SIAM J. SCI. COMPUT.
Vol. 41, No. 5, pp. C522–C547

SCALABLE LINEAR SOLVERS BASED ON ENLARGED KRYLOV SUBSPACES WITH DYNAMIC REDUCTION OF SEARCH DIRECTIONS*

LAURA GRIGORI† AND OLIVIER TISSOT†

Abstract. Krylov methods are widely used for solving large sparse linear systems of equations. On distributed architectures, their performance is limited by the communication needed at each iteration of the algorithm. In this paper, we study the use of so-called enlarged Krylov subspaces for reducing the number of iterations, and therefore the overall communication, of Krylov methods. In particular, we consider a reformulation of the conjugate gradient method using these enlarged Krylov subspaces: the enlarged conjugate gradient method. We present the parallel design of two variants of the enlarged conjugate gradient method, as well as their corresponding dynamic versions, where the number of search directions is dynamically reduced during the iterations. For a linear elasticity problem with heterogeneous coefficients, using a block Jacobi preconditioner, we show that this implementation scales up to 16,384 cores and is up to 6.9 times faster than the PETSc implementation of PCG.

Key words. Krylov subspace methods, conjugate gradient, communication reducing algorithms

AMS subject classifications. 65F10, 68W10

DOI. 10.1137/18M1196285

1. Introduction.

The discretization of partial differential equations, used to model physical phenomena, or optimization problems leads to linear systems of the form $Ax = b$, where A is a sparse matrix. When A becomes very large, iterative methods based on Krylov subspaces are the method of choice [32]. In this paper, we consider the case where $A \in \mathbb{R}^{n\times n}$ is symmetric ($A^\top = A$) positive definite ($x^\top Ax > 0$ for all $x \neq 0$). The conjugate gradient method [19], and its preconditioned form, is a well-known method for solving such linear systems.

However, solving these linear systems efficiently on large scale computers remains a challenging problem. One difficulty is the high cost of communication compared to the computation on these machines [7, 6]. Recently, a lot of effort has been put into enhancing the performance of Krylov methods by avoiding global communication [4, 3], overlapping communication with computation [13], or decreasing the number of iterations by searching in multiple directions at once [33, 14]. In this paper, we focus on the third approach, more precisely on the enlarged conjugate gradient (ECG) method [14, 16].

After recalling Orthodir and Orthomin variants of ECG, we show the explicit link between the two methods. This gives a rigorous justification of an observation already made concerning the robustness of Orthodir compared to Orthomin in [16] in the case when the search directions are A-orthogonalized as in formula (2.8) given

*Submitted to the journal's Software and High-Performance Computing section July 3, 2018; accepted for publication (in revised form) June 7, 2019; published electronically October 11, 2019.
https://doi.org/10.1137/18M1196285
Funding: This work was supported by the NLAFET project as part of the European Union's Horizon 2020 research and innovation program under grant agreement 6716334. The computational resources were provided by the High Performance Computing Center North (HPC2N) at Umeå (Sweden) and the National Energy Research Scientific Computing Center (NERSC) at Berkeley (U.S.), which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-05CH11231.

†INRIA Paris, Sorbonne Université, Université Paris-Diderot SPC, CNRS, Laboratoire Jacques-Louis Lions, ALPINES team (laura.grigori@inria.fr, olivier.tissot@inria.fr).

in the following section. However, we note that we do not study here the maximum attainable accuracy of the two variants. In [22], the authors consider the GMRES method and they show that Orthomin is better from this point of view in this case. We, however, study theoretically the convergence behavior of ECG, assuming exact arithmetic. We greatly improve the previous result in [14] and show that ECG acts as if the smallest eigenvalues were somehow deflated. Then we present the parallel design of ECG. We consider both Orthodir and Orthomin variants, as well as dynamic versions of these variants that reduce dynamically the number of search directions in order to reduce the extra arithmetic cost in ECG compared to standard CG. In practice, we observe that enlarging the Krylov subspaces can drastically reduce the number of iterations. Indeed, in the numerical experiments it is used with a block Jacobi preconditioner and acts as a second level that, in a way, deflates the smallest eigenvalues; this is in accordance with the theory. This leads to a significant speed-up over standard PCG. For instance, for a 3D linear elasticity problem with heterogeneous coefficients with 4.5 million unknowns and 165 million nonzero entries, we observe that ECG is up to 5.7 times faster than the PETSc implementation of PCG, both using a block Jacobi preconditioner. This test case is known to be difficult because the standard one-level preconditioners are not expected to be very effective [34]. As it increases the arithmetic intensity and reduces the communication, it is well suited for modern and future architectures that exhibit massive parallelism. For the previous elasticity problem, we show that the method can scale up to $16,384$ threads, each one being bound to one physical core, which means that each core owns nearly 280 unknowns.

In summary, the contributions of the paper are the following. We provide a rigorous justification of the lack of robustness of Orthomin compared to Orthodir observed experimentally in [16]. We give a proof of the speed of convergence of ECG which greatly improves the previous existing result presented in [14]. This shows that ECG acts as a second-level preconditioner that mitigates the effect of the smallest eigenvalues on the convergence of the iterative method. Hence it is sufficient to use as preconditioner a highly parallel method such as block Jacobi which bounds the largest eigenvalue of the preconditioned matrix. Finally, we introduce a parallel design embedding several variations of ECG whose scalability is assessed on different matrices and up to $16,384$ cores. We want to point out that our aim is not to design a specific solver for elliptic partial differential equations such as GenEO [34] or multigrid preconditioners with some tuning. For a detailed comparison of such solvers, we refer the reader to Jolivet's thesis [23]. It is very likely that for these test cases, these solvers are more effective than ECG with a block Jacobi preconditioner. Nevertheless, unlike these methods, ECG is an algebraic method. It does not require any information from the underlying partial differential equation and does not rely on any assumption, except that the matrix is symmetric positive definite (SPD). Hence it can be seen as a black-box solver and integrated very easily in any existing code.

2. Enlarged Krylov conjugate gradients.

2.1. Block Krylov methods. In 1980, O'Leary introduced the block CG method [29] for solving SPD systems with several right-hand sides. In this seminal paper, she proved that block CG can converge significantly faster than CG. This idea was then generalized and extended to other standard Krylov methods, such as GMRES [30, 27] or BiCGSTAB [12]. Later, Gutknecht [17] introduced a general framework for defining block Krylov subspaces.

Recently, block Krylov methods have received increasing attention in the HPC

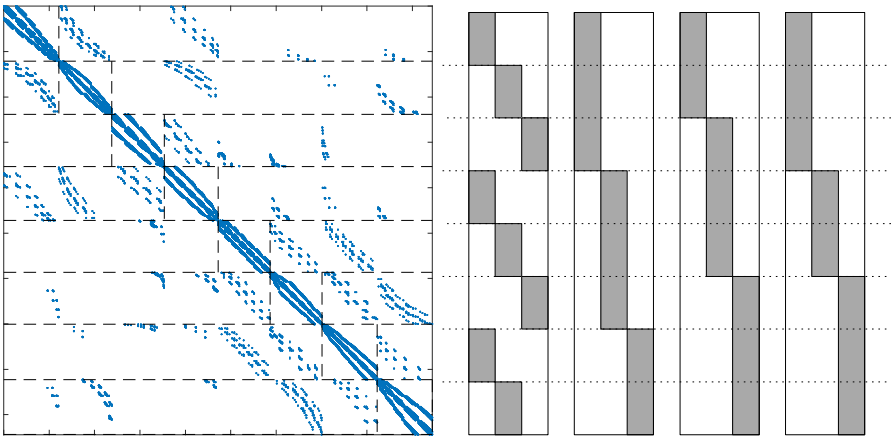


FIG. 1. *Illustration of the ordering of A into eight subdomains obtained with METIS* [25] *and several admissible splittings of* r_0 *into three vectors.*

field [24, 33, 26]. They appear to be well suited for modern computers' architectures with a high level of parallelism because they allow one to reduce the number of global synchronizations, while also featuring a higher arithmetic intensity at the cost of some extra computations.

2.2. Enlarged Krylov subspaces. In [14], the authors define so-called enlarged Krylov subspaces. First, the matrix A is reordered by partitioning its graph into $\mathcal{N}$ subdomains (using METIS [25], for example). Then the initial residual r_0 is split into t vectors denoted by $R_0^{e(i)}$, $1 \leq i \leq t$. In the original paper, the authors use $t = \mathcal{N}$. It is important to note that the case $t < \mathcal{N}$ can be dealt with many ways as long as $r_0 = \sum_{i=1}^{t} R_0^{e(i)}$ (Figure 1). This is of particular interest in practice because typically $\mathcal{N}$ will correspond to the number of MPI processes. The parameter t is called the enlarging factor. In practice, for a given t, we have performed numerical experiments which show that the splitting of r_0 does not have any significant impact on the convergence of the method. The intuition behind this observation is that all the admissible splittings that we considered are *equivalent* up to a *renumbering of the domains*. In the numerical experiments, we construct the initial enlarged residual $R_0^e = [R_0^{e(1)}, \ldots, R_0^{e(t)}]$ as the leftmost example in Figure 1.

Then the enlarged Krylov subspace of order k denoted by $\mathcal{K}_{k,t}(A, r_0)$ is defined as the block Krylov subspace of order k associated to A and the enlarged residual R_0^e. More precisely, and following the notation introduced in [17],

$$\mathcal{K}_{k,t}(A, r_0) = \mathcal{K}_k^{\square}(A, R_0^e) \tag{2.1}$$

$$= \text{span}^{\square}\left\{R_0^e, AR_0^e, \ldots, A^{k-1}R_0^e\right\}. \tag{2.2}$$

Using this definition and following [16], it is possible to derive two variants (Orthomin and Orthodir) of the enlarged conjugate gradient (ECG) algorithm (Algorithm 2.1). More precisely, the enlarged approximate solution is a matrix of size $n \times t$ denoted by X_k, and the sum of its columns gives the approximate solution of the original system. We denote by R_k the enlarged approximate residual, and similarly we obtain the approximate residual of the original system by summing its columns. P_k is a matrix of size $n \times t$ called search directions; it corresponds to the A-orthonormalization

of another matrix of size $n \times t$ that we denote by Z_k. In fact, the algorithms first construct Z_k and then P_k, which is used for updating both the approximated solution and the residual. We denote by α_k the optimal step; unlike in the CG algorithm, it is not a scalar but a matrix of size $t \times t$. Depending on the method for constructing Z_{k+1}, it is possible to derive two variants of ECG: Orthomin and Orthodir.

Orthomin (Omin) corresponds to block CG [29]:

$$\beta_k = (AP_k)^\top R_k, \tag{2.3}$$

$$Z_{k+1} = R_k - P_k\beta_k. \tag{2.4}$$

This method is very similar to the one originally proposed by Hestenes and Stiefel [19] because it constructs the new descent directions Z_{k+1} using R_k and P_k.

Orthodir (Odir) corresponds to the block Lanczos algorithm but with the inner product induced by A:

$$\gamma_k = (AP_k)^\top (AP_k), \tag{2.5}$$

$$\rho_k = (AP_{k-1})^\top (AP_k), \tag{2.6}$$

$$Z_{k+1} = AP_k - P_k\gamma_k - P_{k-1}\rho_k. \tag{2.7}$$

It is the block equivalent of the homonym method defined in [1]. Unlike the previous variant, Z_{k+1} is constructed using P_k and P_{k-1}.

