Jolivet, P.; Roman, JE.; Zampini, S. (2021). KSPHPDDM and PCHPDDM: Extending PETSc with advanced Krylov methods and robust multilevel overlapping Schwarz preconditioners. Computers & Mathematics with Applications. 84:277-295. https://doi.org/10.1016/j.camwa.2021.01.003

KSPHPDDM and PCHPDDM: extending PETSc with advanced Krylov methods and robust multilevel overlapping Schwarz preconditioners

Pierre Jolivet[a,*], Jose E. Roman[b], Stefano Zampini[c]

[a] *CNRS, IRIT–ENSEEIHT, Toulouse, France*
[b] *Universitat Politècnica de València, València, Spain*
[c] *King Abdullah University of Science and Technology, Thuwal, Saudi Arabia*

Abstract

Contemporary applications in computational science and engineering often require the solution of linear systems which may be of different sizes, shapes, and structures. The goal of this paper is to explain how two libraries, PETSc and HPDDM, have been interfaced in order to offer end-users robust overlapping Schwarz preconditioners and advanced Krylov methods featuring recycling and the ability to deal with multiple right-hand sides. The flexibility of the implementation is showcased and explained with minimalist, easy-to-run, and reproducible examples, to ease the integration of these algorithms into more advanced frameworks. The examples provided cover applications from eigenanalysis, elasticity, combustion, and electromagnetism.

Keywords: Krylov methods, domain decomposition preconditioners, distributed-memory parallel computing

1. Introduction

Computational science and engineering today enjoys unprecedented opportunities to transform the way society solves many of its most urgent technological problems through predictive simulations. At the heart of simulations are robust and scalable solution algorithms. The efficient production of applications requires a rich ecosystem of state-of-the-art reusable libraries from which domain specialists can benefit [1]. In this paper, we focus on the interoperability of two such libraries.

On the one hand, PETSc [2, 3], the Portable and Extensible Toolkit for Scientific computation, is a well-established, actively developed software from the community. It may be used to efficiently discretize partial differential equations and solve algebraic linear or nonlinear systems of time-dependent equations. Among its strengths, it offers many advanced features regarding preconditioning and tailored matrix formats, and it can interact with third-party libraries, such as *hypre* [4] for linear solvers, TetGen [5] for mesh generation, p4est [6] for adaptive mesh refinement, to cite a few. The extensibility and robustness of the framework convinced computational scientists to use PETSc as one of the discretization and/or algebraic backend in many different higher-level projects, see https://www.mcs.anl.gov/petsc for a comprehensive list.

*Corresponding author
Email addresses: pierre.jolivet@enseeiht.fr (Pierre Jolivet), jroman@dsic.upv.es (Jose E. Roman), stefano.zampini@kaust.edu.sa (Stefano Zampini)

On the other hand, HPDDM [7], the High-Performance unified framework for Domain Decomposition Methods (HPDDM) is a much smaller project focusing on robust and scalable domain decomposition preconditioners and advanced iterative methods [8] which can efficiently deal with linear systems with multiple right-hand sides, i.e., block iterative methods, and when there is a recurrence of varying coefficient matrices and right-hand sides, i.e., recycling iterative methods.

Here, we describe the integration of the HPDDM solvers suite consisting of advanced iterative methods and robust domain decomposition preconditioners into PETSc. The design principles of the interface are showcased considering minimalist and easy-to-run examples, with a focus on fulfilling reproducibility requirements as advocated by the Association for Computing Machinery https://www.acm.org/publications/policies/artifact-review-badging, as well as to ease the integration of these algorithms into more advanced frameworks and facilitate the analysis and comparison of performance results [9].

Fruits of the proposed enhancement are discussed for applications from SLEPc [10], the Scalable Library for Eigenvalue Problem computations, an add-on library that extends PETSc with classes for linear and nonlinear eigenvalue problems, as well as with tools to facilitate the use of shift-and-invert spectral transformations.

The paper is divided in two main parts. Section 2 introduces the interface to recycling and block Krylov methods (`KSPHPDDM`) and discusses the readiness of the PETSc library with respect to block Krylov methods. Different applications from eigenanalysis are also provided. Section 3 introduces the suite of robust multilevel overlapping Schwarz methods (`PCHPDDM`) with examples from linear and nonlinear partial differential equations. Eventually, concluding remarks are provided in section 4.

All PETSc keywords and options are typeset in `typewriter` font, these are public and documented at https://www.mcs.anl.gov/petsc/petsc-master/docs. Codes not already available in other public repositories are available at https://github.com/prj-/jolivet2020petsc. In Appendix A, a short guide to build the required set of libraries for running these codes is provided.

2. Advanced Krylov methods in PETSc: KSPHPDDM

2.1. Related work

Krylov subspace methods are widely used in numerical linear algebra for solving linear systems of equations because of their memory efficiency [11, 12]. They mainly rely on matrix–vector operations such as multiplications or transpose multiplications and, for robust convergence when dealing with challenging high-dimensional systems, on the application of appropriate preconditioners. PETSc, as of version 3.14.0, already offers 44 different types of Krylov methods within the `KSP` base class. Some of the most used types are `KSPGMRES`, `KSPCG`, or `KSPBCGS`, which respectively implement the generalized minimal residual method [13] (GMRES), conjugate gradient [14] (CG), or biconjugate gradient stabilized method [15] (BCGS).

Because Krylov methods may need long-term recurrences to mitigate round-off errors during the generation of subspaces, it is common to introduce a restart parameter in order to control their memory consumption and the volume of global communications needed when orthonormalizing a candidate basis vector. Such restarts may hinder convergence of iterative methods, and even introduce convergence plateaus. Recycling techniques have been introduced to attenuate these effects. On the one hand, PETSc implements the loose GMRES [16] (LGMRES) and the

deflated GMRES [17, 18] (DGMRES) as `KSPLGMRES` and `KSPDGMRES`. However, neither handle variable preconditioning, unlike `KSPFGMRES` or `KSPGCR` which respectively implement the flexible GMRES [19] (FGMRES) and the generalized conjugate residual method [20] (GCR). Furthermore, `KSPDGMRES` does not support complex arithmetic. Moreover, extending the recycling capabilities of such methods to sequence of linear systems with smoothly varying coefficient matrices and right-hand sides is not trivial. On the other hand, HPDDM offers support for the generalized conjugate residual method with inner orthogonalization and deflated restarting [21] (GCRODR), which does not suffer from the aforementioned limitations.

Another important aspect of Krylov methods is their ability to deal with multiple right-hand sides simultaneously. Block Krylov methods [22] are designed for solving linear systems $AX = B$, where X and B are tall-and-skinny matrices with $k \geq 1$ columns. Such methods, while having higher arithmetic intensities, generate larger subspaces and typically converge in fewer iterates. These methods are not currently offered in PETSc and they are less frequently implemented in general purpose libraries because they require somehow more involved kernels such as matrix–matrix multiplication, instead of matrix–vector product, and may involve different inner-product realizations [23]. Still, systems with multiple right-hand sides are ubiquitous, e.g., in tomography [24], data analytics [25], eigensolvers [26], geophysics [27], quantum chromodynamics [28], and optimization with time-dependent partial differential equations as constraints.

Trilinos [29] is another well-known software for scientific computing and it provides iterative methods through its Belos package [30]. Although having PETSc and Trilinos interoperate is possible, there is currently no `KSP` interface to Belos. Furthermore, some of the Krylov methods implemented in HPDDM and discussed in section 2.2, are not available in Belos.

2.2. Interfacing HPDDM Krylov methods in PETSc

In this section, we provide details of the interface between HPDDM Krylov methods and the various PETSc classes. It is assumed that the right-hand sides, and consequently the solutions, are always dense vectors or matrices.

`KSPHPDDM`, the interface between HPDDM Krylov methods and PETSc, is automatically registered since PETSc version 3.12 when configuring PETSc with the extra flag `--download-hpddm`. HPDDM linear solvers can be selected using the command line option `-ksp_type hpddm`, or programmatically via `KSPSetType(ksp, KSPHPDDM)`. Then, the following Krylov methods can be accessed:

- pseudo-block GMRES or flexible GMRES [19];
- pseudo-block CG or flexible CG [31];
- pseudo-block GCRODR or flexible GCRODR [32];
- block GMRES or flexible GMRES [27], with deflation at each restart;
- block CG [33];
- breakdown-free block CG [34], with deflation at each iteration;
- block GCRODR or flexible GCRODR, with deflation at each restart.

While being mathematically equivalent to their "standard" counterparts, the pseudo-block variants fuse together multiple similar operations like matrix–vector products to achieve higher arithmetic intensity, or to decrease the number of global synchronizations needed for scalar products.

HPDDM Krylov methods may be selected using the options `-ksp_hpddm_type (gmres|cg|gcrodr|bgmres|bcg|bfbcg|bgcrodr|preonly)`, or programmatically by using `KSPHPDDMSetType(ksp, KSPHPDDMType)`. Preconditioning variants can be specified via `-ksp_hpddm_variant (left|right|`

`flexible`) to select whether the preconditioner is applied on the left, on the right, or if it cannot be represented as a linear operator. In this latter case, the preconditioner is always applied on the right, except for the conjugate gradient methods BCG, BFCG, and BFBCG, that only handle left preconditioning. It is also possible to set the preconditioning side through the more common PETSc option `-ksp_pc_side (left|right)`. In addition, convergence monitoring with the `KSPMonitor` interface is fully supported, as well as the specification of customized convergence testing via the `KSPSetConvergenceTest` callback.

Recycling Krylov methods try to extract convergence information by solving a small standard or a generalized dense eigenproblem at the end of each cycle or when convergence is reached, and then reuse it appropriately for subsequent solves [35, 36, 37]. Such deflation subspaces, stored as dense tall-and-skinny matrices, can be accessed with `KSPHPDDMGetDeflationSpace`. User-defined deflation subspaces can also be specified via `KSPHPDDMSetDeflationSpace`.

To fully support HPDDM solvers, in version 3.14.0 of PETSc, we added interface routines for solving systems with multiple right-hand sides, `KSPMatSolve`(`ksp`, `X`, `Y`), and to apply preconditioners, `PCMatApply`(`pc`, `X`, `Y`). Both input X and output Y matrices are currently limited to be dense tall-and-skinny matrices. With `KSPHPDDM`, (pseudo-)block Krylov methods will be used. Instead, when no specialized implementation is available, PETSc will perform the solution phase in a column by column fashion. Furthermore, a function `KSPSetMatSolveBlockSize` is also provided to decompose a single large block of column vectors into multiple sub-blocks. `KSPMatSolve` is then called repeatedly until all sub-blocks are traversed. Similar considerations apply to the underlying calls to `PCMatApply` and `MatProductNumeric`. This was inspired by MUMPS [38] option ICNTL(27)[1].

While solving linear systems, possibly with multiple right-hand sides, HPDDM will repeatedly call the following PETSc routines:

- `MatMult`(A, x, y) for $y = Ax$;
- `MatMatMult`(A, X, `MAT_REUSE_MATRIX`, `PETSC_DEFAULT`, Y)[2] for $Y = AX$;
- `PCApply`(M, x, y) to apply a preconditioner to x, i.e., for $y = M^{-1}x$;
- `PCMatApply`(M, X, Y) for $Y = M^{-1}X$;

The rest of the operations are performed directly inside HPDDM. Specifically, within Krylov methods using the Arnoldi process, or when recycling is requested, one has to compute QR factorizations of tall-and-skinny dense matrices to orthonormalize candidate basis vectors. Such operations are performed using the CholQR algorithm [39], or via the (modified) Gram–Schmidt method. These orthonormalization variants can be selected at runtime with the option `-ksp_hpddm_qr (cholqr|mgs|cgs)`. For Hessenberg matrices generated by the Arnoldi process, it is common to update their QR decomposition using Givens rotations [12]. For block Hessenberg matrices, Householder reflectors are used instead [40].

For matrix–matrix products, there are currently specialized implementations for the following `MatTypes`:

- `MATAIJ`: standard sequential or parallel sparse matrix, based on compressed sparse row format;
- `MATSEQBAIJ`: block sparse matrix, based on block compressed sparse row format;
- `MATSEQSBAIJ`: symmetric block sparse matrix stored in upper triangular form;
- `MATSHELL`: user-defined matrix;
- `MATNEST`: block-defined matrix with nested submatrices;

[1] http://mumps.enseeiht.fr/doc/userguide_5.3.3.pdf, section 6.1

[2] or `MatProductNumeric`(Y) with PETSc 3.14.0 and above

- `MATAIJCUSPARSE`: sequential or parallel sparse matrix, offloaded to a NVIDIA GPU using cuSPARSE [41].

