[
\Rightarrow (x_k)^T A x_k - b^T x_k = (x_k)^T A x_0 - b^T x_0
\]

\[
\Rightarrow \phi(x_k) = \frac{1}{2} (x_k)^T A x_k - b^T x_k = -\frac{1}{2} (x_k)^T A x_k + (x_k)^T A x_0 - b^T x_0
\]

By showing that \( \phi(x) \geq \phi(x_k) \), for all \( x \in x_0 + K_{t,k} \), then we have proven that \( \phi(x_k) = \min \{ \phi(x), \forall x \in x_0 + K_k \} \).

\[
\phi(x) - \phi(x_k) = \frac{1}{2} x^T A x - b^T x - \left[ \frac{1}{2} (x_k)^T A x_k + (x_k)^T A x_0 - b^T x_0 \right]
\]

\[
= \frac{1}{2} x^T A x - b^T x + \frac{1}{2} (x_k)^T A x_k - (x_k)^T A x_0 + (x_k)^T A x_0 - b^T x_0
\]

\[
= \frac{1}{2} x^T A x - (x_k)^T A x + \frac{1}{2} (x_k)^T A x_k - (x_k)^T A x_0, \quad \text{since} \; b^T z = (x_k)^T A z
\]

\[
= \frac{1}{2} x^T A x - (x_k)^T A x + \frac{1}{2} (x_k)^T A x_k
\]

\[
\geq 0, \quad \text{since} \; A \; \text{is positive definite}
\]

**Theorem 3.9.** \( \phi(x_k) = \min \{ \phi(x), \forall x \in x_0 + K_{t,k} \} \) if and only if \( ||x^* - x_k||_A = \min \{ ||x^* - x||_A, \forall x \in x_0 + K_{t,k} \} \), where \( x^* \) is the exact solution of \([1]\).

**Proof.** \( f(x) = ||x^* - x||_A = (x^*)^T A x^* - 2 (x^*)^T A x + x^T A x = b^T x^* - 2 b^T x + x^T A x = b^T x^* + 2 \phi(x) \).

The minimum of \( f(x) \) is given by \( f'(x) = \nabla \phi(x) = 0 \).

3.1.3 Convergence analysis

The conjugate gradient method of Hestenes and Stiefel is known to converge in \( K \) iterations where \( K \leq n \), if the matrix \( A \in \mathbb{R}^{n \times n} \) is SPD. Moreover, the \( k^{th} \) error of CG \( \tau_k = ||x^* - x_k|| \leq 2 \left( \frac{\sqrt{2} - 1}{\sqrt{2} + 1} \right)^k ||x_0||_A \)

where \( K = ||A||_2 ||A^{-1}||_2 = \frac{\lambda_{\max}}{\lambda_{\min}} \) is the L2-condition number of the matrix A.

Assuming that the \( k^{th} \) residual of the new conjugate gradient methods \( r_k \perp K_{t,k} \), then by Theorem 3.8 and Theorem 3.9 we have that

\[
||x_k||_A = ||x^* - x_k||_A = \min \{ ||x^* - x||_A, \forall x \in x_0 + K_{t,k} \} \quad \text{(6)}
\]

\[
\leq \min \{ ||x^* - \tau||_A, \forall \tau \in x_0 + K_k \} \quad \text{since} \; K_k \subset K_{t,k} \quad \text{(7)}
\]

\[
\leq ||\tau_k||_A \quad \text{(8)}
\]

Therefore, our methods converge at least as fast as the Classical Conjugate Gradient method, assuming that the Petrov Galerikene condition is respected \( (r_k \perp K_{t,k}) \). Hence, the enlarged Krylov subspace CG methods will converge in \( K \) iterations, where \( K \leq K \leq n \).
3.2 Multiple search direction with orthogonalization Conjugate Gradient (MSDO-CG) Method

The MSD-CG method introduced in Gu et al. [14], can be viewed as an enlarged Krylov method where \( P_0 = [T(r_0)] \), and at the \( k^{th} \) iteration \( p^k_i = T_i(r_{k-1}) + \beta^k_i p^{k-1}_i \) for \( i = 1, 2, ..., t \), \( P_k = [p^1_k, p^2_k, ..., p^t_k] \), \( x_k = x_{k-1} + P_k \alpha_k \) and \( r_k = r_{k-1} - A P_k \alpha_k \) with \( \alpha_k = (P^T_k A P_k)^{-1} P^T_k r_{k-1} \) and \( \beta^k = [\beta^1_k, \beta^2_k, ..., \beta^t_k] = (P^T_{k-1} A P_{k-1})^{-1} P^T_{k-1} A r_{k-1} \). However, the \( P_k \)'s are not A-orthogonal implying that \( r_k \notin \mathcal{K}_{t,k} \). Thus, MSD-CG is not a projection method.

The multiple search directions with orthogonalization CG (MSDO-CG) is an enlarged Krylov projection method that solves the system \( (Ax = b) \), by approximating the solution at the \( k^{th} \) iteration with the vector \( x_k = x_{k-1} + P_k \alpha_k \) such that
\[
\phi(x_k) = \min_{\alpha} \{ \phi(x), \forall x \in \mathcal{K}_{t,k} \},
\]
where \( P_k \alpha_k \in \mathcal{K}_{t,k}, P_k \) is an \( n \times t \) block vector containing the \( t \) subdomain search directions, and \( \alpha_k \) is a vector of size \( t \).

The minimum of \( \phi(x) \) is given by \( \nabla \phi(x) = 0 \), which is equivalent to \( Ax - b = 0 \). Thus, by minimizing \( \phi(x) \), we are solving the system \( (1) \). Note that since \( \phi(x_k) = \min_{\alpha} \{ \phi(x), \forall x \in x_0 + \mathcal{K}_{t,k} \} \), then
\[
\phi(x_k) = \phi(x_{k-1} + P_k \alpha_k) = \min_{\alpha} \{ \phi(x_{k-1} + P_k \alpha_k), \forall \alpha \in \mathbb{R}^t \}.
\]
Once \( x_k \) has been chosen, either \( x_{k-1} \) is the desired solution of \( Ax = b \), or \( t \) new domain search direction vectors \( P_{k+1} \) and a new approximation \( x_{k+1} = x_k + P_{k+1} \alpha_{k+1} \) are computed. Similarly to MSD-CG, \( P_{k+1} = [p^1_{k+1}, p^2_{k+1}, ..., p^t_{k+1}] \), where \( p^1_k = T_i(r_0) \) and \( p^i_{k+1} = T_i(r_k) + \beta^i_{k+1} p^i_k \) for \( i = 1, 2, ..., t \). But unlike MSD-CG, MSDO-CG is a projection method. Hence, we A-orthonormalize all the search directions, \( P_{k+1} \), to ensure that \( \mathbf{r}_{k+1} \perp \mathcal{K}_{t,k+1} \) as discussed in section 3.2.2. By imposing the orthogonality condition, \( \mathbf{r}_{k+1} \perp \mathcal{K}_{t,k+1} \), it is guaranteed that MSDO-CG converges at least as fast as CG as proven in section 3.2.3.

This procedure is repeated until convergence. Thus, we need to find the recursion relations of \( r_k, P_k, \alpha_k, \) and \( \beta_k = [\beta^1_k, \beta^2_k, ..., \beta^t_k] \).

3.2.1 The residual \( r_k \)

By definition, the residual \( r_k = b - Ax_k \), where \( x_k \in \mathcal{K}_{t,k} \). Thus \( r_k \in \mathcal{K}_{t,k+1} \). As for the recursion relation of \( r_k \), we simply replace \( x_k \) by its expression and obtain the following:
\[
r_k = b - Ax_k = b - A(x_{k-1} + P_k \alpha_k) = r_{k-1} - A P_k \alpha_k.
\]
Moreover, if the orthogonality condition, \( r_k \perp \mathcal{K}_{t,k} \), is ensured, then \( (r_k)^T r_i = 0 \), for all \( i < k \). Hence, the residuals form an orthogonal set.

**Theorem 3.10.** The orthogonality condition \( (r_i)^T y = 0 \) for all \( y \in \mathcal{K}_{t,k} \), implies the A-orthogonality of the block search directions \( P_k P_j^T A P_j = 0 \), for all \( i \neq j, \) and \( i, j \leq k \).

**Proof.** By definition, \( P_i \in \mathcal{K}_{t,i} \) and \( \mathcal{K}_{t,i} \subset \mathcal{K}_{t,i+1} \). Thus \( P_i \in \mathcal{K}_{t,i+c} \) for \( c \geq 0 \). By the Petrov-Galerkin condition \( r_{i-k} P_i = 0 \) for \( i \leq k - 1 \) and \( r_{k+1} P_i = 0 \). Thus, \( r_{k+1} P_i = r_{k+1} P_i - \alpha_k P_k^T A P_i = 0 \) for \( i \leq k - 1 \). This implies that \( P_k^T A P_i = 0 \) for \( i \leq k - 1 \) since \( \alpha \neq 0 \). Therefore, the A-orthogonality of the search directions.

3.2.2 The domain search direction \( P_k \)

By definition, the domain search direction is \( P_k = [p^1_k, p^2_k, ..., p^t_k] \) where \( p^1_i = T_i(r_0) \) and \( p^i_k = T_i(r_{k-1}) + \beta^i_{k-1} p^{i-1}_k \) for \( i = 1, 2, ..., t \).

Inria
The recursion relation of $P_k$ is defined as follows

$$P_k = T(r_{k-1}) + P_{k-1} \text{diag}(\beta_k),$$  

where $\text{diag}(\beta_k)$ is a $t \times t$ matrix with the vector $\beta_k$ on the diagonal.

The domain search directions defined in (10) are not $\alpha$-orthogonal to each others. To ensure that the orthogonality condition is valid, at each iteration $k$ the block vector $P_k$ is $\alpha$-orthonormalized against all the previous $P_i$, where $i = 1, 2, \ldots, k - 1$. Then the column vectors of $P_k$ are $\alpha$-orthonormalized against each others. Thus, the obtained search directions $\hat{P}_k$ satisfy $(\hat{P}_k)^T A \hat{P}_k = 0$ for all $i \neq k$. Moreover, $(\hat{P}_k)^T A \hat{P}_k = I$, where $I$ is the identity matrix, assuming that the column vectors of $P_k$ are linearly independent with respect to each others and the previous directions, or alternatively none of the column vectors of $\hat{P}_k$ is zero. Note that, once $P_k = T(r_{k-1}) + P_{k-1} \text{diag}(\beta_k)$ is defined, it is directly $\alpha$-orthonormalized. Thus, in the sections that follow, we denote by $P_k$ the $\alpha$-orthonormalized search directions and we do not use the $\hat{P}_k$ notation to be consistent with the initial definitions in the previous sections.

There are several $\alpha$-orthonormalization methods. First, for $\alpha$-orthonormalizing $P_k$ against all the previous $P_i$, where $i = 1, 2, \ldots, k - 1$, one can use classical Gram Schmidt (CGS), modified Gram Schmidt (MGS), or classical Gram Schmidt with reorthogonalization (CGS2) where we apply the CGS algorithm twice for numerical stability reasons. As for $\alpha$-orthonormalizing $\hat{P}_k$, there are many methods that are discussed in [20] [24], but we will only refer to CGS, CGS2, MGS, A-CholQR and Pre-CholeQR. We seek a combination of both $\alpha$-orthonormalizations that is stable and parallelizable with reduced communication. For that reason, in section 4 we test the MSDO-CG method with the different combinations of the $\alpha$-orthonormalization methods and we conclude that the MSDO-CG is numerically most stable when we use MGS, CGS2+A-CholQR, or CGS2+Pre-CholQR. In section 5.1 we discuss the parallelization of the MSDO-CG algorithm with the stable $\alpha$-orthonormalization methods.

Note that in Appendix B, we discuss the $\alpha$-orthonormalization using modified Gram Schmidt and classical Gram Schmidt. We also present versions of the algorithms that reduce communication along with their parallelizations. For example, Algorithm [17] is a block Gram Schmidt $\alpha$-orthonormalization based on classical Gram Schmidt that $\alpha$-orthonormalizes $P_k$ against previous vectors. And Algorithm 20 $\alpha$-orthonormalizes $P_k$’s vectors against each others using a classical Gram Schmidt.

### 3.2.3 Finding the expression of $\alpha_{k+1}$ and $\beta_{k+1}$

At each iteration the step $\alpha_{k+1}$ is chosen such that

$$\phi(x_{k+1}) = \min \{ \phi(x_k + P_{k+1} \alpha), \forall \alpha \in \mathbb{R}^t \}$$

Let $F(\alpha) = \phi(x_k + P_{k+1} \alpha)$ where $\phi(x) = \frac{1}{2} x^T A x - t^T b$.

Then, $F(\alpha) = \frac{1}{2} (x_k + P_{k+1} \alpha)^T A (x_k + P_{k+1} \alpha) - (x_k + P_{k+1} \alpha)^T b$

$$= \phi(x_k) + \frac{1}{2} [(x_k)^T A P_{k+1} \alpha + \alpha^T (P_{k+1})^T A x_k + \alpha^T (P_{k+1})^T A P_{k+1} \alpha] - \alpha^T (P_{k+1})^T b$$

$$= \phi(x_k) + \frac{1}{2} [(x_k)^T A P_{k+1} \alpha - \alpha^T (P_{k+1})^T A x_k] + \frac{1}{2} \alpha^T (P_{k+1})^T A P_{k+1} \alpha - \alpha^T (P_{k+1})^T r_k$$

$$= \phi(x_k) + \frac{1}{2} \alpha^T (P_{k+1})^T A P_{k+1} \alpha - \alpha^T (P_{k+1})^T r_k$$

The minimum of $F(\alpha)$ is given by $F'(\alpha) = 0$.

$$\Rightarrow F'(\alpha) = (P_{k+1})^T A P_{k+1} \alpha - (P_{k+1})^T r_k = 0$$

Therefore,

$$\alpha_{k+1} = (P_{k+1}^T A P_{k+1})^{-1} (P_{k+1}^T r_k).$$

As for $\beta_{k+1}$, it should be chosen to ensure that $P_{k+1}$ is $\alpha$-orthogonal to $P_k$. $P_{k+1} = T(r_k) + P_k \text{diag}(\beta_{k+1})$ and $P_k^T A P_{k+1} = P_k^T A T(r_k) + P_k^T A P_k \text{diag}(\beta_{k+1})$. Since $P_k$ is an $\alpha$-orthonormal matrix, $P_k^T A P_k = I$, $\text{diag}(\beta_{k+1})$ should be equal to $-P_k^T A T(r_k)$. But nothing guarantees that $P_k^T A T(r_k)$ is.
a diagonal matrix. So we choose $\beta_{k+1} = (P_k^t A P_k)^{-1} P_k^t A r_k$ which guarantees that $P_{k+1} * \mathbb{1}_t$ is A-orthogonal to $P_k$, similarly to MSD-CG. Moreover, in case $P_k^t A T(r_k)$ is a diagonal matrix, then our choice of $\beta_{k+1}$ implies that $P_{k+1}$ is A-orthogonal to $P_k$. If $t = 1$, then MSDO-CG is reduced to the classical conjugate gradient.

Note that, since the vectors of $P_{k+1}$ are A-orthonormalized ($P_k^t A P_k = I$), then $\alpha_{k+1}$ and $\beta_{k+1}$ systems are reduced to $\alpha_{k+1} = P_k^t r_k$ and $\beta_{k+1} = -P_k^t A r_k$.

### 3.2.4 The MSDO-CG Algorithm

After deriving the recurrence relations of $x_k, r_k, p_k, \alpha_k,$ and $\beta_k$, we present the MSDO-CG algorithm in Algorithm 1. We do not specify the A-orthonormalization methods, since this choice will be based first on the numerical stability of the method (section 4), then on its parallelization with the least communication possible (section 5).

Thus we present the MSDO-CG algorithm (Algorithm 1) and the computed flops per iteration except for the A-orthonormalizations. To reduce communication and computation in the A-orthonormalizations, be it MGS (Algorithms 14 and 15), CGS (Algorithms 18 and 21), A-CholQR (Algorithm 25), or Pre-CholQR (Algorithm 27), we replace $W_{k+1} = A P_{k+1}$ by

$$
\begin{align*}
W_1 &= A P_1 \\
W_{k+1} &= A T(r_k) + A P_k \text{diag}(\beta) \quad \forall k \geq 1
\end{align*}
$$

**Algorithm 1** MSDO-CG algorithm

<table>
<thead>
<tr>
<th>Input: $A$, the $n \times n$ symmetric positive definite matrix</th>
<th>Flops</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: $b$, the $n \times 1$ right-hand side; $x_0$, the initial guess or iterate</td>
<td>$2nnz + 2n - 1$</td>
</tr>
<tr>
<td>Input: $\epsilon$, the stopping tolerance; $k_{\text{max}}$, the maximum allowed iterations</td>
<td>$2nnz - (t - 1)n$</td>
</tr>
<tr>
<td>Output: $x_k$, the approximate solution of the system $Ax = b$</td>
<td>not included here</td>
</tr>
</tbody>
</table>

1. $r = b - A x_0$, $p_0 = ||r||_2^2$, $k = 1$
2. Let $P_1 = T(r_0)$ and $W_1 = A P_1$
3. A-orthonormalize $P_1$
4. while ($\sqrt{\rho} > \epsilon ||b||_2$ and $k < k_{\text{max}}$) do
5. $\alpha = (P_k^t W_k)^{-1} (P_k^t r) = P_k^t r$
6. $x = x + P_k \alpha$
7. $r = r - W_k \alpha$
8. $\rho = ||r||_2^2$
9. $\beta = -(P_k^t W_k)^{-1} (W_k^t r) = -W_k^t r$
10. $P_{k+1} = T(r) + P_k \text{diag}(\beta)$
11. $W_{k+1} = A T(r_k) + W_k \text{diag}(\beta)$
12. A-orthonormalize $P_{k+1}$ against all $P_i$’s for $i \leq k$
13. A-orthonormalize $P_k$
14. $k = k + 1$
15. end while

The total number of flops computed sequentially after $k_c$ iterations, except for the A-orthonormalizations, is

$$
\text{Total Flops} = 4nnz - nt + 5n - 1 + k_c [11nt - 2t + 2n - 1 + 2nnz + n + 1]
$$

The total number of flops computed sequentially after $k_c$ iterations, except for the A-orthonormalizations, is

$$
\text{Total Flops} = 4nnz - nt + 5n - 1 + k_c [11nt + 3n - 2t + 2nnz]
$$

which is of the order of $nnz k_c + ntk_c$ flops, where $nnz$ is the number of nonzero entries in the $n \times n$ matrix $A$ and $t$ is the number of search directions computed at each iteration.
It must be noted that since the $P_i$’s are $A$-orthonormal to each others, then the $t \times t$ matrix $P_k^t W_k = P_k^t A P_k$ is the identity matrix. Hence, solving for $\alpha_k$ and $\beta_k$ is simply performing matrix vector multiplication.