Both Orthodir and Orthomin produce Z_{k+1}, which is A-orthogonal to P_i for $i \leq k$. Then the search directions P_{k+1} are defined as

$$P_{k+1} = Z_{k+1}(Z_{k+1}^\top AZ_{k+1})^{-1/2}. \tag{2.8}$$

Unlike the CG algorithm, a breakdown occurs if $Z_{k+1}^\top AZ_{k+1}$ is singular, i.e., Z_{k+1} is not full rank. Although rare, this situation can happen in practice, and several variants have been developed in order to handle this case [21, 16, 11, 29]. Overall, both Orthomin and Orthodir generate P_{k+1} such that

$$P_{k+1}^\top AP_i = 0 \quad \forall i \leq k, \tag{2.9}$$

$$P_{k+1}^\top AP_{k+1} = I. \tag{2.10}$$

Consequently, the ECG method can be summarized in Algorithm 2.1. Another difference with the original block CG algorithm is that the search directions are A-orthonormalized at each iteration: P_k is used as search directions instead of Z_k. It has been shown numerically that using this variant can increase the numerical stability of the method [11].

Given a preconditioner M^{-1}, the idea for applying left preconditioning to the (block) CG method is to remark that $M^{-1}A$ is self-adjoint with respect to the M-inner product [32]. Then, by replacing the occurrences of A by $M^{-1}A$ and the occurrences of the transpose sign $^\top$ by $^\top M$ in the algorithm (Algorithm 2.1), it follows the preconditioned ECG method. In fact, some simplifications occur and the algorithm remains exactly the same, except the definition of Z_k, which is slightly different. More precisely, it follows that the preconditioned Orthomin method corresponds to

$$\beta_k = (AP_k)^\top M^{-1}R_k, \tag{2.11}$$

$$Z_{k+1} = M^{-1}R_k - P_k\beta_k \tag{2.12}$$

and the preconditioned Orthodir method corresponds to

$$\gamma_k = (AP_k)^\top (M^{-1}AP_k), \tag{2.13}$$

$$\rho_k = (AP_{k-1})^\top (M^{-1}AP_k), \tag{2.14}$$

$$Z_{k+1} = M^{-1}AP_k - P_k\gamma_k - P_{k-1}\rho_k. \tag{2.15}$$

In both cases, the initialization also slightly differs because $Z_1 = M^{-1}R_0^e$. Overall, the preconditioner is applied once per iteration, as in the standard CG method.

Algorithm 2.1. Preconditioned ECG algorithm.

$P_0 = 0$, $R_0 = R_0^e$, $Z_1 = M^{-1}R_0$
for $k = 1, \ldots, k_{\max}$ **do**
  $P_k = Z_k(Z_k^\top AZ_k)^{-1/2}$
  $\alpha_k = P_k^\top R_{k-1}$
  $X_k = X_{k-1} + P_k\alpha_k$
  $R_k = R_{k-1} - AP_k\alpha_k$
  if $||\sum_{i=1}^t R_k^{(i)}||_2 < \varepsilon$ **then**
    stop
  end if
  construct Z_{k+1} using (2.11)–(2.12) (Orthomin) or (2.13)–(2.15) (Orthodir)
end for
$x_k = \sum_{i=1}^t X_k^{(i)}$

2.3. Equivalence between Orthodir and Orthomin. In what follows, we assume exact arithmetic and we study the connection between these two methods with the aim of deriving formulas that link the approximate quantities of both variants. Indeed, by construction the approximate solutions computed by Orthodir and Orthomin are equal. Hence the approximate residuals are also equal. But this does not imply that the search directions generated are equal even if they belong to the same space. We denote with a tilde the variables related to Orthomin and with a hat the variables related to Orthodir; e.g., $\widehat{P}_k$ are the A-orthonormalized search directions generated during Orthodir.

For the sake of brevity, we only consider the case where no breakdowns occur so that $\widehat{Z}_k$, $\widehat{P}_k$ and $\widetilde{Z}_k$, $\widetilde{P}_k$ are all well-defined.

Since Orthomin and Orthodir rely on the same projection process [16] (they both search an approximate solution in $\mathcal{K}_{t,k}$, such that the corresponding residual is orthogonal to $\mathcal{K}_{t,k}$), we know that $\widehat{X}_k = \widetilde{X}_k$. It follows that

$$\widehat{P}_k\widehat{\alpha}_k = \widetilde{P}_k\widetilde{\alpha}_k, \tag{2.16}$$

$$\widehat{P}_k\widehat{P}_k^\top = \widetilde{P}_k\widetilde{P}_k^\top. \tag{2.17}$$

Hence there exists $\delta_k \in \mathbb{R}^{t\times t}$ orthogonal and such that $\widetilde{P}_k = \widehat{P}_k\delta_k$.

A simple computation using the previous relationships gives

$$\widehat{Z}_{k+1}\delta_k\widetilde{\alpha}_k = A\widetilde{P}_k\widetilde{\alpha}_k - \widehat{P}_k\widehat{P}_k^\top AA\widetilde{P}_k\widetilde{\alpha}_k - \widehat{P}_{k-1}\widehat{P}_{k-1}^\top AA\widetilde{P}_k\widetilde{\alpha}_k, \tag{2.18}$$

$$= A\widetilde{P}_k\widetilde{\alpha}_k - \widetilde{P}_k\widetilde{P}_k^\top AA\widetilde{P}_k\widetilde{\alpha}_k - \widetilde{P}_{k-1}\widetilde{P}_{k-1}^\top AA\widetilde{P}_k\widetilde{\alpha}_k. \tag{2.19}$$

On the other hand, by the definition of ECG we have

$$R_k - R_{k-1} = -A\widetilde{P}_k\widetilde{\alpha}_k \tag{2.20}$$

and

$$\widetilde{P}_{k-1}^{\top} A R_k = 0. \tag{2.21}$$

Hence it follows that

$$-\widehat{Z}_{k+1}\delta_k\widetilde{\alpha}_k = R_k - \widetilde{P}_k\widetilde{P}_k^{\top} A R_k - R_{k-1} + \widetilde{P}_{k-1}\widetilde{P}_{k-1}^{\top} A R_{k-1} + \widetilde{P}_k\widetilde{P}_k^{\top} A R_{k-1} \tag{2.22}$$

$$= R_k - \widetilde{P}_k\widetilde{\beta}_k - \widetilde{Z}_k + \widetilde{P}_k\widetilde{P}_k^{\top} A R_{k-1} \tag{2.23}$$

and, furthermore,

$$\widetilde{P}_k\widetilde{P}_k^{\top} A R_{k-1} = \widetilde{Z}_k(\widetilde{Z}_k^{\top} A\widetilde{Z}_k)^{-1}\widetilde{Z}_k^{\top} A R_{k-1} \tag{2.24}$$

$$= \widetilde{Z}_k(\widetilde{Z}_k^{\top} A\widetilde{Z}_k)^{-1}\widetilde{Z}_k^{\top} A\widetilde{Z}_k \tag{2.25}$$

$$= \widetilde{Z}_k. \tag{2.26}$$

Indeed, a direct computation gives

$$\widetilde{Z}_k^{\top} A\widetilde{Z}_k = \widetilde{Z}_k^{\top} A(R_{k-1} - \widetilde{P}_{k-1}\widetilde{P}_{k-1}^{\top} A R_{k-1}) \tag{2.27}$$

$$= \widetilde{Z}_k^{\top} A R_{k-1} \tag{2.28}$$

because by construction $\widetilde{Z}_k^{\top} A\widetilde{P}_{k-1} = 0$. Thus, we have

$$\widetilde{Z}_{k+1} = -\widehat{Z}_{k+1}\delta_k\widetilde{\alpha}_k. \tag{2.29}$$

This result is a generalization of a previous result presented by Ashby, Manteuffel, and Saylor [1, p. 1550] for standard CG. In fact, the authors show that $\widetilde{z}_k = \Pi_{i=0}^{k}(-\widetilde{\alpha}_k)\widehat{z}_k$, but they never consider explicitly the A-orthonormalized search directions. In particular, they define z_{k+1} using z_k (for Omin) and z_{k-1} (for Odir). This explains the slight difference between our generalization and their result.

When k becomes large, $\widetilde{\alpha}_k = \widetilde{P}_k^{\top} R_{k-1}$ and $||\widetilde{\alpha}_k||_2$ is more likely to be small because R_{k-1} is supposed to converge to 0 and $\widetilde{P}_k$ is A-orthonormalized—the same reasoning applies for $\widehat{\alpha}_k$. This result is very interesting because it shows that, since δ_k is an orthogonal matrix, when k becomes large, $||\widetilde{Z}_{k+1}||_2$ can be significantly smaller than $||\widehat{Z}_{k+1}||_2$. Hence the conditioning of $\widetilde{Z}_{k+1}^{\top} A\widetilde{Z}_{k+1}$ could be much worse than that of $\widehat{Z}_{k+1}^{\top} A\widehat{Z}_{k+1}$, possibly leading to a breakdown when computing its Cholesky factorization (line 5 in Algorithm 2.1). It is remarkable to notice that even for the standard CG method, this has already been noticed by Ashby, Manteuffel, and Saylor in [1, pp. 1551–1552]: "If BCA[1] is indefinite, Omin may still be used, but the previous direction vector, $\hat{p}_{i-1}$, should be stored. Then, if $\widehat{\alpha}_i = 0$ (or is nearly zero), control can switch to the 3-term recursion of Odir to get p_{i+1}." In practice, this phenomenon is indeed observed: there are cases where Orthomin breaks down while Orthodir does not [16]. In conclusion, Orthodir is expected to be more reliable than Orthomin. However, Orthodir is also more costly than Orthomin: the construction of Z_{k+1} requires twice as many flops and memory as for Orthomin.

[1] B denotes the SPD matrix that represents the scalar product, i.e., A. C denotes the preconditioner, i.e., M^{-1}. And A denotes the SPD matrix associated to the system.

2.4. Convergence study. As previously mentioned, O'Leary [29] proved that block CG can converge significantly faster than standard CG. In [14], it is proved that ECG converges at least as fast as CG, but there is no further information on the speed of convergence of ECG.

In what follows, we derive a much sharper bound for ECG's convergence. It is, however, very likely that this new bound is still too pessimistic compared to the real convergence of the method. In particular, it is well known that the corresponding bound for the CG method derived using a Chebyshev polynomial does not take into account the nonlinear behavior of the convergence of the method [28]. In fact, the results that exist for describing the error of the CG method completely are not straightforward to generalize to the block CG method and hence to the ECG method. Furthermore, in order to simplify the analysis we neglect round-off errors, and it is also well known that they can significantly impact the convergence of the CG method [28]. However, the theoretical study of the behavior of the ECG method in finite arithmetic is out of the scope of the present paper. Our aim is to make a first step in understanding the convergence behavior of the ECG method, but we are aware that a lot remains to be done.

This section is dedicated to the proof of the following result.

