Enlarged Krylov Subspace Conjugate Gradient Methods for Reducing Communication

Laura Grigori, Sophie Moufawad, Frédéric Nataf

Enlarged Krylov Subspace Conjugate Gradient Methods for Reducing Communication rapport de recherche Inria

Laura Grigori, Sophie Moufawad , Frederic Nataf

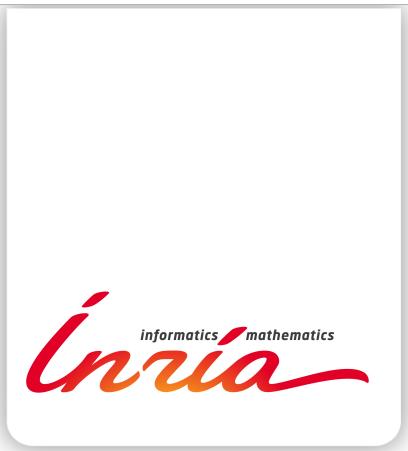
RESEARCH REPORT

N° 8597

September 2014

Project-Teams
ALPINES

ISSN 0249-6399 ISRN INRIA/RR--8597--FR+ENG



Enlarged Krylov Subspace Conjugate Gradient Methods for Reducing Communication rapport de recherche Inria

Laura Grigori*, Sophie Moufawad *, Frederic Nataf †

Équipes-Projets
ALPINES

Rapport de recherche n° 8597 — September 2014 — 46 pages

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

Mots-clés : algèbre linéaire, méthodes itératives

* INRIA
† UPMC

RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

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 $\mathcal{K}_k(A, r_0)$, $\mathcal{K}_k(A, r_0) \subset \mathcal{K}_{t,k+1}(A, r_0)$. 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 guarentees 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 $x_1, x_2, x_3, x_4, ..., x_k$ that minimizes some measure of error over the corresponding spaces

$$x_0 + \mathcal{K}_i(A, r_0), \quad i = 1, ..., k$$

where x_0 is the initial iterate, r_0 is the initial residual, and $\mathcal{K}_i(A, r_0) = span\{r_0, Ar_0, A^2r_0, ..., A^{i-1}r_0\}$ 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 $\mathcal{K}_{t,k}(A, r_0)$ is a superset of the Krylov subspace $\mathcal{K}_k(A, r_0)$, $\mathcal{K}_k(A, r_0) \subset \mathcal{K}_{t,k+1}(A, r_0)$. 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, which was introduced by Hestenes and Stiefel in 1952 (section 2.1). 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 MSDO-CG we A-orthonormalize the search directions, to obtain a projection method that guarentees 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 subdomaines 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 dicretization of 2D poisson equations (POISSON2D), 3D elasctisity equations (ELASTICITY3D), and 2D and 3D convection-diffusion equations such as NH2D, SKY2D, SKY3D, and 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 NH2D, POISSON2D, and ELASTICITY3D, and 95% to 98% less iteration with respect to CG for the matrices SKY2D, SKY3D, and 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: $x_k \in x_0 + \mathcal{K}_k(A, r_0)$
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, $\mathcal{K}_k(A, r_0) = span\{r_0, Ar_0, A^2r_0, ..., A^{k-1}r_0\}$ 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 $\mathcal{L}_k = \mathcal{K}_k(A, r_0)$.

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 differenet versions of CG have been introduced (refer to [9] 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 [1] which was recently introduced, solves the system $Ax = b$ by starting with t distinct initial guesses. This is equivalent to solving the system $AX = b * \mathbb{1}_t$ (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 α_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. $P_k^t AP_i$ 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 3.2).

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 δ, $p(\delta)$ is the vector formed by the subset of the entries of p whose indices belong to δ. For a matrix A, $A(\delta, :)$ is a submatrix formed by the subset of the rows of A whose indices belong to α. Similarly, $A(:, \alpha)$, is a submatrix formed by the subset of the columns of A whoses indices belong to α. And $A(\alpha, \beta) = [A(\alpha, :)](:, \beta)$, is formed by the β columns of the submatrix $A(\alpha, :)$

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

$$\begin{cases} Ax &= b, \\ A &= A^t, \\ x^tAx &> 0, \forall x \neq 0. \end{cases} \tag{1}$$

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 $\phi(x) = \frac{1}{2}(x)^t Ax - b^t x$ over the corresponding space $x_0 + \mathcal{K}_k(A, r_0)$, where $k > 0$, $p_k \in \mathcal{K}_k(A, r_0)$ is the k^{th} search direction, and α_k is the step along the search direction.

The minimum of $\phi(x)$ is given by $\bigtriangledown\phi(x) = 0$, which is equivalent to $\bigtriangledown\phi(x) = Ax - b = 0$. Thus, by minimizing $\phi(x)$ we are solving the system (1). As the name of the method indicates, the gradients $\bigtriangledown\phi(x_i)$ for all i should be conjugate. And since CG is a Krylov projection method, the residual $r_k = b - Ax_k$ should respect the Petrov-Galerkin condition

$$r_k \perp \mathcal{L}_k,$$

where r_k is orthogonal to some well-defined subspace $\mathcal{L}_k \subseteq \mathbb{R}^n$ (or $\subseteq \mathbb{C}^n$) of dimension k. In CG, the subspace $\mathcal{L}_k$ is the same as the Krylov subspace $\mathcal{K}_k$. Thus, $(r_k)^t y = 0$, for all $y \in \mathcal{K}_k$. Hence, the residuals form an orthogonal set, $(r_k)^t r_i = 0$, for all $i < k$.

Moreover, the Petrov-Galerkin condition $r_k \perp \mathcal{K}_k(A, r_0)$ is equivalent to the conjugacy of the gradients $\bigtriangledown\phi(x_k)^t \bigtriangledown \phi(x_i) = 0$, for all$i \neq k$. Once x_k has been chosen, either x_k is the required approximate solution of $Ax = b$ or a new search direction $p_{k+1} \neq 0$ must be determined to compute the new approximation $x_{k+1} = x_k + \alpha_{k+1} p_{k+1}$. This procedure is repeated until convergence or untill the maximum number of allowed iterations has been reached without convergence. The convergence criteron is set as

$$||r_k||_2 \leqslant \epsilon ||b||_2, \quad for\ some\ \epsilon \in \mathbb{R},$$

where $r_k = b - Ax_k \in \mathcal{K}_{k+1}(A, r_0)$ is the k^{th} residual.

Theorem 2.1. *The Petrov-Galerkin condition $(r_k)^t y = 0$, for all $y \in \mathcal{K}_k$ implies the A-orthogonality of the search directions $p_i^t A p_j = 0$, for all $i \neq j$ and $i, j \leqslant k$.*

Proof. By definition, $p_i \in \mathcal{K}_i$ and $\mathcal{K}_i \subset \mathcal{K}_{i+1}$. Thus $p_i \in \mathcal{K}_{i+c}$ for $c \geqslant 0$. By the Petrov-Galerkin condition $r_{k-1}^t p_i = 0$ and $r_k^t p_i = 0$ for $i \leqslant k-1$. Thus, $r_k^t p_i = r_{k-1}^t p_i - \alpha p_k^t A p_i = 0$ for $i \leqslant k-1$. This implies that $p_k^t A p_i = 0$ for $i \leqslant k-1$ 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 $p_k \in \mathcal{K}_k$ is chosen according to the following recursion relation:

$$\begin{cases} p_1 &= r_0 \\ p_k &= r_{k-1} + \beta_k p_{k-1} \end{cases} \tag{2}$$

where p_1 is set equal to r_0 since the initial residual is equal to negative the gradient $-\bigtriangledown\phi(x_0)$ 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. $p_k^t A p_i = 0$ for all $i \leqslant k-1$. For $i < k-1$, we have

$$p_k^t A p_i = r_{k-1}^t A p_i + \beta_k p_{k-1}^t A p_i = \beta_k p_{k-1}^t A p_i \tag{3}$$

since $r_{k-1}^t A p_i = 0$ by Petrov-Galerkin condition. In addition, $r_{k-1}^t p_i = r_{k-2}^t p_i - \alpha_{k-1} p_{k-1}^t A p_i = 0$ with $r_{k-2}^t p_i = 0$ since $i \leqslant k-2$. Thus, $p_{k-1}^t A p_i = 0$. Therefore, $p_k^t A p_i = 0$ 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 $p_k^t A p_{k-1} = 0$

At each iteration, the step $\alpha_k = \frac{(p_k)^t r_{k-1}}{(p_k)^t A p_k} = \frac{||r_{k-1}||_2^2}{||p_k||_A^2}$ is chosen such that,

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

Using the definition of α_k, $\beta_k = -\frac{(r_{k-1})^t A p_{k-1}}{(p_{k-1})^t A p_{k-1}} = \frac{||r_{k-1}||_2^2}{||r_{k-2}||_2^2}$.

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

$$\begin{cases} AX &= B, \\ A &= A^t, \\ x^tAx &> 0, \forall x \neq 0 \end{cases} \tag{4}$$

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 $R_0 = B - AX_0$, $P_1 = R_0\gamma_1$ with γ_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 $\mathcal{K}_{k+1}(A, R_0) = block - span\{R_0, AR_0, A^2R_0, ..., A^kR_0\}$ is the block Krylov subspace. Every $n \times t$ block $Z \in \mathcal{K}_{k+1}(A, R_0)$ is defined as $Z = \sum_{i=1}^{k} A^iR_0\zeta_i$ where ζ_i is a $t \times t$ matrix. By the Petrov-Galerkin condition we have that $R_{k+1} \perp \mathcal{K}_{k+1}(A, R_0)$. Then, $R_{k+1}^tY = 0$ for all $Y \in \mathcal{K}_{k+1}(A, R_0)$, which implies that $R_{k+1}^tR_i = 0$ and $R_{k+1}^tAP_i = 0$ for all $i < k + 1$.

Then, for $k \geqslant 0$ the iterates are defined similarly to CG:

$$\begin{aligned} X_{k+1} &= X_k + P_{k+1}\alpha_{k+1} && \in \mathcal{K}_{k+1}(A, R_0) \\ R_{k+1} &= R_k - AP_{k+1}\alpha_{k+1} && \in \mathcal{K}_{k+2}(A, R_0) \\ P_{k+2} &= (R_{k+1} + P_{k+1}\beta_{k+2})\gamma_{k+2} && \in \mathcal{K}_{k+2}(A, R_0) \end{aligned}$$

where

$$\begin{aligned} \alpha_{k+1} &= (P_{k+1}^tAP_{k+1})^{-1}\gamma_k^t(R_k^tR_k) \\ \beta_{k+2} &= \gamma_{k+1}^{-1}(R_k^tR_k)^{-1}(R_{k+1}R_{k+1}) \end{aligned}$$

Note that α_{k+1} is chosen such that $\phi(X_{k+1}) = min\{\phi(X_k + P_{k+1}\alpha), \text{for all}\alpha \in \mathbb{R}^{t,t}\}$. As for β_{k+1}, it is chosen to ensure the A-orthogonality of the P_{k+1} and P_k ($(P_{k+1})^tAP_k = 0$). Whereas γ_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, $(P_{k+1})^tAY = 0$, for all $Y \in \mathcal{K}_k(A, R_0)$. This leads to the A-orthogonality of the search direction $\Longrightarrow (P_{k+1})^tAP_i = 0$, for all $i < k + 1$. We present the Block-CG algorithm in Appendix A (Algorithm **??**).

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 $AX = b * ones(1, t)$ where X_0 is a block-vector containing the t initial guesses, $R_0 = AX_0 - b * \mathbb{1}_t$ 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 (δ_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 α_k is a vector of size t.

Given an initial guess x_0, the residual is defined as $r_k = b - Ax_k$ for $k \geqslant 0$. The first set of domain search directions is defined by the initial residual r_0, such that $p_i^1(\delta_i) = r_0(\delta_i)$ for $i = 1, 2, ..., t$ and zero otherwise. Then, for $k > 1$ the domain search directions are defined as follows, $p_i^k = T_i(r_{k-1}) + \beta_i^k p_i^{k-1}$ for $i = 1, 2, ..., t$ where β_i^k is a scalar and T_i is an operator that projects a vector onto the subdomain δ_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,

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

As for $\alpha_k = (P_k^t A P_k)^{-1} P_k^t r_{k-1}$, 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. $P_k^t A P_i$ is not equal to zero for all i not equal to k. Thus $\beta_k = (P_{k-1}^t A P_{k-1})^{-1} P_{k-1}^t A r_{k-1}$ is chosen so that the global search direction $p^k = \sum_{i=1}^t p_i^k$ is A-orthogonal to the previous domain search direction p_i^{k-1}, i.e. $(p^k)^t A P_{k-1} = 0$, for $i = 1, 2, .., t$. 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 $\mathcal{K}_k$ 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 guarentees 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, MSDO-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 3.3.1.

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 $\delta = \{1, 2, .., n\}$ is divided into t distinct

subdomains δ_i, where $\delta = \cup_{i=1}^{t} \delta_i$.

We define $T_i(x)$ to be the operator that projects the vector x onto the subdomain δ_i. Let $y = T_i(x)$, then $y(\delta_i) = x(\delta_i)$ and zero elsewhere. Then, we define $T(x)$ to be an operator that transforms the $n \times 1$ vector x into t vectors of size $n \times 1$ that correspond to the projection of x onto the subdomains δ_i for $i = 1, 2, .., t$. If the obtained t vectors are assembled in increasing order into a block vector X, then we have $X(\delta_i, i) = x(\delta_i)$ for all i and zero elsewhere. We will refer to R_0 as the block containing the t vectors obtained from $T(r_0)$. Note that $R_0 \neq T(r_0)$ since R_0 is a matrix, whereas $T(r_0) = \{T_1(r_0), T_2(r_0), \ldots., T_t(r_0)\}$ 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*

$$\begin{aligned}\mathcal{K}_{t,k} &= span\{T(r_0), AT(r_0), A^2T(r0), ..., A^{k-1}T(r_0)\} \\ &= span\{T_1(r_0), T_2(r_0), ..., T_t(r_0), AT_1(r_0), AT_2(r_0), ..., AT_t(r_0), ..., A^{k-1}T_1(r_0), ..., A^{k-1}T_t(r_0)\}\end{aligned}$$

be an enlarged Krylov subspace of dimension $k \leqslant z \leqslant tk$ generated by the matrix A and the vector r_0, and associated to a given partition defined by δ_i for $i = 1, 2, .., t$.

The enlarged Krylov subspaces $\mathcal{K}_{t,k}(A, r_0)$ are increasing subspaces, yet bounded. We denote by k_{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 $\mathcal{K}_{t,k}$ ($\mathcal{K}_k \subset \mathcal{K}_{t,k}$).*

Proof. Let $y \in \mathcal{K}_k$ where $\mathcal{K}_k = span\{r_0, Ar_0, .., A^{k-1}r_0\}$. Then

$$y = \sum_{j=0}^{k-1} a_j A^j r_0 = \sum_{j=0}^{k-1} a_j A^j R_0 * \mathbb{1}_t = \sum_{j=0}^{k-1} \sum_{i=1}^{t} a_j A^j T_i(r_0) \in \mathcal{K}_{t,k}$$

since $r_0 = R_0 * \mathbb{1}_t = [T_1(r_0)\, T_2(r_0) \ldots. T_t(r_0)] * \mathbb{1}_t$. □

Krylov subspace methods search for an approximate solution $x_k \in x_0 + \mathcal{K}_k$. A corollary of theorem 3.2 is that we can search for an approximate solution x_k 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 guarenteed that the intersection of the two subspaces, $\mathcal{K}_{t,k}$ and $span\{A^kT_1(r_0), A^kT_2(r_0), ..., A^kT_t(r_0)\}$, is empty.

Theorem 3.3. *By definition 3.1 of the enlarged Krylov subspace,*

$$\mathcal{K}_{t,k+1} = \mathcal{K}_{t,k} + span\{A^kT_1(r_0), A^kT_2(r_0), ..., A^kT_t(r_0)\}$$

If $A^kT_v(r_0) \in \mathcal{K}_{t,k}$ for all $1 \leqslant v \leqslant t$, then $A^{k+q}T_i(r_0) \in \mathcal{K}_{t,k}$ for some $1 \leqslant i \leqslant t$ 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 $1 \leqslant v \leqslant t$, we show that $A^{k+1}T_i(r_0) \in \mathcal{K}_{t,k}$, where $1 \leqslant i \leqslant t$.

