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

D3.4

Final versions of LA solvers with documentation and performance evaluation

Project and Deliverable Information Sheet

EoCoE	Project Ref:	INFRAEDI-824158
	Project Title:	Energy Oriented Center of Excellence : toward exascale for energy
	Project Web Site:	http://www.eocoe.eu
	Deliverable ID:	D3.4
	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:	M42 30/06/2022
	Actual Date of Delivery:	M41 31/05/2022
	EC Project Officer:	Matteo Mascagni

* - 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 :	Final versions of LA solvers with documentation and performance evaluation
	ID :	D3.4
	Available at:	http://www.eocoe.eu
	Software tool:	LaTeX
Authorship	Written by:	Pasqua D'Ambra
	Contributors:	Alfredo Buttari, Fabio Durastante, Salvatore Filippone, Carola Kruse, Yvan Notay, Herbert Owen
	Reviewed by:	Sebastian Lührs, Herbert Owen

Contents

List of Figures

List of Tables

1. Introduction

Solving Linear Algebra (LA) problems is a main computational kernel in three out of five EoCoE II Scientific Challenges (SCs) 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, named *Scalable Solvers*, was to design and implement exascale-enabled LA solvers for the selected applications and to integrate them into the flagship codes. This goal was pursued by extending general-purpose libraries, which were made available as background from the LA teams involved in the project, and sometimes by developing specific solvers for the applications. In both cases a strong co-design between LA solvers and partners from the SCs was needed.

This deliverable, as well as the previous D3.3, is not only a report, but also references *Code Demonstrators* for LA solvers. For general purpose solvers, this is in fact quite straightforward: they all have a dedicated Web page from where the code can be obtained and tested. In Section 3 below, we provide the list of these Web pages.

A large part of the WP3 results has been already described in D3.3 [16]. This last deliverable presents some new results and performance evaluations obtained in the last months of the project by some of the LA teams. New performance analysis has been carried out by using the PSCToolkit and AGMG libraries for the Water SC (see Section 4). Results of parallel implementation of GmgPolar on multi-core architectures are discussed in Section 5. Furthermore, new weak scalability results, while using PSCToolkit in Alya for the Wind SC, are discussed in Section 6. Finally, in Section 7 we reported some new experiments and comparisons of PSCToolkit, exploiting Nvidia GPU accelerators, with Nvidia LA solvers based on Algebraic MultiGrid preconditioners (AmgX) and new achievements in using MUMPS as coarsest solver in the HPDDM linear solver.

2. Acronyms

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

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
SOLEDGE3X:	Transport and turbulence in the edge plasma of tokamaks
TOKAM3X:	Transport and turbulence in the edge plasma of tokamaks
	LA software libraries
AGMG:	Iterative solution with AGgregation-based algebraic MultiGrid [15]
AMG4PSBLAS:	Algebraic MultiGrid for PSBLAS [5]
Fabulous:	Fast Accurate Block Linear Krylov Solver [9]
MaPHyS:	Massively Parallel Hybrid Solver [13]
MUMPS:	MUltifrontal Massively Parallel sparse direct Solver [14]
PaStiX:	Parallel Sparse matriX package [19]
PSBLAS:	Parallel Sparse Basic Linear Algebra Subroutines [10]
PSCToolkit:	Parallel Sparse Computation Toolkit [21]
HPDDM:	High-Performance unified framework for Domain Decomposition Methods [20]

3. Code demonstrators and documentations

	Location of code demonstrators for LA packages
AGMG:	`http://agmg.eu`
AMG4PSBLAS:	`https://psctoolkit.github.io/`
Fabulous:	`https://gitlab.inria.fr/solverstack/fabulous`
MaPHyS:	`https://gitlab.inria.fr/solverstack/maphys`
MUMPS:	`http://mumps-solver.org/`
PaStiX:	`https://gitlab.inria.fr/solverstack/pastix`
PSBLAS:	`https://psctoolkit.github.io/`
PSCToolkit:	`https://psctoolkit.github.io/`
HPDDM:	`https://github.com/hpddm/hpddm`

4. Task 3.2: Linear Algebra solvers for Water

4.1 PSCToolkit in ParFlow

Partners: CNR, UNITOV, FZJ
Software packages: PSCToolkit, AMG4PSBLAS, PSBLAS, ParFlow

We had already developed a set of interfaces between the PSCToolkit core libraries (PSBLAS and AMG4PSBLAS) to the SUNDIALS/KINSOL library, see Deliverable 3.1 and 3.2 [17, 8], and extended it to cover the new releases of the library as reported in Deliverable 3.3 [16]. We recall that this integration choice was dictated by the fact that ParFlow in turns uses KINSOL for the implementation of the Incomplete-Newton method. The advancements discussed here regard the investigation of an *update* and *reuse* strategy for the preconditioners implemented in AMG4PSBLAS to deal with sequences of changing Jacobians arising from the computation of the search directions in the Incomplete-Newton method. The aim of this effort is to reduce the number of full preconditioner builds and thus enhance the overall parallel efficiency of the code. We tested the new approach on larger mesh sizes and number of cores with respect to what was discussed in [16, Section 4.1]. The next two sections reports an analysis of the algorithmic and scalability performances for a test problem using the strategies implemented in AMG4PSBLAS. The complete set of results, together with a theoretical convergence and spectral analysis of the involved Multigrid preconditioners is contained in the preprint [1] that has been submitted and is currently under review. We emphasize here that in our work on Richards equation, as solved in Parflow, we explore some connections between the saturation of the soil and the ill conditioning of the Jacobian matrices. The information on eigenvalues justifies the effectiveness of some preconditioner approaches which are applied also in Parflow in the solution of Richards equation. Furthermore, we demonstrate the effectiveness of the linear solvers available from PSCToolkit in facing the exascale challenge with Parflow.

All the experiments of the next two sections were done by a mini-app implementing the same solution approach as in Parflow and were executed on the CPU cores, with no usage of hyperthreading, of the *Marconi-100 supercomputer* (18^{th} in the November 2021 TOP500 list [1].) *Marconi*'s nodes are built on an *IBM Power System AC922*, they contain two banks of 16 cores *IBM POWER93* 3.1 GHz processors and are equipped with 256 GB of RAM. The inter-node communication is handled by a Dual-rail *Mellanox EDR Infiniband* network by *IBM* with 220/300 GB/s of nominal and peak frequency. All the code is compiled with the `gnu/8.4.0` suite and linked against the `openmpi/4.0.3` and `openblas/0.3.9` libraries. We use only inner functionalities of the PSBLAS 3.7.0.2 and AMG4PSBLAS 1.0 libraries, with no use of optional third party libraries.

4.1.1 Weak Scalability

For complying with the physical requirements of the problem, i.e., the limited amount of data with respect to the vertical axis when compared with the area extension of the domain, we employ a 2D block

[1] See the relative list on the website www.top500.org/

decomposition of the parallelepipedal domain Ω of size $[0, L_x] \times [0, L_y] \times [0, L]$, i.e., we partition only in the horizontal directions. This domain decomposition corresponds to a mesh partitioning in which each parallel process owns all dofs in the z direction, while uniform partitioning is applied in the x and y directions.