THEOREM 2.1. *Let x_k be the approximate solution given by the ECG method in exact precision with an enlarging factor t at step $k \geq 1$. Then we have*

$$||x_k - x_*||_A \leq ||\hat{e}_0||_A \min_{p \in \mathbb{P}_1^k} \max_{i \in \{t,\dots,n\}} |p(\lambda_i)|, \tag{2.30}$$

where $\mathbb{P}_1^k$ denotes the set of polynomials of degree at most k and such that $p(0) \equiv 1$, and $\hat{e}_0$ is a constant independent of k defined as

$$\hat{e}_0 \equiv E_0(\Phi_1^\top E_0)^{-1} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}, \tag{2.31}$$

where E_0 denotes the initial enlarged error and $\Phi_1 = \begin{pmatrix} \phi_1 & \dots & \phi_t \end{pmatrix}$ denotes the matrix whose columns are the t eigenvectors associated to the smallest eigenvalues of A.

Proof. In [14], it is shown that the approximate solution of ECG at iteration $k \geq 1$, denoted by x_k, verifies

$$||x_* - x_k||_A = \min_{y \in \mathcal{K}_{k,t}(A, r_0)} ||x_* - y||_A. \tag{2.32}$$

Our strategy is to find a $y \in \mathcal{K}_{k,t}(A, r_0)$ that satisfies

$$||x_* - y||_A \leq C \min_{p \in \mathbb{P}_1^k} \max_{i \in \{t,\dots,n\}} |p(\lambda_i)|, \tag{2.33}$$

where C is a constant which does not depend on k. For any element z of $\mathcal{K}_{k,t}(A, r_0)$, the error $x_* - z$ reads as

$$x_* - z = \sum_{j=1}^{t} p_{kj}(A) E_0^{(j)}, \tag{2.34}$$

where p_{kj} is a polynomial of degree not exceeding k, and $E_0^{(j)}$ denotes the jth column of the initial enlarged error, denoted by E_0.

Let $\Phi\Lambda\Phi^\top$ be the spectral decomposition of A, i.e, $A = \Phi\Lambda\Phi^\top$, where the eigenvalues are sorted in an increasing order in the diagonal matrix Λ, and the columns of Φ are the corresponding eigenvectors such that Φ is orthonormal. We further denote $\Lambda_1 = \text{diag}(\lambda_1, \ldots, \lambda_t)$, $\Lambda_2 = \text{diag}(\lambda_{t+1}, \ldots, \lambda_n)$ and $\Phi_1 = \begin{pmatrix}\phi_1 & \ldots & \phi_t\end{pmatrix}$, $\Phi_2 = \begin{pmatrix}\phi_{t+1} & \ldots & \phi_n\end{pmatrix}$.

Let

$$\hat{e}_0 \equiv E_0(\Phi_1^\top E_0)^{-1} \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}. \tag{2.35}$$

It directly follows that

$$\hat{e}_0 = \Phi\Phi^\top \hat{e}_0 = \phi_t + \Phi_2\Phi_2^\top \hat{e}_0. \tag{2.36}$$

Indeed, by construction $\hat{e}_0$ is the vector in $\text{Range}(E_0) = \text{span}\{E_0^{(1)}, \ldots, E_0^{(t)}\}$ such that its orthogonal projection onto $\text{Range}(\Phi_1)$ is exactly equal to ϕ_t [31, Lemma 4]. In particular, there exist t real numbers $\zeta_1, \ldots, \zeta_t$ such that $\hat{e}_0 = \sum_{j=1}^t \zeta_j E_0^{(j)}$. Thus, if we choose $p_{kj} \equiv \zeta_j p_k$ in (2.34), where $p_k \in \mathbb{P}_1^k$, we can define $y \in \mathcal{K}_{k,t}(A, r_0)$ such that

$$x_* - y \equiv p_k(A)\hat{e}_0. \tag{2.37}$$

Given our choice of $\hat{e}_0$, we have

$$x_* - y = \Phi p_k(\Lambda)\Phi^\top \hat{e}_0 \tag{2.38}$$

$$= \Phi p_k(\Lambda)\Phi^\top(\phi_t + \Phi_2\Phi_2^\top \hat{e}_0) \tag{2.39}$$

$$= \phi_t p_k(\lambda_t) + \Phi_2 p_k(\Lambda_2)\Phi_2^\top \hat{e}_0. \tag{2.40}$$

Thus, we finally obtain the desired bound

$$||x_* - y||_A \leq ||\hat{e}_0||_A \min_{p \in \mathbb{P}_1^k} \max_{i \in \{t, \ldots, n\}} |p(\lambda_i)|. \tag{2.41}$$

∎

It is possible to use Chebyshev polynomials [29, 32] to bound the min-max quantity

$$\min_{p \in \mathbb{P}_1^k} \max_{t \leq i \leq n} |p(\lambda_i)| \leq 2\left(\frac{\sqrt{\kappa_t} - 1}{\sqrt{\kappa_t} + 1}\right)^k, \tag{2.42}$$

where $\kappa_t = \frac{\lambda_n}{\lambda_t}$.

COROLLARY 2.2. *Let x_k be the approximate solution given by the ECG method with an enlarging factor t at step $k \geq 1$. Then we have*

$$||x_k - x_*||_A \leq 2||\hat{e}_0||_A \left(\frac{\sqrt{\kappa_t} - 1}{\sqrt{\kappa_t} + 1}\right)^k, \tag{2.43}$$

where $\hat{e}_0$ is defined by (2.31)*, and $\kappa_t = \frac{\lambda_n}{\lambda_t}$.*

This final result is similar in its form to the following well-known result for the CG method: a constant independent of k, $2||\hat{e}_0||_A$ multiplied by a geometric factor, $\left((\sqrt{\kappa_t}-1)/(\sqrt{\kappa_t}+1)\right)^k$. The main difference is that the geometric factor can be much smaller for ECG, especially if $\lambda_1 \ll \lambda_t$. In fact, this geometric factor looks similar to that of the *deflated CG* method [10]. This method can be seen as applying the CG method on the so-called deflated operator. This deflated operator is the same as the original one, except that some of its (smallest) eigenvalues are shifted to be equal to 1.[2] Thus, it allows one to remove the possibly bad effect of these eigenvalues on the convergence. However the main drawback of the deflated CG method is that it requires the knowledge of these eigenvalues and their corresponding eigenvectors. Furthermore, assuming that t vectors are deflated, Dostál [10] proved the following:

$$\kappa(\tilde{A}) \leq \frac{\lambda_n}{\sqrt{(1-\gamma^2)\lambda_{t+1}^2 + \gamma^2\lambda_1^2}}, \tag{2.44}$$

where $\kappa(\tilde{A})$ denotes the condition number of the deflated operator, and γ represents the distance between the exact eigenspace associated to the t smallest eigenvalues and the space spanned by the deflated vectors—it can be assumed to be strictly smaller than 1; see [10]. If we plug this condition number in the well-known result of the convergence of the CG method, the geometric factor depends on γ, which in turn depends on the distance between the exact eigenspace associated to the t smallest eigenvalues and the space spanned by the deflated vectors [35]. The results about the convergence of the deflated CG method and that of the ECG method are different, making their comparison difficult. However, the geometric factor of ECG is independent of any error with respect to the exact eigenvectors, but the constant is higher. On the other hand, if a low accuracy is required, it is likely that the deflated CG method delivers a proper approximation in fewer iterations, in particular if the deflated vectors approximate well the eigenvectors associated to the smallest eigenvalues. Furthermore, we want to point out that the proof can be easily adapted to show that one could select in Φ_1 any t eigenvectors associated to eigenvalues at the end of the spectrum of A, i.e., a mixture of t smallest and largest eigenvalues. Following [29, p. 312], it is possible to show an estimate similar to (2.43) using well-chosen Chebyshev polynomials, resulting in a lower bound depending on the spectrum of A.

We now briefly comment on our strategy for proving the result, and we review similar existing results. The strategy of the proof of Theorem 2.1 mimics that used by Saad in [31], but in the context of the Lanczos and block Lanczos methods. In particular, the choice of $\hat{e}_0$ is driven by [31, Lemma 4]. This is of course due to the close connection between the ECG method, the block CG method, and the block Lanczos method. For instance, O'Leary proved a similar result for the block CG method [29, Theorem 5]. More precisely, the geometric factors are the same. However, unlike that of O'Leary, our proof does not rely on Chebyshev polynomials, and it is also more simple. Hence the constant in front of the geometric factor has a simpler expression in our case.

2.5. Dynamic reduction of the search directions. In what follows, we recall an approach for reducing the block size in the Orthodir method during the iterations

[2] There exist a lot of different algorithmic variants of the method, e.g., to shift the eigenvalues to 0, but they are all theoretically equivalent [35].

presented in [16]. The idea is to reduce the added arithmetic and memory costs of Orthodir over Orthomin, while maintaining its good convergence behavior. As explained in the survey [17], the key idea to reduce the block size is to monitor the rank of R_k. Once R_k becomes rank deficient, it means that a part of the approximate solution has already converged at iteration k. More precisely, for $i \geq k-1$ there exists a linear combination (independent of i) of columns of X_i that remains constant. As a consequence, there exists a linear combination of search directions that is no longer useful for computing the approximate solution. The idea is to remove these search directions in the next iterations. As R_{k-1} is an $n \times t$ matrix with n large, it is preferable to avoid computing the rank of R_{k-1} directly. In [16], it is shown that the rank of $\alpha_k = P_k^\top R_{k-1}$ can be computed instead.

The method presented in [16] can be divided into two parts. At each iteration of the algorithm (Algorithm 2.2), a singular value decomposition (SVD) of α_k is computed (line 5). If the numerical rank of α_k is below a given tolerance, $\varepsilon_{\text{def}} \equiv \varepsilon/\sqrt{t}$ (following [16]), then the search directions are reduced accordingly (line 12) and some of them are kept in order to keep the A-orthogonality property (lines 11, 21, and 22). Although computing the SVD of α_k at each iteration induces an extra cost compared to Orthodir, this operation does not involve any communications and it is negligible because α_k is a small matrix of size $t \times t$. Furthermore, as the search directions P_k are reduced, the dominant operation of Krylov iterations in terms of flops, the matrix product (AP_k), and the application of the preconditioner ($M^{-1}AP_k$) become cheaper.

Algorithm 2.2. ECG D-Odir algorithm.

1: $P_0 = 0$, $R_0 = Z_1 = R_0^e$, $H = []$
2: **for** $k = 1, \ldots, k_{\max}$ **do**
3: $\quad P_k = Z_k(Z_k^\top A Z_k)^{-1/2}$
4: $\quad \alpha_k = P_k^\top R_{k-1}$
5: $\quad \alpha_k = U_k \Sigma_k V_k^\top$
6: $\quad$ let s_k be the number of singular values of α_k bigger than $\varepsilon_{\text{def}} = \frac{\varepsilon}{\sqrt{t}}$ (see [16])
7: $\quad$ **if** $s_k < s_{k-1}$ **then**
8: $\quad\quad \alpha_k = U_k^\top \alpha_k$
9: $\quad\quad P_k = P_k U_k$
10: $\quad\quad \alpha_k = \alpha_k(1 : s_k, :)$
11: $\quad\quad H = [H, P(:, s_k : s_{k-1})]$
12: $\quad\quad P_k = P_k(:, 1 : s_k)$
13: $\quad$ **end if**
14: $\quad X_k = X_{k-1} + P_k \alpha_k$
15: $\quad R_k = R_{k-1} - AP_k \alpha_k$
16: $\quad$ **if** $||\sum_{i=1}^{t} R_k^{(i)}||_2 < \varepsilon ||r_0||$ **then**
17: $\quad\quad$ stop
18: $\quad$ **end if**
19: $\quad \gamma_k = (AP_k)^\top (AP_k)$
20: $\quad \rho_k = (AP_{k-1})^\top (AP_k)$
21: $\quad \delta_k = (AH)^\top (AP_k)$
22: $\quad Z_{k+1} = AP_k - P_k \gamma_k - P_{k-1} \rho_k - H \delta_k$
23: **end for**
24: $x_k = \sum_{i=1}^{t} X_k^{(i)}$

2.6. Curing breakdowns in Orthomin. As explained previously, the Orthomin version of ECG can break down. There exist several methods to overcome this issue, and in the following we recall the *breakdown-free block CG* method defined in [21]. Starting from the original algorithm of O'Leary [29], the authors propose to perform a rank-revealing QR decomposition of Z_{k+1} and then drop its null part before A-orthonormalizing it. They show that in exact arithmetic this allows one to continue the algorithm with nearly no further modification. The resulting algorithm is given in Algorithm 2.3.