$A^kT_i(r_0) = \sum_{u=0}^{k-1} \sum_{v=1}^{t} \alpha_{u,v} A^u T_v(r_0)$ since $A^kT_i(r_0) \in \mathcal{K}_{t,k}$. Then

$$\begin{aligned}A^{k+1}T_i(r_0) &= \sum_{u=0}^{k-1} \sum_{v=1}^{t} \alpha_{u,v} A^{u+1} T_v(r_0) = \sum_{u=0}^{k-2} \sum_{v=1}^{t} \alpha_{u,v} A^{u+1} T_v(r_0) + \sum_{v=1}^{t} \alpha_{k-1,v} A^k T_v(r_0) \\ &= \sum_{u=0}^{k-2} \sum_{v=1}^{t} \alpha_{u,v} A^{u+1} T_v(r_0) + \sum_{v=1}^{t} \alpha_{k-1,v} (\sum_{u=0}^{k-1} \sum_{y=1}^{t} \beta_{u,y} A^u T_y(r_0)) \\ &= \sum_{u=0}^{k-1} \sum_{v=1}^{t} \gamma_{u,v} A^u T_v(r_0) \in \mathcal{K}_{t,k}\end{aligned}$$

Assume true for q
Assume that $A^{k+q}T_i(r_0) \in \mathcal{K}_{t,k}$ where $1 \leqslant i \leqslant t$, 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 $A^{k+q+1}T_i(r_0) \in \mathcal{K}_{t,k}$

$$\begin{aligned} A^{k+q+1}T_i(r_0) &= \sum_{u=0}^{k-1}\sum_{v=1}^{t}\alpha_{u,v}A^{u+1}T_v(r_0) = \sum_{u=0}^{k-2}\sum_{v=1}^{t}\alpha_{u,v}A^{u+1}T_v(r_0) + \sum_{v=1}^{t}\alpha_{k-1,v}A^kT_v(r_0) \\ &= \sum_{u=0}^{k-2}\sum_{v=1}^{t}\alpha_{u,v}A^{u+1}T_v(r_0) + \sum_{v=1}^{t}\alpha_{k-1,v}(\sum_{u=0}^{k-1}\sum_{y=1}^{t}\beta_{u,y}A^uT_y(r_0)) \\ &= \sum_{u=0}^{k-1}\sum_{v=1}^{t}\gamma_{u,v}A^uT_v(r_0) \in \mathcal{K}_{t,k} \end{aligned}$$

□

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

Theorem 3.4. *If $A^kT_i(r_0) \in \mathcal{K}_{t,k} + span\{A^kT_1(r_0), ..., A^kT_{i-1}(r_0), A^kT_{i+1}(r_0), ..., A^kT_t(r_0)\}$, then $A^{k+q}T_i(r_0) \in \mathcal{K}_{t,k+q} + span\{A^{k+q}T_1(r_0), ..., A^{k+q}T_{i-1}(r_0), A^{k+q}T_{i+1}(r_0), ..., A^{k+q}T_t(r_0)\}$ for all $1 \leqslant i \leqslant t$ and $q > 0$.*

Proof. If $A^kT_i(r_0) \in \mathcal{K}_{t,k} + span\{A^kT_1(r_0), ..., A^kT_{i-1}(r_0), A^kT_{i+1}(r_0), ..., A^kT_t(r_0)\}$, then $A^kT_i(r_0) = \sum_{u=0}^{k-1}\sum_{v=1}^{t}\alpha_{u,v}A^uT_v(r_0) + \sum_{\substack{v=1\\v\neq i}}^{t}\alpha_{k,v}A^kT_v(r_0)$. Thus,

$$\begin{aligned} A^{k+q}T_i(r_0) &= \sum_{u=0}^{k-1}\sum_{v=1}^{t}\alpha_{u,v}A^{u+q}T_v(r_0) + \sum_{\substack{v=1\\v\neq i}}^{t}\alpha_{j,v}A^{k+q}T_v(r_0) \\ &\in \mathcal{K}_{t,k+q} + span\{A^{k+q}T_1(r_0), ..., A^{k+q}T_{i-1}(r_0), A^{k+q}T_{i+1}(r_0), ..., A^{k+q}T_t(r_0)\} \end{aligned}$$

□

A corollary of Theorem 3.4 is that if $t - i_k$ vectors of the form $A^kT_y(r_0)$ with $y = i_k + 1, ..., t$ belong to the subspace $\mathcal{K}_{t,k} + span\{A^kT_1(r_0), A^kT_2(r_0), ..., A^kT_{i_k}(r_0)\}$, then the $t - i_k$ vectors of the form $A^{k+q}T_y(r_0)$ belong to the subspace $\mathcal{K}_{t,k+q} + span\{A^{k+q}T_1(r_0), A^{k+q}T_2(r_0), ..., A^{k+q}T_{i_j}(r_0)\}$.

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

Proof. By definition of k_{max}, we have that for all $q > 0$

$$\mathcal{K}_{t,1} \subsetneq ... \subsetneq \mathcal{K}_{t,k_{max}-1} \subsetneq \mathcal{K}_{t,k_{max}} = \mathcal{K}_{t,k_{max}+q}.$$

Then for all $k < k_{max}$, the dimension of the enlarged Krylov subspaces $\mathcal{K}_{t,k}$ is stricltly increasing by some number $i_k \neq 0$ with respect to the dimension of $\mathcal{K}_{t,k-1}$.

If the t new vectors are linearly independent and none of them belongs to $\mathcal{K}_{t,k-1}$, then the t vectors are added to the basis of $\mathcal{K}_{t,k}$ and $dim(\mathcal{K}_{t,k}) = dim(\mathcal{K}_{t,k-1}) + t$, where $i_k = t$ and $dim()$ is the dimension of a subspace . In case the t new vectors are linearly dependent and none of them belongs to $\mathcal{K}_{t,k-1}$, then only one vector is added to the basis of $\mathcal{K}_{t,k}$ and $dim(\mathcal{K}_{t,k}) = dim(\mathcal{K}_{t,k-1}) + 1$, 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 dependant 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 $dim(\mathcal{K}_{t,k}) = (\mathcal{K}_{t,k-1}) + i_k$, where $1 < i_k < t$.

In general, $dim(\mathcal{K}_{t,k}) = dim(\mathcal{K}_{t,k-1}) + i_k$, where $1 \leqslant i_k \leqslant t$. Similarly, $dim(\mathcal{K}_{t,k+1}) = dim(\mathcal{K}_{t,k}) + i_{k+1}$, where $1 \leqslant i_{k+1} \leqslant t$. 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 dependant 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 $\mathcal{K}_{t,k-1} + span\{A^{k-1}T_1(r_0),, ..., A^{k-1}T_{i_k}(r_0)\}$. Then by Theorem 3.4 and its corollary, the $t - i_k$ vectors of the form $A^{k+q}T_i(r_0)$ belong to the subspace $\mathcal{K}_{t,k+q} + span\{A^{k+q}T_1(r_0), A^{k+q}T_2(r_0), ..., A^{j+q}T_{i_k}(r_0)\}$ 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 $k_{max} \leqslant p_{max}$.*

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 $A^{p_{max}+q-1}r_0 \in \mathcal{K}_{p_{max}} \subset \mathcal{K}_{t,p_{max}}$, and $A^{p_{max}+q-1}r_0 = \sum_{j=1}^{p_{max}} \sum_{i=1}^{t} \alpha_{j,i}A^{j-1}T_i(r_0)$. Thus $A^{p_{max}+q-1} \sum_{i=1}^{t} T_i(r_0) = \sum_{j=1}^{p_{max}} \sum_{i=1}^{t} \alpha_{j,i}A^{j-1}T_i(r_0)$.

Suppose that $A^{p_{max}+q-1}T_i(r_0) \notin \mathcal{K}_{t,p_{max}}$ for all $1 \leqslant i \leqslant t$. Then $A^{p_{max}+q-1} \sum_{i=1}^{t} T_i(r_0) = \sum_{j=1}^{p_{max}+q-1} \sum_{i=1}^{t} \alpha_{j,i}A^{j-1}T_i(r_0)$. 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 $A^{p_{max}+q-1}T_i(r_0) \in \mathcal{K}_{t,p_{max}}$ for all $1 \leqslant i \leqslant t$.

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

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 $\mathcal{K}_{p_{max}} = span\{r_0, Ar_0, .., A^{p_{max}-1}r_0\}$ and $\mathcal{K}_{p_{max}} = \mathcal{K}_{p_{max}+q}$ for $q > 0$. Since $\mathcal{K}_{p_{max}} \subset \mathcal{K}_{t,p_{max}}$, the solution $x_{sol} \in x_0 + \mathcal{K}_{t,p_{max}}$, where $p_{max} \geqslant k_{max}$ 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 $\mathcal{K}_{t,k_{max}} \neq \mathcal{K}_{t,p_{max}}$. However, by definition of k_{max} and since $k_{max} \leqslant p_{max}$, we have that $\mathcal{K}_{t,k_{max}} = \mathcal{K}_{t,p_{max}}$. 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: $r_k \perp \mathcal{K}_{t,k}$
$$\iff (r_k)^t y = 0, \quad \text{for all} \quad y \in \mathcal{K}_{t,k}$$

where $\mathcal{K}_{t,k}$ is a well-defined subspace of dimension $k \leqslant z \leqslant tk$.

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 $x_0 + \mathcal{K}_{t,k}$.

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

Proof. By the Petrov-Galerkin condition we have that $r_k \perp \mathcal{K}_{t,k}$

$$\begin{aligned} \implies (r_k)^t y &= 0, \ \forall y \in \mathcal{K}_{t,k} \\ (b - Ax_k)^t y &= 0, \ \forall y \in \mathcal{K}_{t,k} \\ b^t y - (x_k)^t A y &= 0, \ \forall y \in \mathcal{K}_{t,k} \end{aligned}$$

Let $y = x_k - x_0 \in \mathcal{K}_{t,k}$

$$\begin{aligned} &\implies (x_k)^t A(x_k - x_0) - b^t(x_k - x_0) = 0 \\ &\implies (x_k)^t A x_k - b^t x_k = (x_k)^t A x_0 - b^t x_0 \\ &\implies \phi(x_k) = \frac{1}{2}(x_k)^t A x_k - b^t x_k = -\frac{1}{2}(x_k)^t A x_k + (x_k)^t A x_0 - b^t x_0 \end{aligned}$$

By showing that $\phi(x) \geqslant \phi(x_k)$, for all $x \in x_0 + \mathcal{K}_{t,k}$ then we have proven that

$$\phi(x_k) = min\{\phi(x), \forall x \in x_0 + \mathcal{K}_k\}. \tag{5}$$

$$\begin{aligned} \phi(x) - \phi(x_k) &= \frac{1}{2}x^t A x - b^t x - [-\frac{1}{2}(x_k)^t A x_k + (x_k)^t A x_0 - b^t x_0] \\ &= \frac{1}{2}x^t A x - b^t z + \frac{1}{2}(x_k)^t A x_k - (x_k)^t A x_0, \ where\, z = x - x_0 \in \mathcal{K}_{t,k} \\ &= \frac{1}{2}x^t A x - (x_k)^t A z + \frac{1}{2}(x_k)^t A x_k - (x_k)^t A x_0, \ since\, b^t z = (x_k)^t A z \\ &= \frac{1}{2}x^t A x - (x_k)^t A x + \frac{1}{2}(x_k)^t A x_k \\ &= \frac{1}{2}(x - x_k)^t A (x - x_k) \geqslant 0, \ since\, A\, is\, positive\, definite \quad \square \end{aligned}$$

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

Proof. $f(x) = ||x^* - x||_A = (x^*)^t A x^* - 2(x^*)^t A x + x^t A x = b^t x^* - 2b^t x + x^t A x = b^t x^* + 2\phi(x)$. The minimum of $f(x)$ is given by $f'(x) = \bigtriangledown\phi(x) = 0$. $\square$

3.1.3 Convergence analysis

The conjugate gradient method of Hestenes and Stiefel is known to converge in $\overline{K}$ iterations where $\overline{K} \leqslant n$, if the matrix $A \in \mathbb{R}^{n,n}$ is SPD. Moreover, the k^{th} error of CG $\overline{e}_k = ||x^* - \overline{x}_k|| \leqslant 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k ||\overline{e}_0||_A$ where $\kappa = ||A||_2||A^{-1}||_2 = \frac{\lambda_{max}}{\lambda_{min}}$ is the L2-condition number of the matrix A.

Assuming that the k^{th} residual of the new conjugate gradient methods $r_k \perp \mathcal{K}_{t,k}$, then by Theorem 3.8 and Theorem 3.9 we have that

$$\begin{aligned} ||e_k||_A = ||x^* - x_k||_A &= min\{||x^* - x||_A, \forall x \in x_0 + \mathcal{K}_{t,k}\} && (6) \\ &\leqslant min\{||x^* - \overline{x}||_A, \forall \overline{x} \in x_0 + \mathcal{K}_k\} \text{ since } \mathcal{K}_k \subset \mathcal{K}_{t,k} && (7) \\ &\leqslant ||\overline{e}_k||_A && (8) \end{aligned}$$

Therefore, our methods converge at least as fast as the Classical Conjugate Gradient method, assuming that the Petrov Galeriken condition is respected ($r_k \perp \mathcal{K}_{t,k}$). Hence, the enlarged Krylov subspace CG methods will converge in K iterations, where $K \leqslant \overline{K} \leqslant n$.

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 $p_i^k = T_i(r_{k-1}) + \beta_i^k p_i^{k-1}$ 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 $r_k = r_{k-1} - AP_k\alpha_k$ with $\alpha_k = (P_k^t AP_k)^{-1} P_k^t r_{k-1}$ and $\beta_k = [\beta_1^k, \beta_2^k, ..., \beta_t^k] = (P_{k-1}^t AP_{k-1})^{-1} P_{k-1}^t Ar_{k-1}$. However, the P_k's are not A-othogonal implying that $r_k \not\perp \mathcal{K}_{t,k}$. 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 (1) (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

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

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 α_k is a vector of size t.

The minimum of $\phi(x)$ is given by $\bigtriangledown\phi(x) = 0$, which is equivalent to $Ax - b = 0$. Thus, by minimizing $\phi(x)$, we are solving the system (1). Note that since $\phi(x_k) = min\{\phi(x), \forall x \in x_0 + \mathcal{K}_{t,k}\}$, then

$$\phi(x_k) = \phi(x_{k-1} + P_k\alpha_k) = min\{\phi(x_{k-1} + P_k\alpha), \forall \alpha \in \mathbb{R}^t\}. \quad (9)$$

Once x_k has been chosen, either x_k 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 $p_i^1 = T_i(r_0)$ and $p_i^{k+1} = T_i(r_k) + \beta_i^{k+1} p_i^k$ for $i = 1, 2, ..., t$. But unllike MSD-CG, MSDO-CG is a projection method. Hence, we A-orthonormalize all the search directions, P_{k+1}, to ensure that $r_{k+1} \perp \mathcal{K}_{t,k+1}$ as discussed in section 3.2.2. By imposing the orthogonality condition, $r_{k+1} \perp \mathcal{K}_{t,k+1}\}$, it is guarenteed that MSDO-CG converges at least as fast as CG as prooven in section 3.1.3.

This procedure is repeated until convergence. Thus, we need to find the recursion relations of r_k, P_k, α_k, and $\beta_k = [\beta_1^k, \beta_2^k, ..., \beta_t^k]^t$.

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:

$$\begin{aligned} r_k &= b - Ax_k \\ &= b - A(x_{k-1} + P_k\alpha_k) \\ &= r_{k-1} - AP_k\alpha_k \end{aligned}$$

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 $(r_k)^t y = 0$ for all $y \in \mathcal{K}_{t,k}$, implies the A-orthogonality of the block search directions $P_i^t AP_j = 0$, for all $i \neq j$, and $i, j \leqslant k$.*

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 $c \geqslant 0$. By the Petrov-Galerkin condition $r_{k-1}^t P_i = 0$ for $i \leqslant k-1$ and $r_k^t P_i = 0$. Thus, $r_k^t P_i = r_{k-1}^t P_i - \alpha_k^t P_k^t AP_i = 0$ for $i \leqslant k-1$. This implies that $P_k^t AP_i = 0$ for $i \leqslant k-1$ 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 $p_i^1 = T_i(r_0)$ and $p_i^k = T_i(r_{k-1}) + \beta_i^k p_i^{k-1}$ for $i = 1, 2, ..., t$. $p_i^k \in \mathcal{K}_{t,k}$ for $i = 1, 2, ..., t$ and $P_k\alpha_k \in \mathcal{K}_{t,k}$.