### 3.3 Long Recurrence Enlarged Conjugate Gradient (LRE-CG) Method

In this section, we introduce the long recurrence enlarged conjugate gradient (LRE-CG) method which is an enlarged Krylov projection method that solves the system $Ax = b$ by approximating the solution at the $k^{th}$ iteration with the vector $x_k = x_{k-1} + Q_k \alpha_k \in \mathcal{K}_{t,k}$ such that

$$\phi(x_k) = \min \{ \phi(x), \forall x \in x_0 + \mathcal{K}_{t,k} \},$$

where $Q_k \alpha_k \in \mathcal{K}_{t,k}$ and $Q_k$ is an $n \times t k$ matrix containing the orthonormal basis vectors of $\mathcal{K}_{t,k}$ and $\phi(x) = \frac{1}{2} x^t A x - x^t b$. The LRE-CG method does not have short recurrences as MSDO-CG, but it has similarities with GMRES in that the whole basis is used to define the new approximate solution rather than $t$ search directions. As mentioned earlier, the minimum of $\phi(x)$ is given by $\nabla \phi(x) = 0$ which is equivalent to $Ax - b = 0$. Thus, by minimizing $\phi(x)$ we are solving the system $Ax = b$. Since $\phi(x_k) = \min \{ \phi(x), \forall x \in x_0 + \mathcal{K}_{t,k} \}$, then

$$\phi(x_k) = \phi(x_{k-1} + Q_k \alpha_k) = \min \{ \phi(x_{k-1} + Q_k \alpha), \forall \alpha \in \mathbb{R}^{tk} \}.$$  \(11\)

Once $x_k$ has been chosen, either $x_k$ is the exact solution of $Ax = b$, or $t$ new basis vectors and the new approximation $x_{k+1} = x_k + Q_{k+1} \alpha_{k+1}$ are computed. This procedure is repeated until convergence.

Thus, we need to find the recursion relations of $r_k$ and $\alpha_k$. By definition, the residual $r_k = b - Ax_k$ where $x_k \in x_0 + \mathcal{K}_{t,k}$. Thus $r_k \in \mathcal{K}_{t,k+1}$. The recursion relation of $r_k$ can be simply obtained by replacing $x_k$ by its expression as follows:

$$r_k = b - Ax_k$$

$$= b - A(x_{k-1} + Q_k \alpha_k)$$

$$= r_{k-1} - AQ_k \alpha_k.$$

At each iteration the step $\alpha_{k+1}$ is chosen such that

$$\phi(x_{k+1}) = \min \{ \phi(x_k + Q_{k+1} \alpha), \forall \alpha \in \mathbb{R}^t(k + 1) \}.$$  \(11\)

Let $F(\alpha) = \phi(x_k + Q_{k+1} \alpha)$ where $\phi(x) = \frac{1}{2} x^t A x - x^t b$. Then,

$$F(\alpha) = \frac{1}{2}(x_k + Q_{k+1} \alpha)^t A (x_k + Q_{k+1} \alpha) - (x_k + Q_{k+1} \alpha)^t b$$

$$= \phi(x_k) + \frac{1}{2} \left[ (x_k)^t AQ_{k+1} \alpha + \alpha^t (Q_{k+1})^t A x_k + \alpha^t (Q_{k+1})^t AQ_{k+1} \alpha - \alpha^t (Q_{k+1})^t b \right]$$

$$= \phi(x_k) + \frac{1}{2} \left[ (x_k)^t AQ_{k+1} \alpha - \alpha^t (Q_{k+1})^t A x_k \right] + \frac{1}{2} \alpha^t (Q_{k+1})^t AQ_{k+1} \alpha - \alpha^t (Q_{k+1})^t r_k$$

$$= \phi(x_k) + \frac{1}{2} \alpha^t (Q_{k+1})^t AQ_{k+1} \alpha - \alpha^t (Q_{k+1})^t r_k, \quad \text{since } A \text{ is SPD}.$$  \(11\)

The minimum of $F(\alpha)$ is given by $F'(\alpha) = 0$

$$\Rightarrow F'(\alpha) = (Q_{k+1})^t AQ_{k+1} \alpha = (Q_{k+1})^t r_k = 0.$$  \(11\)

Therefore, $\alpha_{k+1} = (Q_{k+1}^t AQ_{k+1})^{-1} (Q_{k+1}^t r_k)$.

By minimizing $\phi(x)$, the Petrov-Galerkin condition, $r_k \perp \mathcal{K}_{t,k}$, is ensured (Theorem 3.11). Therefore, $(r_k)^t r_i = 0$, for all $i < k$, and the residuals form an orthogonal set.

**Theorem 3.11.** The Petrov-Galerkin condition in LRE-CG, $r_k \perp \mathcal{K}_{t,k}$, is equivalent to $x_k$ being the minimum of $\phi(x)$ in $x_0 + \mathcal{K}_{t,k}$.

**Proof.**
1. \( x_k \) is the minimum of \( \phi(x) \) in \( x_0 + \mathcal{K}_{t,k} \) implies \( r_k \perp \mathcal{K}_{t,k} \)
   The minimum of \( F(\alpha) = \phi(x_k) = \phi(x_{k-1} + Q_k \alpha) \) is given by
   \( F'(\alpha) = (Q_k)^T A Q_k \alpha - (Q_k)^T r_{k-1} = 0 \). Since \( x_k \) is the minimum, then \( \alpha = \alpha_k \) and \( F'(\alpha) = -Q_k^t r_k = 0 \). Thus \( r_k \perp \mathcal{K}_{t,k} \).

2. \( r_k \perp \mathcal{K}_{t,k} \) implies \( x_k \) is the minimum of \( \phi(x) \) in \( x_0 + \mathcal{K}_{t,k} \) (Proof by contradiction).
   Assume that \( r_k \perp \mathcal{K}_{t,k} \) and \( x_k \) is not the minimum of \( \phi(x) \) in \( x_0 + \mathcal{K}_{t,k} \). Then \( F'(\alpha_k) \neq 0 \). Hence \( Q_k^t r_k \neq 0 \) and \( r_k \) is not orthogonal to \( \mathcal{K}_{t,k} \). This contradicts our assumption. Thus \( x_k \) is the minimum of \( \phi(x) \).

3.3.1 The LRE-CG Algorithm

After deriving the expressions and the recursion relations of
\[
\begin{align*}
   x_k &= x_{k-1} + Q_k \alpha_k, \\
   r_k &= r_{k-1} - A Q_k \alpha_k, \\
   \alpha_k &= (Q_k^T A Q_k)^{-1} (Q_k^T r_{k-1}),
\end{align*}
\]
we present in Algorithm 2 the LRE-CG algorithm and the performed flops, except for the orthonormalization. We refer to the cost of solving the \( tk \times tk \) linear system from step 5 in Algorithm 2 as \( \text{Solve}_\alpha(tk) \).

**Algorithm 2** The LRE-CG algorithm

| Input: \( A \), the \( n \times n \) symmetric positive definite matrix |
| Input: \( b \), the \( n \times 1 \) right-hand side; \( x_0 \), the initial guess or iterate |
| Input: \( \epsilon \), the stopping tolerance; \( k_{\text{max}} \), the maximum allowed iterations |
| Output: \( x_k \), the approximate solution of the system \( Ax = b \) |

1. \( r_0 = b - A x_0, \rho_0 = ||r_0||_2^2, k = 1 \) \( 2 \text{nnz} + 2n - 1 \)
2. Let \( W = T(r_0) \), normalise its vectors and then let \( Q = W \) \( 3n \)
3. while \(( \sqrt{\rho_{k-1}} > \epsilon ||b||_2 \) and \( k < k_{\text{max}} \)) do \( 2n \)
4. \( G = (Q^T A Q) \) \( (2n \text{nnz} - n)tk + (2n - 1)t^2 k^2 \)
5. \( \alpha = G^{-1}(Q^T r) \) \( 2tkn \)
6. \( x = x + Q \alpha \) \( 2tkn \)
7. \( r = r - A Q \alpha \) \( 2tkn \)
8. \( \rho_k = ||r||_2^2 \) \( 2n - 1 \)
9. Let \( W = AW \) \( (2n \text{nnz} - n)t \)
10. Orthonormalise the vectors of \( W \) against the vectors of \( Q \) \( \text{not included in here} \)
11. Orthonormalise the vectors of \( W \) and let \( Q = [Q \ W] \) \( \text{not included in here} \)
12. \( k = k+1 \) \( 1 \)
13. end while

The cost of the LRE-CG, using Algorithm 2 except for the orthonormalization in steps 10 and 11, is
Algorithm 3 The LRE-CG Algorithm

\textbf{Input:} $A$, the $n \times n$ symmetric positive definite matrix
\textbf{Input:} $b$, the $n \times 1$ right-hand side; $x_0$, the initial guess or iterate
\textbf{Input:} $\epsilon$, the stopping tolerance; $k_{max}$, the maximum allowed iterations
\textbf{Output:} $x_k$, the approximate solution of the system $Ax = b$

1: $r_0 = b - Ax_0$, $p_0 = ||r_0||^2_2$, $k = 1$
2: Let $W = T(r_0)$, normalise its vectors and then let $Q = W$ 
3: \textbf{while} ($\sqrt{p_k^{-1} > \epsilon||b||_2$ and $k < k_{max}$) \textbf{do}
4: \hspace{1em} $Z = AW$ 
5: \hspace{1em} $E = W^tZ$
6: \hspace{1em} \textbf{if} $k = 1$ \textbf{then}
7: \hspace{2em} $D = Z$ and $G = E$
8: \hspace{1em} \textbf{else}
9: \hspace{2em} $F = Q(:, 1 : t(k - 1))^tZ$ 
10: \hspace{2em} $G = \begin{pmatrix} G & F^t \\ F & E \end{pmatrix}$ and $D = [D \ W]$ 
11: \hspace{1em} \textbf{end if}
12: \hspace{1em} $\alpha = G^{-1}(Q^t r)$ 
13: \hspace{1em} $x = x + Q \alpha$
14: \hspace{1em} $r = r - D \alpha$
15: \hspace{1em} $p_k = ||r||^2_2$
16: \hspace{1em} Let $W = Z$
17: \hspace{1em} Orthonormalise the columns of $W$ against those of $Q$ 
18: \hspace{1em} Orthonormalise the vectors of $W$ and let $Q = [Q \ W]$ 
19: \hspace{1em} $k = k + 1$
20: \textbf{end while}

Total Flops = $2nnz + 7n - 1 + k_c[(2nnz + 5n - 1)t\frac{k_c + 1}{2} + 2n + 2nnzt - nt]$ 
+ $\sum_{k=1}^{k_c} Solve_o(tk) + (2n - 1)\frac{t^2}{2}(k_c + 1)(2k_c + 1)$ 
= $2nnz + 7n - 1 + k_c[(2nnz + 5n - 1)t\frac{k_c + 1}{2} + 2n + 2nnzt - nt] + \sum_{k=1}^{k_c} Solve_o(tk)$ 
+ $(2n - 1)\frac{t^2}{2}(2k_c^2 + 3k_c + 1)$ 
$\approx nnztk_c^2 + nt^2k_c^2 + \sum_{k=1}^{k_c} Solve_o(tk)$,

where the first term $nnztk_c^2$ corresponds to the multiplication $AQ$ and the second term $nt^2k_c^2$ corresponds to the orthonormalization with respect to previous vectors. As for the memory requirements, we have to store the matrix $A$ and $tk_c + 2$ vectors of size $n \times 1$. And there should be enough memory for the $tk_c \times tk_c$ matrix $Q^tAQ$.

However, the multiplication $Q_k^tAQ_k$ can be reduced since $Q_k = [Q_{k-1}W_k]$. Let $Z_k = AW_k$, then $D_k = AQ_k = [AQ_{k-1}Z_k]$. At iteration $k - 1$, $D_{k-1} = AQ_{k-1}$ is computed. Thus at iteration $k$, only $Z_k = AW_k$ is computed. As for $G_k = Q_k^tAQ_k$, it is equal to $G_k = Q_k^tAQ_k = Q_k^tD_k = \left( \begin{array}{ccc} Q_{k-1}^tD_{k-1} & Q_{k-1}^tZ_k \\ W_k^tD_{k-1} & W_k^tZ_k \end{array} \right) = \left( \begin{array}{cc} G_{k-1} & \hat{Q}_{k-1} \\ Z_k^tW_k & W_k^tZ_k \end{array} \right) = \left( \begin{array}{cc} G_{k-1} & F_k \\ E_k \end{array} \right)$,

where $G_{k-1}$ is computed at iteration $k - 1$, $F_k = Q_k^tZ_k$, and $E_k = W_k^tZ_k$. Thus computing $G_k = Q_k^tAQ_k$ can be reduced to computing $F_k$ and $E_k$. 

RR n° 8597
Then, the cost of \( k_c \) iterations of LRE-CG using Algorithm 3 except for the orthonormalization, is

\[
\text{Total Flops} = 2\text{nnz} + 7n - 1 + k_c([2\text{nnz} + 2n + \frac{1}{2}t + (n - \frac{1}{2})t^2 + (6n - 1)t k_e + (2n - 1)tk_e^2] + 2n] + \sum_{k=1}^{k_c} \text{Solve}_\alpha(tk) \
= 2\text{nnz} + 7n - 1 + k_c([2\text{nnz} + 2n + \frac{1}{2}t + (n - \frac{1}{2})t^2 + (6n - 1)t k_e + (2n - 1)tk_e^2] + 2n] + \sum_{k=1}^{k_c} \text{Solve}_\alpha(tk) \\
\approx 3ntk_c^2 + nt^2k_c^2 + nt^2k_e + \sum_{k=1}^{k_c} \text{Solve}_\alpha(tk)
\]

Note that \( tk_c \) should be much smaller than \( n \), or otherwise the cost of Algorithm 3 would be \( O(n^3 + \text{Solve}_\alpha(n)) \), and \( n \) vectors of size \( n \) have to be stored in addition to the \( n \times n \) system \( Q_k^T AQ_k \).

One remedy to this problem, that we do not address in this paper, is to restart LRE-CG after some iterations. But this restart might have an effect on convergence as in restarted GMRES. Another alternative is to choose a linearly independent subset of the \( t \) computed vectors at each iteration \( i \). This reduces the size of the system solved at each iteration. A third alternative is to compute at each iteration \( i \), \( t_i \) vectors and then choose a linearly independent set of cardinality \( \hat{t}_i \), where \( l_0 = t, t_i \leq t, \hat{t}_i = t_{i+1} \). This reduces not only the size of the system solved at each iteration, but also the memory requirements and the number of computed vectors per iteration. In exact precision, the second and third alternatives are equivalent by Theorem 3.4 since if a vector \( T_j(r_0) \) is linearly dependent on \( \{T_1(r_0),...,T_{j-1}(r_0),T_{j+1}(r_0),...,T_t(r_0)\} \) then \( AT_j(r_0) \) is linearly dependent on \( \{AT_1(r_0),...,AT_{j-1}(r_0),AT_{j+1}(r_0),...,AT_t(r_0)\} \). However, this has to be tested in finite precision. Note that there is an additional cost for choosing a linearly independent subset of the \( t \) or \( t_i \) vectors.

The \( tk \times tk \alpha \) system can be solved using iterative methods like Jacobi method or Krylov subspace methods. Moreover, the \( s \)-step or communication avoiding Krylov subspace methods can be used to reduce communication. We use Matlab’s backslash to solve the \( \alpha \) systems in the convergence tests that follow.

4 Convergence Results

After introducing the new CG methods, MSDO-CG and LRE-CG, we compare their convergence behavior with respect to different A-orthonormalization and orthonormalization schemes respectively. Then we compare the convergence behavior of both methods with respect to CG, Coop-CG, MSD-CG on several matrices for different number of partitions (2, 4, 8, 16, 32 and 64 partitions) or number of initial guesses (for Coop-CG only). The matrices are first reordered using Metis’s k-way partitioning [18] that defines the subdomains \( \delta_i \). Then \( x \) is chosen randomly using Matlab’s rand function and \( b = Ax \), except for the Elasticity3DMatrix where \( A \) and \( b \) are available. In the tables 2, 3, and 4 “Iter” is the number of iterations, \( k_c \), needed for convergence and “Err” is the relative error \( \frac{||x - \hat{x}|}{||x||} \) at convergence.

The first matrix Poisson2Dis a block tridiagonal matrix obtained from Poisson’s equation (sparse) using Matlab’s gallery(‘poisson’,100) function. The matrices referred to as NH2D, Sky2D, Sky3D, and AN3D, arise from boundary value problem of the convection diffusion equations

\[
\begin{align*}
\eta(x)u + \text{div}(a(x)u) - \text{div}(\kappa(x)\nabla u) &= f \text{ in } \Omega \\
u &= 0 \text{ on } \partial \Omega_D \\
\frac{\partial u}{\partial n} &= 0 \text{ on } \partial \Omega_N
\end{align*}
\]

where \( \Omega = [0, 1]^n \), \( n = 2, 3 \) and \( \partial \Omega_N = \partial \Omega \setminus \partial \Omega_D \). The function \( \eta \), the vector field \( a \), and the tensor \( \kappa \) are the given coefficients of the partial differential operator. In the 2D case, we have \( \partial \Omega_D = [0, 1] \times [0, 1] \times [0, 1] \), and in the 3D case, we have \( \partial \Omega_D = [0, 1] \times [0, 1] \times [0, 1] \). We focus on the following cases:

- NH2D: A non-homogeneous problem with large jumps in the coefficients. The coefficient \( \eta \) and \( a \) are both zero. The tensor \( \kappa \) is isotropic and discontinuous. It jumps from the constant value 10³ in the ring \( \frac{1}{2} \leq ||x-c|| \leq 1 \), \( c = (\frac{1}{2}, \frac{1}{2})^T \), to 0 outside.

\[\text{Inria}\]
• **Sky2D and Sky3D Skyscraper problems**: The tensor $\kappa$ is isotropic and discontinuous. The domain contains many zones of high permeability which are isolated from each other

$$\kappa(x) = \begin{cases} 
10^4 \ast (10 \ast x_2) + 1 & \text{if } 10x_i \text{ is odd, } i = 1, 2 \\
1, & \text{otherwise.}
\end{cases}$$

where we note $[x]$ as the integer value of $x$. Sky2D and Sky3D are discretized on a 2D and 3D cartesian grids respectively.

• **Ani3D Anisotropic layers**: the domain is made of 10 anisotropic layers with jumps of up to four orders of magnitude and an anisotropy ratio of up to $10^3$ in each layer. The domain is divided into 10 layers parallel to $z = 0$, of size 0.1, in which the coefficients are constant. We have $\kappa_y = 10\kappa_x$ and $\kappa_z = 1000\kappa_x$. The velocity field is zero.