The following preconditioners have specialized implementations for dealing efficiently with multiple vectors:

1. `PCKSP`: embedded Krylov method;
2. `PCMAT`: matrix multiplication;
3. `PCH2OPUS`: hierarchical matrices [42, 43];
4. `PCHPDDM`: see section 3;
5. `PCASM` and `PCGASM`: overlapping Schwarz methods;
6. `PCBJACOBI`: block Jacobi;
7. `PCLU` or `PCCHOLESKY`: exact LU or Cholesky factorization;
8. `PCILU` or `PCICC`: incomplete LU or Cholesky factorization.

One appealing feature of domain decomposition-like preconditioners (items 4 to 6) is that they most often rely on exact or inexact factorizations (items 7 and 8) as subdomain solvers. In these cases, it is possible to access the so-called factored matrix F via `PCFactorGetMatrix`, and then call `MatMatSolve`(F, X, Y) to take advantage of blocked forward eliminations and backward substitutions from the various factorization packages interfaced with PETSc. In the case of a sparse matrix, this strategy is possible with: MUMPS [38], SuiteSparse [44], MKL PARDISO or CPARDISO [45], and SuperLU [46] or SuperLU_DIST [47]. For the case of a dense matrix: ScaLAPACK and Elemental [48] are supported. Future work will consider extending the multigrid framework `PCMG`, as well as other preconditioning classes to increase arithmetic intensity of the preconditioner application phase for blocks of right-hand sides.

2.3. Applications and numerical results

2.3.1. Reproducibility of the results from Parks et al. [21]

Alongside the paper introducing GCRODR [21], a MATLAB implementation was provided and is since then available[3]. It comes with a sequence of ten "linear systems from a finite element fracture mechanics problem constructed by Philippe H. Geubelle and Spandan Maiti." The goal of this paragraph is to explain how the results can be reproduced with PETSc and `KSPHPDDM`: the matrix files used are available at https://gitlab.com/petsc/datafiles/-/tree/master/matrices/hpddm/GCRODR while the driver code is part of the PETSc test suite and available at https://www.mcs.anl.gov/petsc/petsc-master/src/ksp/ksp/tutorials/ex75.c.html. In order to check the correctness of the `KSPHPDDM` interface, the following tests are performed:

- in MATLAB, unpreconditioned GCRODR(40, 20), ICC(0) left-preconditioned GCRODR(40, 20), and Jacobi right-preconditioned GCRODR(40, 20);
- in PETSc with no preconditioning and no restart GMRES(∞) `KSPGMRES`;
- in PETSc with `KSPHPDDM`, all of the above.

The notation GCRODR(n, m) indicates a restart after n iterations and a recycling subspace of dimension m. In all of the cases, convergence is declared when the initial unpreconditioned residual is reduced by 10 orders of magnitude. Except for the right-preconditioned GCRODR, these tests are the same as the ones from the original GCRODR paper [21]. All PETSc tests are performed using four MPI processes and can be launched using the following command lines.

[3] https://www.sandia.gov/~mlparks/GCRODR.zip

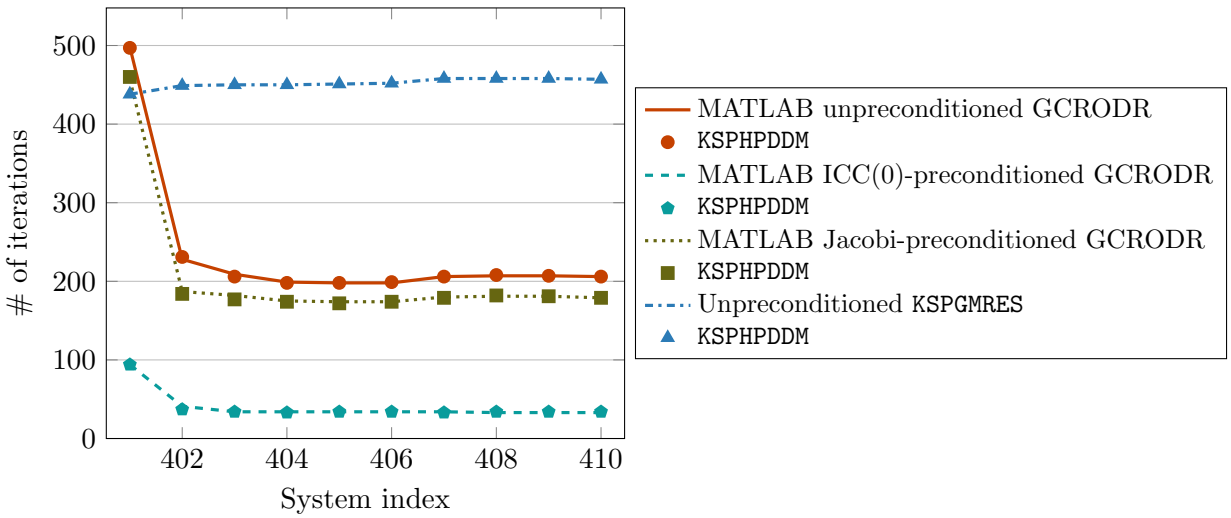


Figure 1: Number of iterations needed to converge for various configurations as originally tested by Parks et al. [21]. Note that the numbers of iterations needed by `KSPHPDDM` match with the respective MATLAB or PETSc reference implementation.

```
$ mpirun -n 4 ./ex75 -ksp_converged_reason -pc_type none -ksp_rtol 1e-10 -ksp_gmres_restart 40   \
  -ksp_type hpddm -ksp_hpddm_type gcrodr -ksp_hpddm_recycle 20                                  \
  -load_dir ${DATAFILESPATH}/matrices/hpddm/GCRODR
$ mpirun -n 4 ./ex75 -ksp_converged_reason -redundant_pc_type icc -ksp_rtol 1e-10               \
  -ksp_type hpddm -ksp_hpddm_type gcrodr -ksp_gmres_restart 40 -ksp_hpddm_recycle 20           \
  -load_dir ${DATAFILESPATH}/matrices/hpddm/GCRODR -pc_type redundant
$ mpirun -n 4 ./ex75 -ksp_converged_reason -pc_type jacobi -ksp_rtol 1e-10 -ksp_gmres_restart 40 \
  -ksp_type hpddm -ksp_hpddm_type gcrodr -ksp_hpddm_recycle 20                                  \
  -ksp_pc_side right -load_dir ${DATAFILESPATH}/matrices/hpddm/GCRODR
$ mpirun -n 4 ./ex75 -ksp_converged_reason -pc_type none -ksp_rtol 1e-10 -ksp_gmres_restart 500  \
  -ksp_type hpddm -load_dir ${DATAFILESPATH}/matrices/hpddm/GCRODR
```

The iteration numbers needed to reach convergence are gathered in figure 1. The cases —— and ● (resp. ·-·-· and ▲) reproduce Figure 4.2 in [21], while cases - - - and ⬟ partially reproduce Figure 4.3.

2.3.2. Performance of primitives for block Krylov methods

The readiness of PETSc, as of version 3.14.0, is now discussed when it comes to delivering efficient implementations of the core matrix–matrix multiplication needed by block Krylov methods and when a restart occurs in recycling Krylov methods.

In particular, we consider the first three sequential matrix types from section 2.2, namely `MATSEQAIJ`, `MATSEQBAIJ`, and `MATSEQSBAIJ`. Their distributed memory counterparts, e.g., `MATMPIAIJ`, rely on the sequential implementations, with each process storing two such matrices, one with local rows and columns, and another with local rows and "nonlocal" columns. To keep this study succinct, we only evaluate the performance of the sequential implementations, which can help in drawing conclusions for the intraprocess performance of the parallel formats. For this reason, from now on, we drop the `SEQ` substring. The performance of the multiplication primitives of the three aforementioned types will also be compared against those obtained with the MKL inspector–

executor sparse BLAS routines [45], using a single OpenMP thread. There is an ongoing effort to better integrate these routines in PETSc, but at the time of writing, direct calls to the MKL were used.

For benchmarking, we first discretize the Poisson equation on a cube with trilinear finite elements and obtain a matrix A of dimension one million. Though the numerical values of A are of no interest here, the sparsity pattern is rather common and frequently encountered when discretizing partial differential equations. We then generate a random symmetric dense b-by-b matrix T, for $b \in \{1, 3, 6\}$, and the matrix $\mathbb{A} = A \otimes T$ is assembled into all three formats described above, using a block size of b for the block formats. The performance of the `MatProductNumeric` operation is evaluated for tall-and-skinny dense matrices with a varying number of columns $N \in \{2, 8, 16, 32, 64\}$. For the single column case $N = 1$, the `MatMult` operation is used. All results have been obtained using double-precision arithmetic and 32-bit integers. The scaled efficiency measured in GFLOP per second is reported with respect to the baseline `MATAIJ` implementation with $N = 1$ in figure 2, where performances have been averaged over five consecutive product operations. They can be reproduced by using the mini-app `MatProduct.c` from https://github.com/prj-/jolivet2020petsc and any input `MATAIJ` stored in binary format, here using the default name `binaryoutput`, available at http://jolivet.perso.enseeiht.fr/binaryoutput.

```
$ mpicc MatProduct.c -O3 -I${PETSC_DIR}/${PETSC_ARCH}/include -I${PETSC_DIR}/include \
  -L${PETSC_DIR}/${PETSC_ARCH}/lib -lpetsc -o MatProduct
$ mpirun -n 1 ./MatProduct -f binaryoutput -log_view \
  -bs 1,3,6 -N 1,2,8,16,32,64 -type aij,aijmkl,baij,baijmkl,sbaij,sbaijmkl
```

Some conclusions may be drawn:

- with a block size of 1, top plot $b = 1$ in figure 2, using any type but `MATAIJ` is counterproductive when $N > 1$, see and , even when using the MKL, see and . With 8 or more columns, the performance of `MATAIJ` plateaus and the efficiency reaches approximately 200%, while the performance of the MKL primitives stagnate at around 175%, see . For the single column case, `MATSBAIJ` and `MATAIJMKL` deliver slightly better performances;
- for block sizes lower than or equal to 5, loops for matrix–vector and matrix–matrix multiplications are unrolled by hand for block formats. This yields rather disappointing performance, except for `MATAIJ`, see in middle plot $b = 3$, and its 350% efficiency, even with a moderate number of columns. For block formats and , given the small value of b, it could be beneficial to switch to optimized libraries for small matrices, e.g., LIBXSMM [49]. In fact, MKL implementation for block formats is here clearly outperforming PETSc, see and ;
- for block sizes larger than 5, block entry multiplication with PETSc block formats is performed using `?gemv` or `?gemm` operations. For `MATAIJ` in the bottom plot corresponding to $b = 6$, the efficiency quickly caps near 400% for $N \in \{8, 16, 32, 64\}$, while reaching 500% for large number of columns for the block formats, see and for $N = 64$. With these numbers of columns, PETSc and MKL perform similarly.