The recursion relation of P_k is defined as follows

$$P_k = T(r_{k-1}) + P_{k-1}diag(\beta_k), \tag{10}$$

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

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

There are several A-orthonormalization methods. First, for A-orthonormalizing P_k against all the previous P_i, where $i = 1, 2, .., k-1$, 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 A-orthonormalizing 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 A-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 A-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 A-orthonormalization methods.

Note that in Appendix B, we discuss the A-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 A-orthonormalization based on classical Gram Schmidt that A-orthonormalizes P_k against previous vectors. And Algorithm 20 A-orthonormalizes P_k's vectors against each others using a classical Gram Schmidt.

3.2.3 Finding the expression of α_{k+1} and β_{k+1}

At each iteration the step α_{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 $\phi(x) = \frac{1}{2}x^tAx - x^tb$.

$$\begin{aligned}
\text{Then, } F(\alpha) &= \frac{1}{2}(x_k + P_{k+1}\alpha)^tA(x_k + P_{k+1}\alpha) - (x_k + P_{k+1}\alpha)^tb \\
&= \phi(x_k) + \frac{1}{2}[(x_k)^tAP_{k+1}\alpha + \alpha^t(P_{k+1})^tAx_k + \alpha^t(P_{k+1})^tAP_{k+1}\alpha] - \alpha^t(P_{k+1})^tb \\
&= \phi(x_k) + \frac{1}{2}[(x_k)^tAP_{k+1}\alpha - \alpha^t(P_{k+1})^tAx_k] + \frac{1}{2}\alpha^t(P_{k+1})^tAP_{k+1}\alpha - \alpha^t(P_{k+1})^tr_k \\
&= \phi(x_k) + \frac{1}{2}\alpha^t(P_{k+1})^tAP_{k+1}\alpha - \alpha^t(P_{k+1})^tr_k, \quad since\ \ A\ is\ SPD
\end{aligned}$$

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

$$\Rightarrow F'(\alpha) = (P_{k+1})^tAP_{k+1}\alpha - (P_{k+1})^tr_k = 0$$

Therefore, $\boxed{\alpha_{k+1} = (P_{k+1}^tAP_{k+1})^{-1}(P_{k+1}^tr_k)}$.

As for β_{k+1}, it should be chosen to ensure that P_{k+1} is A-orthogonal to P_k. $P_{k+1} = T(r_k) + P_kdiag(\beta_{k+1})$ and $P_k^tAP_{k+1} = P_k^tAT(r_k) + P_k^tAP_kdiag(\beta_{k+1})$. Since P_k is an A-orthonormal matrix, $P_k^tAP_k = I$, $diag(\beta_{k+1})$ should be equal to $-P_k^tAT(r_k)$. But nothing guarantees that $P_k^tAT(r_k)$ is

a diagonal matrix. So we choose $\boxed{\beta_{k+1} = (P_k^t A P_k)^{-1} P_k^t A r_k}$ which guarantees that $P_{k+1} * \mathbb{1}_t$ is A-orthogonal to P_k, similarly to MSD-CG. Moreover, in case $P_k^t AT(r_k)$ is a diagonal matrix, then our choice of β_{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 ($P_{k+1}^t A P_{k+1} = I$), then α_{k+1} and β_{k+1} systems are reduced to $\boxed{\alpha_{k+1} = P_{k+1}^t r_k}$ and $\boxed{\beta_{k+1} = -P_k^t A r_k}$.

3.2.4 The MSDO-CG Algorithm

After deriving the recurrence relations of x_k, r_k, P_k, α_k, and β_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 5.1).

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 $W_{k+1} = AP_{k+1}$ by

$$\begin{cases} W_1 &= AP_1 \\ W_{k+1} &= AT(r_k) + AP_k diag(\beta) \quad \forall k \geqslant 1 \\ &= AT(r_k) + W_k diag(\beta) \end{cases}$$

Algorithm 1 MSDO-CG algorithm — Flops

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: ϵ, the stopping tolerance; k_{max}, the maximum allowed iterations
Output: x_k, the approximate solution of the system $Ax = b$

1:	$r = b - Ax_0$, $\rho = \|\|r\|\|_2^2$, $k = 1$	$2\text{nnz} + 2n - 1$
2:	Let $P_1 = T(r_0)$ and $W_1 = AP_1$	$2\text{nnz} - (t-1)n$
3:	A-orthonormalize P_1	not included here
4:	**while** ($\sqrt{\rho} > \epsilon \|\|b\|\|_2$ and $k < k_{max}$) **do**	$2n$
5:	$\alpha = (P_k^t W_k)^{-1}(P_k^t r) = P_k^t r$	$(2n-1)t$
6:	$x = x + P_k \alpha$	$(2t-1)n + n$
7:	$r = r - W_k \alpha$	$(2t-1)n + n$
8:	$\rho = \|\|r\|\|_2^2$	$2n - 1$
9:	$\beta = -(P_k^t W_k)^{-1}(W_k^t r) = -W_k^t r$	$(2n-1)t$
10:	$P_{k+1} = T(r) + P_k diag(\beta)$	$2nt$
11:	$W_{k+1} = AT(r_k) + W_k diag(\beta)$	$2\text{nnz} - (t-1)n + 2nt$
12:	A-orthonormalize P_{k+1} against all P_i's for $i \leqslant k$	not included here
13:	A-orthonormalize P_{k+1}	not included here
14:	$k = k + 1$	1
15:	**end while**	

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

$$\begin{aligned} \text{Total Flops} &= 4\text{nnz} - nt + 5n - 1 + k_c[11nt - 2t + 2n - 1 + 2\text{nnz} + n + 1] \\ &= 4\text{nnz} - nt + 5n - 1 + k_c[11nt + 3n - 2t + 2\text{nnz}] \\ &\approx 4\text{nnz} + 5n + k_c[11nt + 2\text{nnz}], \end{aligned}$$

which is of the order of $\text{nnz}k_c + ntk_c$ 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 iteratin.

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 α_k and β_k is simply perfoming 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 $x_k = x_{k-1} + Q_k \alpha_k \in x_0 + \mathcal{K}_{t,k}$ 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 $n \times tk$ 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 $\bigtriangledown \phi(x) = 0$ 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

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

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 α_k. By definition, the residual $r_k = b - A x_k$ 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:

$$\begin{aligned} r_k &= b - A x_k \\ &= b - A(x_{k-1} + Q_k \alpha_k) \\ &= r_{k-1} - A Q_k \alpha_k. \end{aligned}$$

At each iteration the step α_{k+1} is chosen such that

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

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

$$\begin{aligned} F(\alpha) &= \frac{1}{2}(x_k + Q_{k+1}\alpha)^t A (x_k + Q_{k+1}\alpha) - (x_k + Q_{k+1}\alpha)^t b \\ &= \phi(x_k) + \frac{1}{2}[(x_k)^t A Q_{k+1}\alpha + \alpha^t (Q_{k+1})^t A x_k + \alpha^t (Q_{k+1})^t A Q_{k+1}\alpha] - \alpha^t (Q_{k+1})^t b \\ &= \phi(x_k) + \frac{1}{2}[(x_k)^t A Q_{k+1}\alpha - \alpha^t (Q_{k+1})^t A x_k] + \frac{1}{2}\alpha^t (Q_{k+1})^t A Q_{k+1}\alpha - \alpha^t (Q_{k+1})^t r_k \\ &= \phi(x_k) + \frac{1}{2}\alpha^t (Q_{k+1})^t A Q_{k+1}\alpha - \alpha^t (Q_{k+1})^t r_k, \quad since \ \ A \ is \ SPD. \end{aligned}$$

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

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

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

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
   $F'(\alpha) = (Q_k)^t AQ_k\alpha - (Q_k)^t r_{k-1} = 0$. 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

$$\begin{aligned} x_k &= x_{k-1} + Q_k\alpha_k, \\ r_k &= r_{k-1} - AQ_k\alpha_k, \\ \alpha_k &= (Q_k^t AQ_k)^{-1}(Q_k^t r_{k-1}), \end{aligned}$$

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 $Solve_\alpha(tk)$.

Algorithm 2 The LRE-CG algorithm		Flops
	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: ϵ, the stopping tolerance; k_{max}, the maximum allowed iterations	
	Output: x_k, the approximate solution of the system $Ax = b$	
1:	$r_0 = b - Ax_0$, $\rho_0 = \|\|r_0\|\|_2^2$, k = 1	2nnz + 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 $k < k_{max}$) **do**	$2n$
4:	$G = (Q^t AQ)$	$(2\text{nnz} - n)tk + (2n-1)t^2k^2$
5:	$\alpha = G^{-1}(Q^t r)$	$(2n-1)tk + Solve_\alpha(tk)$
6:	$x = x + Q\alpha$	$2tkn$
7:	$r = r - AQ\alpha$	$2tkn$
8:	$\rho_k = \|\|r\|\|_2^2$	$2n - 1$
9:	Let $W = AW$	$(2\text{nnz} - n)t$
10:	Orthonormalise the vectors of W against the vectors of Q	not included in here
11:	Orthonormalise the vectors of W and let $Q = [Q\ W]$	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

$$
\begin{aligned}
\text{Total Flops} &= 2\text{nnz} + 7n - 1 + k_c[(2\text{nnz} + 5n - 1)t\tfrac{k_c+1}{2} + 2n + 2\text{nnz}t - nt] \\
&\quad + \textstyle\sum_{k=1}^{k_c} Solve_\alpha(tk) + (2n-1)\tfrac{t^2}{6}(k_c+1)(2k_c+1) \\
&= 2\text{nnz} + 7n - 1 + k_c[(2\text{nnz} + 5n - 1)t\tfrac{k_c+1}{2} + 2n + 2\text{nnz}t - nt] + \textstyle\sum_{k=1}^{k_c} Solve_\alpha(tk) \\
&\quad + (2n-1)\tfrac{t^2}{6}(2k_c^2 + 3k_c + 1) \\
&= 2\text{nnz} + 7n - 1 + (2n-1)\tfrac{t^2}{6} + k_c[(2\text{nnz} + 5n - 1)t\tfrac{k_c+1}{2} + 2n + 2\text{nnz}t - nt \\
&\quad + (2n-1)\tfrac{t^2}{6}(2k_c+3)] + \textstyle\sum_{k=1}^{k_c} Solve_\alpha(tk) \\
&\approx \text{nnz}tk_c^2 + nt^2k_c^2 + \textstyle\sum_{k=1}^{k_c} Solve_\alpha(tk),
\end{aligned}
$$

where the first term $\text{nnz}tk_c^2$ 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

$$
G_k = Q_k^tAQ_k = Q_k^tD_k = \begin{pmatrix} Q_{k-1}^tD_{k-1} & Q_{k-1}^tZ_k \\ W_k^tD_{k-1} & W_k^tZ_k \end{pmatrix} = \begin{pmatrix} G_{k-1} & Q_{k-1}^tZ_k \\ Z_k^tQ_{k-1} & W_k^tZ_k \end{pmatrix} = \begin{pmatrix} G_{k-1} & F_k \\ F_k^t & E_k \end{pmatrix},
$$

where G_{k-1} is computed at iteration $k-1$, $F_k = Q_{k-1}^tZ_k$, 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.

Algorithm 3 The LRE-CG Algorithm — Flops

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: ϵ, the stopping tolerance; k_{max}, the maximum allowed iterations
Output: x_k, the approximate solution of the system $Ax = b$

1:	$r_0 = b - Ax_0$, $\rho_0 = \|\|r_0\|\|_2^2$, 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 $k < k_{max}$) **do**	$2n$
4:	$Z = AW$	$(2\text{nnz} - n)t$
5:	$E = W^tZ$	$(2n-1)t^2$
6:	**if** $k == 1$ **then**	
7:	$D = Z$ and $G = E$	
8:	**else**	
9:	$F = Q(:, 1 : t(k-1))^tZ$	$(2n-1)t^2(k-1)$
10:	$G = \begin{pmatrix} G & F \\ F^t & E \end{pmatrix}$ and $D = [D \;\; Z]$	
11:	**end if**	
12:	$\alpha = G^{-1}(Q^tr)$	$(2n-1)tk + Solve_\alpha(tk)$
13:	$x = x + Q\alpha$	$2tkn$
14:	$r = r - D\alpha$	$2tkn$
15:	$\rho_k = \|\|r\|\|_2^2$	$2n - 1$
16:	Let $W = Z$	
17:	Orthonormalise the columns of W against those of Q	not included in here
18:	Orthonormalise the vectors of W and let $Q = [Q \; W]$	not included in here
19:	k = k+1	1
20:	**end while**	

Then, the cost of k_c iterations of LRE-CG using Algorithm 3, except for the orthonormalization, is

$$\begin{aligned}\text{Total Flops} &= 2\text{nnz} + 7n - 1 + k_c[2\text{nnz}t - nt + (6n-1)t\tfrac{k_c+1}{2} + (2n-1)t^2\tfrac{k_c+1}{2} + 2n] + \sum_{k=1}^{k_c} Solve_\alpha(tk) \\ &= 2\text{nnz} + 7n - 1 + k_c[(2\text{nnz} + 2n - \tfrac{1}{2})t + (n-\tfrac{1}{2})t^2 + (6n-1)t\tfrac{k_c}{2} + (2n-1)t^2\tfrac{k_c}{2} \\ &\quad +2n] + \sum_{k=1}^{k_c} Solve_\alpha(tk) \\ &\approx 3ntk_c^2 + nt^2k_c^2 + nt^2k_c + \sum_{k=1}^{k_c} Solve_\alpha(tk)\end{aligned}$$

Note that tk_c should be much smaller than n, or otherwise the cost of Algorithm 3 would be $O(n^3 + Solve_\alpha(n))$, and n vectors of size n have to be stored in addition to the $n \times n$ system $Q_{k_c}^t A Q_{k_c}$.

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 $\widehat{t}_i$, where $t_0 = t$, $t_i \leqslant t$, $\widehat{t}_i \leqslant t_i$, and $\widehat{t}_i = t_{i+1}$. 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 $tk \times tk$ α 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 α 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 repect 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 kway partitioning [18] that defines the subdomains δ_i. Then x is chosen randomly using matlab's rand function and $b = A * x$, except for the ELASTICITY3Dmatrix where A and b are available. In tables 2, 3, and 4, "Iter" is the number of iterations, k_c, needed for convergence and "Err" is the relative error $\frac{||x-x_{k_c}||_2}{||x||_2}$ at convergence.

The first matrix POISSON2Dis a block tridiagonal matrix obtined from Poisson's equation (sparse) using matlab's gallery('poisson',100) function. The matrices refered to as NH2D, SKY2D, SKY3D, and ANI3D, arise from boundary value problem of the convection diffusion equations

$$\begin{aligned}\eta(x)u + div(\mathbf{a}(x)u) - div(\kappa(x)\nabla u) &= f \; in \; \Omega \\ u &= 0 \; on \; \partial\Omega_D \\ \frac{\partial u}{\partial n} &= 0 \; on \; \partial\Omega_N\end{aligned}$$

where $\Omega = [0,1]^n$, $(n = 2, \text{or } 3)$ and $\partial\Omega_N = \partial\Omega \backslash \partial\Omega_D$. The function η, the vector field $\mathbf{a}$, and the tensor κ are the given coefficients of the partial differential operator. In the 2D case, we have $\partial\Omega_D = [0,1]\times\{0,1\}$, 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 η and $\mathbf{a}$ are both zero. The tensor κ is isotropic and discontinuous. It jumps from the constant value 10^3 in the ring $\frac{1}{2\sqrt{2}} \leqslant |x - c| \leqslant \frac{1}{2}$, $c = (\frac{1}{2}, \frac{1}{2})^T$, to 1 outside.

- SKY2D and SKY3D Skyscraper problems : The tensor κ is isotropic and discontinuous. The domain contains many zones of high permeability which are isolated from each other
$$\kappa(x) = \begin{cases} 10^3 * ([10 * x_2] + 1) & if\ \ [10x_i]\ \ is\ \ odd,\ i = 1, 2 \\ 1, & otherwise. \end{cases}$$
where we note $[x]$ as the integer value of x. SKY2Dand SKY3Dare discretized on a 2D and 3D cartisian grids respetively.