From a practical point of view, the size of P_{k+1} can be reduced, but at each iteration Z_{k+1} is of size $n \times t$ because the size of R_k remains constant. Hence the matrix product AP_k is cheaper, but the application of the preconditioner $M^{-1}R_k$ is not. Furthermore, computing a rank-revealing QR factorization of Z_{k+1} cannot be neglected because Z_{k+1} is of size $n \times t$. One should keep in mind that the purpose of this method is to improve the stability of Orthomin. Thus, it is likely that it requires more flops than the dynamic variant of Orthodir.

Algorithm 2.3. BF-ECG algorithm.

$P_0 = 0$, $R_0 = Z_1 = R_0^e$
for $k = 1, \ldots, k_{\max}$ **do**
  $P_k = Z_k(Z_k^\top A Z_k)^{-1/2}$
  $\alpha_k = P_k^\top R_{k-1}$
  $X_k = X_{k-1} + P_k \alpha_k$
  $R_k = R_{k-1} - AP_k \alpha_k$
  if $|| \sum_{i=1}^{t} R_k^{(i)}||_2 < \varepsilon ||r_0||$ **then**
    stop
  end if
  $\beta_k = (AP_k)^\top R_k$
  $Z_{k+1} = R_k - P_k \beta_k$
  $Z_{k+1} = \mathrm{RRQR}(Z_{k+1}, \sqrt{\varepsilon_{\text{machine}}})$ (using Algorithm 3.1)
end for
$x_k = \sum_{i=1}^{t} X_k^{(i)}$

3. Parallel design.

3.1. Data distribution. As is usually the case in parallel implementations of Krylov methods, we assume that the unknowns are distributed among the processors. We also assume that each processor owns different unknowns. Thus, all the variables whose size scales as the size of the linear system (X_k, R_k, P_k, AP_k, Z_k) are distributed rowwise among the processors according to the distribution of the unknowns. All variables whose size scales as the enlarging factor t (α_k, β_k, γ_k, ρ_k) are replicated on all the processors. Locally, they are stored contiguously and column by column (see Figure 2). There is no allocation or deallocation of memory during the iterations. In particular, when using dynamic Orthodir or breakdown-free Orthomin the memory is not freed when the block size is reduced. The local memory consumption of preconditioned Orthodir and Orthomin on N_{proc} processors is summarized in Table 1. For completeness, we also add the local memory consumption of the standard CG algorithm, described in [32], for instance, where only five vectors and two scalars are needed.

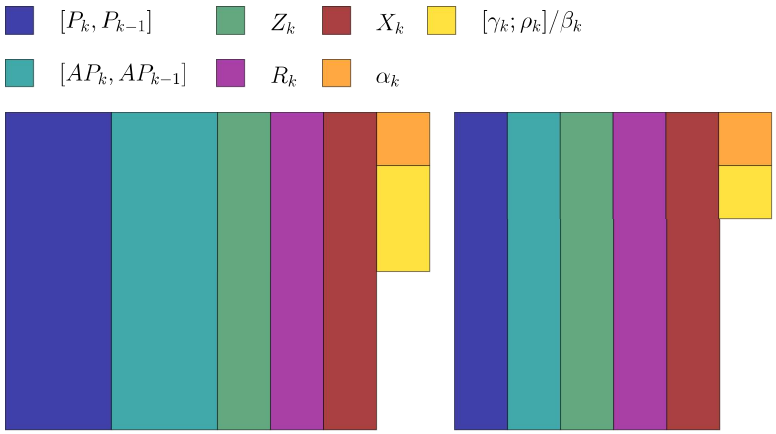


FIG. 2. *Local distribution of the data: Orthodir on the left and Orthomin on the right.*

TABLE 1

Complexity and memory consumption of Orthodir, Orthomin, and CG where t is the enlarging factor, n is the number of rows of A, and N_{proc} is the number of processors. Parentheses indicate the number of calls to `MPI_Allreduce`.

	# flops	# messages	# words	Memory
Omin	$16\frac{nt^2}{N_{\text{proc}}} + 4\frac{nt}{N_{\text{proc}}} + \frac{1}{3}t^3$	$4\log_2(N_{\text{proc}})$ (4)	$4t^2$	$5\frac{nt}{N_{\text{proc}}} + 2t^2$
Odir	$20\frac{nt^2}{N_{\text{proc}}} + 5\frac{nt}{N_{\text{proc}}} + \frac{1}{3}t^3$	$4\log_2(N_{\text{proc}})$ (4)	$5t^2$	$7\frac{nt}{N_{\text{proc}}} + 3t^2$
CG	$10\frac{n}{N_{\text{proc}}}$	$2\log_2(N_{\text{proc}})$ (2)	2	$5\frac{n}{N_{\text{proc}}}$

3.2. Cost analysis of ECG. Our implementation of ECG is based on the reverse communication interface [9]. For one iteration of ECG, it requires external routines to apply the sparse matrix product and the preconditioner to a set of vectors. Indeed, the implementation of these routines highly depends on the linear system to be solved. This is why we do not take into account these operations in our cost analysis.

Given n, t such that $t \ll n$, we denote by V, W tall and skinny matrices of size $n \times t$ whose rows are distributed among the processors, and α is a matrix of size $t \times t$ replicated on the N_{proc} processors. Following [26], it is possible to decompose the iterations of ECG (and, more generally, block CG) into the following kernels:

- $V \leftarrow V + W\alpha$ (`tsmm` in [26]),
- $\alpha \leftarrow V^\top W$ (`tsmtsm` in [26]),
- Cholesky factorization of α (`potrf`),
- triangular solve of α with several right-hand sides (`trsm`).

Following the preconditioned ECG algorithm (Algorithm 2.1), each iteration of Orthodir and Orthomin consists of 3 `tsmm` (lines 7, 8, and 12), 4 `tsmtsm` (lines 5, 6, 9, and 12), 1 `potrf` (line 5), and 2 `trsm` (line 5). Indeed, line 5 of the algorithm (Algorithm 2.1) can be decomposed as

$AZ_k \leftarrow A * Z_k$	*sparse matrix set of vectors*
$C \leftarrow$ `tsmtsm`(Z_k, AZ_k)	*form* $Z_k^\top AZ_k$
$C \leftarrow$ `potrf`(C)	*Cholesky factorization*
$P_k \leftarrow$ `trsm`(Z_k, C)	
$AP_k \leftarrow$ `trsm`(AZ_k, C)	*update* P_k *and* AP_k

Doing so allows us to avoid calling the sparse matrix set of vectors product for computing AP_k at the price of an extra `trsm`. Hence the difference between the two algorithms is the construction of Z_{k+1} (line 12). The `tsmtsm` and `tsmm` for constructing Z_{k+1} in Orthodir (see (2.5)–(2.7)) cost twice as much as for Orthomin (see (2.3)–(2.4)).

As matrices of size $t \times t$ are replicated among the processors, we notice that `tsmm`, the Cholesky factorization of α, and the triangular solve of α are local operations without any communication. Hence we use the corresponding LAPACK routines: `gemm`, `potrf` (dense Cholesky factorization), and `trsm` (dense triangular solve with several right-hand sides). However, V and W are distributed and `tsmtsm` is not a local operation. The LAPACK routine `gemm` is called to compute the local product $V_i^\top W_i$ followed by a call to `MPI_Allreduce`.

Thus, the only kernel operation that requires a communication is `tsmtsm`, and four calls to `MPI_Allreduce` are done per iteration. It is usually assumed that during a call to `MPI_Allreduce` the number of messages sent and received on the network is equal to $\log_2(N_{\text{proc}})$—although the exact number depends on the MPI implementation [36]. Moreover, it is a blocking operation: when completed, all the processors are synchronized. This is why in practice, as in standard CG, the communication cost is dominated by two calls to `MPI_Allreduce`: the one after the sparse matrix set of vectors product (line 5) and the one after the preconditioner (line 12) because they occur after operations with a potential load imbalance between processors.

In summary, the detailed costs of one iteration of Orthodir and Orthomin in terms of flops, words, and messages are indicated in Table 1. For the sake of comparison, we recall the complexity of the CG algorithm described in [32]. We also report the number of `MPI_Allreduce` in parentheses, in addition to the order of magnitude of the number of messages. In summary, one iteration of ECG is approximately t^2 times more costly in terms of flops than one iteration of CG. While the number of messages is of the same order, the number of words is also t^2 times larger. Indeed, there is a trade-off between these extra costs and the reduction of the number of iterations due to using enlarged Krylov subspaces as search spaces.

3.3. Cost of dynamic reduction of ECG. The implementation of the dynamic reduction of the search directions within Orthodir follows Algorithm 2.2. In practice, we use LAPACK routine `gesvd` and only compute the left singular vectors of α_k, denoted by U_k. We check the singular values obtained. If there are some smaller than $\frac{\varepsilon}{\sqrt{t}}$, which is the criterion proposed in [16], we call `geqrf` on U in order to perform the updates PU_k, APU_k, and $U_k^\top \alpha_k$ in-place with `ormqr`. Since P_k and AP_k are stored in a column major fashion, the selection of the columns is done at no cost. Similarly, H is not explicitly defined. However, the selection of the first rows of α_k implies an in-place memory rearrangement.

The implementation of breakdown-free Orthomin is similar to Orthomin, except the computation of a rank-revealing QR decomposition of Z_{k+1}. As Z_{k+1} is distributed, it is not reasonable to use a LAPACK kernel to compute it. Instead, we use a modification of the Chol-QR algorithm [38] (Algorithm 3.1) which is a cheaper but

less stable alternative to TS-RRQR [8, 6]. Its implementation is very easy using the LAPACK routine `pstrf` (Cholesky with pivoting) for computing (R, π) at line 2 of Algorithm 3.1. We use the default tolerance of `pstrf` for detecting the rank deficiency of Z_{k+1}; this is why the overall RRQR in Algorithm 2.3 (line 12) is computed up to $\sqrt{\varepsilon_{\text{machine}}}$. This of course might cause numerical issues, e.g., if the stopping criterion is set to a value smaller than $\sqrt{\varepsilon_{\text{machine}}}$.

Algorithm 3.1. Chol-RRQR.

