Block Iterative Methods and Recycling for Improved Scalability of Linear Solvers

Pierre Jolivet, Pierre-Henri Tournier

Block Iterative Methods and Recycling for Improved Scalability of Linear Solvers

Pierre Jolivet*
*CNRS
IRIT–ENSEEIHT
pierre.jolivet@enseeiht.fr

Pierre-Henri Tournier‡§
‡Laboratoire J.-L. Lions, UPMC
§Inria Paris, ALPINES research team
tournier@ann.jussieu.fr

***Abstract*—Contemporary large-scale Partial Differential Equation (PDE) simulations usually require the solution of large and sparse linear systems. Moreover, it is often needed to solve these linear systems with different or multiple Right-Hand Sides (RHSs). In this paper, various strategies will be presented to extend the scalability of existing multigrid or domain decomposition linear solvers using appropriate recycling strategies or block methods—i.e., by treating multiple right-hand sides simultaneously.**

The scalability of this work is assessed by performing simulations on up to 8,192 cores for solving linear systems arising from various physical phenomena modeled by Poisson's equation, the system of linear elasticity, or Maxwell's equation.

This work is shipped as part of on open-source software, readily available and usable in any C/C++, Python, or Fortran code. In particular, some simulations are performed on top of a well-established library, PETSc, and it is shown how our approaches can be used to decrease time to solution down by 30%.

***Index Terms*—Iterative methods, distributed algorithms, Maxwell's equation**

I. Introduction

Discretizations of PDEs used to model physical phenomena typically lead to larger and larger systems that cannot be solved directly and require both 1) advanced preconditioning techniques and 2) efficient iterative methods. In recent years, a lot of efforts have been made to design highly scalable preconditioners for computational fluid dynamics [1], [2] or solid mechanics [3], [4], for example with multigrid [5] or domain decomposition [6] methods. Most of these advanced preconditioners, however, rely on basic iterative methods, such as the Generalized Minimal RESidual method [7] (GMRES) or the Preconditioned Conjugate Gradient [8] (PCG). Still, numerous new or modified iterative methods have been developed to: pipeline reductions [9], [10], avoid synchronizations [11], [12], decrease the number of iterations by means of multiple search directions [13], [14] or Krylov subspace recycling [15], [16]. Iterative methods tailored to tackle efficiently problems with multiple right-hand sides have also blossomed [17]–[19]. Finally, it can also be beneficial to couple block methods and recycling [20], [21].

The contribution of this paper is threefold, we present:

- a uniform implementation of a pseudo-block[1] and block Krylov solver based on an existing theoretical work [22],
- large-scale experiments using the aforementioned implementation on top of a well-established parallel library, PETSc [23], [24],
- a scalable solver for Maxwell's equation with multiple right-hand sides using overlapping Schwarz methods with optimized boundary conditions [25], [26].

The paper is organized as follows. In section II, we present related work and limitations of current implementations of (pseudo-)block iterative methods with or without recycling. In section III, we analyze the theoretical work presented by Parks et al. [22] in the context of high-performance computing. We also extend this study to the case of non-variable sequence of linear systems—i.e., when only right-hand sides are changing but not the linear operator itself—and to variable preconditioning, as already done theoretically [27], [28]. In section IV, we use PETSc to generate large linear systems on up to 8,196 cores. We then compare our open-source implementation[2] against existing subspace recycling strategies already implemented in the framework and we show that our approach can be used to decrease time to solution down by 30%. In section V, we investigate the potential of (pseudo-)block iterative methods over standard methods when direct solvers are used to define a preconditioner, for example in the context of domain decomposition methods. Eventually, we integrate these block methods inside a solver for Maxwell's equation and show a relative speedup of up to 450% against more traditional solvers.

II. Related work

A. Subspace recycling

The work presented in this paper is mostly based on the Generalized Conjugate Residual method with inner Orthogonalization and Deflated Restarting method [22] (GCRO-DR), which itself is an extension of a prior work by de Sturler [29]. GCRO-DR was developed in the context of fatigue and fracture modeling via finite element analysis where it is usually required to solve a sequence of linear systems:

$$A_i X_i = B_i \qquad i = 1, 2, \ldots \tag{1}$$

where the coefficient matrices $A_i \in \mathbb{K}^{n \times n}$ and the right-hand sides $B_i \in \mathbb{K}^{n \times p}$ might change from one index i to the next.

[1] method were operations for each RHS are fused together, cf. section V-B1

[2] available at https://github.com/hpddm/hpddm

SC16; Salt Lake City, Utah, USA; November 2016
978-1-4673-8815-3/16/$31.00

In the original paper, each linear system is solved with a single right-hand side, i.e. $p = 1$. A MATLAB implementation has ever since been available to try this method, with either left or right preconditioning. A flexible variant of GCRO has then been proposed [27], and eventually a flexible variant of GCRO-DR was derived [28]. In the latter reference, it is proved that under certain circumstances, FGCRO-DR is algebraically equivalent to FGMRES-DR [30], another flexible variant of well-established iterative method that uses recycling to improve the numerical efficiency of restarted GMRES [31]. The main advantage of the class of GCRO-DR based methods over the class of GMRES-DR based methods is that they may be used for the solution of sequences of linear systems.

B. Block iterative methods

One of the first iterative methods to be adapted to handle multiple right-hand sides at once, i.e. $p > 1$, was the Conjugate Gradient method [32]. A first study of the Block GMRES was proposed in the thesis of Vital [33], followed by new theoretical results [17]. Plenty of applications of these methods have been proposed since then [34], [35], mainly for the simulation of wave propagation phenomena [36], [37] or lattice quantum chromodynamics [38] with multiple sources. Since each of these sources may yield a different right-hand side, efficient block methods are needed to handle demanding simulations with sequences of tall and skinny right-hand sides B_i.

C. Distributed iterative methods

In the context of large-scale distributed sparse linear algebra, there are multiple libraries available for solving linear systems with iterative methods:

- *hypre* [39], DUNE [40], and PARALUTION [41] are all shipped with standard iterative methods like GMRES or CG, but they lack recycling strategies and cannot handle linear systems with multiple right-hand sides.
- PETSc [23] comes with more advanced iterative methods like Loose GMRES [42] and Deflated GMRES [43], [44], but as implemented, these methods cannot be used to recycle Krylov subspace from one linear system solve to the next. The two aforementioned iterative methods cannot handle variable preconditioning as implemented in PETSc.
- Trilinos [45], through its Belos package [46], is the library that provides some of the most advanced iterative methods, and in particular: (pseudo-)Block GMRES, GCRO-DR, Block GCRO-DR. Note however some limitations of the package: no support of pseudo-Block GCRO-DR, no support of variable preconditioning with subspace recycling, and there is no binding to languages other than C++. Finally, it is not possible to speed up the recycling process when using non-variable linear systems—i.e., in eq. (1), $\forall i \in [\![1, 2, \ldots]\!], A_i = A_1$. In section III-B, it will be shown that this should be used whenever possible in order to increase performance.

III. Iterative methods with recycling

In this section, we will describe some interesting details of the GCRO-DR method introduced by Parks et al. [22] and share some insights into our implementation. We will also present how to deal efficiently with non-variable linear systems and variable preconditioning.

A. Generalized Conjugate Residual method with inner Orthogonalization and Deflated Restarting

In order to keep this paper as self-contained as possible, we recall in fig. 1 the original GCRO-DR of Parks et al. [22], extended to the case of multiple right-hand sides. The notations for GMRES(m) and GCRO-DR(m, k) are as follows:

- n is the size of all linear systems,
- p is the number of right-hand sides,
- m is the maximum dimension of Krylov subspaces,
- k is the dimension of recycled Krylov subspaces,
- V_{m+1} is an Arnoldi basis of $m + 1$ blocks of dimension $n \times p$,
- $\overline{H}_m$ is a block Hessenberg matrix of dimension $p \cdot (m+1) \times p \cdot m$ with blocks $\{h_{i,j}\}_{\substack{1 \leqslant i \leqslant m+1 \\ i \leqslant j+1 \leqslant m+1}}$ of size $p \times p$, H_m is the restriction of $\overline{H}_m$ to its first $p \cdot m$ rows.

The QR decomposition of a single-column matrix—i.e., a vector—$R_j = QR$ is unique and defined as $Q = \dfrac{R_j}{||R_j||}$ and $R = ||R_j||$ (line 11 and line 24 are the traditional way of defining the first vector of the Arnoldi basis for GMRES). The main difference between GMRES (resp. Augmented GMRES) and GCRO-DR is the solution of the eigenvalue problem line 16 (resp. generalized eigenvalue problem line 33).

In the original paper, as well as in Belos, the left-hand side of the eigenvalue problem (line 16) is defined as:

$$H = H_m + H_m^{-H} \begin{bmatrix} 0_{p\cdot(m-1)\times p\cdot(m-1)} & 0_{p\cdot(m-1)\times p} \\ 0_{p\times p\cdot(m-1)} & h_{m+1,m}^H h_{m+1,m} \end{bmatrix}.$$

Since our implementation of (Block) GMRES computes the QR factorization of $\overline{H}_m$ incrementally—i.e., p column(s) of Q and R are determined per iteration—we prefer to compute the following left-hand side, which is cheaper to evaluate:

$$H = H_m + QR^{-H} \begin{bmatrix} 0_{p\cdot(m-1)\times p\cdot(m-1)} & 0_{p\cdot(m-1)\times p} \\ 0_{p\times p\cdot(m-1)} & h_{m+1,m}^H h_{m+1,m} \end{bmatrix}. \tag{2}$$

Note that when $p = 1$, H is a Hessenberg matrix, and the eigenvalue problem $Hz_\lambda = \theta_\lambda z_\lambda$ can be solved with the specialized LAPACK routines `?hseqr` and `?hsein` instead of—as done in Belos—the general routine `?geev`.

The generalized eigenvalue problem (line 33) may be defined with the following matrix pair:

$$\begin{aligned} T &= G_m^H G_m, \\ W &= G_m^H \begin{bmatrix} C_k^H U_k & 0_{p\cdot k\times p\cdot(m-k)} \\ V_{m-k+1}^H U_k & I_{p\cdot(m-k+1)\times p\cdot(m-k)} \end{bmatrix}, \end{aligned} \tag{3a}$$

where the matrix G_m is defined as:

$$G_m = \begin{bmatrix} D_k & E_k \\ 0_{p\cdot(m-k+1)\times p\cdot k} & \overline{H}_{m-k} \end{bmatrix}.$$

D_k is a diagonal matrix whose entries are the $p \cdot k$ coefficients used to scale U_k (line 32). We will see in section III-C that alternative definitions of the right-hand side matrix W in eq. (3) are possible and may yield better performance.