To solve the sequence of linear system arising from the inexact Newton method we used the right preconditioned to Restarted GMRES with restarting step equal to 10 – GMRES(10) – as implemented in PSBLAS. We stop the iterations when the relative residual is less than $\eta = 10^{-7}$ or when a maximum number of iterations equal to 200 was done. Different preconditioners from AMG4PSBLAS were selected, as described in the following:

AS: the Additive Schwarz method divides the set of row/column indices of M, $\Omega^N = \{1, 2, \ldots, N\}$, into m, possibly overlapping, subsets Ω_i^N of size N_i. For each subset, we can define the restriction operator R_i which maps a vector $x \in \mathcal{R}^N$ to the vector $x_i \in \mathcal{R}^{N_i}$ made of the components of x having indices in Ω_i^N, and the corresponding prolongation operator $P_i = (R_i)^T$. The restriction of M to the subspace Ω_i^N is then defined by the Galerkin product $M_i = R_i M P_i$. The Additive Schwarz preconditioner for M is defined as the following matrix:

$$M_{AS}^{-1} = \sum_{i=1}^{m} P_i (M_i)^{-1} R_i,$$

where M_i is supposed to be nonsingular and an inverse (or an approximation of it) can be computed by an efficient algorithm.

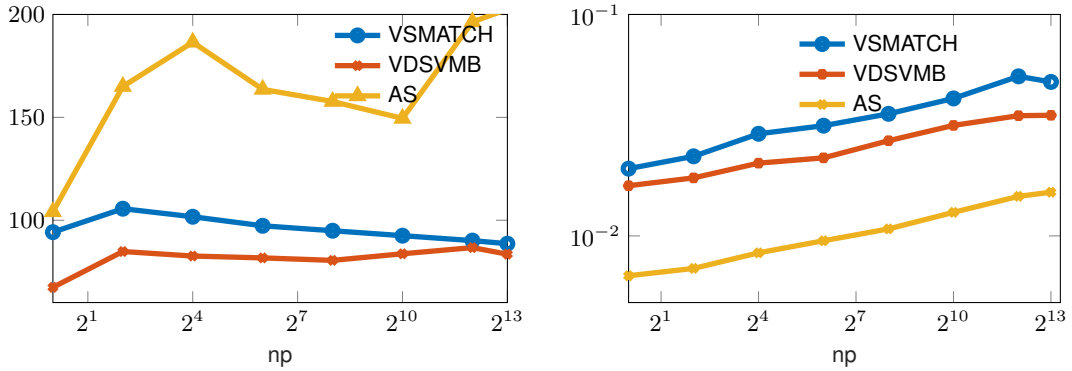
VDSVMB: the smoothed aggregation scheme introduced in [22], and applied in a parallel setting by a decoupled approach, where each process applies the coarsening algorithm to its subset of dofs, ignoring interactions with dofs owned by other processes [3]. We apply it, inside a symmetric V-cycle with one iteration of hybrid backward/forward Gauss-Seidel as pre/post-smoother at the intermediate levels. As coarsest-level solver we use a parallel iterative procedure based on the preconditioned Conjugate Gradient method with block-Jacobi as preconditioner, where ILU with one level of fill-in is applied on the local diagonal blocks. The coarsest-level iterative procedure is stopped when the relative residual is less than 10^{-4} or when a maximum number of 30 iterations is reached. In both the DSVMB and SMATCH procedures, the coarsening is stopped when the size of the coarsest matrix includes no more than 200 dofs per core;

VSMATCH: the smoothed aggregation scheme introduced in [7, 6, 4]. It relies on a parallel coupled aggregation of dofs based on a maximum weighted graph matching algorithm, where the maximum size of aggregates can be chosen in a flexible way by a user-defined parameter so that computational complexity and convergence properties of the final preconditioner can be balanced. The strategy is used with the same overall settings of the VDSVMB method with the additional request of having a maximum size of aggregates equal to 8.

In order to reduce the setup costs of the AMG preconditioners in non-linear and/or time-dependent simulations, we support a re-use strategy of operators that is often used in our Richards solution approach. Namely, we compute the multilevel hierarchy of coarser matrices only on the first Jacobian of the sequence (at the first time step), then for subsequent Jacobians we just update the smoothers on the current approximation of the matrix, while keeping fixed the coarser matrices and transfer operators. Moreover, at each subsequent time step, we reuse the Jacobian (and its related approximation) from the previous one; we leave KINSOL control over the possible need to recompute a Jacobian at each new Newton iteration.

To perform a meaningful weak scaling with respect to the physical properties of the underlying problem, we consider a growing domain $\Omega(np) = [0, 2^p \times 4.0] \times [0, 2^q \times 4.0] \times [0, 1.0]$ splitted on $np = p \times q$ processes for increasing $p = 0, \ldots, 7$, $q = 0, \ldots, 6$, and a corresponding mesh with a total of points equal to $N(p \times q) = (2^p N_x, 2^q N_y, N_z)$, where $N_x = N_y = 50$, and $N_z = 40$. We run experiments up to 8192 CPU cores for solving a problem with a global size up to about 829 millions of dofs.

From Figure 1 we observe that the AS method has the worst algorithmic scalability, showing a general increase of the linear iterations needed to solve problems of increasing size. The average number



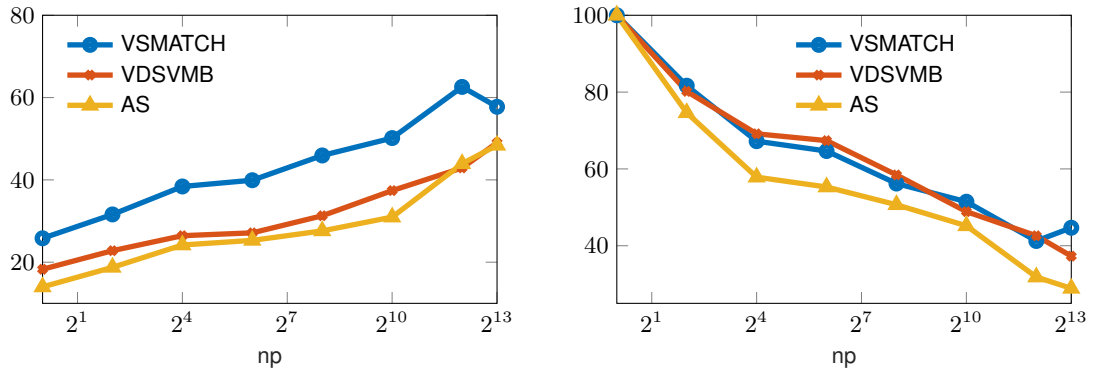
(a) Average number of linear iteration for the overall simulation

(b) Execution Time per linear iteration $T(s)$

Figure 1: Weak scalability (KINSOL/PSCToolkit). Average number of linear iterations and execution time per linear iteration with different preconditioners.

of iterations has a slow and small decrease from 16 to 1024 cores, while there is a rapid increase from 150 to 203 iterations going from 1024 up to 8192 cores, confirming that convergence properties of the one-level Schwarz-type domain decomposition methods are dependent on the number of involved subdomains. On the other hand, if we look at the AMG preconditioners, where a better coupling among the subdomains is considered, we see that the average number of iterations is smaller and has a very small variation for increasing number of subdomains. In particular, we observe that for the VDSVMB preconditioner the average number of linear iterations has an increase from 67 to 84 iterations going from 1 to 8192 cores. The VSMATCH preconditioner, after an initial small increase, displays a decrease which shows that the coupled aggregation algorithm based on matching is able to produce an effective preconditioner with very good algorithmic scalability properties.

