Horizon 2020
European Union funding
for Research & Innovation

E-Infrastructures

H2020-INFRAEDI-2018-1

INFRAEDI-2-2018: Centres of Excellence on HPC

EoCoE-II

Energy oriented Center of Excellence:

toward exascale for energy

Grant Agreement Number: INFRAEDI-824158

D3.2

Preliminary results and performance evaluation of Linear Algebra solvers

Project and Deliverable Information Sheet

EoCoE	Project Ref:	INFRAEDI-824158
	Project Title:	Energy oriented Centre of Excellence
	Project Web Site:	http://www.eocoe.eu
	Deliverable ID:	D3.2
	Lead Beneficiary:	CNR
	Contact:	Pasqua D'Ambra
	Contact's e-mail:	pasqua.dambra@cnr.it
	Deliverable Nature:	Report
	Dissemination Level:	PU*
	Contractual Date of Delivery:	M18 30/06/2020
	Actual Date of Delivery:	M17 31/05/2020
	EC Project Officer:	Evangelia Markidou

* - The dissemination level are indicated as follows: PU – Public, CO – Confidential, only for members of the consortium (including the Commission Services) CL – Classified, as referred to in Commission Decision 2991/844/EC.

Document Control Sheet

Document	Title :	Preliminary results and performance evaluation of LA solvers
	ID :	D3.2
	Available at:	http://www.eocoe.eu
	Software tool:	LaTeX
Authorship	Written by:	Pasqua D'Ambra, Fabio Durastante
	Contributors:	Alfredo Buttari, Salvatore Filippone, Luc Giraud, Carola Kruse, Martin Kühn, Herbert Owen, Yvan Notay, Berenice Vallier
	Reviewed by:	Herbert Owen, Sebastian Luehrs

Contents

List of Figures

List of Tables

1. Introduction

Solving Linear Algebra (LA) problems is a main computational kernel in four out of five EoCoE II Scientific Challenges (SC) and thus the availability of exascale-enabled LA solvers is fundamental in preparing the SC applications for the new exascale ecosystem. More specifically, "LA problem" refers here to the solution of systems of algebraic linear equations, with numbers of unknowns and equations that are increasingly larger going towards exascale.

The goal of WP3 is to design and implement exascale-enabled LA solvers for the selected applications and to integrate them into the flagship codes.

This deliverable presents first results and performance evaluations obtained by the LA solvers, sometimes specifically developed and/or extended for the applications, on test cases designed in collaboration with the partners from SC. We observe that many interesting results have already been obtained and a fruitful cooperation among the SC partners and the LA experts has been set. Some activities, planned in the project proposal, have not yet started for some delay in the recruitment of collaborators, and they will be finalized in the second half of the project. On the other hand, the available funds for about all the WP3 partners do not allow to hire collaborators for 3 years, then, it is reasonable that activities have different time progress. The different tasks are involved in the testing of the performances of the new LA solvers for the application areas of *water*, *fusion*, and *wind*.

Specifically, for the Task 3.2, LA solvers for Water, we present both preliminary results relative to the integration of PSBLAS/MLD2P4 library into the ParFlow code, and the integration of the AGMG code into the PETSc software library and, through it, into the SHEMAT-suite.

Regarding Task 3.3, LA solvers for Fusion, we present some preliminary results for two types of Plasma Fusion simulations. Firstly we discuss the application of an implicitly extrapolated geometric multigrid algorithm with specific relaxation strategies to deal with the specific geometries used in the GyselaX library. Secondly, we show results based on the usage of the AGMG solver for the simulation of a two-species plasma in a divertor configuration with SOLEDGE3X.

For Task 3.4, LA solvers for Wind, the main application code is the Alya software, and we discuss the results obtained from the integration of the linear solvers and preconditioners in several test problems. In more detail, we discuss the usage of the MUMPS library as a solver in the simulation of a solid mechanic problem; the results obtained with the multigrid algorithms from PSBLAS/MLD2P4 and AGMG as solvers and preconditioners for the pressure equation for a Large Eddy Simulation on a wind-problem benchmark; and the usage of Maphys for the solution of the linear systems arising in two model problems: the simulation of a wind farm and the simulation of the airflow through the nose during a sniff.

Finally, in Task 3.5, regarding the transversal activities, we discuss some results related to the exploitation of the MUMPS library within the HHG geometric multigrid solver for large-scale simulations.

2. Acronyms

Table 1: Acronyms for the partners and institutes therein.

Acronym	Partner and institute
BSC:	Barcelona Supercomputing Center
CEA:	Commissariat à l'énergie atomique et aux énergies alternatives
CNR:	Consiglio Nazionale delle Ricerche
CNRS:	Centre Nationale de la Recherche Scientifique
CERFACS:	Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique
FZJ:	Forschungszentrum Jülich GmbH
INRIA:	Institut National de Recherche en Informatique et en Automatique
IRIT:	Institut de Recherche en Informatique de Toulouse
IRFM:	Institute for Magnetic Fusion Research
MPG:	Max-Planck- Gesellschaft zur Förderung der Wissenschaften e.V
RWTH:	Rheinisch-Westfälische Technische Hochschule Aachen, Aachen University
ULB:	Université Libre de Bruxelles
UNITOV:	University of Rome Tor-Vergata

Table 2: Acronyms of software packages.

Acronym	Software and codes
	Application codes
Alya:	High Performance Computational Mechanics
libNEGF:	General library for Non Equilibrium Green's Functions
GyselaX:	GYrokinetic SEmi-LAgrangian
ParFlow:	Parallel Flow
SHEMAT:	Simulator of HEat and MAss Transport
TOKAM3X:	Transport and turbulence in the edge plasma of tokamaks
SOLEDGE3X:	Transport and turbulence in the edge plasma of tokamaks
	LA software libraries
AGMG:	Iterative solution with AGgregation-based algebraic MultiGrid [44]
Fabulous:	Fast Accurate Block Linear Krylov Solver [30]
MaPHyS:	Massively Parallel Hybrid Solver [41]
MLD2P4:	MultiLevel Domain Decomposition Parallel Preconditioners Package based on PSBLAS [26]
MUMPS:	MUltifrontal Massively Parallel sparse direct Solver [43]
PSBLAS:	Parallel Sparse Basic Linear Algebra Subroutines [31]
PaStiX:	Parallel Sparse matriX package [49]

3. Task 3.1: Linear Algebra solvers for Materials

Partners: INRIA, FZJ
Software packages: libNEGF

There were some delays in this Task, since the flagship code was changed during the first half of the project and also because FZJ partner lost a resource. The activities related to the new flagship code, libNEGF which replaced PVnegf, are postponed at least until beginning of fall 2020. The new code uses a completely different computational approach than PVnegf, therefore, at the moment we do not know which are the possible needs in terms of Linear Algebra for the new code. However, we observe that a very small amount of WP3 man power was originally allocated to this Task which can be finalized in the second half of the project.

4. Task 3.2: Linear Algebra solvers for Water

The main focus in this Task is on integration of the PSBLAS and MLD2P4 libraries for solving Poisson-type systems arising in inexact Newton method for solving non-linear Richard's equation in groundwater simulations. Some other work is devoted to use AGMG as linear solver in the SHEMAT-suite. The activities proceed with no relevant issues through a fruitful collaboration with the colleagues from the application side.

4.1 PSBLAS and MLD2P4 towards ParFlow

Partners: CNR, FZJ
Software packages: MLD2P4, PSBLAS, ParFlow

During the first phase of the project, as reported in the Deliverable 3.1 [46], we analyzed the issue of including an interface to PSBLAS (dev. 3.7) and MLD2P4 (dev. 2.2) linear solvers and preconditioner to the ParFlow code. The conclusion we reached is that the optimal way of performing this interfacing is through the KINSOL package from SUite of Nonlinear and DIfferential/ALgebraic equation Solvers (SUNDIALS) [24]. More specifically, this is due to the fact that ParFlow uses the Newton solvers included in KINSOL to approximate the solution of the nonlinear algebraic systems arising from the discretization of their models. Therefore we developed for this Task an interface that enables the usage of the solver and preconditioners from PSBLAS and MLD2P4 inside the Newton steps in KINSOL so that they can be invoked from the call to KINSOL in ParFlow. We report in Figure 1 the overall structure of the software.

This implies the construction of three separate modules for KINSOL encapsulating, respectively, the interfaces to (parallel sparse) linear algebra routines for distributed vectors and matrices (a new `N_Vector` and `SUNMatrix` APIs), and the interfaces to the Krylov iterative linear solvers and multilevel preconditioners (a new matrix–based `SUNLinsolver` API). These interfaces are written in C from the KINSOL library and use the C/Fortran2003 interfaces from PSBLAS/MLD2P4, guaranteeing a full interoperability of the data structures, i.e., they do not require producing any auxiliary copy of KINSOL objects for translation into PSBLAS objects; everything can be manipulated from KINSOL directly into the native formats for PSBLAS and MLD2P4. The details about the implementation of the relevant APIs, and the operators made available by the interfacing are described in the documentation for the interface [28] that can be downloaded from `https:`

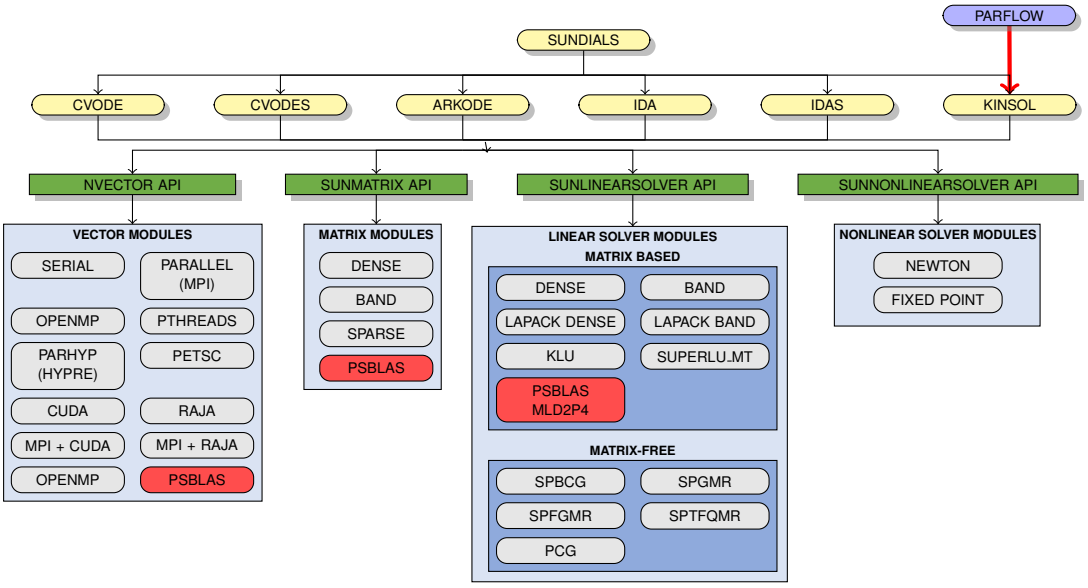


Figure 1: High-level diagram of the inclusion between ParFlow, the KINSOL library from the SUNDIALS suite and the PSBLAS/MLD2P4 modules. The interface discussed here are highlighted in red.

`//github.com/Cirdans-Home/kinsol-psblas`.

In the following we present some preliminary results on a reference test case available from the KINSOL distribution in which we use both the linear algebra routines and iterative solvers provided by PSBLAS as well as one of the preconditioner available in MLD2P4. We stress that the interfacing for this case did require the definition and implementation of some new vector and matrix operation in PSBLAS, unlike the case of the solver for Task 3.4 described in Section 6.2. Hence, this activity has been performed on the current *development* version of the software, and of the sister packages MLD2P4 and PSBLAS-EXT; please refer also to the codesign aspects discussed in the Deliverable 3.1 [46]. In the second half of the project we will define test cases from the ParFlow code, in collaboration with FZJ, and will test our libraries on realistic large-scale simulations.

Preliminary results on a KINSOL test case

We present here some preliminary results on a small-scale nonlinear problem arising from the the minimization of the energy functional

$$\mathfrak{J}(u) = \int_\Omega \left(\frac{\varepsilon^2}{2}|\nabla u|^2 + \frac{1}{2}u^2 - \frac{1}{4}u^4 \right), \qquad \Omega = [0,1]^2, \tag{1}$$

over a class of admissible *smooth* functions V vanishing on the boundary of the unit square $\partial\Omega$, i.e., of finding

$$u = \arg\min_{u \in V} \mathfrak{J}(u).$$

Smooth critical points for (1) satisfy the Euler–Lagrange equation

$$\begin{cases} -\varepsilon^2 \nabla^2 u - u + u^3 = 0, & \mathbf{x} \in \Omega. \\ u = 0, & \mathbf{x} \in \partial\Omega. \end{cases}$$

We consider here a modified version of the equation in order to have a closed–form solution for comparing the error

$$\begin{cases} -\varepsilon^2 \nabla^2 u - u + u^3 = f, & \mathbf{x} \in \Omega. \\ u = 0, & \mathbf{x} \in \partial\Omega. \end{cases} \tag{2}$$

For this case we consider a finite difference discretization of (2) with an increasing number of degrees of freedom per MPI core to analyze weak scalability properties of the whole procedure, i.e., of the complete nonlinear solution procedure in KINSOL by means of the routines from the PSBLAS/MLD2P4 interface. The Jacobian matrices for the Newton iteration for this test problem are symmetric and positive definite, hence we chose to employ the Conjugate Gradient linear solver preconditioned by a single V-cycle iteration of an algebraic multigrid method with two step of Hybrid Forward/Backward Gauss-Seidel pre/post smoother, and a single sweep of a Block-Jacobi solver with an ILU(0) factorization on blocks as coarse grid solver. The coarse grid corrections are based on the smoothed decoupled aggregation described in detail in [27]. In this setting a new preconditioner is computed whenever the Newton solver decides to compute a new Jacobian. The preliminar results illustrated in Figure 2 are obtained on a laptop with Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz processor and 16 GiB SODIMM DDR4 Synchronous 2400 MHz RAM. Even if on a limited number of cores, we observe that the combination of the incomplete

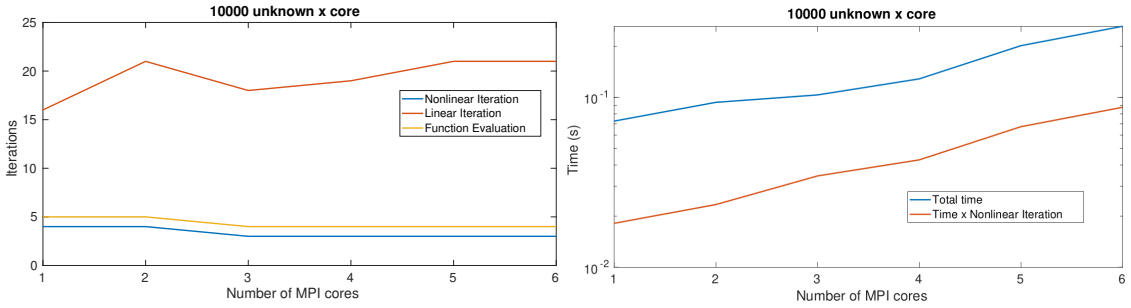


Figure 2: Weak scalability: iteration number of the linear solvers, number of nonlinear iterations and of function evaluations with $1e4$ dofs per core (left panel), total time and time per nonlinear iteration step (right panel). The timings comprise the time needed to assemble the Jacobians and the relative preconditioners in the Newton iteration.

solution of the linear systems on multiple cores does not affect in a significant way the number of non-linear iterations and function evaluations, and similarly the number of linear iteration is sufficiently stable. We stress that, due to the test setup, the communication overhead might be much smaller if compared to a system in which a communication network is used. Similarly, if we consider the complete solution time, containing both the time needed to assemble the Jacobians and the time needed to setup the multilevel preconditioner, it has a reasonable growth for the size of the considered problems and the computing platform. The code used to generate this example is provided with the set of PSBLAS example, and it is described in full detail in the guide [28].

4.2 PSBLAS and MLD2P4 on GPUs

Partners: CNR, UNITOV, FZJ
Software packages: MLD2P4, PSBLAS, Parflow

During the first phase of the project we also focused on the improvement of PSBLAS and MLD2P4 functionalities, implementing Krylov solvers and AMG preconditioners, for

running on Graphics Processing Units (GPUs), in order to prepare the libraries to efficiently exploit of high-throughput many-core processors currently found in many of the fastest supercomputers in the Top 500 list [2]. The above functionalities will be of immediate use in Parflow through the PSBLAS and MLD2P4 interface to the KINSOL package described in section 4.1.

Since AMG methods are obtained by combining different components (smoother, coarsening algorithm, coarsest-level solver, restriction and prolongation operators), a full exploitation of GPU capabilities requires each component to be optimized for this type of architecture. We first focused on the application phase of AMG preconditioners, and in particular on the choice and implementation of AMG smoothers and coarsest-level solvers capable of harnessing the computational power offered by a cluster of GPUs. We consider inexact block-Jacobi smoothers and solvers that use sparse approximate inverses to perform the local solves required by each block, instead of the usual factorization methods. The choice of sparse approximate inverses is motivated by the much larger amount of parallelism exposed by sparse matrix-vector products as compared to the parallelism available in sparse triangular solves. The approximate inverses are computed with a plugin for MLD2P4 that has been described in [16]. In particular, implementations of the following methods are available:

- INVK(I,J): the approximate inversion of an ILU factorization based on pattern-levels; for example, an INVK(0,1) would start with an ILU(0) factorization, and then compute an approximate inverse of its factors with 1 level of fill-in;
- INVT$(\epsilon_1, n_1, \epsilon_2, n_2)$: a similar approximate inverse of triangular factors computed through ILUT(ϵ, n);
- AINV-LLK: a method based on biconjugation as described in [16].