Like in some communication-avoiding iterative methods, GCRO-DR requires the orthogonalization of $p \cdot k$ vectors at once (line 4). When it comes to subspace recycling methods or block iterative methods, Gram-Schmidt schemes are often used to perform this [37]. Belos uses by default the Iterated Modified Gram-Schmidt method, but it is also possible to switch to the TSQR method, first studied in the context of CA-GMRES [47]. In our implementation, we propose to use the CholQR method [48] since its efficiency has already been proved—once again in the context of CA-GMRES [49].

B. Non-variable linear systems

For some time-dependent PDEs, it is necessary to solve sequences of linear systems where the operator is the same throughout the sequence, and only the right-hand sides are varying. E.g., when solving the heat equation implicitly:

$$\frac{\partial u}{\partial t} - \Delta u = f, \tag{4}$$

where f is a source term, or when solving the Navier–Stokes equation using projection methods [50]. In fig. 1, the conditional statements line 3 and line 31 were not part of the original GCRO-DR method, but introduced afterwards [51]. When GCRO-DR is called with a sequence of identical linear operators $A_{i+1} = A_i$, there is indeed no need to compute the original distributed QR decomposition (lines 4–6), and it is not mandatory to update the recycled subspace U_k (line 37). The additional computations of GCRO-DR compared to GMRES after the first cycle are thus:

- the initial orthogonalization of the residual matrix (line 9) and the update of the initial guess (line 8),
- the orthogonalization w.r.t. $(I - C_k C_k^H)$ at each iteration (line 26) for generating the Arnoldi basis,
- the update of the approximate solution X_j at the end of the jth cycle involves more work (lines 28–29).

C. Variable preconditioning

Nonlinear or nondeterministic preconditioners are often needed, e.g., when using Krylov subspace methods as smoothers in multigrid preconditioners [52], [53]. As first proposed by Parks et al. [22], and as implemented in Belos, GCRO-DR cannot handle variable preconditioning. A first flexible variant of GCRO-DR was proposed by Carvalho et al. [28]. In the corresponding Technical Report[3], the authors

[3]http://www.cerfacs.fr/algor/reports/2010/TR_PA_10_10.pdf

1: $R_0 = B_i - A_i X_0$
2: **if** U_k is defined (from solving a previous system) **then**
3: **if** $A_i \neq A_{i-1}$ **then**
4: $[Q, R] =$ distributed qr$(A_i U_k)$
5: $C_k = Q$
6: $U_k = U_k R^{-1}$
7: **end if**
8: $X_1 = X_0 + U_k C_k^H R_0$
9: $R_1 = R_0 - C_k C_k^H R_0$
10: **else**
11: $[V_1, S_1] =$ distributed qr(R_0)
12: perform m steps of (Block) GMRES, thus generating V_{m+1} and $[Q, R] =$ qr$(\overline{H}_m)$
13: find Y_m such that $RY_m = Q^{-1} \begin{bmatrix} S_1 \\ 0_{p\cdot(m-1)\times p} \end{bmatrix}$
14: $X_1 = X_0 + V_m Y_m$
15: $R_1 = B_i - A_i X_1$
16: solve $Hz_\lambda = \theta_\lambda z_\lambda$ ▷ cf. eq. (2)
17: store the k eigenvectors z_λ associated to the smallest eigenvalues in magnitude in P_k
18: $[Q, R] =$ qr$(\overline{H}_m P_k)$
19: $C_k = V_{m+1} Q$
20: $U_k = V_m P_k R^{-1}$
21: **end if**
22: $j = 1$
23: **while** EPS(R_j, ε) **do**
24: $[V_k, S_k] =$ distributed qr(R_j)
25: $j \mathrel{+}= 1$
26: perform $m - k$ steps of (Block) GMRES with the linear operator $(I - C_k C_k^H) A_i$, thus generating V_{m+1-k}, $[Q, R] =$ qr$(\overline{H}_{m-k})$, and $E_k = C_k A_i V_{m-k}$
27: find Y_{m-k} such that $RY_{m-k} = Q^{-1} \begin{bmatrix} S_k \\ 0_{p\cdot(m-k-1)\times p} \end{bmatrix}$
28: $Y_k = C_k^H R_{j-1} - E_k Y_{m-k}$
29: $X_j = X_{j-1} + U_k Y_k + V_{m-k} Y_{m-k}$
30: $R_j = B_i - A_i X_j$
31: **if** $A_i \neq A_{i-1}$ **then**
32: scale the columns of U_k so that they are of unit norm
33: solve $Tz_\lambda = \theta_\lambda W z_\lambda$ ▷ cf. eq. (3)
34: store the k eigenvectors z_λ associated to the smallest eigenvalues in magnitude in P_k
35: $[Q, R] =$ qr$(\overline{H}_m P_k)$
36: $C_k = \begin{bmatrix} C_k & V_{m-k+1} \end{bmatrix} Q$
37: $U_k = \begin{bmatrix} U_k P_k & V_{m-k} P_k \end{bmatrix} R^{-1}$
38: **end if**
39: **end while**

40: **function** EPS(R, ε)
41: **for each** column r of R **do**
42: **if** $||r|| > \varepsilon$ **then**
43: **return** true
44: **return** false
45: **end function**

Fig. 1. (Block) GCRO-DR as drafted by Parks et al. [22].

propose an alternative right-hand side matrix for the generalized eigenvalue problem (line 33). Instead of defining W as in eq. (3a), they use:

$$\begin{aligned} W &= G_m^H V_{m+1}^H V_m, \\ &= G_m^H \begin{bmatrix} I_{p\cdot m \times p\cdot m} \\ 0_{p\times p\cdot m} \end{bmatrix}. \end{aligned} \tag{3b}$$

We will see why this is attractive in practice in the following paragraph.

D. Cost analysis

For conciseness, no preconditioner has been mentioned in this section. However, when one is used, part of the initialization process (lines 4–6) must be adapted so that the preconditioner must be applied to the block of p vectors U_k (resp. $A_i U_k$) when using right (resp. left) preconditioning (line 4). This remark only holds when using a non-variable preconditioner.

The memory cost and the FLOP count of (Block) GCRO-DR and its flexible variant have already been studied in some of the aforementioned papers. We want to focus here on the synchronization and communication overhead introduced by these methods in the context of large-scale distributed computing. Our implementation uses the Message Passing Interface, and, as done frequently in implementations of Krylov subspace methods, we store:

- in a distributed fashion, matching the distribution of the linear systems A_i, all variables of the size of the system, i.e., $R_j, U_k, C_k,$ and V_k. Persistent memory for the recycled vectors U_k and C_k between cycles is allocated using a singleton class.
- redundantly on each MPI process all variables of the dimension of the Krylov subspace, i.e., $\overline{H}_m, P_k, E_k,$ and Y_m.

All additional communications in GCRO-DR are reductions that scale logarithmically with the number of processes:

- the distributed QR factorizations (line 11 and line 24) require a single reduction when using the CholQR or TSQR methods, or k reductions when using the Classical Gram-Schmidt method,
- the update of the first guess (line 8) requires once again a single reduction (resp. k reductions) when using the Classical (resp. Modified) Gram-Schmidt method,
- once a subspace is recycled, each (Block) GMRES cycle (line 26) requires one additional reduction per iteration in order to orthogonalize against C_k each vector in the Arnoldi basis. Notwithstanding preconditioning that might require global communications, the number of reductions per GCRO-DR cycle is then $2(m-k)$ instead of m for GMRES. A typical value chosen for k is then $k = \frac{m}{2}$ to ensure the same number of reductions per cycle, but this is not a golden rule.
- the solution of the least square problem (line 27) must be updated with a reduction (line 28).

All other additional operations are performed redundantly on each process, using BLAS or LAPACK routines. However, the assembly of the right-hand side matrix of the generalized eigenvalue problem eq. (3) may require another reduction. Indeed, when using the original formulation of W recalled eq. (3a), there are two matrix–matrix products that can be computed simultaneously and reduced once. When using the formulation eq. (3b), there is no global communication. The best choice (in terms of number of iterations) of eigenvalue problems between eq. (3a) and eq. (3b) is problem-dependent, as observed in the Technical Paper previously cited[1].

IV. Large-scale experiments

The purpose of this section is to show 1) how recycling may improve the efficiency of two toy problems, 2) that our framework may easily interact with most existing C/C++, Python, or Fortran applications. We will be using PETSc to generate our problems and to define preconditioners, that will then be passed to our implementation of GCRO-DR using callback functions. When GMRES or GCRO-DR are used as the outer iterative methods, we use the default value of PETSc of 30 to be the maximum dimension of Krylov subspaces before the methods restart.

A. Hardware and software settings

Results were obtained on Curie, a system composed of 5,040 nodes with two eight-core Intel Sandy Bridge clocked at 2.7 GHz. The interconnect is an InfiniBand QDR full fat tree and the MPI implementation exploited was BullxMPI version 1.2.9.2. All binaries and shared libraries were compiled with Intel compilers and Math Kernel Library support (for dense linear algebra computations). The latest available release of PETSc was used (version 3.7.3).

B. Poisson's equation

This PDE may be used to model many physical phenomena, for example in computational fluid dynamics [54]. It can also be seen as the steady-state heat equation, cf. eq. (4):