The best time to solution (Figure 2) is in general obtained by using the AS preconditioner up to 1024 cores, due to the smaller computational cost of this preconditioner. On the other hand, its worse algorithmic scalability (i.e., the large increase in the number of linear iterations) results in larger or comparable global execution times with respect to the most effective VDSVMB preconditioner when the number of cores increases. If we look at the scaled efficiency shown in percentage scale in Figure 2b, the



(a) Total time to solution $T(s)$

(b) Scaled Efficiency

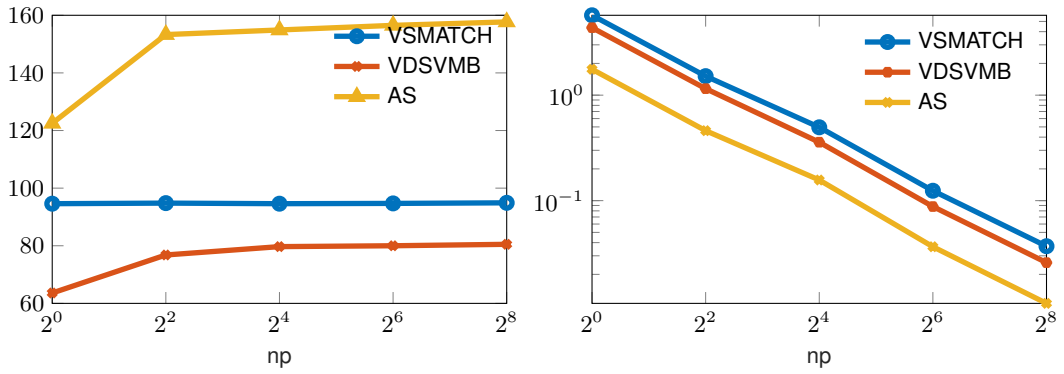
Figure 2: Weak scalability (KINSOL/PSCToolkit): Total execution time and Scaled efficiency shown in percentage.

AMG preconditioners show a similar behavior, while VSMATCH confirms its better efficiency when the largest number of cores is used.

4.1.2 Strong Scalability

We fix the target domain as the parallelepiped $[0, 64] \times [0, 64] \times [0, 1]$, discretized with $N_x = N_y = 800$ mesh points in the x and y directions, and $N_z = 40$ mesh points in the vertical direction, for a total number of 20 millions of dofs, on 1 to 256 computational cores; in particular we used a number of cores $np = 4^p$ $p = 0, \ldots, 4$, so that the computational domain is uniformly partitioned in an increasing number of vertical subdomains with square basis, for increasing number of parallel cores.

We tested the same preconditioners used in the weak scalability analysis for the restarted GMRES(10) with the same parameters. We focused again on the algorithmic performances – average iteration numbers with respect to the increasing number of cores, and on the efficiency of the solve phase measured in terms of total times (s) and time-per-iteration. From Figure 3 we observe that, as expected, the AMG

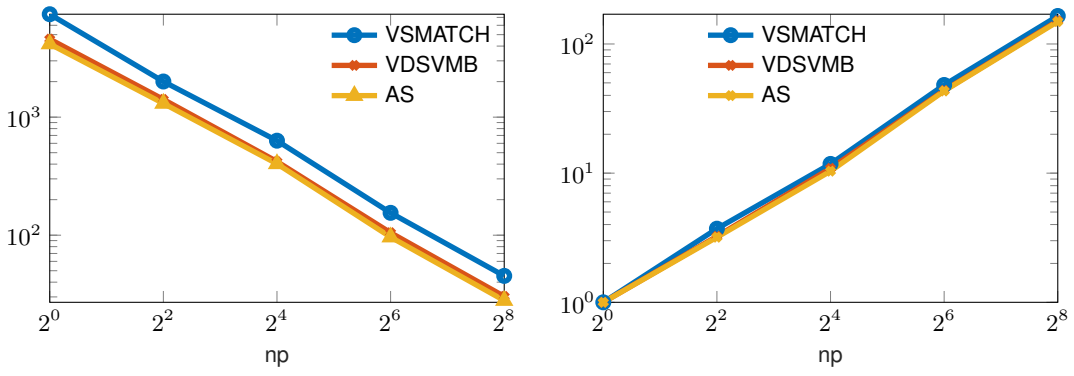


(a) Average number of linear iteration for the overall simulation

(b) Execution Time per linear iteration $T(s)$

Figure 3: Strong scalability (KINSOL/PSCToolkit). Average number of linear iterations and execution time per linear iteration with different preconditioners.

preconditioners require a smaller number of iterations than the AS method; the latter shows an increase in the number of iterations as the number of cores, and therefore of subdomains, increases. VDSVMB always requires the smallest number of iterations showing the ability of the DSVMB coarsening procedure to setup a good quality matrix hierarchy on the first Jacobian. On the other hand, we observe that when increasing the number of cores, the number of linear iterations also increases, due to the decoupled parallel approach of the coarsening. The VSMATCH algorithm, thanks to the coupled approach, produces a number of iterations that is essentially unaffected by the number of processes, even though it is always larger than VDSVMB. If we look at the time per linear iteration in Figure 3 (b), we observe, as expected, that the AS



(a) Total time to solution $T(s)$

(b) Speedup

Figure 4: Strong scalability (KINSOL/PSCToolkit). Total execution time and Speedup

preconditioner has the smallest time per iteration. Indeed, due to its one-level nature, its application cost is smaller than that of the multigrid methods. It also shows a regular decreasing in time for increasing number of parallel cores. If we look at the time per iteration of the AMG preconditioners, we observe that VDSVMB and VSMATCH have very similar behavior, and a regular decrease for larger number of cores. On the other hand, VDSVMB always achieves a smaller time per iteration than VSMATCH, due to its better coarsening ratio. Indeed, it coarsens the original fine matrix in an efficient way, producing coarse matrices that are both smaller and of good quality. This feature makes the VDSVMB preconditioner competitive with respect to the AS method. Both preconditioners produce similar global solution times, as shown in Figure 4. We can observe that, in all the cases, the overall simulation has a regular decreasing computational time for increasing number of cores, leading to a global speedup ranging from 150 to 169, depending on the preconditioner, on 256 computational cores. This corresponds to a satisfactory parallel efficiency ranging from 59% to 66%.

4.2 AGMG for the SHEMAT-Suite

Partners: RWTH, ULB
Software packages: AGMG, SHEMAT

The initial plan was to make an interface between AGMG and PETSc[2], an US package that contains various solvers. This package is used by PETSHEM, a code for multi-phase and multi-components flow simulations originating from SHEMAT. This way AGMG would be available to PETSHEM as well. However, after discussion between the partners, it has been found both more productive and more coherent with respect to the objectives of the scientific tasks to directly interface AGMG within the SHEMAT-Suite for single-phase heat flow simulations in porous media.