The previous kernels have been combined by using the AMG framework provided by the MLD2P4 library, to get multigrid cycles suitable for clusters of GPUs. This has been made possible by the modular architecture of MLD2P4, which has allowed the extension with new data storage formats and new local solvers for block-Jacobi iterations, as well as the use of the PSBLAS GPU plugin. We illustrate the behaviour of the AMG preconditioner described so far on linear systems arising from a model problem which mimics the groundwater model in Parflow. We performed *weak scalability* tests, keeping approximately 16×10^6 DOFs per processor. The results obtained are representative of the behaviour of the AMG preconditioner on linear systems coming from general isotropic elliptic equations. The experiments were carried out using the Piz Daint supercomputer (ranked 6^o in the Top 500 list), operated by the Swiss National Supercomputing Centre and available for our CoE through the PRACE Research Infrastructure. The linear systems were solved using the GPU implementation of the Conjugate Gradient (CG) method provided by PSBLAS with the GPU plugin, coupled with a V-cycle preconditioner using one sweep of the block-Jacobi method as pre-smoother and as post-smoother, with INVK(0,1) as local solver, ten sweeps of BJAC(INVK(0,1)) were applied as coarsest-level solver. The multilevel hierarchy was built by running (on CPUs) the decoupled smoothed aggregation algorithm available in MLD2P4, using its default parameters. The zero vector was chosen as starting guess and the CG iterations were stopped when the ratio between the 2-norm of the residual and the 2-norm of the right-hand side became smaller than 10^{-6}. A variant of the ELLPACK sparse storage scheme named HLL was used when running the solve phase on GPUs. We obtain good weak scalability (see Fig. 3) on up to 512 GPUs and more than 8×10^9 DOFs,

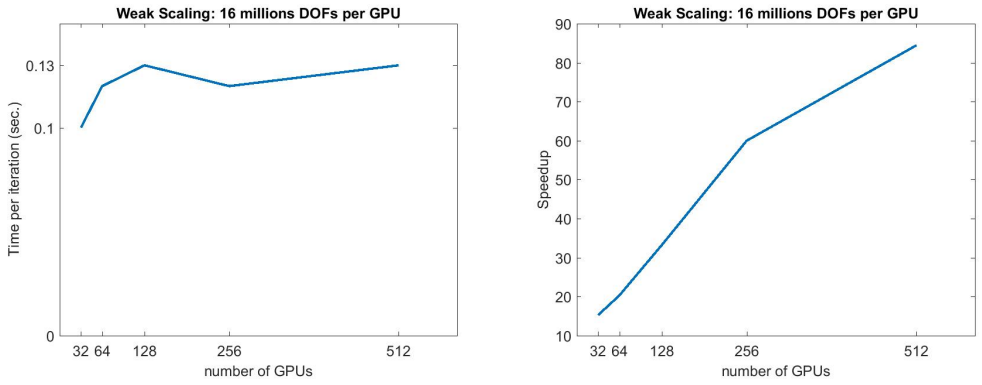


Figure 3: PSBLAS and MLD2P4 on cluster of GPUs. Weak scaling: 16×10^6 DOFs per GPU

with a speedup (defined as T_1/T_p, where T_1 is the solve time on 1 GPU and T_p is the solve time when p GPUs on different computational nodes are used) up to about 85 when 512 GPUs are used. An almost constant time per iteration ranging from 0.1 to 0.13 sec. shows a very good implementation scalability of the preconditioned Krylov solver.

4.3 AGMG for the SHEMAT-Suite

Partners: RWTH, ULB
Software packages: AGMG, SHEMAT

A post-doc has been hired in the end of 2019 to fulfill this task of integrating the AGgregation-based algebraic MultiGrid (AGMG) solver (www.agmg.eu) into the Portable, Extensible Toolkit for Scientific Computation (PETSc) (www.mcs.anl.gov/petsc). It will provide an alternative to the Hyper-BoomerAMG currently used for solving mulit-phase, steam or multi-phase- multi-component flow and transport problems with PETSHEM, a simulation code based on SHEMAT-Suite. In the following section, we aim to highlight the principle and the benefits for this integration for simulation with PETSHEM related to geothermal energy. We will present:

- Principle of PETSc;
- Principle of AGMG;
- Implementation on SHEMAT-Suite.

This is a preparatory work so far, the main work on this task is scheduled from mid-May to October 2020.

Principle of PETSc

PETSc consists of a suite of libraries (similar to classes in C++) that provide the building blocks for the implementation of large-scale application codes in serial and parallel ([11], [10]). Fig. 4 is a diagram of these librairies. Each library manipulates a particular family of objects and the operations one would like to perform on the objects. Some of the PETSc modules deal with: index sets, vectors, matrices and dozens of preconditioners, including algebraic multigrid (AMG). Each class of PETSc is implemented by using a C

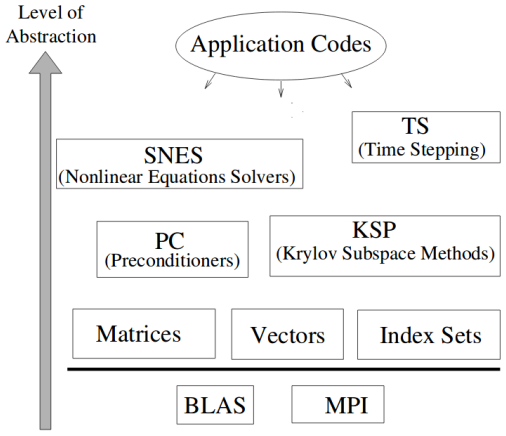


Figure 4: Libraries in PETSc including the Preconditioners

structure that contains the data and function pointers for operations on the data. One class consists of three parts: (i) a common part shared by all PETSc classes; (ii) another common part shared by all PETSc implementations of the class; (iii) a private part used by only one particular implementation written in PETSc. By example, all matrix classes share a function table of operations that may be performed on the matrix; all PETSc matrix implementations share some additional data fields, including matrix parallel layout, while a particular matrix implementation in PETSc has its own data fields for storing the actual matrix values and sparsity pattern.

PETSc uses the MPI standard for all message-passing communication. PETSc includes an expanding suite of parallel linear solvers, nonlinear solvers, and time integrators. PETSc provides many mechanisms needed within parallel application codes, such as parallel matrix and vector assembly routines. PETSc provides then clean, parallel and effective codes for the various phases of solving Elliptic partial differential equations (PDEs), with a uniform approach for each class of problem. This design enables easy comparison and use of different algorithms. The libraries enable easy customization and extension of both algorithms and implementations. This approach promotes code reuse and flexibility and separates the issues of parallelism from the choice of algorithms.

Principle of AGMG

Many application software need an efficient solver for linear systems from the discretization of PDEs. Direct solvers are generally employed for effective reducing the error function at all scales. Classic iterative methods mainly act locally and are very effective in reducing highly oscillatory modes but not smooth modes.

AGMG solves systems of linear equations with an aggregation-based algebraic multigrid method ([44]. The multigrid algorithm consists of solving the problem on a coarser grid. It defines a hierarchy of levels - with the top and bottom being referred to as 'fine' and 'coarse', respectively. On each level (except the coarsest), we require (i) an operator A, and an operator used for preconditioning M, (ii) a 'smoother', (iii) an operator to restrict a solution vector to the next coarsest level (R), and (iv) an operator to prolongate a solution vector from the coarse level below (P). On the coarsest level in the hierarchy

we require a 'coarse' level solver. It yields an approximate solution which is essentially correct from a large-scale viewpoint. The algebraic multigrid (AMG) is a black box algorithm constructing automatically the coarsening from the input matrix. However, the algorithm is not completely user friendly since some difficulties exist in selecting the most opportune variant and the parameters.

Fig. 5 shows a conceptual representation of the AGMG. The approach is more user-friendly and more robust by coarsening based on plain aggregation. The software is expected to be efficient for large systems arising from the discretization of scalar second order elliptic PDEs. It may, however, be tested on any problem, as long as all diagonal entries of the system matrix are positive. It is indeed purely algebraic. It is available both as a software library for FORTRAN or C/C++ programs, and as a Matlab function. The

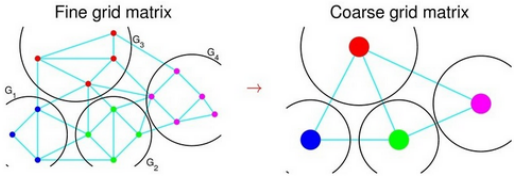


Figure 5: Conceptual scheme of the AGgregation-based algebraic MultiGrid (AGMG) solver

application of AGMG has been shown for both Alya and ParFlow softwares. The main benefits of AGMG highlighted during these studies are: (i) its extra robustness; (ii) its simple connectivity on coarse grids; (iii) its easy parallelization; (iv) its use of only one variant thus its user-friendliness and (v) its faster than other classical iterative solvers.

Implementation on SHEMAT-Suite

SHEMAT-Suite (Simulator for HEat and MAss Transport) is a flow and transport simulation code for a wide variety of thermal and hydrogeological problems in two and three dimensions ([23], [51]). Specifically, SHEMAT-Suite can simulate flow, heat and species transport through saturated porous media. Originating from SHEMAT-Suite, a code for simulating multi-phase, steam and multi-phase-multi-component flow and transport through porous media has been developed in order to solve problems related to deep or superhot geothermal reservoirs or to CO2 sequestration. This code is called PETSHEM (Portable Extensible Toolkit for the Simulation of HEat and Mass) and uses solvers provided by PETSc for solving the PDEs (e.g. [20], [19]). Currently, the BoomerAMG from PETSc's Hypre package is used for solving linear systems. This task aims for replacing the BoomerAMG by AGMG via PETSc for improving the preconditioning process.

This work is performed in collaboration with the PETSHEM and AGMG developers Henrik Büsing and Yvan Notay. After fruitful discussions between both of them, the following workflow has been decided. A Fortran routine will be created to interface between PETSc and AGMG. The routine has to be to be short to be easily executed and outside from any PETSc structure. The routine will take as arguments, PETSc matrices required by AGMG. It will include also the functionalities proposed by PETSc. Two series of test will be made: (i) a sequential Fortran routine for a simple 2D case with only one field ; (ii) a MPI Fortran routine for a more complex case solved in parallel.

5. Task 3.3: Linear Algebra solvers for Fusion

In this Task activities developed in the first half of the project were carried out by Cerfacs, in collaboration with MPG-IPP and IRFM-CEA, and were focused on the GyselaX code. Cerfacs developed a specialized high-order geometric multigrid method, as explained in section 5.1. Further activities were carried out by IRFM-CEA, in collaboration with ULB, for testing AGMG in the code SOLEDGE3X. Some other activities planned on this Task by IRIT-CNRS and INRIA, will be developed in the second half of the project, due to some delays in hiring collaborators.

5.1 Geometric Multigrid Solver for Plasma Fusion Simulations in GyselaX

Partners: CERFACS, MPG-IPP, IRFM-CEA
Software packages: GyselaX

During the first phase of the project (see[46]), several meetings between the MPG-IPP and CERFACS were held to discuss the co-design of the scalable geometric multigrid solver for an elliptic PDE defined on stretched polar grids coming from the GyselaX code, mainly developed by CEA-IRFM. Further meetings between CEA-IRFM and CERFACS took place in September 2019.

Based on the inputs, a geometric multigrid algorithm with suited parameters was chosen. Multigrid methods can achieve optimal complexity for many problems and are among the most efficient solvers for elliptic model problems such as the gyrokinetic Poisson equation; see, e.g., [18, 58].

Typical density profiles [61, 54] used in the gyrokinetic Poisson equation decay rapidly from the core to the edge region of the tokamak; see Figure 6. In addition to the rapid decay of the coefficient, a local refinement of the mesh to pass from the core to the edge region is used; see, e.g., [35, 47]. Further variety in the coefficients of the partial differential equation is introduced by the description of the geometry by curvilinear coordinates.

While symmetry of differential operators is naturally conserved when using finite elements, special attention has to be paid when developing finite difference stencils for nonuniform grids or when the differential operator has varying coefficients. Particular finite difference stencils to overcome these issues have been developed within the project [36]. These stencils lead to a novel approach to obtain a matrix-free implementation of the discretized operator. This is the key to reduce the memory footprint and in consequence it also helps to reduce the amount of data that must be transferred to the processing units for computing a matrix-vector multiplication. In order to cope with the artificial singularity introduced by curvilinear coordinates, *discretizations across the origin* have been proposed; numerically they yield the same results as Dirichlet boundary conditions as the innermost circle approaches the origin [37].

Based on these discretizations, multigrid methods have been designed. Multigrid methods are well-studied methods but are less common for geometries described by curvilinear (e.g., polar) coordinates (see [56, 12, 17, 58, 42]). Different studies had to be undertaken to optimize its parameters with respect to the geometry obtained from fusion plasma applications. Zebra relaxation is of particular interest for anisotropic operators; see [56]. For curvilinear coordinates, the two natural zebra smoothing operations are

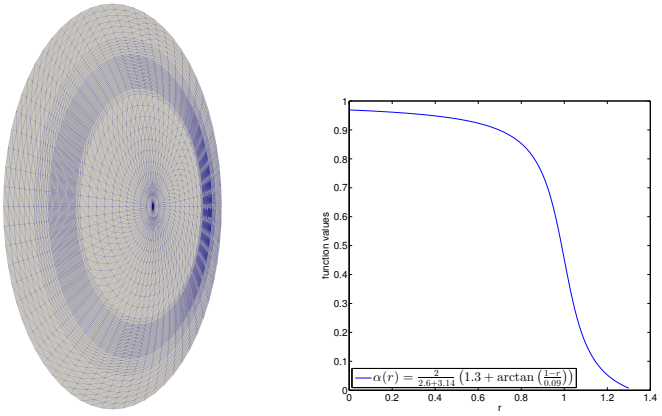


Figure 6: Deformed curvilinear (left) geometry with coordinates $(r, \theta) \in [r_1, 1.3] \times [0, 2\pi]$. Rapidly decaying density profile (right). Around the decay of the coefficient, the meshes are locally refined in r; here, with $h_{\text{max}}/h_{\text{min}} = 8$.

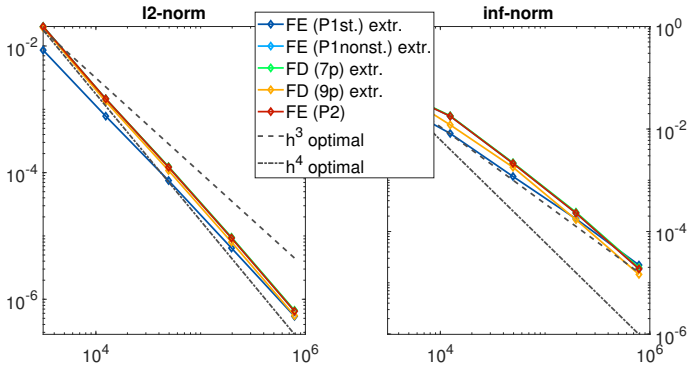


Figure 7: Error convergence in $\ell 2$- and $\inf$-norm for direct solution of the two-level extrapolation method combined with finite difference and finite element discretizations for $-\nabla \cdot (\alpha \nabla u) = f$ in $\Omega_L = (\mathbf{0.1}, 1.3) \times [0, 2\pi)$, f given by [62], $\Omega = F(\Omega_L)$ a curvilinear geometry as in Fig. 6, Dirichlet boundary conditions in r, periodic boundary conditions in θ.

denoted circle and radial zebra relaxation. For operators where the anisotropy changes across the domain, alternating zebra relaxation has been proposed; see [56]. To reduce the workload and to parallelize the smoothing operation, we propose circle relaxation around the origin and change to radial relaxation if the local smoothing factor of circle relaxation becomes superior; cf. [12, 37].

To satisfy the need of higher order methods, implicit extrapolation was incorporated into the multigrid cycle. The combination of extrapolation with multilevel and multigrid solvers seems in many ways natural and has thus recently seen renewed interest; see [48, 25, 57]. Our extrapolation methods are so-called implicit variants and have been developed in [53, 52, 33]. In order to integrate these implicit extrapolation methods into the multigrid cycle, only the smoothing operations and intergrid transfer operators of the finest level have to be adapted [33, 34, 37]. Proofs for the extrapolation between two levels and nonstandard integration rules are given in [36], the numerical results show almost identical behavior for our matrix-free finite difference discretizations; see [36, 37].