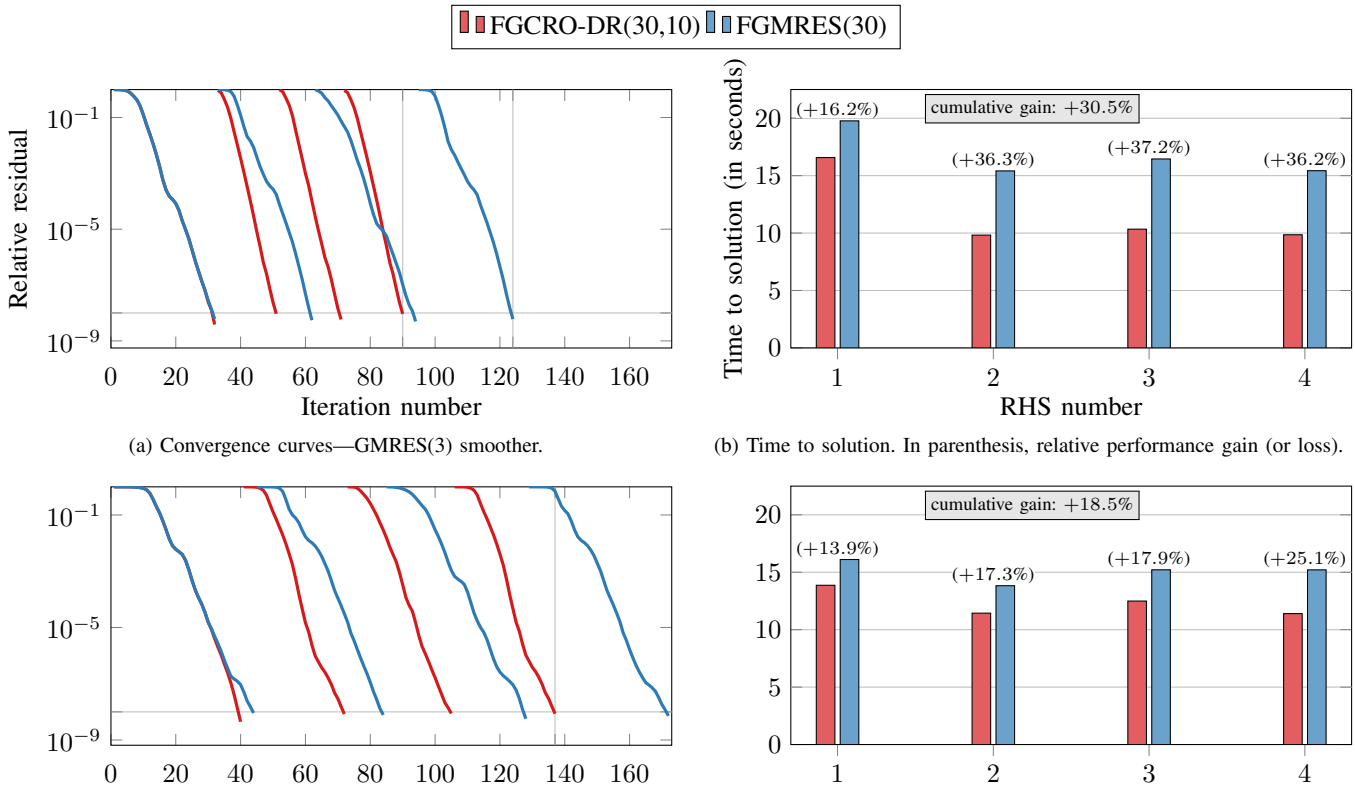
$$-\Delta u = f.$$

Example number 32 of the PETSc distribution[4] discretizes this continuous problem using a simple two-dimensional Cartesian grid and a standard five-point stencil. This yields a linear system A. We slightly modified the example to generate four successive right-hand sides:

$$f_i(x,y) = \frac{1}{\nu_i} e^{-\frac{(1-x)^2}{\nu_i}} e^{-\frac{(1-y)^2}{\nu_i}},$$

where $\{\nu_i\} = \{0.1, 10, 0.001, 100\}$. The goal of the script is now to solve the sequence of four linear systems one after another, like one would have to do when solving a time-dependent problem. We will be using the Geometric Algebraic Multigrid preconditioner [24] (GAMG). It is an implementation of the smoothed aggregation multigrid method, and is shipped by default with PETSc. To make the multigrid cycles nonlinear, three iterations of GMRES are used as a smoother.

[4] http://www.mcs.anl.gov/petsc/petsc-3.7.3/src/ksp/ksp/examples/tutorials/ex32.c.html



(a) Convergence curves—GMRES(3) smoother.

(b) Time to solution. In parenthesis, relative performance gain (or loss).

(c) Convergence curves—GMRES(1) smoother, higher threshold for GAMG.

(d) Time to solution. In parenthesis, relative performance gain (or loss).

Fig. 2. Performance analysis of FGCRO-DR against FGMRES for solving Poisson's equation discretized with 283 million unknowns with four varying RHSs on 8,192 processors.

It is likely that there are better setup parameters, but we don't want to focus on the performance of the preconditioner, and rather on the performance of the iterative methods (FGMRES vs. FGCRO-DR). The command line used to define the linear systems, setup the preconditioners and the iterative method is:

```
mpirun -np 8192 ./ex32 -da_grid_x 4210 -da_refine 2
  -da_grid_y 4210 -ksp_rtol 1.0e-8 -pc_type gamg
  -pc_gamg_threshold 0.0725 -pc_gamg_square_graph 2
  -ksp_type fgmres -mg_levels_ksp_type gmres
  -mg_levels_ksp_max_it 3
```

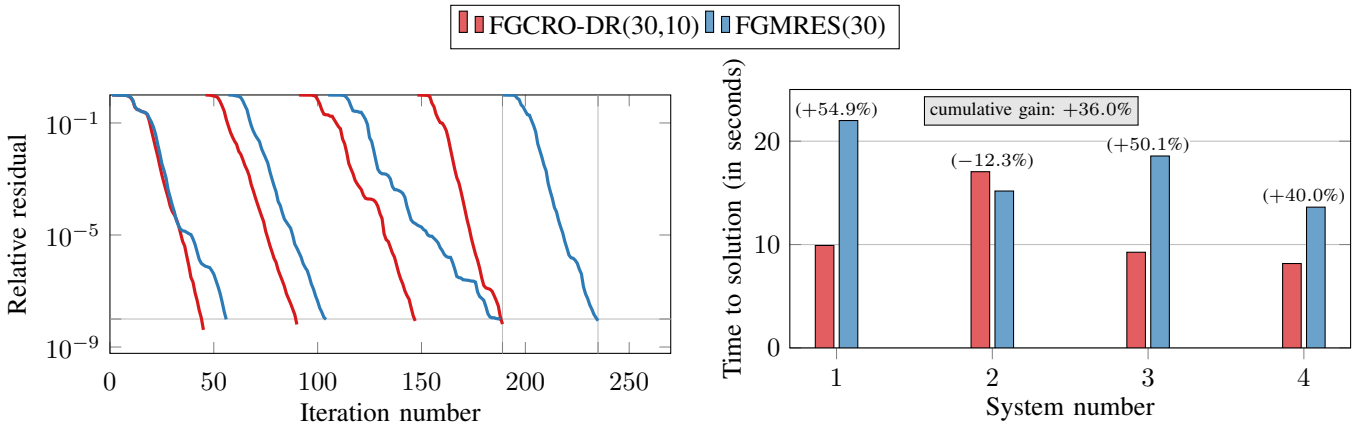
This generates a linear system of 280 million unknowns, for which the preconditioner is setup in 160 seconds. The matrix as well as the preconditioner are only assembled for the first right-hand side. They can be reused as is for successive solves. We compare the Flexible GMRES method implemented in PETSc and our implementation of Flexible GCRO-DR, using a recycled subspace of dimension 10. This dimension was chosen after some preliminary experiments, but it can be set between 1 and $m-1$. In fig. 2a, the convergence curves of both methods are displayed. In fig. 2b, the time to solution for each RHS is displayed. Overall, FGMRES (resp. FGCRO-DR) performs 124 (resp. 90) iterations. There is almost no restart in the previous experiment, thanks to the good numerical properties of the preconditioner. We will now use a slightly cheaper preconditioner, which induces a lower setup cost at the price of more iterations. The PETSc option `-pc_gamg_threshold`—a parameter to select edges in aggregation graphs—is adjusted accordingly:

```
mpirun -np 8192 ./ex32 -da_grid_x 4210 -da_refine 2
  -da_grid_y 4210 -ksp_rtol 1.0e-8 -pc_type gamg
  -pc_gamg_threshold 0.076 -pc_gamg_square_graph 2
  -ksp_type fgmres -mg_levels_ksp_type gmres
  -mg_levels_ksp_max_it 1
```

In fig. 2c, the convergence curves of both methods are displayed. In fig. 2d, the time to solution for each RHS is displayed. Overall, FGMRES (resp. FGCRO-DR) performs 172 (resp. 137) iterations. All these numbers are application-dependent, but it is clear that when using costly preconditioners, any decrease in number of iterations is worthwhile. An important observation that can be made looking at figs. 2b and 2d is that the cumulative solve time of FGMRES with the more robust but costlier preconditioner (blue bars in fig. 2b is greater than the cumulative time of FGCRO-DR with the less robust but cheaper preconditioner (red bars fig. 2d). Thus, recycling can also be used to spend less time in assembling highly robust preconditioners by relaxing setup parameters—e.g., threshold criterion for multigrid preconditioners, overlap width for domain decomposition methods, or level of fill-in for incomplete factorizations.

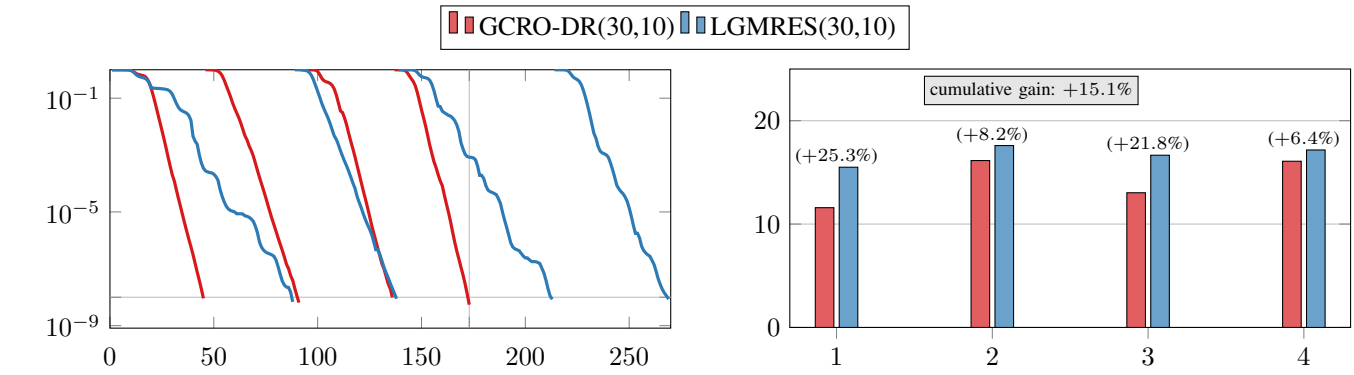
C. The system of linear elasticity

This PDE is used in computational solid mechanics, for example to model small deformations of a rigid body. In the



(a) Convergence curves—flexible preconditioning, CG(4) smoother.

(b) Time to solution. In parenthesis, relative performance gain (or loss).



(c) Convergence curves—right preconditioning, Chebyshev smoother.

(d) Time to solution. In parenthesis, relative performance gain (or loss).