The mini-app is then used to benchmark intranode performance of matrix–matrix multiplications, either with multiple OpenMP threads, or by offloading the operation to a NVIDIA GPU and using cuSPARSE [41] as interfaced in PETSc.

```
$ export OMP_NUM_THREADS=20 && export MKL_NUM_THREADS=20
$ mpirun -n 1 ./MatProduct -f binaryoutput -log_view \
  -bs 1,3,6 -N 1,2,8,16,32,64 -type aijmkl,baijmkl,sbaijmkl,aijcusparse
```

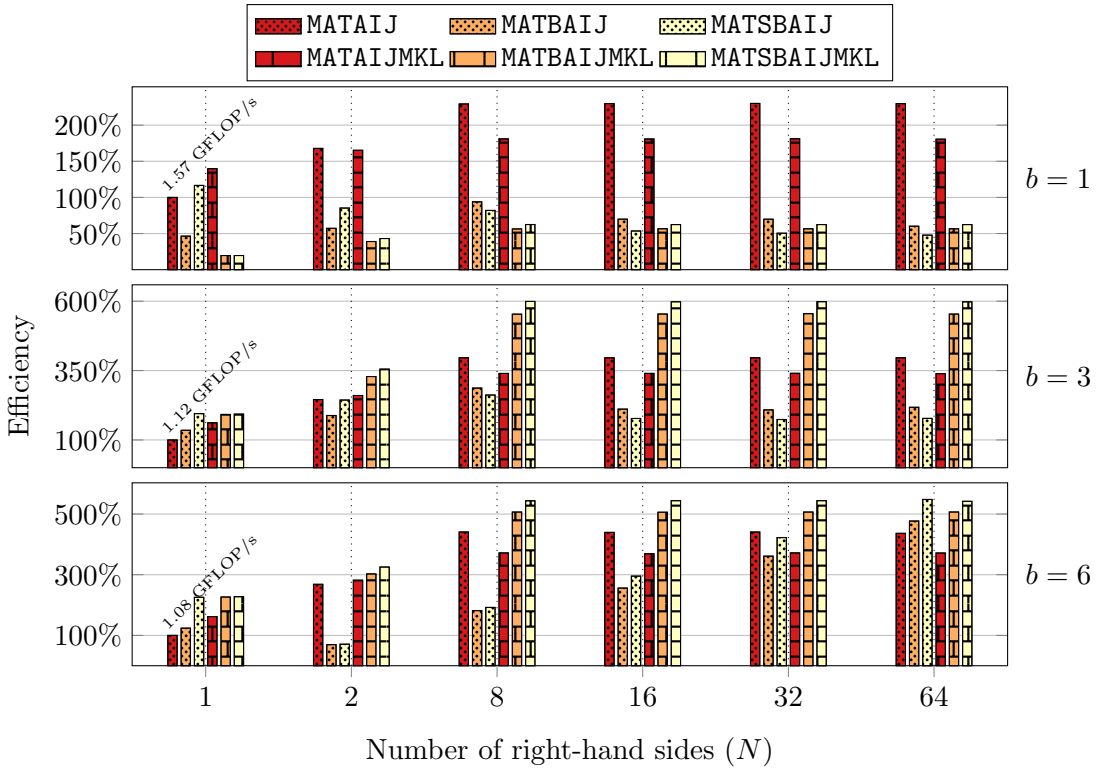


Figure 2: Performance of the matrix–matrix multiplication for different sparse matrix formats.

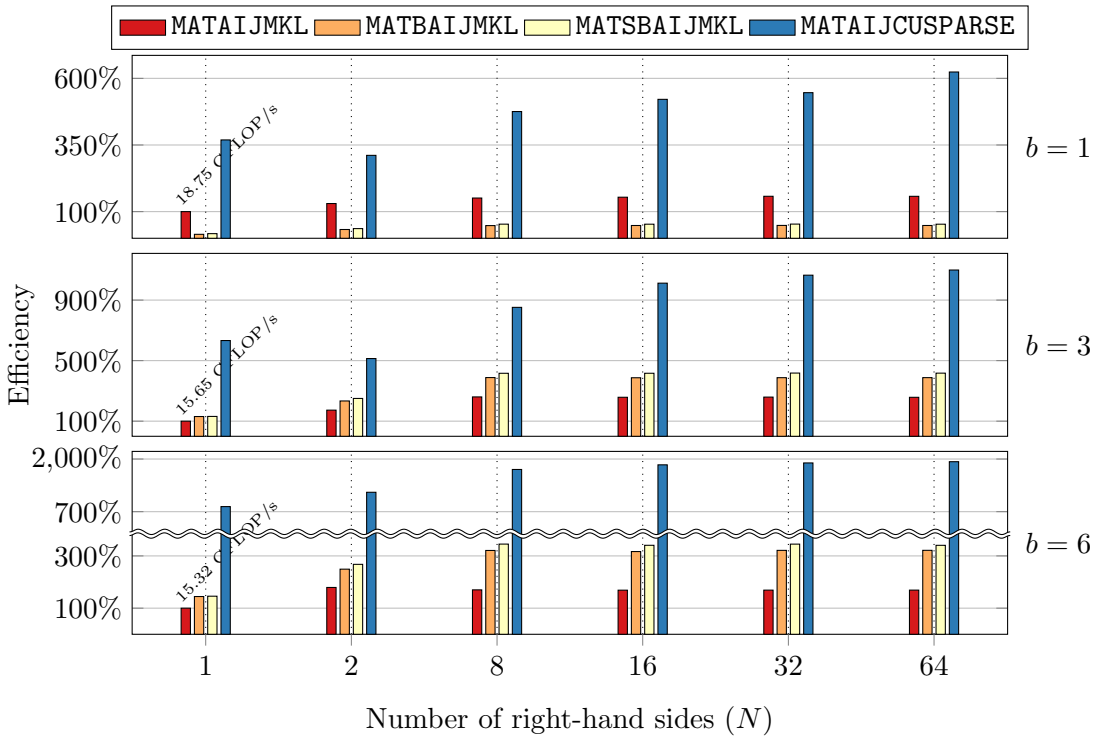


Figure 3: Performance of the matrix–matrix multiplication as implemented in PETSc using intranode parallelism.

Results reported in figure 3 have been scaled with respect to the baseline implementation `MATAIJMKL` with $N = 1$. They have been obtained on a single node of Jean Zay, a system composed of 261 nodes with two 20-core Intel Xeon Gold 6248 clocked at 2.5 GHz and four NVIDIA Tesla V100 SXM2. Neither transfers between host and device nor time spent in `mkl_sparse_optimize` have been taken into account. For the sake of completeness, the performance of the matrix–vector multiplication with `MATAIJMKL` and different numbers of threads is also reported in figure 4. Clearly, it is not advised to use the full socket to perform this type of workload for such small sparse matrices and skinny dense matrices. However, these results help in drawing a fair comparison between a full CPU socket and a GPU device.

The CPU configuration which reaches the highest percentage of peak is `MATSBAIJMKL` with $b = 3$ and $N = 64$, rightmost middle plot in figure 3, with approximately 63 GFLOP/s. This format is not yet available in PETSc, but it is handled by the mini-app. Still, it is less than 1% of peak for an Intel Xeon Gold 6248. On the GPU, `MATAIJCUSPARSE` with $b = 6$ and $N = 64$, rightmost bottom plot in figure 3, performs at around 306 GFLOP/s, about 4% of peak for an NVIDIA Tesla V100 SXM2. Future work may consider using interlaced layouts for storing the dense right-hand sides and extend the current PETSc functionality in order to maximize performance while maintaining interface flexibility and the user friendliness of the library.

2.3.3. Linear stability analysis

In this section, we apply recycling Krylov methods in the context of linear stability analysis and consider the solution of the following generalized eigenvalue problem:

$$J(q_b)x = \lambda \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} x, \tag{1}$$

where J is the Jacobian of the incompressible steady-state Navier–Stokes equation evaluated using a given base flow $q_b = \begin{bmatrix} u_b \\ p_b \end{bmatrix}$ and M is the discretization of the mass matrix on the space of velocities. The mini-app employed for the numerical results is built on top of FreeFEM [50] and it is available at https://github.com/prj-/moulin2019al. Interested readers are referred to [51] for more details and for larger runs.

There are different methods to compute the eigenvalues of equation (1) near a complex-valued shift σ. In this paragraph we consider a Krylov–Schur method [52], which, for interior eigenvalues, relies on spectral transformations and on the solution of successive linear systems such as:

$$\left(J(q_b) - \sigma \begin{bmatrix} M & 0 \\ 0 & 0 \end{bmatrix} \right) x_i = b_i, \tag{2}$$

which, in SLEPc, are parameterized using the `-st_` prefix. In this work, the above linear system is preconditioned using a modified augmented Lagrangian approach [53, 51]

The effectiveness of subspace recycling is shown for finding the 5 eigenpairs closest to the shift $\sigma = 10^{-6} + 0.6\mathrm{i}$ for a flow past a cylinder at Reynolds 100. Results can be reproduced using the following four commands. We note here that the first two commands are merely used to generate the base flow q_b with a `SNES`, a PETSc object used to solve nonlinear problems, using a continuation method on the Reynolds number.

```
$ mpirun -n 4 FreeFem++-mpi Nonlinear-solver.edp -Re 50 -v 0
$ mpirun -n 4 FreeFem++-mpi Nonlinear-solver.edp -Re 100 -v 0
$ mpirun -n 4 FreeFem++-mpi Eigensolver.edp -Re 100 -v 0 -st_ksp_rtol 1.0e-4                \
  -st_ksp_type fgmres -st_ksp_gmres_restart 200 -st_ksp_converged_reason
$ mpirun -n 4 FreeFem++-mpi Eigensolver.edp -Re 100 -v 0 -st_ksp_rtol 1.0e-4                \
  -st_ksp_type hpddm -st_ksp_gmres_restart 200 -st_ksp_converged_reason                     \
  -st_ksp_hpddm_variant flexible -st_ksp_hpddm_recycle 10 -st_ksp_hpddm_type gcrodr
```

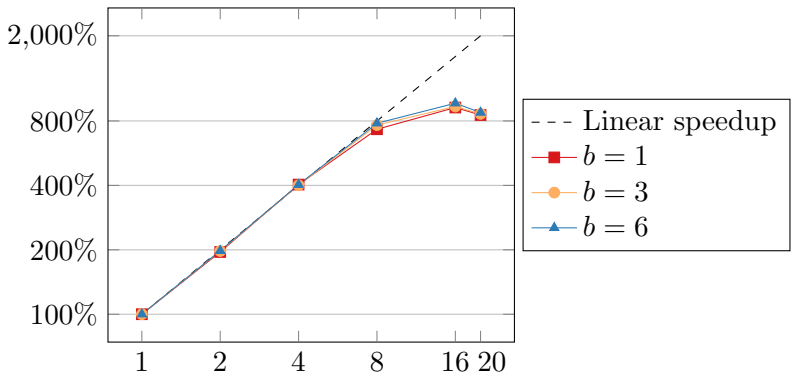


Figure 4: Scalability of `MatMult` with `MATAIJMKL` and different number of threads.

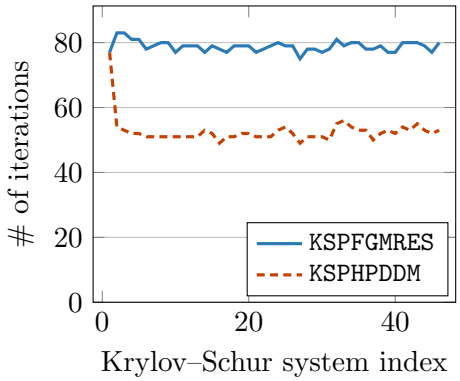


Figure 5: Number of inner iterations for systems equation (2) with ---- or without —— recycling.

The Krylov–Schur algorithm converges in six outer iterations (restarts), with a total of 46 inner solves equation (2). Recycling of Krylov subspaces with `KSPHPDDM` provides algorithmic speedup by lowering the number of iterations per inner solve, as reported in figure 5, from 80 with `KSPFGMRES` to 45. A similar speedup can be observed in terms of runtime, with the time spent for the solution of the eigenvalue problem being reduced from 25.1 min to 17.5 min. These timings, and all that follow, have been obtained on Irène, a system composed of 1,656 nodes with two 24-core Intel Xeon Platinum 8168 clocked at 2.7 GHz.

2.3.4. Blocking inside SLEPc

The implementations of the locally optimal block preconditioned conjugate gradient [26] (LOBPCG) and the contour integral spectrum slicing method [54] (CISS) in SLEPc were previously not using blocking when applying the preconditioner or solving linear systems. In version 3.14.0, these solvers have been adapted to employ `KSPMatSolve` and `PCMatApply`. In the case of LOBPCG, which is a purely blocked method, all steps were already implemented as block operations, except the applications of the preconditioner, which undermined the benefits of blocking. The performance of LOBPCG implemented in SLEPc as `EPSLOBPCG` is studied using three different preconditioners:

- `PCASM`: one-level overlapping Schwarz method with one level of overlap and exact Cholesky factorizations in each subdomain;
- `PCHPDDM`: multilevel overlapping Schwarz method, see section 3;
- `PCGAMG`: algebraic multigrid method [55].

More details on these solvers are given in the next section where focus is put on preconditioning rather than Krylov methods. We note here that only `PCASM` and `PCHPDDM` currently handle blocking, see the list section 2.2, and that `PCHPDDM` and `PCGAMG` have lower contraction factors than the cheaper alternative `PCASM`.

As a testbed, we consider the following generalized eigenvalue problem on a three-dimensional cube:

$$-\nabla \cdot (\kappa \nabla u) = \lambda u.$$

This continuous equation is discretized with third-order Lagrange finite elements using FreeFEM. The script `blocking-slepc.edp` available at https://github.com/prj-/jolivet2020petsc can be used for reproducibility. The following timings and convergence histories have been obtained on 2,304 processes of Irène, for solving a problem of dimension $7.46 \cdot 10^7$.