In the following we report on the integration procedure and on the comparison of AGMG and the numerical solver BiCGStab (BiConjugate Gradient STABilized method) for solving conductive heat flow problems in geothermal applications.

4.2.1 Integration into the SHEMAT-Suite code

A FORTRAN interface for AGMG has been integrated into the SHEMAT-Suite code. The AGMG solver has been integrated in SHEMAT-Suite as one solver option which is defined by the user in the input file. In addition, the solver tolerance has to be defined by the user in the SHEMAT-Suite input file. The provided AGMG routine differs from the BiCGStab solver implemented in SHEMAT in terms of the used relative stopping criterion. Therefore, both solvers are not directly comparable. For better comparability, the AGMG solver routine was adapted to match the tuned BiCGStab relative stopping criterion as described in the following paragraph.

Stopping criterion adaption

The original relative stopping criterion in AGMG is

$$\|A \cdot X - F\| \leq TOL \cdot \|F\| \tag{1}$$

where A is the matrix, X is the computed solution (or an initial guess), F is the right-hand side vector, and TOL is the user defined relative solver tolerance. Whereas for the BiCGStab implementation in SHEMAT-Suite it is

$$res \leq TOL \cdot res0 \tag{2}$$

[2] `https://www.mcs.anl.gov/petsc/`

where $res = \|A \cdot X - F\|$ and $res0 = \sqrt{(2 \cdot e_old)}$, with e_old being the last "best" res^2 from the previous iteration. The AGMG criterion is adapted in the following way to equal the BiCGSTab criterion:

$$t = TOL \cdot \frac{res0}{\|F\|} \tag{3}$$

Thus, when applying its usual policy, AGMG will check:

$$\|A \cdot X - F\| \leq t \cdot \|F\| = (TOL \cdot \frac{res0}{\|F\|}) \cdot \|F\| = TOL \cdot res0 \tag{4}$$

4.2.2 Solver test and comparison

In the following, AGMG is compared to the BiCGStab solver implemented in SHEMAT-Suite using a 2D and a 3D test model. We use the AGMG solver with the adapted relative stopping criterion and the option $ijob = 10$ for the tests. The BiCGStab solver uses its relative stopping criterion, and the incomplete LU factorization (ILU) for preconditioning. Both test models solve conductive heat flow problems, with coupling of temperature and pressure (e.g. fluid density is a function of pressure and temperature) for three sub-settings that differ in complexity:

1. Solution of the heat flow equation with a given constant pressure of 0.1 MPa.
2. Solution of the heat flow equation with a given natural pressure gradient with depth.
3. Coupled solution of heat and fluid flow equations, where the fluid flow equation is always solved with the BiCGStab solver.

In the following, these sub-settings are referred to as model variants 1, 2 and 3, respectively.

2D test model

The 2D testmodel has the following characteristics:

- Model domain: 500 m in x-direction and 500 m deep, with 1 m discretization
- Constant temperature and pressure at the top boundary (Dirichlet BC)
- Constant heat flow at the bottom boundary (Neuman BC) (0.07 $\mathrm{W\,m^{-2}}$)

The three model variants are simulated subsequently with the BiCGStab solver and the AGMG solver respectively for solving the heat flow equation. Both solvers use the same relative tolerance of 10^{-10}. The difference in temperature solution between both solvers is in the range of 10^{-11} K for pure temperature simulation and in the range of 10^{-9} K for coupled temperature and pressure simulation. Figure 5 shows the relative differences between the solutions, normalized to the average temperature of the respective BiCGStab solution (around 18.9 °C). It is in the range of 10^{-13} for pure temperature simulation (variant 1) and in the range of 10^{-11} for coupled temperature and pressure simulation (variants 2 and 3). This means that the difference induced by the change of the linear solver is below model precision, i.e. both linear solvers give the same results. Tests were ran on an intel-R2208WFTZS at 2.1GHz inside 2-socket nodes with 384 GB memory, where each socket contains 24 cores.

Tables 1, 2 and 3 present the required number of iterations and respective linear solver times per outer iteration (i.e. nonlinear Picard iterations) for both solvers for the three model variants. The AGMG solver needs considerably less iterations than the BiCGStab solver. Thus, it requires much less CPU time, as each solver iteration takes roughly the same time for both solvers. On average, one AGMG iteration takes about 0.03 s and one BiCGStab iteration takes about 0.02 s. AGMG needs less than 5 % of the time of the BiCGStab solver for the presented simulations. The speedup due to the AGMG solver is $S = t_{AGMG}/t_{BiCG}$, with a total linear solver time t. The average speedup due to the AGMG solver for the three variants of the 2D testmodel is around 24.

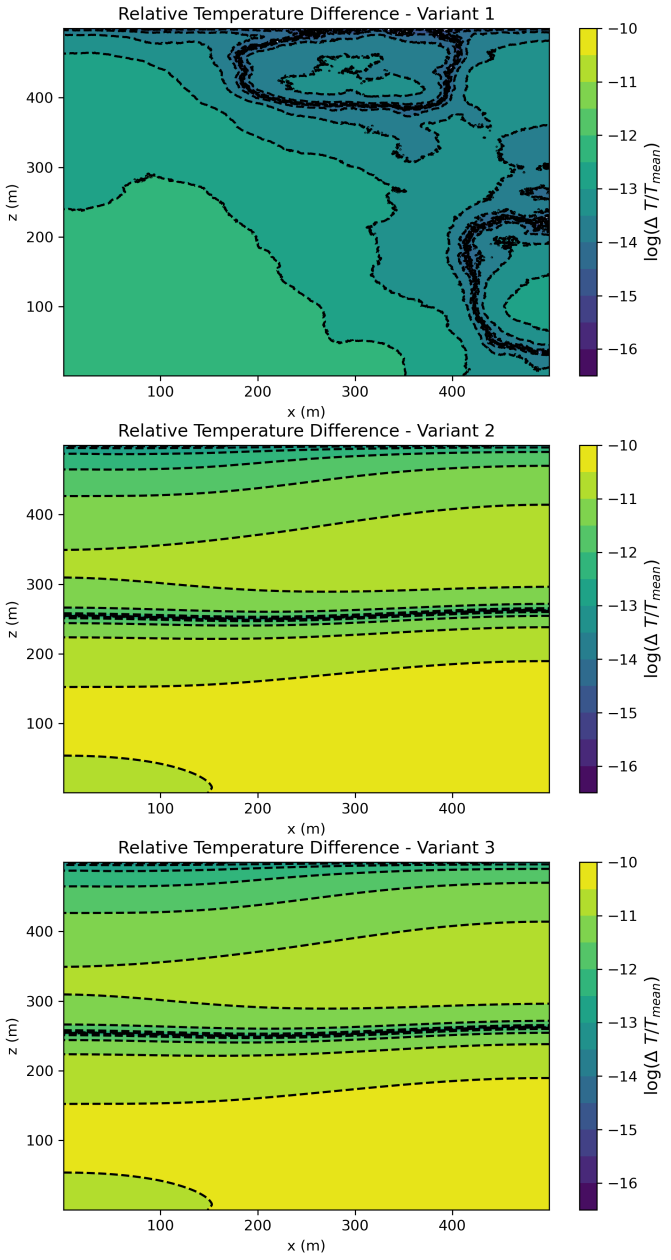


Figure 5: Relative temperature difference between the AGMG and the BiCGStab solutions in the 2D model domain for variants 1 (top) to 3 (bottom), depicted as its logarithm.