We finally present some results for our implicitly extrapolated geometric multigrid al-

$n_r \times n_\theta$	its	$\widehat{\rho}$	$\|err\|_{\ell_2}$	*ord.*	$\|err\|_\infty$	*ord.*	its	$\widehat{\rho}$	$\|err\|_{\ell_2}$	*ord.*	$\|err\|_\infty$	*ord.*
Circular geometry												
	FE P1 (nonstandard integ.)						*FD 5p*					
49×64	37	0.61	3.6e-03	-	1.6e-02	-	37	0.61	3.6e-03	-	1.6e-02	-
97×128	38	0.61	2.4e-04	3.92	1.5e-03	3.45	38	0.61	2.4e-04	3.91	1.5e-03	3.44
193×256	39	0.62	1.8e-05	3.76	1.8e-04	3.08	39	0.62	1.8e-05	3.75	1.8e-04	3.09
385×512	39	0.62	1.4e-06	3.64	2.2e-05	3.00	39	0.62	1.4e-06	3.65	2.2e-05	3.00
Deformed geometry												
	FE P1 (nonstandard integ.)						*FD 9p*					
49×64	76	0.78	7.9e-03	-	3.0e-02	-	73	0.78	7.6e-03	-	2.6e-02	-
97×128	81	0.80	6.1e-04	3.69	4.3e-03	2.82	78	0.79	5.6e-04	3.76	2.9e-03	3.13
193×256	83	0.80	4.8e-05	3.67	4.5e-04	3.24	78	0.79	4.2e-05	3.72	3.6e-04	3.01
385×512	85	0.80	3.7e-06	3.71	4.5e-05	3.35	79	0.79	3.2e-06	3.71	4.5e-05	3.00

Table 3: **Comparison of extrapolated discretizations.** Multigrid with extrapolation based on finite element and finite difference discretizations on circular and deformed geometry with $r_1 = 1e-5$ and discretization across the origin

gorithm; see Figure 7 and Table 3. We conduct one step of pre- and one step of postsmoothing, i.e., $\nu = \nu_1 + \nu_2 = 2$. In prospect of a parallel implementation, we only use V-cycles. We use a strong convergence criterion by demanding a relative residual reduction by a factor of 10^8. The maximum number of iterations is set to 150. We provide the finest mesh size as $n_r \times n_\theta$. We also provide the iteration count of the multigrid algorithm needed to convergence as *its* as well as the mean residual reduction factor. For all simulations, we give the error of iterative solution compared to the exact solution evaluated at the nodes in the (weighted) $\|\cdot\|_{\ell_2}$-norm and the $\|\cdot\|_\infty$-norm. We also provide the error reduction order as *ord.* for both norms.

The resulting multigrid algorithm using optimized radial-circle smoothing to take into account the anisotropies and implicit extrapolation to raise the order of convergence still has optimal complexity per iteration. The implicit extrapolation scheme raises the error convergence, in the inf-norm, from 2 to 3 and, in the l_2-norm, from 2 to almost 4. Finally, the parallel implementation will be realized in the near future and in strong cooperation with the MPG-IPP

5.2 SOLEDGE3X with AGMG and PaStiX

Partners: CEA-IRFM, INRIA, ULB
Software packages: SOLEDGE3X, AGMG, PaStiX

SOLEDGE3X is the evolution of the code TOKAM3X, which is referred in the project proposal description. The name has been changed because this is a major evolution and that former TOKAM3X represents only a subset of SOLDEGE3X capabilities.

In the initial version of the code TOKAM3X/SOLEDGE3X, the direct solver PASTIX was used as linear system solver. However, it suffers memory issues when increasing the simulation size. In the framework of the EoCoE-II project, SOLEDGE3X has been interfaced with both AGMG [44] and PETSc [10] to provide alternative linear system solvers.

To obtain preliminary results, a strong scaling test has been achieved on the OCCIGEN cluster (CINES). The architecture is based on Intel Haswell nodes with 24 CPUs each.

The case chosen is a production case. It is a divertor configuration where the computation domain is based on multi-domain structured 3D grid. The number of flux surfaces is 119 and the number of poloidal planes is 64. The number of grid points is about 4×10^6. One simulates the quarter of a torus. The discretization grid is depicted on Figure 8.

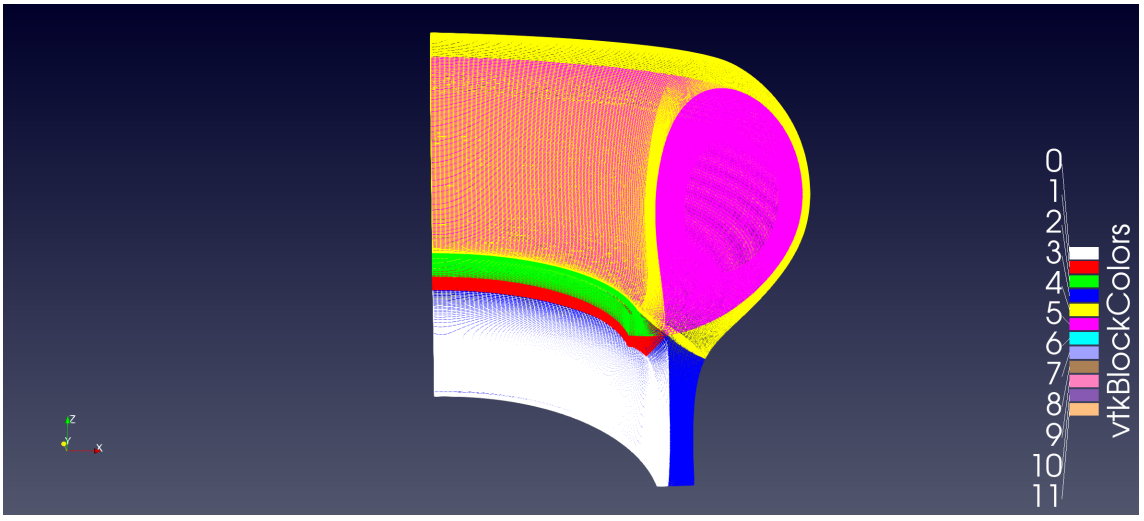


Figure 8: Simulation grid for a divertor configuration: structured grid with 6 subdomains.

The computation time to solve a full system of equations for a two species plasma (deuterons+electrons) has been measured varying the number of MPI tasks. The greater part of the computation time is spent in the inversion of the discretized electric potential equation, as seen on Figure 9, where results are reported for the case when one uses the PETSc solver (similar picture is obtained with AGMG).

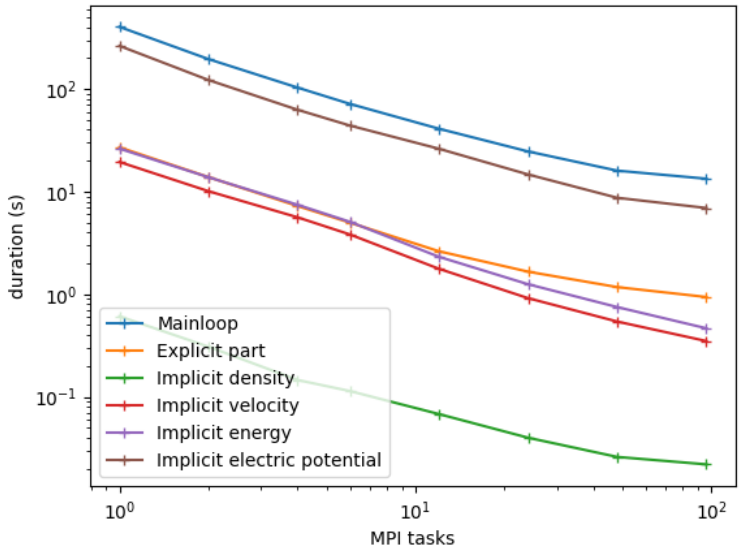


Figure 9: Detail of computation time per operator (PETSc case). Mainloop is the sum of all operators. Most of the time is spent in Implicit solve of the electric potential equation.

The linear systems to solve are 3D inhomogeneous, strongly anisotropic discretized Laplacian with Robin boundary conditions. The conditioning of the matrix is pretty bad. As stated above, two iterative solvers have been implemented and tested:

- PETSC used with BCGS Krylov solver and GAMG preconditioner (an algebraic multigrid type preconditioner) [10];
- AGMG [44].

In both cases, the convergence tolerance was set to 10^{-8}.

The following MPI/OpenMP configurations are used to scan the range of task number from 1 to 96.

case	N nodes	N tasks per node	N threads per task	N tasks	N cpus
1	1	1	2	1	2
2	1	2	2	2	4
3	1	4	2	4	8
4	1	6	2	6	12
5	1	12	2	12	24
6	2	12	2	24	48
7	4	12	2	48	96
8	8	12	2	96	192

Note also that attempt was made to run the case with the PASTIX direct solver (version 5) for comparison but it was not possible to do so due to memory limits (the whole memory of the 8 computing nodes was not sufficient).

The results are reported on Figure 10. The two solvers show quite similar performance in term of computation time.

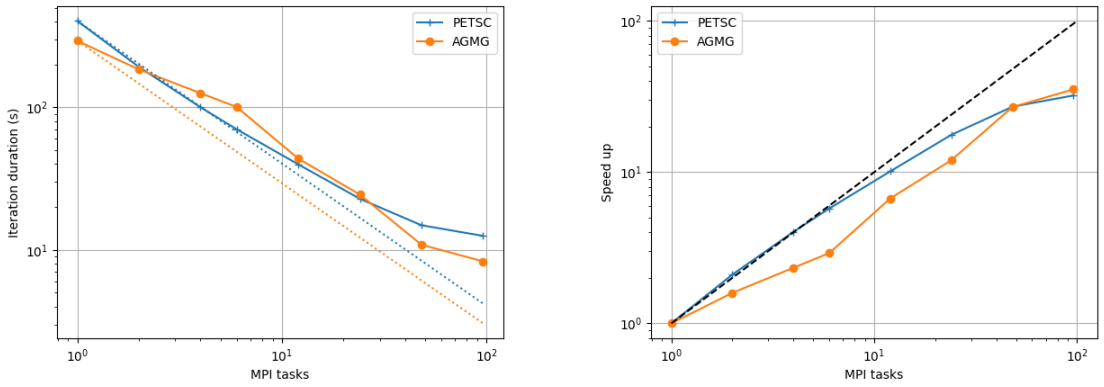


Figure 10: Total time spent in linear system solutions (left) and corresponding speed-up (right).

The speed-up is somewhat larger with PETSc on intermediate cases, but one has to take into account that, for each solver, speed-up is reported with respect to the sequential case *with the same solver* (1 task configuration in the above table), while AGMG is actually faster sequentially. The trend seen on the largest numbers of nodes is also slightly better

with AGMG, although there is room for improvement with both solvers.

6. Task 3.4: Linear Algebra solvers for Wind

A large effort of WP3 man power is focused on the integration of the Linear Algebra libraries to improve robustness and parallel performance of different Alya software modules. Activities proceed without relevant issues, although interactions among the partners and models analysis, as expected, led to few changes in the planned activities in terms of the most promising libraries for the different Alya modules. Preliminary results of the integration of MUMPS in the software module for solid physics are discussed in section 6.1. Results of the integration of PSBLAS and MLD2P4 in the module for CFD, when a Large Eddy Simulation (LES) model is employed, are presented in section 6.2. Some comparison results between AGMG and PSBLAS/MLD2P4 are presented in section 6.3. Finally Section 6.4 discusses preliminary results obtained by the integration of the Maphys library into the module for CFD, when a Reynolds Averaged Navier-Stokes (RANS) model is used.

6.1 MUMPS integration in Alya

Partners: BSC, IRIT-CNRS
Software packages: MUMPS, Alya

A solid mechanics problem is selected to evaluate the computational performance of the parallell direct solver MUMPS [43] coupled with the Alya HPC system [1]. The selected test case corresponds to a structural part named as mono-stringer or also called skin-stiffener, which is typically used in aircraft structures to sustain compression and bending loads. The geometry and FE details are shown in Fig. 11.

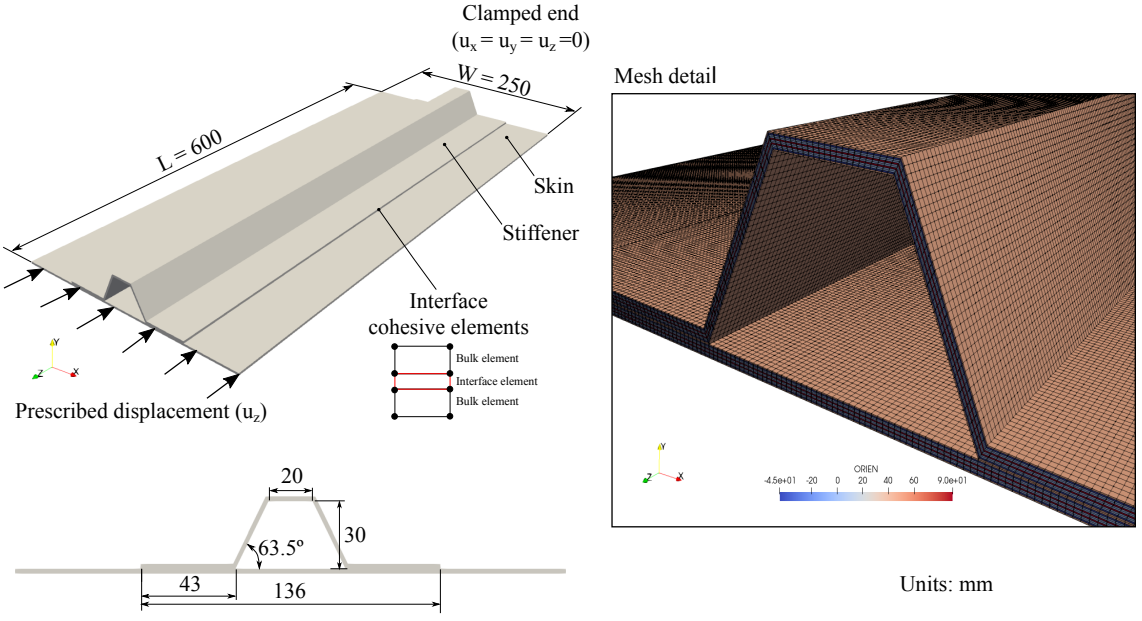


Figure 11: Geometry and mesh details of the test case

The mono-stringer is fully clamped at one end and a prescribed displacement is

applied at the other end in the z-direction. The element size used is 0.184x1x1 mm resulting a total number of 3.118.800 elements $\approx 9.4e6$ DOF. The model also includes interface cohesive elements for the damage onset and propagation of the skin/stiffener debonding [59]. The mono-stringer is made of fiber reinforced material $T800/M21$ with elastic material properties and interface properties summarized in Tab. 4.

E_{11}	$E_{22} = E_{33}$	$\nu_{12} = \nu_{13}$	ν_{23}	$G_{12} = G_{13}$	G_{23}
141GPa	8.54GPa	0.299	0.45	4.43GPa	3.02GPa
G_{Ic}	G_{IIc}	τ_I	τ_{II}	K_p	η_{B-K}
0.308	0.828	20MPa	32.8MPa	$1e5$N/mm^3	1.75

Table 4: Elastic and interface material properties for the $T800/M21$ material

The model is solved using the Implicit Newmark-Beta scheme from [14] using $\beta = 0.65$, $\gamma = 0.9$ and $\alpha = 0$. In order to improve convergence of the solution a dynamic implicit analysis is used with a smooth step to reduce the inertial forces while applying the prescribed displacement.

Regarding to the algebraic solver, the parallel direct solver MUMPS is used. This case is very attractive for parallel direct solvers due to the heterogeneity of the material and the differences in orders of magnitudes in the matrix entries due to the damage onset and propagation. By now, only the linear elastic part of the problem is studied, which corresponds to a fully symmetric global system matrix. Hence, a single time step is applied with a small prescribed displacement of 4e-3mm due to the smooth step function. All the simulations have been performed on the Marenostrum 4 Supercomputer (ranked 30th in the Top 500 list [2]). In the following, the different setups of the MUMPS solver are described and discussed.

The latest MUMPS development version was tested on the mono-stringer problem under different configurations. MUMPS is a parallel direct solver for sparse linear systems of equations. It is based on the multifrontal method and can exploit distributed memory as well as shared memory parallelism through the MPI and OpenMP programming models, respectively. A first set of experiments was run on 5 nodes using 4 MPI processes per node and 12 threads per MPI process, as reported in the second column of Table 5 (each MareNostrum node is equipped with two Intel Xeon Platinum processors with 24 cores each). Therefore, the total number of used cores is 240. The first row of Table 5 shows the execution time for the Factorization and Solve (i.e., forward elimination and backward substitution) steps for a base version and should be used as a reference; FR stands for full-rank as opposed to BLR explained below. By "base" we intent that MUMPS was executed with all the default options. The second line in this table shows the execution time obtained when activating the $\mathcal{L}_0$ multithreading feature. The associated method was developed by L'Excellent *et al.* [39] and is currently being stabilized and integrated in MUMPS. This method achieves advanced multithreaded parallelism in those parts of the workload which are less computationally intensive and more memory bound. The $\mathcal{L}_0$ feature brings an improvement of roughly 20%.