Input: P, ε

Output: Q_1 orthogonal such that $P\pi = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$, where π is a permutation and all the diagonal elements of R_{11} are larger than ε

1: $\mu \leftarrow P^\top P$

2: Compute (R, π) such that $\pi^\top \mu \pi = R^\top R$ with $R = \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$ and all the diagonal elements of R_{11} are larger than ε^2

3: $P_1 \leftarrow P\pi(:, 1 : size(R_{11}))$

4: $Q_1 \leftarrow P_1 R_{11}^{-1}$

4. Numerical experiments.

4.1. Description of the parallel environment. In the experiments, we use a block Jacobi preconditioner, associating at each block an MPI process. Before calling ECG, each MPI process factorizes the diagonal block of A corresponding to the local row panel that it owns. The initial enlarged residual $R_0^e = \begin{pmatrix} R_0^{e(1)} & \dots & R_0^{e(t)} \end{pmatrix}$ is constructed as the leftmost example in Figure 1. At each iteration of ECG, each MPI process performs a backward and forward solve locally in order to apply the preconditioner. Hence the application of the block Jacobi preconditioner does not need any communication. It is likely that there exist better preconditioners than block Jacobi for our test cases; however, we are interested in the iterative method rather than in the preconditioner. In particular, we do not want to target specific applications and aim at being as generic as possible. Although in theory it is possible to apply any preconditioner within this implementation, in practice it is essential that applying this preconditioner to several vectors at the same time is not too costly, e.g, a sublinear complexity with respect to the number of vectors.

The following experiments are performed on a machine located at Umeå University as part of High Performance Computing Center North (HPC2N), called Kebnekaise. It is a heterogeneous machine formed by a mix of Intel Xeon E5-2690v4 (Broadwell) with 2×14 cores (and E7-8860v4 for large memory computations), Nvidia K80 GPU, and Intel Xeon Phi 7250 (Knight's Landing) with 68 cores. In our experiments, we use the so-called compute nodes, which are formed by Intel Xeon E5-2690v4 (Broadwell) with 2×14 cores. For a detailed description of the machine, we refer the reader to the online documentation.[3]

We compile the code (and its dependencies) using Intel toolchain installed on the machine: `mpiicc` (based on `icc` version 18.0.1 20171018) and MKL [37] version 2018.1.163. We use PETSc [2] in order to compare ECG implementation to PETSc PCG implementation. In particular, PETSc is configured to use MKL-PARDISO as the exact solver for sparse matrices in the block Jacobi preconditioner. For partitioning the matrix, we are using the METIS library downloaded and installed by PETSc.

[3] https://www.hpc2n.umu.se/resources/hardware/kebnekaise

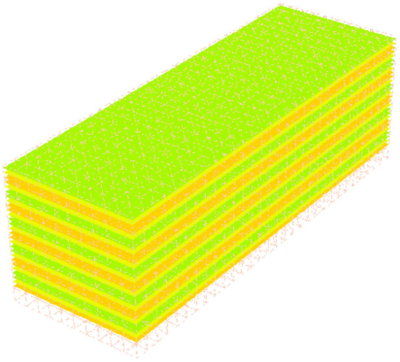


FIG. 3. *Heterogeneity pattern of Young's modulus and Poisson's ratio for elasticity matrices.*

4.2. Test cases. The Ela matrices arise from the linear elasticity problem with Dirichlet and Neumann boundary conditions defined as follows:

$$\text{div}(\sigma(u)) + f = 0 \quad \text{on } \Omega, \tag{4.1}$$

$$u = 0 \quad \text{on } \partial\Omega_D, \tag{4.2}$$

$$\sigma(u) \cdot n = 0 \quad \text{on } \partial\Omega_N, \tag{4.3}$$

where Ω is some regular domain, e.g, a parallelepiped. We denote by $\partial\Omega_D$ the Dirichlet boundary, $\partial\Omega_N$ is the Neumann boundary, f is some body force, and u is the unknown displacement field. We denote by $\sigma(.)$ the Cauchy stress tensor given by Hooke's law: it can be expressed in terms of Young's modulus E and Poisson's ratio ν. For a more detailed description of the problem, see [15]. We consider a heterogeneous beam made of several layers of a hard material $(E_1, \nu_1) = (2 \times 10^{11}, 0.25)$ and a soft material $(E_2, \nu_2) = (10^7, 0.45)$, i.e., discontinuous E and ν (Figure 3). The matrices Ela_N correspond to this equation on a beam discretized with FreeFem++ [18] using a triangular mesh that is refined as N increases and the P1 finite element scheme. More precisely, the mesh used for generating the Ela_4 matrix contains $1600 \times 30 \times 30$ points on the corresponding vertices. This mesh is coarsened by dividing the number of vertices in each dimension by $2^{1/3}$ in order to construct the Ela_3 matrix, and so on and so forth for the Ela_2 and Ela_1 matrices. This test case is known to be difficult because the matrix is ill conditioned. In particular, the standard one-level preconditioners are not expected to be very effective [34].

As previously pointed out, ECG is an algebraic method that does not rely on any particular assumption on the matrix, except that it is SPD. As an illustration, we also test the implementation on the five largest SPD matrices coming from the Sparse Matrix Collection of Tim Davis (see [5]). Numerical properties of the test matrices are summarized in Table 2.

4.3. Results. In all the experiments, the tolerance is set as the default tolerance of PETSc, i.e., 10^{-5}, and the maximum number of iterations is set to $25,000$. The right-hand side is chosen uniformly random and normalized, and the initial guess is set to 0. We do not use any kind of threading and use 28 MPI processes per node. Unless otherwise stated, we use one OpenMP thread per MPI process—we also perform numerical experiments to observe the effect of threading in the last section.

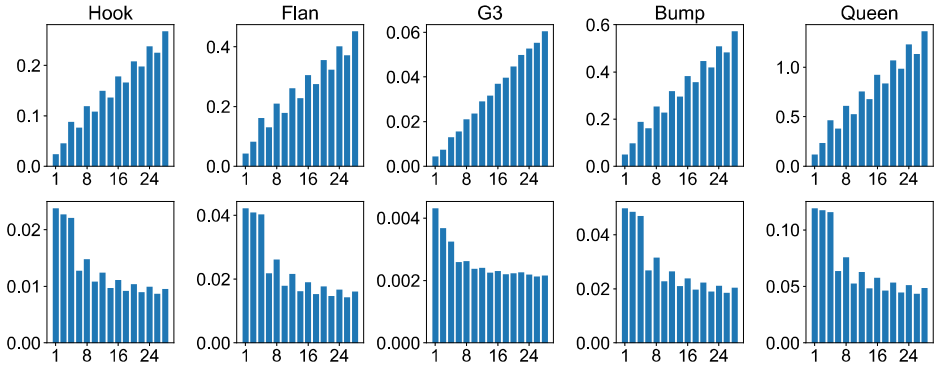
TABLE 2
Test matrices.

Name	Size	Nonzeros	Problem
Hook_1498	1,498,023	59,374,451	Structural problem
Flan_1565	1,564,794	117,406,044	Structural problem
G3_circuit	1,585,478	7,660,826	Circuit simulation problem
Bump_2911	2,911,419	130,378,257	Reservoir simulation
Queen_4147	4,147,110	316,548,962	Structural problem
Ela_1	615,168	21,373,272	Linear elasticity
Ela_2	1,210,800	42,611,160	Linear elasticity
Ela_3	2,383,125	84,726,039	Linear elasticity
Ela_4	4,615,683	165,388,197	Linear elasticity

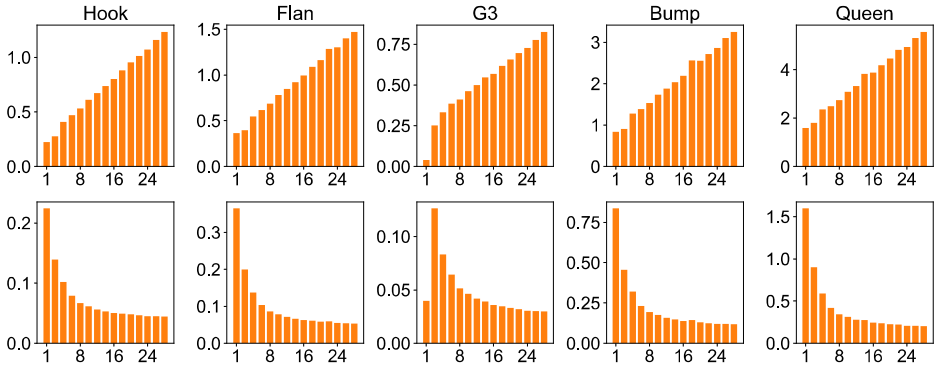
4.3.1. Impact of the enlarging factor. As a first step, we illustrate the behavior of the two operations that we did not take into account in our complexity analysis: the sparse matrix-set of vectors product, and the application of the block Jacobi preconditioner to a set of vectors. In Figure 4, we plot the runtimes we obtained for these two operations when varying the number of vectors in the right-hand side, with a fixed number of MPI processes set to 56, and for the matrices coming from Tim Davis's collection. The sparse matrix-set of vectors is computed using PETSc's routine `MatMatMult`, and the block Jacobi preconditioner is computed using the MKL-PARDISO direct solver. For both kernels, we observe a sublinear increase in the runtime; i.e., when the number of right-hand sides increases, the time per right-hand side is significantly reduced. However, we observe that the sparse matrix-set of vectors seems more sensitive to the number of right-hand sides, e.g., the runtime with eight right-hand sides is higher than with 10—we suspect that this is due to some cache effects. In comparison, the behavior of the application of the block Jacobi preconditioner is more regular. We want to point out that G3_circuit has a very particular, and undesired, behavior because the runtime is increasing superlinearly when the number of right-hand sides is increasing. This is likely due to the particular structure of the matrix—it is much sparser than the others—and the internal functioning of MKL-PARDISO; it makes ECG not suitable for this matrix.

We now study the impact of the enlarging factor t on the methods. More precisely, we fix the number of processors to 56 and we vary the value of t for the four methods: Orthodir (Odir), Orthodir with dynamic reduction of the search directions (D-Odir), Orthomin (Omin), and breakdown-free Orthomin (BF-Omin).

The results obtained for the matrices coming from Tim Davis's collection are summarized in Table 3. First of all, we observe that for all these matrices Odir and Omin are indeed similar: the number of iterations is almost the same whatever the value of t. In particular, Omin does not break down for these matrices, and that is why we do not report the results we obtained with BF-Omin: they are the same as with Omin but with a slight increase in the runtime. Also, we do not report the runtime of D-Odir when t is strictly smaller than 6 because the runtime is not reduced, and even slightly higher sometimes. For Flan_1565 and Hook_1498, we observe that the runtime is decreasing when t increases, and this decrease is greater for Flan_1565 (a bit smaller than a factor of 2). The "best" value of t is different for the two matrices (6 for Hook and 14 for Flan), but we also observe that there is not a very large variation of the runtime from one value of t to another. For instance, values ranging from 6 to 10 deliver almost the same runtime for Hook_1498. As we pointed out previously,



(a) Results for the sparse matrix-dense matrix product using PETSc. At the top, we indicate the total runtime (y-axis) with respect to the number of right-hand sides (x-axis). At the bottom, it is the total runtime divided by the number of right-hand sides (y-axis) with respect to the number of right-hand sides (x-axis).