	BiCGStab		AGMG	
nliter	# iterations	solver time (s)	# iterations	solver time (s)
1	510	10.89	22	0.70
2	680	14.51	20	0.64
3	605	12.63	18	0.55
4	590	12.57	17	0.50
5	651	14.04	15	0.46
6	592	12.43	14	0.46
Total	3628	77.07	106	3.31

Table 1: Detailed overview of linear system solver times for variant 1 - the initial temperature simulation with constant pressure.

	BiCGStab		AGMG	
nliter	# iterations	solver time (s)	# iterations	solver time (s)
1	660	13.95	19	0.64
2	595	12.44	18	0.57
3	579	11.89	16	0.50
4	603	12.21	15	0.43
Total	2437	50.49	68	2.14

Table 2: Detailed overview of linear system solver times for variant 2 - temperature simulation with natural pressure gradient.

	BiCGStab		AGMG	
nliter	# iterations	solver time (s)	# iterations	solver time (s)
1	660	14.12	19	0.56
2	595	12.53	18	0.53
3	579	12.01	16	0.51
4	603	12.30	15	0.47
Total	2437	50.96	68	2.07

Table 3: Detailed overview of linear system solver times for variant 3 - coupled temperature and pressure simulation.

3D test model

For testing the functionality and performance of the AGMG solver integration in SHEMAT-Suite in 3D, a model with the following characteristics is used:

- 3D model domain with $107 \times 113 \times 70 = 846370$ grid cells, 150 m discretization in x- and y-direction and 50 m discretization in z-direction.
- Four model units with different thermal properties (see Fig. 6a)).
- Constant temperature in cells that represent the topographic surface (Dirichlet BC).
- Spatially varying basal heat flow at the model base (Neumann BC).

We subsequently simulated model variants 2 and 3 with a relative solver tolerance of 10^{-8} for both AGMG and BiCGStab. In several outer iterations, the BiCGStab solver stopped before final precision was reached, because the residuum did not change any more.

Figure 6 presents the relative differences between the solutions, normalized to the average temperature of the respective BiCGStab solution (around 37.4 °C). The coupled simulation (variant 3) reveals a larger difference between the two linear solvers than the simpler variant 2, where only the temperature equation is solved. However, the difference is still in the range of model precision, concluding that both linear solvers give the same results.

Tables 4 and 5 show the linear system solver times for both solvers for variants 2 and 3 respectively. For variant 3, the linear solver time for the heat flow solution is less than for variant 2, because it uses the solution from variant 2 as the initial temperature field. When simulating variant 3 without this initially simulated temperature field, using a constant initial temperature of 20 °C instead, the linear solver time is longer. Additionally, the resulting temperature differs by maximum 8.3×10^{-7} K. This is larger than the difference between the two solvers. The average speedup due to the AGMG solver for both variants of the 3D testmodel is around 3.

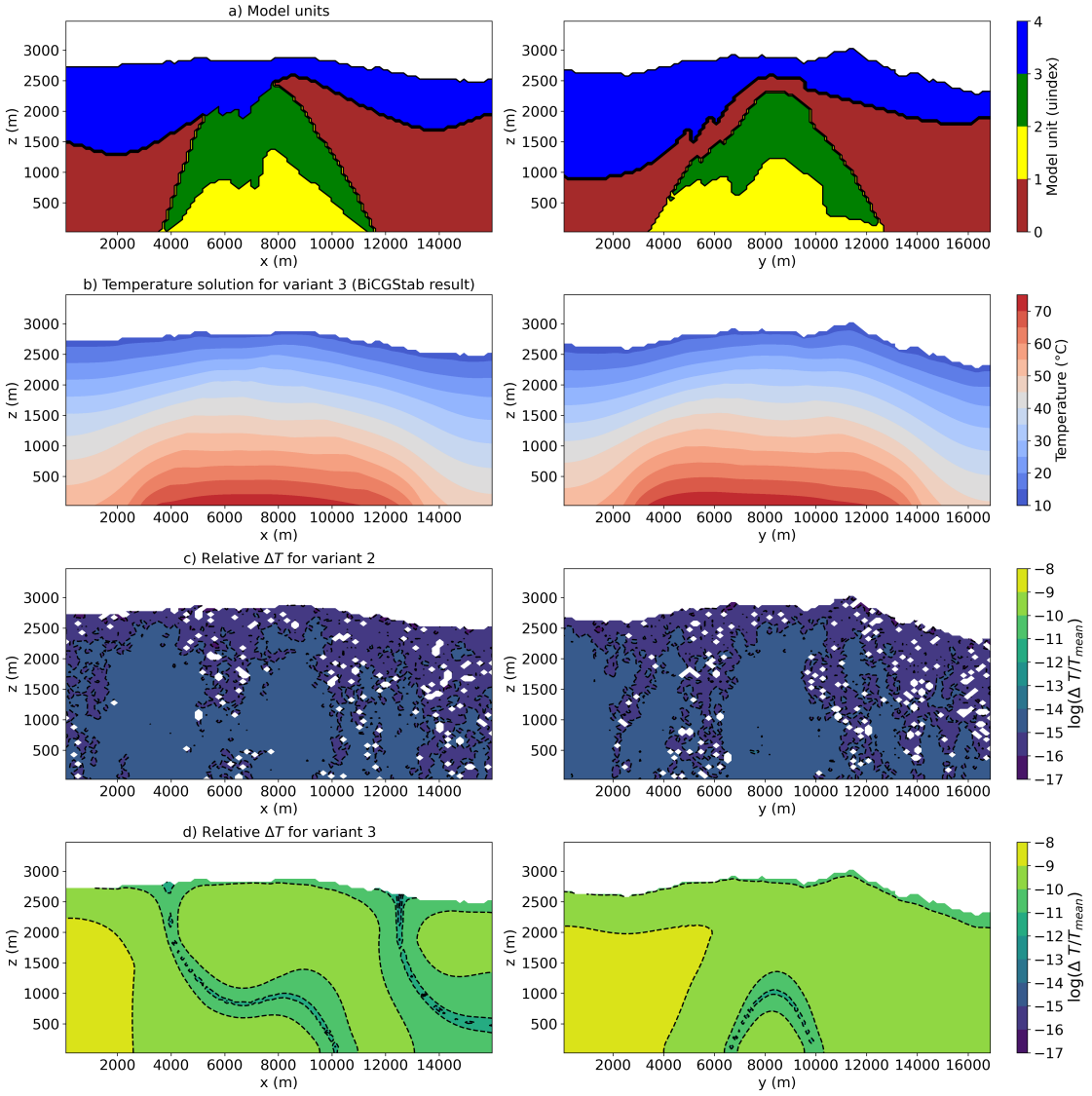


Figure 6: Results of the 3D test model along cross-sections in x-direction (left) and y-direction (right) through the center of the model domain. a) Geometry of the four model units that have different thermal properties. b) Simulated temperature solution for variant 3 obtained with BiCGStab. Note that variant 2 has nearly the same solution, since there is no advection. c) and d) Relative temperature difference between the AGMG and the BiCGStab temperature solutions for variants 2 and 3 depicted as its logarithm. White areas represent zero temperature difference.