Next, we have experimented with the block low-rank (BLR) feature. This technique relies on the use of low-rank approximations to asymptotically reduce the operational complexity and memory consumption of linear algebra operations such as the sparse factorization and solve [7]. This reduced complexity comes at the cost of a loss of accuracy which can be reliably controlled through a single parameter which we refer to as *the BLR threshold*. Rows 3-10 of Table 5 show the results obtained with the BLR feature for a

	N$times$MPI×OMP	Facto time	Solve time	res.	flops (% FR)
base FR	$5 \times 4 \times 12$	130.0	1.52	6.3D-16	100.0
$\mathcal{L}_0$ FR	$5 \times 4 \times 12$	108.2	1.19	5.7D-16	100.0
BLR 10^{-10}	$5 \times 4 \times 12$	61.4	0.54	1.6D-9	15.3
BLR 10^{-9}	$5 \times 4 \times 12$	63.8	0.53	1.3D-8	12.5
BLR 10^{-8}	$5 \times 4 \times 12$	63.3	0.55	2.1D-7	10.2
BLR 10^{-7}	$5 \times 4 \times 12$	56.9	0.48	1.7D-6	7.9
BLR 10^{-6}	$5 \times 4 \times 12$	58.8	0.45	7.9D-6	5.7
BLR 10^{-5}	$5 \times 4 \times 12$	55.9	0.43	5.9D-5	4.2
BLR 10^{-4}	$5 \times 4 \times 12$	46.0	0.38	4.7D-4	2.5
BLR 10^{-3}	$5 \times 4 \times 12$	45.2	0.35	1.1D-3	1.7
$\mathcal{L}_0$ FR	$20 \times 12 \times 4$	110.1	0.79	8.8D-16	100.0
$\mathcal{L}_0$ FR	$20 \times 4 \times 12$	67.8	0.60	7.1D-16	100.0

Table 5: Experimental results obtained with MUMPS on the mono-stringer problem. The first column shows the MUMPS features used in each test. The second shows the experimental setting for parallelism in number of nodes × number of MPI per node × number of OpenMP threads per MPI. The third and fourth columns show, respectively, the factorization and solve times. The fifth shows the backward error (i.e., scaled residual) and the sixth shows the operational complexity expressed as the number of floating point operations with respect to the full-rank case.

threshold from 10^{-10} to 10^{-3}. As we can see, this leads to a considerable reduction of the execution time. Column 5 shows the obtained backward error which follows quite accurately the BLR threshold. The last column, instead, shows the operational cost. This is expressed as the number of floating point operations relative to the standard, full-rank, case. It can be observed that the gain in time is smaller than that in flops. For example for a 10^{-9} threshold, despite a flops reduction of a factor eight, the execution time is reduced only by a factor less than two. This behavior was thoroughly analyzed and explained in the literature [8] and is due to the fact that when BLR feature is used, the granularity of operations and, thus, their efficiency, is lower with respect to the full-rank case. The use of the BLR feature is of particular interest in this application because this problem is solved within iterations of a non-linear solver. Therefore a controlled loss of accuracy in each iteration may lead to a higher number of iterations but, overall, to a lower execution time. It must be noted that all these experiments were done in double precision but for thresholds that are higher than 10^{-7} it is possible to use single precision without loss of accuracy; this may provide an additional speedup of up to a factor 2.

The last two rows of Table 5 show experiments done with the full-rank version, using the $\mathcal{L}_0$ feature with 960 cores. These results show that, provided that a favorable combination of MPI and OpenMP parallelism is chosen, MUMPS continues to scale quite well despite the relatively small size of the problem.

6.2 PSBLAS and MLD2P4 at work in Alya

Partners: BSC, CNR
Software packages: MLD2P4, PSBLAS, Alya

During the first phase of the project, as reported in the Deliverable 3.1 [46], we developed a software module included in the Alya's kernel, to interface PSBLAS (rel. 3.6) and MLD2P4 (rel. 2.0) to the code. This allows to Alya the exploitation of linear solvers and preconditioners from those libraries for solving linear systems arising from the

different physics modules. In this project, our main aim is to test the libraries for systems stemming from fluid dynamics simulation based on the NASTIN module, which deals with the incompressible Navier-Stokes equations for turbulent flows. In the following we present some preliminary results on a test case defined by our partners from BSC. We discuss in detail the parallel efficiency of some of the most suitable solvers and preconditioners available in the libraries for the selected test case.

Test Case Description

The mathematical model is the set of 3D incompressible unsteady Navier-Stokes equations for the Large Eddy Simulations (LES) of turbulent flows in a bounded domain with mixed boundary conditions. The LES formulation is closed by an appropriate expression for the subgrid-scale viscosity; in this analysis, the eddy-viscosity model proposed in [60] is used. Discretization is based on a low-dissipation mixed finite-element scheme, using linear finite elements both for velocity and pressure unknowns. A non-incremental fractional-step method is used to stabilise the pressure, whereas for the explicit time integration of the set of discrete equations a fourth order Runge-Kutta explicit method is applied [38]. The pressure field is obtained at each step by solving a discretization of a Poisson-type equation. The test case is based on the Bolund experiment, a classical benchmark for microscale atmospheric flow models over complex terrain [13]. The Reynolds number based on the friction velocity is approximately $RE_\tau = 1.0e7$. We run some strong scalability tests for a fixed size unstructured mesh of tethrahedra with 5570786 central nodes (dofs) as well as weak scalability tests fixing different mesh sizes per cores, for incresing number of cores up to 12288 cores and a mesh size up to $345276325 \approx 0.35e9$ dofs. At each time step, we solve the spd linear systems arising from the pressure equation employing the Conjugate Gradient (CG) method of PSBLAS, coupled with different preconditioners implemented in MLD2P4. In detail, we used right preconditioned CG starting from an initial guess for pressure from the previous step, and stop iterations when the Euclidean norm of the relative residual is not larger than $TOL = 1e-3$. A general row-block data distribution based on Metis 4.0 is applied for the parallel runs. The simulations have been performed with the Alya code interfaced to PSBLAS (3.6) and MLD2P4 (2.0), built with GNU compilers 7.2, on the Marenostrum 4 Supercomputer composed of 3456 nodes with 2 Intel Xeon Platinum 8160 chips with 24 cores per chip (ranked 30^o in the Top 500 list [2], with more than 10 petaflops of peak performance), operated by BSC. The facility was made available by a grant dedicated to the EoCoE II project from PRACE.

Strong Scalability

In this section we present strong scalability results obtained on the Bolund experiment for the fixed size problem including $n = 5570786$ dofs. We run tests by using a time step equal to 3e-3 sec. and, at each time step, we solve the symmetric positive-definite (spd) linear systems arising from the pressure equation employing different preconditioners for a flexible version of the CG solver, as implemented in PSBLAS 3.6. A number of cores from 48 to 768 is used to analyze parallel efficiency and convergence behaviour of the linear solver for the first 100 time steps of the simulation. Note that in the Alya code a master-slave approach is employed, where the master process is not involved in the parallel computations. The preconditioners discussed in the following and available through the PSBLAS/MLD2P4 software libraries include both one-level and algebraic multilevel

methods. After various experiments, we selected the diagonal preconditioner (DIAG), for comparison aims with the diagonal preconditioner implemented in the original Alya code and generally used for simulations, and the Block-Jacobi method coupled with Incomplete LU factorization employing 1 level of fill-in for the diagonal blocks (BJACILU1), as representative of the one-level Domain Decomposition (DD) methods available in MLD2P4. From the multilevel set we selected a symmetric V-cycle employing an algebraic multilevel hierarchy with 4 levels built by applying the decoupled smoothed aggregation coarsening implemented in MLD2P4, 4 iterations of forward/backward Gauss-Seidel smoother at the intermediate levels and two different parallel iterative coarsest solvers: 4 iterations of the Block-Jacobi method with ILU(1) on the diagonal blocks (MLBJAC) and the CG method preconditioned by ILU(1) with a stopping criterion based on the reduction of the relative residual of 4 orders of magnitude or a maximum number of iterations equal to 30 (MLRKR). In Fig. 12 we report a comparison of the different methods in terms of the total number of iterations of the linear solvers and of the solve time per iteration (in seconds). Note that in the figures we also have results obtained when the diagonal preconditioned CG (AlyaDCG) and a version of Deflated CG (AlyaDFCG) available from the original Alya code are used in the simulations. We can observe that the total number of linear iterations is smaller and very stable for increasing number of cores, when multilevel preconditioners are applied. Indeed, total number of linear iterations ranges from 334 to 337 in the case of MLRKR and from 380 to 441 for MLBJAC. The same stability is observed for AlyaDFCG where the total number of linear iterations ranges from 4524 and 4530. An increase in the total number of linear iterations is shown by the BJAC preconditioner where variability for different number of cores is also observed. The worst behaviour, as expected, is obtained when the simple diagonal preconditioners are applied, indeed both AlyaDCG and DCG require a very large number of linear iterations with large variability during the overall simulation.

In all cases, the time needed per each iteration decreases for increasing number of cores and, as expected, it is larger for the multilevel preconditioners, where the cost for the preconditioner application at each CG iteration is larger than the more simple one-level preconditioners. In the case of MLRKR the solve time per iteration ranges from 0.16 sec. on 48 cores to 0.02 sec. on 768 cores (about $3.6e-9$ sec. per dof), showing a good implementation scalability. Looking at the behaviour of the different preconditioners during the simulation, we can observe in Fig. 13 the number of linear iterations at each time step when the simulation is carried out on 48 cores. We see that MLRKR and MLBJAC stabilize in a few time steps to the value of 3 iterations per time step. A similar stability is observed for AlyaDFCG which shows an averaged number of iteration equal to 45 for all the time steps. On the contrary, larger variability during the simulations is shown by the other preconditioners, especially for AlyaDCG and DCG.

In Fig. 14 (on the top) we can see the total solve time spent in the linear solvers and the resulting speedup for the preconditioners. Here we define speedup as the ratio $Sp = T_{48}/T_p$, where T_{48} is the total time for solving linear systems when $p = 48$ total cores are involved in the simulation, and T_p is the total time spent in linear solvers when $p = 96, 192, 384, 768$. We observe that the best behaviour is obtained when MLRKR is applied. It shows the smallest solve time per each number of cores, yet decreasing for increasing number of cores. DCG also shows a comparable solve time when the largest number of cores are used, due to the very small cost of its application at each linear iteration (see Fig. 12 on the bottom) which compensates the large number of total iterations. We can see that the one-level BJACILU1 shows a solve time which is generally comparable to that

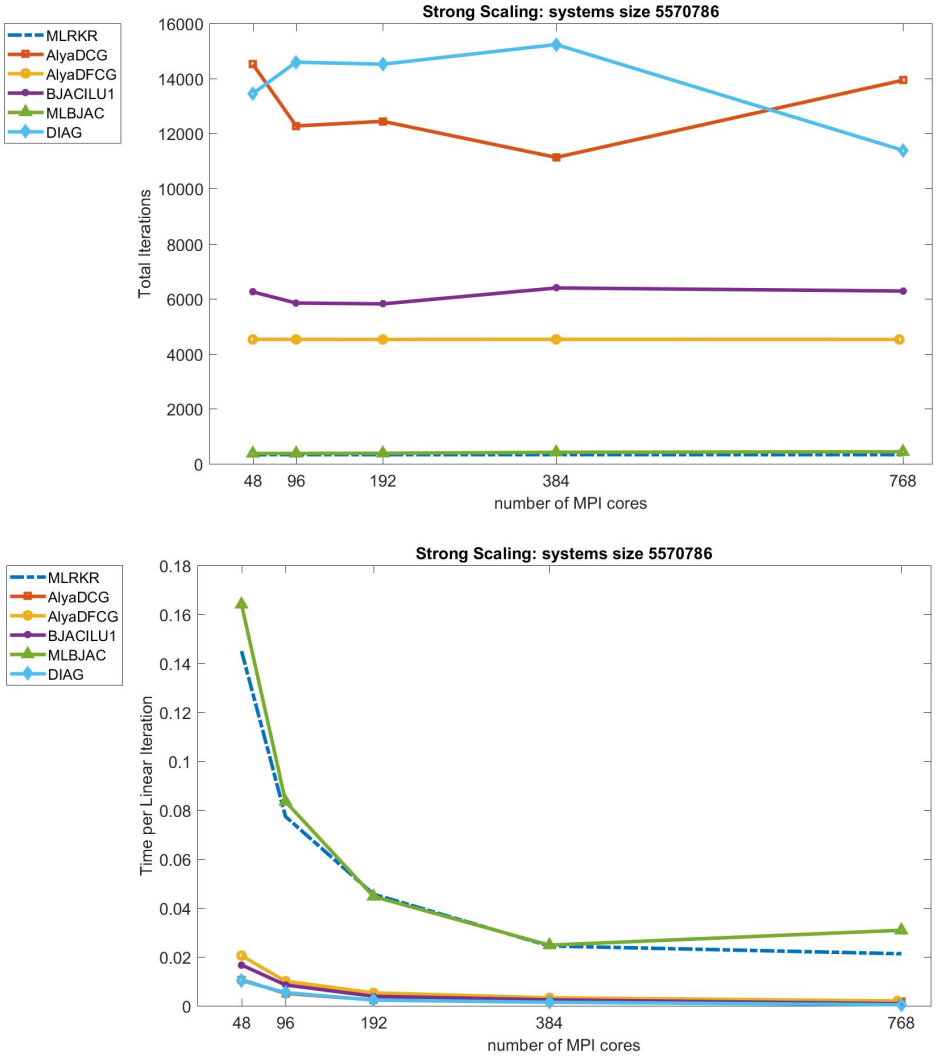


Figure 12: Strong scalability: iteration number (top) and time per iteration (bottom) of the linear solvers

obtained by AlyaDFCG but in the case of the largest number of cores, where BJACILU1 is better due to its smaller solve time per iteration. Speedup of the solve phase for all the preconditioners are reported in Fig. 14 (on the bottom). We can see that the speedup obtained by the best preconditioner (MLRKR) is about 7 (to the best of 16) when 768 cores are used, corresponding to a parallel efficiency of about 44%, while superlinear speedup are obtained when DCG and BJACILU1 are applied.

In Fig. 15 we report the time for setup of the MLD2P4 preconditioners and the resulting speedup (scaled to 48 cores as for solve time). We observe that the setup cost for all the preconditioners is very small and it is largely compensated by the time savings obtained by using them at each time step of the solution phase.

In conclusion, the selected solvers from PSBLAS and MLD2P4 libraries outperform the original Alya solvers and in the better case of FCG coupled with the multilevel preconditioner MLRKR speedup obtained with respect the best Alya solver (AlyaDFCG) ranges between about 1.4 on 768 cores to more than 2 on 48 cores.

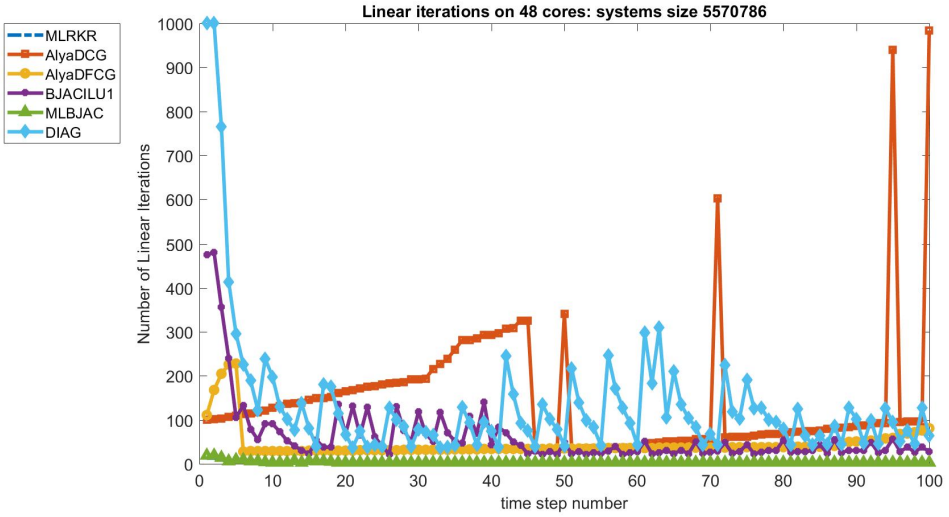


Figure 13: Linear iterations per time step on 48 cores

Weak Scalability

In this section we analyze performance of the multilevel versions of the MLD2P4 preconditioners, when we fix 3 different mesh sizes per core for increasing number of cores, with the final aim to analyze weak scalability properties and efficiency of resource usage. Indeed, main aim in parallel computation is both to use the available resources at the best and to be able to efficiently solve larger problems when larger resources are employed. We limit our analysis to the multilevel version of the MLD2P4 preconditioners which turned out as the best choice both in terms of iteration number and solve time in the previous section. We considered the test case Bolund, described in Section 6.2, with mesh size from $\approx 5.5e+6$ up to $\approx 0.35e9$ dofs. Three different configurations of computational cores are employed from 48 up to 3072, from 96 up to 6144 and from 192 to 12288, corresponding to 3 different mesh sizes per core equal to $1.1e5$, $5.9e4$, and $2.9e4$, respectively. We run simulations for a total of 20 time steps starting from the time equal to 1.5e-1. Note that due to the increasing mesh sizes, time steps used during simulations for increasing number of cores are generally different due to the CFL stability constraint for velocity which is dealt in explicit way. In Figs. 16-17 we can see the total number of iterations needed for the overall simulations and solve time per iteration, when different mesh sizes per core are used. We observe that, as expected, the multilevel preconditioners from MLD2P4 show the smaller number of iterations in all cases, w.r.t. the Alya solvers. MLRKR ranges from 80 iterations for the smallest mesh size and for all the three smallest number of cores to 187 in the case of the largest size when 12288 cores are used, showing a small increase in the number of iterations for increasing number of cores. A similar behaviour is observed for MLBJAC, where we have a slight larger increase of iterations when the number of cores increases. Furthermore, if we look at the time spent at each linear iteration (see Fig. 17), we see that the multilevel preconditioners show a very small increase, especially when the largest mesh size per core is used. This shows a very good implementation scalability of the library code, which improves when the surface to volume effect is reduced, i.e. when the ratio between data communication and data computation is reduced. In all cases, the time per iteration is not larger than 0.28 sec. on 12288 (see MLRKR when the medium