**POISSON2D, Nh2D and Sky2D** are discretized on a $100 \times 100$ 2D cartesian grid. **Sky3D and Ani3D** are discretized on a $20 \times 20 \times 20$ grid.

As for the **Elasticity3D** matrix, it arises from the linear elasticity problem with Dirichlet and Neumann boundary conditions, defined as follows:

$$\text{div}(\sigma(u)) + f = 0 \quad \text{on } \Omega,$$
$$u = u_D \quad \text{on } \partial \Omega_D,$$
$$\sigma(u),n = g \quad \text{on } \partial \Omega_N,$$

where $\Omega$ is a $3D$ $30 \times 10 \times 10$ parallelepiped, $\Omega_D$ is the Dirichlet boundary, $\Omega_N$ is the Neumann boundary, $u$ is the unknown displacement field, $f$ is some body force, $\sigma(u)$ is the Cauchy stress tensor given by Hooke’s law. The **Elasticity3D** matrix was discretized with P1 finite elements and a triangular mesh using FreeFem++ [15]. For a detailed description of the problem refer to [12]. Table 1 briefly describes the test matrices.

<table>
<thead>
<tr>
<th>Matrix</th>
<th>Size</th>
<th>Nonzeros</th>
<th>Symetric</th>
<th>2D/3D</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>POISSON2D</td>
<td>10000</td>
<td>49600</td>
<td>Yes</td>
<td>2D</td>
<td>Poisson equations</td>
</tr>
<tr>
<td>Nh2D</td>
<td>10000</td>
<td>49600</td>
<td>Yes</td>
<td>2D</td>
<td>Boundary value</td>
</tr>
<tr>
<td>Sky2D</td>
<td>10000</td>
<td>49600</td>
<td>Yes</td>
<td>2D</td>
<td>Boundary value</td>
</tr>
<tr>
<td>Sky3D</td>
<td>8000</td>
<td>53600</td>
<td>Yes</td>
<td>3D</td>
<td>Skyscraper</td>
</tr>
<tr>
<td>Ani3D</td>
<td>8000</td>
<td>53600</td>
<td>Yes</td>
<td>3D</td>
<td>Anisotropic Layers</td>
</tr>
<tr>
<td>Elasticy3D</td>
<td>11253</td>
<td>373647</td>
<td>Yes</td>
<td>3D</td>
<td>Linear Elasticity P1 FE</td>
</tr>
</tbody>
</table>

In table 2 we compare the convergence behavior of the MSDO-CG method (Algorithm 1) with different A-orthonormalization schemes for A-orthonormalizing $P_k$ against previous $P_i$’s (MGS, CGS, CGS2) and then A-orthonormalizing $P_k$ against itself (MGS, CGS, CGS2, A-CholQR, Pre-CholQR) and for different number of partitions $t = 2, 4, 8, 16, 32, 64$ that correspond to the maximum number of vectors added at each iteration to the enlarged Krylov subspace, $K_{t,k}$. We have tested different combinations of A-orthonormalization, but we only show MGS (MGS+MGS), CGS+A-CholQR, CGS+Pre-CholQR, CGS2+Pre-CholQR, and CGS2+Pre-CholQR. Note that MSDO-CG with CGS A-orthonormalization (CGS+CGS) did not converge neither with CGS2 A-orthonormalization (CGS2+CGS2) nor with CGS2+CGS or CGS+CGS2 A-orthonormalization. The reason is that the search directions are not A-orthogonal to satisfactory precision. And by Theorem 3.10, this implies that $r_k \not\subset K_{t,k}$. Thus, nothing guarantees convergence since we have shown in section 3.1.3 that MSDO-CG converges faster than CG if $r_k \perp K_{t,k}$. Moreover, we did not test combinations of MGS and QR factorizations since MGS is expensive in terms of communication compared to the other methods (section 5.1). But we tested MSDO-CG with MGS for comparison purposes since MGS is known for its numerical stability.
Table 2: Comparison of the convergence of MSDO-CG with different A-orthonormalization schemes, with respect to number of partitions (Pa) with \( x_0 = 0 \).

<table>
<thead>
<tr>
<th>MSDO-CG with different A-orthonormalization Methods</th>
<th>MGS</th>
<th>CGS+A-CholQR</th>
<th>CGS+Pre-CholQR</th>
<th>CGS2+A-CholQR</th>
<th>CGS2+Pre-CholQR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pa</td>
<td>Iter</td>
<td>Err</td>
<td>Iter</td>
<td>Err</td>
<td>Iter</td>
</tr>
<tr>
<td>2</td>
<td>200</td>
<td>4E-5</td>
<td>204</td>
<td>3E-5</td>
<td>204</td>
</tr>
<tr>
<td>4</td>
<td>167</td>
<td>2E-5</td>
<td>167</td>
<td>2E-5</td>
<td>167</td>
</tr>
<tr>
<td>8</td>
<td>139</td>
<td>1E-5</td>
<td>139</td>
<td>1E-5</td>
<td>139</td>
</tr>
<tr>
<td>16</td>
<td>121</td>
<td>5E-6</td>
<td>121</td>
<td>5E-6</td>
<td>121</td>
</tr>
<tr>
<td>32</td>
<td>94</td>
<td>2E-6</td>
<td>94</td>
<td>2E-6</td>
<td>94</td>
</tr>
<tr>
<td>64</td>
<td>69</td>
<td>2E-6</td>
<td>69</td>
<td>2E-6</td>
<td>69</td>
</tr>
</tbody>
</table>

As shown in table 2, MSDO-CG with MGS A-orthonormalization converges for all the tested matrices and as we increase \( t \), the number of iterations needed for convergence decreases. As we mentioned earlier, MSDO-CG with CGS A-orthonormalization did not converge. Therefore, we replaced CGS with CGS+A-CholQR and with CGS+Pre-CholQR A-orthonormalization. We notice that MSDO-CG with CGS+A-CholQR A-orthonormalization and MSDO-CG with CGS+Pre-CholQR A-orthonormalization have the same convergence behavior. For the matrices \( \text{POISSON2D} \) and \( \text{NH2D} \), both methods converge with the same number of iterations as MSDO-CG with MGS A-orthonormalization. However, for the matrix \( \text{SKY2D} \), both methods did not converge. As for the matrices \( \text{SKY3D} \) and \( \text{AN3D} \), both methods converged only for \( t = 2 \) partitions, and \( t = 2, 4, 8 \) partitions respectively. The reason for this difference in behavior for different matrices is the condition number \( (\text{cond}_2 = ||A||_2||A^{-1}||_2) \). The condition number of the matrices \( \text{POISSON2D} \) and \( \text{NH2D} \) is \( 6 \times 10^3 \), whereas that of the matrices \( \text{SKY3D}, \text{AN3D} \) is much higher. This indicates that the matrices \( \text{POISSON2D} \) and \( \text{NH2D} \) are better conditioned than \( \text{SKY3D}, \text{AN3D} \).
and Sky2D is $1 \times 10^6$, $2 \times 10^6$, and $3 \times 10^7$ respectively. Although it was shown in [20] that Pre-CholQR A-orthonormalization is more stable than A-CholQR, however MSDO-CG with CGS+A-CholQR A-orthonormalization and MSDO-CG with CGS+Pre-CholQR A-orthonormalization are both numerically unstable.

Thus, we replace CGS with CGS2 where the A-orthonormalization is performed twice for numerical stability. Then, the MSDO-CG with CGS2+A-CholQR A-orthonormalization and MSDO-CG with CGS2+Pre-CholQR A-orthonormalization converge as fast as MSDO-CG with MGS A-orthonormalization for all $t$ and all the tested matrices. Hence, we conclude that MGS, CGS2+A-CholQR, and CGS2+Pre-CholQR A-orthonormalizations are stable enough to be used in the MSDO-CG method (Algorithm 1).

Table 3: Comparison of the convergence of the LRE-CG method with different orthonormalization schemes, with respect to number of partitions $P_a$, with $x_0 = 0$.

<table>
<thead>
<tr>
<th>Orthonormalization Methods</th>
<th>$\text{LRE-CG with different}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\text{MGS+MGS}$</td>
</tr>
<tr>
<td>$P_a$</td>
<td>Iter</td>
</tr>
<tr>
<td>Poisson2D</td>
<td>2</td>
</tr>
<tr>
<td>tol = $10^{-6}$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>64</td>
</tr>
<tr>
<td>Nh2D</td>
<td>2</td>
</tr>
<tr>
<td>tol = $10^{-8}$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>64</td>
</tr>
<tr>
<td>Sky2D</td>
<td>2</td>
</tr>
<tr>
<td>tol = $10^{-8}$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>64</td>
</tr>
<tr>
<td>Sky3D</td>
<td>2</td>
</tr>
<tr>
<td>tol = $10^{-8}$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>64</td>
</tr>
<tr>
<td>An13D</td>
<td>2</td>
</tr>
<tr>
<td>tol = $10^{-8}$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>64</td>
</tr>
</tbody>
</table>

In Table 3, we compare the convergence behavior of the LRE-CG method (Algorithm 3) with different orthonormalization schemes, with respect to number of partitions $P_a$, with $x_0 = 0$. 

In table 3 we compare the convergence behavior of the LRE-CG method (Algorithm 3) with differ
Table 4: Comparison between the convergence of the different CG versions with respect to number of partitions or initial guesses for Coop-CG with \( x_0 = 0 \).

<table>
<thead>
<tr>
<th></th>
<th>CG</th>
<th>Coop-CG</th>
<th>MSD-CG</th>
<th>MSDO-CG</th>
<th>LRE-CG</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Poisson2D</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( tol = 10^{-6} )</td>
<td>2</td>
<td>195</td>
<td>2E-5</td>
<td>206</td>
<td>2E-7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>195</td>
<td>2E-5</td>
<td>171</td>
<td>1E-7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>195</td>
<td>2E-5</td>
<td>137</td>
<td>1E-7</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>195</td>
<td>2E-5</td>
<td>106</td>
<td>3E-8</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>195</td>
<td>2E-5</td>
<td>80</td>
<td>1E-8</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>195</td>
<td>2E-5</td>
<td>59</td>
<td>1E-8</td>
</tr>
<tr>
<td><strong>NH2D</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( tol = 10^{-8} )</td>
<td>2</td>
<td>259</td>
<td>4E-7</td>
<td>206</td>
<td>2E-7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>259</td>
<td>4E-7</td>
<td>179</td>
<td>1E-7</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>259</td>
<td>4E-7</td>
<td>157</td>
<td>2.0E-5</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>259</td>
<td>4E-7</td>
<td>107</td>
<td>2E-8</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>259</td>
<td>4E-7</td>
<td>81</td>
<td>2E-8</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>259</td>
<td>4E-7</td>
<td>59</td>
<td>1E-8</td>
</tr>
<tr>
<td><strong>SKY2D</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( tol = 10^{-8} )</td>
<td>2</td>
<td>5951</td>
<td>4E-4</td>
<td>4893</td>
<td>2E-4</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5951</td>
<td>4E-4</td>
<td>3737</td>
<td>9E-5</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>5951</td>
<td>4E-4</td>
<td>3391</td>
<td>1E-5</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>5951</td>
<td>4E-4</td>
<td>2437</td>
<td>9E-6</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>5951</td>
<td>4E-4</td>
<td>1406</td>
<td>4E-6</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>5951</td>
<td>4E-4</td>
<td>802</td>
<td>2E-6</td>
</tr>
<tr>
<td><strong>SKY3D</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( tol = 10^{-8} )</td>
<td>2</td>
<td>902</td>
<td>1E-5</td>
<td>795</td>
<td>8E-6</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>902</td>
<td>1E-5</td>
<td>627</td>
<td>1E-5</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>902</td>
<td>1E-5</td>
<td>542</td>
<td>4E-6</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>902</td>
<td>1E-5</td>
<td>414</td>
<td>3E-6</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>902</td>
<td>1E-5</td>
<td>290</td>
<td>1E-6</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>902</td>
<td>1E-5</td>
<td>183</td>
<td>8E-7</td>
</tr>
<tr>
<td><strong>AN3D</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( tol = 10^{-8} )</td>
<td>2</td>
<td>4187</td>
<td>4E-5</td>
<td>3584</td>
<td>5E-5</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>4146</td>
<td>4E-5</td>
<td>3371</td>
<td>4E-5</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>4146</td>
<td>4E-5</td>
<td>2865</td>
<td>4E-5</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>4146</td>
<td>4E-5</td>
<td>2314</td>
<td>3E-5</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>4146</td>
<td>4E-5</td>
<td>1615</td>
<td>2E-5</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>4146</td>
<td>4E-5</td>
<td>1002</td>
<td>1E-5</td>
</tr>
<tr>
<td><strong>Elasticity3D</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( tol = 10^{-8} )</td>
<td>2</td>
<td>987</td>
<td>4E-12</td>
<td>718</td>
<td>3e-12</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>987</td>
<td>4E-12</td>
<td>534</td>
<td>8e-12</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>987</td>
<td>4E-12</td>
<td>425</td>
<td>6e-11</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>987</td>
<td>4E-12</td>
<td>348</td>
<td>6E-11</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>987</td>
<td>4E-12</td>
<td>294</td>
<td>9e-12</td>
</tr>
<tr>
<td></td>
<td>64</td>
<td>987</td>
<td>4E-12</td>
<td>235</td>
<td>1e-11</td>
</tr>
</tbody>
</table>

Different orthonormalization schemes for orthonormalizing \( W \) against the \( n \times tl \) matrix \( Q \) (MGS, CGS) and then orthonormalizing \( W \) against itself (MGS, CGS, TSQR) and for different number of partitions \( t = 2, 4, 8, 16, 32, 64 \) that correspond to the maximum number of vectors added at each iteration to the enlarged Krylov subspace, \( \mathcal{K}_{t,k} \). We start by testing the convergence of LRE-CG with MGS (MGS+MGS) orthonormalization. It converges for all the tested matrices since it is numerically stable, and the number
of iterations needed for convergence decreases when increasing the number of partitions $t$. However, as mentioned in section 5.1, MGS is expensive in terms of communication ($O(tk\log(t))$ messages per iteration, where $t$ processors A-orthonormalized $t$ vectors against $tk$ vectors). Thus, we tested the LRE-CG method with Classical Gram Schmidt (CGS) orthogonalization which requires sending $O(tk\log(t))$ messages per iteration. The LRE-CG with CGS converges in the same number of iterations as LRE-CG with MGS for the matrices $\text{POISSON2D}$ and $\text{Ni2D}$. However, for the other matrices, it does not converge for the given stopping criteria except for $t = 2$ as shown in table 5. The reason is that the matrix $C = Q^TAQ$ is becoming close to singular, with $\text{rank}(C) < tk$, as the iterations proceed due to the loss of orthogonality in the CGS orthogonalization. The number of iterations in parentheses in table 5 is not the number of iterations for convergence but it denotes the iteration at which the $C$ matrix becomes close to singular.

In CA-GMRES [21], the authors use a parallelizable tall and skinny QR (TSQR) factorization [4] for orthonormalizing the $n \times t$ tall and skinny matrix instead of CGS. They have shown that the combination of CGS for orthonormalization $W$ against $Q$ and TSQR for orthonormalizing $W$ is stable. We have tested LRE-CG with CGS and TSQR (CGS+TSQR) orthonormalization, and it has the same convergence behavior as LRE-CG with MGS (MGS+MGS) orthonormalization (table 5). Thus, we conclude that MGS, and CGS+TSQR orthonormalizations are stable enough to be used in the LRE-CG method from Algorithm 3.

In table 5 we compare the convergence behavior of MSDO-CG with MGS A-orthonormalization, LRE-CG with MGS orthonormalization, Coop-CG and MSD-CG with respect to CG for several matrices with different number of partitions $t = 2, 4, 8, 16, 32, 64$ . The MSDO-CG, COOP-CG and LRE-CG have better convergence than CG, and LRE-CG has the best convergence. The MSD-CG converges, but requires more iterations than CG, three times more iterations for the matrices $\text{Sky2D, Sky3D, Ani3D, and ELASTICITY3D}$. As for Coop-CG, which starts with $t$ different initial guesses and solves two systems of fixed size $t \times t$, its convergence is slightly better than MSDO-CG for the matrices $\text{POISSON2D}$, $\text{Ni2D}$, and ELASTICITY3D. But it requires much more iterations than both MSDO-CG and LRE-CG for the other matrices ($\text{Sky2D, Sky3D, Ani3D}$). Moreover, the results may vary depending on the $t$ initial guesses that are used for the different matrices.

Table 5: The memory requirements in words, for MSDO-CG and LRE-CG with a varying number of partitions (Pa) or $t$ and for different matrices.