	BiCGStab		AGMG	
nliter	# iterations	solver time (s)	# iterations	solver time (s)
1	102	7.27	28	2.90
2	104	7.75	27	2.90
3	107	7.49	26	2.64
4	100	6.95	24	2.52
5	144	10.33	22	2.32
6	118	8.13	19	2.09
7	101	6.93	17	1.84
Total	776	54.85	163	17.21

Table 4: Detailed overview of linear system solver times for the 3D testmodel variant 2.

	BiCGStab		AGMG	
nliter	# iterations	solver time (s)	# iterations	solver time (s)
1	110	7.51	26	3.18
2	163	11.87	23	2.75
3	125	8.94	21	2.62
4	105	7.39	19	2.50
5	105	7.32	16	2.14
Total	608	43.03	105	13.19

Table 5: Detailed overview of linear system solver times for the 3D testmodel variant 3 - coupled heat and fluid flow simulation.

		BiCGStab		AGMG	
timestep	nliter	# iterations	solver time (s)	# iterations	solver time (s)
1	1	7	0.61	2	0.23
2	1	7	0.62	2	0.23
3	1	7	0.61	2	0.18
4	1	2	0.23	3	0.22
	2	7	0.61	2	0.18
	3	8	0.67	2	0.20
5	1	2	0.22	3	0.21
	2	7	0.60	2	0.17
	3	7	0.61	2	0.17
Total		54	4.78	20	1.79

Table 6: Detailed overview of linear system solver times for the transient 3D testmodel - heat flow simulation for five timesteps.

Transient Simulation

The previous tests were steady state simulations. In addition, AGMG was tested for a transient simulation of heat flow for the 3D model described in 4.2.2 (variant 2). For the testing purpose, we simulate heat flow over five years discretized in five linear time steps with temporarily varying surface temperature.

Again, the difference induced by the change of the linear solver is below model precision, i.e. both solvers produce the same result. The maximum relative difference in the final temperature solutions is $4.9 \cdot 10^{-15}$. Table 6 lists the linear system solver times for both BiCGStab and AGMG, with a total linear system solver times of 4.78 s and 1.79 s, respectively. The absolute difference in simulation time is not as severe as for the steady state simulations, because BiCGStab now also needs only a few iterations per outer nonlinear iteration and time step. The speedup, however, is with 2.7 comparable to the speedup of the steady state model.

Large-scale test

After we tested the AGMG integration successfully, we use the following 3D model for testing the large-scale performance:

- 3D model with three horizontal layers with different thermal conductivity.
- Model discretization: 5 m $\times$ 5 m $\times$ 5 m
- Boundary conditions: basal heat flow at bottom, surface temperature at top.
- Steady-state solution of the heat flow equation with natural pressure gradient (variant 2)

	BiCGStab				AGMG			
# grid cells	# iterations	solver time (s)	relative solver time	time per grid cell (10^{-4} s)	# iterations	solver time (s)	relative solver time	time per grid cell (10^{-5} s)
$5.4 \cdot 10^5$	2796	116.63	1.00	2.16	105	9.73	1.00	1.80
$5.4 \cdot 10^6$	2543	1101.34	9.44	2.04	88	68.02	6.99	1.26
$1.08 \cdot 10^7$	2479	2113.46	18.12	1.96	88	140.34	14.42	1.30

Table 7: Performance of the two linear solvers for the largescale 3D testmodel with increasing number of grid cells.

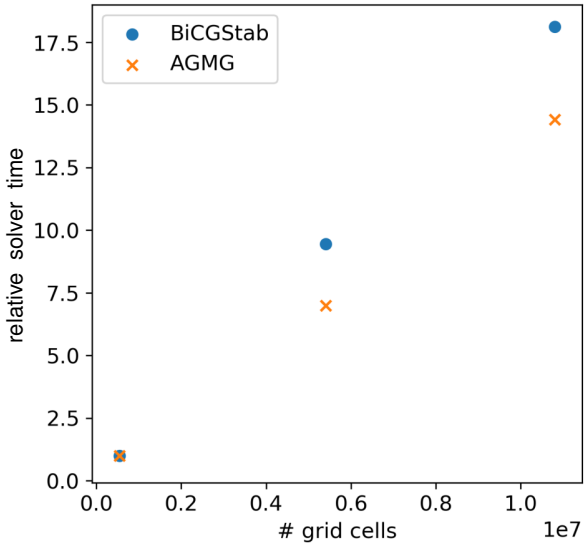


Figure 7: Solver time relative to the time for the smallest model with increasing number of unknowns for the 3D heat flow simulation (blue circles - BiCGStab, orange crosses - AGMG).

- Relative tolerance for linear solver: 10^{-8}

The basic model characteristics, such as depth, rock properties, and boundary conditions stay the same while the number of cells is increased subsequently in three steps from $5.4 \cdot 10^5$ cells to $1.08 \cdot 10^7$ cells. Table 7 and Fig. 7 show the resulting linear solver times with increasing number of grid cells.

Overall, AGMG requires considerably less compute time than BiCGStab for solving the heat flow problem in the large-scale, finely discretized SHEMAT-Suite models. Even for the largest model with $1.08 \cdot 10^7$ cells, AGMG needs less than three minutes, whereas BiCGStab requires around 35 minutes for solving the linear system. The speedup due to the AGMG solver is between 12 and 16. The linear increase in compute time with increasing number of grid cells is lower for AGMG than for BiCGSTab. This is illustrated by the solver time relative to the time for the smallest model, shown in Figure 7. Each solver shows a nearly constant time per grid cell or unknown (see Table 7), which is expected for optimal order methods. Thus, it can be concluded that both BiCGStab and AGMG within SHEMAT show near-optimal behavior for conductive heat flow problems.

In addition, the 3D largescale model reveals that the BiCGStab solver is sensitive to the spacial model discretization, whereas AGMG is more robust. This can be seen when comparing the performance of the 3D largescale model with $5.4 \cdot 10^5$ cells to the 3D testmodel variant 2 from Section 4.2.2 (see Tables 5 and 7). The coarser discretized model from Section 4.2.2 (150 m $\times$ 150 m $\times$ 50 m discretization) needs considerably less time and iterations than this model, which has a much finer discretization of 5 m $\times$ 5 m $\times$ 5 m, although both models have a comparable number of unknowns. The linear solver time per grid cell differs by one order of magnitude (5×10^{-5} s vs. 2×10^{-4} s). In contrast, the AGMG solver is robust to spatial discretization; the required linear solver time per grid cell is nearly the same for both models (around 2×10^{-5} s). The speedup due to AGMG is around 10 times larger for the 3D model with fine

discretization than for the coarsely discretized 3D test model from Section 4.2.2.