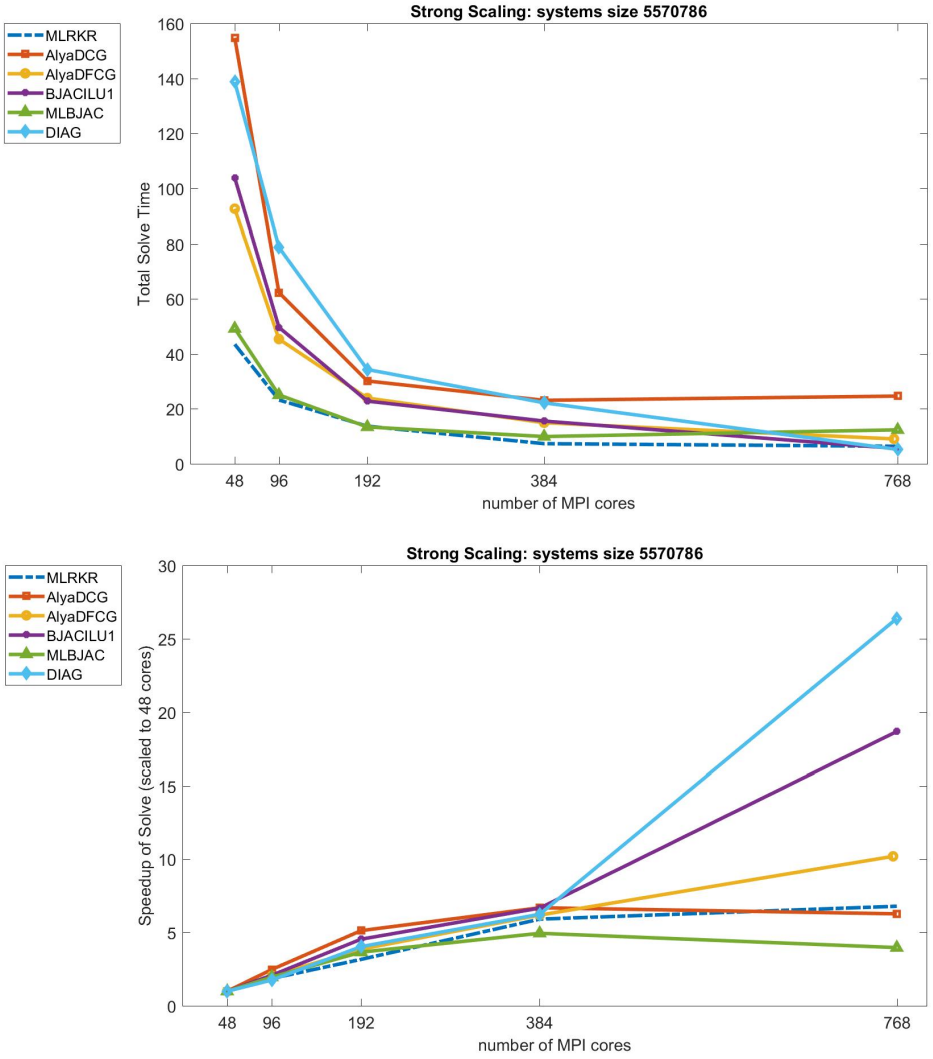


Figure 14: Strong scalability: total solve time (top) and speedup (bottom) of the linear solvers

size per core is used), corresponding to a worst case time per dof of about 8.1e-10 sec. We also observe that MLBJAC has a smaller time per iteration than MLRKR, especially when the largest number of cores is used. This depends on the less data communication required by 4 iterations of the distributed Block-Jacobi method w.r.t. the preconditioned CG for the coarsest systems solution. On the contrary, Alya solvers show a large increase in the linear iterations when the number of cores increases, in details AlyaDCG ranges from 1513 to the worst case of 8086 iterations on 12288 cores, while AlyaDFCG ranges from 964 to the worst case of 3371 iterations on 12288 cores. As expected and already observed from the strong scalability results, the time per iteration of both the Alya original solvers is smaller than that of the MLD2P4 preconditioners, In Figs. 18-19 we can see the total time for solve and speedup of the solve phase, respectively. We observe that, the larger cost per iteration of the multilevel preconditioners is largely compensated by their good algorithmic scalability. Indeed, the large reduction in the number of iterations produces smaller solution time both for MLBJAC and MLRKR w.r.t. the Alya original solvers. The best preconditioner is MLBJAC in the case of the smallest and medium mesh

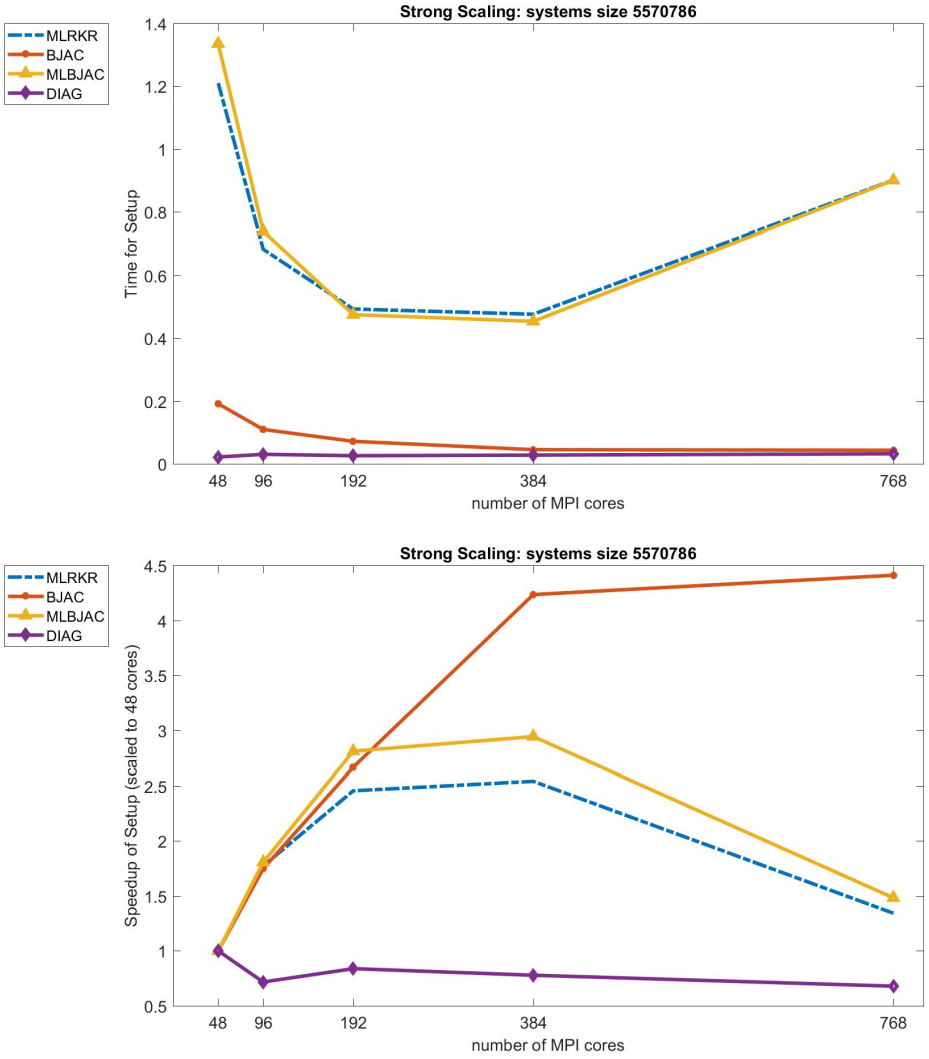


Figure 15: Strong scalability: MLD2P4 preconditioners setup time (top) and speedup (bottom)

size per core, where its slight larger number of iterations w.r.t. MLRKR is compensated by the smaller time per iteration, while MLRKR and MLBJAC are comparable when the largest mesh size per core is used and only up to 3072 cores are used to solve the largest global size problem. Very good speedup values are obtained by the MLBJAC preconditioner for all the mesh size per core and number of computational cores, with the best efficiency of about 54% when the maximum mesh size per core and 3072 cores are used. MLRKR has similar behaviour in this case, while its efficiency degrades for increasing number of cores and decreasing mesh size per core. In Figs. 20-21 we can see the time for MLD2P4 preconditioners setup and speedup, respectively. We see that the setup time for building the multilevel MLD2P4 preconditioners has a slight increase for increasing number of iterations and both the preconditioners have, as expected, a very similar behaviour, since they are different only in the coarsest solver, which appears more expensive for MLBJAC in the case of the largest mesh size per core. Indeed, in this case, the cost of ILU(1) factorization of the diagonal blocks has a slight larger weight on the overall setup. However, we can observe that the cost of the preconditioner setup is largely

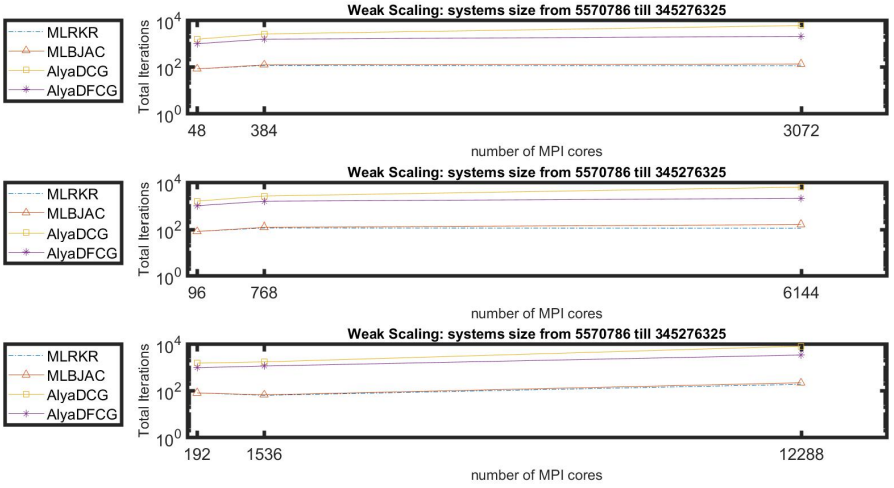


Figure 16: Weak scalability: iteration number of the linear solvers. $1.1e5$ dofs per core (top), $5.9e4$ dofs per core (middle), $2.9e4$ dofs per core (bottom)

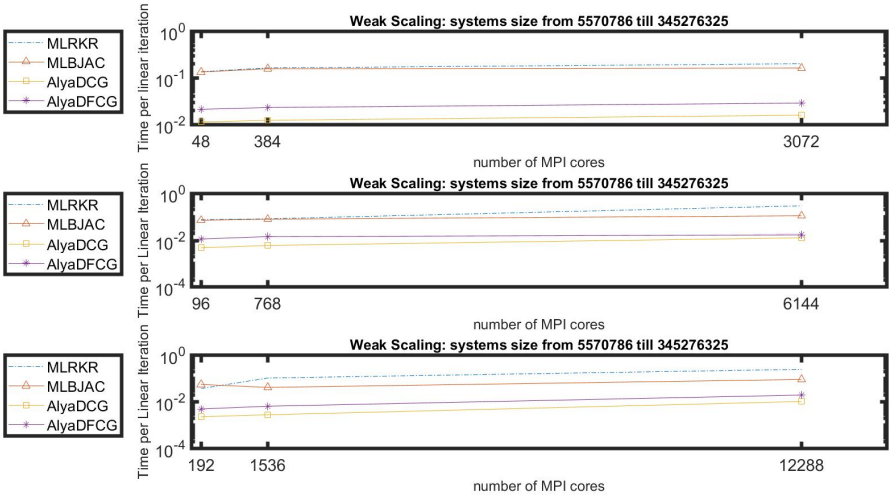


Figure 17: Weak scalability: time per iteration of the linear solvers. $1.1e5$ dofs per core (top), $5.9e4$ dofs per core (middle), $2.9e4$ dofs per core (bottom)

amortized in using the same preconditioner for solving at each time step of the LES. In Fig. 22 we show the resource usage, computed as the product between the total time for solving linear systems and the number of cores, for the different configurations of cores and mesh size per core. This could be considered a good estimate of the cost of the simulations in the different cases. We see that the smaller cost is observed when the maximum mesh size per core is used confirming that a smaller surface to volume ratio, that is a smaller communication to computation ratio, corresponds to efficient use of available resources.

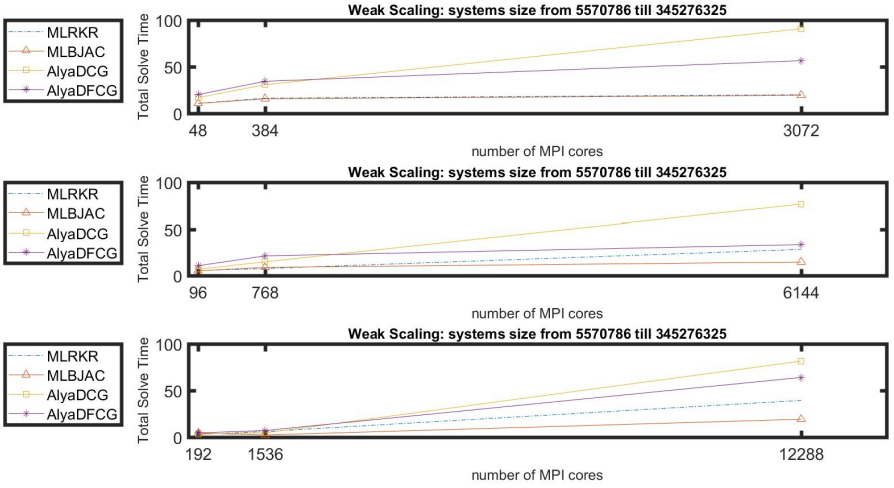


Figure 18: Weak scalability: total solve time of the linear solvers. $1.1e5$ dofs per core (top), $5.9e4$ dofs per core (middle), $2.9e4$ dofs per core (bottom)

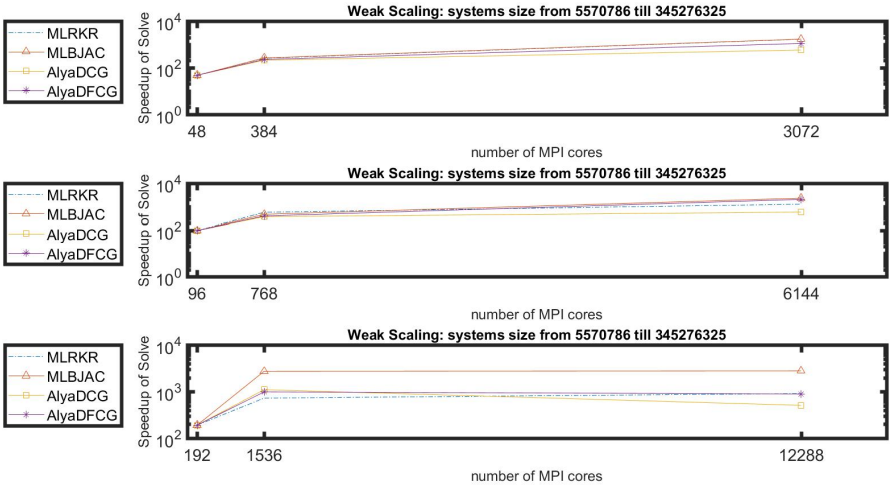


Figure 19: Weak scalability: speedup of the linear solvers. $1.1e5$ dofs per core (top), $5.9e4$ dofs per core (middle), $2.9e4$ dofs per core (bottom)

6.3 AGMG performance comparison in Alya

Partners: BSC, ULB
Software packages: AGMG, Alya

AGMG has been interfaced with Alya since EoCoE I and several tests have already been performed; the results have been reported in [29]. It has proven to be vastly superior to Alya's own solvers. Therefore, it has been preferred to put AGMG testing on standby and wait for other solvers to be ready in Alya so that a better inter-comparison could be performed. Now that results with PSBLAS/MLD2P4 have been obtained, we have

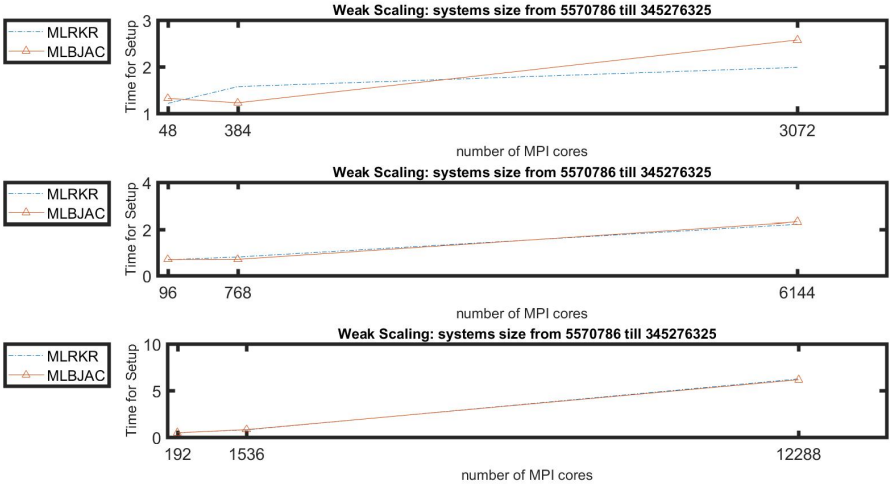


Figure 20: Weak scalability: MLD2P4 preconditioners setup time. $1.1e5$ dofs per core (top), $5.9e4$ dofs per core (middle), $2.9e4$ dofs per core (bottom)

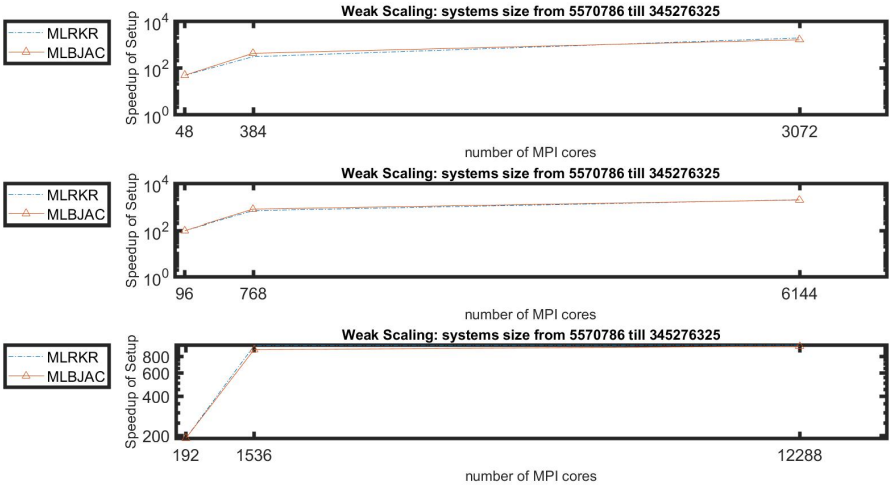


Figure 21: Weak scalability: speedup of MLD2P4 preconditioners setup. $1.1e5$ dofs per core (top), $5.9e4$ dofs per core (middle), $2.9e4$ dofs per core (bottom)

decided to repeat the same weak scalability test presented in the previous section for the Bolund experiment with AGMG. Therefore, the case description is not repeated, and we concentrate only on the results.