- 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 100×100 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:
$$\begin{aligned} div(\sigma(u)) + f &= 0 && \text{on } \Omega, \\ u &= u_D && \text{on } \partial\Omega_D, \\ \sigma(u).n &= g && \text{on } \partial\Omega_N, \end{aligned}$$
where Ω is a $3D$ $30 \times 10 \times 10$ parallelepiped, Ω_D is the Dirichlet boundary, Ω_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 1: The test matrices

Matrix	Size	Nonzeros	Symetric	2D/3D	Problem
POISSON2D	10000	49600	Yes	2D	Poisson equations
NH2D	10000	49600	Yes	2D	Boundary value
SKY2D	10000	49600	Yes	2D	Boundary value
SKY3D	8000	53600	Yes	3D	Skyscraper
ANI3D	8000	53600	Yes	3D	Anisotropic Layers
ELASTICITY3D	11253	373647	Yes	3D	Linear Elasticity P1 FE

In table 2 we compare the convergence behvior 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, $\mathcal{K}_{t,k}$. We have tested different combinations of A-orthonormalization, but we only show MGS (MGS+MGS), CGS+A-CholQR, CGS+Pre-CholQR, CGS2+A-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 seach directions are not A-othogonal to satisfactory precision. And by Theorem 3.10, this implies that $r_k \not\perp \mathcal{K}_{t,k}$. Thus, nothing guarentees convergence since we have shown in section 3.1.3 that MSDO-CG converges faster than CG if $r_k \perp \mathcal{K}_{t,k}$. 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 partions (Pa) with $x_0 = 0$.

		MSDO-CG with different A-Orthonormalization Methods									
		MGS		CGS+A-CholQR		CGS+Pre-CholQR		CGS2+A-CholQR		CGS2+Pre-CholQR	
	Pa	Iter	Err	Iter	Err	Iter	Err	Iter	Err	Iter	Err
POISSON2D $tol = 10^{-6}$	2	200	4E-5	204	3E-5	204	3E-5	204	3E-5	204	3E-5
	4	167	2E-5	167	2E-5	167	2E-5	167	2E-5	167	2E-5
	8	139	1E-5	139	1E-5	139	1E-5	139	1E-5	139	1E-5
	16	121	5E-6	121	5E-6	121	5E-6	121	5E-6	121	5E-6
	32	94	2E-6	94	2E-6	94	2E-6	94	2E-6	94	2E-6
	64	69	2E-6	69	2E-6	69	2E-6	69	2E-6	69	2E-6
NH2D $tol = 10^{-8}$	2	256	1E-7	256	1E-7	256	1E-7	256	1E-7	256	1E-7
	4	208	1E-7	208	1E-7	208	1E-7	208	1E-7	208	1E-7
	8	169	8E-8	169	8E-8	169	8E-8	169	8E-8	169	8E-8
	16	138	6E-8	138	6E-8	138	6E-8	138	6E-8	138	6E-8
	32	107	2E-8	107	2E-8	107	2E-8	107	2E-8	107	2E-8
	64	77	1E-8	77	1E-8	77	1E-8	77	1E-8	77	1E-8
SKY2D $tol = 10^{-8}$	2	1559	8E-4	–	–	–	–	1562	8E-4	1559	9E-4
	4	917	4E-4	–	–	–	–	917	4E-4	917	4E-4
	8	532	3E-4	–	–	–	–	531	2E-4	534	2E-4
	16	307	1E-4	–	–	–	–	307	1E-4	307	1E-4
	32	178	6E-5	–	–	–	–	178	6E-5	178	6E-5
	64	126	3E-6	–	–	–	–	124	2E-6	124	2E-6
SKY3D $tol = 10^{-8}$	2	610	4E-5	611	4E-5	611	4E-5	611	4E-5	638	1E-5
	4	420	2E-5	–	–	–	–	424	1E-5	418	2E-5
	8	228	1E-5	–	–	–	–	230	1E-5	228	2E-5
	16	134	1E-5	–	–	–	–	134	1E-5	134	1E-5
	32	87	1E-6	–	–	–	–	83	1E-5	83	1E-5
	64	53	6E-6	–	–	–	–	51	1E-5	51	1E-5
ANI3D $tol = 10^{-8}$	2	893	6e-5	893	6e-5	893	6e-5	893	6e-5	893	6e-5
	4	749	8e-5	749	8e-5	749	8e-5	749	8e-5	749	8e-5
	8	498	8e-5	506	9e-5	511	8e-5	498	7e-5	503	7e-5
	16	328	1e-4	–	–	–	–	326	1e-4	326	1e-4
	32	192	2e-4	–	–	–	–	192	1e-4	192	1e-4
	64	122	5e-5	–	–	–	–	122*	4e-5	122*	4e-5

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 POISSON2D and NH2D, both methods converge with the same number of iterations as MSDO-CG with MGS A-orthonormalization. However, for the matrix SKY2D, both methods did not converge. As for the matrices SKY3D and ANI3D, 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 ($cond_2 = ||A||_2||A^{-1}||_2$). The condition number of the matrices POISSON2D and NH2D is 6×10^3, whereas that of the matrices SKY3D, ANI3D

and SKY2D is 1×10^6 , 2×10^6, and 3×10^7 respetively. 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 concude 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 partions Pa, with $x_0 = 0$.

		LRE-CG with different Orthonormalization Methods					
		MGS+MGS		CGS+CGS		CGS+TSQR	
	Pa	Iter	Err	Iter	Err	Iter	Err
POISSON2D $tol = 10^{-6}$	2	193	2E-5	193	2E-5	193	2E-5
	4	153	1E-5	153	1E-5	153	1E-5
	8	123	8E-6	123	8E-6	123	8E-6
	16	95	4E-6	95	4E-6	95	4E-6
	32	70	2E-6	70	2E-6	70	2E-6
	64	52	1E-6	52	1E-6	52	1E-6
NH2D $tol = 10^{-8}$	2	245	1E-7	245	1E-7	245	1E-7
	4	188	1E-7	188	1E-7	188	1E-7
	8	149	5E-8	149	5E-8	149	5E-8
	16	112	3E-8	112	3E-8	112	3E-8
	32	82	2E-8	82	2E-8	82	2E-8
	64	60	1E-8	60	1E-8	60	1E-8
SKY2D $tol = 10^{-8}$	2	1415	5E-04	1415	8E-4	1415	5E-04
	4	757	1E-4	(140)	–	754	1E-4
	8	398	1E-4	(112)	–	398	1E-4
	16	220	9E-5	(70)	–	220	1E-4
	32	126	5E-5	(51)	–	126	5E-5
	64	75	3E-5	(29)	–	75	4E-5
SKY3D $tol = 10^{-8}$	2	557	2E-5	570	1E-5	563	1E-5
	4	373	2E-5	(140)	–	377	1E-5
	8	211	1E-5	(54)	–	211	1E-5
	16	119	9E-6	(37)	–	119	9E-6
	32	69	9E-6	(18)	–	69	9E-6
	64	43	4E-6	(15)	–	42	1E-5
ANI3D $tol = 10^{-8}$	2	875	7e-5	875	7E-5	875	7e-5
	4	673	8e-5	(185)	–	673	8e-5
	8	449	1e-4	(116)	–	449	1e-4
	16	253	2e-4	(16)	–	253	2e-4
	32	148	2e-4	(9)	–	148	2e-4
	64	92	1e-4	(13)	–	92	1e-4

In table 3, we compare the convergence behavior of the LRE-CG method (Algorithm 3) with dif-

Table 4: Comparison between the convergence of the different CG versions with respect to number of partions or initial guesses for Coop-CG with $x_0 = 0$.

		CG		Coop-CG		MSD-CG		MSDO-CG		LRE-CG	
	Pa	Iter	Err	Iter	Err	Iter	Err	Iter	Err	Iter	Err
POISSON2D $tol = 10^{-6}$	2	195	2E-5	206	2E-7	235	3E-1	200	4E-5	193	2E-5
	4	195	2E-5	171	1E-7	252	7E-1	167	2E-5	153	1E-5
	8	195	2E-5	137	1E-7	245	7E-1	139	1E-5	123	8E-6
	16	195	2E-5	106	3E-8	249	7E-1	121	5E-6	95	4E-6
	32	195	2E-5	80	1E-8	240	7E-1	94	2E-6	70	2E-6
	64	195	2E-5	59	1E-8	253	7E-1	69	2E-6	52	1E-6
NH2D $tol = 10^{-8}$	2	259	4E-7	206	2E-7	363	3E-1	256	1E-7	245	1E-7
	4	259	4E-7	179	1E-7	343	7E-1	208	1E-7	188	1E-7
	8	259	4E-7	157	2.02E-5	372	7E-1	169	8E-8	149	5E-8
	16	259	4E-7	107	2E-8	373	7E-1	138	6E-8	112	3E-8
	32	259	4E-7	81	2E-8	324	7E-1	107	2E-8	82	2E-8
	64	259	4E-7	59	1E-8	457	7E-1	77	1E-8	60	1E-8
SKY2D $tol = 10^{-8}$	2	5951	4E-4	4893	2E-4	17907	3E-1	1559	8E-4	1415	5E-04
	4	5951	4E-4	3737	9E-5	66979	7E-1	917	4E-4	757	1E-4
	8	5951	4E-4	3391	1E-5	25298	7E-1	532	3E-4	398	1E-4
	16	5951	4E-4	2437	9E-6	23486	7E-1	307	1E-4	220	9E-5
	32	5951	4E-4	1406	4E-6	15448	7E-1	178	6E-5	126	5E-5
	64	5951	4E-4	802	2E-6	23981	7E-1	126	3E-6	75	3E-5
SKY3D $tol = 10^{-8}$	2	902	1E-5	795	8E-6	3070	2E-1	610	4E-5	557	2E-5
	4	902	1E-5	627	1E-5	11572	6E-1	420	2E-5	373	2E-5
	8	902	1E-5	542	4E-6	3207	7E-1	228	1E-5	211	1E-5
	16	902	1E-5	414	3E-6	4225	7E-1	134	1E-5	119	9E-6
	32	902	1E-5	290	1E-6	3149	7E-1	87	1E-6	69	9E-6
	64	902	1E-5	183	8E-7	2719	7E-1	53	6E-6	43	4E-6
ANI3D $tol = 10^{-8}$	2	4187	4e-5	3584	5e-5	12404	2e-1	893	6e-5	875	7e-5
	4	4146	4e-5	3371	4e-5	17311	6e-1	749	8e-5	673	8e-5
	8	4146	4e-5	2865	4e-5	22339	7e-1	498	8e-5	449	1e-4
	16	4146	4e-5	2314	3e-5	21989	7e-1	328	1e-4	253	2e-4
	32	4146	4e-5	1615	2e-5	17042	7e-1	192	2e-4	148	2e-4
	64	4146	4e-5	1002	1e-5	19257	1e-4	122	5e-5	92	1e-4
ELASTICITY3D $tol = 10^{-8}$	2	987	4e-12	718	3e-12	3065	8e-1	764	3e-12	634	4e-12
	4	987	4e-12	534	8e-12	3497	8e-1	622	4e-12	480	2e-12
	8	987	4e-12	425	6e-11	3101	8e-1	472	1e-12	334	1e-12
	16	987	4e-12	348	6e-11	4239	1e-0	343	1e-12	235	1e-12
	32	987	4e-12	294	9e-12	–	–	234	1e-12	170	1e-12
	64	987	4e-12	235	1e-11	–	–			117	7e-13

ferent orthonormalization schemes for orthonormalizing W against the $n \times tk$ 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 ($O(tklog(t))$ 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 $O(tlog(t))$ messages per iteration. The LRE-CG with CGS converges in the same number of iterations as LRE-CG with MGS for the matrices POISSON2D and NH2D. However, for the other matrices, it does not converge for the given stopping criteria except for $t = 2$ as shown in table 3. The reason is that the the matrix $C = Q^t AQ$ is becoming close to singular, with $rank(C) < tk$, as the iterations proceed due to the loss of orthogonality in the CGS orthogonalization. The number of iterations in parentheses in table 3 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 orthonormalizing 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 3). Thus, we concude that MGS, and CGS+TSQR orthonormalizations are stable enough to be used in the LRE-CG method from Algorithm 3.

In table 4, 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 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 POISSON2D, NH2D, and ELASTICITY3D. But it requires much more iterations than both MSDO-CG and LRE-CG for the other matrices (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.

		POISSON2D		NH2D		SKY2D		SKY3D		ANI3D		ELASTICITY3D	
	Pa	Iter	Mem	Iter	Mem	Iter	Mem	Iter	Mem	Iter	Mem	Iter	Mem
MSDO-CG	2	200	4040004	256	5160004	1559	31220004	610	9792004	893	14320004	764	17239600
	4	167	6740016	208	8380016	917	36740016	420	13488016	749	24016016	622	28064998
	8	139	11220064	169	13620064	532	42660064	228	14672064	498	31952064	472	42603922
	16	121	19540256	138	22260256	307	49300256	134	17296256	328	42128256	343	61959274
	32	94	30421024	107	34581024	178	57301024	87	22545024	192	49425024	234	84646090
	64	69	44824096	77	49944096	126	81304096	53	27668096	122	62996096		
LRE-CG	2	193	4028996	245	5160100	1415	36328900	557	10168996	875	17078500	634	15899134
	4	153	6514544	188	8105504	757	39468784	373	14178064	673	28798864	480	25314666
	8	123	10828256	149	13360864	398	41997856	211	16369344	449	41654464	334	37230106
	16	95	17530400	112	21151264	220	47610400	119	18873216	253	48786304	235	56471386
	32	70	27437600	82	33145376	126	56597024	69	22555264	148	60333696	170	90832426
	64	52	44375584	60	53165600	75	71060000	43	29605504	92	81788544	117	140355114

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 NH2D, 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 ANI3D 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 z flops is $\gamma_c z$, where γ_c is the inverse floating-point rate, also called the floating-point throughput (seconds per floating-point operation). The estimated time for sending a mesages of size k is $\alpha_c + \beta_c k$, where α_c is the latency (with units of seconds) and β_c is the inverse bandwidth (seconds per word). Hence, the estimated runtime of an algorithm with a total of z computed flops and s sent mesages each of size k is the sum of their corresponding estimated times $\gamma_c z + \alpha_c s + \beta_c$.

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 δ_i, for $i = 1, 2, .., t$ the subsets of indices obtained from the partitioning. That is $\delta_i \cap \delta_h = \phi$ for all $i \neq h$, $\cup_{h=1}^{t} \delta_h = \{1, 2, 3, ..., n\}$, 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 $x_0(\overline{\delta}_i)$, where $\overline{\delta}_i = Adjacent(G(A), \delta_i)$ is the adjacent of δ_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 δ_i for $i = 1, 2, .., t$ 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 $x_0(\overline{\delta}_i)$, 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 $i = 1, 2, .., t$, $j = 1, 2, .., c$, and $\delta_i = \cup_{j=1}^{c} \delta_{i,j}$. Then, in Algorithms 4 and 5, $log(t)$ is replaced by $log(ct)$, and $\frac{n}{t}$ is replaced by $\frac{n}{ct}$.

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 22, requires sending $4log(t)$ messages. On the other hand, Algorithm 25 of A-CholQR requires sending $log(t)$ messages, and Pre-CholQR Algorithm 27 requires sending $3log(t)$ messages. The CGS2+A-CholQR and CGS2+Pre-CholQR A-orthonormalizations can be called communication avoiding since they require sending $5log(t)$ and $7log(t)$ 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 4.