By default, SLEPc applies at most 5 iterations of GMRES on each column of a so-called set of active columns. Since the option `-eps_lobpcg_blocksize 40` is used for computing 20 eigenpairs, the size of the set varies between 40 to 1. This size corresponds to the number of right-hand sides that will be solved for throughout LOBPCG iterations. Instead of using `KSPGMRES`, which does not handle blocking, `KSPHPDDM` is used, with the pseudo-block GMRES. Thus, assuming the size of the set of active columns is 40, one LOBPCG iteration performs at most 5 pseudo-block GMRES iterations with 40 vectors simultaneously, using `MatProductNumeric` and `PCMatApply`, instead of doing at most 40×5 successive GMRES iterations, using `MatMult` and `PCApply`.

Results are gathered in table 1. In the first row, the numbers of outer LOBPCG iterations are reported with the three different inner preconditioners mentioned above. On the one hand, `PCASM` is known to yield a preconditioned operator whose condition number grows as the number of subdomains, here processes, increases. Thus, the number of outer iterations is higher than with `PCHPDDM` or `PCGAMG` since the inner solves are not converging to meaningful solutions in just 5 iterations. On the other hand, both multilevel preconditioners perform similarly, with a minimal difference in the number of outer iterations. The second and third rows record the number of times `PCApply` and `PCMatApply` are called. Since `PCGAMG` does not handle blocking, at each pseudo-block GMRES iteration, `PCApply` is called for each active column separately. `PCASM` and `PCHPDDM` handle blocking, so they only rely on `PCMatApply`. In the fourth row, the time spent setting up the preconditioners is given. `PCASM` is extremely cheap but not very robust numerically, while both multilevel preconditioners have a similar setup time. The time spent in the full eigensolver is reported in the last row. Clearly, having both a `KSP` and a `PC` which can efficiently deal with multiple vectors is mandatory to achieve reasonable performance with such an outer solver that heavily relies on blocking.

Table 1: Performance of `EPSLOBPCG` with three different inner preconditioners: `PCASM` and `PCHPDDM`, which utilize `PCMatApply`, and `PCGAMG`, which utilizes `PCApply`.

	`PCASM`	`PCHPDDM`	`PCGAMG`
# of outer iterations	54	14	15
`PCApply`	—	—	4,049
`PCMatApply`	378	98	—
`PCSetUp` (sec)	2.9	21.6	26.5
`EPSSolve` (sec)	265.7	121.6	391.8

The script available at https://github.com/prj-/jolivet2020petsc may be used to solve the same eigenproblem using `EPSCISS` by providing the additional command line argument `-eps_type ciss`. However, this eigensolver is mostly suited for finding interior eigenpairs, whereas the left-most part of the spectrum, closest to 0 for a symmetric positive definite problem, is here sought. In this scenario, `EPSCISS` is not a suitable solver and for fairness, its algorithmic performance is not reported, but the options from the script can be readily used for problems where it is an appropriate choice. We finally note that blocking support is also available and tested in SLEPc for `EPSSUBSPACE` since version 3.14.0.

3. Robust multilevel overlapping Schwarz preconditioners: PCHPDDM

3.1. Related work

Domain decomposition methods are, alongside multigrid methods, one of the dominant paradigms for defining efficient and robust preconditioners in modern large-scale applications dealing with partial differential equations. There are many monographs on domain decomposition methods [56, 57, 58, 59, 60, 61].

Basic one-level methods such as the block Jacobi and the (overlapping) additive Schwarz method are implemented in PETSc as `PCBJACOBI` and `PCASM` respectively, with some additional customization [62] available for `PCASM`. However, none of these methods provide automatic support for a coarse-level solver, which is mandatory to obtain algorithmic scalability with large numbers of processes. Furthermore, PETSc provides non-overlapping domain decomposition methods. `PCNN` implements a basic balancing Neumann–Neumann method [63], while more advanced preconditioning techniques are available with `PCBDDC` [64], including adaptive selection of primal constraints [65], support for $H(\mathrm{curl})$ [66] and $H(\mathrm{div})$ [67] conforming finite elements, and isogeometric analysis [68]. For multigrid, `PCMG` provides a unified framework for geometric or algebraic preconditioners, which is used for the implementation of the smoothed-aggregation `PCGAMG` [55]. There are also interfaces to other well-known multigrid packages such as `PCHYPRE` [4] or `PCML` [69] from Trilinos. Of course, there are many other available domain decomposition preconditioners, either accessible through high-level libraries other than PETSc, e.g., FROSch [70] in Trilinos, or as stand-alone packages, e.g., BDDCML [71] or FEMPAR-BDDC [72].

HPDDM implements the same one-level overlapping Schwarz methods as PETSc as well as optimized Schwarz methods [73], which may be used in PETSc with `PCSetModifySubMatrices`, see [74]. It also provides support for defining robust two-level methods using the generalized eigenvalue problem on the overlap (GenEO) framework, for finite elements [75, 76] and boundary elements [77]. Results in this work present the first high-level access to GenEO from PETSc and its composable solver infrastructure [78]. We note that GenEO has been recently implemented in other libraries as well [79].

3.2. Interfacing HPDDM overlapping Schwarz methods in PETSc

`PCHPDDM`, the interface between HPDDM overlapping Schwarz methods and PETSc, is automatically registered since PETSc version 3.12 when configuring PETSc with the extra flag `--download-hpddm --download-slepc`. It is then possible to select the corresponding `PC` using the command line option `-pc_type hpddm`, or the routine `PCSetType(pc, PCHPDDM)`.

In this section, we are interested in constructing a preconditioner for a given coefficient matrix A. Without user intervention, `PCHPDDM` is strictly equivalent to `PCASM`. That is, if A is distributed among N processes, the action of the preconditioner M^{-1} is:

$$M^{-1} = \sum_{i=1}^{N} \tilde{R}_i^T (R_i A R_i^T)^{-1} R_i, \tag{3}$$

where $\{R_i\}_{i=1}^N$ are restriction operators that act on a global vector and return a local vector on each process, possibly with some overlap. $\{\tilde{R}_i\}_{i=1}^N$ are the same operators except that coefficients on the overlap are set to 0 [62]. The action of each $\{R_i A R_i^T\}_i^{-1}$ can be parameterized through a local `KSP`. When A is the discretization of a linear partial differential operator $\mathcal{L}$ on a domain Ω, it is extremely common to exhibit parallelism by distributing Ω on N processes, possibly with

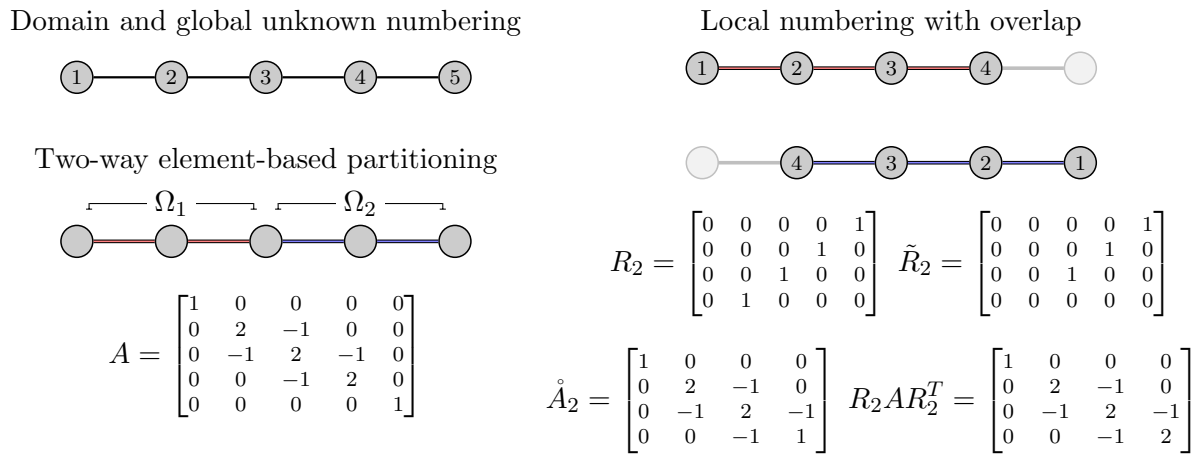


Figure 6: Notation and basics of overlapping Schwarz methods.

some overlap, and to construct the $\{R_i\}_{i=1}^N$ so that they map global unknowns to unknowns local to each process.

In addition to the local operators $\{R_i A R_i^T\}_{i=1}^N$, GenEO needs users to supply the local unassembled matrices $\{\mathring{A}_i\}_{i=1}^N$, also known as Neumann matrices, representing the discretization of $\mathcal{L}$ on the extended local subdomain with overlap, endowed with natural boundary conditions. In the case of the one-dimensional Poisson equation with homogeneous Dirichlet boundary conditions discretized using trilinear finite elements, an explicit representation of these operators is given figure 6 for a two-domain decomposition.

With extra Neumann information at hand, it is then possible to enrich the original one-level preconditioner from equation (3) using a spectral coarse grid built in the following way:

1. have SLEPc solve the local generalized eigenvalue problem concurrently:

$$\mathring{A}_i y_i = \lambda_i \tilde{R}_i R_i^T (R_i A R_i^T) R_i \tilde{R}_i^T y_i; \tag{4}$$

2. retrieve the ν_i smallest eigenpairs $\{y_{i_j}, \lambda_{i_j}\}_{j=1}^{\nu_i}$ and assemble a local deflation dense matrix $W_i = \begin{bmatrix} \tilde{R}_i R_i^T y_{i_1} & \cdots & \tilde{R}_i R_i^T y_{i_{\nu_i}} \end{bmatrix}$;
3. define a global deflation matrix $P = \begin{bmatrix} R_1^T W_1 & \cdots & R_N^T W_N \end{bmatrix}$ and a new two-level preconditioner using the Galerkin product of A and P:

$$M'^{-1}_{\text{additive}} = P \left(P^T A P\right)^{-1} P^T + \sum_{i=1}^{N} \tilde{R}_i^T (R_i A R_i^T)^{-1} R_i. \tag{5}$$

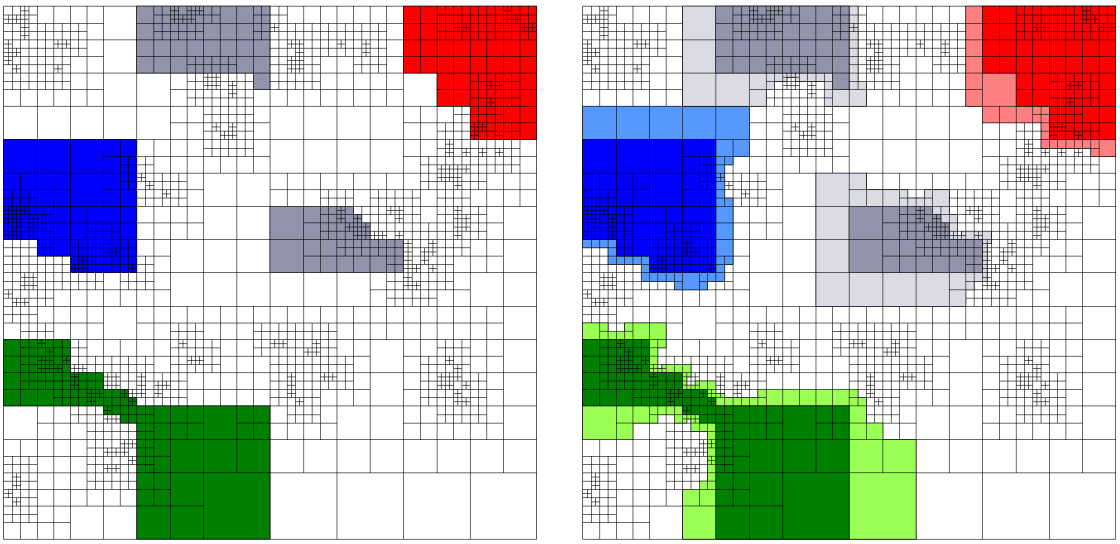
Note that $\{\tilde{R}_i R_i^T\}_{i=1}^N$ defines a partition of unity, i.e.,

$$\sum_{i=1}^{N} R_i^T \tilde{R}_i R_i^T R_i = I.$$

Such a construction of a two-level method yields a preconditioner with which it is possible to bound the condition number of the preconditioned operator. Thus, the convergence rate of a preconditioned Krylov method such as the conjugate gradient can be guaranteed a priori.