In Table 6, the number of iterations for each solver is compared. It remains nearly unaltered for each of the meshes. The only variation appears for the AGMG solver when 1 level of mesh multiplication is applied(44.8M unknowns mesh). To simplify the presentation, we shall refer to the three meshes as coarse, medium, and fine from now onward. PSBLAS/MLD2P4 takes fewer iterations to converge for all cases. For both solvers, the highest number of iterations is obtained for the medium mesh. It nearly doubles compared to the coarse mesh. However, for the fine mesh, very good results are obtained. For

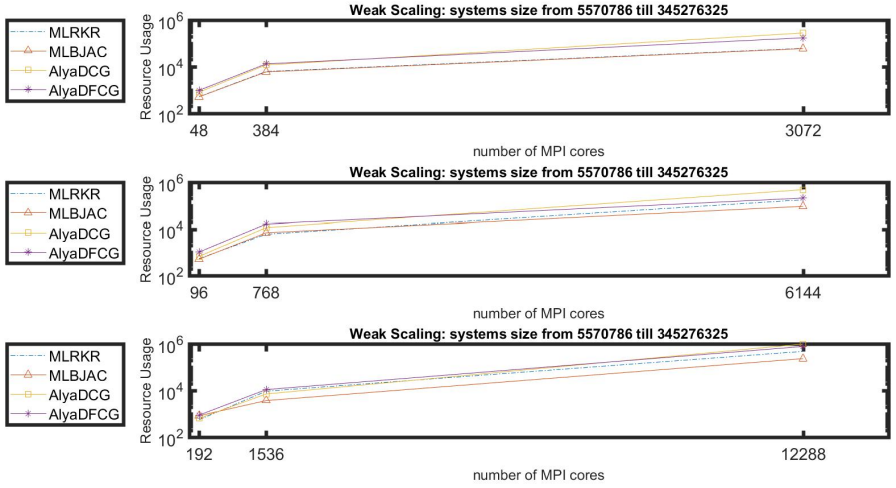


Figure 22: Weak scalability: resource usage. $1.1e5$ dofs per core (top), $5.9e4$ dofs per core (middle), $2.9e4$ dofs per core (bottom)

Cores	Total Million Unknowns	AGMG	PSBLAS
48	5.6	8	3
96	5.6	8	3
192	5.6	8	3
384	44.8	14	5
768	44.8	14	5
1536	44.8	13	5
3072	358.4	6	4
6144	358.4	6	4
12288	358.4	6	4

Table 6: Number of iterations comparison between AGMG and PSBLAS/MLD2P4

PSBLAS/MLD2P4, the number of iterations increases only from 3 to 4. For AGMG, the algorithmic scalability is very good; when going from the coarse to the fine mesh (an increase of 64 times in the number of unknowns), the number of solver iterations decreases. Thus, we can say that both solvers have very good algorithmic scalability, but that an unexpected behavior is obtained for the medium mesh. The comparison between both solvers using the number of iterations is of little interest because the CPU time per iteration can vary. For comparison purposes between solvers, we prefer to look at the CPU time required to perform those iterations.

In Table 7, the CPU times in seconds for each solver are compared. Contrary to the approach used in the PSBLAS section where the average value for twenty time-steps was used here, we will use the minimum value. Since this might seem a little strange at first, we discuss the reason for this choice in more detail in the next paragraph. In any case, since both the minimum values for AGMG and PSBLAS/MLD2P4 are used, the comparison is fair.

The reason for using the minimum value instead of the average one is that there exist

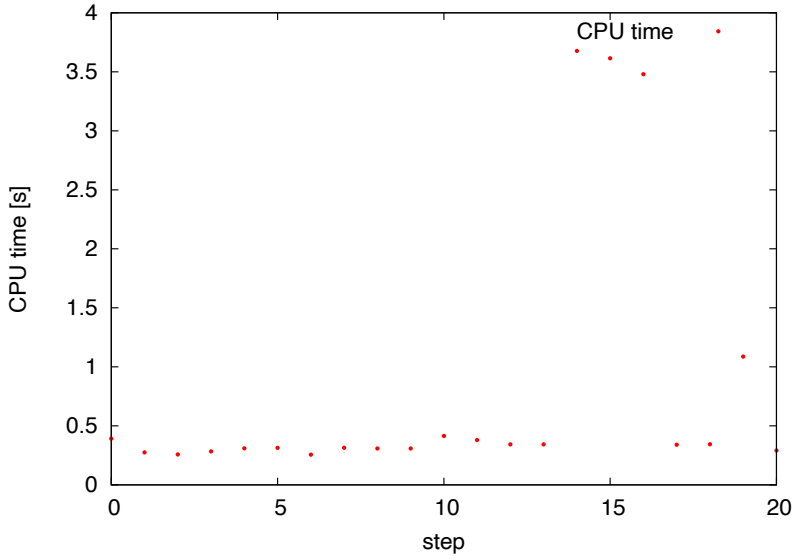


Figure 23: CPU time variation for the fine mesh with 12288 cores

significant noise when looking at the CPU times for the twenty time-steps. We believe the problem must be coming from Marenostrum's hardware or software (operating system or MPI implementation). In Figure 23, we show the CPU times for the twenty time-steps under analysis. We use the PSBLAS/MLD2P4 run with 12288 cores, which does 4 iterations for all twenty time-steps, but similar results can be observed with AGMG. One can see that for time steps 14, 15, and 16, the CPU time is approximately ten times higher than for the remaining time steps. If one repeats the simulation times steps that take significantly more CPU time, appear at random time steps. Since these behaviors appear for two different solvers and at random time steps, our best guess is that they must be attributed to Marenostrum, possibly to the turbo boost and speed step technologies that are activated by default [3, Section 4.4]. More elaborate procedures than using the minimum value could have been proposed for filtering out unreasonable values. However, for this preliminary evaluation, we believe the minimum value is more than adequate. From our previous experience with other supercomputers, we can say that frequency of appearance of outlier values and their magnitude can depend on the machine. Therefore, we are looking forward to testing these cases in other machines. Actually, since Alya is part of the Unified European Application Benchmark Suite (UEABS) [5, 4] it could be a good idea to introduce such tests there. It is interesting to note that despite linear algebra packages are a key ingredient in the solution of a wide range of scientific problems, there is no specific benchmark on linear algebra in UEABS. Initially, we could put the tests inside Alya's benchmark, but probably latter, a specific Linear Algebra benchmark could be setup.

From Table 7, one can see that PSBLAS/MLD2P4 is always faster than AGMG. The weak scalability for the case with 116k unknowns per core (runs with 48, 384, and 3072 cores) is good for AGMG and it appears a bit better than the one for PSBLAS/MLD2P4.

Cores	Total Million Unknowns	AGMG - CPU time [s]	PSBLAS - CPU time [s]
48	5.6	0.419	0.368
96	5.6	0.231	0.192
192	5.6	0.130	0.099
384	44.8	0.743	0.606
768	44.8	0.430	0.316
1536	44.8	0.293	0.169
3072	358.4	0.524	0.523
6144	358.4	0.543	0.294
12288	358.4	0.843	0.205

Table 7: CPU times comparison between AGMG and PSBLAS/MLD2P4

With the coarse mesh, PSBLAS/MLD2P4 is 12% faster than AGMG, but with the fine mesh, they take nearly the same time. In the cases with 58k and 29k unknowns per core, the weak scalability of AGMG gets worse. Actually, if one looks at the results with AGMG on the fine mesh, no strong scalability exists. The use of more cores increases the CPU time instead of decreasing it. Despite not being perfect, the strong scalability results with PSBLAS/MLD2P4 for the fine mesh, are much better than those with AGMG. This is in agreement with the information displayed in AGMG's web page, where the smallest number of unknowns per core used is 175k [45]. In order to highlight the fact that AGMG needs a large number of unknowns per core to work well, we have rerun the fine mesh with 1536 cores, leading to an average of 233k unknowns per core. In this case, PSBLAS/MLD2P4 takes 1.05 sec while AGMG takes 0.82 sec. Thus, with a big enough load per core AGMG can be faster than PSBLAS/MLD2P4. As a preliminary conclusion, for AGMG to be a competitive option, the number of unknowns per core needs to be higher than 200k. This behavior agrees with what appears in the AGMG web page [45]. Comparing with our day to day simulations with Alya, this number is a bit too high. However, if GPUs turn out to be the dominant hardware architecture, the typical size of unknowns per core will probably increase.

6.4 Parallel performance of the `Maphys` solver in Alya

Partners: BSC, INRIA
Software packages: Maphys, Alya

This section deals with the usage of `Maphys` [41] into the Alya high-performance computing simulation code. Our goal with this study was to evaluate and improve the performances of the `Maphys` coarse space correction mechanism into an applicative context. The theory [6] of the coarse space correction mechanism fits into the abstract Schwarz (aS) framework, where we derive a bound for the condition number of all deflated aS methods provided that the coarse space consists of the assembly of local components that contain the kernel of some local operators following the GENEO [55] philosophy. Those local components are computed by solving local generalized eigenvalues problems, whose eigenvectors serve to define the coarse spasìce. It allows us to design numerically scalable domain decomposition solvers, where the condition number is fully under control and does not depend neither from the number of subdomains (that very often correspond to the number of MPI processes) nor from possible specific features of the underlying PDE (e.g., number of heterogeneity layers). While the numerical scalability is proven, the efficient

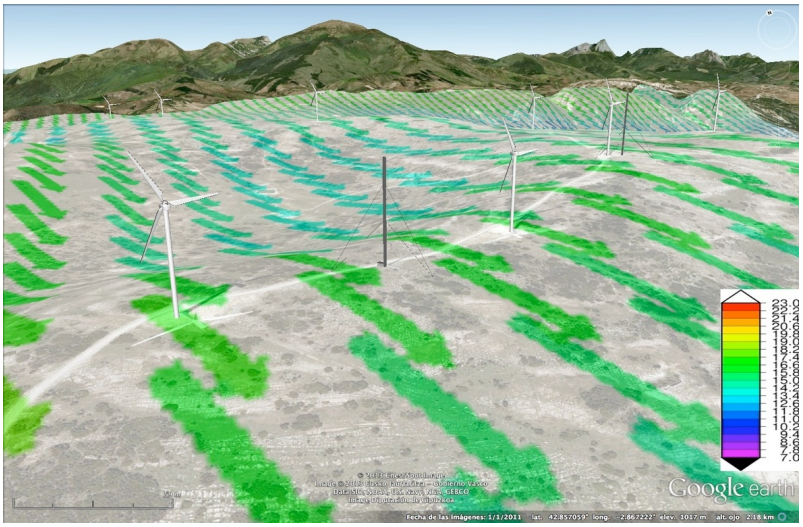


Figure 24: The wind-farm test case.

parallel implementation of the construction and application of the coarse space correction within a fully features package requires some attention and possibly some flexibility in the algorithmic design. The current version of `Maphys` implements the coarse space mechanism for hybrid solvers that rely on a Schur complement approach that are often more robust but more computationally intensive.

For that purpose, we confronted Alya's internal solvers and `Maphys` solver for linear algebra on test cases implemented into Alya, with a particular focus on test cases leading to find the solution of symmetric positive definite (SPD) systems. In this case, coarse space correction or deflation mechanisms can be used into both Alya's internal solvers and `Maphys` .

Two test cases have been chosen for this study:

- the simulation of a wind farm,
- and the simulation of the airflow through the nose during a sniff.

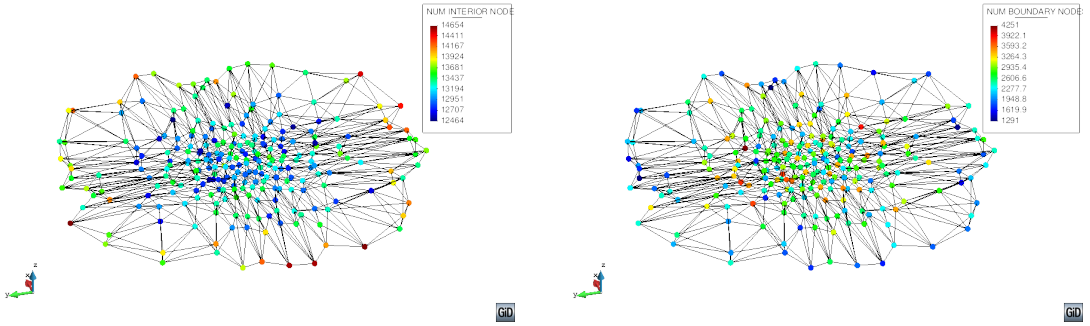
Simulation of a wind-farm. The simulation of a wind-farm, Figure 24, has first been chosen as a candidate for a detailed analysis of `Maphys` solver in the frame of the Alya simulation code. This simulation involves the Navier-Stokes equations together with a k-ϵ turbulence model.

The mesh consists of a circled and flat domain with boundary layer elements. Only HEX08 elements are used for its discretization. The basic mesh contains 3.7M elements, 3.8M nodes. An example of domain decomposition on 255 subdomains is given by Figure 25, where one can observe that the domain decomposition is almost 2-dimensional.

The equation solved by `Maphys` in this test case is the pressure equation. Its discretization leads to a SPD linear system. In this case, it is possible to consider the use of Alya's deflated CG and of `Maphys` coarse space.

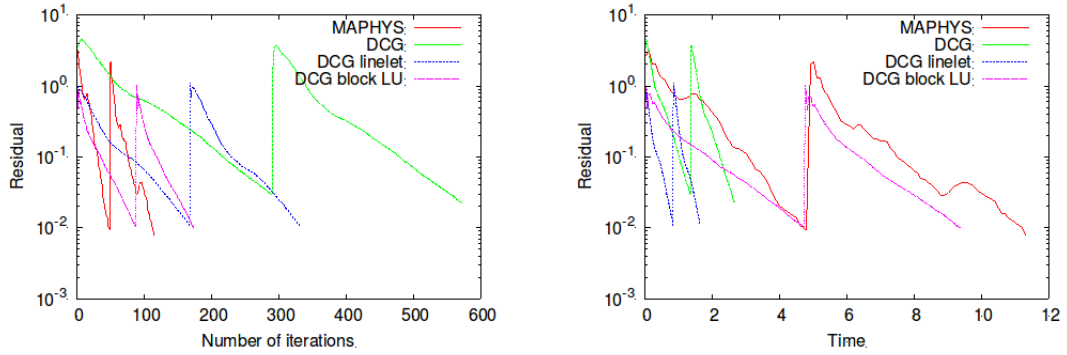
For more details on the simulation of a wind-farm into Alya, please refer to [9].

Figure 26 shows a convergence history on this wind-farm test case, simulated on



(a) Number of interior vertices per domain (b) Number of interface vertices per domain

Figure 25: Wind-farm test case: pseudo-2D domain decomposition into 255 subdomains.

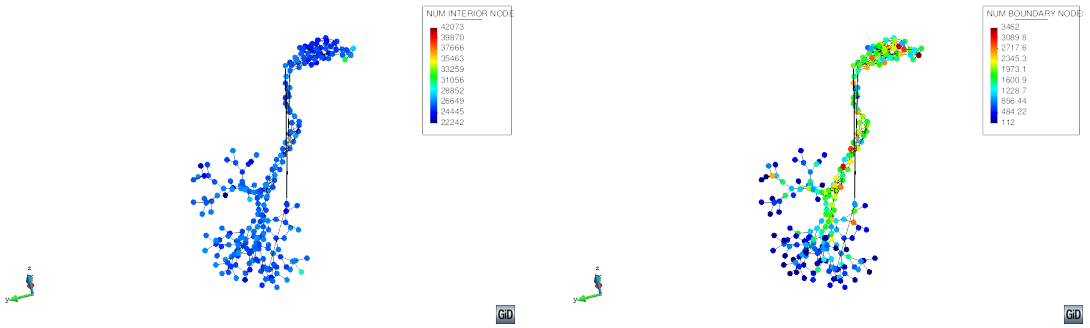


(a) Number of iterations (b) Time

Figure 26: Wind-farm test case: convergence history on 511 subdomains. `Maphys` without factorization time.