Algorithm 4 Parallel MSDO-CG with CGS2+A-CholQR A-orthonormalization — Estimated Time

Input: δ_i, the set of indices assigned to processor i
Input: $A(\delta_i,:)$, the $\frac{n}{t} \times n$ row part of A
Input: $b(\delta_i)$, the $\frac{n}{t} \times 1$ row part of b; $x_0(\overline{\delta}_i)$, the $|\overline{\delta}_i| \approx \frac{n}{t} + c_i$ row part of r_0
Input: ϵ, the stopping tolerance; k_{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 $r(\delta_i) = b(\delta_i) - A(\delta_i, \overline{\delta}_i)x_0(\overline{\delta}_i)$, and let $k = 1$ — $\gamma_c(2\frac{\text{nnz}}{t})$
3: Processor i computes $r(\delta_i)^t r(\delta_i)$ and receives the full $\rho = ||r||_2^2$ — $\gamma_c(2\frac{n}{t})$
via an all reduce (overlapped with the next computation) — $+(\alpha_c + \beta_c)log(t)$
4: Processor i sends $P_1(\delta_i,:) = [T(r)](\delta_i,:) = [0,..,0,r(\delta_i),0,..,0]$ to its — $\gamma_c(2\frac{\text{nnz}}{t})$
m_{MB} neighboring processors and receives from them the corresponding — $+\alpha_c m_{MB}$
blocks to obtain $P_1(\overline{\delta}_i,:)$. Then it computes $W_1(\delta_i) = A(\delta_i, \overline{\delta}_i)P_1(\overline{\delta}_i,:)$ — $+\beta_c \frac{n}{t} m_{MB}$
5: Call A-CholQR algorithm 25 to A-orthonormalize P_1 — $\gamma_c 4nt + (\alpha_c + \beta_c t^2)log(t)$
6: **while** ($\sqrt{\rho} > \epsilon||b||_2$ and $k < k_{max}$) **do**
7: Processor i computes $P_k(\delta_i,:)^t r(\delta_i)$ and receives the full — $\gamma_c(2\frac{n}{t} - 1)t$
$\alpha = P_k^t r$ via an all reduce — $+(\alpha_c + t\beta_c)log(t)$
8: Processor i computes $x(\delta_i) = x(\delta_i) + P_k(\delta_i,:)\alpha$ — $4\gamma_c \frac{n}{t} t$
and $r(\delta_i) = r(\delta_i) - W_k(\delta_i,:)\alpha$
9: Processor i computes $r(\delta_i)^t r(\delta_i)$ and receives $\rho = ||r||_2^2$ — $\gamma_c(2\frac{n}{t} - 1)$
via an all reduce (overlapped with the next communication) — $+(\alpha_c + \beta_c)log(t)$
10: Processor i computes $-W_k(\delta_i,:)^t r(\delta_i)$ and receives the full — $\gamma_c(2\frac{n}{t} - 1)t$
$\beta = -W_k^t r$ via an all reduce — $+(\alpha_c + t\beta_c)log(t)$
11: Processor i computes $P_{k+1}(\delta_i,:) = [T(r)](\delta_i,:) + P_k(\delta_i,:)diag(\beta)$ — $2\gamma_c \frac{n}{t} t$
12: Processor i sends $[T(r)](\delta_i,:)$ to its m_{MB} neighboring processors — $\gamma_c(2\frac{\text{nnz}}{t})$
and receives from them the corresponding blocks to obtain — $+\alpha_c m_{MB}$
$[T(r)](\overline{\delta}_i,:)$. Then it computes $Z(\delta_i) = A(\delta_i, \overline{\delta}_i)[T(r)](\overline{\delta}_i,:)$ — $+\beta_c \frac{n}{t} m_{MB}$
13: Processor i computes $W_{k+1}(\delta_i,:) = Z(\delta_i,:) + W_k(\delta_i,:)diag(\beta)$ — $2\gamma_c \frac{n}{t} t$
14: Call CGS2 Algorithm 22 to A-orthonormalize P_{k+1} against P_i — $12\gamma_c ntk$
for all $i \leqslant k$ — $+2(2\alpha_c + \beta_c t^2 k)log(t)$
15: Call A-CholQR Algorithm 25 to A-orthonormalize P_{k+1} — $\gamma_c 4nt + (\alpha_c + \beta_c t^2)log(t)$
16: $k = k + 1$
17: **end while**
18: **end for**

In Algorithm 4 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 β 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 muliplication, specifically $A[T(r)]$. We denote by $m_{MB} = max\{m_i|\ \ i = 1, 2, .., t\}$ the largest number of neighboring processors, where $m_i \leqslant m_{MB} \leqslant (t-1)$ for all i. Note that processor i has to compute $A(\delta_i, \overline{\delta}_i)[T(r)](\overline{\delta}_i, :)$ where $\overline{\delta}_i = Adjacent(G(A), \delta_i)$. 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 $\overline{\delta}_i \cap \delta_j \neq \phi$. 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 $\frac{n}{t} \times 1$ 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 $j = 1, 2, .., m_i$. Then it computes $A(\delta_i, \overline{\delta}_i)[T(r)](\overline{\delta}_i)$ by performing approximately $\frac{2\text{nnz}-n}{t}$ flops.
In summary, without the A-orthogonalization at steps 14 and 15, the estimated time of k_c iterations of Algorithm 4, where we ignore lower order terms, is

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

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

$$\begin{aligned}\text{Time}_{\text{MSDO-CG}}(k_c) \approx\ & \gamma_c(2n\tfrac{\text{nnz}}{t} + 12ntk_c + 10nt + 17n)k_c + \alpha_c(7log(t) + m_{MB})k_c \\ & +\beta_c(\tfrac{n}{t}m_{MB} + t^2k_clog(t))k_c.\end{aligned}$$

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 4, 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 $2log(t)$ messages. On the other hand, the TSQR orthonormalization using binary trees as discussed in [4] requires sending $log(t)$ messages. The combination of BCGS and TSQR was discussed in [21] and it requires sending only $3log(t)$ 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 $m_{MB} = max\{m_i|\ \ i = 1, 2, .., t\}$ the largest number of neighboring processors, where $m_{MB} \leqslant (t-1)$ for all i. Note that processor i has to compute $A(\delta_i, \overline{\delta}_i)W(\overline{\delta}_i, :)$, where $\overline{\delta}_i = Adjacent(G(A), \delta_i)$. 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 $\overline{\delta}_i \cap \delta_j \neq \phi$. 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 $\frac{n}{t} \times 1$ to

Algorithm 5 Parallel LRE-CG with BCGS+TSQR Algorithm — Estimated time

Input: δ_i, the set of indices assigned to processor i; $A(\delta_i, :)$, the $\frac{n}{t} \times n$ row part of A
Input: $b(\delta_i)$, the $\frac{n}{t} \times 1$ row part of b; $x_0(\overline{\delta}_i)$, the $|\overline{\delta}_i| \times 1$ row part of r_0
Input: ϵ, the stopping tolerance; k_{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 $r(\delta_i) = b(\delta_i) - A(\delta_i, \overline{\delta}_i)x_0(\overline{\delta}_i)$ — $\gamma_c(2\frac{\text{nnz}}{t})$
3: Processor i computes $r(\delta_i)^t r(\delta_i)$ and receives the full $\rho = ||r||_2^2$ via an all reduce (overlapped with the next computation) — $\gamma_c(2\frac{n}{t} - 1) + (\alpha_c + \beta_c)log(t)$
4: Let $Q(\delta_i, :) = W(\delta_i, :) = [T(r)](\delta_i, :)$, and normalise its vectors — $\gamma_c(2\frac{n}{t}t)$
5: Processor i sends $W(\delta_i, :) = [T(r)](\delta_i, :)$ to its m_i neighboring processors and receives from them the corresponding blocks to obtain $W(\overline{\delta}_i, :)$. Let $k = 1$ — $+\alpha_c m_{MB} + \beta_c \frac{n}{t} m_{MB}$
6: **while** ($\sqrt{\rho_{k-1}} > \epsilon||b||_2$ and $k < k_{max}$) **do**
7: Processor i computes $Z(\delta_i, :) = A(\delta_i, \overline{\delta}_i)W(\overline{\delta}_i, :)$ — $2\gamma_c\frac{\text{nnz}}{t}t$
8: Processor i computes $W(\delta_i, :)^t Z(\delta_i, :)$ and receives $E = W^t Z$ via an "all reduce" — $\gamma_c 2\frac{n}{t}t^2 + (\alpha_c + t^2\beta_c)log(t)$
9: **if** $k == 1$ **then**
10: $D = Z$ and $G = E$
11: **else**
12: Processor i computes $Q(\delta_i, 1 : t(k-1))^t Z(\delta_i, :)$ and receives $F = Q(:, 1 : t(k-1))^t Z$ via an "all reduce" — $\gamma_c 2\frac{n}{t}t^2 k + \alpha_c log(t)$
13: $G = \begin{pmatrix} G & F \\ F^t & E \end{pmatrix}$ and $D = [D \;\; Z]$ — $+\beta_c t^2(k-1)log(t)$
14: **end if**
15: Processor i computes $Q(\delta_i, :)^t r(\delta_i)$ and receives $Q^t r$ via an "all reduce" — $\gamma_c 2\frac{n}{t}tk + (\alpha_c + tk\beta_c)log(t)$
16: $\alpha = G^{-1}(Q^t r)$ — $Time_\alpha(tk)$
17: Processor i computes $x(\delta_i) = x(\delta_i) + Q(\delta_i, :)\alpha$ — $\gamma_c(2tk\frac{n}{t})$
18: Processor i computes $r(\delta_i) = r(\delta_i) - D(\delta_i, :)\alpha$ — $\gamma_c(2tk\frac{n}{t})$
19: Processor i computes $r(\delta_i)^t r(\delta_i)$ and receives $\rho_k = ||r||_2^2$ via an "all reduce" (overlapped with the next computation) — $\gamma_c(2\frac{n}{t} - 1) + (\alpha_c + \beta_c)log(t)$
20: Let $W = Z$ and orthogonalise the columns of W against those of Q using BCGS — $\gamma_c 4ntk + 2\alpha_c log(t) + t^2 k\beta_c log(t)$
21: Orthonormalise the vectors of W using TSQR and let $Q = [Q \; W]$ — $\gamma_c(2\frac{n}{t}t^2 + \frac{2}{3}t^3 log(t)) + (\alpha_c + \beta_c\frac{t}{2})log(t)$
22: Processor i sends $W(\delta_i, :)$ to its m_i neighboring processors and receives from them the corresponding blocks to obtain $W(\overline{\delta}_i, :)$ — $+\alpha_c m_{MB} + \beta_c \frac{n}{t} t m_{MB}$
23: k = k+1
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 $\frac{n}{t} \times t$ 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 $E = W^tZ$, $F = Q(:, 1 : t(k-1))^tZ$ and Q^tr 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^2k words in one message. Another alternative is to compute $Q(\delta_i, 1 : t(k-1))^tZ(\delta_i, :)$ 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 $Time_\alpha(tk)$ is the estimated time for solving the $tk \times tk$ α system in line 16. At the k^{th} iteration of Algorithm 5 the total flops, except for the α system, is $2\text{nnz} + (6nt - 2t^2 + 6n - t)k + 2nt + \frac{2}{3}t^3log(t) + 2n + 2\frac{n}{t} - 1$. And $4log(t) + m_{MB}$ messages are sent with $((2t^2 + t)k + \frac{t^2}{2} + t)log(t) + nm_{MB}$ 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

$$\begin{aligned} Time_{\text{LRE-CG}}(k_c) \approx\ & \gamma_c(2\text{nnz} + 3ntk_c + \tfrac{2}{3}t^3log(t))k_c + \alpha_c(4log(t) + m_{MB})k_c \\ & +\beta_c[t^2k_clog(t) + \tfrac{3}{2}t^2log(t) + m_{MB}n]k_c + \textstyle\sum_{k=1}^{k_c} Time_\alpha(tk) \end{aligned}$$

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 gadient 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

$$\begin{aligned} \mathcal{K}_{t,k}(L^{-1}AL^{-t}, r_0) &= span\{T(r_0), L^{-1}AL^{-t}T(r_0), (L^{-1}AL^{-t})^2T(r0), ..., (L^{-1}AL^{-t})^{k-1}T(r_0)\} \\ &= span\{T_1(r_0), T_2(r_0), ..., T_t(r_0), L^{-1}AL^{-t}T_1(r_0), L^{-1}AL^{-t}T_2(r_0), ..., \\ &\quad L^{-1}AL^{-t}T_t(r_0), ..., (L^{-1}AL^{-t})^{k-1}T_1(r_0), ..., (L^{-1}AL^{-t})^{k-1}T_t(r_0)\}, \end{aligned}$$

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 $\widehat{A} = L^{-1}AL^{-t}$, $\widehat{b} = L^{-1}b$, and $\widehat{x} = y$, then all the theorems and properties discussed in section 3.1 are valid for the system $\widehat{A}\widehat{x} = \widehat{b}$.

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.

$$M = \begin{pmatrix} M_{1,1} & 0 & 0 & 0 \\ 0 & M_{2,2} & 0 & 0 \\ 0 & 0 & M_{3,3} & 0 \\ 0 & 0 & 0 & M_{4,4} \end{pmatrix} = \begin{pmatrix} L_{1,1} & 0 & 0 & 0 \\ 0 & L_{2,2} & 0 & 0 \\ 0 & 0 & L_{3,3} & 0 \\ 0 & 0 & 0 & L_{4,4} \end{pmatrix} \begin{pmatrix} L_{1,1}^t & 0 & 0 & 0 \\ 0 & L_{2,2}^t & 0 & 0 \\ 0 & 0 & L_{3,3}^t & 0 \\ 0 & 0 & 0 & L_{4,4}^t \end{pmatrix} \tag{12}$$

We present in Algorithm 6 the split preconditioned MSDO-CG with CGS2+$\widehat{A}$-CholQR $\widehat{A}$-orthonormalization of the system $\widehat{A}\widehat{x} = \widehat{b}$, where $\widehat{A} = L^{-1}AL^{-t}$, $\widehat{b} = L^{-1}b$, $\widehat{x} = y$, and $y = L^t x$. We omit the W recursion due to numerical errors since $W = \widehat{A}P = L^{-1}AL^{-t}P$ consists of performing backward and forward substitution in addition to the matrix vector multiplication. Thus, we use a version of the CGS2 $\widehat{A}$-orthonormalization (Algorithm 23) that computes $W = \widehat{A}P = L^{-1}AL^{-t}P$ and outputs it. As for the $\widehat{A}$-CholQR, by assuming that $W = \widehat{A}P = L^{-1}AL^{-t}P$ is computed, then we can use Algorithm 26 with input $W = \widehat{A}P$. 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 A-orthonormalization, the computation of W, and the backward substitution $L^t x = y$. Note that the split preconditioned MSDO-CG with MGS $\widehat{A}$-orthonormalization did not converge. This might be due to numerical errors in solving the tk backward and forward substitutions in the MGS $\widehat{A}$-orthonormalization. However, the MSDO-CG with CGS2+$\widehat{A}$-CholQR $\widehat{A}$-orthonormalization converges very well as shown in section 6.1.