4.2.3 Summary and Conclusions

The presented tests show that AGMG is integrated into SHEMA-Suite successfully as a linear system solver for conductive heat flow problems, which have to be solved in context of geothermal reservoir simulation. AGMG gives the same results as the BiCGStab solver that is usually used in SHEMAT-Suite for all presented test models comprising 2D, 3D, steady-state and transient simulations. The AGMG solver performs considerably better than the BiCGStab solver in terms of required solver iterations and computing time. The speedup due to AGMG is between 2.7 and 24.6 for the presented simulations. The maximum speedup is observed for the 2D model that has a very fine discretization of 1 m, whereas the minimum speedup is observed for the 3D model with coarse discretization of 150 m in x- and y-direction and 50 m in z-tirection. It seems that especially models with a fine spatial discretization benefit from the AGMG solver. Comparison between the two 3D models with similar number of unknowns but different spatial discretization reveals that AGMG is robust to the discretization, whereas BiCG needs more time the finer the discretization. In addition, the largescale test revealed that AGMG shows a nearly constant time per unknown, which is near-optimal behavior.

The observed speedup by the AGMG solver is promising for production runs with SHEMAT-Suite that are comparable to the large-scale models presented here. Especially, finer discretized problems can benefit from the AGMG solver. In addition, Monte Carlo simulations are often required for stochastic uncertainty analysis, where several hundred or even thousand solutions have to be obtained. Thus, much computing time will be saved by using the AGMG solver in SHEMAT-Suite.

5. Task 3.3: Linear Algebra solvers for Fusion

5.1 Geometric Multigrid Solver for Plasma Fusion Simulations in GyselaX

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

The goal of this collaboration between CERFACS, MPG-IPP and IRFM-CEA was to develop a fast iterative solver for the solution of the 2D gyrokinetic Poisson equation for possible integration into the flagship code GyselaX. In order to satisfy requests from the application side (specific geometries, equation coefficients, etc.) and, to design a tailored PDE solver, meetings between the respective partners have been continually held all along the project. For the simulation of plasma movements in a tokamak, a 5D Vlasov equation and a 3D Poisson equation are coupled, where the solution of each one provides input for the other one at each time step of the simulation. The 3D Poisson equation may be reduced to a 2D gyrokinetic Poisson equation that is solved on a large number of disk-like cross sections of the tokamak geometry. Here, the need for an efficient parallelism clearly arises. Simulation codes such as the flagship code GyselaX handles this need through a hybrid parallelism: distributed parallelism for the simultaneous solution across the cross sections, coupled with node level parallelism for the solution of the Poisson problem on each cross section. For the solution of the latter problem, an algorithm that shows a high potential for parallelism, and a low computational complexity, is required. Geometric multigrid methods are among the best candidates to achieve this.

In the first part of EoCoE II, we defined an energy minimizing symmetric finite difference method to discretize the gyrokinetic Poisson problem [11]. The difficulty was to integrate the variable coefficients present in the PDE equation, and to address the deformed disk-like geometry of the cross sections. Second, we developed a geometric multigrid algorithm for the physical geometry of the cross sections represented in polar coordinates, or via transformations leading to more realistic stretched domains [12]. The solver was designed to obtain an optimal memory consumption and computational complexity, i.e. linear with respect to the problem size, as well as being suited for parallelization. Furthermore, we included an implicit

extrapolation technique, which increases the order of approximation from 2, as typically observed for finite difference discretizations of the Poisson equation, to an order between 3 and 4. Cubic order convergence would, for example, be expected for quadratic finite elements, but these have the disadvantage of a high computational overhead. The proposed extrapolation technique comes, however, with minimal cost, as it requires an intelligent reordering of the system matrix and does not lead to further fill-in of the matrix. The techniques that we used for the symmetric finite difference discretization, the components of the multigrid solver (smoothing, interpolation and prolongation, ...) and details of the extrapolation technique have been described in detail in the previous deliverables [17, 8, 16]. Note that the preprints of the articles, concerning the formulation of the symmetric finite difference stencil and the new geometric multigrid algorithm with extrapolation, have now been published respectively in SISC [11] and in the Springer Journal of Scientific Computing [12]. Both journals are A-rated high-impact journals.

The prototype of the multigrid solver, called *GmgPolar* had been implemented in Matlab in the first half of EoCoE II. For the second half of the project, we focused on porting the code to C++ with an efficient and parallel implementation using OpenMP and CUDA on GPUs. The first step was then a direct translation of the Matlab code to C++ using a naive coding approach. In a second step, the code was extended to solve the complete gyrokinetic Poisson equation, i.e.

$$\begin{aligned} -\nabla \cdot (\alpha \nabla u) + \beta u &= f \quad \text{in} \quad \Omega, \\ u &= u_D \quad \text{on} \quad \partial\Omega \end{aligned} \tag{5}$$

where $\Omega \subset \mathbb{R}^2$ is a disk-like domain, $f : \Omega \to \mathbb{R}$, $f \in \mathcal{C}^0(\overline{\Omega})$, is the forcing term, $\alpha, \beta : \Omega \to \mathbb{R}$ with $\alpha \in \mathcal{C}^1(\Omega) \cap \mathcal{C}^0(\overline{\Omega})$ and $\beta \in \mathcal{C}(\overline{\Omega})$ are coefficients corresponding to *density profiles*. GmgPolar was also completely optimized in order to increase the efficiency of computation on modern computing units. For example, the matrix-vector multiplication of the system matrix A with a vector x is done 'on the fly', i.e. in a matrix free fashion. This strategy allows to reduce the memory usage, as it avoids storing the system matrix. By carefully factorizing and reordering the computation of the matrix entries, we can increase the number of instruction per cycles on the CPU, in particular thanks to vectorization. We now present some experiments performed on a single node of the cluster Kraken[3] from CERFACS. Kraken uses Skylake Intel Xeon Gold cores at 2.3GHz inside 2-socket nodes with 96 GB memory, where each socket contains 18 cores. Figure 8 gives the execution time for 10 matrix-free applications of A, with matrix sizes ranging between $4.0 \cdot 10^1$ and $8.4 \cdot 10^6$. While the execution time increases linearly with the matrix size for both implementations, we have a gain of 2 orders of magnitude once making a better use of the capabilities of modern computing.

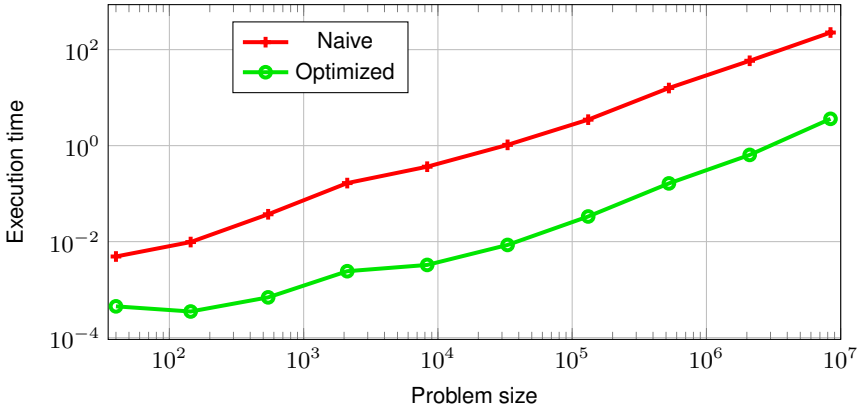


Figure 8: (GyselaX-GmgPolar) Execution time for the matrix-free application of A using a naive and an optimized approach.