While step #3 is common to most domain decomposition or multigrid preconditioners, HPDDM takes advantage of the special block sparse structure of the deflation matrix P. Moreover, the size of the coarse operator P^TAP is much lower than the size of A, and coarsening is much more aggressive than when using algebraic or geometric multigrid methods. It is thus natural to remap the coarse operator on a subset of the processes used to decompose the original matrix A. In HPDDM, the coarse operator is computed and redistributed simultaneously. Interested readers are referred to the paper describing the technical details of the implementation [7]. Note that similar techniques have been proven beneficial for other types of solvers in PETSc [80].

In order to use GenEO from PETSc, the auxiliary operators $\{\mathring{A}_i\}_{i=1}^N$ and the corresponding local to global map of degrees of freedom, see figure 6, can be provided using the routine `PCHPDDMSetAuxiliaryMat`. In special cases where the matrix has been constructed using the PETSc discretization infrastructure, the relevant information is automatically computed by PETSc. Figure 7a (resp. figure 7b) is an example of a non-overlapping (resp. overlapping) decomposition using a non-conforming grid with `DMPLEX` [81]. The mesh has been constructed internally by `DMPLEX` using a random pattern of non-conforming 2:1 refinements and a space-filling curve distribution by p4est [6]. The required elements needed to properly assemble the Neumann operators in a conforming way are pictured with a lighter color. Other `DMType`s, including user-defined, may be supported by implementing the proper `DMCreateNeumannOverlap` callback. Support for the structured grid infrastructure (`DMDA`) could be easily added.



(a) Initial non-overlapping decomposition.

(b) Overlapping decomposition on which auxiliary operators $\{\mathring{A}_i\}_{i=1}^N$ are assembled.

Figure 7: Automatic generation of overlapping subdomains when using a `DMPLEX`. For clarity, only four subdomains, one of which is disconnected, are colored.

With the local matrix pencils available, new options are automatically registered such as `-pc_hpddm_levels_1_eps_nev` or `-pc_hpddm_levels_1_eps_threshold`, which may be used to customize the eigenpairs SLEPc will compute in step #2. The default `-eps_nev` value is 20, i.e., the dimension of the coarse space is $20 \times N$ under the assumption that all concurrent SLEPc solves

converge.

For performance, it is better to compute the same number of eigenvectors on each process, since in this case, the coarse operator will be assembled using symmetric or general blocked matrix formats. With usually large block sizes, this has consequent impacts on memory, floating-point, and message passing performances. In addition, standard eigensolvers such as `EPSKRYLOVSCHUR` from SLEPc, cannot inexpensively evaluate the exact number of eigenpairs in a given interval of $\mathbb{R}^+$. Instead, it is often more efficient to provide an upper bound on the number of eigenpairs.

The option `-pc_hpddm_coarse_p` can be used to provide the size of the subcommunicator that will be used to remap the coarse operator. With the default value of 1, the coarse matrix P^TAP is centralized on a single process. The action of the inverses in equation (5) can be parameterized through the `-pc_hpddm_coarse_ksp_type` and `-pc_hpddm_levels_1_ksp_type` options respectively.

3.2.1. Coarse corrections customization

Some other parameters may be adjusted to define how the one-level preconditioner and the coarse-grid operator interact. Besides the additive correction shown in equation (5), end-users may select more numerically efficient corrections [82, 83] as:

$$M'^{-1}_{\text{deflated}} = Q + M^{-1}\left(I - AQ\right) \quad \text{(default)}$$
$$M'^{-1}_{\text{balanced}} = Q + \left(I - AQ\right)M^{-1}\left(I - AQ\right),$$

with $Q = P\left(P^TAP\right)^{-1}P^T$ and M^{-1} defined in equation (3). This is parameterized using the option `-pc_hpddm_coarse_correction (deflated|additive|balanced)` or the routine `PCHPDDMSetCoarseCorrectionType`. All of the correction formulas can be applied to either a single vector or a block of multiple vectors optimally. Indeed, the one-level preconditioner M^{-1} is applied with either `PCApply` or `PCMatApply`, and the restriction–correction–interpolation operator Q is applied in a block fashion in HPDDM, and not column by column.

3.2.2. Extension of the GenEO framework

The local generalized eigenvalue problems equation (4) can also be adjusted. In addition to the local unassembled Neumann matrices $\{\mathring{A}_i\}_{i=1}^N$, users can prescribe additional local operators $\{B_i\}_{i=1}^N$, defined on the overlapping decomposition, via the routine `PCHPDDMSetRHSMat`(pc, B_i). In the case of local matrices with optimized Robin transmission conditions, the following generalized eigenvalue problems, called GenEO-2 in the literature [84], are then solved in each subdomain:

$$\mathring{A}_i y_i = \lambda_i B_i y_i.$$

This may be useful when dealing with nearly indefinite systems, e.g., nearly incompressible elasticity or the system of Stokes. Indeed, for such equations, one needs to compute a large number of eigenvectors from the classical GenEO eigenproblem equation (4) to generate a robust preconditioner. GenEO-2 alleviates this phenomenon. In the case where the local operators $\{B_i\}_{i=1}^N$ are the discrete mass matrices on the subdomain interfaces, the so-called Dirichlet-to-Neumann coarse operator is constructed. This has proven to be efficient for solving the heterogeneous Helmholtz equation [85].

3.2.3. Multilevel extension

In the previous sections, it was shown how to supply the required information to `PCHPDDM` in order to build robust two-level overlapping domain decomposition preconditioners. Since the

dimension of the coarse operator linearly depends on the number of subdomains of the fine level decomposition, switching to a multilevel scheme may alleviate the increasing cost of solving coarse linear systems. A multilevel extension of the GenEO framework has been recently proposed [86], with the advantage that the multilevel hierarchy can be automatically constructed without any additional information from the user.

`PCHPDDM` follows the same numbering of `PCBDDC`: the finest level is always numbered with 1, and the level index increases as the hierarchy is traversed, up until the coarsest level, whose solver options are prefixed by `coarse_`. In order to register an additional level $l' = l + 1$, the following conditions must be met at level l:

- there must be more than one subdomain;
- at least one local eigenvector must be computed per coarse subdomain.

For example, using $N = 16$ subdomains on the finest level, the options

```
-pc_hpddm_levels_1_eps_threshold 0.4 -pc_hpddm_coarse_p 8
```

will define a two-level method with the coarse operator being distributed among 8 processes, assuming that there is globally enough λ_i in equation (4) smaller than the prescribed threshold 0.4. Similarly, the options

```
-pc_hpddm_levels_1_eps_nev 10 -pc_hpddm_levels_2_p 8
-pc_hpddm_levels_2_eps_nev 10 -pc_hpddm_coarse_p 2
```

will define a three-level method with the second level distributed among 8 processes, while the coarsest level will be built using 10 eigenvalues per subdomain from the second level and remapped onto 2 processes. Additional examples for the solver customization are provided in section 3.3

3.2.4. PCHPDDM for non-overlapping domain decomposition

Because of the intrinsic nature of balancing domain decomposition methods, `PCNN` and `PCBDDC` must be supplied with a `MatIS` which stores local unassembled matrices and local to global mappings. These local matrices are equivalent to the $\{\mathring{A}_i\}_{i=1}^N$ defined section 3.2, assuming the domain decomposition is without overlap, see for example Ω_1 and Ω_2 from figure 6 in a finite element context. Since the assembled form of these operators can be reconstructed, it is possible to obtain the Dirichlet operators $\{R_i A R_i^T\}_{i=1}^N$ from section 3.2 as well. All tools needed by the GenEO framework are thus readily available. Note however that one-level overlapping Schwarz methods are known for converging slowly when there is no overlap. The proposed methodology is not strictly equivalent to the BDD-GenEO method [87], in which local Schur complements on subdomain interfaces are computed explicitly, and then used in concurrent dense generalized eigenvalue problems.

3.3. Applications and numerical results

3.3.1. System of elasticity

We first report on solving the three-dimensional system of linear elasticity with highly heterogeneous elastic modulus. Its strong formulation is given by:

$$\begin{aligned} \operatorname{div} \sigma(u) + f &= 0 \quad \text{in } \Omega, \\ u &= 0 \quad \text{on } \Gamma_D, \\ \sigma(u) \cdot n &= 0 \quad \text{on } \Gamma_N. \end{aligned} \tag{6}$$

The physical domain Ω is a beam of dimensions $[0,6] \times [0,1] \times [0,1]$. The Cauchy stress tensor σ is given by Hooke's law: it can be expressed in terms of Young's modulus E and Poisson's ratio ν,

$$\sigma_{ij}(u) = \begin{cases} 2\mu\varepsilon_{ij}(u) & i \neq j, \\ 2\mu\varepsilon_{ii}(u) + \lambda \mathrm{div}(u) & i = j, \end{cases}$$

where

$$\varepsilon_{ij}(u) = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_i} + \frac{\partial u_j}{\partial x_j}\right), \ \mu = \frac{E}{2(1+\nu)}, \text{ and } \lambda = \frac{E\nu}{1-2\nu}.$$

Γ_D is the subset of the boundary of Ω corresponding to $x = 0$. Γ_N is defined as the complementary of Γ_D with respect to the boundary of Ω. Equation (6) is discretized using trilinear finite elements resulting in 593×10^6 unknowns with approximately 45 nonzero coefficients per row in the discrete coefficient matrix. The physical domain Ω is decomposed in 13,824 subdomains using the automatic graph partitioner ParMETIS [88]. There are heterogeneities due to jumps in E and ν. The following discontinuous piecewise constant values are considered: $(E_1, \nu_1) = (2 \times 10^{11}, 0.25)$ in blue regions, while $(E_2, \nu_2) = (10^7, 0.45)$ in red regions, see figure 8. The code used to produce these results was borrowed from the original paper about the multilevel extension of GenEO [86] and it is available at https://github.com/prj-/aldaas2019multi.

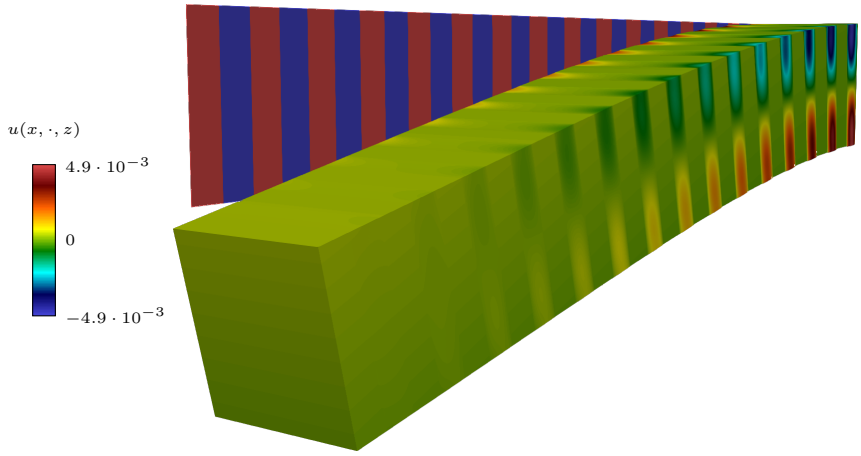


Figure 8: Deformed geometrical configuration of a 3D clamped beam subject to gravity. The striped plane in the background shows a cut of the resting configuration with the jumps in the coefficient $E_1 = 10^7$ and $E_2 = 2 \times 10^{11}$, aligned with those of ν.

The following preconditioners are compared:

1. `PCGAMG`;
2. `PCHPDDM` with an exact coarse solver;
3. `PCHPDDM` with inexact coarse solvers using GenEO multilevel extension.