Algorithm 6 Split preconditioned MSDO-CG algorithm with CGS2+$\widehat{A}$-CholQR $\widehat{A}$-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: ϵ, the stopping tolerance; k_{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 $y = L^t x$

1: $y = L^t x_0$, $r = L^{-1}(b - Ax_0)$, $\rho = ||r||_2^2$, $nb = ||L^{-1}b||_2$, $k = 1$
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** ($\sqrt{\rho} > \epsilon(nb)$ and $k < k_{max}$) **do**
4: $\alpha = (P_k^t W_k)^{-1}(P_k^t r) = P_k^t r$
5: $y = y + P_k\alpha$ and $r = r - W_k\alpha$
6: $\rho = ||r||_2^2$
7: $\beta = -(P_k^t W_k)^{-1}(W_k^t r) = -W_k^t r$
8: $P_{k+1} = T(r) + P_k diag(\beta)$
9: $L^{-1}AL^{-t}$-orthonormalize P_{k+1} against all P_i's for $i \leqslant k$ 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 $L^t x = y$

In Algorithm 7, we present the split preconditioned LRE-CG algorithm with BCGS+TSQR orthonormalization of the system $\widehat{A}\widehat{x} = \widehat{b}$, where $\widehat{A} = L^{-1}AL^{-t}$, $\widehat{b} = L^{-1}b$, $\widehat{x} = y$, and $y = L^t x$. 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 forward and backward 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: ϵ, the stopping tolerance; k_{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 $y = L^t x$

$y = L^t x_0$, $r = L^{-1}(b - Ax_0)$, $\rho_0 = ||r||_2^2$, $nb = ||L^{-1}b||_2$,k = 1
Let $W = T(r_0)$, normalise its vectors and then let $Q = W$
while ($\sqrt{\rho_{k-1}} > \epsilon(nb)$ and $k < k_{max}$) **do**
  $\alpha = (Q^t L^{-1}AL^{-t}Q)^{-1}(Q^t r)$
  $y = y + Q\alpha$
  $r = r - L^{-1}AL^{-t}Q\alpha$ and $\rho_k = ||r||_2^2$
  Let $W = L^{-1}AL^{-t}W$
  Orthonormalise the vectors of W against the vectors of Q
  Orthonormalise the vectors of W and let $Q = [Q\ W]$ and $k = k + 1$
end while
Solve $L^t x = y$

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: ϵ, the stopping tolerance; k_{max}, the maximum allowed iterations
Output: x_k, the approximate solution of the system $Ax = b$

$y = L^t x_0$, $r = L^{-1}(b - Ax_0)$, $\rho_0 = ||r||_2^2$, $nb = ||L^{-1}b||_2$,k = 1
Let $W = T(r_0)$, normalise its vectors and then let $Q = W$
while ($\sqrt{\rho_{k-1}} > \epsilon(nb)$ and $k < k_{max}$) **do**
  $Z = L^{-1}AL^{-t}W$, $E = W^t Z$
  if $k == 1$ **then**
    $D = Z$ and $G = E$
  else
    $D = [D\ \ Z]$, $F = Q(:, 1 : t(k-1))^t Z$, and $G = \begin{pmatrix} G & F \\ F^t & E \end{pmatrix}$
  end if
  $\alpha = G^{-1}(Q^t r)$
  $y = y + Q\alpha$
  $r = r - D\alpha$ and $\rho_k = ||r||_2^2$
  Let $W = Z$
  Orthonormalise the columns of W against those of Q using BCGS
  Orthonormalise the vectors of W using TSQR and let $Q = [Q\ W]$ and $k = k + 1$
end while
Solve $L^t x = y$

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 $L^t x = y$ at the end.

6.1 Convergence

We compare the convergence of split preconditioned MSDO-CG with CGS2+CholQR $\widehat{A}$-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 $x_0 = 0$.

				Split Preconditioned Methods					
		CG		PCG		MSDO-CG		LRE-CG	
	Pa	Iter	Err	Iter	Err	Iter	Err	Iter	Err
POISSON2D $tol = 10^{-6}$	2	195	2E-5	35	1E-5	30	2E-6	30	2E-6
	4			40	1E-5	28	4E-6	28	2E-6
	8			48	2E-5	30	6E-6	27	2E-6
	16			50	1E-5	28	1E-6	25	1E-6
	32			57	2E-5	26	8E-7	23	5E-7
	64			66	2E-5	23	1E-6	20	3E-7
NH2D $tol = 10^{-8}$	2	259	4E-7	47	3E-8	37	6E-9	37	6E-9
	4			55	7E-8	34	2E-8	34	1E-8
	8			65	1E-7	36	1E-8	33	1E-8
	16			71	3E-7	33	1E-8	30	8E-9
	32			83	1E-7	29	1E-8	27	4E-9
	64			88	5E-7	26	5E-9	23	4E-9
SKY2D $tol = 10^{-8}$	2	5855	4E-4	74	3E-7	40	4E-7	40	4E-7
	4			80	2E-6	43	1E-7	36	5E-7
	8			144	2E-5	48	3E-7	31	3E-7
	16			162	1E-4	46	1E-7	27	2E-7
	32			210	3E-4	39	1E-7	23	2E-7
	64			260	2E-7	31	8E-8	20	2E-7
SKY3D $tol = 10^{-8}$	2	902	2E-5	37	2E-6	24	2E-7	24	2E-7
	4			113	2E-5	54	1E-7	43	1E-7
	8			120	8E-6	54	7E-8	33	9E-8
	16			154	1E-5	49	1E-7	28	5E-8
	32			208	1E-5	60	2E-8	30	4E-8
	64			213	1E-5	46	1E-8	22	3E-8
ANI3D $tol = 10^{-8}$	2	4184	4e-5	26	1E-5	31	3E-7	31	3e-7
	4			43	4E-6	39	5e-7	39	6E-7
	8			47	5E-7	39	6E-7	39	5E-7
	16			54	7E-7	43	1E-6	41	6E-7
	32			61	2E-7	47	4e-7	41	1E-6
	64			66	8E-7	46	2E-7	38	4E-7

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 POISSON2D, NH2D, and SKY2D, the number of iteration of the preconditioned LRE-CG and MSDO-CG decreases. As for the matrices SKY3D and ANI3D, 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 $\widehat{A}$-orthonormalization, and LRE-CG with CGS+TSQR orthonormalization method with a fixed Block Jacobi preconditioner , with respect to number of partitions Pa, with $x_0 = 0$.

				Split Preconditioned Methods					
		CG		PCG		MSDO-CG		LRE-CG	
	Pa	Iter	Err	Iter	Err	Iter	Err	Iter	Err
POISSON2D $tol = 10^{-6}$	2	195	2E-5	66	2E-5	62	2E-5	61	7E-6
	4					54	9E-6	50	8E-6
	8					47	4E-6	41	4E-6
	16					39	3E-6	33	1E-6
	32					31	2E-6	25	8E-7
	64					25	8E-7	20	3E-7
NH2D $tol = 10^{-8}$	2	259	4E-7	88	5E-7	82	1E-7	76	7E-8
	4					67	5E-8	63	5E-8
	8					57	3E-8	57	1E-8
	16					46	1E-8	39	2E-8
	32					36	2E-8	36	4E-9
	64					28	7E-9	23	4E-9
SKY2D $tol = 10^{-8}$	2	5773	5E-04	261	2E-4	223	2E-5	184	6E-7
	4					152	4E-7	99	5E-7
	8					109	2E-7	66	4E-7
	16					72	1E-7	44	4E-7
	32					52	5E-8	29	1E-7
	64					34	7E-8	20	2E-7
SKY3D $tol = 10^{-8}$	2	902	2E-5	225	4E-6	191	3E-6	181	5E-6
	4					163	6E-6	135	1E-6
	8					126	2E-6	78	1E-7
	16					94	8E-8	48	9E-8
	32					61	7E-8	28	1E-7
	64					47	3E-8	21	1E-7
ANI3D $tol = 10^{-8}$	2	4184	4E-5	69	8E-7	68	8E-7	66	7e-7
	4					66	4e-7	63	4E-7
	8					61	4E-7	57	3E-7
	16					58	5E-7	52	6E-7
	32					53	6e-7	46	1E-6
	64					45	3E-7	37	8E-7

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 t. First, the graph of 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

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 ANI3D, as the number of partitions is doubled, the number of iterations till convergence decreases by only $10\% - 20\%$. Thus, it is efficent 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 soltion 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 t 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 t_i vectors at each iteration i, where $t_0 = t$, and $t_i \leqslant t$. Then choose $\widehat{t}_i$ linearly independant vectors where $\widehat{t}_i \leqslant t_i$, and $t_{i+1} = \widehat{t}_i$.

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 efficent 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

[1] A. Bhaya, P. Bliman, G. Niedu, and F. A. Pazos. A cooperative conjugate gradient method for linear systems permitting multithread implementation of low complexity. Proceedings of the 51th IEEE Conference on Decision and Control, CDC 2012, Maui, HI, USA. 638-643, December 2012.

[2] A. T. Chronopoulos and W. Gear. s-step Iterative Methods For Symmetric Linear Systems. J. of Comput. Appl. Math., 25(2):153-168, 1989.

[3] T. A. Davis. The University of Florida Sparse Matrix Collection, 1994. Matrices found at http://www.cise.ufl.edu/research/sparse/matrices/

[4] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-avoiding parallel and sequential qr factorizations. SIAM J. Sci. Comput., 34:206-239, 2012.

[5] J. Demmel, M. Heath, and H. van der Vorst, Parallel Numerical Linear Algebra, Acta Numer., Cambridge University Press, Cambridge, UK, 111197, 1993.

[6] J. Erhel. A parallel gmres version for general sparse matrices. Electronic Transactions on Numerical Analysis, 3:160176, 1995.

[7] R. Fletcher. Conjugate gradient methods for indefinite systems. In G.A. Watson, editor, Numerical Analysis, volume 506 of Lecture Notes in Mathematics, pages 7389. Springer Berlin Heidelberg, 1976.

[8] P. Ghysels, T. J. Ashby, K. Meerbergen, and W. Vanroose. Hiding Global Communication Latency in the GMRES Algorithm on Massively Parallel Machines. SIAM J. Sci. Comput., 35(1):4871, 2013.

[9] G. H. Golub and D. P. O'Leary. Some History of the Conjugate Gradient and Lanczos Algorithms: 1948-1976. SIAM Review, 31(1):50-102, March 1989.

[10] P. Gosselet, D. Rixen, F. Roux, and N. Spillane. Simultaneous-FETI and related block strategies: robust domain decomposition methods for engineering problems. Technical Report, Laboratoire de Mécanique et Technologie - LMT , Institute of Applied Mechanics [Garching] , Laboratoire Jacques-Louis Lions - LJLL , Onera - The French Aerospace Lab - Chatillon , Programa ingeneria matematica, August 2014.

[11] L. Grigori, and S. Moufawad. Communication Avoiding ILU0 Preconditioner. Technical Report, ALPINES - INRIA Paris-Rocquencourt, March 2013.

[12] L. Grigori, F. Nataf, and S. Yousef. Robust algebraic Schur complement preconditioners based on low rank corrections. Technical Report, ALPINES - INRIA Paris-Rocquencourt, June 2014.

[13] W. Gropp, Update on Libraries for Blue Waters, http://jointlab.ncsa.illinoisedu/events/workshop3/pdf/presentations/Gropp-Update-on-Libraries.pdf.

[14] T. Gu, X. Liu, Z. Mo, and X. Chi. Multiple search direction conjugate gradient method I: methods and their propositions. International Journal of Computer Mathematics, 81(9):1133-1143, 2004.

[15] F. Hecht. New development in freefem++. Journal of Numerical Mathematics, 20(3-4):251-265, 2012.

[16] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. Journal of research of the National Bureau of Standards, 49:409436, 1952.

[17] M. Hoemmen. Communication-Avoiding Krylov Subspace Methods. PhD thesis, EECS Department, University of California, Berkeley, 2010.

[18] G. Karypis and V. Kumar, Metis 4.0: Unstructured graph partitioning and sparse matrix ordering system, Technical Report, Department of Computer Science, University of Minnesota, 1998.

[19] C. Lanczos. Solution of systems of linear equations by minimized iterations. J. Res. Natl. Bur. Stand, 49:33-53, 1952.

[20] B. Lowery and J. Langou. Stability Analysis of QR factorization in an Oblique Inner Product. ArXiv e-prints. January 2014.

[21] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick. Minimizing communication in sparse matrix solvers. In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC 09, pages 112, New York, NY, USA, 2009. ACM.

[22] D. P. O'Leary. The block conjugate gradient algorithm and related methods. Linear Algebra and Its Applications, 29:293-322, 1980.

[23] D. J. Rixen. Substructuring and Dual Methods in Structural Analysis. PhD thesis, Université de Liège, Belgium, Collection des Publications de la Faculté des Sciences appliquées, n.175, 1997.

[24] M. Rozloznik, M. Tuma, A. Smoktunowicz, and J. Kopal. Numerical stability of orthogonalization with a non-standard inner product. BIT Numerical Mathematics, 52(4):1035-1058, December 2012.

[25] Y. Saad and M. H. Schultz. Gmres: a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. Stat. Comput., 7(3):856869, July 1986.

[26] R. Thakur and W. Gropp. Improving the performance of Collective Operations in MPICH, Mathematics and Computer Science Division Argonne National Laboratory, in Proceeding of the 10th European PVM/MPI Users, 2003.

[27] H. Van der Vorst. Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. SIAM Journal on Scientific and Statistical Computing, 13(2):631-644, 1992

[28] J. Van Rosendale. Minimizing inner product data dependence in conjugate gradient iteration. In Proceeding IEEE International Conference on Parallel Processing, 1983.

[29] H. F. Walker. Implementation of the gmres method using householder transformations. SIAM J. Sci. Stat. Comput., 9(1):152163, January 1988.

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: ϵ, the stopping tolerance; k_{max}, the maximum allowed iterations
Output: x_k, the approximate solution of the system $Ax = b$

$r_0 = b - Ax_0$, $\rho = ||r_0||_2^2$, $k = 1$
for $i = 1, .., t$ **do** Let $P(:, i) = T_i(r_0)$
end for
while ($\sqrt{\rho} > \epsilon||b||_2$ and $k < k_{max}$) **do**
  $\alpha = (P^tAP)^{-1}(P^tr)$
  $x = x + P\alpha$
  $r = r - AP\alpha$
  $\beta = (P^tAP)^{-1}(P^tAr)$
  for $i = 1, .., t$ **do** Let $P(:, i) = T_i(r) + \beta(i)P(:, i)$
  end for
  $\rho = ||r||_2^2$
  $k = k + 1$
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 ($< . , . >_A = < . , A . >$) rather than the L2 inner product ($< . , . >$). 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 $\widetilde{P}_{k+1}$, such that $\widetilde{P}_{k+1}^t A \widetilde{P}_{k+1} = I$. There are two main classes for computing this oblique QR factorization of $P_{k+1} = \widetilde{P}_{k+1}R$. The first class is to factorize the matrix $A = B^tB$ using Cholesky decomposition or eigenvalue decomposition, which is expensive. Then $P_{k+1}^t AP_{k+1} = (BP_{k+1})^t(BP_{k+1})$, where the oblique QR factorization of P_{k+1} is transformed into a Eucleadean QR factorization of the matrix $BP_{k+1} = Q_BR_B$ with $\widetilde{P}_{k+1} = B^{-1}Q_B$ 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 $P_{k+1}^t AP_{k+1}$. 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 ($(tk + 1)log(t) + 2(t - 1)log(t)$ 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 $2log(t)$ 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 $P_{k+1}^t AP_{k+1}$ which is referred to as A-CholQR and requires

only $log(t)$ messages. We also present the Pre-CholQR version that was introduced in [20] and requires $3log(t)$ 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 $i < k+1$, then 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. $P_i(:,j)^t AP_i(:,j) = 1$ for all $j = 1, 2, .., t$ and $i = 1, 2, .., k$, 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,.. , P_{k+1}, the $k+1$ sets of search directions
Output: P_{k+1}, the search directions A-orthonomalized against P_1, P_2,.. , 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: $P_{k+1}(:,o) = P_{k+1}(:,o) - (P_i(:,j)^t AP_{k+1}(:,o))P_i(:,j)$
5: **end for**
6: **end for**
7: $P_{k+1}(:,o) = \frac{P_{k+1}(:,o)}{||P_{k+1}(:,o)||_A} = \frac{P_{k+1}(:,o)}{\sqrt{P_{k+1}(:,o)^t AP_{k+1}(:,o)}}$ %A-normalize
8: **end for**

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

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-orthonomalized 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: $P_{k+1}(:,i) = P_{k+1}(:,i) - (P_{k+1}(:,j)^t AP_{k+1}(:,i))P_{k+1}(:,j)$
4: **end for**
5: $P_{k+1}(:,i) = \frac{P_{k+1}(:,i)}{||P_{k+1}(:,i)||_A} = \frac{P_{k+1}(:,i)}{P_{k+1}(:,i)^t AP_{k+1}(:,i)}$ %A-normalize
6: **end for**

Similarly, the cost of the inner loop is $2\text{nnz} + 3n - 1$ flops and that of the outer loop is $2\text{nnz} + n + 1$, but the total cost is $(2\text{nnz} + 3n - 1)(t-1)\frac{t}{2} + (2\text{nnz} + n + 1)t$ flops, which is of the order of $\text{nnz}t^2 + nt^2$.