Fig. 3. Performance analysis of GCRO-DR against FGMRES and Loose GMRES for solving four varying 3D linear elasticity systems of 192 million unknowns on 8,000 processors.

context of shape optimization, it is often necessary to solve multiple, slowly varying systems, in order to adapt a shape so that it minimizes a given shape-dependent cost function (e.g., the compliance of a structure) [55]. The displacement formulation of the steady-state system of linear elasticity is:

$$-\nabla \cdot \boldsymbol{\sigma} = \mathbf{f},$$

where $\boldsymbol{\sigma}$ is the stress tensor, and $\mathbf{f}$ represents body forces. Example number 56 of the PETSc distribution[5] discretizes this PDE on the unit cube with $\mathbb{Q}_1$ finite elements. To generate a sequence of four varying systems—indexed by $i \in [\![1, \ldots, 4]\!]$, we use a set of five parameters:

$$\{s_i\} = \{30, 0.1, 20, 10\} \qquad \{r_i\} = \{0.5, 0.45, 0.4, 0.35\}$$
$$\{x_i\} = \{0.5, 0.4, 0.4, 0.4\} \qquad \{y_i\} = \{0.5, 0.5, 0.4, 0.4\}$$
$$\{z_i\} = \{0.5, 0.45, 0.4, 0.35\}$$

to define parametrically a small, moving, spherical inclusion:

$$\forall (x, y, z) \in [0; 1]^3, (x - x_i)^2 + (y - y_i)^2 + (z - z_i)^2 < r_i^2.$$

In this small inclusion, the material coefficient E_i is defined as $E_i = \frac{E}{s_i}$, E is the Young modulus everywhere but in the inclusion. Once again, we use a multigrid preconditioner, equipped with the near-nullspace of the operators made of six rigid body modes, to solve the linear systems efficiently. The command line used to define the linear systems, setup the preconditioners and the iterative method is:

```
mpirun -np 8000 ./ex56 -ne 399 -ksp_rtol 1.0e-8
  -ksp_type fgmres -pc_type gamg
  -mg_levels_ksp_type cg -mg_levels_ksp_max_it 4
```

This generates four linear systems of 283 million unknowns, for which the preconditioner must be setup for each different matrix—in average, this takes 50 seconds. We choose once again on purpose the smoother to be four iterations of CG to make the multigrid cycles nonlinear. It is unlikely to be the most efficient smoother, but it makes the use of the flexible variant of GCRO-DR (which is not implemented inside Belos) mandatory. In fig. 3a, the convergence curves of FGMRES and FGCRO-DR methods are displayed. In fig. 3b, the time to solution for each RHS is displayed. Unlike in the previous experiment, an eigenvalue problem must be solved at each restart (lines 31–38 in fig. 1). Likewise, it is costlier to update the initial guess (line 8) since a distributed QR factorization must be computed first. Overall, FGMRES (resp. FGCRO-DR) performs 235 (resp. 189) iterations.

We now propose a comparison of our implementation of

[5]http://www.mcs.anl.gov/petsc/petsc-3.7.3/src/ksp/ksp/examples/tutorials/ex56.c.html

GCRO-DR and Loose GMRES as implemented in PETSc. Unfortunately, the flexible variant of LGMRES is not in PETSc, so we precondition the systems on the right instead:

```
mpirun -np 8000 ./ex56 -ne 399 -ksp_rtol 1.0e-8
  -ksp_type lgmres -ksp_pc_side right -pc_type gamg
  -ksp_lgmres_augment 10
```

This time, the multigrid cycles are linear since the default smoother used by PETSc is the Chebyshev iterative method. There is no need for the flexible variant of GCRO-DR. In fig. 3c, the convergence curves of LGMRES and GCRO-DR are displayed. In fig. 3d, the time to solution for each RHS is displayed. Clearly, the better numerical properties of GCRO-DR over LGMRES plays a huge role here, since GCRO-DR needs 96 fewer iterations to converge (269 LGMRES iterations vs. 173 GCRO-DR iterations).

V. Large-scale solver for Maxwell's equation

Maxwell's equation describes the propagation of electromagnetic waves. Here, we consider a nonmagnetic linear isotropic medium of dielectric permittivity ε and conductivity σ. Assuming that the fields behave periodically with respect to time, for example in the case of a time-periodic incident signal at angular frequency ω, the complex amplitude $\mathbf{E}$ of the associated electric field $\mathcal{E}(\mathbf{x},\mathbf{t}) = \Re(\mathbf{E}(\mathbf{x})e^{-\mathrm{i}\omega\mathbf{t}})$ is solution of the following second order time-harmonic Maxwell equation:

$$\nabla \times (\nabla \times \mathbf{E}) - \mu_0\Big(\omega^2\varepsilon + \mathrm{i}\omega\sigma\Big)\mathbf{E} = 0, \tag{5}$$

where μ_0 is the permeability of free space.

High-order curl-conforming finite elements of Nédélec type [56] are now well-established in computational electromagnetism, thanks to their accuracy and low numerical dispersion and dissipation errors [57]. However, linear systems arising from such discretizations are ill-conditioned [58]. This, combined with the fact that the underlying PDE is indefinite, highlights the need for a robust and efficient preconditioner. Indeed, we show in fig. 4 that standard preconditioners such as the Additive Schwarz Method (ASM) or GAMG cannot solve the linear system arising from our application described in the next paragraph as rapidly as our preconditioner defined eq. (6). Moreover, eq. (5) is solved with many right-hand sides in our application so we will know investigate the efficiency of block methods in this section. Recycling techniques presented in the previous paragraph will be also used to combine the advantages of both approaches.

A. Description of the application

Some of the computational methods described in this paper have been implemented in the context of an application in microwave imaging as part of the ANR project MEDIMAX, which aims at developing a robust and accurate inversion tool associated with the direct electromagnetic problem modeled by Maxwell's equation in the frequency domain in highly heterogeneous media. The targeted application is medical imaging, and in particular brain imaging for stroke detection and diagnosis. By exposing head tissues to low-level microwave incident field and capturing the scattered field by an array of antennas, the estimation of the dielectric properties of the brain tissues—ε and σ in eq. (5)—can be approximated by solving an inverse problem and a diagnosis can be inferred.

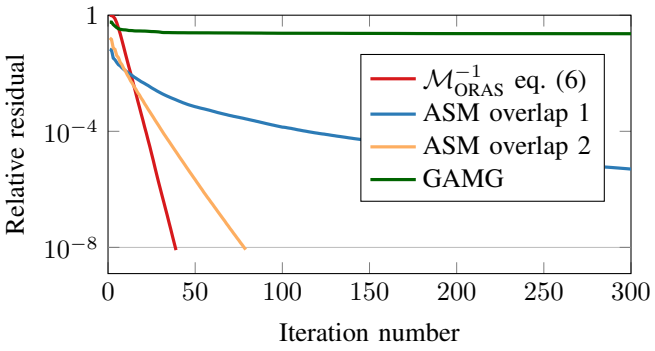


Fig. 4. GMRES convergence curves of some standard preconditioners, as well as our own described in section V-A, for solving Maxwell's equation discretized with 50 million double-precision complex unknowns on 512 MPI processes.

Simulation results presented in this work have been obtained on the imaging system prototype developed by EMTensor GmbH [59]. The system is composed of 160 antennas: 5 rings of 32 open ceramic-loaded rectangular waveguides around a cylindrical metallic chamber, depicted in fig. 5a. Each antenna can act as transmitter and receiver.

The object to be imaged is introduced into the imaging chamber. Each of the 160 antennas alternatively transmits a signal.The retrieved data then consist in the reflection and transmission coefficients measured by the 160 receiving antennas which will be used as input for the inverse problem. Each transmitting antenna corresponds to a different incident signal and thus to a different right-hand side in the discretized system. In section V-C, some numerical results for 32 RHSs, corresponding to one ring of transmitting antennas, are presented.

Let us now introduce our domain decomposition preconditioner. First, the mesh $\mathcal{T}$ in fig. 5b corresponding to the imaging system in fig. 5a is generated using 18 million tetrahedra. It is then partitioned in N non-overlapping meshes $\{\mathcal{T}_i\}_{1\leqslant i\leqslant N}$ using standard graph partitioners, cf. fig. 5c. If δ is a positive integer, the overlapping decomposition $\{\mathcal{T}_i^\delta\}_{1\leqslant i\leqslant N}$ is defined recursively as follows: $\mathcal{T}_i^\delta$ is obtained by including all elements of $\mathcal{T}_i^{\delta-1}$ plus all adjacent elements of $\mathcal{T}_i^{\delta-1}$. For $\delta = 0$, $\mathcal{T}_i^\delta = \mathcal{T}_i$. Let V be the finite element space defined on $\mathcal{T}$, and $\{V_i^\delta\}_{1\leqslant i\leqslant N}$, the local finite element spaces defined on $\{\mathcal{T}_i^\delta\}_{1\leqslant i\leqslant N}$. Now consider the restrictions $\{R_i\}_{1\leqslant i\leqslant N}$ from V to $\{V_i^\delta\}_{1\leqslant i\leqslant N}$, and a local partition of unity $\{D_i\}_{1\leqslant i\leqslant N}$ such that:

$$\sum_{j=1}^{N} R_j^T D_j R_j = I_{n\times n}.$$

Algebraically speaking, if n is the global number of unknowns and $\{n_i\}_{1\leqslant i\leqslant N}$ are the numbers of degrees of freedom in each

local finite element spaces, then R_i is a Boolean matrix of size $n_i \times n$, and D_i is a diagonal matrix of size $n_i \times n_i$, for all $1 \leqslant i \leqslant N$.
Using the partition of unity, one can define the following one-level preconditioner as an extension of the Restricted Additive Schwarz method proposed by Cai and Sarkis [60]:

$$\mathcal{M}^{-1}_{\text{ORAS}} = \sum_{i=1}^{N} R_i^T D_i B_i^{-1} R_i, \tag{6}$$

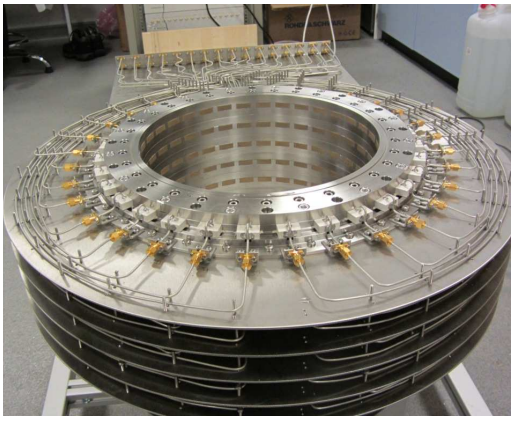
where the $\{B_i\}_{1 \leqslant i \leqslant N}$ are local operators that resemble the submatrices $\left\{R_i A R_i^T\right\}_{1 \leqslant i \leqslant N}$, but with more efficient transmission conditions between subdomains, e.g. [61]. What is important to notice here, is that when a direct solver is used to compute the action of B_i^{-1} on multiple vectors, it can be done in a single forward elimination and backward substitution as long as the vectors are stored contiguously. In the next section, it will be shown that this can increase the performance of a direct solver tremendously, cf. fig. 6.
All operators related to the domain decomposition can be easily generated using finite element Domain-Specific Languages. We will be using FreeFem++ [62] since it has already been proven that it can enable large-scale simulations using overlapping Schwarz methods [63], but our framework interacts with other DSLs such as Feel++ [64].