The flexible GMRES is used and customized with the options `-ksp_gmres_modifiedgramschmidt -ksp_gmres_restart 50 -ksp_type fgmres`. The options specific to each solver described above are given below.

```
# cf. item 1
-pc_type gamg
-prefix_push pc_gamg_
 -threshold 0.03
 -square_graph 4
 -sym_graph true
 -asm_use_agg true
 -repartition true
-prefix_pop
-prefix_push mg_levels_
 -pc_asm_overlap 0
 -sub_pc_type cholesky
-prefix_pop
-prefix_push mg_coarse_
 -pc_type redundant
 -redundant_pc_type cholesky
-prefix_pop
```

```
# cf. item 2
-pc_type hpddm
-prefix_push pc_hpddm_
 -prefix_push levels_1_
  -pc_type     asm
  -eps_nev     40
  -sub_pc_type cholesky
  -sub_pc_factor_mat_solver_type mumps
  -st_pc_factor_mat_solver_type mumps
 -prefix_pop
 -prefix_push coarse_
  -p 24
  -pc_factor_mat_solver_type mkl_cpardiso
 -prefix_pop
 -define_subdomains
 -has_neumann
-prefix_pop
```

By default, the coarse problem in `PCHPDDM` is solved using an exact LU or Cholesky factorization, in this case using 24 processes and MKL CPARDISO. Switching to an inexact solver using the multilevel extension of GenEO, cf. item 3, is straightforward.

```
# cf. item 3
-prefix_push pc_hpddm_levels_2_
 -p           M
 -ksp_type    gmres
 -ksp_rtol    1.0e-2
 -ksp_pc_side right
 -pc_type     asm
 -eps_nev     80
 -sub_pc_type cholesky
 -sub_pc_factor_mat_solver_type mkl_pardiso
 -st_pc_factor_mat_solver_type mkl_pardiso
-prefix_pop
```

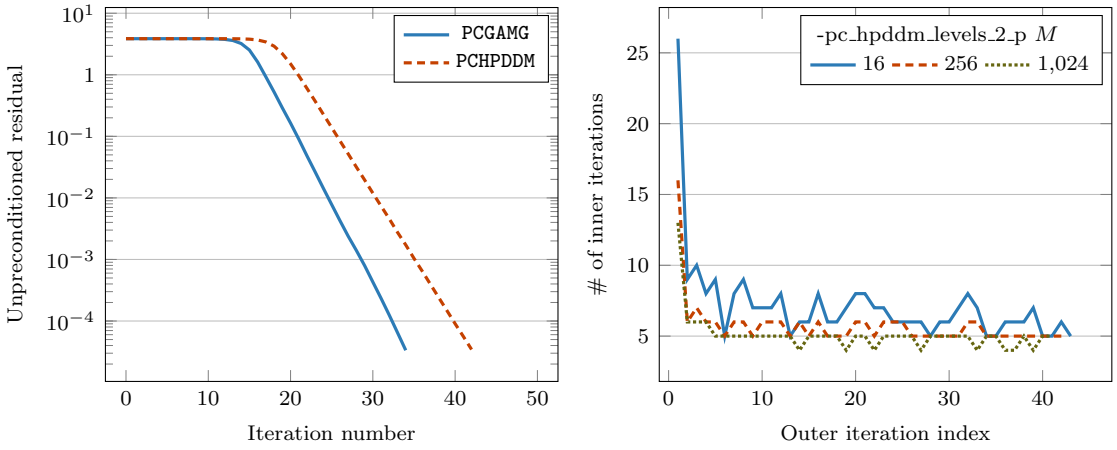
In this third configuration, M is the number of subdomains used to define the second-level domain decomposition, which is an aggregation of first-level subdomains.

Considering the parametrization used for the `PCHPDDM` solvers, items 2 and 3 yield the same number of outer iterates. In figure 9a, the convergence history of `PCGAMG` and `PCHPDDM` are reported. In figure 9b, the number of inner coarse iterations are reported for various values of M. In the case of a two-level method (item 2) this number is equal to one and is thus not reported. Note that in this test case, `PCGAMG` is faster than all `PCHPDDM` alternatives. On the one hand, for `PCGAMG`, the setup phase takes 70 seconds, while the solution phase requires 19 seconds. On the other hand, the fastest `PCHPDDM` configuration, which corresponds to the configuration with 256 second-level subdomains `----`, requires 64 seconds to setup and 36 seconds for the solution of the linear system. Out of the 64 seconds needed for `PCHPDDM` setup, 31 are spent in SLEPc `EPSSolve` for solving GenEO at each level but the coarsest. With respect to coarsening, `PCGAMG` has an operator complexity of 1.501, while for `PCHPDDM`, it is 1.015.

3.3.2. Liouville–Bratu–Gelfand equation

The strong formulation of this nonlinear problem is given by:

$$-\nabla \cdot (\kappa \nabla u) - 6.2e^u = 0, \tag{7}$$



(a) Numerical comparison of `PCGAMG` and `PCHPDDM`.

(b) Number of inner coarse iterations for various values of M, the number of second-level subdomains.

Figure 9: Convergence histories of `PCGAMG` and `PCHPDDM` for solving the 3D system of linear elasticity. For `PCHPDDM`, a comparison of the number of inner coarse iterations is displayed when using an inexact method, cf. item 3.

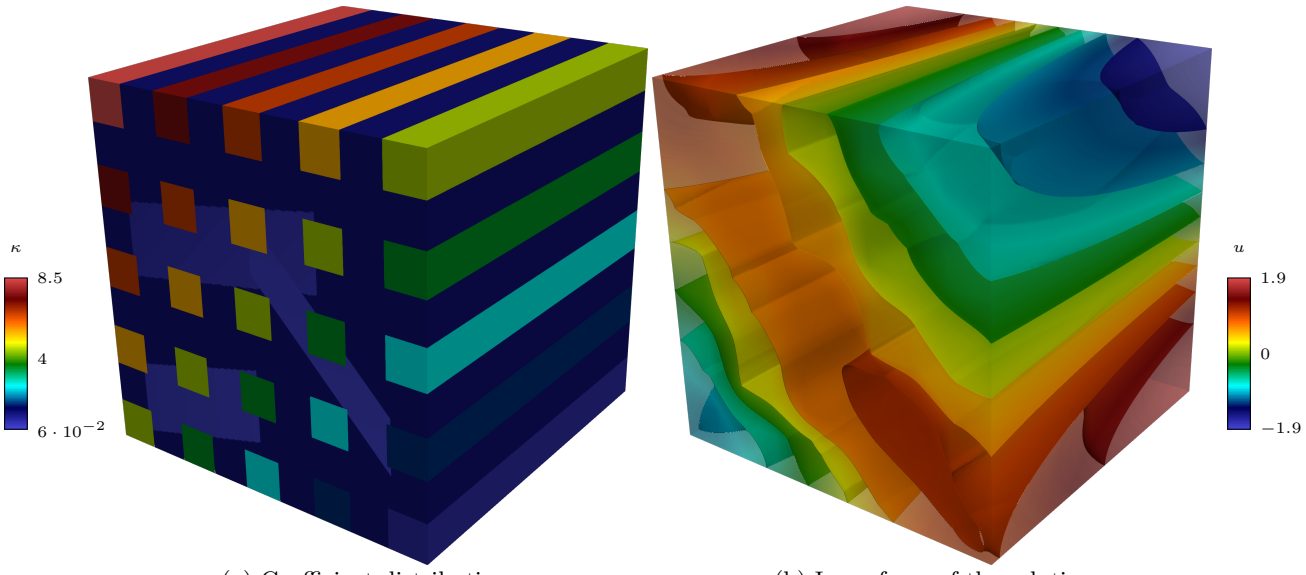
where κ is a heterogeneous coefficient distribution, see figure 10a. This equation may model the temperature distribution in combustion models. Here, it is merely used to test `PCHPDDM` in the context of solving successive linearized equations. The physical domain Ω is the unit cube. It is decomposed in 10,272 subdomains. Equation (7) is discretized using second-order Lagrange finite elements. The number of unknowns is 217×10^6, with approximately 29 nonzero coefficients per row. It is solved using `SNES`, with `PCHPDDM` being the preconditioner for solving each linearized system, see figure 10b. The GenEO framework has seldom been used in the context of nonlinear problems, but here, it is shown that it can solve the linearized equations from equation (7) in few iterates. There are two strategies to define the preconditioner:

- whenever the Jacobian is being updated in the `SNESSetJacobian` function, new unassembled Neumann matrices at the current linearization point can be updated as well and supplied via `PCHPDDMSetAuxiliaryMat`
- by reusing the `PCHPDDM` hierarchy assembled when solving the first linearized system, using the `SNESSetLagPreconditioner` option.

Since the system is not too stiff here, both strategies lead to the same convergence history, which is displayed in figure 11a. The complete set of options is given in figure 11b. In case the preconditioner is being rebuilt at each of the four linearization steps, 81.1 s are spent in `PCSetUp` and 27.7 s in `KSPSolve`. Setting up the preconditioner only once is in this scenario a compelling way of amortizing this cost. Results can be reproduced using the FreeFEM script `bratu.edp` available at https://github.com/prj-/jolivet2020petsc.

3.3.3. Using PCHPDDM as a coarse grid solver for PCBDDC

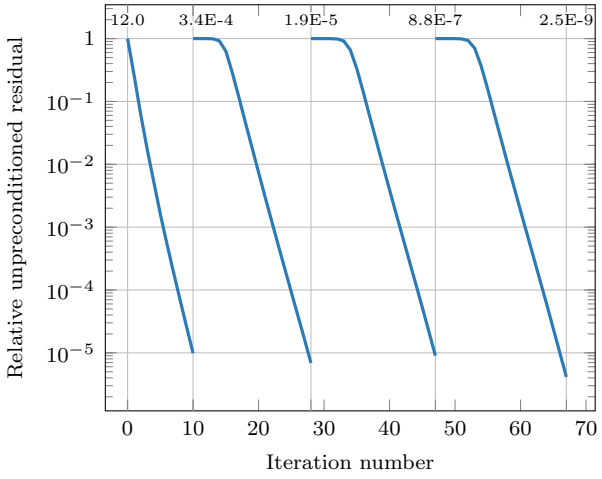
A last problem shows how one may switch between a multilevel BDDC solver to a two-level BDDC solver using `PCHPDDM` for solving the coarse problem. The problem is generated using the MFEM [89] definite Maxwell example available at https://github.com/mfem/mfem/blob/master/examples/petsc/ex3p.cpp. The strong formulation of the continuous problem is given



(a) Coefficient distribution.

(b) Isosurfaces of the solution.

Figure 10: Solving the Liouville–Bratu–Gelfand equation using `SNESNEWTONLS` and `PCHPDDM`.



(a) Convergence histories of the four linearized systems. At the top of each curve, for the first iterate, the value of the function norm is displayed.

```
-snes_type              newtonls
-snes_linesearch_type basic
-ksp_pc_side            right
-pc_type                hpddm
-prefix_push pc_hpddm_
 -prefix_push levels_1_
  -pc_type      asm
  -eps_nev      10
  -sub_pc_type cholesky
  -sub_pc_factor_mat_solver_type mumps
 -prefix_pop
 -coarse_p 128
 -define_subdomains
 -has_neumann
-prefix_pop
```

(b) `SNES`, `KSP`, and `PC` options for solving equation (7)

Figure 11: Numerical performance of `PCHPDDM` in a nonlinear context for solving the Liouville–Bratu–Gelfand equation. Default PETSc tolerances are used: 10^{-5} decrease of relative residuals for linear solves, 10^{-8} decrease of function norms for nonlinear solves.

by:

$$\begin{aligned} \nabla \times (\nabla \times E) + E &= 0 \quad \text{in } \Omega, \\ E \times n &= (1 + 16\pi^2) \begin{bmatrix} \sin 4\pi y & \sin 4\pi z & \sin 4\pi x \end{bmatrix}^T \text{on } \partial\Omega. \end{aligned} \tag{8}$$

It is discretized using first order Nédélec finite elements and `PCBDDC` is automatically customized to obtain a stable method with these elements [66]. The number of unknowns is 5.6×10^6. The following command line options are used.

```
$ mpirun -n 512 ./ex3p -m ../../data/fichera.mesh -f 4 --petscopts rc_ex3p_bddc --nonoverlapping
```

The option file shows how `PCHPDDM` may be composed into `PCBDDC`.

```
$ cat rc_ex3p_bddc
-ksp_type      fgmres
-ksp_norm_type unpreconditioned
-ksp_rtol      1.0e-8

-prefix_push pc_bddc_
 -use_deluxe_scaling
 -levels             1
 -adaptive_threshold 2.0
 -coarsening_ratio   8

 -neumann_pc_factor_mat_solver_type   mumps
 -dirichlet_pc_factor_mat_solver_type mumps
# continue on the right column
```