<table>
<thead>
<tr>
<th>Pa</th>
<th>Iter</th>
<th>Mem</th>
<th>Iter</th>
<th>Mem</th>
<th>Iter</th>
<th>Mem</th>
<th>Iter</th>
<th>Mem</th>
<th>Iter</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>200</td>
<td>4040004</td>
<td>256</td>
<td>516004</td>
<td>1559</td>
<td>3122004</td>
<td>610</td>
<td>9792004</td>
<td>893</td>
<td>1320004</td>
</tr>
<tr>
<td>4</td>
<td>167</td>
<td>6740016</td>
<td>208</td>
<td>8380016</td>
<td>917</td>
<td>36740016</td>
<td>420</td>
<td>13488016</td>
<td>749</td>
<td>24016016</td>
</tr>
<tr>
<td>8</td>
<td>139</td>
<td>11220064</td>
<td>169</td>
<td>13620064</td>
<td>532</td>
<td>42660064</td>
<td>228</td>
<td>14672064</td>
<td>498</td>
<td>31952064</td>
</tr>
<tr>
<td>16</td>
<td>121</td>
<td>19540256</td>
<td>138</td>
<td>22260256</td>
<td>307</td>
<td>49300256</td>
<td>134</td>
<td>17296256</td>
<td>328</td>
<td>42128256</td>
</tr>
<tr>
<td>32</td>
<td>94</td>
<td>30421024</td>
<td>107</td>
<td>34581024</td>
<td>178</td>
<td>57301024</td>
<td>87</td>
<td>22545024</td>
<td>192</td>
<td>49425024</td>
</tr>
<tr>
<td>64</td>
<td>69</td>
<td>44824096</td>
<td>77</td>
<td>49944096</td>
<td>126</td>
<td>81304096</td>
<td>53</td>
<td>27668096</td>
<td>122</td>
<td>62996096</td>
</tr>
</tbody>
</table>

MSDO-CG

<table>
<thead>
<tr>
<th>Pa</th>
<th>Iter</th>
<th>Mem</th>
<th>Iter</th>
<th>Mem</th>
<th>Iter</th>
<th>Mem</th>
<th>Iter</th>
<th>Mem</th>
<th>Iter</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>193</td>
<td>4028996</td>
<td>245</td>
<td>5160100</td>
<td>1415</td>
<td>36328900</td>
<td>557</td>
<td>10168996</td>
<td>875</td>
<td>17078500</td>
</tr>
<tr>
<td>4</td>
<td>153</td>
<td>6515454</td>
<td>188</td>
<td>8105504</td>
<td>757</td>
<td>39468784</td>
<td>373</td>
<td>14178064</td>
<td>673</td>
<td>28798864</td>
</tr>
<tr>
<td>8</td>
<td>123</td>
<td>10828256</td>
<td>149</td>
<td>13360864</td>
<td>398</td>
<td>41997856</td>
<td>211</td>
<td>16369344</td>
<td>449</td>
<td>41654464</td>
</tr>
<tr>
<td>16</td>
<td>95</td>
<td>17530400</td>
<td>112</td>
<td>21151264</td>
<td>230</td>
<td>47610400</td>
<td>119</td>
<td>18873216</td>
<td>253</td>
<td>48786304</td>
</tr>
<tr>
<td>32</td>
<td>70</td>
<td>27437600</td>
<td>82</td>
<td>33415376</td>
<td>126</td>
<td>56597024</td>
<td>69</td>
<td>22555264</td>
<td>148</td>
<td>60333696</td>
</tr>
<tr>
<td>64</td>
<td>52</td>
<td>44375584</td>
<td>60</td>
<td>53165600</td>
<td>75</td>
<td>71060000</td>
<td>43</td>
<td>29605504</td>
<td>92</td>
<td>81788544</td>
</tr>
</tbody>
</table>

LRE-CG

For the tested matrices, LRE-CG has slightly better convergence than MSDO-CG, since it uses the whole basis to define the new approximate solution rather than $t$ search directions. For the matrices
POISSON2D and Nth2D, LRE-CG and MSDO-CG have almost the same convergence as CG for \( t = 2 \), and then as \( t \) is doubled the iterations needed for convergence is decreased by 20% to 30%. For \( t = 2 \), LRE-CG requires 35% and 40% less iterations than CG for the matrices Elasticity3D and Sky3D respectively. And as \( t \) is doubled the number of iterations needed for convergence is decreased by 25% to 30%, and 32% to 45% respectively. For \( t = 2 \), LRE-CG requires 60% and 80% less iterations than CG for the matrices Sky2D and Ant3D respectively. And as \( t \) is doubled, the number of iterations needed for convergence is decreased by 45% to 50% and 25% to 40% respectively.

As it is clear from the convergence tests, by doubling \( t \) the number of iterations needed for convergence is not always reduced by 50% for all the matrices. However, as shown in the previous sections the memory requirements for MSDO-CG and LRE-CG, except for the matrix \( A \), is \((tk_c + 2)n + (tk_c)^2\) and \((tk_c + t + 2)n + t^2\) respectively, where \( n \) is the size of the matrix, \( k_c \) is the number of computed iterations. Thus by doubling \( t \), the memory requirements for MSDO-CG and LRE-CG for performing \( k \) iterations is at least doubled. But, when \( t \) is doubled \( k_c \) is decreased. Thus, the memory requirements increase and at most double, when \( t \) is doubled as shown in Table 5. Hence, \( t \) should be relatively small depending on the size of the matrix, on the performed iterations and on the available memory.

5 Parallel Model and Expected Performance

In this section we describe the parallelization of the MSDO-CG method (section 5.1) and the LRE-CG method (section 5.2) with computed flops, number of messages and words sent and the estimated parallel runtime. The estimated time for computing \( \gamma_c z \), where \( \gamma_c \) is the inverse floating-point rate, also called the floating-point throughput (seconds per floating-point operation). The estimated time for sending a message of size \( k \) is \( \alpha_c + \beta_k k \), where \( \alpha_c \) is the latency (with units of seconds) and \( \beta_k \) is the inverse bandwidth (seconds per word). Hence, the estimated runtime of an algorithm with a total of \( z \) computed flops and \( s \) sent messages each of size \( k \) is the sum of their corresponding estimated times \( \gamma_c z + \alpha_c s + \beta_k \).

For simplicity, we assume that the algorithms are executed on a distributed memory machine formed by \( t \) processors, where \( t \) corresponds to the number of vectors computed at each iteration. We partition the graph of \( A \) into \( t \) subdomains using \( k \)-way partitioning or another graph partitioning. We denote by \( \delta_i \), for \( i = 1, 2, \ldots, t \), the subsets of indices obtained from the partitioning. That is \( \delta_i \cap \delta_h = \emptyset \) for all \( i \neq h \). \( \bigcup_{h=1}^{t} \delta_h = \{1, 2, 3, \ldots, n\} \), and \( \delta_i \approx \frac{n}{t} \). Then each processor \( i \) is assigned the \( \frac{n}{t} \times n \) rowwise part of the matrix \( A(A(\delta_i,:), :) = A(:, \delta_i) \) since \( A \) is SPD). The \( \frac{n}{t} \times 1 \) rowwise part of the vector \( b(b(\delta_i)) \), and the vector \( x_0(\delta_i) \), where \( \delta_i = \text{Adjacent}(G(A), \delta_i) \) is the adjacent of \( \delta_i \) in the graph of \( A \). Processor \( i \) computes \( x_k(\delta_i) \).

However, for performance reasons and due to the multicore nature of most architectures, it is possible to use a number of processors greater than \( t \), preferably a multiple of \( t \). In this case, we start by partitioning the graph of \( A \) into \( t \) subdomains using \( k \)-way partitioning or another graph partitioning, where \( \delta_i \) for \( i = 1, 2, \ldots, t \) are the subsets of indices obtained from the partitioning. This partitioning is used to define the \( T(.) \) operator and eventually the enlarged Krylov subspace. Assuming that we have \( ct \) processors, then every \( c \) processors are assigned an \( \frac{n}{t} \times n \) rowwise part of the matrix \( A, A(\delta_i,:) \), \( \frac{n}{t} \times 1 \) rowwise part of the vector \( b(b(\delta_i)) \) and the vector \( x_0(\delta_i) \), and should output \( x_k(\delta_i) \). In other words, we partition each of our \( t \) subdomains into \( c \) non-overlapping subdomains to obtain a total of \( ct \) subdomains with set of indices \( \delta_{i,j} \), where \( i = 1, 2, \ldots, t, j = 1, 2, \ldots, c \), and \( \delta_i = \bigcup_{j=1}^{c} \delta_{i,j} \). Then, in Algorithms 2 and 3, \( \log(t) \) is replaced by \( \log(ct) \), and \( \frac{1}{t} \) is replaced by \( \frac{1}{ct} \).

Inria
5.1 MSDO-CG

In this section we describe the parallelization of the MSDO-CG algorithm and we estimate its runtime in terms of flops, number of messages, and words sent. As mentioned in section 4, MGS, CGS2+A-CholQR, and CGS2+Pre-CholQR A-orthonormalizations are numerically the most stable and allow the convergence of MSDO-CG for the matrices in our test set. As discussed in Appendix B, the most parallelizable versions of MGS, Algorithms 14 and 15, require sending \((tk+1)\log(t)\) and \(2(t-1)\log(t)\) messages respectively. Whereas CGS2, Algorithm 23, requires sending \(4\log(t)\) messages. On the other hand, Algorithm 25 of A-CholQR requires sending \(\log(t)\) messages, and Pre-CholQR Algorithm 27 requires sending \(3\log(t)\) messages. The CGS2-A-CholQR and CGS2+Pre-CholQR A-orthonormalizations can be called communication avoiding since they require sending \(5\log(t)\) and \(7\log(t)\) messages respectively, unlike the MGS A-orthonormalization. Since both methods are stable and CGS2+A-CholQR requires less communication, we present the Parallel MSDO-CG with CGS2+A-CholQR A-orthonormalization in Algorithm 3.

Algorithm 4: Parallel MSDO-CG with CGS2+A-CholQR A-orthonormalization

<table>
<thead>
<tr>
<th>Input:</th>
<th>δi, the set of indices assigned to processor i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input:</td>
<td>A(δi,:), the (\frac{n}{p} \times n) row part of A</td>
</tr>
<tr>
<td>Input:</td>
<td>b(δi), the (\frac{n}{p} \times 1) row part of b; x0(δi), the (</td>
</tr>
<tr>
<td>Input:</td>
<td>ε, the stopping tolerance; (k_{max}), the maximum allowed iterations</td>
</tr>
<tr>
<td>Output:</td>
<td>xk(δi), the row part of the approximate solution of (Ax = b)</td>
</tr>
</tbody>
</table>

1. for each processor \(i = 1 : t\) in parallel do
2. Processor \(i\) computes \(r(\delta_i) = b(\delta_i) - A(\delta_i, \bar{\delta}_i)x_0(\bar{\delta}_i)\), and let \(k = 1\) \(\gamma_c(2^{\frac{mnz}{2}})\)
3. Processor \(i\) computes \(r(\delta_i)^tr(\delta_i)\) and receives the full \(\rho = ||r||_2^2\) \(\gamma_c(2^{\frac{mnz}{2}})\)
4. via an all reduce (overlapped with the next computation) \(\gamma_c(2^{\frac{mnz}{2}})\)
5. Processor \(i\) sends \(P_i(\delta_i,:) = [T(r)](\delta_i,:) = [0, \ldots, 0, r(\delta_i), 0, \ldots, 0]\) to its \(m_{MB}\) neighboring processors and receives from them the corresponding blocks to obtain \(P_i(\bar{\delta}_i,:)\). Then it computes \(W_i(\delta_i) = A(\delta_i, \bar{\delta}_i)P_i(\bar{\delta}_i,:)\) \(\gamma_c(\frac{2^{\frac{mnz}{2}}}{t})\)
6. Call A-CholQR algorithm 25 to A-orthonormalize \(P_i\) \(\gamma_cAn(t + (\alpha_c + \beta_c^2)\log(t))\)
7. while \((\sqrt{t} > \epsilon||b||_2\) and \(k < k_{max}\) do
8. Processor \(i\) computes \(P_k(\delta_i,:)r(\delta_i)\) and receives the full \(\alpha = P_k(\delta_i,:)r\) via an all reduce \(\gamma_c(2^{\frac{mnz}{t}} - 1)t\)
9. Processor \(i\) computes \(x(\delta_i) = x(\delta_i) + P_k(\delta_i,:)\alpha\) \(\gamma_c(2^{\frac{mnz}{t}} - 1)t\)
10. and \(r(\delta_i) = r(\delta_i) - W_k(\delta_i,:)\alpha\) \(\gamma_c(2^{\frac{mnz}{t}} - 1)t\)
11. Processor \(i\) computes \(r(\delta_i)^tr(\delta_i)\) and receives \(\rho = ||r||_2^2\) \(\gamma_c(2^{\frac{mnz}{t}} - 1)t\)
12. via an all reduce (overlapped with the next communication) \(\gamma_c(2^{\frac{mnz}{t}} - 1)t\)
13. Processor \(i\) computes \(P_{k+1}(\delta_i,:) = [T(r)](\delta_i,:) + P_k(\delta_i,:)\text{diag}(\beta)\) \(\gamma_c(2^{\frac{mnz}{t}} - 1)t\)
14. Processor \(i\) sends \([T(r)](\delta_i,:)\) to its \(m_{MB}\) neighboring processors \(\gamma_c(2^{\frac{mnz}{t}} - 1)t\)
15. and receives from them the corresponding blocks to obtain \([T(r)](\bar{\delta}_i,:)\). Then it computes \(Z(\delta_i) = A(\delta_i, \bar{\delta}_i)[T(r)](\bar{\delta}_i,:)\) \(\gamma_c(2^{\frac{mnz}{t}} - 1)t\)
16. Processor \(i\) computes \(W_{k+1}(\delta_i,:) = Z(\delta_i,:) + W_k(\delta_i,:)\text{diag}(\beta)\) \(\gamma_c(2^{\frac{mnz}{t}} - 1)t\)
17. Call CGS2 Algorithm 22 to A-orthonormalize \(P_{k+1}\) against \(P_i\) \(\gamma_cAn(tk + 2(\alpha_c + \beta_c^2)\log(t))\)
18. end while
19. end for
In Algorithm 3 we have two types of communication. The first is an “all reduce” communication that requires synchronization between all the processors and is equivalent to \(\log(t)\) messages, each of the same size (refer to 26). For example, in line 10 of Algorithm 4 the “all reduce” is equivalent to \(\log(t)\) messages each of size \(t\) words, since \(\beta\) is a vector of size \(t\).

The second type of communication is a point-to-point communication between each processor \(i\) and its \(m_i\) neighboring processors for computing a matrix - block of vectors multiplication, specifically \(A[T(r)]\). We denote by \(m_{MB} = \max\{m_i\} \quad i = 1, 2, \ldots, t\) the largest number of neighboring processors, where \(m_i \leq m_{MB} \leq (t - 1)\) for all \(i\). Note that processor \(i\) has to compute \(A(\delta_i, \delta_i)[T(r)](\delta_i, :)\) where \(\delta_i = \text{Adjacent}(G(A), \delta_i)\). Then, the neighboring processors of a given processor \(i\) are defined as all the processors \(j\) from which processor \(i\) needs some rows of \([T(r)]\) to compute its part of \(A[T(r)]\). In other words, neighboring processors are all the processors \(j\) for which \(\delta_i \cap \delta_j \neq \emptyset\). Moreover, \([T(r)](\delta_i, :)\) is all zeros except for the \(i^{th}\) column which is equal to \(r(\delta_i)\). Thus, processor \(i\) sends \(r(\delta_i)\) of size \(\frac{2}{t} \times 1\) to its neighboring processors once \(r(\delta_i)\) is computed at step 8. Since \(r(\delta_i)\) is used in the computation at step 12, this communication is overlapped with the computations from step 9 to 11. Simultaneously, processor \(i\) receives \(r(\delta_j)\) from all its neighboring processors \(j\) for \(j = 1, 2, \ldots, m_i\). Then it computes \(A(\delta_i, \delta_i)[T(r)](\delta_j, :)\) by performing approximately \(\frac{\sum_{q} m_{MB}}{t}\) flops.

In summary, without the A-orthonormalization at steps 14 and 15, the estimated time of \(k_c\) iterations of Algorithm 4, where we ignore lower order terms, is

\[
\gamma_c(11n + 2n\frac{nnz}{t})k_c + \alpha_c(2\log(t) + m_{MB})k_c + \beta_c(\frac{n}{t}m_{MB} + 2\log(t))k_c.
\]

At iteration \(k\), the CGS2-A-CholQR A-orthonormalization requires sending \(5\log(t)\) messages with \((t + 2k + 2)\log(t)\) words and performing approximately \(12ntk + 4nt + 6n\) flops. After \(k_c\) iterations the estimated time for the A-orthonormalization is \(\gamma_c(12ntk + 16nt + 6n)k_c + \alpha_c(5\log(t))k_c + \beta_c(t + 2(k_c + 1) + 2)\log(t)k_c\). Thus, the estimated time of \(k_c\) iterations of algorithm 4 is

\[
\text{Time}_{\text{MGS+CGS}}(k_c) \approx \gamma_c(2nt\frac{nnz}{t} + 12ntk_c + 10nt + 17n)k_c + \alpha_c(7\log(t) + m_{MB})k_c + \beta_c(\frac{n}{t}m_{MB} + k^2\log(t))k_c.
\]

### 5.2 LRE-CG

In this section we describe the parallelization of the LRE-CG algorithm and we estimate its runtime in terms of flops, number of messages, and words sent. As mentioned in section 5, MGS and the CGS+TSQR orthonormalizations are numerically the most stable and allow the convergence of LRE-CG for the matrices in our test set. The parallel version of MGS orthonormalization is similar to that of the A-orthonormalization discussed in Appendix B, Algorithms 14 and 15 and requires sending \((tk + 1)\log(t)\) and \(2(t - 1)\log(t)\) messages respectively. Whereas the CGS orthonormalization can be parallelized in a block format like Algorithm 18 and requires sending \(2\log(t)\) messages. On the other hand, the TSQR orthonormalization using binary trees as discussed in 44 requires sending \(\log(t)\) messages. The combination of BCGS and TSQR was discussed in 21 and it requires sending only \(3\log(t)\) messages as compared to the \((tk + 2t - 1)\log(t)\) messages of MGS. We present the Parallel LRE-CG with BCGS+TSQR orthonormalization in Algorithm 5.

In Algorithm 5 there are two types of communication, similarly to Algorithm 4. The first type of communication is a point-to-point communication between each processor \(i\) and its \(m_i\) neighboring processors for computing the matrix - block of vectors multiplication \(Z = AW\) in line 7. We denote by \(m_{MB} = \max\{m_i\} \quad i = 1, 2, \ldots, t\) the largest number of neighboring processors, where \(m_{MB} \leq (t - 1)\) for all \(i\). Note that processor \(i\) has to compute \(A(\delta_i, \delta_i)W(\delta_i, :)\), where \(\delta_i = \text{Adjacent}(G(A), \delta_i)\). Then, the neighboring processors of a given processor \(i\) are defined as all the processors \(j\) from which processor \(i\) needs some rows of \(W\) to compute its part of \(AW\). In other words, neighboring processors are all the processors \(j\) for which \(\delta_i \cap \delta_j \neq \emptyset\). Note that for the first iteration, \(W(\delta_i, :) = [T(r)](\delta_i, :)\) is all zeros except for the \(i^{th}\) column which is equal to \(r(\delta_i)\). Thus, processor \(i\) sends \(r(\delta_i)\) of size \(\frac{2}{t} \times 1\) to
Algorithm 5  Parallel LRE-CG with BCGS+TSQR Algorithm

Input: $\delta_i$, the set of indices assigned to processor $i$; $A(\delta_i,:)$, the $\frac{n}{p} \times n$ row part of $A$

Input: $b(\delta_i)$, the $\frac{n}{p} \times 1$ row part of $b$; $x_0(\delta_i)$, the $\frac{n}{p} \times 1$ row part of $r_0$

Input: $\epsilon$, the stopping tolerance; $k_{\text{max}}$, the maximum allowed iterations

Output: $x_k(\delta_i)$, the row part of the approximate solution of $Ax = b$

1: for each processor $i = 1 : t$ in parallel do
2: Processor $i$ computes $r(\delta_i) = b(\delta_i) - A(\delta_i,\bar{\delta}_i)x_0(\bar{\delta}_i)$
3: Processor $i$ computes $r(\delta_i)^T r(\delta_i)$ and receives the full $\rho = ||r||^2_2$
4: via an all reduce (overlapped with the next computation) $\gamma_c(\frac{2nMz}{p} - 1)$
5: Let $Q(\delta_i,:) = W(\delta_i,:) = [T(r)](\delta_i,:)$, and normalise its vectors
6: Processor $i$ sends $W(\delta_i,:) = [T(r)](\delta_i,:)$ to its $m_i$ neighboring processors and receives from them the corresponding blocks
7: to obtain $W(\delta_i,:)$. Let $k = 1$
8: while $(\sqrt{p^{k-1}} > \epsilon ||b||_2$ and $k < k_{\text{max}}$) do
9: Processor $i$ computes $z(\delta_i,:) = A(\delta_i,\bar{\delta}_i)W(\bar{\delta}_i,:)$
10: Processor $i$ computes $W(\delta_i,:)^T Z(\delta_i,:)$ and receives
11: $E = W^T Z$ via an “all reduce” $\gamma_c(\frac{2nMz}{p})t^2$
12: Processor $i$ computes $Q(\delta_i, 1 : t(k-1)) = Z(\delta_i,:)$ and
13: receives $F = Q(:, 1 : t(k-1))$ via an “all reduce” $\gamma_c(2 \frac{nMz}{p}) t^2 k$
14: $E$ and $D = [D \quad Z]$
15: Processor $i$ computes $Q(\delta_i,:)r(\delta_i)$ and receives $Q^T r$ via an
16: “all reduce” $\gamma_c(2 \frac{nMz}{p}) tk$
17: $\alpha = G^{-1}(Q^T r)$
18: Processor $i$ computes $x(\delta_i) = x(\delta_i) + Q(\delta_i,:)\alpha$
19: Processor $i$ computes $r(\delta_i) = r(\delta_i) - D(\delta_i,:)\alpha$
20: via an “all reduce” (overlapped with the next computation) $\gamma_c(2 \frac{nMz}{p}) - 1$
21: Let $W = Z$ and orthogonalise the columns of $W$ against those of $Q$ using BCGS
22: Orthonormalise the vectors of $W$ using TSQR
23: Processor $i$ sends $W(\delta_i,:)$ to its $m_i$ neighboring processors and receives from them the corresponding blocks to obtain $W(\delta_i,:)$
24: end while
25: end for

its neighboring processors once $r(\delta_i)$ is computed. However, for the next iteration the $W$ is no longer sparse, therefore $W(\delta_i,:)$ of size $\frac{n}{p} \times t$ is sent.