B. Block iterative methods

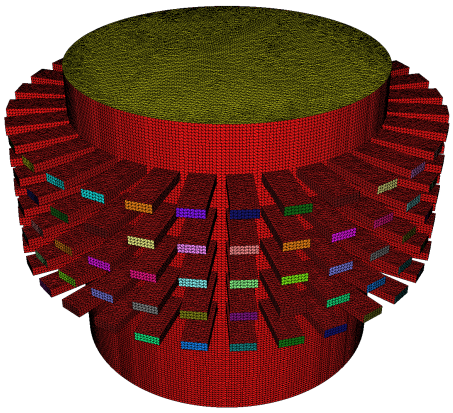
1) Pseudo-block and block methods: First introduced in the thesis of Langou [18], the notion of pseudo-block methods was formalized in the Belos package from Trilinos in 2007. The idea behind pseudo-block iterative methods is to fuse multiple operations to achieve higher arithmetic intensity, or to decrease the number of global synchronizations. For example, if one needs to perform m GMRES iterations to reach convergence for each p RHSs, a naive algorithm would require $m \cdot p$ dot products for evaluating the norm of each candidate vector of the Arnoldi basis. If the p GMRES cycles are fused together, the required number of dot products is lowered to m instead. Pseudo-block methods are designed to leverage the computational power of multicore architectures, while trying to mitigate the overhead of global synchronization by exchanging more data less often. In contrast, block methods are mathematical reformulations of standard iterative methods to handle multiple RHSs. They tend to converge faster at the cost of more computations and greater volume of data exchange.

2) Cost analysis: Our implementation can handle right, left, or variable preconditioning, for (pseudo-)Block GMRES and (pseudo-)Block GCRO-DR. The algorithm fig. 1 was written such that it can be used for both standard and Block GCRO-DR. In terms of memory, pseudo-block methods require p times more storage. For block methods, Hessenberg matrices are $p \times p$ bigger, and Arnoldi basis and recycled subspaces are p times thicker, cf. section III-A for the notations. This high memory cost is the reason why the restart parameter for BGMRES and BGCRO-DR is usually lesser than for standard methods. A more thorough analysis is available for the interested reader [65]. In terms of arithmetic operations and messages, the most demanding kernels are, as in any iterative method, sparse matrix–dense matrix products, i.e. $Y = AX$ and preconditioner–dense matrix operations, i.e. $V = M^{-1}Y$. Traditionally, the matrix A is distributed on the global MPI communicator, and computing sparse matrix–vector products requires peer-to-peer communications. It is possible to extend this communication pattern to the case of sparse matrix–dense matrix products as long as the MPI buffers are p times bigger. The same goes for preconditioner–dense matrix operations. Most importantly, those two kernels are usually based on a combination of MPI data exchanges and local work. The efficiency of the local kernels usually scales fairly well with an increasing number of RHSs, because it means a higher arithmetic intensity. This especially applies to standard assembled sparse matrix operations which are almost all memory bound.

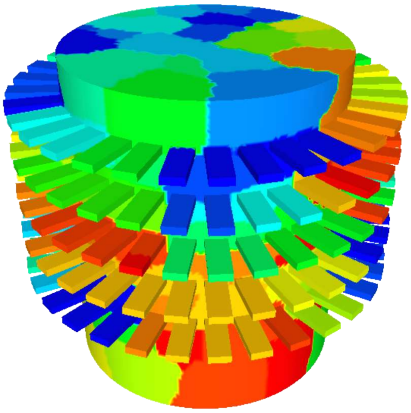
3) Scalability of a direct solver with multiple right-hand sides: As already mentioned, a domain decomposition



(a) Actual system.

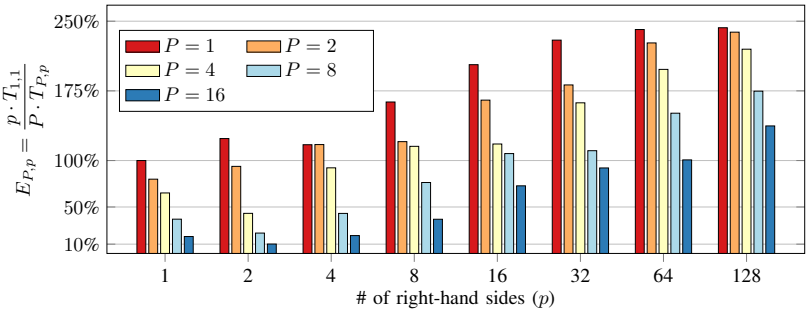


(b) Corresponding mesh.



(c) Decomposition into 128 subdomains.

Fig. 5. Imaging chamber of EMTensor (no copyright infringement intended).



(a) Efficiency relative to the time to solution with one right-hand side and one thread.

		# of right-hand sides (p)							
		1	2	4	8	16	32	64	128
P	1	1.58	2.55	5.39	7.74	12.42	21.99	41.89	83.13
	2	0.99	1.68	2.69	5.24	7.65	13.92	22.28	42.39
	4	0.61	1.83	1.71	2.74	5.36	7.79	12.74	22.96
	8	0.53	1.80	1.83	2.07	2.94	5.71	8.36	14.45
	16	0.54	1.95	2.05	2.14	2.17	3.43	6.27	9.2

(b) Time of the solution phase (in seconds) $T_{P,p}$.

Fig. 6. Scalability analysis of PARDISO for solving the same system with varying number of threads and RHSs.

preconditioner will ensure fast convergence of the iterative solver. Since a direct method is used in each subdomain, it is interesting to understand how it scales when performing forward eliminations and backward substitutions with multiple RHSs. In this small numerical experiment, the three dimensional Maxwell's equation is solved in the unit cube with high-order finite elements [58], [66]. The discrete linear system is made of approximately 300k unknowns, with roughly 83 nonzero double-precision complex entries per row. Since the matrix is symmetric, only its upper triangular part is stored. Each RHS is generated randomly. The direct solver used is PARDISO [67], [68] from the MKL, but other direct solvers might yield better results when solving systems with multiple RHSs [69]. Once the matrix is factorized, we solve the linear systems with 2^p, $p \in [\![0, \dots, 7]\!]$, RHSs. Figure 6 gathers these results, as well as a scalability analysis of PARDISO when solving these multiple systems with 2^P, $P \in [\![0, \dots, 4]\!]$, threads. Timings of fig. 6b are averages of two consecutive runs. Curie, the machine used in all of our numerical experiments, is made of nodes with two eight-core sockets. This means that all these runs are performed on a single node. In fig. 6a, we plot the efficiency defined as:

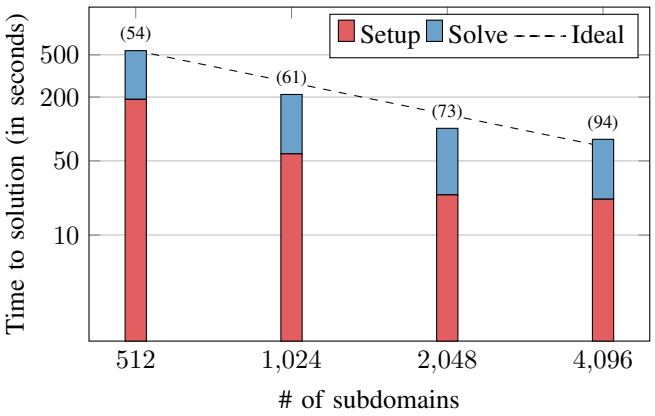
$$E_{P,p} = \frac{p \cdot T_{1,1}}{P \cdot T_{P,p}}.$$

It is interesting to note that even with only one thread ($P = 1$), we have a nice superlinear efficiency. This is likely due to the use of BLAS 3 [70] instead of BLAS 2 routines, and thus, better arithmetic intensity. When using multiple threads, having multiple RHSs is sometimes the only way to achieve reasonable performance. When $P = 16$, with two RHSs ($p = 2$), PARDISO has an abysmal efficiency of 10%. However, as the number of RHSs increases, it is possible once again to reach a regime where the efficiency is superlinear (in this case, $p = 64$ is the tipping point).

We have thus displayed experimentally one of the advantages of (pseudo-)block methods when direct solvers are used in the definition of the global preconditioner. By increasing the workload, it is possible to achieve higher efficiency. This remark is also valid for sparse matrix–dense matrix products, and similar results are obtained when benchmarking, for example, the `?csrmm` routine from Intel MKL [71].

C. Scaling analysis

To assess the efficiency of our preconditioner, we will first perform a strong scaling analysis. We consider in this test case that the imaging chamber of fig. 5a is filled with an homogeneous dissipative matching solution, suited for brain imaging applications. Given a global mesh as depicted in fig. 5b, we increase the number of MPI processes to solve the linear system of 119 million double-precision complex unknowns yielded by the discretization of Maxwell's equation using high-order edge elements of degree 2. As seen in fig. 4, even when the problem is relatively small, standard preconditioners fail to converge or converge slower than $\mathcal{M}^{-1}_{\text{ORAS}}$. Because we use complex-valued scalars, *hypre*, and in particular its Maxwell solver AMS [72], cannot be used. MueLu [73] from



(a) Time to solution. Each color represents the fraction of the total time spent in the respective tasks. In parenthesis, number of iterations.

N	Setup	Solve	# of iterations	Speedup
512	456.0	91.8	54	–
1,024	160.7	51.0	61	2.6
2,048	69.7	31.7	73	5.4
4,096	56.2	23.7	94	6.9

(b) Timings (in seconds) of the setup and the solution phases.

Fig. 7. Strong scaling analysis of the Maxwell solver for a system of 119 million double-precision complex unknowns.

Trilinos can only solve Maxwell's equation in eddy current formulation[6], and it is not clear how it handles high-order edge elements.

Figure 7a is a plot of the time to solution, including both the setup and the solution phases, for solving the linear system on 512 up to 4,096 subdomains. The setup time does not account for the mesh partitioning and the assembly of the finite element matrices. We map one subdomain per MPI process, and use one thread per MPI process. The global unstructured mesh is partitioned using SCOTCH [74], we use a geometric overlap of two elements ($\delta = 2$), the local solver is PARDISO from Intel MKL, and the iterative method is the Full GMRES which is stopped once the relative unpreconditioned residual is lower than 10^{-8}. The overall speedup is almost optimal, with a ratio of almost 7 between the time to solution using 512 and 4,096 subdomains, cf. fig. 7b for the exact figures. Since simple, yet efficient, optimized boundary conditions are used, the number of iterations slightly increases with the number of MPI processes. This explains why the fraction of the total time spent in the solve phase for the run on 4,096 subdomains (30%) is greater than for the run on 512 subdomains (17%).