(b) Results for the block Jacobi preconditioner's application with MKL-PARDISO. At the top, we indicate the total runtime (y-axis) with respect to the number of right-hand sides (x-axis). At the bottom, it is the runtime divided by the number of right-hand sides (y-axis) with respect to the number of right-hand sides (x-axis).

FIG. 4. *Runtime results (in seconds) for the key operations within an iteration of ECG: the application of the matrix to a set of vectors (Figure* 4a*) and the application of the block Jacobi preconditioner (Figure* 4b*). The number of processors is set to* 56*, and both operations are repeated* 10 *times.*

G3_circuit is not adapted for ECG, and the runtime increases quite significantly when t increases. This poor performance is very likely due to the behavior of the routine for applying the block Jacobi preconditioner (see Figure 4b) because the number of iterations is decreasing rather effectively when t increases. For Bump_2911 and Queen_4147, we also observe that the runtime is increasing when t increases. Unlike G3_circuit, for these two matrices the sparse matrix-set of vectors product and the application of the preconditioner are efficient when increasing t. In this case, the cause is inherent to the method: the number of iterations is not decreasing enough to compensate the increase in runtime of one iteration when the value of t increases. This is particularly illustrated on Queen_4147, with Odir from $t = 2$ to $t = 28$: the number of iterations is decreased by only 20%. Furthermore, one could notice that the increase in runtime is indeed sublinear with respect to the increase of t. In all

TABLE 3

Runtime results (T_{tot}, in seconds) and the corresponding iteration count (iter) for the matrices coming from Tim Davis's collection with $N_{proc} = 56$. The enlarging factor is denoted by t. The ++ means that the maximum number of iterations $(25,000)$ was reached.

		Hook		Flan		G3		Bump		Queen	
Method	t	T_{tot}	iter	T_{tot}	iter	T_{tot}	iter	T_{tot}	iter	T_{tot}	iter
Odir	2	16.2	357	51.1	776	8.3	390	97.5	703	285.5	1056
	4	12.8	217	40.8	456	9.6	336	109.6	607	350.4	992
	6	11.7	176	32.5	341	11.1	288	112.1	572	350.6	940
	8	12.2	149	31.0	268	11.3	251	121.6	532	408.9	942
	10	11.9	131	28.6	233	11.6	213	128.1	515	415.5	901
	12	13.0	122	29.5	204	12.7	201	142.0	503	457.9	877
	14	13.3	114	28.4	185	13.3	182	154.7	507	476.2	863
	16	14.3	108	29.6	171	13.5	170	165.6	472	527.7	860
	18	14.6	104	28.7	158	14.0	157	162.6	455	537.2	841
	20	15.9	100	30.8	148	14.5	149	181.7	447	610.6	831
	22	16.3	96	30.4	141	15.7	145	197.8	462	626.8	820
	24	16.8	91	31.3	132	15.9	139	213.2	465	685.4	819
	26	17.5	89	31.5	127	16.8	133	221.1	455	708.9	827
	28	18.3	86	32.5	121	17.1	128	241.2	465	742.6	799
D-Odir	6	11.4	177	32.4	345	10.7	290	112.7	580	353.2	963
	8	11.8	150	30.8	269	11.6	253	121.1	540	409.9	967
	10	11.4	132	28.2	235	11.4	215	125.3	523	351.8	927
	12	12.5	124	29.0	206	12.2	203	139.1	516	460.5	895
	14	12.5	115	27.5	188	12.9	184	149.3	532	475.1	882
	16	13.3	110	28.7	173	13.0	171	153.4	483	514.8	885
	18	13.4	106	27.5	161	13.5	158	156.1	465	533.5	863
	20	14.4	101	29.2	150	14.0	151	171.4	455	502.2	861
	22	15.5	97	28.7	143	14.8	147	178.9	475	606.7	843
	24	15.2	92	29.4	134	14.8	141	193.5	488	637.2	851
	26	15.7	90	29.5	130	15.7	135	198.2	484	674.2	858
	28	16.4	87	30.4	123	16.2	131	209.9	486	710.4	827
Omin	2	14.7	357	50.9	776	8.5	390	96.0	701	284.1	1057
	4	12.4	217	40.2	456	9.3	336	108.8	610	345.6	991
	6	11.2	176	31.4	341	10.0	288	109.5	575	341.8	937
	8	11.6	149	31.4	268	10.7	251	123.3	532	401.2	944
	10	11.3	131	27.6	233	10.6	213	123.6	515	403.2	901
	12	12.4	122	28.4	204	11.5	201	137.4	504	450.4	878
	14	12.6	114	27.0	185	12.2	182	149.0	507	466.2	866
	16	13.5	108	28.4	171	13.7	170	152.3	471	512.6	858
	18	13.8	104	27.5	158	12.7	157	157.0	456	466.5	842
	20	15.2	100	29.6	148	13.2	149	173.6	447	594.0	833
	22	15.4	96	29.1	141	14.1	145	188.9	462	599.7	821
	24	16.0	91	29.8	132	14.3	139	204.1	465	641.0	821
	26	16.5	89	29.8	127	15.2	133	210.6	455	685.1	830
	28	17.3	86	31.0	121	15.4	128	231.5	465	712.6	801

the cases, the D-Odir method allows one to reduce the runtime with respect to Odir when t becomes large; e.g, for $t = 28$, the runtime is decreased for all the matrices. Unfortunately, this reduction is not significant enough to make ECG attractive for G3_circuit, Bump_2911, and Queen_4147. For these matrices, the runtime is still increasing with t. For Hook_1498 and Flan_1565, it allows one to mitigate the added costs of Odir compared to Omin. In fact, as illustrated in Figure 5 for the Flan_1565 matrix, the reduction of the search directions usually occurs when the convergence has already started. This is why in practice we observe gains, in terms of runtime with respect to Odir, of the order of 5% to 10%.

We perform the exact same experiments on Ela_1 matrix, also using 56 MPI

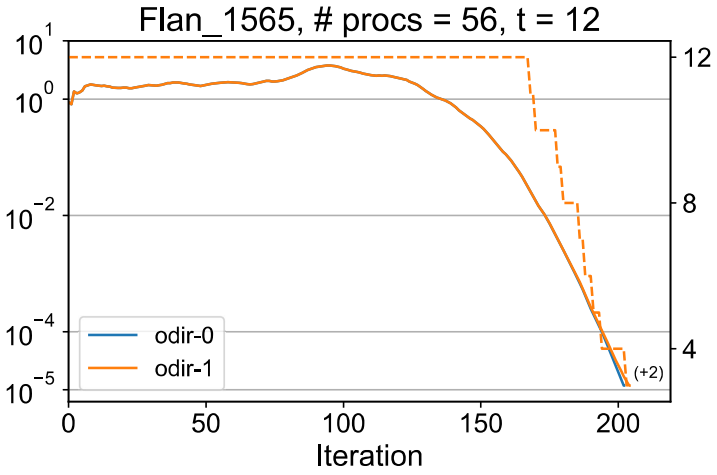


FIG. 5. *Convergence (scale on the left) of D-Odir (odir-1) compared to Odir (odir-0) for the Flan matrix. The dashed line represents the number of search directions for D-Odir (scale on the right). Between parentheses, we indicate the difference of iteration count to reach convergence between D-Odir and Odir (+ means that D-Odir took more iterations to converge).*

TABLE 4

Runtime results (T_{tot}, in seconds) and the corresponding iteration count (iter) for Ela_1 with $N_{proc} = 56$. The enlarging factor is denoted by t. The ++ means that the maximum number of iterations (25,000) was reached, and the – means that a breakdown occurred.

	Odir		D-Odir		Omin		BF-Omin	
t	T_{tot}	iter	T_{tot}	iter	T_{tot}	iter	T_{tot}	iter
2	60.1	3320		++		–	218.1	13,939
4	32.0	1362		++		–	138.4	5,357
6	23.6	868		++		–	524.6	19,898
8	15.1	461	14.1	457		–	492.3	16,426
10	14.0	375	12.3	363		–	515.8	15,074
12	14.5	334	14.2	403		–	854.4	23,176
14	14.1	296	12.2	300		–	568.8	13,333
16	15.0	265	13.4	285		–		++
18	14.9	252	13.2	254		–		++
20	16.2	231	12.5	237		–		++
22	15.5	224	13.9	228		–		++
24	17.2	213	12.8	220		–	998.0	15,896
26	17.3	206	14.5	209		–	1,273.8	18,528
28	18.8	200	14.6	202		–		++

processes, and we summarize the results in Table 4. First of all, we observe that Omin breaks down for this matrix for all the values of t tested. Using BF-Omin effectively cures the breakdowns, but the convergence becomes very slow, and for some values of t the method simply does not converge within the prescribed maximum number of iterations. The resulting runtimes are both very slow (with respect to Odir) and unstable because from one value of t to another the resulting number of iterations is changing significantly. For instance, for $t = 12$ the number of iterations is roughly $13,000$, and for $t = 14$ it is roughly $23,000$. On the contrary, Odir is very stable and converges for all the values of t we have tested. The resulting runtime is decreasing

up to $t = 8$, being almost a factor of 4 times less than for $t = 2$, and then it starts to increase slightly when t is larger than 20. Finally, D-Odir is also unstable when t is smaller than 6 and it cannot reach the required accuracy within the prescribed maximum number of iterations. We want to emphasize that one should not expect to obtain a significant gain with D-Odir when t is small. However, when t is large, as for Tim Davis's matrices, D-Odir is slightly faster than Odir. For instance, when $t = 20$, D-Odir is 25% faster than Odir.

In conclusion, D-Odir is the best method over the different variants of ECG that were tested: it shows a good trade-off between the stability of Odir and the efficiency of Omin. The results support the theoretical convergence study done in the previous section. ECG(t) is acting as if the t smallest eigenvalues of the matrix are deflated. Nevertheless, there exist matrices such as Bump_2911 or Queen_4147 for which the reduction of the number of iterations does not compensate the extra cost of ECG compared to standard CG, even when using the dynamic reduction of the search directions. Also, the resulting runtime highly depends on the optimization of the sparse matrix product and the application of the preconditioner to a set of vectors. For example, ECG shows very poor performances for G3_circuit because the sparse matrix-set of vectors product is not optimized for this matrix.

4.3.2. Strong scaling study. Following the parameter study, we perform a strong scaling study on Hook_1498, Flan_1565, Queen_4147, and Ela_4. As G3_circuit is not particularly well suited for the method, we do not perform the strong scaling study on this matrix. For the sake of brevity, we omit the results obtained with Bump_2911 because they are similar to those obtained with Queen_4147. There is an interplay between the choice of t and the number of MPI processes. Indeed, increasing the number of MPI processes deteriorates the quality of the preconditioner and reduces its application cost. However, for the sake of simplicity, we decide to keep the value of t constant while increasing the number of MPI processes.