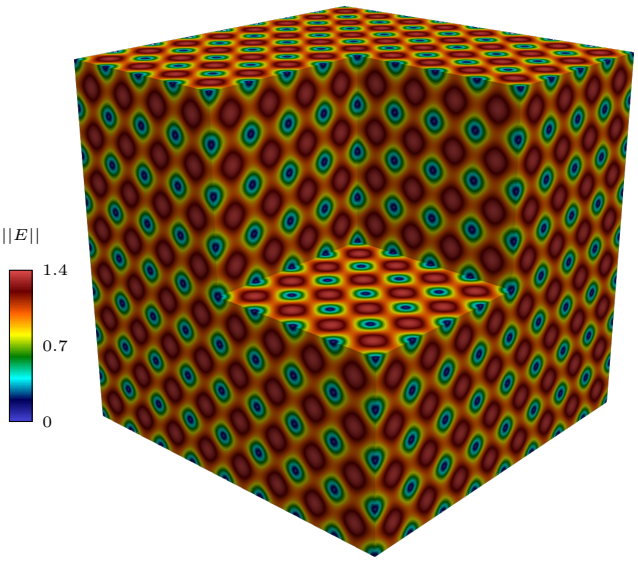
```
# continued from the left column
 -prefix_push coarse_
  -ksp_converged_reason
  -ksp_type      gmres
  -ksp_max_it    100
  -ksp_rtol      1.0e-1
  -pc_type       hpddm
  -ksp_norm_type preconditioned
  -prefix_push pc_hpddm_
   -levels_1_pc_type     asm
   -levels_1_eps_nev     10
   -levels_1_sub_pc_type cholesky
   -coarse_pc_type       cholesky
  -prefix_pop
 -prefix_pop
-prefix_pop
```

Deluxe scaling is used to build an adaptive second level with `PCBDDC`. On this second level with 16,284 unknowns, new subdomains are built by aggregating the coarse element matrices of 8 fine-level subdomains. The BDDC coarse problem composed of $\frac{512}{8} = 64$ subdomains is then solved using `PCHPDDM`, which itself builds another adaptive level using 10 local eigenvectors per subdomain. The coarsest level is aggregated on a single process and solved using an exact Cholesky factorization. The magnitude of the electric field is plotted in figure 12a. The convergence history of the solver is shown in figure 12b. The outer solver reaches the prescribed convergence tolerance in 17 iterations ----. The number of inner iterations for solving the BDDC coarse problem is reported as well ——.

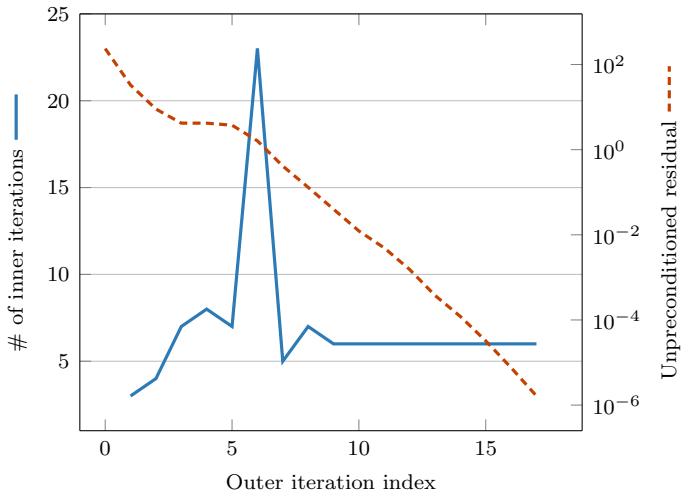
4. Conclusion and perspectives

In this paper, we presented the interface between PETSc and HPDDM and discussed the new Krylov and overlapping Schwarz methods, providing several numerical examples and describing various runtime options. Overall, the interface paves the way for having recycling and block Krylov methods plus robust overlapping Schwarz methods using the GenEO framework in PETSc.

Concerning PETSc, the infrastructure for dealing with blocks of vectors has been laid out. However, only a little fraction of the built-in matrix types and preconditioners can currently handle them efficiently, and additional storage formats may be considered to maximize performance.



(a) Magnitude of the electric field.



(b) Convergence history of `PCBDDC` using `PCHPDDM` as a coarse solver.

Figure 12: MFEM example ex3p for solving the Fichera corner problem.

Concerning SLEPc, we plan to implement new solvers or extend existing ones to further exploit block solve primitives, for instance, block Krylov–Schur [90]. The current code for Arnoldi iterations would then handle block Arnoldi iterations, and shift-and-invert solves would operate on the whole block instead of column by column.

Concerning HPDDM, its integration inside PETSc offers a much-needed flexibility compared to the standard implementation. Indeed, it is now possible to decide at runtime many solvers parameters that are instead determined at compile-time in a stand-alone HPDDM implementation.

The interface between PETSc and HPDDM has room for future improvements. First, HPDDM handles mixed-precision, e.g., using single-precision scalars for assembling coarse operators while using double-precision scalars at the fine level. The topic of mixed-precision is under scrutiny for the next major overhaul of PETSc. Second, HPDDM does not handle GPU efficiently currently, but the interface with PETSc provides a very thorough testbed with `Mat` and `PC` implementations that do exploit GPU.

Acknowledgments

The authors would like to thank S. Balay, J. Brown, V. Hapla, M. Knepley, and B. Smith for reviewing the successive merge requests in PETSc repository and for their feedback on this manuscript. This work was granted access to the GENCI-sponsored HPC resources of:

- TGCC@CEA under allocation A0070607519;
- IDRIS@CNRS under allocation AP010611780.

Jose E. Roman was supported by the Spanish Agencia Estatal de Investigación (AEI) under project SLEPc-DA (PID2019-107379RB-I00).

Appendix A. Code reproducibility

In order to reproduce the various results from this paper, a short how-to is provided in this appendix to get a minimalist functioning set of required libraries. First, PETSc version 3.14.2 can be downloaded at https://gitlab.com/petsc/petsc/-/archive/v3.14.2/petsc-v3.14.2.zip. While some adjustments may be needed to ensure that the proper MPI implementation and BLAS/LAPACK libraries are used by PETSc build system, the configure line should be somehow similar to:

```
./configure --download-hypre --download-metis
            --download-slepc --download-hpddm --download-mfem
            --with-scalar-type=real PETSC_ARCH=real-build
```

MFEM will then be available for reproducing results from section 3.3.3. As a stand-alone library, PETSc can be used to reproduce results from sections 2.3.1 and 2.3.2. After building PETSc with real scalars, it must also be built with complex scalars.

```
./configure --with-metis-dir=real-build
            --download-slepc --download-hpddm
            --with-scalar-type=complex PETSC_ARCH=complex-build
```

Eventually, one can download FreeFEM version 4.7-1 at https://github.com/FreeFem/FreeFem-sources/archive/v4.7-1.zip and use the following configure line:

```
./configure --with-petsc=${PETSC_DIR}/real-build/lib
            --with-petsc_complex=${PETSC_DIR}/complex-build/lib
```

After building FreeFEM, results from sections 2.3.3, 2.3.4, 3.3.1 and 3.3.2 can be reproduced.

References

[1] R. Bartlett, I. Demeshko, T. Gamblin, G. Hammond, M. A. Heroux, J. Johnson, A. Klinvex, X. S. Li, L. C. McInnes, J. D. Moulton, D. Osei-Kuffuor, J. Sarich, B. F. Smith, J. Willenbring, U. M. Yang, xSDK foundations: Toward an extreme-scale scientific software development kit, Supercomputing Frontiers and Innovations 4 (1) (2017).
URL https://xsdk.info

[2] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, PETSc web page (2020).
URL http://www.mcs.anl.gov/petsc

[3] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, A. Dener, V. Eijkhout, W. D. Gropp, D. Karpeyev, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, H. Zhang, PETSc users manual, Tech. Rep. ANL-95/11 - Revision 3.13, Argonne National Laboratory (2020).

[4] R. Falgout, U. M. Yang, *hypre*: A library of high performance preconditioners, Computational Science—ICCS 2002 (2002) 632–641.
URL https://www.llnl.gov/casc/hypre

[5] H. Si, TetGen: A quality tetrahedral mesh generator and 3D Delaunay triangulator, Tech. Rep. 13 (2013).
URL http://wias-berlin.de/software/tetgen

[6] C. Burstedde, L. C. Wilcox, O. Ghattas, p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees, SIAM Journal on Scientific Computing 33 (3) (2011) 1103–1133.
URL http://www.p4est.org

[7] P. Jolivet, F. Hecht, F. Nataf, 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.
[8] P. Jolivet, P.-H. Tournier, Block iterative methods and recycling for improved scalability of linear solvers, in: Proceedings of the 2016 International Conference for High Performance Computing, Networking, Storage and Analysis, SC16, IEEE, 2016.
[9] T. Hoefler, R. Belli, Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results, in: Proceedings of the 2015 International Conference for High Performance Computing, Networking, Storage and Analysis, SC15, 2015.
[10] V. Hernandez, J. E. Roman, V. Vidal, SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems, ACM Transactions on Mathematical Software 31 (3) (2005) 351–362.
URL https://slepc.upv.es
[11] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Van der Vorst, Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, 1994.
[12] Y. Saad, Iterative Methods for Sparse Linear Systems, 2nd Edition, SIAM, 2003.
[13] Y. Saad, M. H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM Journal on Scientific and Statistical Computing 7 (3) (1986) 856–869.
[14] M. R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, Journal of Research of the National Bureau of Standards 49 (6) (1952) 409–436.
[15] H. A. Van der Vorst, Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems, SIAM Journal on Scientific and Statistical Computing 13 (2) (1992) 631–644.
[16] A. H. Baker, E. R. Jessup, T. Manteuffel, A technique for accelerating the convergence of restarted GMRES, SIAM Journal on Matrix Analysis and Applications 26 (4) (2005) 962–984.
[17] J. Erhel, K. Burrage, B. Pohl, Restarted GMRES preconditioned by deflation, Journal of Computational and Applied Mathematics 69 (2) (1996) 303–318.
[18] D. N. Wakam, F. Pacull, Memory efficient hybrid algebraic solvers for linear systems arising from compressible flows, Computers & Fluids 80 (2013) 158–167.
[19] Y. Saad, A flexible inner-outer preconditioned GMRES algorithm, SIAM Journal on Scientific Computing 14 (2) (1993) 461–469.
[20] S. C. Eisenstat, H. C. Elman, M. H. Schultz, Variational iterative methods for nonsymmetric systems of linear equations, SIAM Journal on Numerical Analysis 20 (2) (1983) 345–357.
[21] M. L. Parks, E. de Sturler, G. Mackey, D. D. Johnson, S. Maiti, Recycling Krylov subspaces for sequences of linear systems, SIAM Journal on Scientific Computing 28 (5) (2006) 1651–1674.
[22] M. H. Gutknecht, Block Krylov space methods for linear systems with multiple right-hand sides: An introduction, in: A. Siddiqui, I. Duff, O. Christensen (Eds.), Modern Mathematical Models, Methods and Algorithms for Real World Systems, 2006, pp. 420–447.
[23] A. Frommer, K. Lund, D. B. Szyld, Block Krylov subspace methods for functions of matrices, Electronic Transactions on Numerical Analysis 47 (2017) 100–126.
[24] 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, S. Semenov, Microwave tomographic imaging of cerebrovascular accidents by using high-performance computing, Parallel Computing 85 (2019) 88–97.
[25] V. Kalantzis, A. C. I. Malossi, C. Bekas, A. Curioni, E. Gallopoulos, Y. Saad, A scalable iterative dense linear system solver for multiple right-hand sides in data analytics, Parallel Computing 74 (2018) 136–153.
[26] A. V. Knyazev, Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method, SIAM Journal on Scientific Computing 23 (2) (2001) 517–541.
[27] H. Calandra, S. Gratton, J. Langou, X. Pinel, X. Vasseur, Flexible variants of block restarted GMRES methods with application to geophysics, SIAM Journal on Scientific Computing 34 (2) (2012) A714–A736.
[28] T. Sakurai, H. Tadano, Y. Kuramashi, Application of block Krylov subspace algorithms to the Wilson–Dirac equation with multiple right-hand sides in lattice QCD, Computer Physics Communications 181 (1) (2010) 113–117.
[29] 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) 31 (3) (2005) 397–423.
URL https://trilinos.github.io
[30] E. Bavier, M. Hoemmen, S. Rajamanickam, H. Thornquist, Amesos2 and Belos: Direct and iterative solvers for