We now report results with our solver for 32 RHSs on a more difficult test case, in order to demonstrate the efficiency of recycling and block methods. A non-dissipative plastic cylinder of diameter $12\,\mathrm{cm}$ is immersed in the imaging chamber and surrounded by matching liquid. The 32 RHSs correspond to the second ring (from the top) of 32 transmitting antennas. The corresponding linear system has 89 million unknowns. The results are obtained on 4,096 subdomains, this time with one subdomain and two OpenMP threads per MPI process—so we use a total of 8,192 cores on Curie. We propose eight alternatives to solve the system for all RHSs:

1) 32 consecutive solves with GMRES(50) (reference),
2) 32 consecutive solves with GCRO-DR(50, 10),
3) 1 solve with pseudo-BGMRES(50) and 32 RHSs,
4) 1 solve with BGMRES(50) and 32 RHSs,
5) 4 consecutive solves with pseudo-BGCRO-DR(50, 10) and 8 RHSs,
6) 1 solve with pseudo-BGCRO-DR(50, 10) and 32 RHSs,
7) 4 consecutive solves with BGCRO-DR(50, 10) and 8 RHSs,
8) 1 solve with BGCRO-DR(50, 10) and 32 RHSs.

Iterative methods are stopped once the relative unpreconditioned residual of each RHS is lower than 10^{-8}. In all cases, the system matrix is assembled once, as well as the preconditioner defined eq. (6). For each alternative, the preconditioner is setup only once. This step lasts $43.2\,\mathrm{s}$. In fig. 8, the speedup with respect to the naive approach—alternative 1)—is displayed. Setting up the preconditioner represents between 1.3% and 6% of the total time to solution ($= 43.2\,\mathrm{s}$ + the second column of fig. 8). Thus, it is of paramount importance to increase the efficiency of the solution phase. Variants of (pseudo-)block methods guarantee at least a speedup of nearly 2. We notice that the best approach in terms of computation time is 7), which is a combination of recycling and block methods. It can be used to decrease the overall time to solution by 450%. Numerically, the best approach is 8), which divides the number of iterations by an astonishing factor of 158. However, the cost of working on the complete block of 32 RHSs becomes quite high, and it is best to mix recycling techniques and smaller blocks of 8 RHSs. We currently do not use block size reduction inside block methods [19], [21], [75], but we perform rank-revealing CholQR (line 11 and line 24 in fig. 1) for detecting breakdowns at each restart and residuals appear to be far from being colinear in our application. It is not clear to us if the cost of performing deflation at each iteration would be beneficial, since we already perform a rather low number of iterations with block methods—alternatives 4), 7), and 8).

Alternative	p	Solve	# of it.	per RHS	Speedup
1)	1	3,078.4	20,068	627	–
2)	1	1,836.9	10,701	334	1.7
3)	32	1,577.9	653	–	2.0
4)	32	724.8	158	–	4.2
5)	8	1,357.8	1,508	377	2.3
6)	32	1,376.1	469	–	2.2
7)	8	677.6	524	131	4.5
8)	32	992.3	127	–	3.1

Fig. 8. Timings (in seconds) of the solution phase, and speedups relative to alternative 1). The number of iterations per RHS is an average over all $\frac{32}{p}$ solves (and thus not reported when $p = 32$).

D. Perspectives

The application presented here is a good illustration of the efficiency of recycling and block methods in speeding up computations arising in wave scattering and wave propagation problems, which often involve multiple right-hand sides corresponding to different angles of incident waves or different locations of excitation sources. Recycling techniques can also be applied in optimization problems, which generally consist in solving a sequence of slowly-varying linear systems, where the coefficient matrix depends on the choice of parameters. For such problems, recycling strategies can help in reducing significantly the total number of iterations over all linear systems. Incorporating these techniques in the development of an efficient inversion algorithm in the context of our application in brain imaging described in this paper is the focus of an ongoing work [76].

VI. Conclusion

In this paper, we have presented various applications using recycling strategies or block methods. Large-scale experiments were obtained on 8,192 cores, using either our own application built on top of a finite element domain-specific language, or a well-established linear algebra backend: PETSc. Both of the approaches rely on our open-source framework for the solution phase. It can currently handle right, left, or

[6]which assumes that $\mu_0\omega^2\varepsilon\mathbf{E} = 0$ in eq. (5)

variable preconditioning, for (pseudo-)Block GMRES and (pseudo-)Block GCRO-DR, and may be called from C/C++, Python, or Fortran. We have shown experimentally that recycling is an elegant way to decrease the time to solution for solving sequence of linear systems with millions of unknowns. This is especially true when only the right-hand sides are changing in the sequence. We have also studied the behavior of linear solvers when systems are made of multiple right-hand sides available simultaneously in the context of medical imaging. Using a scalable Maxwell solver based on optimized Schwarz methods, we have shown that using block methods can greatly increase the efficiency of both direct and hybrid direct–iterative solvers. We hope that this work will motivate developers of linear algebra backends such as PETSc or *hypre* to handle linear systems with multiple right-hand sides.

Acknowledgments The authors would like to thank F. Nataf for the discussions about domain decomposition methods for Maxwell's equation and X. Vasseur for the discussions about recycling strategies and block methods. This work was granted access to the HPC resources of TGCC@CEA under the allocations 2016-067519 and 2016-067730 made by GENCI. This work has been supported in part by ANR through project MEDIMAX, ANR-13-MONU-0012. The first author was partially funded by the French Association of Mechanics (AFM) for this work.

REFERENCES

[1] H. Sundar, G. Biros, C. Burstedde, J. Rudi, O. Ghattas, and G. Stadler, "Parallel Geometric-Algebraic Multigrid on Unstructured Forests of Octrees," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC12*. IEEE Computer Society Press, 2012.

[2] J. Rudi, A. C. I. Malossi, T. Isaac, G. Stadler, M. Gurnis, P. W. Staar, Y. Ineichen, C. Bekas, A. Curioni, and O. Ghattas, "An Extreme-Scale Implicit Solver for Complex PDEs: Highly Heterogeneous Flow in Earth's Mantle," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC15*. ACM, 2015.

[3] P. Jolivet, F. Hecht, F. Nataf, and C. Prud'homme, "Scalable Domain Decomposition Preconditioners for Heterogeneous Elliptic Problems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC13*. ACM, 2013.

[4] A. Klawonn and O. Rheinbach, "Highly scalable parallel domain decomposition methods with an application to biomechanics," *ZAMM–Journal of Applied Mathematics and Mechanics*, vol. 90, no. 1, pp. 5–32, 2010.

[5] U. M. Yang, *Parallel Algebraic Multigrid Methods—High Performance Preconditioners*. Springer, 2006.

[6] V. Dolean, P. Jolivet, and F. Nataf, *An Introduction to Domain Decomposition Methods: Algorithms, Theory, and Parallel Implementation*. SIAM, 2015, vol. 144.

[7] Y. Saad and M. H. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, vol. 7, no. 3, pp. 856–869, 1986.

[8] M. R. Hestenes and E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409–436, 1952.

[9] P. Ghysels, T. J. Ashby, K. Meerbergen, and W. Vanroose, "Hiding Global Communication Latency in the GMRES Algorithm on Massively Parallel Machines," *SIAM Journal on Scientific Computing*, vol. 35, no. 1, pp. C48–C71, 2013.

[10] P. Sanan, S. M. Schnepp, and D. May, "Pipelined, Flexible Krylov Subspace Methods," *SIAM Journal on Scientific Computing*, 2016.

[11] A. T. Chronopoulos, "*s*-Step Iterative Methods for (Non)Symmetric (In)Definite Linear Systems," *SIAM Journal on Numerical Analysis*, vol. 28, no. 6, pp. 1776–1789, 1991.

[12] M. Hoemmen, "Communication-avoiding Krylov subspace methods," Ph.D. dissertation, University of California, Berkeley, 2010.

[13] P. Soudais, "Iterative Solution of a 3-D Scattering Problem from Arbitrary Shaped Multidielectric and Multiconducting Bodies," *IEEE Transactions on Antennas and Propagation*, vol. 42, no. 7, pp. 954–959, 1994.

[14] A. H. Baker, J. M. Dennis, and E. R. Jessup, "On Improving Linear Solver Performance: A Block Variant of GMRES," *SIAM Journal on Scientific Computing*, vol. 27, no. 5, pp. 1608–1626, 2006.

[15] L. Giraud, S. Gratton, and E. Martin, "Incremental spectral preconditioners for sequences of linear systems," *Applied Numerical Mathematics*, vol. 57, no. 11, pp. 1164–1180, 2007.

[16] P. Gosselet, C. Rey, and J. Pebrel, "Total and selective reuse of Krylov subspaces for the resolution of sequences of nonlinear structural problems," *International Journal for Numerical Methods in Engineering*, vol. 94, no. 1, pp. 60–83, 2013.

[17] V. Simoncini and E. Gallopoulos, "Convergence properties of block GMRES and matrix polynomials," *Linear Algebra and its Applications*, vol. 247, pp. 97–119, 1996.

[18] J. Langou, "Solving large linear systems with multiple right-hand sides," Ph.D. dissertation, INSA de Toulouse, 2003.

[19] M. Robbé and M. Sadkane, "Exact and inexact breakdowns in the block GMRES method," *Linear Algebra and its Applications*, vol. 419, no. 1, pp. 265–285, 2006.

[20] R. B. Morgan, "Restarted block-GMRES with deflation of eigenvalues," *Applied Numerical Mathematics*, vol. 54, no. 2, pp. 222–236, 2005.

[21] E. Agullo, L. Giraud, and Y.-F. Jing, "Block GMRES method with inexact breakdowns and deflated restarting," *SIAM Journal on Matrix Analysis and Applications*, vol. 35, no. 4, pp. 1625–1651, 2014.

[22] M. L. Parks, E. de Sturler, G. Mackey, D. D. Johnson, and S. Maiti, "Recycling Krylov Subspaces for Sequences of Linear Systems," *SIAM Journal on Scientific Computing*, vol. 28, no. 5, pp. 1651–1674, 2006.

[23] 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, L. C. McInnes, K. Rupp, B. F. Smith, S. Zampini, and H. Zhang, "PETSc web page," http://www.mcs.anl.gov/petsc, 2016.

[24] M. F. Adams, H. H. Bayraktar, T. M. Keaveny, and P. Papadopoulos, "Ultrascalable Implicit Finite Element Analyses in Solid Mechanics With Over a Half a Billion Degrees of Freedom," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC04*. IEEE Computer Society, 2004.

[25] M. J. Gander, "Optimized Schwarz Methods," *SIAM Journal on Numerical Analysis*, vol. 44, no. 2, pp. 699–731, 2006.

[26] C. Japhet and F. Nataf, "The best interface conditions for domain decomposition methods: absorbing boundary conditions," *Absorbing Boundaries and Layers, Domain Decomposition Methods: Applications to Large Scale Computers*, pp. 348–373, 2001.

[27] J. E. Hicken and D. W. Zingg, "A Simplified and Flexible Variant of GCROT for Solving Nonsymmetric Linear Systems," *SIAM Journal on Scientific Computing*, vol. 32, no. 3, pp. 1672–1694, 2010.

[28] L. M. Carvalho, S. Gratton, R. Lago, and X. Vasseur, "A Flexible Generalized Conjugate Residual Method With Inner Orthogonalization and Deflated Restarting," *SIAM Journal on Matrix Analysis and Applications*, vol. 32, no. 4, pp. 1212–1235, 2011.