512 computational cores. The figure plots the residual as a function of the number of iterations on the left and the time to solution on the right. For the `Maphys` solver (in red), the basic (without coarse space) configuration has been considered, with a local dense preconditioning technique. For Alya, the deflated CG algorithm has been employed, jointly with three preconditioning techniques: deflation (green), parallel linelet (blue) and the block LU (purple). As can be seen on the left figure, the number of iterations is lower for `Maphys` than for any of Alya's deflated CG version. However, on the right figure, the time to solution for `Maphys` is approximatively 5 times larger than the best Alya's deflated CG configuration.

These last results motivate the need of a performant and scalable coarse space correction study into `Maphys` . This study has been performed on a more suitable test case for coarse space or deflation technique study, allowing to better illustrate the benefit of using the coarse space mechanism or deflation technique. This other test case involves a pseudo-1D domain decomposition instead of the wind-farm test case's pseudo-2D one, and is the topic of the next paragraph. Indeed, for a 1D decomposition the coarse space correction plays a critical role on the numerical behavior since the condition number growths linearly with the number of domains, while the growth is $\mathcal{O}((\#domains)^{\frac{1}{2}})$ $(\mathcal{O}((\#idomains)^{\frac{1}{3}}))$ for 2D-decomposition (resp. 3D-decomposition). Because we do not have yet access to very large computer with large number of cores we prefer to consider 1D-decomposition where the critical numerical behavior will be easy to observe already for a moderated number of



(a) Number of interior vertices per domain

(b) Number of interface vertices per domain

Figure 27: Respiratory test case: pseudo-1D domain decomposition into 255 subdomains.

cores.

Simulation of the airflow through the nose: the Respiratory test case. The simulation of the airflow through the nose has been chosen to perform an evaluation of different coarse space implementations into `Maphys` . This test case simulates the airflow through the nose and large airways by solving the incompressible Navier-Stokes equations.

Three types of elements are in use for the mesh discretization: TET04, PYR05 and PEN06, for a total of 17.7M elements and 6.9M nodes. The mesh is characterized by a very elongated geometry with small passages in the nasal cavity, leading to a pseudo-1D elongated domain decomposition when parallelizing through partitioning the mesh, see Figure 27. This property makes this test case a very good candidate to evaluate the coarse space of `Maphys` in an applicative context.

On the algebraic solver side, the discretization of the problem leads to a coupled algebraic system to be solved at each time step. This algebraic system is split to solve independently the momentum and the continuity equations. Due to the splitting strategy, it is necessary to solve the momentum and the continuity equations twice per time step. As the problem is non-linear, the matrix changes between each time step.

The continuity equation is considered for the solver comparison study. This equation leads to the assembly of a SPD linear system. Due to the elongated geometry, low frequencies are hardly damped with a classical one level domain decomposition approach. Hence, coarse space or deflation mechanisms are investigated to solve the continuity equation.

For more details about this test case, please refer to [22].

All the simulations presented into this section have been performed on the GENCI's OCCIGEN cluster, hosted by the CINES. The part of the cluster in use is composed of 2 Dodeca-core Haswell Intel Xeon E5-2690 v3 @ 2.6 GHz nodes with 64 and 128 Go RAM per node. The code was compiled with Intel compiler version 17.0.0, and linked with the multi-threaded Intel MKL version 2017.0.0 and Intel MPI version 2017.0.0. All the runs are made such that the nodes of the cluster are fully occupied (hence the number of cores is always a multiple of 24). Notice that on the OCCIGEN cluster, memory swapping is disabled by default. The simulation campaigns were realized with the help of JUBE Benchmarking Environment, allowing to explore parameters and analyze results comfortably.

The parallel benchmarks have been performed in mono-threaded configuration, on 264, 528, 1056 and 2112 MPI processes, leading respectively to 263, 527, 1055 and 2111 subdomains in the domain decompositions (as Alya has a master process). The iterative solvers' stopping criterion is set to 10^{-6}, to be reached in a maximum of 2000 iterations. For each experiment, 10 time steps are performed, each time step requiring two sub-steps.

Results are displayed on Figure 28. This figure consists of four quadrants, showing the solver total time (Fig. 28a), the global preconditioner application time for `Maphys` (Fig. 28b), the speedups (Fig. 28c) and the efficiencies (Fig. 28d) of the solvers depending on their preconditioning strategies.

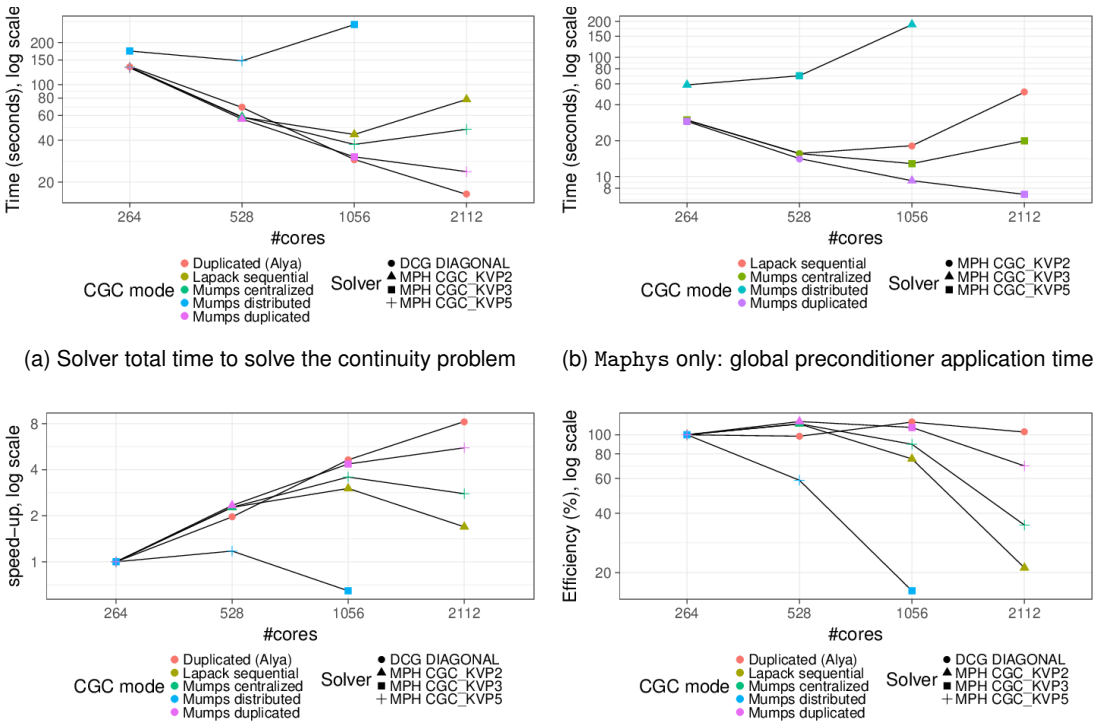
On the Alya's internal solver side, a Deflated Conjugate Gradient method is employed, together with a diagonal preconditioner, denoted `DCG DIAGONAL` on the figures. The numerical mechanism relies on an element-based aggregation to algebraically define the coarse space operator. From a parallel implementation view point, the coarse space operator is replicated on each MPI process and solved redundantly at each Conjugate Gradient iteration. In the reported experiments, the size of the coarse space is set to 10000. The corresponding Coarse Space Correction mode (CSC mode) for Alya is unique, denoted by `Duplicated (Alya)` on Figure 28.

On the `Maphys` side, several two level preconditioning techniques with coarse grid correction are considered for the iterative solution to the Schur system: `MPH CGC_KVPn` on the figures, with `n` the number of eigenvectors computed to define the coarse space contribution from each subdomain [6]. The size of the coarse problem to be solved is then $n \times \#cores$. The four formerly coarse grid correction implementations are displayed on Figure 28. For each CGC mode, only the number of eigenvalue/eigenvector pairs `n` leading to the lowest total computation time is displayed. For the `Mumps centralized`, 12 MPI processes were in use to solve the coarse problem. For the `Mumps duplicated` mode, the coarse problem has been replicated on disjoint groups of 12 MPI processes. As the matrix changes between each time step, `Maphys` has to perform several times its factorization step in order to factorize the local interior problems and to compute the local Schur complements. The preconditioner (local and coarse) are set up to remain fixed through the time steps. If necessary, it could be set up to be recomputed at a predetermined fixed frequency.

By focusing on the first `Mumps distributed` implementation of the coarse grid, one can observe on Figure 28a (in blue), that `Maphys` coarse grid correction performs poorly in front of Alya's internal deflated CG solver. Into this CGC mode, `Maphys` was not able to scale beyond 528 cores, and did not give a solution for 2112 cores (Out Of Memory (OOM) event on the compute nodes). When having a look at the performances of `Mumps distributed` CGC mode concerning the global preconditioner application on Figure 28b (still in blue), one can identify the required computation time for this part of the iterative process of `Maphys` increases with the number of processes, representing then an increasing ratio of the total computation time. The main reason of these results is that the coarse problem is solved with Mumps sparse direct solver with its distributed entry on too many MPI processes, leading to a too fine granularity hence implying poor performances.

In order to improve performances, two other CGC modes have been implemented, namely `Lapack sequential` and `Mumps centralized`. These CGC modes are displayed in greeny-yellow and in green on Figure 28. These two implementation strategies allow to compete with Alya internal solver up to 1056 cores, giving better results on both 264 and 528 cores, see Figure 28a. Notice the results for the `Mumps centralized` version become better than the

`Lapack sequential` version when increasing the number of cores. This is due to the order of the coarse problem that increases with the number of domains in use which makes it worth to exploit the sparsity pattern of the coarse matrix. However, these strategies do not scale beyond 1056 cores. This is mainly due to the global MPI communications required at the beginning and at the end of the global preconditioner application, whose computation time again increases with the number of processes, representing then an increasing ratio of the total computation time, see Figure 28b in greeny-yellow and in green.



(a) Solver total time to solve the continuity problem

(b) `Maphys` only: global preconditioner application time

(c) Solver speedup in solving the continuity equation

(d) Solver efficiencies in solving the continuity equation

Figure 28: Evaluation of `Maphys` scaling with different coarse space implementations on the respiratory test case.

To go beyond the former limitation, a last CGC mode has been implemented: `Mumps duplicated`. This coarse grid parallel implementation is closer to Alya's deflation implementation strategy, and allows to save one global MPI communication in the global preconditioner application process of `Maphys` 's iterative solve part as a comparison to the three former parallel algorithms. On Figure 28b, in purple, one can observe this last global communication bypass allows the global preconditioner application to scale up to the 2112 cores in use for these parallel experiments with this implementation strategy. However, for the solver total time on Figure 28, there is still a gap of approximatively 10 seconds between this last version and Alya's internal solver. This gap is mainly due to the non-ideal scaling of `Maphys` ' solve phase despite the new strategy and because of the factorization phase which also scale less successfully between 1056 and 2112 than before to reach this amount of computing resources.

To sum up, the developments in the frame of this comparative study enabled to significantly improve the efficiency and the scalability potential of `Maphys` (see Figures 28c and 28d) with the use of its coarse grid correction mechanism in the case of SPD linear systems. Indeed, on 1056 cores, the first coarse grid parallel implementation, i.e. `Mumps centralized` CGC mode, led to 13 % efficiency relatively to 264 cores against 109 % with

the most performant CGC mode `Mumps duplicated`. The `Mumps duplicated` CGC mode also enabled to obtain results on 2112 cores with 69% efficiency against 34 % (respectively 21 %) for the less performant `Mumps centralized` (resp. `Lapack sequential`) CGC modes.

In the stable and currently distributed version of `Maphys` , all the hybrid solver variants are based on a Schur complement approach with different preconditioning options. While they lead to extremely robust numerical schemes, their associated computational cost might not be worth for some "easy problems". For that purpose, a fully featured parallel prototype based on python with MPI , named `ddmpy` and fully documented in [50], has been developed that demonstrated a high agility for composing domain decomposition in the aS framework. The schemes can be applied either on the original matrix (denoted K) or the Schur complement (denoted S). We display in Figure 29 the parallel performance of different variants of the novel solver package implemented in `ddmpy` a parallel prototype based on python with MPI fully documented in [50]. The 8 columns correspond to the following preconditioners:

- 0 no preconditioner.
- 0D deflation on a partition-of-unity coarse space (no additional local preconditioning).
- AS1 one-level Additve Schwarz (AS) preconditioner (no coarse correction).
- AS2 two-level Additve Schwarz preconditioner with an additive coarse correction with a partition-of-unity.
- ASD deflated Additve Schwarz preconditioner with a partition-of-unity coarse space MAS,D.
- ASGD3 deflated Additve Schwarz preconditioner with an adaptive (GenEO) coarse space ($n = 3$ eigenvectors per subdomain).
- NND deflated Neuman-Neuman (NN) preconditioner with a partition-of-unity coarse space (when applied on the Schur complement matrix S, this is the BDD method [40].
- NNGD3 deflated Neuman-Neuman preconditioner with an adaptive (GenEO) coarse space ($n = 3$ eigenvectors per subdomain).

The total time to solution is divided into 6 solver steps: Schur Factorization (blue) if the PCG solver is applied on S, the interior block $K_{I_iI_i}$ is factorized in each subdomain to compute the local Schur complement matrix S_i.

- Local Pcd Setup (dark green) if a local preconditioner (AS or NN) is used, the local preconditioner matrix Ai is computed and factorized.
- Coarse Eigen Solve (green) a generalized eigenproblem is solved for the adaptive coarse space.
- Coarse Pcd Setup (light green) the coarse matrix is assembled and factorized redundantly on each CPU core. Direct Solve (orange) the reduced right-hand side for the Schur complement is computed from the global right-hand side and the interior solution from the interface solution if the PCG is applied on the Schur complement S

- Iterative Solve (red) PCG iterations.
- The number of iterations needed to reach convergence is indicated on top of each bar plot.

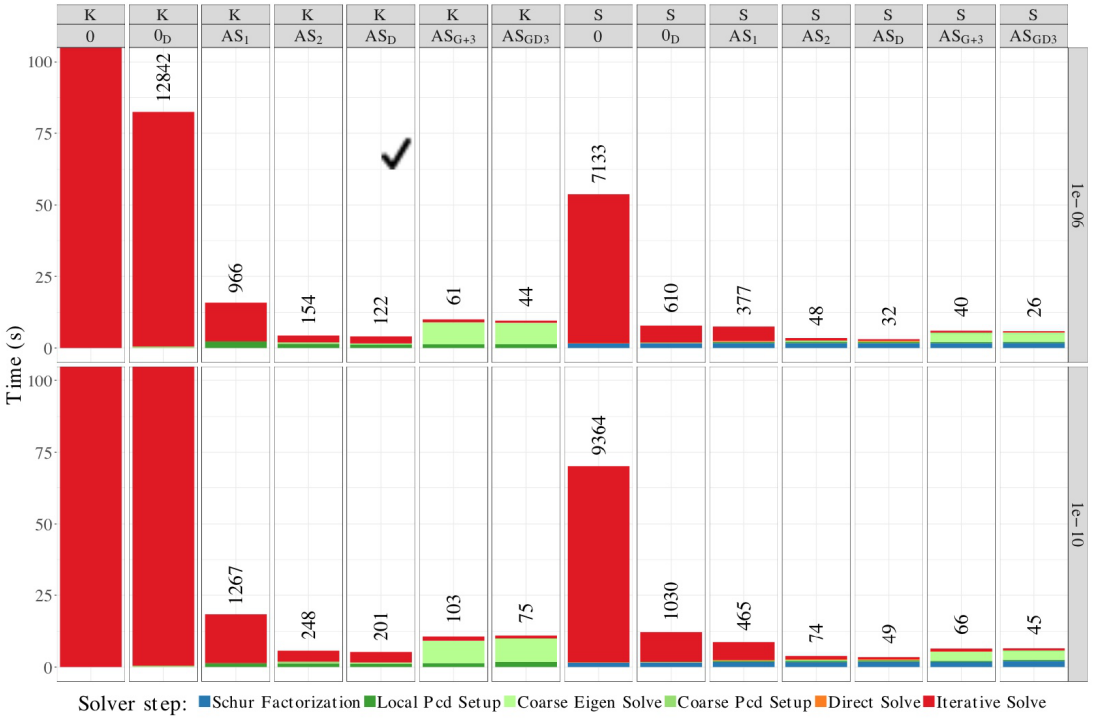


Figure 29: Step by step time for `ddmpy` on Alya test case (1,055 domains)

The availability of such a versatile package allows us to quickly and easily investigate the performance of various solution techniques and select the one best adapted to a particular simulation framework in term of bad-conditioning of the linear system, its size as well as the size of the target computer for a given Alya simulation. This motivates the current redesign of the Maphys package in modern C++, that will be benchmarked in Alya hopefully before the end of the EoCoE-2 project. In particular, the novel capabilities will allow one to work directly on the orignal matrices (option K) in the figure, that is generally a slightly less robust numerical approach, but that could be eventually a good option as it is less computationally demanding.

7. Task 3.5: Transversal activities

This Task is focused on some activities that are transversal to the Scientific Challenges and has been thought to enlarge the impact of the Linear Algebra Workpackage on the exascale transition beyond the specific needs of the flagship codes and also to promote interactions among the WP3 partners for extensions and improvements of their methods and libraries. In the following, some results related to the exploitation of the MUMPS library within a geometric multigrid solver for large-scale simulations are presented. This activity involved CNRS-IRIT and Cerfacs.

7.1 MUMPS as a coarse grid solver in HHG

Partners: CNRS-IRIT, CERFACS
Software packages: MUMPS, HHG

In this section we discuss the use of a block low-rank (BLR), sparse direct method for the solution of the coarse-grid problem in a geometric multigrid (GMG) solver. The details of this work can be found in [21].