The results are shown in Figure 6 (color is available online only). More precisely, we compare PETSc PCG (blue bars), ECG (orange bars), and a modified PETSc PCG (green bars), where the sparse matrix-vector is applied using the `MatMatMult` routine, i.e, the vector is regarded as a dense matrix with one column. For Hook_1498, we use D-Odir and set $t = 10$, which corresponds to one of the best over the values of t we tested in the previous study. We observe that D-Odir is faster than PETSc PCG when the number of MPI processes is relatively low (252 and 504), but when it is large (more than $1,000$) PETSc PCG becomes almost twice as fast. For the other matrices, we also observe that the performance of ECG deteriorates significantly with respect to PETSc PCG when the number of MPI processes becomes large. This is because the routine `MatMatMult` is underoptimized when the number of MPI processes is large and the number of columns in the right-hand side is very low. For example, for Hook_1498 and Flan_1565, when $N_{\text{proc}} = 2,016$, the `MatMult` routine (sparse matrix-vector product) is around 10 times faster than the `MatMatMult` routine, where the right-hand side is regarded as a dense matrix with one column (see Figure 7). The total runtime per right-hand side of `MatMatMult` is indeed slightly lower than the runtime of `MatMult` when the number of right-hand sides is large enough; in this case, the total runtime of `MatMatMult` is significantly larger. However, if the number of right-hand sides is not large enough—which is the case in our strong scaling study—then the runtime per right-hand side is larger than the runtime of the `MatMult` routine. Furthermore, the gap is increasing when the number of MPI processes increases.

Hence we also compare ECG with a modified PETSc PCG (green bars in Fig-

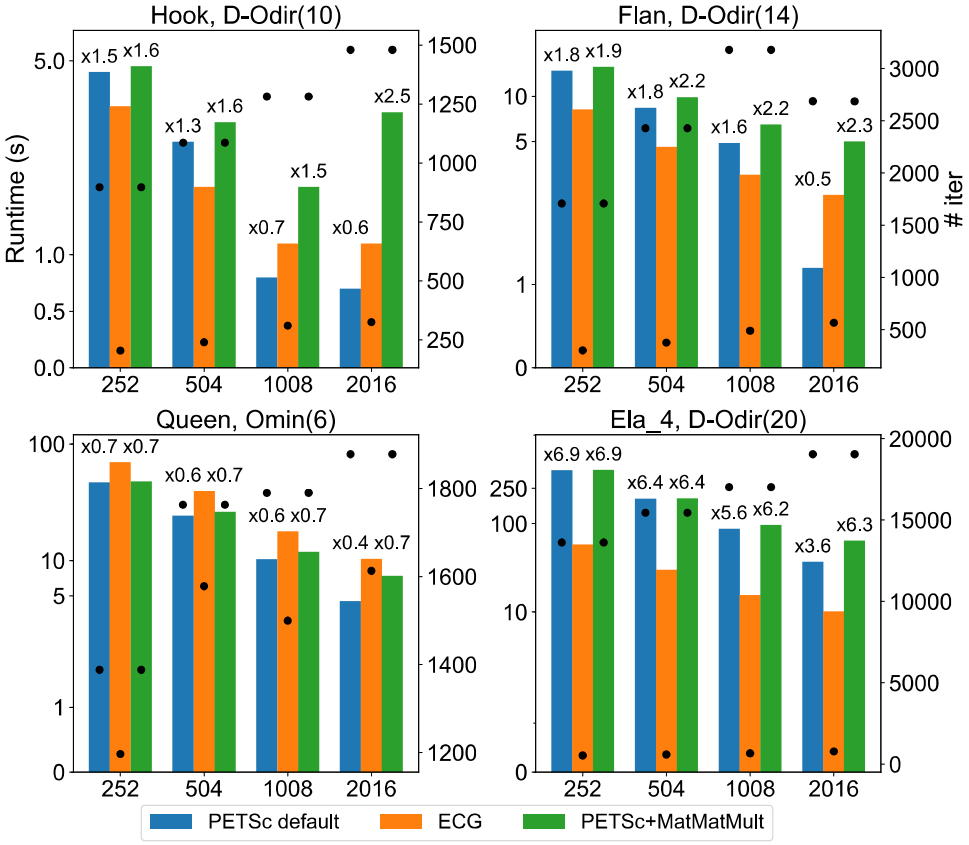


FIG. 6. *Strong scaling results for Hook_1498, Flan_1565, Queen_4147, and Ela_4 with N_{proc} varying from 252 to 2016. We indicate both the runtimes, with bars (left scale), and the iteration counts, with black dots (right scale). The speed-up with respect to ECG is indicated on top of the bars.*

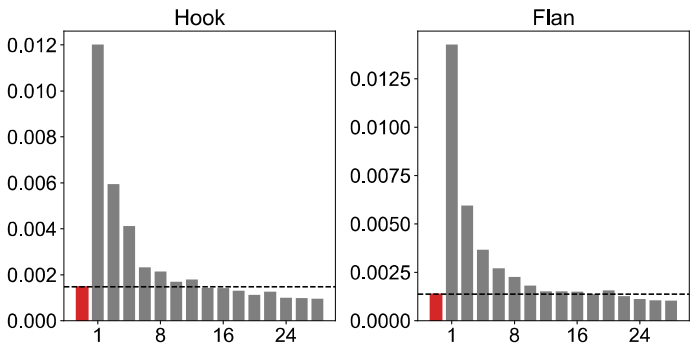


FIG. 7. *Runtime results per right-hand side (in seconds) for the sparse matrix-dense matrix product using PETSc (`MatMatMult` routine) with $N_{proc} = 2,016$ for Hook_1498 and Flan_1565. In red (color is available online only), we indicate the runtime of a call to the `MatMult` routine.*

ure 6) where the routine `MatMatMult` is used for computing the sparse matrix-vector product. We believe that this comparison is relevant because both ECG and this modified PETSc PCG rely on the exact same routine for computing the sparse matrix application to a (set of) vector(s). We indeed observe that this modified version of PETSc PCG is less scalable than the default one; e.g, for Hook_1498 the runtime is increasing from $1,008$ to $2,016$ MPI processes. Furthermore, we observe that this modified version of PETSc PCG is also less scalable than ECG; for Hook_1498, Flan_1565, and Ela_4, the speed-up is slightly increasing from $1,008$ to $2,016$ MPI processes. As previously suggested by the parameter study, Queen_4147 is not very adapted for ECG, but even for this matrix, ECG is scaling as well as the modified PETSc PCG.

In conclusion, we have shown that ECG's scaling is highly dependent on the routine that performs the sparse matrix-set of vectors product. This is of course not very surprising. What is more surprising, however, is the fact that the `MatMult` routine of PETSc is much more scalable than the `MatMatMult` when the number of right-hand sides is small. In practice, this explains the difference in terms of the scalability of ECG compared to PETSc PCG. We believe that it should be possible to optimize the `MatMatMult` routine so that this difference would at least be reduced, or even be removed. Also, we want to emphasize that enlarging the Krylov subspaces is not incompatible with other techniques currently developed in order to increase the performances of Krylov methods. For instance, we mention that we are currently performing four calls to `MPI_Allreduce` per iteration, but that could be reduced to two, and even one with Odir (and D-Odir), by fusing them. Furthermore, we could use pipelining [13] or communication avoiding based on s-step methods [20, 3] on top of ECG—that would require us to take into account a possible loss of numerical stability of the method.

4.3.3. Dependence on the mesh size—weak scaling study. Given the importance of the parameter t regarding the efficiency of the method, we perform a study of the convergence of the method with respect to the mesh size for the elasticity test case. More precisely, we consider the Ela_N matrices ($N = 1, \ldots, 4$). Our major focus is not the weak scaling of ECG, but rather the comparison between PETSc's CG and D-Odir in terms of runtime.

As for the strong scaling study, we use D-Odir(20). The results are summarized in Table 5. We observe that D-Odir(20) is always at least 3.5 times faster than PETSc PCG. However, the gap tends to slightly decrease when the number of MPI processes increases. Indeed, when $N_{\text{proc}} = 256$, D-Odir(20) is up to 3.8 times faster than PETSc PCG. As outlined in the strong scaling study, the `MatMatMult` routine scales poorly (compared to the `MatMult` routine) when the number of columns in the right-hand side is very small. Thus, we also indicate, in the column labelled T_{MMM}, the runtimes when replacing the sparse matrix-vector product within PETSc PCG by a call to `MatMatMult`. In this case, the gap between D-Odir(20) and the modified PETSc PCG is increasing when the number of MPI processes is increasing. For instance, when $N_{\text{proc}} = 2016$, D-Odir(20) is around 6.5 times faster than the modified PETSc PCG, whereas it is 4.8 times faster for the smallest problem.

4.3.4. Impact of threads on performance. One motivation for enlarging the Krylov subspaces is to increase the arithmetic intensity of the resulting methods. This is particularly interesting to take advantage of the so-called manycore architecture, such as Nvidia GPUs, Intel Xeon Phi, or Sunway SW26010 used in the Sunway TaihuLight supercomputer. As the implementation relies on the MKL library which

TABLE 5
Weak scaling study. The dimension of the matrix is denoted by n, and t denotes the enlarging factor. The ratio between PETSc runtime and ECG runtime is indicated between parentheses.

			D-Odir		PETSc CG		
# MPI	n	t	# iter	T_{tot}	# iter	T_{def}	T_{MMM}
252	6.15×10^5	20	360	3.4	8,803	13.0 (×3.8)	16.3 (×4.8)
504	1.21×10^6	20	463	4.9	11,333	17.6 (×3.6)	23.7 (×4.8)
1008	2.38×10^6	20	608	6.8	14,801	23.8 (×3.5)	36.8 (×5.4)
2016	4.61×10^6	20	784	10.2	19,047	36.1 (×3.5)	64.0 (×6.4)

is multithreaded [37], it is straightforward to assess its efficiency on the Xeon Phi processors.

In order to do so, we perform the following experiments on NERSC's supercomputer Cori. It consists of two partitions, one with Intel Haswell processors and another with the last generation of Intel Xeon Phi processors: Knights Landing (KNL). More precisely, the second partition consists of 9,688 single-socket Intel Xeon Phi 7250 (KNL) processors with 68 cores each. For a detailed description of the machine, we refer the reader to the online documentation.[4] We compile the code (and its dependencies) using the default compilers and libraries installed on the machine: `icc` version 18.0.1, `cray-mpich` version 7.6.2, MKL version 2018.1.163, and METIS version 5.1.0. We have installed PETSc, and as for Kebnekaise, it has been linked with MKL so that it uses MKL-PARDISO in the block Jacobi preconditioner. We consider the Ela_4 test case, and we study the impact of threads on the strong scaling of Odir(20). We do not use the dynamic reduction of the search directions in order to keep the cost of one iteration constant during the solve to better understand the effect of threading. We fix the number of MPI processes to 2048, and we increase the number of threads from one to eight—this means at most $2,048 \times 8 = 16,384$ threads, each one being bound to one physical core.

The results obtained are summarized in Table 6. We observe that using more than two threads, and up to eight, always has a significant effect on the speed-up, even when the number of MPI processes is high. For instance, as shown in Table 6, increasing the number of threads from one to eight with a fixed number of $2,048$ MPI processes leads to a decrease in runtime of 2. Of course, we are not close to full efficiency when using multiple threads, but we are still taking advantage of the BLAS 3 routines. Indeed, the corresponding speed-up with PETSc PCG is only 1.5. In particular, we observe that the difference between ECG's speed-up and PETSc PCG's speed-up is increasing when the number of threads is increasing. Thus, ECG is more adapted to the current trend in hardware architecture for reaching exascale, namely manycore processors.

5. Conclusion. In this paper, we have studied the ECG method. It relies on so-called enlarged Krylov subspaces which can be seen as particular cases of block Krylov subspaces. The parallel efficiency of the approach has been assessed, and we have shown that this method is scalable up to $16,384$ cores and it is up to 6.9 times faster than PETSc's implementation of PCG.