[29] E. de Sturler, "Nested Krylov methods based on GCR," *Journal of Computational and Applied Mathematics*, vol. 67, no. 1, pp. 15–41, 1996.

[30] L. Giraud, S. Gratton, X. Pinel, and X. Vasseur, "Flexible GMRES with Deflated Restarting," *SIAM Journal on Scientific Computing*, vol. 32, no. 4, pp. 1858–1878, 2010.

[31] R. B. Morgan, "GMRES with Deflated Restarting," *SIAM Journal on Scientific Computing*, vol. 24, no. 1, pp. 20–37, 2002.

[32] D. P. O'Leary, "The Block Conjugate Gradient Algorithm and Related Methods," *Linear Algebra and its Applications*, vol. 29, pp. 293–322, 1980.

[33] B. Vital, "Étude de quelques méthodes de résolution de problemes linéaires de grande taille sur multiprocesseur," Ph.D. dissertation, Université Rennes 1, 1990.

[34] D. Darnell, R. B. Morgan, and W. Wilcox, "Deflated GMRES for systems with multiple shifts and multiple right-hand sides," *Linear Algebra and its Applications*, vol. 429, no. 10, pp. 2415–2434, 2008.

[35] J. Meng, H.-B. Li, and Y.-F. Jing, "A new deflated block GCROT(m, k) method for the solution of linear systems with multiple right-hand sides," *Journal of Computational and Applied Mathematics*, 2016.

[36] R. Yu, E. de Sturler, and D. D. Johnson, "A Block Iterative Solver for Complex Non-Hermitian Systems Applied to Large-Scale Electronic-Structure Calculations," University of Illinois at Urbana-Champaign, Department of Computer Science, Technical Report UIUCDCS-R-2002-2299, 2002.

[37] H. Calandra, S. Gratton, J. Langou, X. Pinel, and X. Vasseur, "Flexible Variants of Block Restarted GMRES Methods with Application to Geophysics," *SIAM Journal on Scientific Computing*, vol. 34, no. 2, pp. A714–A736, 2012.

[38] T. Sakurai, H. Tadano, and Y. Kuramashi, "Application of block Krylov subspace algorithms to the Wilson–Dirac equation with multiple right-hand sides in lattice QCD," *Computer Physics Communications*, vol. 181, no. 1, pp. 113–117, 2010.

[39] R. Falgout and U. Yang, "*hypre*: A library of high performance preconditioners," *Computational Science—ICCS 2002*, pp. 632–641, 2002.

[40] P. Bastian, F. Heimann, and S. Marnach, "Generic implementation of finite element methods in the distributed and unified numerics environment (DUNE)," *Kybernetika*, vol. 46, no. 2, pp. 294–315, 2010.

[41] D. Lukarski and N. Trost, "PARALUTION web page," http://www.parallution.com, 2016.

[42] A. H. Baker, E. R. Jessup, and T. Manteuffel, "A Technique for Accelerating the Convergence of Restarted GMRES," *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 4, pp. 962–984, 2005.

[43] J. Erhel, K. Burrage, and B. Pohl, "Restarted GMRES preconditioned by deflation," *Journal of computational and applied mathematics*, vol. 69, no. 2, pp. 303–318, 1996.

[44] D. N. Wakam and F. Pacull, "Memory efficient hybrid algebraic solvers for linear systems arising from compressible flows," *Computers & Fluids*, vol. 80, pp. 158–167, 2013.

[45] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps *et al.*, "An overview of the Trilinos project," *ACM Transactions on Mathematical Software (TOMS)*, vol. 31, no. 3, pp. 397–423, 2005.

[46] E. Bavier, M. Hoemmen, S. Rajamanickam, and H. Thornquist, "Amesos2 and Belos: Direct and Iterative Solvers for Large Sparse Linear Systems," *Scientific Programming*, vol. 20, no. 3, pp. 241–255, 2012.

[47] M. Mohiyuddin, M. Hoemmen, J. Demmel, and K. Yelick, "Minimizing Communication in Sparse Matrix Solvers," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC09*. ACM, 2009.

[48] A. Stathopoulos and K. Wu, "A Block Orthogonalization Procedure with Constant Synchronization Requirements," *SIAM Journal on Scientific Computing*, vol. 23, no. 6, pp. 2165–2182, 2002.

[49] I. Yamazaki, S. Tomov, and J. Dongarra, "Mixed-Precision Cholesky QR Factorization and Its Case Studies on Multicore CPU with Multiple GPUs," *SIAM Journal on Scientific Computing*, vol. 37, no. 3, pp. C307–C330, 2015.

[50] D. L. Brown, R. Cortez, and M. L. Minion, "Accurate Projection Methods for the Incompressible Navier–Stokes Equations," *Journal of Computational Physics*, vol. 168, no. 2, pp. 464–499, 2001.

[51] L. Feng, P. Benner, and J. G. Korvink, "Parametric model order reduction accelerated by subspace recycling," in *Proceedings of the 48th IEEE Conference on Decision and Control*. IEEE, 2009, pp. 4328–4333.

[52] H. C. Elman, O. G. Ernst, and D. P. O'Leary, "A Multigrid Method Enhanced by Krylov Subspace Iteration for Discrete Helmholtz Equations," *SIAM Journal on Scientific Computing*, vol. 23, no. 4, pp. 1291–1315, 2001.

[53] S. Cools, B. Reps, and W. Vanroose, "An Efficient Multigrid Calculation of the Far Field Map for Helmholtz and Schrödinger Equations," *SIAM Journal on Scientific Computing*, vol. 36, no. 3, pp. B367–B395, 2014.

[54] A. Amritkar, E. de Sturler, K. Świrydowicz, D. Tafti, and K. Ahuja, "Recycling Krylov subspaces for CFD applications and a new hybrid recycling solver," *Journal of Computational Physics*, vol. 303, pp. 222–237, 2015.

[55] S. Wang, E. de Sturler, and G. H. Paulino, "Large-scale topology optimization using preconditioned Krylov subspace methods with recycling," *International Journal for Numerical Methods in Engineering*, vol. 69, no. 12, pp. 2441–2468, 2007.

[56] J.-C. Nédélec, "Mixed finite elements in $\mathbb{R}^3$," *Numerische Mathematik*, vol. 35, no. 3, pp. 315–341, 1980.

[57] D. Boffi, P. Fernandes, L. Gastaldi, and I. Perugia, "Computational Models of Electromagnetic Resonators: Analysis of Edge Element Approximation," *SIAM Journal on Numerical Analysis*, vol. 36, no. 4, pp. 1264–1290, 1999.

[58] F. Rapetti, "High-order edge elements on simplicial meshes," *ESAIM: Mathematical Modelling and Numerical Analysis*, vol. 41, no. 06, pp. 1001–1020, 2007.

[59] S. Semenov, B. Seiser, E. Stoegmann, and E. Auff, "Electromagnetic Tomography for Brain Imaging: From Virtual to Human Brain," in *2014 IEEE Conference on Antenna Measurements & Applications*. IEEE, 2014.

[60] X.-C. Cai and M. Sarkis, "A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems," *SIAM Journal on Scientific Computing*, vol. 21, no. 2, pp. 792–797, 1999.

[61] V. Dolean, M. J. Gander, and L. Gerardo-Giorda, "Optimized Schwarz Methods for Maxwell's Equations," *SIAM Journal on Scientific Computing*, vol. 31, no. 3, pp. 2193–2213, 2009.

[62] F. Hecht, "New development in FreeFem++," *Journal of Numerical Mathematics*, vol. 20, no. 3-4, pp. 251–266, 2012.

[63] P. Jolivet, V. Dolean, F. Hecht, F. Nataf, C. Prud'homme, and N. Spillane, "High-performance domain decomposition methods on massively parallel architectures with FreeFem++," *Journal of Numerical Mathematics*, vol. 20, no. 3-4, pp. 287–302, 2012.

[64] C. Prudhomme, V. Chabannes, V. Doyeux, M. Ismail, A. Samake, and G. Pena, "Feel++: A Computational Framework for Galerkin Methods and Advanced Numerical Methods," in *ESAIM: Proceedings*, vol. 38. EDP Sciences, 2012, pp. 429–455.

[65] 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. Siddiqi, I. Duff, and O. Christensen, Eds. New Delhi, India: Anamaya Publishers, 2006, pp. 420–447.

[66] M. Bonazzoli and F. Rapetti, "High-order finite elements in numerical electromagnetism: degrees of freedom and generators in duality," *Numerical Algorithms*, 2016.

[67] O. Schenk and K. Gärtner, "Solving unsymmetric sparse systems of linear equations with PARDISO," *Future Generation Computer Systems*, vol. 20, no. 3, pp. 475–487, 2004.

[68] Intel, "MKL web page," https://software.intel.com/en-us/intel-mkl, 2016.

[69] A. Suzuki and F.-X. Roux, "A dissection solver with kernel detection for symmetric finite element matrices on shared memory computers," *International Journal for Numerical Methods in Engineering*, vol. 100, no. 2, pp. 136–164, 2014.

[70] K. Goto and R. Van De Geijn, "High-Performance Implementation of the Level-3 BLAS," *ACM Transactions on Mathematical Software*, vol. 35, no. 1, p. 4, 2008.

[71] H. Anzt, S. Tomov, and J. Dongarra, "Accelerating the LOBPCG Method on GPUs Using a Blocked Sparse Matrix Vector Product," in *Proceedings of the Symposium on High Performance Computing*, ser. HPC '15. Society for Computer Simulation International, 2015, pp. 75–82.

[72] T. V. Kolev and P. Vassilevski, "Parallel Auxiliary Space AMG for $H(\text{curl})$ Problems," *Journal of Computational Mathematics*, vol. 27, no. 5, pp. 604–623, 2009.

[73] J. J. Hu, R. S. Tuminaro, P. B. Bochev, C. J. Garasi, and A. C. Robinson, "Toward an h-Independent Algebraic Multigrid Method for Maxwell's Equations," *SIAM Journal on Scientific Computing*, vol. 27, no. 5, pp. 1669–1688, 2006.

[74] F. Pellegrini and J. Roman, "SCOTCH: A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs," in *High-Performance Computing and Networking*. Springer, 1996, pp. 493–498.

[75] H. Calandra, S. Gratton, R. Lago, X. Vasseur, and L. M. Carvalho, "A Modified Block Flexible GMRES Method with Deflation at Each Iteration for the Solution of Non-Hermitian Linear Systems with Multiple Right-Hand Sides," *SIAM Journal on Scientific Computing*, vol. 35, no. 5, pp. S345–S367, 2013.

[76] P.-H. Tournier, I. Aliferis, M. Bonazzoli, M. de Buhan, M. Darbas, V. Dolean, F. Hecht, P. Jolivet, I. El Kanfoud, C. Migliaccio, F. Nataf, C. Pichot, and S. Semenov, "Microwave Tomographic Imaging of Cerebrovascular Accidents by Using High-Performance Computing," 2016.