The second type of communication is an “all reduce” that requires synchronization between all the processors, and it is equivalent to $\log(t)$ messages each of the same size (refer to [26]). For example, in lines 3 and 19 of Algorithm 5 the “all reduce” is equivalent to $\log(t)$ messages each of size 1 word.
As mentioned, this communication can be overlapped with the next computation. The reception of $E = W^t Z$, $F = Q(:, 1)$ and $F^t r$ via an “all reduce” in lines 8, 12 and 15 of Algorithm 5 is equivalent to $\log(t)$ messages each of size $t^2$ words, $\log(t)$ messages each of size $t^2(k-1)$ words, and $\log(t)$ messages each of size $tk$ words respectively. However, the three computations are independent. Thus, each processor can compute its part of the three aforementioned computations and then receive the full matrices and vectors via $\log(t)$ messages each of size $kt(t+1)$ words, assuming that it is possible to send $t^2 k$ words in one message. Another alternative is to compute $Q(\delta, 1 : (k-1)k)Z(\delta, :)\Delta$ in several steps and overlap the communication with the next computation. The number of steps depends on the machine’s architecture and on the values of $t$ and $k$.

In Algorithm 5 we show the estimated time for each computation and communication, where $\text{Time}_c(tk)$ is the estimated time for solving the $tk \times tk$ $\alpha$ system in line 16. At the $k^{th}$ iteration of Algorithm 5, the total flops, except for the $\alpha$ system, is $2mnz + (6nt - 2t^2 + 6n - t)k + 2nt + \frac{4}{3}t^3 \log(t) + 2n + 2k - 1$. And $4\log(t) + m_{MB}$ messages are sent with $((2t^2 + t)k + 1) \log(t) + nm_{MB}$ words.

Then, by ignoring the lower order terms the estimated time of $k_c$ iterations of Algorithm 5 where $t > 1$, $k_c > 1$, is

$$\text{Time}_{\text{inner}}(k_c) \approx \gamma_c(2mnz + 3ntk_c + \frac{2}{3}t^3 \log(t))k_c + \alpha_c(4\log(t) + m_{MB})k_c + \beta_c[t^2k_c\log(t) + \frac{2}{3}t^2 \log(t) + m_{MB}]k_c + \sum_{k=1}^{k_c} \text{Time}_c(tk)$$

### 6 Preconditioned enlarged Krylov subspace methods

After introducing the enlarged Krylov subspace methods and proving, theoretically and numerically, that these methods converge, we describe the preconditioned enlarged Krylov methods. A system $Ax = b$ can be left, right, or split preconditioned. In the case of conjugate gradient methods, the matrix $A$ is symmetric positive definite (SPD). Hence, the preconditioned matrix should also be SPD. For left and right preconditioning, it is not easy to find some matrix $M$ such that $M^{-1}A$ or $AM^{-1}$ is SPD. But assuming that $M = LL^t$, then the split preconditioned matrix $L^{-1}AL^{-t}$ is SPD with $L^{-t} = (L^t)^{-1}$.

Given an $n \times n$ SPD matrix $A$, $n \times 1$ vector $b$ and some preconditioner $M = LL^t$, then the split preconditioned enlarged Krylov subspace corresponding to the system $L^{-1}AL^{-t}y = L^{-1}b$ with $y = L^tx$ and $M = LL^t$, is defined by

$$\mathcal{K}_{t,k}(L^{-1}AL^{-t}, r_0) = \text{span}\{T(0), L^{-1}AL^{-t}T(0), (L^{-1}AL^{-t})^2T(0), ..., (L^{-1}AL^{-t})^{k-1}T(0)\}$$

$$= \text{span}\{T_1(0), T_2(0), ..., T_t(0), L^{-1}AL^{-t}T_1(0), L^{-1}AL^{-t}T_2(0), ..., L^{-1}AL^{-t}T_t(0), ..., (L^{-1}AL^{-t})^{k-1}T_1(0), ..., (L^{-1}AL^{-t})^{k-1}T_t(0)\},$$

where $r_0 = L^{-1}(b - AL^{-t}y_0) = L^{-1}(b - Ax_0)$, $y_0 = L^tx_0$, and $x_0$ is the initial guess.

Consequently, the split preconditioned enlarged conjugate gradient methods are defined by the orthogonality condition and the subspace condition associated with preconditioned enlarged Krylov subspace. For example, given a split preconditioned system $L^{-1}AL^{-t}y = L^{-1}b$ with $y = L^tx$ and $M = LL^t$, the the enlarged CG Krylov projection methods are defined by $y_k \in y_0 + \mathcal{K}_{t,k}(L^{-1}AL^{-t}, r_0)$ (the subspace condition), and $r_k \perp \mathcal{K}_{t,k}(L^{-1}AL^{-t}, r_0)$ (orthogonality condition), where $r_k = L^{-1}(b - AL^{-t}y_k) = L^{-1}(b - Ax_k)$. Assuming $\hat{A} = L^{-1}AL^{-t}$, $\hat{b} = L^{-1}b$, and $\hat{y} = y$, then all the theorems and properties discussed in section 3.1 are valid for the system $\hat{A}\hat{\Delta} = \hat{b}$.

Given an SPD matrix $M$, then the Cholesky factorization $M = LL^t$ can be used for split preconditioning the system $Ax = b$, where the matrix $L^{-1}AL^{-t}$ is SPD. As the Cholesky factorization of an $n \times n$ matrix can be expensive, another alternative is to use block Jacobi preconditioner with Cholesky factorization of the diagonal blocks. A four blocks Jacobi preconditioner with Cholesky decomposition...
has the following form.

\[
M = \begin{pmatrix}
M_{1,1} & 0 & 0 & 0 \\
0 & M_{2,2} & 0 & 0 \\
0 & 0 & M_{3,3} & 0 \\
0 & 0 & 0 & M_{4,4}
\end{pmatrix} = \begin{pmatrix}
L_{1,1} & 0 & 0 & 0 \\
0 & L_{2,2} & 0 & 0 \\
0 & 0 & L_{3,3} & 0 \\
0 & 0 & 0 & L_{4,4}
\end{pmatrix} = \begin{pmatrix}
L_{1,1} & 0 & 0 & 0 \\
0 & L_{2,2} & 0 & 0 \\
0 & 0 & L_{3,3} & 0 \\
0 & 0 & 0 & L_{4,4}
\end{pmatrix}
\]

We present in Algorithm 6 the split preconditioned MSDO-CG with CGS2+\(\hat{A}\)-CholQR \(\hat{A}\)-orthonormalization of the system \(\hat{A}\hat{x} = \hat{b}\), where \(\hat{A} = L^{-1}AL^{-t}, \hat{b} = L^{-1}b, \hat{x} = y,\) and \(y = L^tx\). We omit the \(W\) recursion due to numerical errors since \(W = \hat{A}P = L^{-1}AL^{-t}P\) consists of performing backward and forward substitution in addition to the matrix vector multiplication. Thus, we use a version of the CGS2 \(\hat{A}\)-orthonormalization (Algorithm 23) that computes \(W = \hat{A}P = L^{-1}AL^{-t}P\) and outputs it. As for the \(\hat{A}\)-CholQR, by assuming that \(W = \hat{A}P = L^{-1}AL^{-t}P\) is computed, then we can use Algorithm 26 with input \(W = \hat{A}P\). The additional cost of preconditioning is computing at each iteration \(k\), four times \(W_{k+1} = L^{-1}AL^{-t}P_{k+1}\) which is equivalent to a backward and forward substitution with \(t\) right hand sides and a matrix vector multiplication. The difference between the preconditioned and unpreconditioned MSDO-CG is in the \(\hat{A}\)-orthonormalization, the computation of \(W\), and the backward substitution \(L^tx = y\). Note that the split preconditioned MSDO-CG with MGS \(\hat{A}\)-orthonormalization did not converge. This might be due to numerical errors in solving the \(tk\) backward and forward substitutions in the MGS \(\hat{A}\)-orthonormalization. However, the MSDO-CG with CGS2+\(\hat{A}\)-CholQR \(\hat{A}\)-orthonormalization converges very well as shown in section 6.1.

**Algorithm 6 Split preconditioned MSDO-CG algorithm with CGS2+\(\hat{A}\)-CholQR \(\hat{A}\)-orthonormalization**

**Input:** \(A\), the \(n \times n\) symmetric positive definite matrix; \(L, n \times n\) split preconditioner

**Input:** \(b\), the \(n \times 1\) right-hand side; \(x_0\), the initial guess or iterate

**Input:** \(\epsilon\), the stopping tolerance; \(k_{\text{max}}\), the maximum allowed iterations

**Output:** \(x\), the approximate solution of \(Ax = b\) by solving for \(L^{-1}AL^{-t}y = L^{-1}b\) and \(y = L^tx\)

1: \(y = L^tx_0, r = L^{-1}(b - Ax_0), \rho = ||r||^2, nb = ||L^{-1}b||_2, k = 1\)
2: Let \(P_1 = T(r)\) and \(L^{-1}AL^{-t}\)-orthonormalize it using Algorithm 26 which outputs \(W_1 = L^{-1}AL^{-t}P_1\)
3: while \((\sqrt{\rho} > \epsilon(nb) \text{ and } k < k_{\text{max}})\) do
4: \(\alpha = (P_k^tW_k)^{-1}(P_k^t r) = P_k^t r\)
5: \(y = y + P_k\alpha\) and \(r = r - W_k\alpha\)
6: \(\rho = ||r||^2\)
7: \(\beta = -\frac{1}{W_k^t P_k W_k} W_k^t r = -W_k^t r\)
8: \(P_{k+1} = T(r) + P_k\text{diag}(\beta)\)
9: \(L^{-1}AL^{-t}\)-orthonormalize \(P_{k+1}\) against all \(P_i\)’s for \(i \leq k\) using Algorithm 23
10: \(L^{-1}AL^{-t}\)-orthonormalize \(P_{k+1}\) using Algorithm 26 which outputs \(W_{k+1} = L^{-1}AL^{-t}P_{k+1}\)
11: \(k = k + 1\)
12: end while
13: Solve \(L^tx = y\)

In Algorithm 7 we present the split preconditioned LRE-CG algorithm with BCGS+TSQR orthonormalization of the system \(\hat{A}\hat{x} = \hat{b}\), where \(\hat{A} = L^{-1}AL^{-t}, \hat{b} = L^{-1}b, \hat{x} = y,\) and \(y = L^tx\). At first glance, it might appear to the reader that the additional cost at iteration \(k\) in Algorithm 7 is solving a forward and backward substitution with \(tk\) right hand sides (\(L^{-1}AL^{-t}Q\)) and a backward and forward substitution with \(t\) right hand sides (\(L^{-1}AL^{-t}W\)). However, by taking a quick look at Algorithm 8 it is clear that the additional cost of preconditioning at iteration \(k\) is solving only a forward and backward substitution.
Algorithm 7 Split preconditioned LRE-CG Algorithm with BCGS+TSQR orthonormalization

Input: $A$, the $n \times n$ symmetric positive definite matrix; $L$, $n \times n$ split preconditioner
Input: $b$, the $n \times 1$ right-hand side; $x_0$, the initial guess or iterate
Input: $\epsilon$, the stopping tolerance; $k_{\text{max}}$, the maximum allowed iterations
Output: $x$, the approximate solution of $Ax = b$ by solving for $L^{-1}AL^{-t}y = L^{-1}b$ and $y = L^tx$

1. $y = L^tx_0$, $r = L^{-1}(b - Ax_0)$, $\rho_0 = ||r||_2^2$, $nb = ||L^{-1}b||_2$, $k = 1$
2. Let $W = T(r_0)$, normalise its vectors and then let $Q = W$
3. while ($\sqrt{\rho_{k-1}} > \epsilon(nb)$ and $k < k_{\text{max}}$) do
   4. $\alpha = (Q^tL^{-1}AL^{-t}(Q^{-1}Q^tr))$
   5. $y = y + Q\alpha$
   6. $r = r - L^{-1}AL^{-t}Q\alpha$ and $\rho_k = ||r||_2^2$
   7. Let $W = L^{-1}AL^{-t}W$
   8. Orthonormalise the vectors of $W$ against the vectors of $Q$
   9. Orthonormalise the vectors of $W$ and let $Q = [Q \ W]$ and $k = k + 1$
10. end while
11. Solve $L^tx = y$

Algorithm 8 Split preconditioned LRE-CG Pseudo Code with BCGS+TSQR orthonormalization

Input: $A$, the $n \times n$ symmetric positive definite matrix
Input: $b$, the $n \times 1$ right-hand side; $x_0$, the initial guess or iterate
Input: $\epsilon$, the stopping tolerance; $k_{\text{max}}$, the maximum allowed iterations
Output: $x_k$, the approximate solution of the system $Ax = b$

1. $y = L^tx_0$, $r = L^{-1}(b - Ax_0)$, $\rho_0 = ||r||_2^2$, $nb = ||L^{-1}b||_2$, $k = 1$
2. Let $W = T(r_0)$, normalise its vectors and then let $Q = W$
3. while ($\sqrt{\rho_{k-1}} > \epsilon(nb)$ and $k < k_{\text{max}}$) do
   4. $Z = L^{-1}AL^{-t}W$, $E = W^tZ$
   5. if $k == 1$ then
      6. $D = Z$ and $G = E$
   7. else
      8. $D = [D \ Z]$, $F = Q(:,1:t(k-1)):t Z$, and $G = \begin{pmatrix} G & F \\ F^t & E \end{pmatrix}$
   9. end if
10. $\alpha = G^{-1}(Q^t r)$
11. $y = y + Q\alpha$
12. $r = r - D\alpha$ and $\rho_k = ||r||_2^2$
13. Let $W = Z$
14. Orthonormalise the columns of $W$ against those of $Q$ using BCGS
15. Orthonormalise the vectors of $W$ using TSQR and let $Q = [Q \ W]$ and $k = k + 1$
16. end while
17. Solve $L^tx = y$

with $t$ right hand sides ($L^{-1}AL^{-t}W$). And this is the only difference with the unpreconditioned version in addition to the backward substitution $L^tx = y$ at the end.
6.1 Convergence

We compare the convergence of split preconditioned MSDO-CG with CGS2+CholQR A-orthonormalization and split preconditioned LRE-CG with CGS+TSQR orthonormalization to CG and split preconditioned CG (PCG). We use Block Jacobi with Cholesky factorization of the block diagonals as a preconditioner.

Table 6: Comparison of the convergence of the split preconditioned CG, MSDO-CG with CGS2+CholQR A-orthonormalization, and LRE-CG with CGS+TSQR orthonormalization method with varying Block Jacobi preconditioners, with respect to number of partitions Pa, with x_0 = 0.

<table>
<thead>
<tr>
<th>Pa</th>
<th>Iter</th>
<th>Err</th>
<th>Iter</th>
<th>Err</th>
<th>Iter</th>
<th>Err</th>
<th>Iter</th>
<th>Err</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>195</td>
<td>2E-5</td>
<td>35</td>
<td>1E-5</td>
<td>2E-6</td>
<td>30</td>
<td>2E-6</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1E-5</td>
<td>2E-6</td>
<td>28</td>
<td>30</td>
<td>6E-6</td>
<td>27</td>
<td>2E-6</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>50</td>
<td>1E-5</td>
<td>2E-6</td>
<td>28</td>
<td>1E-6</td>
<td>25</td>
<td>1E-6</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>37</td>
<td>2E-5</td>
<td>2E-7</td>
<td>26</td>
<td>8E-7</td>
<td>23</td>
<td>5E-7</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>66</td>
<td>2E-5</td>
<td>2E-7</td>
<td>23</td>
<td>1E-6</td>
<td>20</td>
<td>3E-7</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In table 6, we use a different Block Jacobi preconditioner for the different partitions. First, the graph of A is partitioned into t parts that define the enlarged Krylov subspace using Metis's kway edge separator where t = 2, 4, 8, 16, 32, 64. Then the Block Jacobi preconditioner M is defined as the t diagonal blocks of the permuted matrix A. Each of the t blocks is factorized using Cholesky decomposition to obtain a block L. The preconditioned LRE-CG converges faster than the preconditioned MSDO-CG and PCG.
for the different configurations. As the number of partitions or the maximum basis vectors added at each iteration is doubled, the Block Jacobi preconditioned CG needs more iterations to converge. However, for the matrices \texttt{POISSON2D}, \texttt{NH2D}, and \texttt{SKY2D}, the number of iteration of the preconditioned LRE-CG and MSDO-CG decreases. As for the matrices \texttt{SKY3D} and \texttt{AN13D}, the number of iterations of LRE-CG increases then decreases back to the same number of iterations for \( t = 2 \), unlike preconditioned MSDO-CG.