First, we have thoroughly studied the method from a theoretical point of view under the assumption of exact arithmetic. Starting from the theory, we have exhibited the relationship between the two variants of the method (Orthodir and Orthomin)

[4] http://www.nersc.gov/users/computational-systems/cori/configuration/

TABLE 6
Runtime results (in seconds) on the Ela_4 matrix when $N_{proc} = 2,048$. We indicate the speed-up when increasing the number of threads for each method.

	PETSc PCG		Odir(20)	
# omp	T_{tot}	speed-up	T_{tot}	speed-up
1	83.3	–	32.6	–
2	75.4	1.1	25.0	1.3
4	62.5	1.3	19.5	1.7
8	56.1	1.5	16.6	2.0

and thus explained their differences in terms of robustness. We also have studied its convergence rate, and we have shown that the ECG method is acting as if t arbitrary eigenvalues at the end of the spectrum of the matrix were somehow deflated, where t is the enlarging factor (the initial block size). Then we have described the parallel design of the method, including the two variants as well as their dynamic versions, where the number of search directions is adaptively reduced during the iterations [21, 16]. Numerical experiments show that enlarging the Krylov subspaces allows us to reduce significantly the number of iterations with respect to the standard PCG method. Furthermore, the reduction of the search directions allows us to reduce the cost of the extra arithmetic operations induced by the method. Overall, the proposed solver is up to 6.9 times faster than PETSc PCG for an elasticity matrix. Also, as our implementation is based on BLAS 3 kernels only, it improves the scaling when increasing the number of threads per MPI process. Thus, it is scaling up to 16,384 threads, each one bound to a physical core, and it is well adapted for manycore architectures.

Throughout this work, the only assumption we make is that the matrix is SPD. Hence the resulting methods are very generic and completely algebraic. For instance, it is straightforward to use D-Odir for solving linear systems with several right-hand sides. Similarly, ECG can be used with any preconditioner that could be used with CG. Thus, it can be integrated very easily in any existing code. Of course, many parameters come into play when the performance is considered, and ECG does not always overtake the standard CG method in terms of runtime. However, we have observed large speed-ups, up to 6.9 on elasticity matrices, for example. As ECG increases the arithmetic intensity and reduces the communication, it is well suited for modern and future architectures that exhibit massive parallelism. In order to observe this in practice, it is crucial that both the application of the operator (matrix) and of the preconditioner to a set of vectors scale sublinearly with respect to the number of vectors. We have found that it was not always the case with the `MatMatMult` routine of PETSc (with respect to `MatMult`). Thus, an interesting direction for future research is to reduce, and maybe even to remove, this gap. According to the theoretical study and the numerical experiments, ECG is particularly well adapted for matrices with a small number of small eigenvalues.

REFERENCES

[1] S. F. ASHBY, T. A. MANTEUFFEL, AND P. E. SAYLOR, *A taxonomy for conjugate gradient methods*, SIAM J. Numer. Anal., 27 (1990), pp. 1542–1568, https://doi.org/10.1137/0727091.

[2] S. BALAY, S. ABHYANKAR, M. F. ADAMS, J. BROWN, P. BRUNE, K. BUSCHELMAN, L. DALCIN, V. EIJKHOUT, W. D. GROPP, D. KAUSHIK, M. G. KNEPLEY, D. A. MAY, L. C. MCINNES, K. RUPP, P. SANAN, B. F. SMITH, S. ZAMPINI, H. ZHANG, AND H. ZHANG, *PETSc Users*

Manual, Tech. Report ANL-95/11 - Revision 3.8, Argonne National Laboratory, Lemont, IL, 2017, http://www.mcs.anl.gov/petsc.

[3] E. Carson, N. Knight, and J. Demmel, *Avoiding communication in nonsymmetric Lanczos-based Krylov subspace methods*, SIAM J. Sci. Comput., 35 (2013), pp. S42–S61, https://doi.org/10.1137/120881191.

[4] A. T. Chronopoulos, *s-step iterative methods for (non)symmetric (in)definite linear systems*, SIAM J. Numer. Anal., 28 (1991), pp. 1776–1789, https://doi.org/10.1137/0728088.

[5] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), 1, https://doi.org/10.1145/2049662.2049663.

[6] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-optimal parallel and sequential QR and LU factorizations*, SIAM J. Sci. Comput., 34 (2012), pp. A206–A239, https://doi.org/10.1137/080731992.

[7] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick, *Minimizing communication in sparse matrix solvers*, in Proceedings of the ACM/IEEE Supercomputing SC9 Conference, 2009.

[8] J. W. Demmel, L. Grigori, M. Gu, and H. Xiang, *Communication avoiding rank revealing QR factorization with column pivoting*, SIAM J. Matrix Anal. Appl., 36 (2015), pp. 55–89, https://doi.org/10.1137/13092157X.

[9] A. J. Dongarra, V. Eijkhout, and A. Kalhan, *Reverse Communication Interface for Linear Algebra Templates for Iterative Methods*, Tech. Report, 1995, http://www.netlib.org/lapack/lawnspdf/lawn99.pdf.

[10] Z. Dostál, *Conjugate gradient method with preconditioning by projector*, Int. J. Comput. Math., 23 (1988), pp. 315–323, https://doi.org/10.1080/00207168808803625.

[11] A. Dubrulle, *Retooling the method of block conjugate gradients*, Electron. Trans. Numer. Anal., 12 (2001), pp. 216–233.

[12] A. el Guennouni, K. Jbilou, and H. Sadok, *A block version of BiCGSTAB for linear systems with multiple right-hand sides*, Electron. Trans. Numer. Anal., 16 (2003), pp. 129–142.

[13] P. Ghysels and W. Vanroose, *Hiding global synchronization latency in the preconditioned conjugate gradient algorithm*, Parallel Comput., 40 (2014), pp. 224–238, https://doi.org/10.1016/j.parco.2013.06.001.

[14] L. Grigori, S. Moufawad, and F. Nataf, *Enlarged Krylov subspace conjugate gradient methods for reducing communication*, SIAM J. Matrix Anal. Appl., 37 (2016), pp. 744–773, https://doi.org/10.1137/140989492.

[15] L. Grigori, F. Nataf, and S. Yousef, *Robust Algebraic Schur Complement Preconditioners Based on Low Rank Corrections*, Research Report RR-8557, 2014, https://hal.inria.fr/hal-01017448.

[16] L. Grigori and O. Tissot, *Reducing the Communication and Computational Costs of Enlarged Krylov Subspaces Conjugate Gradient*, Research Report RR-9023, 2017, https://hal.inria.fr/hal-01451199.

[17] M. H. Gutknecht, *Block Krylov space methods for linear systems with multiple right-hand sides: An introduction*, in Modern Mathematical Models, Methods and Algorithms for Real World Systems, A. H. Siddiqi, I. S. Duff, and O. Christensen, eds., Anamaya, New Delhi, India, 2007, pp. 420–447.

[18] F. Hecht, *New development in freefem++*, J. Numer. Math., 20 (2012), pp. 251–265.

[19] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Research Nat. Bur. Standards, 49 (1952), pp. 409–436.

[20] M. Hoemmen, *Communication-Avoiding Krylov Subspace Methods*, Ph.D. thesis, University of California, Berkeley, Berkeley, CA, 2010.

[21] H. Ji and Y. Li, *A breakdown-free block conjugate gradient method*, BIT, 57 (2017), pp. 379–403, https://doi.org/10.1007/s10543-016-0631-z.

[22] P. Jiránek, M. Rozložník, and M. H. Gutknecht, *How to make simpler GMRES and GCR more stable*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 1483–1499, https://doi.org/10.1137/070707373.

[23] P. Jolivet, *Domain Decomposition Methods. Application to High-Performance Computing*, theses, Université de Grenoble, Grenoble, France, 2014, https://tel.archives-ouvertes.fr/tel-01155718.

[24] P. Jolivet and P. Tournier, *Block iterative methods and recycling for improved scalability of linear solvers*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '16, IEEE Press, Piscataway, NJ, 2016, 17, http://dl.acm.org/citation.cfm?id=3014904.3014927.

[25] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1998), pp. 359–392, https://doi.org/10.1137/

S1064827595287997.

[26] M. Kreutzer, J. Thies, M. Röhrig-Zöllner, A. Pieper, F. Shahzad, M. Galgon, A. Basermann, H. Fehske, G. Hager, and G. Wellein, *GHOST: Building blocks for high performance sparse linear algebra on heterogeneous systems*, Internat. J. Parallel Programming, 45 (2017), pp. 1046–1072, https://doi.org/10.1007/s10766-016-0464-z.

[27] J. Langou, *Iterative Methods for Solving Linear Systems with Multiple Right-Hand Sides*, Ph.D. thesis, CERFACS, Toulouse, France, 2003.

[28] J. Málek and Z. Strakoš, *Preconditioning and the Conjugate Gradient Method in the Context of Solving PDEs*, SIAM Spotlights 1, SIAM, Philadelphia, 2015, https://doi.org/10.1137/1.9781611973846.

[29] D. P. O'Leary, *The block conjugate gradient algorithm and related methods*, Linear Algebra Appl., 29 (1980), pp. 293–322.

[30] M. Robbé and M. Sadkane, *Exact and inexact breakdowns in the block GMRES method*, Linear Algebra Appl., 419 (2006), pp. 265–285.

[31] Y. Saad, *On the rates of convergence of the Lanczos and the block-Lanczos methods*, SIAM J. Numer. Anal., 17 (1980), pp. 687–706, https://doi.org/10.1137/0717059.

[32] Y. Saad, *Iterative Methods for Sparse Linear Systems*, SIAM, Philadelphia, 2003, https://doi.org/10.1137/1.9780898718003.

[33] N. Spillane, *An adaptive multipreconditioned conjugate gradient algorithm*, SIAM J. Sci. Comput., 38 (2016), pp. A1896–A1918, https://doi.org/10.1137/15M1028534.

[34] N. Spillane, V. Dolean, P. Hauret, F. Nataf, C. Pechstein, and R. Scheichl, *Abstract robust coarse spaces for systems of PDEs via generalized eigenproblems in the overlaps*, Numer. Math., 126 (2014), pp. 741–770, https://doi.org/10.1007/s00211-013-0576-y.

[35] J. M. Tang, R. Nabben, C. Vuik, and Y. A. Erlangga, *Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods*, J. Sci. Comput., 39 (2009), pp. 340–370, https://doi.org/10.1007/s10915-009-9272-6.

[36] R. Thakur, R. Rabenseifner, and W. Gropp, *Optimization of collective communication operations in MPICH*, Int. J. High Perform. Comput. Appl., 19 (2005), pp. 49–66, https://doi.org/10.1177/1094342005051521.

[37] E. Wang, Q. Zhang, B. Shen, G. Zhang, X. Lu, Q. Wu, and Y. Wang, *Intel math kernel library*, in High-Performance Computing on the Intel® Xeon Phi, Springer, Cham, 2014, pp. 167–188.

[38] I. Yamazaki, S. Tomov, and J. Dongarra, *Mixed-precision Cholesky QR factorization and its case studies on multicore CPU with multiple GPUs*, SIAM J. Sci. Comput., 37 (2015), pp. C307–C330, https://doi.org/10.1137/14M0973773.