Here, we employ the hierarchical hybrid grids (*HHG*) [32, 15] solver that implements GMG methods for the solution of linear systems and which achieves excellent performance on state-of-the-art petascale supercomputers where problems with more than 10^{13} degrees of freedom (*DOF*) have been solved. We study the solution of saddle point problems arising from the Stokes equation. Our method of choice is a monolithic MG method using an Uzawa smoother combined with a classical mildly variable multigrid V-cycle, where the number of smoothing steps is linearly increased on coarser levels.

The problems under study here permit the use of Krylov space methods as solver on the coarsest level. These methods require a number of iterations growing mildly with the size of the coarsest grid. Though asymptotically not optimal, the incurred overhead is in many cases still acceptable as long as the runtime is dominated by the multigrid processing of finer grids. However, for numerically challenging problems and when the coarsest grid size is relatively large, such simple coarse grid solvers may become a bottleneck, especially since each iteration incurs a significant overhead. In this work, we consider the use of a modern, sparse direct solver based on block low-rank (BLR) approximations, namely, MUMPS [43]. As explained in Section 6.1, the BLR method can significantly reduce the asymptotic complexity of the solver at the price of a controlled loss of accuracy [7, 8].

The use of a direct method for the solution of the coarse grid problem must be carefully designed in order to overcome potential scalability issues. The deterioration of the parallel efficiency on coarser grid levels is especially problematic in MG solvers. On coarser grid levels, the amount of computation decreases at a faster rate than the communication volume, and so the communication overhead becomes larger. To alleviate this trend, we adopted a redistribution of the grids on fewer processes which we refer to as *agglomeration*. The general idea here is to adapt the number of working processes to the size of the problem to achieve a better balance between communication and computation: the coarser the problem, the smaller the number of processes involved, in order to avoid an unnecessary large volume of communication.

The case under consideration, called *jump-410*, here is a Stokes-type problem and we refer the reader to [21] for a thorough description. Experiments were conducted on Hazel Hen, a petascale supercomputer at the HLRS in Stuttgart ranked on position 35 of the TOP500 list (November 2019) [2]. Hazel Hen is a Cray XC40 system with Haswell Intel Xeon E5-2680 v3 processors. Each compute node is a 2-socket system, where the 12 cores of each processor constitute a separate NUMA (non-uniform memory access) domain. Hazel Hen offers 64 GB per NUMA domain, which means around 5.3 GB per core. Hazel Hen uses the Cray Aries interconnect.

Table 8 shows the weak scaling of a V_{var}-cycle application on the considered problem when PMINRES is used as a coarse grid solver. The relatively high number of PMINRES iterations (in the last column) suggests that the use of a direct solver may improve the overall efficiency.

Table 9 shows the results obtained with MUMPS, either in FR or BLR mode, is used on the coarse grid, instead. Additionally MUMPS was used in double or single

proc.	DOF		jump-410					
	fine	coarse	it	total	fine	coarse	eff.	C.it
1920	$5.37 \cdot 10^9$	$9.22 \cdot 10^4$	15	1186.0	1132.6	53.4	1.00	68.13
15360	$4.29 \cdot 10^{10}$	$6.96 \cdot 10^5$	13	1188.0	1091.3	96.8	0.87	48.62
43200	$1.21 \cdot 10^{11}$	$1.94 \cdot 10^6$	14	1404.0	1241.5	162.5	0.79	48.43

Table 8: Total run-times (in seconds) of the V_{var} application: total, fine and coarse grid timings for the jump-410 problem. The number of iterations of the MG method (*it*) and the average number of iterations of the coarse grid solver (*C.it*) are also displayed.

precision arithmetic; in the case where BLR is used with a threshold that is higher than the single precision roundoff, using single precision leads to a faster execution without loss of accuracy. For the three problem sizes, the agglomeration gathered the coarse problem on 40, 160 and 225 processes, respectively. The cost of the agglomeration is included in the column labeled "coarse".

proc.	DOF		BLR ϵ	iter	time (s)				par. eff.	scaled res.
	fine	coarse			total	fine	fac.	coarse		
1920	$5.37 \cdot 10^9$	$9.22 \cdot 10^4$	Full Rank	15	1169.0	1166.1	2.4	0.46	1.00	$1.9 \cdot 10^{-17}$
			10^{-3}	15	1179.0	1175.9	2.7	0.40	0.99	$3.4 \cdot 10^{-04}$
			10^{-3} + single	15	1139.0	1136.2	2.5	0.36	1.03	$1.5 \cdot 10^{-03}$
15360	$4.29 \cdot 10^{10}$	$6.96 \cdot 10^5$	Full Rank	13	1120.0	1080.7	36.3	3.01	0.90	$3.1 \cdot 10^{-18}$
			10^{-3}	13	1117.9	1091.6	24.8	1.52	0.90	$1.4 \cdot 10^{-04}$
			10^{-3} + single	13	1091.0	1066.9	22.3	1.78	0.93	$2.4 \cdot 10^{-04}$
43200	$1.21 \cdot 10^{11}$	$1.94 \cdot 10^6$	Full Rank	14	1382.0	1197.3	176.2	8.53	0.79	$1.0 \cdot 10^{-18}$
			10^{-5}	14	1297.0	1205.7	87.1	4.36	0.84	$3.5 \cdot 10^{-07}$
			10^{-5} + single	14	1282.0	1193.6	79.3	4.30	0.85	$3.6 \cdot 10^{-07}$
			10^{-3}	19	1755.0	1671.8	78.4	4.76	0.84	$1.4 \cdot 10^{-04}$

Table 9: Weak scaling of the V_{var}–cycle with a sparse direct block low-rank coarse level solver. The parallel efficiency compares the average total run-time of each run to the average total run-time of the smallest case with no BLR.

The results in Table 9 show that the time spent at every iteration on the coarse grid is much smaller with respect to the case where PMINRES is used. Although this gain is dampened by the cost of the factorization (this is done just once prior to the beginning of the V_{var} cycle), the overall execution time is smaller.

It must be noted that these results were obtained using only MPI and no OpenMP parallelism. However, it is well known that direct methods can efficiently exploit shared memory parallelism thanks to their high arithmetic intensity. We believe that these results can be considerably improved by choosing a suitable combination of MPI and OpenMP parallelism for the direct solver; this is the object of future work.

References

[1] The alya system - large scale computational mechanics. https://www.bsc.es/es/computer-applications/alya-system.

[2] The top500 list 2019. https://www.top500.org/lists/2019/11/.

[3] Marenostrum4 user's guide, accessed June 2020. https://www.bsc.es/support/MareNostrum4-ug.pdf.

[4] Prace benchmark-suite, accessed May 2020. https://prace-ri.eu/training-support/technical-documentation/benchmark-suites/.

[5] Unified european application benchmark suite, accessed May 2020. https://repository.prace-ri.eu/git/UEABS/ueabs/.

[6] E. Agullo, L. Giraud, and L. Poirel. Robust preconditioners via generalized eigenproblems for hybrid sparse linear solvers. *SIAM Journal on Matrix Analysis and Applications*, 40(2):417–439, 2019.

[7] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary. On the complexity of the block low-rank multifrontal factorization. *SIAM J. on Scientific Computing*, 39(4):A1710–A1740, 2017.

[8] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary. Performance and scalability of the block low-rank multifrontal factorization on multicore architectures. *ACM Transaction on Mathematical Software*, 45:2:1–2:26, 2019.

[9] M. Avila, A. Folch, G. Houzeaux, B. Eguzkitza, L. Prieto, and D. Cabezón. A parallel CFD model for wind farms. *Procedia Computer Science*, 18:2157 – 2166, 2013. 2013 International Conference on Computational Science.

[10] 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, H. Zhang, and H. Zhang. *PETSc users manual. Technical report ANL-95/11 - Revision 3.7, Argonne National Laboratory*, 2016.

[11] S. Balay, W.D. Gropp, L.C. McInnes, and B.F. Smith. Efficient management of parallelism in object oriented numerical software libraries. *Modern software tools in scientific computing*, 1(1):163–202, 1997.

[12] S. R. M. Barros. The Poisson equation on the unit disk: a multigrid solver using polar coordinates. *Applied Mathematics and Computation*, 25(2):123–135, 1988.

[13] A. Bechmann, N. Sorensen, J. Berg, J. Mann, and P.E.. Rethore. The bolund experiment, part ii: flow over a steep, three-dimensional hill. *Bound Layer Meteorol.*, 141(2):245—271, 2011.

[14] T. Belytschko, L. Wing Kam, and B. Moran. *Nonlinear Finite Elements for Continua and Structures*. John Wiley & Sons, Ltd, 2000.

[15] B. Bergen and F. Hülsemann. Hierarchical hybrid grids: Data structures and core algorithms for multigrid. *Numer. Linear Algebra Appl.*, 11:279–291, 2004.

[16] D. Bertaccini and S. Filippone. Sparse approximate inverse preconditioners on high performance GPU platforms. *Comput. Math. Appl.*, 71(3):693–711, 2016.

[17] S. Börm and R. Hiptmair. Analysis of tensor product multigrid. *Numerical Algorithms*, 26(3):219–234, 2001.

[18] A. Brandt and O.E. Livne. *Multigrid techniques: 1984 guide with applications to fluid dynamics*, volume 67. Society for Industrial and Applied Mathematics, Philadelphia, 2011.

[19] H. Busing. Efficient Solution Techniques for Multi-phase Flow in Porous Media. *Springer International Publishing AG 2018*, 9(1):1–8, 2018.

[20] H. Busing, J. Willkomm, C.H. Bishof, and C. Clauser. Using exact Jacobians in an implicit newton method for solving multiphase flow in porous media. *Int. J. Computational Science and Engineering*, 9(5):499–507, 2014.

[21] A. Buttari, M. Huber, P. Leleux, T. Mary, U. Ruede, and B. Wohlmuth. Block low rank single precision coarse grid solvers for extreme scale multigrid methods. working paper or preprint, April 2020.

[22] H. Calmet, A.M. Gambaruto, A.J. Bates, M. Vázquez, G. Houzeaux, and D.J. Doorly. Large-scale CFD simulations of the transitional and turbulent regime for the large human airways during rapid inhalation. *Computers in Biology and Medicine*, 69:166 – 180, 2016.

[23] C. Clauser. *Numerical simulation of reactive flow in hot aquifers: SHEMAT and processing SHEMAT*. Springer Science & Business Media, 2003.

[24] A.M. Collier, R. Hindmarsh, A.C. Serban, and C.S. Woodward. *User Documentation for kinsol v4.1.0*. Center for Applied Scientific Computing Lawrence Livermore National Laboratory.

[25] R. Dai, P. Lin, and J. Zhang. An efficient sixth-order solution for anisotropic Poisson equation with completed Richardson extrapolation and multiscale multigrid method. *Computers & Mathematics with Applications*, 73(8):1865–1877, 2017.

[26] P. D'Ambra, D. di Serafino, and S. Filippone. MLD2P4: a package of parallel algebraic multilevel domain decomposition preconditioners in Fortran 95. *ACM Transaction on Mathematical Software*, 37(3):7–23, 2010.

[27] P. D'Ambra, D. di Serafino, and S. Filippone. MLD2P4 User's and Reference Guide, rel. 2.2, 2018. https://github.com/sfilippone/mld2p4-2/.

[28] P. D'Ambra, F. Durastante, and S. Filippone. *PSBLAS-KINSOL interface*. Consiglio Nazionale delle Ricerche, Istituti per le Applicazioni del Calcolo "M. Picone".

[29] Achilles et al. D1.4 m36 application support outcome, 2018. https://www.eocoe.eu/wp-content/uploads/2019/03/d1.4-applicationsupport.pdf.

[30] Fabulous team. Fabulous software and documentation. `https://gitlab.inria.fr/solverstack/fabulous`.

[31] S. Filippone and A. Buttari. Object-oriented techniques for sparse matrix computations in Fortran 2003. *ACM Transaction on Mathematical Software*, 38(4):23:1–23:20, 2012.

[32] F. Hülsemann, B. Bergen, and U. Rüde. Hierarchical hybrid grids as basis for parallel numerical solution of PDE. In *European Conference on Parallel Processing*, pages 840–843. Springer, 2003.

[33] M. Jung and U. Rüde. Implicit extrapolation methods for multilevel finite element computations. *SIAM Journal on Scientific Computing*, 17(1):156–179, 1996.

[34] M. Jung and U. Rüde. Implicit extrapolation methods for variable coefficient problems. *SIAM Journal on Scientific Computing*, 19(4):1109–1124, 1998.

[35] Y.J. Kim, M.G. Yoo, SH Kim, and Y.S Na. Development of vector following mesh generator for analysis of two-dimensional tokamak plasma transport. *Computer Physics Communications*, 186:31–38, 2015.

[36] M. J. Kühn, C. Kruse, and U. Rüde. Energy-minimizing, symmetric finite differences for anisotropic meshes and energy functional extrapolation. 2019. In preparation.

[37] M. J. Kühn, C. Kruse, and U. Rüde. Implicitly extrapolated geometric multigrid on disk-like domains for the gyrokinetic Poisson equation from fusion plasma applications. 2019. In preparation.

[38] V. Lehmkuhl, G. Houzeaux, H. Owen, G. Chrysokentis, and I. Rodriguez. A low-dissipation finite element scheme for scale resolving simulations of turbulent flows. *Journal of Computational Physics*, 390:51 – 65, 2019.

[39] J.Y. L'Excellent and M.W. Sid-Lakhdar. A study of shared-memory parallelism in a multifrontal solver. *Parallel Computing*, 40(3-4):34–46, 2014.

[40] J. Mandel. Balancing domain decomposition. *Communications in Numerical Methods in Engineering*, 9:233–241, 1993.

[41] MaPHyS team. MaPHyS software and documentation. `https://gitlab.inria.fr/solverstack/maphys`.

[42] N. Masthurah, Ni'mah I., Muttaqien F. H., and Sadikin R. On comparison of multigrid cycles for Poisson solver in polar plane coordinates. In *2015 9th International Conference on Telecommunication Systems Services and Applications (TSSA)*, pages 1–5, Nov 2015.

[43] MUMPS team. MUMPS: Multifrontal massively parallel solver. `http://mumps-solver.org/`.

[44] Y. Notay. AGMG software and documentation. `http://agmg.eu`.

[45] Y. Notay. Agmg performance, accessed May 2020. http://www.agmg.eu.

[46] Y. Notay, P. D'Ambra, M.J. Kühn, and P. Tamain. Co-design of la solvers, specification of characteristics and interfaces of the la solvers for all target applications, 2019. Deliverable 3.1 of EoCoE II.

[47] S. Pamela, G. Huijsmans, A.J. Thornton, A. Kirk, S.F. Smith, M. Hoelzl, T. Eich, JET Contributors, MAST Team, JOREK Team, et al. A wall-aligned grid generator for non-linear simulations of mhd instabilities in tokamak plasmas. *Computer Physics Communications*, 243:41–50, 2019.

[48] K. Pan, D. He, and H. Hu. An extrapolation cascadic multigrid method combined with a fourth-order compact scheme for 3D Poisson equation. *Journal of Scientific Computing*, 70(3):1180–1203, 2017.

[49] PaStiX team. PaStiX software and documentation. https://gitlab.inria.fr/solverstack/pastix.

[50] L. Poirel. *Algebraic domain decomposition methods for hybrid (iterative/direct) solvers*. Theses, Université de Bordeaux, November 2018.

[51] V. Rath, A. Wolf, and H. Bücker. Joint three-dimensional inversion of coupled groundwater flow and heat transfer based on automatic differentiation: Sensitivity calculation, verification, and synthetic examples. *Geophysical Journal International*, 167(1):453–466, 2006.

[52] U. Rüde. Extrapolation and related techniques for solving elliptic equations. Technical Report TUM-I9135, Institut für Informatik, TU München, 1991. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.9471.

[53] U. Rüde. The hierarchical basis extrapolation method. *SIAM Journal on Scientific and Statistical Computing*, 13(1):307–318, 1992.

[54] E. Sonnendrücker, 2019. Private communication.

[55] Nicole Spillane. *Méthodes de décomposition de domaine robustes pour les problèmes symétriques définis positifs*. PhD thesis, 2014. Thèse de doctorat dirigée par Nataf, Frédéric Mathématiques Appliquées Paris 6 2014.

[56] K. Stüben and U. Trottenberg. Multigrid methods: fundamental algorithms, model problem analysis and applications. In *Multigrid methods*, pages 1–176. Springer, 1982.

[57] S. Tikhovskaya. Solving a singularly perturbed elliptic problem by a cascadic multigrid algorithm with Richardson extrapolation. In *International Conference on Finite Difference Methods*, pages 533–541. Springer, 2018.

[58] U. Trottenberg, C. W. Oosterlee, and A. Schüller. *Multigrid*. Academic Press, London San Diego, 2001.

[59] A. Turon, E.V. González, C. Sarrado, G. Guillamet, and P. Maimí. Accurate simulation of delamination under mixed-mode loading using a cohesive model with a mode-dependent penalty stiffness. *Composite Structures*, 184:506 – 511, 2018.

[60] A. W. Vreman. An eddy-viscosity subgrid-scale model for turbulent shear flow: Algebraic theory and applications. *Physics of Fluids*, 16(10):3670—36681, 2004.

[61] E. Zoni. *Theoretical and numerical studies of gyrokinetic models for shaped tokamak plasmas*. PhD thesis, Technische Universität München, München, 2019.

[62] E. Zoni and Y. Güçlü. Solving hyperbolic-elliptic problems on singular mapped disk-like domains with the method of characteristics and spline finite elements. *Journal of Computational Physics*, 398:108889, 2019.