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 $AP_{k+1}(:,1)$ is computed $t-1$ times, $AP_{k+1}(:,2)$ is computed $t-2$ times and generally, $AP_{k+1}(:,i)$ is computed $t-i$ times, which means that the matrix A is accessed $(t-1)\frac{t}{2}$ 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 — Flops

Input: P_1, P_2,.. , P_{k+1}, the $k+1$ sets of search directions
Input: W_1, W_2,.. , W_{k+1}, the $k+1$ sets of AP_i
Output: P_{k+1}, the search directions A-orthonomalized against P_1, P_2,.. , P_k

for $o = 1 : t$ **do** %loop over the vectors of P_{k+1}
  for $i = 1 : k$ **do** %loop over the different P_i's
    for $j = 1 : t$ **do** %loop over the vectors of P_i
      $P_{k+1}(:,o) = P_{k+1}(:,o) - (W_i(:,j)^t P_{k+1}(:,o))P_i(:,j)$ — $4n-1$
      $W_{k+1}(:,o) = W_{k+1}(:,o) - (W_i(:,j)^t P_{k+1}(:,o))W_i(:,j)$ — $2n$
    end for
  end for
  $pap_{k+1} = W_{k+1}(:,o)^t P_{k+1}(:,o)$ — $2n-1$
  $P_{k+1}(:,o) = \frac{P_{k+1}(:,o)}{\sqrt{pap_{k+1}}}$ and $W_{k+1}(:,o) = \frac{W_{k+1}(:,o)}{\sqrt{pap_{k+1}}}$ — $2n+2$
end for

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 — Flops

Input: P_{k+1}, the search directions to be A-orthonormalized
Input: W_{k+1}, AP_{k+1}
Output: P_{k+1}, the A-orthonomalized search directions
Output: W_{k+1}, AP_{k+1} where P_{k+1} is the A-orthonomalized search directions

for $i = 1 : t$ **do** %loop over the vectors of P_{k+1}
  for $j = 1 : (i-1)$ **do** %A-orthogonalize against the vectors $P_{k+1}(:, 1 : i-1)$
    $P_{k+1}(:,i) = P_{k+1}(:,i) - (W_{k+1}(:,j)^t P_{k+1}(:,i))P_{k+1}(:,j)$ — $4n-1$
    $W_{k+1}(:,i) = W_{k+1}(:,i) - (W_{k+1}(:,j)^t P_{k+1}(:,i))W_{k+1}(:,j)$ — $2n$
  end for
  $pap_{k+1} = W_{k+1}(:,i)^t P_{k+1}(:,i)$ — $2n-1$
  $P_{k+1}(:,i) = \frac{P_{k+1}(:,i)}{\sqrt{pap_{k+1}}}$ and $W_{k+1}(:,i) = \frac{W_{k+1}(:,i)}{\sqrt{pap_{k+1}}}$ — $2n+2$
end for

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 of 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 $P_{k+1}(:,o)$ against the vector $P_i(:,j)$, where the vector $P_{k+1}(:,o)$ 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 $P_{k+1}(:,o) = P_{k+1}(:,o) - (P_{k+1}(:,o)^t W_i(:,j))P_i(:,j)$ 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, $P_{k+1} = P_{k+1} - [P_i(:,j)]_t diag(P_{k+1}^t W_i(:,j))$.

Algorithm 14 A-orthonormalization against previous vectors with MGS — Flops

Input: P_1, P_2,.. , P_{k+1}, the $k+1$ sets of search directions
Input: W_1, W_2,.. , W_{k+1}, the $k+1$ sets of AP_i
Output: P_{k+1}, the search directions A-orthonomalized against P_1, P_2,.. , P_k

1: **for** $i = 1 : k$ **do** %loop over the different P_i's
2: **for** $j = 1 : t$ **do** %loop over the vectors of P_i
3: $P_{k+1} = P_{k+1} - [P_i(:,j)]_t diag(W_i(:,j)^t P_{k+1})$ — $(4n-1)t$
4: $W_{k+1} = W_{k+1} - [W_i(:,j)]_t diag(W_i(:,j)^t P_{k+1})$ — $2nt$
5: **end for**
6: **end for**
7: **for** $o = 1 : t$ **do**
8: $pap_{k+1}(o) = W_{k+1}(:,o)^t P_{k+1}(:,o)$ — $2n-1$
9: **end for**
10: $pap_{k+1} = (\sqrt{pap_{k+1}})$ — t
11: $P_{k+1} = P_{k+1} diag(pap_{k+1})^{-1}$ and $W_{k+1} = W_{k+1} diag(pap_{k+1})^{-1}$ — $(2n+2)t$

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 $P_{k+1}(:,i+1:t)$ against the A-normalised vector $P_{k+1}(:,i)$ as shown in Algorithm 15. Let $[P_{k+1}(:,j)]_{t-i}$ be an $n \times (t-i)$ block containing $t-i$ duplicates of the vector $P_{k+1}(:,j)$. Then $P_{k+1}(:,i+1:t) = P_{k+1}(:,i+1:t) - [P_{k+1}(:,j)]_{t-i} diag(W_{k+1}(:,i)^t P_{k+1}(:,i+1:t))$

Algorithm 15 A-orthonormalization against each others with MGS

Input: P_{k+1}, the search directions to be A-orthonormalized
Input: W_{k+1}, AP_{k+1}
Output: P_{k+1}, the A-orthonomalized search directions
Output: W_{k+1}, AP_{k+1} where P_{k+1} is the A-orthonomalized search directions

1: **for** $i = 1 : (t-1)$ **do** %A-orthogonalize against the vectors $P_{k+1}(:, 1:i-1)$
2: $P_{k+1}(:, i+1:t) = P_{k+1}(:, i+1:t) - [P_{k+1}(:,j)]_{t-i} diag(W_{k+1}(:,i)^t P_{k+1}(:, i+1:t))$
3: $W_{k+1}(:, i+1:t) = W_{k+1}(:, i+1:t) - [W_{k+1}(:,j)]_{t-i} diag(W_{k+1}(:,i)^t P_{k+1}(:, i+1:t))$
4: $pap_{k+1} = W_{k+1}(:, i+1)^t P_{k+1}(:, i+1)$
5: $P_{k+1}(:, i+1) = \frac{P_{k+1}(:,i+1)}{\sqrt{pap_{k+1}}}$ and $W_{k+1}(:, i+1) = \frac{W_{k+1}(:,i+1)}{\sqrt{pap_{k+1}}}$
6: **end for**

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

sors with distributed memory, and each processor pi is assigned a rowwise part of all W_j ($W_j(\delta_{pi},:)$) for $j = 1, 2, .., k+1$ and the same rowwise part of all P_j ($P_j(\delta_{pi},:)$) for $j = 1, 2, .., k+1$ where $\delta_{pi} \cap \delta_h = \phi$ for all $pi \neq h$ and $\cup_{h=1}^{t} \delta_h = \{1, 2, 3, ..., n\}$.

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

Finally, each processor pi computes its corresponding part of the dot product $W_{k+1}(\delta_i, o)^t P_{k+1}(\delta_i, o)$ for all $o = 1, 2, .., t$ and an "all reduce" is used to send pap_{k+1}'s value to all the processors. Then, each processor A-normalizes $P_{k+1}(\delta_{pi}, o)$ and $W_{k+1}(\delta_{pi}, o)$. 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 $tlog(t)$ words. So, in total $(tk + 1)log(t)$ messages and $(tk + 1)tlog(t)$ words are sent in Algorithm 12. Hence, by ignoring lower order terms we obtain

$$Time_{MGS1_{Aort}} \approx \gamma_c 6nt^2k + \alpha_c tklog(t) + \beta_c t^2 klog(t)$$

As for the parallelization of algorithm 15, it is similar to that of algorithm 14 where at each inner iteration processor pi computes a part of the matrix vector multiplication $W_{k+1}(\delta_{pi}, i)^t P_{k+1}(\delta_{pi}, i+1 : t)$ and then receives the whole $1 \times t$ vector, $W_{k+1}(:, i)^t P_{k+1}(:, i + 1 : t)$, using an "all reduce". Then, it computes $P_{k+1}(\delta_{pi},:)$, $W_{k+1}(\delta_{pi},:)$ and a part of the dot product pap_{k+1} and receives the whole dot product by an "all reduce". Finally, each processor A-normalizes its part of $P_{k+1}(:, i)$ and $W_{k+1}(:, i)$. 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 $2(t - 1)log(t)$ messages are sent in Algorithm 15 where $(t - 1)(t + 1)log(t)$ words are sent. Hence, by ignoring lower order terms we obtain

$$Time_{MGS2_{Aort}} \approx \gamma_c 3nt^2 + \alpha_c 2tlog(t) + \beta_c t^2 log(t)$$

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.

$$\begin{aligned} \text{More precisely, } \widetilde{P}_{k+1}(:, o) &= P_{k+1}(:, o) - \sum_{i=1}^{k} \sum_{j=1}^{t} (P_i(:, j)^t A P_{k+1}(:, o)) P_i(:, j) \\ &= P_{k+1}(:, o) - \sum_{i=1}^{k} P_i (P_i^t A P_{k+1}(:, o)) \end{aligned}$$

If we let $W_{k+1} = AP_{k+1}$, then $\widetilde{P}_{k+1}(:, o) = P_{k+1}(:, o) - \sum_{i=1}^{k} P_i(P_i^t W_{k+1}(:, o))$. Moreover, $\widetilde{P}_{k+1} = P_{k+1} - \sum_{i=1}^{k} P_i(P_i^t W_{k+1})$. Let $Q_k = [P_1, P_2, ..., P_k]$, then $\widetilde{P}_{k+1} = P_{k+1} - Q_k(Q_k^t W_{k+1})$. This represents a Block classical gram schmidt (BCGS) version of the A-orthonormalization.

The total flops performed in Algorithm 17 is $2(2\text{nnz} - n)t + (2n - 1)t^2k + 2t^2kn + 3nt = 4\text{nnz}t + nt + [4nt^2 - t^2]k \approx 4\text{nnz}t + 4nt^2k$.

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 pi is assigned a rowwise part of A ($A(\delta_{pi},:)$), a rowwise part of Q_k ($Q_k(\delta_{pi},:)$) and a rowwise part of P_{k+1}, where $\delta_{pi} \cap \delta_h = \phi$ for all $pi \neq h$ and $\cup_{h=1}^{t} \delta_h = \{1, 2, 3, ..., n\}$.

Algorithm 16 A-orthonormalization against previous vectors with CGS

Input: A, the $n \times n$ symmetric positive definite matrix
Input: P_1, P_2,.. , P_{k+1}, the $k+1$ sets of search directions
Output: $\widetilde{P}_{k+1}$, the search directions A-orthonomalized against P_1, P_2,.. , P_k

1: Let $\widetilde{P}_{k+1} = P_{k+1}$
2: **for** $o = 1 : t$ **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: $\widetilde{P}_{k+1}(:, o) = \widetilde{P}_{k+1}(:, o) - (P_i(:, j)^t A P_{k+1}(:, o)) P_i(:, j)$
6: **end for**
7: **end for**
8: $\widetilde{P}_{k+1}(:, o) = \frac{\widetilde{P}_{k+1}(:,o)}{||\widetilde{P}_{k+1}(:,o)||_A} = \frac{\widetilde{P}_{k+1}(:,o)}{\sqrt{\widetilde{P}_{k+1}(:,o)^t A \widetilde{P}_{k+1}(:,o)}}$ %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: $Q_k = [P_1, P_2, ..., P_k]$, the tk search directions
Input: P_{k+1}, the t search directions to be A-orthonormalized
Output: $\widetilde{P}_{k+1}$, the search directions A-orthonomalized against P_1, P_2,.. , P_k

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

First, processor pi 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 $Q_k(\delta_{pi}, :)^t W_{k+1}(\delta_{pi}, :)$ and obtains the full $tk \times t$ matrix $Q_k^t W_{k+1}$ using an "all reduce". Then, processor pi computes $\widetilde{P}_{k+1}(\delta_{pi}, :) = P_{k+1}(\delta_{pi}, :) - Q_k(\delta_{pi}, :)(Q_k^t W_{k+1})$. Another "all reduce" is needed so that processor pi has the full $\widetilde{P}_{k+1}$ needed to compute $\widetilde{W}_{k+1}(\delta_{pi}, :) = A(\delta_{pi}, :)\widetilde{P}_{k+1}$. Processor pi computes t partial dot products of the form $\widetilde{P}_{k+1}(\delta_{pi}, o)^t W_{k+1}(\delta_{pi}, o)$ and obtains the full dot products via an all reduce. Finally each processor A-normalizes its part of P_{k+1}, i.e $\widetilde{P}_{k+1}(\delta_{pi}, i) = \frac{\widetilde{P}_{k+1}(\delta_{pi},i)}{\widetilde{P}_{k+1}(:,i)^t W_{k+1}(:,i)}$ for all $i = 1, 2, .., t$. 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 $W_{k+1} = AP_{k+1}$ has already been computed and it is an input to Algorithm 18 along with $\mathcal{W}_k = AQ_k = [W_1, W_2, ..., W_k]$. 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 $\widetilde{P}_{k+1}$. We assume that it is possible to send a message of size t^2k words at once. Thus, $2log(t)$ messages are sent with $(tk+1)tlog(t)$ 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

$$Time_{BCGS_{Aort}} \approx \gamma_c 6ntk + \alpha_c 2log(t) + \beta_c t^2 klog(t)$$

Algorithm 18 A-orthonormalization against previous vectors with BCGS — Flops

Input: $Q_k = [P_1, P_2, ..., P_k]$, the tk search directions
Input: P_{k+1}, the t search directions to be A-orthonormalized
Input: $W_{k+1} = AP_{k+1}$; $\mathcal{W}_k = AQ_k$
Output: $\widetilde{P}_{k+1}$, the search directions A-orthonomalized against P_1, P_2,.. , P_k; $\widetilde{W}_{k+1} = A\widetilde{P}_{k+1}$

1: $\widetilde{P}_{k+1} = P_{k+1} - Q_k(Q_k^t W_{k+1})$ — $(2n-1)t^2k + (2tk-1)nt + nt$
2: $\widetilde{W}_{k+1} = W_{k+1} - \mathcal{W}_k(Q_k^t W_{k+1})$ — $2nt^2k$
3: **for** $i = 1 : t$ **do** %loop over the vectors of P_{k+1} and A-normalize
4: $\widetilde{P}_{k+1}(:,i) = \frac{\widetilde{P}_{k+1}(:,i)}{||\widetilde{P}_{k+1}(:,i)||_A} = \frac{\widetilde{P}_{k+1}(:,i)}{\sqrt{\widetilde{P}_{k+1}(:,i)^t \widetilde{W}_{k+1}(:,i)}}$ — $3n$
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 $P_{k+1}^t AP_{k+1}$ is equal to ones, we A-orthonormalize it ($P_{k+1}^t AP_{k+1} = I$) 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
Input: P_{k+1}, the search directions to be A-orthonormalized
Output: P_{k+1}, the A-orthonomalized search directions

1: Let $\widetilde{P}_{k+1} = P_{k+1}$
2: **for** $i = 1 : t$ **do** %loop over the vectors of P_{k+1}
3: **for** $j = 1 : (i-1)$ **do** %A-orthogonalize against the vectors $P_{k+1}(:, 1 : i-1)$
4: $\widetilde{P}_{k+1}(:,i) = \widetilde{P}_{k+1}(:,i) - (\widetilde{P}_{k+1}(:,j)^t AP_{k+1}(:,i))\widetilde{P}_{k+1}(:,j)$
5: **end for**
6: $\widetilde{P}_{k+1}(:,i) = \frac{\widetilde{P}_{k+1}(:,i)}{||\widetilde{P}_{k+1}(:,i)||_A} = \frac{\widetilde{P}_{k+1}(:,i)}{\sqrt{\widetilde{P}_{k+1}(:,i)^t A\widetilde{P}_{k+1}(:,i)}}$ %A-normalize
7: **end for**

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

$$P_{k+1} = \widetilde{P}_{k+1} R$$

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

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & \cdots & r_{1,t} \\ & r_{2,2} & r_{2,3} & \cdots & r_{2,t} \\ & & r_{3,3} & \cdots & r_{3,t} \\ & & & \ddots & \vdots \\ & & & & r_{t,t} \end{pmatrix} = \begin{pmatrix} ||p_1||_A & < \tilde{p}_1, p_2 >_A & < \tilde{p}_1, p_3 >_A & \cdots & < \tilde{p}_1, p_t >_A \\ & r_{2,2} & < \tilde{p}_2, p_3 >_A & \cdots & < \tilde{p}_2, p_t >_A \\ & & r_{3,3} & \cdots & < \tilde{p}_3, p_t >_A \\ & & & \ddots & \vdots \\ & & & & r_{t,t} \end{pmatrix}$$

Although the CGS A-orthonormalization is equivalent to a QR factorization with the A inner poduct, we were not able parallelize it using reduction trees with the same communication pattern as in TSQR [4]. 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 — Flops