Table 7: Comparison of the convergence of the split preconditioned CG, MSDO-CG with CGS2+CholQR \( \tilde{A} \)-orthonormalization, and LRE-CG with CGS+TSQR orthonormalization method with a fixed Block Jacobi preconditioner, with respect to number of partitions \( Pa \), with \( x_0 = 0 \).

<table>
<thead>
<tr>
<th>\texttt{POISSON2D} ( tol = 10^{-6} )</th>
<th>\texttt{NH2D} ( tol = 10^{-8} )</th>
<th>\texttt{SKY2D} ( tol = 10^{-8} )</th>
<th>\texttt{SKY3D} ( tol = 10^{-8} )</th>
<th>\texttt{AN13D} ( tol = 10^{-8} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Pa )</td>
<td>\text{iter}</td>
<td>\text{err}</td>
<td>\text{iter}</td>
<td>\text{err}</td>
</tr>
<tr>
<td>2</td>
<td>195</td>
<td>2E-5</td>
<td>66</td>
<td>2E-5</td>
</tr>
<tr>
<td>4</td>
<td>54</td>
<td>9E-6</td>
<td>50</td>
<td>8E-6</td>
</tr>
<tr>
<td>8</td>
<td>37</td>
<td>3E-6</td>
<td>33</td>
<td>1E-6</td>
</tr>
<tr>
<td>16</td>
<td>25</td>
<td>8E-7</td>
<td>20</td>
<td>3E-7</td>
</tr>
<tr>
<td>32</td>
<td>28</td>
<td>7E-9</td>
<td>23</td>
<td>4E-9</td>
</tr>
<tr>
<td>64</td>
<td>20</td>
<td>1E-9</td>
<td>19</td>
<td>8E-10</td>
</tr>
</tbody>
</table>

In Table 7 we use a fixed Block Jacobi preconditioner for all the partitions to compare the convergence of the methods with respect to the doubling of the number of partitions \( \ell \). First, the graph of \( \tilde{A} \) is partitioned into 64 parts using Metis’s kway edge separator. Then the Block Jacobi preconditioner \( M \) is defined as the 64 block diagonals of the permuted matrix \( A \). Each of the 64 blocks is factorized using Inria
Cholesky decomposition to obtain a block $L$. Then the matrix $A$ is partitioned once again using kway into $t = 2, 4, 8, 16, 32$ or 64 parts that define the enlarged Krylov subspace. The preconditioner is permuted accordingly. The preconditioned LRE-CG converges faster than the preconditioned MSDO-CG and PCG. As the number of partitions or the maximum basis vectors added at each iteration is doubled, the preconditioned LRE-CG and MSDO-CG converge faster. However, for some matrices, like POISSON2D, NH2D, and AN13D, as the number of partitions is doubled, the number of iterations till convergence decreases by only $10\% - 20\%$. Thus, it is efficient to use a maximum of $t = 4$ partitions which correspond to adding at most $t$ vectors to the basis of the enlarged Krylov subspace. For the matrices SKY2D and SKY3D, the number of iterations decreases by $33\% - 45\%$ and $25\% - 45\%$ respectively. Hence it is possible to use a maximum of $t = 8$ partitions or at most $t = 16$.

7 Conclusion

In this paper we have introduced two new iterative methods, MSDO-CG and LRE-CG, which are based on the enlarged Krylov subspace. After introducing the related existing methods (Block-CG, Coop-Cg and MCD-CG), we have defined the properties of the enlarged Krylov subspace, derived the new methods in the context of projection CG versions, provided parallel versions that reduce communication, and shown that the methods converge at least as fast as Classical CG in exact precision arithmetic. The convergence results show that they also converge faster than CG in finite precision arithmetic. We have also presented the preconditioned versions and tested their convergence with block Jacobi preconditioner.

MSDO-CG is a variation of the MSD-CG version, where we A-orthonormalize the $t$ search directions against previous directions and against each others. Due to the A-orthonormalization, we lose the short recurrence property of CG and we are obliged to save all the $tk_c$ search directions, where $k_c$ is the number of iterations till convergence. In LRE-CG we start by building an orthonormal basis for the enlarged Krylov subspace, then we use the whole basis to update the solution. The main difference between both methods in terms of performance, is that at each iteration of MSDO-CG, we use $t$ search directions to update the new approximate solution. Whereas in LRE-CG, at each iteration $i$, we use the entire basis formed by $ti$ vectors, to update the approximate solution and we solve a $ti \times ti$ system. However, this use of the whole basis leads to a relatively faster convergence than MSDO-CG. One way to limit this increasing cost is by restarting LRE-CG after some iterations. Another alternative is to choose at each iteration $i$, a linearly independant subset of the $ti$ computed vectors. This adds an extra cost, but reduces the size of the system that has to be solved at each iteration. A third alternative is to compute $ti$ vectors at each iteration $i$, where $t_0 = t$, and $t_i \leq t$. Then choose $\tilde{t}_i$ linearly independant vectors where $\tilde{t}_i \leq t_i$, and $t_{i+1} = \tilde{t}_i$.

Although each iteration of the MSDO-CG and LRE-CG methods is at least $t$ times more expensive than the CG iteration in terms of flops, as shown in section 5 both methods use less communication, and Blas2 and Blas3 operations that can be parallelized in a more efficient way than the dot products in CG. Moreover, the MSDO-CG and LRE-CG methods converge faster than CG in terms of iterations as shown in section 4.

Our future work will focus on testing the LRE-CG versions discussed above, that are less expensive in terms of flops and memory requirements than LRE-CG, like restarted LRE-CG or LRE-CG with selected basis vectors. Then, the most stable version will be implemented in a parallel environment. We will also test LRE-CG on other real applications’ matrices, and with different preconditioners. Moreover, we would also like to compare the runtime of the LRE-CG version with the MSDO-CG method on a parallel environment. We will also derive and test other enlarged Krylov methods, like enlarged GMRES which has been derived but not tested yet.
References


Appendix A  MCD-CG Algorithm

Algorithm 9 The MSD-CG Algorithm

| Input: | A, the $n \times n$ symmetric positive definite matrix |
| Input: | b, the $n \times 1$ right-hand side; $x_0$, the initial guess or iterate |
| Input: | $\epsilon$, the stopping tolerance; $k_{\text{max}}$, the maximum allowed iterations |
| Output: | $x_k$, the approximate solution of the system $Ax = b$ |

1: $r_0 = b - Ax_0$, $\rho = ||r_0||^2_2$, $k = 1$
2: for $i = 1, \ldots, t$ do $P(:, i) = T_i(r_0)$
3: end for
4: while $(\sqrt{\rho} > \epsilon ||b||_2$ and $k < k_{\text{max}}$) do
5:   $\alpha = (P^tAP)^{-1}(P^t\rho)$
6:   $x = x + P\alpha$
7:   $r = r - AP\alpha$
8:   $\beta = (P^tAP)^{-1}(P^tAr)$
9:   for $i = 1, \ldots, t$ do $P(:, i) = T_i(r) + \beta(i)P(:, i)$
10: end for
11: $\rho = ||r||^2_2$
12: $k = k + 1$
13: end while

Appendix B  The A-orthogonalization of search directions

In this section we describe the A-orthogonalization of the $t$ newly computed vectors $P_{k+1}$ against all the previous vectors $P_i$’s for $i < k+1$, and then against each others. Here $A$ is a symmetric positive definite matrix. The A-orthogonalization is simply an orthogonalization with the A inner product ($\langle \cdot, \cdot \rangle_A = \langle \cdot, \cdot \rangle$) rather than the L2 inner product ($\langle \cdot, \cdot \rangle$). A-orthonormalizing a tall and skinny $n \times t$ matrix $P_{k+1}$, or alternatively computing the oblique QR factorization of $P_{k+1}$, has been discussed in [24] and [20] in terms of stability and ease of parallelization. The goal is to get a $\hat{P}_{k+1}$, such that $\hat{P}_{k+1}^tA\hat{P}_{k+1} = I$.

There are two main classes for computing this oblique QR factorization of $P_{k+1} = \hat{P}_{k+1}R$. The first class is to factorize the matrix $A = B^tB$ using Cholesky decomposition or eigenvalue decomposition, which is expensive. Then $P_{k+1}^tAP_{k+1} = (BP_{k+1})^t(BP_{k+1})$, where the oblique QR factorization of $P_{k+1}$ is transformed into a Euclidean QR factorization of the matrix $BP_{k+1} = Q_{B}R_{B}$ with $\hat{P}_{k+1} = B^{-1}Q_{B}$ and $R = R_{B}$. The second class consists of avoiding any factorization of $A$, like CGS, CGS2, MGS, and the Cholesky factorization of the $t \times t$ matrix $P_{k+1}^tAP_{k+1}$. For A-orthonormalizing $P_{k+1}$ against all the previous vectors $P_i$ with $i < k + 1$, it is possible to use CGS, CGS2, MGS and A-choleskyBGS which was discussed in Hoemmen’s thesis ([17], page 115).

Thus, we start by discussing the A-orthonormalization using modified Gram Schmidt in section B.1. However, this version is not easily parallelized and requires a lot of communication ($((t + 1)\log(t) + 2(t - 1)\log(t))$ messages) as compared to the classical Gram Schmidt version. Then, in section B.2, we adapt the A-orthonormalization of the vectors of $P_{k+1}$ against $P_i$’s for $i < k + 1$ using the classical Gram Schmidt (CGS) to obtain a Block Gram Schmidt (BGS) version (Algorithm 14) with a inner product that requires only $2 \log(t)$ messages. As for the A-orthonormalization of the $P_{k+1}$ vectors against each others, we introduce a parallelizable version with reduced communication ($(2t - 1)\log(t)$ messages). Note that CGS2, section B.2.3, consists of calling the algorithm CGS twice. Thus its cost is twice the cost of CGS. In section B.3, we briefly discuss the A-orthonormalization of $P_{k+1}$ using the Cholesky factorization (CholQR) of the $t \times t$ matrix $P_{k+1}^tAP_{k+1}$ which is referred to as A-CholQR and requires

Inria

L. Grigori & S. Moufawad & F. Nataf
only $\log(t)$ messages. We also present the Pre-CholQR version that was introduced in [20] and requires $3\log(t)$ messages.

### B.1 Modified Gram Schmidt A-orthonormalization

We start by introducing A-orthonormalization of the vectors of $P_{k+1}$ against the vectors of all the previous $P_i$’s for $i < k + 1$, against each others in the first section. In the second section, we discuss versions that save flops and reduce communication. Finally, in the last section the parallelization of both kernels is described.

#### B.1.1 The A-orthonormalization using MGS

Assuming that the vectors of $P_i$ are A-normalized, i.e. $P_i(:, j)^t A P_i(:, j) = 1$ for all $j = 1, 2, \ldots, t$ and $i = 1, 2, \ldots, k$, then the A-orthonormalization of the vectors of $P_{k+1}$ against the vectors of all the previous $P_i$’s for $i < k + 1$ is defined as follows:

**Algorithm 10** A-orthonormalization against previous vectors with MGS

**Input:** $A$, the $n \times n$ symmetric positive definite matrix  
**Input:** $P_1, P_2, \ldots, P_{k+1}$, the $k + 1$ sets of search directions  
**Output:** $P_{k+1}$, the search directions A-orthonormalized against $P_1, P_2, \ldots, P_k$

1: for $\alpha = 1 : t$ do  
2: for $i = 1 : k$ do  
3: for $j = 1 : t$ do  
4: $P_{k+1}(:, \alpha) = P_{k+1}(:, \alpha) - (P_i(:, j)^t A P_{k+1}(:, \alpha)) P_i(:, j)$  
5: end for  
6: end for  
7: $P_{k+1}(:, \alpha) = \frac{P_{k+1}(:, \alpha)}{\|P_{k+1}(:, \alpha)\|_A}$  
8: end for

At each inner iteration, one matrix-vector multiplication has to be computed ($AP_{k+1}(:, \alpha)$), 1 dot product, and 1 saxpy, which costs $2nnz - n + (2n - 1) + 2n = 2nnz + 3n - 1$ flops. Then, at each outermost iteration, one matrix-vector multiplication is computed ($AP_{k+1}(:, \alpha)$), 1 dot product, 1 square root and 1 division which costs $2nnz - n + (2n - 1) + 2 = 2nnz + n + 1$. The total cost of Algorithm 10 is $(2nnz + 3n - 1)t^2k + (2nnz + n + 1)t$, which is of the order of $nnz t^2 k + nt^2k$.

As for the A-orthonormalization of the vectors of $P_{k+1}$ against each others, it is defined as follows:

**Algorithm 11** A-orthonormalization against each others using MGS

**Input:** $A$, the $n \times n$ symmetric positive definite matrix  
**Input:** $P_{k+1}$, the search directions to be A-orthonormalized  
**Output:** $P_{k+1}$, the A-orthonormalized search directions

1: for $i = 1 : t$ do  
2: for $j = 1 : (i - 1)$ do  
3: $P_{k+1}(:, i) = P_{k+1}(:, i) - (P_{k+1}(:, j)^t A P_{k+1}(:, i)) P_{k+1}(:, j)$  
4: end for  
5: $P_{k+1}(:, i) = \frac{P_{k+1}(:, i)}{\|P_{k+1}(:, i)\|_A}$  
6: end for

Similarly, the cost of the inner loop is $2nnz + 3n - 1$ flops and that of the outer loop is $2nnz + n + 1$, but the total cost is $(2nnz + 3n - 1)(t - 1)\frac{t}{2} + (2nnz + n + 1)t$ flops, which is of the order of $nnz t^2 + nt^2$.
B.1.2 Saving flops in the A-orthonormalization using MGS

Since the A-orthonormalizations are expensive in term of flops, we present another alternative for computing the A-orthonormalizations that reduces the computed flops at the expense of storing more vectors. In Algorithm 11 and Algorithm 10, some matrix vector multiplications are repeatedly computed. For example, in Algorithm 11, $AP_{k+1}(;i)$ is computed $t-1$ times, $AP_{k+1}(;i)$ is computed $t-2$ times and generally, $AP_{k+1}(;i)$ is computed $t-i$ times, which means that the matrix $A$ is accessed $(t-1)\frac{i}{2}$ times for every call of the algorithm. Thus, it is possible after A-orthogonalizing a vector $P_{k+1}(;i)$ to compute and store $w_i = AP_{k+1}(;i)$. This eliminates the redundant flops and reduces the number of accesses of $A$ to $t$ times, but there is a need to store $t$ extra vectors ($w_i$).

Moreover, it is possible to further reduce the computations and the number of times $A$ is accessed at the expense of storing $tk$ vectors as shown in Algorithm 12 where the multiplication $W_{k+1} = AP_{k+1}$ is first performed by only reading the matrix $A$ once. Then the vectors $W_{k+1}(;i)$ are updated and stored.

The A-orthonormalization against previous vectors with flops reduction can be performed as follows:

Algorithm 12 A-orthonormalization against previous vectors with MGS

\begin{verbatim}
Input: $P_1, P_2, \ldots, P_{k+1}$, the $k+1$ sets of search directions
Input: $W_1, W_2, \ldots, W_{k+1}$, the $k+1$ sets of $AP_i$

Output: $P_{k+1}$, the search directions A-orthonormalized against $P_1, P_2, \ldots, P_k$

1: for $o = 1 : t$ do %loop over the vectors of $P_{k+1}$
2:   for $i = 1 : k$ do %loop over the different $P_i$’s
3:     for $j = 1 : t$ do %loop over the vectors of $P_i$
4:        $P_{k+1}(; o) = P_{k+1}(; o) - (W_i(;j)^tP_{k+1}(;o))P_{k+1}(;j)$
5:        $W_{k+1}(; o) = W_{k+1}(; o) - (W_i(;j)^tP_{k+1}(;o))W_i(;j)$
6:     end for
7:   end for
8: end for

$P_{k+1}(; o) = \frac{P_{k+1}(;o)}{\sqrt{pp_{k+1}}}$ and $W_{k+1}(; o) = \frac{W_{k+1}(;o)}{\sqrt{pp_{k+1}}}$

10: end for
\end{verbatim}

Then, the cost of the A-orthonormalization against previous vectors in Algorithm 12 is $(6n-1)t^2k + (4n+1)t$ of the order of $6nt^2k$ flops.

Algorithm 13 Flops reduction in A-orthonormalization against each others with MGS

\begin{verbatim}
Input: $P_{k+1}$, the search directions to be A-orthonormalized
Input: $W_{k+1}, AP_{k+1}$

Output: $P_{k+1}$, the A-orthonormalized search directions
Output: $W_{k+1}, AP_{k+1}$ where $P_{k+1}$ is the A-orthonormalized search directions

1: for $i = 1 : t$ do %loop over the vectors of $P_{k+1}$
2:   for $j = 1 : (i-1)$ do %A-orthogonalize against the vectors $P_{k+1}(;1:i-1)$
3:      $P_{k+1}(; i) = P_{k+1}(; i) - (W_{k+1}(;j)^tP_{k+1}(;i))P_{k+1}(;j)$
4:      $W_{k+1}(; i) = W_{k+1}(; i) - (W_{k+1}(;j)^tP_{k+1}(;i))W_{k+1}(;j)$
5:   end for
6: $pp_{k+1} = W_{k+1}(; i)^tP_{k+1}(; i)$
7: $P_{k+1}(; i) = \frac{P_{k+1}(;i)}{\sqrt{pp_{k+1}}}$ and $W_{k+1}(; i) = \frac{W_{k+1}(;i)}{\sqrt{pp_{k+1}}}$
8: end for
\end{verbatim}
As for the A-orthonormalization of $P_{k+1}$’s vectors against each others using MGS with flops reductions, it can be performed as in Algorithm 13. The cost of this version of A-orthonormalization (Algorithm 13) is $(6n-1)(t-1)\frac{t}{2} + (4n+1)t$, which is the order of $3nt^2$.

B.1.3 Parallelization of the A-orthonormalization using MGS

In Algorithm 12 at each inner iteration we are A-orthonormalizing the updated vectors $P_{k+1}(;i,o)$ against the vector $P_i(;j)$, where the vector $P_{k+1}(;i,o)$ is changed at each inner iteration. Thus it is not possible to have a block MGS by eliminating all the for loops. However, it is possible to eliminate one for loop in Algorithm 12 as shown in Algorithm 14, by A-orthonormalizing the whole block $P_{k+1}$ against the vector $P_i(;j)$, where $P_{k+1}(;o) = P_{k+1}(;o) - (P_{k+1}(;o)^TP_i(;j))P_i(;j)$ for all $o = 1, 2, ..., t$. Let $[P_i(;j)]_t$ be an $n \times t$ block containing $t$ duplicates of the vector $P_i(;j)$. Then, $P_{k+1} = P_{k+1} - [P_i(;j)]_t \text{diag}(P_{k+1,1}^TP_i(;j))$.