Artifact Description: Block Iterative Methods and Recycling for Improved Scalability of Linear Solvers

Pierre Jolivet*
*CNRS
IRIT–ENSEEIHT
pierre.jolivet@enseeiht.fr

Pierre-Henri Tournier‡§
‡Laboratoire J.-L. Lions, UPMC
§Inria Paris, ALPINES research team
tournier@ann.jussieu.fr

APPENDIX

A. Abstract

This description contains the information needed to launch some experiments of the SC16 paper "Block Iterative Methods and Recycling for Improved Scalability of Linear Solvers". More precisely, we explain how to compile and run the modified PETSc examples used in section IV. The results from section V can be reproduced using a finite element library interfaced with HPDDM, but this artifact description is not focused on that part of the paper.

B. Description

1) Check-list (artifact meta information):

- **Algorithm:** GCRO-DR with right, left, or variable preconditioning
- **Program:** C binary, C and C++ libraries
- **Compilation:** icpc version 16.0.2.181 (gcc version 4.9.1 compatibility) with the -O3 flag
- **Output:** time to solution and number of iterations
- **Experiment workflow:** install PETSc, clone HPDDM, compile the HPDDM C library, compile the modified PETSc examples, run the binaries, observe the results
- **Experiment customization:** number of MPI processes, threads, and grid points, standard parameters of Krylov methods...
- **Publicly available?:** yes

2) How delivered: HPDDM can be cloned from GitHub using the following URL: https://github.com/hpddm/hpddm. The examples taken from the PETSc distribution are in the folder `examples/petsc`.

3) Hardware dependencies: none. Note that we will link binaries with shared objects. As such, systems with severe limitations when it comes to dynamic loading—e.g., IBM BlueGene/Q—are not covered in this document (but it is not a problem to use static libraries instead).

4) Software dependencies: HPDDM requires a C++11 compliant compiler such as: g++ 4.7.2 and above, clang++ 3.3 and above, icpc 15.0.0.090 and above, or pgc++ 15.1 and above. icpc 16.0.1.150 and below and pgc++ are partially bugged when activating C++11 support. Please apply the following patch to the sources of HPDDM first if you are using one of these compilers:

```
$ sed -i\ '' 's/type\* = nullptr/type* = (void*)0 \
  /g; s/static constexpr const char/const char    \
  /g' include/*.hpp examples/*.cpp
```

BLAS and LAPACK are needed for dense linear algebra computations but can be automatically downloaded by PETSc. PETSc is available at the following URL: http://ftp.mcs.anl.gov/pub/petsc/release-snapshots/petsc-lite-3.7.3.tar.gz. Do not forget to turn off debugging and error detection if you need to compile PETSc (`--with-debugging=0 --with-errorchecking=0`).

C. Installation

1) clone HPDDM and enter the newly created directory

```
$ git clone https://github.com/hpddm/hpddm
$ cd hpddm
```

2) create an appropriate `Makefile.inc` by defining:
   a) `MPICXX`, a C++ compiler wrapping an MPI implementation
   b) `CXXFLAGS`, to activate C++11 support and such
   c) `BLAS_LIBS`, to link with BLAS and LAPACK
   d) or if you are using Intel Math Kernel Library, define `MKL_INCS` and `MKL_LIBS` instead

   Here are some minimalist `Makefile.inc` examples, be sure to link PETSc and HPDDM with the same BLAS and MPI implementations as they are not all ABI compatible:

   i. for Linux-based systems with the legacy BLAS

```
MPICXX    = mpic++
CXXFLAGS  = -std=c++11 -O3 -fPIC
BLAS_LIBS = -L/usr/lib -lblas -llapack
```

   ii. for Linux-based systems with Intel MKL and GOMP

```
MPICXX    = mpic++
CXXFLAGS  = -std=c++11 -O3 -fPIC
MKL_LIBS = -lgomp -L${MKLROOT}/lib/intel64          \
  -lmkl_core -lmkl_intel_lp64 -lmkl_gnu_thread
MKL_INCS = -I${MKLROOT}/include
```

   iii. for macOS systems with Apple BLAS

```
MPICXX    = mpic++
CXXFLAGS  = -std=c++11 -O3 -fPIC
BLAS_LIBS = -framework Accelerate
```

   iv. for macOS systems with Intel MKL and IOMP

```
MPICXX    = mpic++
CXXFLAGS = -std=c++11 -O3 -fPIC
MKL_LIBS = -L/opt/intel/lib -L/opt/intel/mkl/lib  \
  -liomp5 -lmkl_core -lmkl_intel_lp64             \
  -lmkl_intel_thread
MKL_INCS = -I/opt/intel/mkl/include
```

3) compile the C library

```
$ LIST_COMPILATION=c make lib
```

4) copy the modified PETSc examples into your PETSc installation

```
$ cp examples/petsc/ex32.c examples/petsc/ex56.c  \
  ${PETSC_DIR}/src/ksp/ksp/examples/tutorials
```

5) store the working directory in an environment variable and make sure that the shared library can be found, e.g., for some systems:

```
$ export HPDDM_DIR=`pwd`
$ export LD_LIBRARY_PATH=`pwd`/lib:\
  ${LD_LIBRARY_PATH}
```

D. Experiment workflow

Now that the HPDDM C library is compiled, PETSc toolchain will be used for generating the binaries. For that matter, change directory and compile the modified PETSc examples:

```
$ cd ${PETSC_DIR}/src/ksp/ksp/examples/tutorials
$ make ex32 ex56 CFLAGS="-I${HPDDM_DIR}/interface \
  -L${HPDDM_DIR}/lib -lhpddm_c"
```

Most HPDDM and PETSc options may be set via command line, so there is almost no need to recompile either the library or the binaries. In the rest of the artifact description, we will explain the most important options to set up in order to reproduce the results of the paper.

E. Evaluation and expected result

To make sure that everything runs smoothly, here are two commands (one for each modified PETSc example) that should run on most platforms (from laptops to supercomputers):

```
$ mpirun -np 8 ./ex32 -hpddm_recycle_same_system  \
  -ksp_pc_side right -ksp_rtol 1.0e-6             \
  -hpddm_recycle 10 -hpddm_krylov_method gcrodr   \
  -hpddm_gmres_restart 30 -da_refine 2
$ mpirun -np 8 ./ex56 -ne 9 -ksp_pc_side right    \
  -ksp_rtol 1.0e-6 -hpddm_gmres_restart 30        \
  -hpddm_krylov_method gcrodr -hpddm_recycle 10
```

The output for example 32 should include the following lines:

PETSc (GMRES)			HPDDM (GCRO-DR)		
1	81	0.005241	1	64	0.005964
2	65	0.003395	2	28	0.001851
3	77	0.003898	3	27	0.001860
4	65	0.003308	4	28	0.001987
	288	0.015842		147	0.011662

The first column is the index of the linear system solved, the second column is the number of iterations needed to reach convergence, and the third column is the time to solution (excluding setup) in seconds. The last line is the sum of all rows.

The output for example 56 should include the following lines (which have the same structure as described previously):

PETSc (GMRES)			HPDDM (GCRO-DR)		
1	128	0.018176	1	70	0.014209
2	77	0.010872	2	60	0.014578
3	98	0.013834	3	79	0.018486
4	106	0.014781	4	38	0.009578
	409	0.057663		247	0.056851

F. Experiment customization

In the paper, numerical experiments were carried out with the two previously compiled examples but with the following adjusted parameters: grid size, preconditioner type, dimension of recycled Krylov subspaces. All PETSc options were disclosed in the paper, but due to double-blind review policy, HPDDM options were omitted. Here are the exact command lines including both sets of options:

- for section IV-B

```
$ mpirun -np 8192 ./ex32 -ksp_rtol 1.0e-8          \
  -pc_type gamg -ksp_type fgmres -da_refine 2      \
  -mg_levels_ksp_type gmres -da_grid_x 4210        \
  -mg_levels_ksp_max_it 3 -da_grid_y 4210          \
  -hpddm_krylov_method gcrodr -hpddm_recycle 10    \
  -hpddm_gmres_restart 30 -hpddm_tol 1.0e-8        \
  -hpddm_variant flexible -pc_gamg_square_graph 2  \
  -hpddm_recycle_strategy B                        \
  -hpddm_recycle_same_system                       \
  -pc_gamg_threshold 0.0725
$ mpirun -np 8192 ./ex32 -ksp_rtol 1.0e-8          \
  -pc_type gamg -ksp_type fgmres -da_refine 2      \
  -mg_levels_ksp_type gmres -da_grid_x 4210        \
  -mg_levels_ksp_max_it 1 -da_grid_y 4210          \
  -hpddm_krylov_method gcrodr -hpddm_recycle 10    \
  -hpddm_gmres_restart 30 -hpddm_tol 1.0e-8        \
  -hpddm_variant flexible -pc_gamg_square_graph 2  \
  -hpddm_recycle_same_system                       \
  -hpddm_recycle_strategy B                        \
  -pc_gamg_threshold 0.076
```

- for section IV-C

```
$ mpirun -np 8000 ./ex56 -ne 399 -ksp_rtol 1.0e-8 \
  -ksp_type fgmres -pc_type gamg                   \
  -mg_levels_ksp_type cg -mg_levels_ksp_max_it 4   \
  -hpddm_krylov_method gcrodr -hpddm_recycle 10    \
  -hpddm_gmres_restart 30 -hpddm_tol 1.0e-8        \
  -hpddm_variant flexible -hpddm_recycle_strategy A
$ mpirun -np 8000 ./ex56 -ne 399 -ksp_rtol 1.0e-8 \
  -ksp_type lgmres -pc_type gamg                   \
  -ksp_lgmres_augment 10 -ksp_pc_side right        \
  -mg_levels_ksp_type chebyshev -hpddm_recycle 10  \
  -hpddm_krylov_method gcrodr -hpddm_tol 1.0e-8    \
  -hpddm_gmres_restart 30 -hpddm_variant flexible  \
  -hpddm_recycle_strategy A
```

The list of all available PETSc (resp. HPDDM) options may be displayed by appending the option `-help` (resp. `-hpddm_help`) to the command line arguments. Alternatively, these options are also described at the following URL: http://www.mcs.anl.gov/petsc/documentation (resp. https://github.com/hpddm/hpddm/blob/master/doc/cheatsheet.pdf)

G. Notes

For GCRO-DR, `-hpddm_recycle_strategy A` (resp. `B`) means solving the generalized eigenvalue problem in fig. 1 (line 33) with the right-hand side matrix W defined eq. (3a) (resp. eq. (3b)).

In the last experiment of section IV-C, we use FGCRO-DR instead of GCRO-DR with right preconditioning because this leads to less operations (at a cost of additional storage, which is typical of flexible iterative methods).