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

1:	$W_{k+1} = AP_{k+1}$	$(2\text{nnz} - n)t$
2:	$R(1,1) = \sqrt{P_{k+1}(:,1)^t W(:,1)}$	$2n$
3:	$\widetilde{P}_{k+1}(:,1) = \frac{P_{k+1}(:,1)}{R(1,1)}$	n
4:	**for** $i = 2 : t$ **do**	
5:	$R(i-1, i:t) = \widetilde{P}_{k+1}(:,i-1)^t W_{k+1}(:,i:t)$	$(2n-1)(t-i+1)$
6:	$\widetilde{P}_{k+1}(:,i) = P_{k+1}(:,i) - \widetilde{P}_{k+1}(:,1:i-1)R(1:i-1,i)$	$(2(i-1)-1)n + n$
7:	$R(i,i) = \sqrt{\widetilde{P}_{k+1}(:,i)^t A \widetilde{P}_{k+1}(:,i)}$	$(2\text{nnz} - n) + 2n$
8:	$\widetilde{P}_{k+1}(:,i) = \frac{\widetilde{P}_{k+1}(:,i)}{R(i,i)}$	n
9:	**end for**	

The total flops of Algorithm 20 is of the order of $\text{nnz}t + nt^2$.

$$\begin{aligned}
\text{Total} &= 2\text{nnz}t - nt + 3n + \textstyle\sum_{i=2}^{t}[(2n-1)(t-i+1) + 2(i-1)n + (2\text{nnz} + 2n)] \\
&= 2\text{nnz}t - nt + 3n + \textstyle\sum_{i=2}^{t}[(2n-1)(t+1) - (2n-1)i + 2ni - 2n + 2\text{nnz} + 2n] \\
&= 2\text{nnz}t - nt + 3n + \textstyle\sum_{i=2}^{t}[(2n-1)(t+1) + i + 2\text{nnz}] \\
&= 2\text{nnz}t - nt + 3n + [(2n-1)(t+1) + 2\text{nnz}](t-1) + \tfrac{t(t+1)}{2} - 1 \\
&= 4\text{nnz}t - 2\text{nnz} - nt + 3n + (2n-1)(t^2-1) + \tfrac{t^2+t}{2} - 1 \\
&= 4\text{nnz}t - 2\text{nnz} - nt + n + 2nt^2 - t^2 + \tfrac{t^2+t}{2} \\
&= 4\text{nnz}t - 2\text{nnz} - nt + n + 2nt^2 + \tfrac{-t^2+t}{2}
\end{aligned}$$

The parallelization of Algorithm 20 starts by distributing the data similarly to Algorithm 17. Processor pi computes $A(:,\delta_{pi})P_{k+1}(\delta_{pi},:)$ and receives W_{k+1} via an "all reduce", and computes $P_{k+1}(\delta_{pi},1)^t W(\delta_{pi},1)$, 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 $\widetilde{P}_{k+1}(\delta_{pi},1) = \frac{P_{k+1}(\delta_{pi},1)}{R(1,1)}$.

At each iteration , processor pi computes $\widetilde{P}_{k+1}(\delta_{pi},i-1)^t W_{k+1}(\delta_{pi},i:t)$ and receives the full $R(i-1,i:t)$ by an all reduce. Then, it computes $\widetilde{P}_{k+1}(\delta_{pi},i) = P_{k+1}(\delta_{pi},i) - \widetilde{P}_{k+1}(\delta_{pi},1:i-1)R(1:i-1,i)$ and recieves the $P_{k+1}(\beta_{pi},i)$ from m_{MB} adjacent processors where $\beta_{pi} = Adjacent(G(A),\delta_{pi})$. Then it computes $\widetilde{W}_{k+1}(\delta_{pi},i) = A(\delta_{pi},\beta_{pi})\widetilde{P}_{k+1}(\beta_{pi},i)$ and $\widetilde{P}_{k+1}(\delta_{pi},i)^t \widetilde{W}_{k+1}(\delta_{pi},i)$ and receives the full $\widetilde{P}_{k+1}(:,i)^t A \widetilde{P}_{k+1}(:,i)$, needed to compute $R(i,i)$, via an all reduce. Finally, it computes $\widetilde{P}_{k+1}(\delta_{pi},i) = \frac{\widetilde{P}_{k+1}(\delta_{pi},i)}{R(i,i)}$. 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 $(1 + \sum_{i=2}^{t} t + 2 - i)log(t) = \frac{t(t+1)}{2}log(t)$ words where $\frac{3nt^2+nt+\frac{t(1-t)}{2}}{t} = 3nt + n + \frac{(1-t)}{2}$ flops are performed in parallel. Hence, by ignoring lower order terms we obtain

$$Time_{QRCGS_{Aort}} \approx \gamma_c 3nt + \alpha_c 2tlog(t) + \beta_c t^2 log(t)$$

Algorithm 21 QR factorization with A inner product using CGS	Flops
Input: P_{k+1}, the search directions to be A-orthonormalized; W_{k+1}, $= AP_{k+1}$	
Output: $\widetilde{P}_{k+1}$, the A-orthonomalized search directions; $\widetilde{W}_{k+1}$	
Output: R, the upper triangular matrix such that $P_{k+1} = \widetilde{P}_{k+1}R$	
$R(1,1) = \sqrt{P_{k+1}(:,1)^t W(:,1)}$	$2n$
$\widetilde{P}_{k+1}(:,1) = \frac{P_{k+1}(:,1)}{R(1,1)}$ and $\widetilde{W}_{k+1}(:,1) = \frac{W_{k+1}(:,1)}{R(1,1)}$	$2n$
for $i = 2 : t$ **do**	
$\quad R(i-1, i:t) = \widetilde{P}_{k+1}(:,i-1)^t W_{k+1}(:,i:t)$	$(2n-1)(t-i+1)$
$\quad \widetilde{P}_{k+1}(:,i) = P_{k+1}(:,i) - \widetilde{P}_{k+1}(:,1:i-1)R(1:i-1,i)$	$(2(i-1)-1)n + n$
$\quad \widetilde{W}_{k+1}(:,i) = W_{k+1}(:,i) - \widetilde{W}_{k+1}(:,1:i-1)R(1:i-1,i)$	$2(i-1)n$
$\quad R(i,i) = \sqrt{\widetilde{P}_{k+1}(:,i)^t \widetilde{W}_{k+1}(:,i)}$	$2n$
$\quad \widetilde{P}_{k+1}(:,i) = \frac{\widetilde{P}_{k+1}(:,i)}{R(i,i)}$ and $\widetilde{W}_{k+1}(:,i) = \frac{\widetilde{W}_{k+1}(:,i)}{R(i,i)}$	$2n$
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}, $= AP_{k+1}$; $\mathcal{W}_k$, $= AQ_k$
Output: $\widetilde{P}_{k+1}$, the search directions A-orthonomalized against P_1, P_2,.. , P_k; $\widetilde{W}_{k+1} = A\widetilde{P}_{k+1}$
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 $\widetilde{P}_{k+1}$ and $\widetilde{W}_{k+1}$ as output

In the case of $L^{-1}AL^{-t}$-orthonormalization 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 $P_{k+1} = \widetilde{P}_{k+1}R$ where $\widetilde{P}^t_{k+1}A\widetilde{P}_{k+1} = I$. Thus, $P^t_{k+1}AP_{k+1} = (\widetilde{P}_{k+1}R)^t A\widetilde{P}_{k+1}R = R^tR$ and R can be obtained by performing a Cholesky factorization of the SPD matrix $P^t_{k+1}AP_{k+1}$. Then, $\widetilde{P}_{k+1} = P_{k+1}R^{-1}$ is obtained by solving the lower triangular system $R^t\widetilde{P}^t_{k+1} = P^t_{k+1}$ with multiple right-hand sides. This procedure is called A-CholQR and summarized in Algorithm 24 [24, 20]. 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 δ_i subset of indices defined previously, $P_{k+1}(\delta_i, :)$ and $W_{k+1}(\delta_i, :)$. And each processor i should compute $\widetilde{P}_{k+1}(\delta_i, :)$ and $\widetilde{W}_{k+1}(\delta_i, :)$. Then each processor i computes $W_{k+1}(\delta_i, :)^t P_{k+1}(\delta_i, :)$ and receives the $t \times t$ matrix C via an "all reduce" or equivalently

Algorithm 23 $L^{-1}AL^{-t}$-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: Q_k, the tk search directions
Input: P_{k+1}, the t search directions to be $L^{-1}A(L^t)^{-1}$-orthonormalized
Output: $\widehat{P}_{k+1}$, the search directions $L^{-1}A(L^t)^{-1}$-orthonomalized against Q_k
Output: $\widehat{W}_{k+1} = L^{-1}AL^{-t}\widehat{P}_{k+1}$

$W_{k+1} = L^{-1}AL^{-t}P_{k+1}$
$\widetilde{P}_{k+1} = P_{k+1} - Q_k(Q_k^t W_{k+1})$
$\widetilde{W}_{k+1} = L^{-1}AL^{-t}\widetilde{P}_{k+1}$
for $i = 1 : t$ **do** %loop over the vectors of P_{k+1} and $L^{-1}AL^{-t}$-normalize
  $\widetilde{P}_{k+1}(:,i) = \frac{\widetilde{P}_{k+1}(:,i)}{||\widetilde{P}_{k+1}(:,i)||_{L^{-1}AL^{-t}}} = \frac{\widetilde{P}_{k+1}(:,i)}{\sqrt{\widetilde{P}_{k+1}(:,i)^t \widetilde{W}_{k+1}(:,i)}}$ and $\widetilde{W}_{k+1}(:,i) = \frac{\widetilde{W}_{k+1}(:,i)}{\sqrt{\widetilde{P}_{k+1}(:,i)^t \widetilde{W}_{k+1}(:,i)}}$
end for
$\widehat{P}_{k+1} = \widetilde{P}_{k+1} - Q_k(Q_k^t \widetilde{W}_{k+1})$
$\widehat{W}_{k+1} = L^{-1}AL^{-t}\widehat{P}_{k+1}$
for $i = 1 : t$ **do** %loop over the vectors of P_{k+1} and $L^{-1}AL^{-t}$-normalize
  $\widehat{P}_{k+1}(:,i) = \frac{\widehat{P}_{k+1}(:,i)}{||\widehat{P}_{k+1}(:,i)||_{L^{-1}AL^{-t}}} = \frac{\widehat{P}_{k+1}(:,i)}{\sqrt{\widehat{P}_{k+1}(:,i)^t \widehat{W}_{k+1}(:,i)}}$ and $\widehat{W}_{k+1}(:,i) = \frac{\widehat{W}_{k+1}(:,i)}{\sqrt{\widehat{P}_{k+1}(:,i)^t \widehat{W}_{k+1}(:,i)}}$
end for

Algorithm 24 A-CholQR — Flops

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

Compute $W_{k+1} = AP_{k+1}$ — $(2\text{nnz} - n)t$
Compute $C = W_{k+1}^t P_{k+1}$ — $(2n-1)t^2$
Compute the Cholesky factorization of $C = R^t R$ to obtain R — t^2
Solve $R^t \widetilde{P}_{k+1}^t = P_{k+1}^t$ — nt^2
for $i = 1 : t$ **do**
  $\widetilde{P}_{k+1}(:,i) = \frac{\widetilde{P}_{k+1}(:,i)}{||\widetilde{P}_{k+1}(:,i)||_A} = \frac{\widetilde{P}_{k+1}(:,i)}{\sqrt{\widetilde{P}_{k+1}(:,i)^t A \widetilde{P}_{k+1}(:,i)}}$ — $3n$
end for

Algorithm 25 A-CholQR — Flops

Input: P_{k+1}, the search directions to be A-orthonormalized, $W_{k+1} = AP_{k+1}$
Output: $\widetilde{P}_{k+1}$, the A-orthonomalized search directions; $\widetilde{W}_{k+1} = A\widetilde{P}_{k+1}$

Compute $C = W_{k+1}^t P_{k+1}$ — $(2n-1)t^2$
Compute the Cholesky factorization of $C = R^t R$ to obtain R — t^2
Solve $R^t \widetilde{P}_{k+1}^t = P_{k+1}^t$ — $t^2 n$
Solve $R^t \widetilde{W}_{k+1}^t = W_{k+1}^t$ — $t^2 n$
for $i = 1 : t$ **do**
  $\widetilde{P}_{k+1}(:,i) = \frac{\widetilde{P}_{k+1}(:,i)}{||\widetilde{P}_{k+1}(:,i)||_A} = \frac{\widetilde{P}_{k+1}(:,i)}{\sqrt{\widetilde{P}_{k+1}(:,i)^t \widetilde{W}_{k+1}(:,i)}}$ and — $4n$
  $\widetilde{W}_{k+1}(:,i) = \frac{\widetilde{W}_{k+1}(:,i)}{\sqrt{\widetilde{P}_{k+1}(:,i)^t \widetilde{W}_{k+1}(:,i)}}$
end for

$log(t)$ messages and $t^2log(t)$ words. Finally, each processor i can compute the Cholesky factorization of the matrix C to obtain R which is needed to solve $R^t\widetilde{P}_{k+1}(\delta_i,:)^t = P_{k+1}(\delta_i,:)^t$ and $R^t\widetilde{W}_{k+1}(\delta_i,:)^t = W_{k+1}(\delta_i,:)^t$. Thus, it is possible to parallelize the A-CholQR A-orthonormalization, Algorithm 25, by sending $log(t)$ messages with $t^2log(t)$ words and performing $t^2 + \frac{(4n-1)t^2}{t} \approx 4nt$ flops in parallel. Hence, by ignoring lower order terms we obtain

$$Time_{A-CholQR} \approx \gamma_c 4nt + \alpha_c log(t) + \beta_c t^2 log(t)$$

In the case of $\widehat{A}$-orthonormalization of P_{k+1}, where $\widehat{A} = L^{-1}AL^{-t}$ and $L^{-t} = (L^t)^{-1}$, the $\widehat{A}$-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 $\widehat{A}$-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 A-orthonormalized; W_{k+1}, $= L^{-1}AL^{-t}P_{k+1}$
Output: $\widetilde{P}_{k+1}$, the A-orthonomalized search directions; $\widetilde{W}_{k+1} = L^{-1}AL^{-t}\widetilde{P}_{k+1}$

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

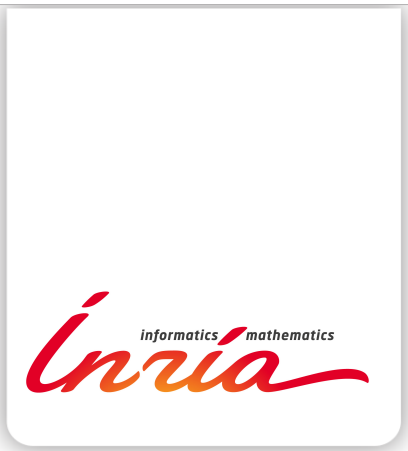
Recently, Lowery and Langou presented a new version of A-CholQR in [20], which they call Pre-CholQR (Algorithm 27). It consists in performing a Euclidean QR factorization with L2 before calling the A-CholQR A-orthonormalization, Algorithm 24. The QR factorization of P_{k+1} can be done using the TSQR [4], which requires sending $log(t)$ messages, each of size $\frac{t^2}{2}$ words and computing $2nt + \frac{2}{3}t^3log(t)$. Then, parallelizing Algorithm 24 requires performing two "all reduce" or $2log(t)$ messages with $(nt + t^2)log(t)$ words. In total, parallelizing Algorithm 27 requires sending $3log(t)$ messages with $(nt + 1.5t^2)log(t)$ words. Hence, by ignoring lower order terms we obtain

$$Time_{PreCholQR} \approx \gamma_c(6nt + +\frac{2}{3}t^3log(t)) + \alpha_c 2log(t) + \beta_c(nt + \frac{3}{2}t^2log(t))$$

Algorithm 27 Pre-CholQR

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

1: Compute the QR factorization of $P_{k+1} = P_{k+1}^{'}R$
2: Call Algorithm 24 with A and $P_{k+1}^{'}$ as input and with $\widetilde{P}_{k+1}$ as output



RESEARCH CENTRE
PARIS – ROCQUENCOURT

Domaine de Voluceau, - Rocquencourt
B.P. 105 - 78153 Le Chesnay Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399