Algorithm 14 A-orthonormalization against previous vectors with MGS Flops

\begin{itemize}
  \item Input: $P_1, P_2, ... , P_{k+1}$, the $k + 1$ sets of search directions
  \item Input: $W_1, W_2, ... , W_{k+1}$, the $k + 1$ sets of $AP_i$
  \item Output: $P_{k+1}$, the search directions A-orthonormalized against $P_1, P_2, ..., P_k$
  \begin{enumerate}
    \item for $i = 1 : k$ do \%loop over the different $P_i$’s
    \item for $j = 1 : t$ do \%loop over the vectors of $P_i$
    \item $P_{k+1} = P_{k+1} - [P_i(;j)]_t \text{diag}(W_i(;j)^TP_{k+1})$ \hfill $(4n-1)t$
    \item $W_{k+1} = W_{k+1} - [W_i(;j)]_t \text{diag}(W_i(;j)^TP_{k+1})$ \hfill $2nt$
    \item end for
    \item end for
  \end{enumerate}

  In Algorithm 13 rather than A-orthonormalizing each vector $P_{k+1}(;i)$ against all previous vectors $P_{k+1}(;j)$, we can A-orthogonalize $P_{k+1}(;i)$ against the A-normalised vector $P_{k+1}(;i)$ as shown in Algorithm 15. Let $[P_{k+1}(;j)]_{t-1}$ be an $n \times (t - i)$ block containing $t-i$ duplicates of the vector $P_{k+1}(;j)$. Then $P_{k+1}(;i+1 : t) = P_{k+1}(;i+1 : t) - [P_{k+1}(;j)]_{t-1} \text{diag}(W_{k+1}(;i)^TP_{k+1}(;i+1 : t))$

Algorithm 15 A-orthonormalization against each others with MGS

\begin{itemize}
  \item Input: $P_{k+1}$, the search directions to be A-orthonormalized
  \item Input: $W_{k+1}, AP_{k+1}$
  \item Output: $P_{k+1}$, the A-orthonormalized search directions
  \item Output: $W_{k+1}, AP_{k+1}$ where $P_{k+1}$ is the A-orthonormalized search directions
  \begin{enumerate}
    \item for $i = 1 : t - 1$ do \%A-orthonormalize against the vectors $P_{k+1}(;1 : i - 1)$
    \item $P_{k+1}(;i+1 : t) = P_{k+1}(;i+1 : t) - [P_{k+1}(;i)]_{t-1} \text{diag}(W_{k+1}(;i)^TP_{k+1}(;i+1 : t))$
    \item $W_{k+1}(;i+1 : t) = W_{k+1}(;i+1 : t) - [W_{k+1}(;i)]_{t-1} \text{diag}(W_{k+1}(;i)^TP_{k+1}(;i+1 : t))$
    \item $pap_{k+1} = \sqrt{pap_{k+1}}$ \%loop over the vectors of $P_i$
    \item $P_{k+1}(;i+1) = \frac{P_{k+1}(;i+1)}{\sqrt{pap_{k+1}}}$ and $W_{k+1}(;i+1) = \frac{W_{k+1}(;i+1)}{\sqrt{pap_{k+1}}}$
    \item end for
  \end{enumerate}

  Then the parallelization of Algorithms 14 and 15 goes as follows. We assume that we have $t$ proces-

RR n° 8597
sors with distributed memory, and each processor \( p_i \) is assigned a rowwise part of all \( W_j \) \( (W_j(\delta_{pi},::)) \) for \( j = 1, 2, \ldots, k + 1 \) and the same rowwise part of all \( P_j \) \( (P_j(\delta_{pi},::)) \) for \( j = 1, 2, \ldots, k + 1 \) where \( \delta_{pi} \cap \delta_{p} = \emptyset \) for all \( p_i \neq h \) and \( \cup_{h=1}^{n} \delta_{h} = \{1, 2, 3, \ldots, n\} \).

At each inner iteration of Algorithm 14 each processor \( p_i \) has to compute \( P_{k+1}(\delta_{pi},::) = P_{k+1}(\delta_{pi},::) - \{P_i(\delta_{pi},::)\} diag(W_i(:,j)^t P_{k+1}(:,j)) \). First, each processor \( p_i \) computes a part of the matrix vector multiplication \( W_i(\delta_{pi},::)^t P_{k+1}(\delta_{pi},::) \). Then, a communication of the form “all reduce” is performed to send the \( 1 \times t \) \( W_i(:,j)^t P_{k+1}(:,j) \)’s value to all the processors. Finally, processor \( p_i \) computes \( P_{k+1}(\delta_{pi},::) \) and \( W_{k+1}(\delta_{pi},::) \).

Finally, each processor \( p_i \) computes its corresponding part of the dot product \( W_{k+1}(\delta_{pi},::)^t P_{k+1}(\delta_{pi},::) \) for all \( o = 1, 2, \ldots, t \) and an “all reduce” is used to send \( p_{ap_{k+1}} \)’s value to all the processors. Then, each processor A-normalizes \( P_{k+1}(\delta_{pi},::) \) and \( W_{k+1}(\delta_{pi},::) \). All the communication in Algorithm 14 is of the form “all reduce” of a \( t \times 1 \) vector which is equivalent to sending \( \log(t) \) messages and \( t \log(t) \) words. So, in total \((t+1)(\log(t)) \) messages and \((tk+1)\log(t) \) words are sent in Algorithm 12.

Hence, by ignoring lower order terms we obtain

\[
T_{\text{Time}_{MGS_{1 \text{sort}}}} \approx \gamma_n t^n k + \alpha_o t_k \log(t) + \beta_o t^2 \log(t)
\]

As for the parallelization of algorithm 15, it is similar to that of algorithm 14 where at each inner iteration processor \( p_i \) computes a part of the matrix vector multiplication \( W_{k+1}(\delta_{pi},i)^t P_{k+1}(\delta_{pi},i+1:j) \) and then receives the whole \( 1 \times t \) vector, \( W_{k+1}(\delta_{pi},i)^t P_{k+1}(\delta_{pi},i+1:t) \), using an “all reduce”. Then, it computes \( P_{k+1}(\delta_{pi},::), W_{k+1}(\delta_{pi},::) \) and a part of the dot product \( p_{ap_{k+1}} \) and receives the whole dot product by an “all reduce”. Finally, each processor A-normalizes its part of \( P_{k+1}(\delta_{pi},i) \) and \( W_{k+1}(\delta_{pi},i) \). Thus, at each iteration 2 “all reduce” communications are performed, where \( t \) words are sent in the first and one word in second. So, in total \((t-1)(t+1)\log(t) \) words are sent. Hence, by ignoring lower order terms we obtain

\[
T_{\text{Time}_{MGS_{2 \text{sort}}}} \approx \gamma_n t^n k + \alpha_o t_k \log(t) + \beta_o t^2 \log(t)
\]

### B.2 Classical Gram Schmidt A-orthonormalization

Since the MGS A-orthonormalization is costly in terms of communication, we introduce the classical Gram Schmidt (CGS) A-orthonormalization and show that it is equivalent to a QR decomposition with A inner product rather than the usual L2 inner product. Then we present the parallelization of the introduced algorithms. In section B.2.1, the A-orthonormalization against previous vectors using CGS is discussed, whereas in section B.2.2 we discuss the A-orthonormalization of the vectors using CGS. Then in section B.2.3 we introduce the CGS A-orthonormalization with reorthogonalization.

#### B.2.1 A-orthonormalization against previous vectors using CGS

The A-orthonormalization of \( P_{k+1} \) against the vectors of all the previous \( P_i \)’s for \( i < k + 1 \) is defined as in Algorithm 16.

More precisely, \( \tilde{P}_{k+1}(::,o) = \tilde{P}_{k+1}(::,o) - \sum_{i=1}^{k} \sum_{j=1}^{t} (P_i(:,j)^t AP_{k+1}(::,o)) P_i(:,j) \)

\( = P_{k+1}(::,o) - \sum_{i=1}^{k} P_i (P_i^t AP_{k+1}(::,o)) \)

If we let \( W_{k+1} = AP_{k+1} \), then \( \tilde{P}_{k+1}(::,o) = P_{k+1}(::,o) - \sum_{i=1}^{k} P_i (P_i^t W_{k+1}(::,o)) \).

Moreover, \( \tilde{P}_{k+1} = P_{k+1} - \sum_{i=1}^{k} P_i (P_i^t W_{k+1}(::,o)) \).

Let \( Q_k = [P_1, P_2, \ldots, P_k] \), then \( \tilde{P}_{k+1} = P_{k+1} - Q_k (Q_k^t W_{k+1}) \).

This represents a Block classical gram schmidt (BCGS) version of the A-orthonormalization.

The total flops performed in Algorithm 17 is \( 2(2nnz-n)t + (2n-1)2t^2 k + 2t^2 k n + 3nt + 4ntz + n^2 t + 2^t k \approx 4nnt + 4nt^2 t + 4nt^2 k \).

As for the parallelization of Algorithm 17, it is straightforward due to the block format. Assuming that we have \( t \) processors with distributed memory, and each processor \( p_i \) is assigned a rowwise part of \( A(\delta_{pi},::) \), a rowwise part of \( Q_k (Q_k(\delta_{pi},::)) \) and a rowwise part of \( P_{k+1} \), where \( \delta_{pi} \cap \delta_{h} = \emptyset \) for all \( p_i \neq h \) and \( \cup_{h=1}^{n} \delta_{h} = \{1, 2, 3, \ldots, n\} \).
Algorithm 16 A-orthonormalization against previous vectors with CGS

**Input:** A, the \( n \times n \) symmetric positive definite matrix  
**Input:** \( P_1, P_2, \ldots, P_k \), the \( k+1 \) sets of search directions  
**Output:** \( \hat{P}_{k+1} \), the search directions A-orthonormalized against \( P_1, P_2, \ldots, P_k \)

1. Let \( \hat{P}_{k+1} = P_{k+1} \)
2. for \( i = 1 : t \) do  
   %loop over the vectors of \( P_{k+1} \)
3.   for \( i = 1 : k \) do  
   %loop over the different \( P_i \)’s
4.     for \( j = 1 : t \) do  
   %loop over the vectors of \( P_i \)
5.       \( \hat{P}_{k+1}(:, o) = \hat{P}_{k+1}(:, o) - (P_i(:, j)^T A P_{k+1}(:, o)) P_i(:, j) \)
6.    end for
7. end for
8. \( \hat{P}_{k+1}(:, o) = \frac{\hat{P}_{k+1}(:, o)}{||\hat{P}_{k+1}(:, o)||_A} = \frac{\hat{P}_{k+1}(:, o)}{\sqrt{\hat{P}_{k+1}(:, o)^T A \hat{P}_{k+1}(:, o)}} \) %A-normalize
9. end for

Algorithm 17 A-orthonormalization against previous vectors with BCGS

**Flops**  
**Input:** A, the \( n \times n \) symmetric positive definite matrix  
**Input:** \( Q_k = [P_1, P_2, \ldots, P_k] \), the \( tk \) search directions  
**Input:** \( P_k \), the \( t \) search directions to be A-orthonormalized  
**Output:** \( \hat{P}_{k+1} \), the search directions A-orthonormalized against \( P_1, P_2, \ldots, P_k \)

1. \( W_{k+1} = AP_{k+1} \)  
\( (2mnz - n)t \)
2. \( \hat{P}_{k+1} = P_{k+1} - Q_k(Q_k^T W_{k+1}) \)  
\( (2n-1)t^2k + (2tk-1)nt + nt \)
3. \( \hat{W}_{k+1} = AP_{k+1} \)  
\( (2mnz - n)t \)
4. for \( i = 1 : t \) do  
   %loop over the vectors of \( P_{k+1} \) and A-normalize
5.     \( \hat{P}_{k+1}(:, i) = \frac{\hat{P}_{k+1}(:, i)}{||\hat{P}_{k+1}(:, i)||_A} = \frac{\hat{P}_{k+1}(:, i)}{\sqrt{\hat{P}_{k+1}(:, i)^T W_{k+1}(:, i)}} \)  
\( 3n \)
6.    end for

First, processor \( p_i \) computes \( A(:, \delta_{pi}) P_{k+1}(:, \delta_{pi}, :) \) and receives the full \( n \times t \) matrix \( W_{k+1} \) via an “all reduce”. Then it computes \( Q_k(Q_k^T W_{k+1}) \) and obtains the full \( tk \times t \) matrix \( Q_k^T W_{k+1} \) using an “all reduce”. Then, processor \( p_i \) computes \( \hat{P}_{k+1}(:, \delta_{pi}, :) = P_{k+1}(:, \delta_{pi}, :) - Q_k(\delta_{pi}, :) (Q_k^T W_{k+1}) \). Another “all reduce” is needed so that processor \( p_i \) has the full \( \hat{P}_{k+1} \) needed to compute \( \hat{W}_{k+1} = A(\delta_{pi}, :) \hat{P}_{k+1} \). Processor \( p_i \) computes \( t \) partial dot products of the form \( \hat{P}_{k+1}(:, \delta_{pi}, :) \hat{W}_{k+1}(:, \delta_{pi}, :) \) and obtains the full dot products via an all reduce. Finally each processor A-normalizes its part of \( \hat{P}_{k+1} \), i.e. \( \hat{P}_{k+1}(:, \delta_{pi}, :) = \frac{\hat{P}_{k+1}(:, \delta_{pi}, :) \hat{W}_{k+1}(:, \delta_{pi}, :)^T}{||\hat{P}_{k+1}(:, \delta_{pi}, :)||_A} \) for all \( i = 1, 2, \ldots, t \). So in total there is a need to perform 4 all reduce for parallelizing algorithm[17]

It is possible to reduce the communication to only two by assuming that \( W_{k+1} = A P_{k+1} \) has already been computed and it is an input to Algorithm 18 along with \( W_k = AQ_k = [W_1, W_2, \ldots, W_k] \). The only communication is an “all reduce” of the \( tk \times t \) matrix \( Q_k^T W_{k+1} \) and another “all reduce” of the vector of size \( t \) containing the norms of the columns of \( \hat{P}_{k+1} \). We assume that it is possible to send a message of size \( t^2k \) words at once. Thus, \( 2\log(t) \) messages are sent with \( (tk + 1)\log(t) \) words where \( \frac{(6n-1)t^2k + 3nt}{t} = (6n-1)tk + 3n \) flops are performed in parallel. Hence, by ignoring lower order terms we obtain

\[ \text{Time}_{BCGS,\text{A-ort}} \approx \gamma_n 6ntk + \alpha_c 2\log(t) + \beta_c t^2k\log(t) \]

RR n° 8597
Algorithm 18 A-orthonormalization against previous vectors with BCGS

**Input:** \( Q_k = [P_1, P_2, ..., P_k] \), the \( t \)k search directions
**Output:** \( \tilde{P}_{k+1} \), the \( t \) search directions to be A-orthonormalized
**Output:** \( \tilde{W}_{k+1} = AQ_k \)

```
1: \( \tilde{P}_{k+1} = P_{k+1} - Q_k(Q_k^t W_{k+1}) \)
2: \( \tilde{W}_{k+1} = W_{k+1} - Q_k(Q_k^t W_{k+1}) \)
3: for \( i = 1 : t \) do
   4: \( \tilde{P}_{k+1}(i,:) = \frac{\tilde{P}_{k+1}(i,:)}{||P_{k+1}(i,:)||} = \frac{\tilde{P}_{k+1}(i,:)}{\sqrt{P_{k+1}(i,:)^t W_{k+1}(i,:)}} \)
   5: end for
```

B.2.2 A-orthonormalization of a set of vectors using CGS

Given a set of vectors \( P_{k+1} \) that are A-normalized, i.e. the diagonal of \( P_{k+1} P_{k+1}^t \) is equal to ones, we can A-orthonormalize it \( (P_{k+1}^t A P_{k+1} = I) \) using a classical Gram Schmidt procedure as in algorithm 19

Algorithm 19 A-orthonormalization against each others using CGS

**Input:** \( A \), the \( n \times n \) symmetric positive definite matrix
**Output:** \( \tilde{P}_{k+1} \), the A-orthonormalized search directions

```
1: Let \( \tilde{P}_{k+1} = P_{k+1} \)
2: for \( i = 1 : t \) do
   3: for \( j = 1 : (i-1) \) do
      4: \( \tilde{P}_{k+1}(i,:) = \tilde{P}_{k+1}(i,:) - (\tilde{P}_{k+1}(i,:)^t A P_{k+1}(i,:)) \tilde{P}_{k+1}(j,:) \)
   5: end for
   6: \( \tilde{P}_{k+1}(i,:) = \frac{\tilde{P}_{k+1}(i,:)}{||P_{k+1}(i,:)||} = \frac{\tilde{P}_{k+1}(i,:)}{\sqrt{P_{k+1}(i,:)^t A P_{k+1}(i,:)}} \) %A-normalize
3: end for
```

The CGS A-orthonormalization can be reformulated as a QR factorization

\[
P_{k+1} = \tilde{P}_{k+1} R
\]

where \( \tilde{P}_{k+1} \) is an A-orthonormal matrix, and \( R \) is a \( t \times t \) upper triangular matrix defined by the entries \( r_{j,i} \) for all \( j = 1, 2, .., t \) and \( i = 1, 2, .., t \).

\[
R = \begin{bmatrix}
  r_{1,1} & r_{1,2} & ... & r_{1,t} \\
  r_{2,2} & r_{2,3} & ... & r_{2,t} \\
  r_{3,3} & ... & r_{3,t} \\
  \vdots & & & \ddots \\
  r_{t,1} & \cdots & \cdots & r_{t,t}
\end{bmatrix} \quad \begin{bmatrix}
  ||p_1||_A < \tilde{p}_1, p_2 > A < \tilde{p}_1, p_3 > A \cdots < \tilde{p}_1, p_t > A \\
  \vdots & & & \ddots \\
  r_{2,2} & \cdots & \cdots & \tilde{p}_3, p_t > A \\
  \vdots & & & \ddots \\
  r_{t,1} & \cdots & \cdots & r_{t,t}
\end{bmatrix}
\]

Although the CGS A-orthonormalization is equivalent to a QR factorization with the A inner product, we were not able parallelize it using reduction trees with the same communication pattern as in TSQR [3]. But we can optimize the communication in algorithm [19] by noticing that once a vector \( p_i \) is orthonormalized, we can compute the corresponding entries of the matrix \( R \), i.e \( R(i, i+1 : t) \). By taking this into consideration, algorithm [19] can be restructured as in algorithm [20].
Algorithm 20 QR factorization with A inner product using CGS

Input: A, the \( n \times n \) symmetric positive definite matrix
Input: \( P_{k+1} \), the search directions to be A-orthonormalized
Output: \( \tilde{P}_{k+1} \), the A-orthonormalized search directions \( \tilde{P}_{k+1}^T A \tilde{P}_{k+1} = I \)
Output: \( R \), the upper triangular matrix such that \( P_{k+1} = \tilde{P}_{k+1} R \)

1: \( W_{k+1} = A \tilde{P}_{k+1} \)
2: \( R(1,1) = \sqrt{\tilde{P}_{k+1}^T A \tilde{P}_{k+1}^T (1,1)} \)
3: \( \tilde{P}_{k+1}(1,1) = \frac{\tilde{P}_{k+1}(1,1)}{R(1,1)} \)
4: for \( i = 2 : t \) do
5: \( R(i-1, i : t) = \tilde{P}_{k+1}^T A \tilde{P}_{k+1}^T (i-1, i : t) \)
6: \( \tilde{P}_{k+1}(i : i) = P_{k+1}(i : i) - \tilde{P}_{k+1}^T A \tilde{P}_{k+1}^T (i : i) R(1 : i - 1, i) \)
7: \( R(i, i) = \sqrt{\tilde{P}_{k+1}^T A \tilde{P}_{k+1}^T (i, i)} \)
8: \( \tilde{P}_{k+1}(i, i) = \frac{\tilde{P}_{k+1}(i, i)}{R(i, i)} \)
9: end for

The total flops of Algorithm 20 is of the order of \( mnzt + nt^2 \).