large sparse linear systems, Scientific Programming 20 (3) (2012) 241–255.
[31] Y. Notay, Flexible conjugate gradients, SIAM Journal on Scientific Computing 22 (4) (2000) 1444–1460.
[32] L. M. Carvalho, S. Gratton, R. Lago, X. Vasseur, A flexible generalized conjugate residual method with inner orthogonalization and deflated restarting, SIAM Journal on Matrix Analysis and Applications 32 (4) (2011) 1212–1235.
[33] D. P. O'Leary, The block conjugate gradient algorithm and related methods, Linear Algebra and its Applications 29 (1980) 293–322.
[34] H. Ji, Y. Li, A breakdown-free block conjugate gradient method, BIT Numerical Mathematics 57 (2) (2017) 379–403.
[35] Y. Saad, M. Yeung, J. Erhel, F. Guyomarc'h, A deflated version of the conjugate gradient algorithm, SIAM Journal on Scientific Computing 21 (5) (2000) 1909–1926.
[36] A. Stathopoulos, A. Abdel-Rehim, K. Orginos, Deflation for inversion with multiple right-hand sides in QCD, in: Journal of Physics: Conference Series, Vol. 180, IOP Publishing, 2009.
[37] K. M. Soodhalter, D. B. Szyld, F. Xue, Krylov subspace recycling for sequences of shifted linear systems, Applied Numerical Mathematics 81 (2014) 105–118.
[38] P. Amestoy, I. Duff, J.-Y. L'Excellent, J. Koster, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM Journal on Matrix Analysis and Applications 23 (1) (2001) 15–41.
URL http://mumps.enseeiht.fr
[39] A. Stathopoulos, K. Wu, A block orthogonalization procedure with constant synchronization requirements, SIAM Journal on Scientific Computing 23 (6) (2002) 2165–2182.
[40] M. H. Gutknecht, T. Schmelzer, Updating the QR decomposition of block tridiagonal and block Hessenberg matrices, Applied Numerical Mathematics 58 (6) (2008) 871–883.
[41] NVIDIA, cuSPARSE web page, https://docs.nvidia.com/cuda/cusparse (2020).
[42] W. Boukaram, G. Turkiyyah, D. Keyes, Hierarchical matrix operations on GPUs: Matrix–vector multiplication and compression, ACM Transactions on Mathematical Software 45 (2019).
[43] I. Ambartsumyan, W. Boukaram, T. Bui-Thanh, O. Ghattas, D. Keyes, G. Stadler, G. Turkiyyah, S. Zampini, Hierarchical matrix approximations of Hessians arising in inverse problems governed by PDEs, SIAM Journal on Scientific Computing 42 (5) (2020) A3397–A3426.
[44] T. A. Davis, Algorithm 832: UMFPACK—an unsymmetric-pattern multifrontal method, ACM Transactions on Mathematical Software 30 (2) (2004) 196–199.
[45] Intel, MKL web page, https://software.intel.com/content/www/us/en/develop/tools/math-kernel-library.html (2020).
[46] X. S. Li, An overview of SuperLU: Algorithms, implementation, and user interface, ACM Transactions on Mathematical Software 31 (3) (2005) 302–325.
[47] X. S. Li, J. Demmel, SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems, ACM Transactions on Mathematical Software 29 (2) (2003) 110–140.
[48] J. Poulson, B. Marker, R. A. Van de Geijn, J. R. Hammond, N. A. Romero, Elemental: A new framework for distributed memory dense matrix computations, ACM Transactions on Mathematical Software 39 (2) (2013).
[49] A. Heinecke, G. Henry, M. Hutchinson, H. Pabst, LIBXSMM: Accelerating small matrix multiplications by runtime code generation, in: Proceedings of the 2016 International Conference for High Performance Computing, Networking, Storage and Analysis, SC16, IEEE, 2016.
URL https://github.com/hfp/libxsmm
[50] F. Hecht, New development in FreeFem++, Journal of Numerical Mathematics 20 (3-4) (2012) 251–266.
URL http://freefem.org
[51] J. Moulin, P. Jolivet, O. Marquet, Augmented Lagrangian preconditioner for large-scale hydrodynamic stability analysis, Computer Methods in Applied Mechanics and Engineering 351 (2019) 718–743.
URL https://github.com/prj-/moulin2019al
[52] G. W. Stewart, A Krylov–Schur algorithm for large eigenproblems, SIAM Journal on Matrix Analysis and Applications 23 (3) (2002) 601–614.
[53] M. Benzi, M. A. Olshanskii, Z. Wang, Modified augmented Lagrangian preconditioners for the incompressible Navier–Stokes equations, International Journal for Numerical Methods in Fluids 66 (4) (2011) 486–508.
[54] T. Sakurai, H. Sugiura, A projection method for generalized eigenvalue problems using numerical integration, Journal of Computational and Applied Mathematics 159 (1) (2003) 119–128.
[55] M. Adams, H. H. Bayraktar, T. M. Keaveny, P. Papadopoulos, Ultrascalable implicit finite element analyses in solid mechanics with over a half a billion degrees of freedom, in: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing, SC04, IEEE Computer Society, 2004, pp. 34:1–34:15.

[56] B. F. Smith, P. Bjørstad, W. D. Gropp, Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, 2004.
[57] V. Dolean, P. Jolivet, F. Nataf, An Introduction to Domain Decomposition Methods: Algorithms, Theory and Parallel Implementation, SIAM, 2015.
[58] A. Toselli, O. B. Widlund, Domain decomposition methods: algorithms and theory, Vol. 34 of Series in Computational Mathematics, Springer, 2005.
[59] C. Pechstein, Finite and Boundary Element Tearing and Interconnecting Solvers for Multiscale Problems, Vol. 90 of Lecture Notes in Computational Science and Engineering, Springer, 2012.
[60] A. Quarteroni, A. Valli, Domain Decomposition Methods for Partial Differential Equations, Vol. 10, Clarendon Press, 1999.
[61] T. Mathew, Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations, Vol. 61, Springer Science & Business Media, 2008.
[62] X.-C. Cai, M. Sarkis, A restricted additive Schwarz preconditioner for general sparse linear systems, SIAM Journal on Scientific Computing 21 (2) (1999) 792–797.
[63] J. Mandel, Balancing domain decomposition, Communications in Numerical Methods in Engineering 9 (3) (1993) 233–241.
[64] S. Zampini, PCBDDC: A class of robust dual–primal methods in PETSc, SIAM Journal on Scientific Computing 38 (5) (2016) S282–S306.
URL https://www.mcs.anl.gov/petsc/petsc-master/docs/manualpages/PC/PCBDDC.html
[65] C. Pechstein, C. R. Dohrmann, A unified framework for adaptive BDDC, Electronic Transactions on Numerical Analysis 46 (2017) 273–336.
[66] S. Zampini, P. Vassilevski, V. Dobrev, T. Kolev, Balancing domain decomposition by constraints algorithms for curl-conforming spaces of arbitrary order, in: International Conference on Domain Decomposition Methods, Springer, 2017, pp. 103–116.
[67] D.-S. Oh, O. B. Widlund, S. Zampini, C. Dohrmann, BDDC algorithms with deluxe scaling and adaptive selection of primal constraints for Raviart–Thomas vector fields, Mathematics of Computation 87 (310) (2018) 659–692.
[68] L. B. Da Veiga, L. F. Pavarino, S. Scacchi, O. B. Widlund, S. Zampini, Isogeometric BDDC preconditioners with deluxe scaling, SIAM Journal on Scientific Computing 36 (3) (2014) A1118–A1139.
[69] M. W. Gee, C. M. Siefert, J. J. Hu, R. S. Tuminaro, M. G. Sala, ML 5.0 smoothed aggregation user's guide, Tech. Rep. SAND2006-2649, Sandia National Laboratories (2006).
URL https://trilinos.github.io/ml.html
[70] A. Heinlein, A. Klawonn, O. Rheinbach, A parallel implementation of a two-level overlapping Schwarz method with energy-minimizing coarse space based on Trilinos, SIAM Journal on Scientific Computing 38 (6) (2016) C713–C747.
[71] J. Šístek, J. Mandel, B. Sousedík, P. Burda, Parallel implementation of multilevel BDDC, in: Numerical Mathematics and Advanced Applications 2011, Springer, 2013, pp. 681–689.
URL http://users.math.cas.cz/~sistek/software/bddcml.html
[72] S. Badia, A. F. Martín, J. Principe, A highly scalable parallel implementation of balancing domain decomposition by constraints, SIAM Journal on Scientific Computing 36 (2) (2014) C190–C218.
[73] M. J. Gander, Optimized Schwarz Methods, SIAM Journal on Numerical Analysis 44 (2) (2006) 699–731.
[74] M. J. Gander, S. Van Criekingen, New coarse corrections for optimized restricted additive Schwarz using PETSc, in: International Conference on Domain Decomposition Methods, Springer, 2019.
[75] N. Spillane, V. Dolean, P. Hauret, F. Nataf, C. Pechstein, R. Scheichl, A robust two-level domain decomposition preconditioner for systems of PDEs, Comptes Rendus Mathématique 349 (23) (2011) 1255–1259.
[76] N. Spillane, V. Dolean, P. Hauret, F. Nataf, C. Pechstein, R. Scheichl, Abstract robust coarse spaces for systems of PDEs via generalized eigenproblems in the overlaps, Numerische Mathematik 126 (4) (2013) 741–770.
[77] P. Marchand, X. Claeys, P. Jolivet, F. Nataf, P.-H. Tournier, Two-level preconditioning for h-version boundary element approximation of hypersingular operator with GenEO, Numerische Mathematik 146 (2020) 597–628.
[78] J. Brown, M. G. Knepley, D. A. May, L. Curfman McInnes, B. F. Smith, Composable linear solvers for multiphysics, in: 2012 11th International Symposium on Parallel and Distributed Computing, IEEE, 2012, pp. 55–62.
[79] R. Butler, T. Dodwell, A. Reinarz, A. Sandhu, R. Scheichl, L. Seelinger, High-performance Dune modules for solving large-scale, strongly anisotropic elliptic problems with applications to aerospace composites, Computer Physics Communications 249 (2020).
[80] D. A. May, P. Sanan, K. Rupp, M. G. Knepley, B. F. Smith, Extreme-scale multigrid components within PETSc,

in: Proceedings of the Platform for Advanced Scientific Computing Conference, 2016.
[81] M. G. Knepley, D. A. Karpeev, Mesh algorithms for PDE with Sieve I: Mesh distribution, Scientific Programming 17 (3) (2009) 215–230.
[82] J. Tang, R. Nabben, C. Vuik, Y. Erlangga, Comparison of two-level preconditioners derived from deflation, domain decomposition and multigrid methods, Journal of Scientific Computing 39 (3) (2009) 340–370.
[83] P. Bastian, G. Wittum, W. Hackbusch, Additive and multiplicative multi-grid—a comparison, Computing 60 (4) (1998) 345–364.
[84] R. Haferssas, P. Jolivet, F. Nataf, An additive Schwarz method type theory for Lions's algorithm and a symmetrized optimized restricted additive Schwarz method, Journal on Scientific Computing 39 (4) (2017) A1345–A1365.
[85] L. Conen, V. Dolean, R. Krause, F. Nataf, A coarse space for heterogeneous Helmholtz problems based on the Dirichlet-to-Neumann operator, Journal of Computational and Applied Mathematics 271 (2014) 83–99.
[86] H. Al Daas, L. Grigori, P. Jolivet, P.-H. Tournier, A multilevel Schwarz preconditioner based on a hierarchy of robust coarse spaces, Journal of Scientific Computing (2019) submitted for publication.
URL https://github.com/prj-/aldaas2019multi
[87] N. Spillane, D. J. Rixen, Automatic spectral coarse spaces for robust finite element tearing and interconnecting and balanced domain decomposition algorithms, International Journal for Numerical Methods in Engineering 95 (11) (2013) 953–990.
[88] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM Journal on Scientific Computing 20 (1) (1998) 359–392.
URL http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview
[89] R. Anderson, J. Andrej, A. Barker, J. Bramwell, J.-S. Camier, J. Cerveny, V. Dobrev, Y. Dudouit, A. Fisher, T. Kolev, W. Pazner, M. Stowell, V. Tomov, I. Akkerman, J. Dahm, D. Medina, S. Zampini, MFEM: A modular finite element methods library, Computers & Mathematics with Applications 81 (2021) 42–74.
URL http://mfem.org
[90] Y. Zhou, Y. Saad, Block Krylov–Schur method for large symmetric eigenvalue problems, Numerical Algorithms 47 (4) (2008) 341–359.