Total \( = 2nnzt - nt + 3n + \sum_{i=2}^{t} \left[ (2n-1)(t-i+1) + 2(i-1)n + (2nnz + 2n) \right] \)
\( = 2nnzt - nt + 3n + \sum_{i=2}^{t} \left[ (2n-1)(t+1) - (2n-1)i + 2ni - 2n - 2nnz + 2n \right] \)
\( = 2nnzt - nt + 3n + \sum_{i=2}^{t} \left[ (2n-1)(t+1) + i + 2nnz \right] \)
\( = 2nnzt - nt + 3n + \sum_{i=2}^{t} \left[ (2n-1)(t+1) + 2nnz \right] \)
\( = 4nnzt - 2nnz - nt + n + 2nt^2 - t^2 \frac{t^2 + 1}{2} \)
\( = 4nnzt - 2nnz - nt + n + 2nt^2 + \frac{-t^2 + t}{2} \)

The parallelization of Algorithm 20 starts by distributing the data similarly to Algorithm 17. Processor \( p_i \) computes \( A(\delta_p, \delta_p) P_{k+1}(\delta_p, i) \) and receives \( W_{k+1} \) via an "all reduce", and computes \( P_{k+1}(\delta_p, 1)^T W(\delta_p, 1) \), and receives the full dot product \( P_{k+1}(1)^T W(1) \), needed to compute \( R(1,1) \), via an "all reduce".

Then, it computes \( \tilde{P}_{k+1}(\delta_p, 1) = R_{(1,1)}^{-1} \).

At each iteration, processor \( p_i \) computes \( \tilde{P}_{k+1}(\delta_p, i-1)^T W_{k+1}(\delta_p, i : t) \) and receives the full \( R(i-1, i : t) \) by an all reduce. Then, it computes \( \tilde{P}_{k+1}(\delta_p, i) = P_{k+1}(\delta_p, i) - \tilde{P}_{k+1}(\delta_p, i-1)^T R(1 : i-1, i) \) and receives the \( P_{k+1}(\delta_p, i) \) from \( m_{MB} \) adjacent processors where \( \delta_p = \text{Adjacent}(G(A), \delta_p) \). Then it computes \( \tilde{W}_{k+1}(\delta_p, i) = A(\delta_p, \delta_p) \tilde{P}_{k+1}(\delta_p, i) \) and \( \tilde{P}_{k+1}(\delta_p, i)^T \tilde{W}_{k+1}(\delta_p, i) \) and receives the full \( \tilde{P}_{k+1}(\delta_p, i)^T A \tilde{P}_{k+1}(\delta_p, i) \), needed to compute \( R(i, i) \), via an all reduce. Finally, it computes \( \tilde{P}_{k+1}(\delta_p, i) = \frac{\tilde{P}_{k+1}(\delta_p, i)}{R_{(i,i)}} \). So, there is a need for a total of \( 2t - 1 \) "all reduce" and \( t \) communications with the \( m_{MB} \) neighboring processors.

It is possible to reduce the communication to only \( 2t - 1 \) "all reduce" by assuming that \( W_{k+1} = AP_{k+1} \) has already been computed and it is an input to Algorithm 21. Then, at each iteration \( i \), an "all reduce" of the vector \( R(i-1, i : t) \) of size \( t-i+1 \) is performed and another "all reduce" of the entry \( R(i, i) \) is performed. Thus, a total of \( (2t-1) \log(t) \) messages are sent with \( (1 + \sum_{i=2}^{t} 2 - i) \log(t) = \frac{t(t+1)}{2} \log(t) \) words where \( 3nt^2 + nt + \frac{n(t+1)}{2} = 3nt + n + \frac{(t-1)}{2} \) flops are performed in parallel. Hence, by ignoring lower order terms we obtain

\[
\text{Time}_{QRCGS_{\text{Aort}}} \approx 3 \alpha_3 nt + \alpha_2 2t \log(t) + \beta c t^2 \log(t)
\]
A-orthonormalizing the search directions and three forward substitutions.

performing a Cholesky factorization of the SPD matrix where

Algorithm 21 QR factorization with A inner product using CGS

| Input: | \( P_{k+1} \), the search directions to be A-orthonormalized; \( W_{k+1} = AP_{k+1} \) | Flops |
| Output: | \( \tilde{P}_{k+1} \), the A-orthonormalized search directions; \( \tilde{W}_{k+1} \) |
| Output: | \( R \), the upper triangular matrix such that \( P_{k+1} = \tilde{P}_{k+1} R \) |

1. \( R(1,1) = \sqrt{P_{k+1}(1,1)W_{k+1}(1,1)} \)
2. \( \tilde{P}_{k+1}(1,:) = \frac{P_{k+1}(1,:)}{R(1,1)} \) and \( \tilde{W}_{k+1}(1,:) = \frac{W_{k+1}(1,:)}{R(1,1)} \)
3. for \( i = 2 : t \) do
4. \( R(i-1:i,:) = \tilde{P}_{k+1}(i-1:i,:)W_{k+1}(i-1:i,:) \)
5. \( \tilde{P}_{k+1}(i,:) = P_{k+1}(i,:) - \tilde{P}_{k+1}(1:i-1,:)R(1:i-1,i) \)
6. \( \tilde{W}_{k+1}(i,:) = W_{k+1}(i,:) - \tilde{W}_{k+1}(1:i-1,:)R(1:i-1,i) \)
7. \( R(i,i) = \sqrt{\tilde{P}_{k+1}(i,:)'\tilde{P}_{k+1}(i,:)} \)
8. \( \tilde{P}_{k+1}(:,i) = \frac{P_{k+1}(:,i)}{R(:,i)} \) and \( \tilde{W}_{k+1}(:,i) = \frac{W_{k+1}(:,i)}{R(:,i)} \)
9. end for

B.2.3 CGS with Reorthogonalization (CGS2)

The CGS with reorthogonalization (CGS) consists of calling the CGS algorithms twice, be it for A-orthonormalizing \( P_{k+1} \) against previous vectors of \( Q_k \) (Algorithm 22), or A-orthonormalizing \( P_{k+1} \).

Algorithm 22 A-orthonormalization of \( P_{k+1} \) against previous vectors of \( Q_k \) using CGS2

| Input: | \( Q_k \), the \( tk \) search directions |
| Input: | \( P_{k+1} \), the \( t \) search directions to be A-orthonormalized |
| Input: | \( W_{k+1} = AP_{k+1}; W_{k+1} = AQ_k \) |
| Output: | \( \tilde{P}_{k+1} \), the search directions A-orthonormalized against \( P_1, P_2, \ldots, P_k; \tilde{W}_{k+1} = A\tilde{P}_{k+1} \) |
1. Call Algorithm 18 with \( P_{k+1} \) and \( W_{k+1} \) as input and with \( P_{k+1} \) and \( W_{k+1} \) as output
2. Call Algorithm 18 with \( P_{k+1} \) and \( W_{k+1} \) as input and with \( P_{k+1} \) and \( W_{k+1} \) as output

In the case of \( L^{-1}AL^{-t} \)-orthogonalization of \( P_{k+1} \) against previous vectors of \( Q_k \) where \( L^{-t} = (L^t)^{-1} \), the CGS2 algorithm is defined in Algorithm 23. Note that we have to solve 6 systems with multiple right hand sides. If \( L \) is a lower triangular matrix, then we perform three backward substitutions and three forward substitutions.

B.3 Cholesky QR A-orthonormalization

A-orthonormalizing the \( n \times t \) full rank matrix \( P_{k+1} \) is equivalent to a QR factorization \( P_{k+1} = \tilde{P}_{k+1} R \) where \( \tilde{P}_{k+1} = \tilde{P}_{k+1} I \). Thus, \( P_{k+1} A P_{k+1} = (\tilde{P}_{k+1} R)' \tilde{P}_{k+1} R = R^t R \) and \( R \) can be obtained by performing a Cholesky factorization of the SPD matrix \( P_{k+1} A P_{k+1} \). Then, \( \tilde{P}_{k+1} = P_{k+1} R^{-1} \) is obtained by solving the lower triangular system \( R^t \tilde{P}_{k+1} = P_{k+1} \) with multiple right-hand sides. This procedure is called A-CholQR and summarized in Algorithm 24. Similarly to the other A-orthonormalization methods, we may assume that \( W_{k+1} \) is already computed, then the obtained A-CholQR is described in Algorithm 25.

The parallelization of Algorithm 25 assumes that we have \( t \) processors and each is assigned a rowwise part of \( P_{k+1} \) and \( W_{k+1} \) corresponding to the \( \delta_i \) subset of indices defined previously, \( P_{k+1} (\delta_i,:) \) and \( W_{k+1} (\delta_i,:) \). And each processor \( i \) should compute \( \tilde{P}_{k+1} (\delta_i,:) \) and \( \tilde{W}_{k+1} (\delta_i,:) \). Then each processor \( i \) computes \( W_{k+1} (\delta_i,:) P_{k+1} (\delta_i,:) \) and \( W_{k+1} (\delta_i,:) P_{k+1} (\delta_i,:) \) via an “all reduce” or equivalently

Inria
Algorithm 23 $L^{-1}AL^{-1}$-orthonormalization against previous vectors of $Q_k$ with CGS2

**Input:** $A$, the $n \times n$ symmetric positive definite matrix; $L$, $n \times n$ preconditioner

**Input:** $P_{k+1}$, the $tk$ search directions

**Output:** $\tilde{P}_{k+1}$, the search directions to be $L^{-1}A(L')^{-1}$-orthonormalized against $Q_k$

**Output:** $\tilde{W}_{k+1} = L^{-1}AL^{-1}\tilde{P}_{k+1}$

1: $\tilde{W}_{k+1} = L^{-1}AL^{-1}P_{k+1}$
2: $\tilde{P}_{k+1} = P_{k+1} - Q_k(Q_k^T\tilde{W}_{k+1})$
3: $\tilde{W}_{k+1} = L^{-1}AL^{-1}\tilde{P}_{k+1}$
4: for $i = 1 : t$ do %loop over the vectors of $P_{k+1}$ and $L^{-1}AL^{-1}$-normalized
5: \[ \tilde{P}_{k+1}(::i) = \frac{P_{k+1}(::i)}{|P_{k+1}(::i)|_A L^{-1}} = \frac{P_{k+1}(::i)}{\sqrt{P_{k+1}(::i)^T W_{k+1}(::i)}} \] \[ \tilde{W}_{k+1}(::i) = \frac{\tilde{W}_{k+1}(::i)}{\sqrt{\tilde{W}_{k+1}(::i)^T W_{k+1}(::i)}} \]
6: end for
7: $\tilde{P}_{k+1} = \tilde{P}_{k+1} - Q_k(Q_k^T\tilde{W}_{k+1})$
8: $\tilde{W}_{k+1} = L^{-1}AL^{-1}\tilde{P}_{k+1}$
9: for $i = 1 : t$ do %loop over the vectors of $P_{k+1}$ and $L^{-1}AL^{-1}$-normalized
10: \[ \tilde{P}_{k+1}(::i) = \frac{P_{k+1}(::i)}{|P_{k+1}(::i)|_A L^{-1}} = \frac{P_{k+1}(::i)}{\sqrt{P_{k+1}(::i)^T W_{k+1}(::i)}} \] \[ \tilde{W}_{k+1}(::i) = \frac{\tilde{W}_{k+1}(::i)}{\sqrt{\tilde{W}_{k+1}(::i)^T W_{k+1}(::i)}} \]
11: end for

Algorithm 24 A-CholQR

**Input:** $A$, the $n \times n$ symmetric positive definite matrix

**Input:** $P_{k+1}$, the search directions to be A-orthonormalized

**Output:** $P_{k+1}$, the A-orthonormalized search directions

1: Compute $W_{k+1} = AP_{k+1}$
2: Compute $C = W_{k+1}^T P_{k+1}$
3: Compute the Cholesky factorization of $C = R^T R$ to obtain $R$
4: Solve $R^T P_{k+1} = P_{k+1}$
5: for $i = 1 : t$ do
6: \[ \tilde{P}_{k+1}(::i) = \frac{P_{k+1}(::i)}{|P_{k+1}(::i)|_A} = \frac{P_{k+1}(::i)}{\sqrt{P_{k+1}(::i)^T A P_{k+1}(::i)}} \]
7: end for

Algorithm 25 A-CholQR

**Input:** $P_{k+1}$, the search directions to be A-orthonormalized, $W_{k+1} = AP_{k+1}$

**Output:** $\tilde{P}_{k+1}$, the A-orthonormalized search directions; $\tilde{W}_{k+1} = A\tilde{P}_{k+1}$

1: Compute $C = W_{k+1}^T P_{k+1}$
2: Compute the Cholesky factorization of $C = R^T R$ to obtain $R$
3: Solve $R^T P_{k+1} = P_{k+1}$
4: Solve $R^T \tilde{W}_{k+1} = \tilde{W}_{k+1}$
5: for $i = 1 : t$ do
6: \[ \tilde{P}_{k+1}(::i) = \frac{P_{k+1}(::i)}{|P_{k+1}(::i)|_A} = \frac{P_{k+1}(::i)}{\sqrt{P_{k+1}(::i)^T W_{k+1}(::i)}} \] \[ \tilde{W}_{k+1}(::i) = \frac{\tilde{W}_{k+1}(::i)}{\sqrt{\tilde{W}_{k+1}(::i)^T W_{k+1}(::i)}} \]
7: end for
log(t) messages and \( t^2 \log(t) \) words. Finally, each processor \( i \) can compute the Cholesky factorization of the matrix \( C \) to obtain \( R \) which is needed to solve \( R^t \hat{P}_{k+1}(\delta_{i,:}) = P_{k+1}(\delta_{i,:})^t \) and \( R^t \hat{W}_{k+1}(\delta_{i,:}) = W_{k+1}(\delta_{i,:})^t \). Thus, it is possible to parallelize the \( \hat{A} \)-CholQR A-orthonormalization, Algorithm 25, by sending \( \log(t) \) messages with \( t^2 \log(t) \) words and performing \( t^2 + \frac{(4n-1)t^2}{t} \approx 4nt \) flops in parallel. Hence, by ignoring lower order terms we obtain

\[
\text{Time}_{\hat{A}-\text{CholQR}} \approx \gamma_c 4nt + \alpha_c \log(t) + \beta_c t^2 \log(t)
\]

In the case of \( \hat{A} \)-orthonormalization of \( \hat{P}_{k+1} \), where \( \hat{A} = L^{-1}AL^{-t} \) and \( L^{-1} = (L^t)^{-1} \), the \( \hat{A} \) CholQR algorithm is defined in Algorithm 26. Note that we have to solve 2 systems with multiple right hand sides. If \( L \) is a lower triangular matrix, then we perform a backward and forward substitution.

**Algorithm 26 \( \hat{A} \)-CholQR**

**Input:** \( A \), the \( n \times n \) symmetric positive definite matrix; \( L, n \times n \) preconditioner

**Input:** \( P_{k+1} \), the search directions to be \( \hat{A} \)-orthonormalized; \( W_{k+1} = L^{-1}AL^{-t}P_{k+1} \)

**Output:** \( \hat{P}_{k+1} \), the \( \hat{A} \)-orthonormalized search directions; \( \hat{W}_{k+1} = L^{-1}AL^{-t}\hat{P}_{k+1} \)

1: Compute \( C = W_{k+1}^tP_{k+1} \)
2: Compute the Cholesky QR factorization of \( C \) to obtain \( R \)
3: Solve \( R^t \hat{P}_{k+1} = \hat{P}_{k+1} \)
4: Compute \( \hat{W}_{k+1} = L^{-1}AL^{-t}\hat{P}_{k+1} \)
5: for \( i = 1: t \) do
6: \( \hat{P}_{k+1}(:,i) = \frac{p_{k+1}(:,i)}{||p_{k+1}(:,i)||_A} = \frac{p_{k+1}(:,i)}{\sqrt{p_{k+1}(:,i)^tW_{k+1}(:,i)}} \) and \( \hat{W}_{k+1}(:,i) = \frac{\hat{W}_{k+1}(:,i)}{\sqrt{\hat{W}_{k+1}(:,i)^t\hat{W}_{k+1}(:,i)}} \)
7: end for

Recently, Lowery and Langou presented a new version of A-CholQR in [20], which they call Pre-CholQR (Algorithm 27). It consists in performing a Euclidean QR factorization with L2 before calling the A-CholQR A-orthonormalization, Algorithm 24. The QR factorization of \( P_{k+1} \) can be done using the TSQR [4], which requires sending \( \log(t) \) messages, each of size \( \frac{t^2}{2} \) words and computing \( 2nt + \frac{2t^3}{3} \log(t) \). Then, parallelizing Algorithm 24 requires performing two “all reduce” or “2log(t)” messages with \( \log(t) \) words. In total, parallelizing Algorithm 27 requires sending \( 3 \log(t) \) messages with \( \log(t) \) words. Hence, by ignoring lower order terms we obtain

\[
\text{Time}_{\text{PreCholQR}} \approx \gamma_c (6nt + \frac{2}{3} t^3 \log(t)) + \alpha_c 2 \log(t) + \beta_c (nt + \frac{3}{2} t^2 \log(t))
\]

**Algorithm 27 Pre-CholQR**

**Input:** \( A \), the \( n \times n \) symmetric positive definite matrix

**Input:** \( P_{k+1} \), the search directions to be \( \hat{A} \)-orthonormalized

**Output:** \( \hat{P}_{k+1} \), the \( \hat{A} \)-orthonormalized search directions

1: Compute the QR factorization of \( P_{k+1} = P_{k+1}^tR \)
2: Call Algorithm 24 with \( A \) and \( P_{k+1} \) as input and with \( \hat{P}_{k+1} \) as output