Additionally, the components of GmgPolar have been designed to allow a high degree of parallelism. In the context of a possible integration of GmgPolar inside GyselaX, we have particularly focused on the

[3] `https://cerfacs.fr/les-calculateurs-du-cerfacs/`

use of OpenMP multithreading to parallelise each component of the multigrid cycle. Let's consider for example the latter. The smoother is a combination of circle and radial relaxations [12]. Our approach for the parallelization of the smoother is then very simple: we define the computation corresponding to a single row in the grid (corresponding to a fixed radius of the disk-like geometry) as a task. Then, by carefully defining dependency relations between the tasks (in particular depending on the stencil), and by using task-based parallelism as implemented in OpenMP, we can launch in parallel the smoothing procedure with a good efficiency. The same principle is used for example to parallelize the application of the matrix and the prolongation/restriction. Figure 9 gives the speed-up when executing the multigrid cycle in GmgPolar to solve a discretized problem of size $5.0 \cdot 10^7$ using 1 to 36 OpenMP threads. We also display the speed-up when considering the smoothing separately. We observe a speed-up for the smoothing of up to 27, and a total speed-up of up to 14. In particular, the parallelization of the prolongation and restriction operators should be improved in order to improve the whole solver. In the end, to solve the system of size $5.0 \cdot 10^7$ in parallel using 36 OpenMP threads, with a relative error of $4.7 \cdot 10^{-7}$ compared to the exact PDE solution, GmgPolar requires around 7 min.

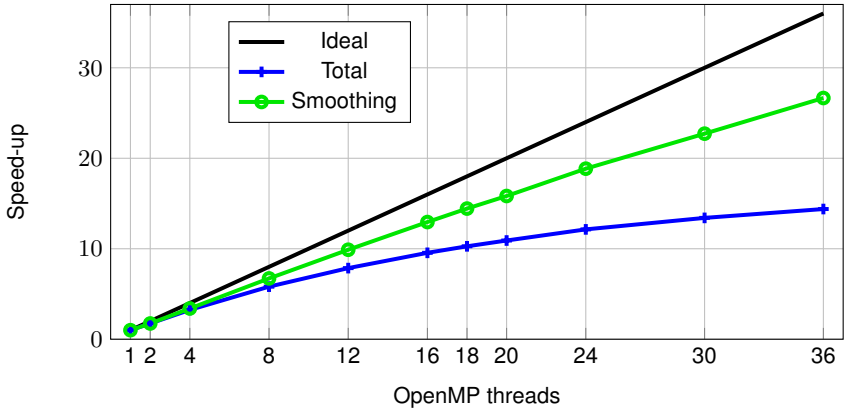


Figure 9: (GyselaX-GmgPolar) Speed-up obtained in GmgPolar using OpenMP parallelization, compared to the sequential case. We display the results obtained for the whole multigrid cycle (*total*), and for the smoothing taken separately (*smoothing*).

In September 2021, a Cerfacs team participated in the CSCS GPU Hackathon with the goal to port the C++ code onto GPUs using CUDA. Throughout this event, the supercomputer Piz Daint from the Swiss National Supercomputing Center was used. We achieved having a first running version of GmgPolar on GPUs with well-performing subroutines, however as it is usual for these kinds of implementations, time is still required for optimization and fine tuning of the overall code (e.g. the setup phase). Unfortunately, due to the short duration of the Hackathon (10 days), the still existing work-load regarding the OpenMP implementation and the lack of funding for additional human resources in the Cerfacs team, we were not able to have a highly performant version of GmgPolar. This presents an interesting task for future work though, as the multigrid algorithm in itself and the matrix-free GmgPolar code should allow an efficient GPU implementation.

The details of the computation for the complexity of the GmgPolar solver, as well as the parallelization using OpenMP and GPUs, are soon to be published in a technical report.

Transversal activities: In an attempt to compare the advantages and disadvantages of the existing spline solver in GyselaX, a solver implemented in the AMREX framework and GmgPolar applied to the cross-section geometry, we are in an ongoing close collaboration with IRFM-CEA and Katharina Kormann at Uppsala Universitet. Final results are expected to be submitted in an article by the end of EoCoE II.

6. Task 3.4: Linear Algebra solvers for Wind

6.1 CFD solver for Alya: PSCToolkit

Partners: BSC, CNR, UNITOV
Software packages: PSCToolkit, PSBLAS, AMG4PSBLAS, Alya

During the first phase of the project, as reported in the Deliverable 3.1 [17], we developed a software module included into the kernel of Alya, to interface PSBLAS (rel. 3.6) and its sibling preconditioner package to the code. This allows to Alya the exploitation of linear solvers and preconditioners from those external packages for solving linear systems arising from the different physics modules. Preliminary results obtained by testing the solvers on systems stemming from fluid dynamics simulation exploiting the NASTIN module, which deals with the incompressible Navier-Stokes equations for turbulent flows, were included in [8]. In [16] we included a comprehensive discussion of results obtained with the new versions of the software packages, as extended and improved within this project (see Transversal Activities in [16]). In details, we discussed both strong scalability and weak scalability results of the PSCToolkit solvers, focusing on the best algorithmic choices and on the comparison among the most promising available AMG preconditioners included in AMG4PSBLAS (the PSCToolkit's AMG preconditioners package). The results were obtained on the Bolund test case, for increasing number of cores up to 12288 and a mesh size up to $\approx 3.5 \times 10^8$ dofs, on the Marenostrum-4 supercomputer, operated by BSC. The above results were also presented at the 32th International Conference on Parallel Computational Fluid Dynamics [18].

In the last months we were able to run some experiments with the Bolund test case, on the cluster module of JUWELS supercomputer, operated by JSC, by increasing number of dofs up to $\approx 2.9 \times 10^9$ for increasing number of cores up to 23551 (corresponding to 512 compute nodes). JUWELS is composed of 2271 compute nodes with 2 Intel Xeon Platinum 8168 CPUs, of 24 cores each (ranked 77^o in the November 2021 TOP500 list[4], with more than 9 petaflops of peak performance). Weak scalability results of one of the most promising solvers included in PSCToolkit are discussed in the following. Comparisons with the native CG solver of Alya demonstrated the large benefits obtained in using the PSCToolkit's solver. For details on the Bolund test case we refer the reader to [16]; we only remind here that, at each time step of the simulation, we solved the symmetric positive definite linear systems arising from the pressure equation employing a flexible version of the Conjugate Gradient (FCG) method of PSBLAS, coupled with an AMG preconditioner available from AMG4PSBLAS. In detail, we used preconditioned FCG 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 $TOL = 10^{-3}$. A general row-block data distribution based on a parallel geometric partitioning using Space Filling Curve (SFC)[2] is applied for the parallel runs.

Due to the limited access to the JUWELS resources and taking into account the previous performance analysis, for our new experiments we only considered one of the most promising preconditioner implemented in AMG4PSBLAS; in particular we selected a symmetric V-cycle employing 4 iterations of the hybrid forward/backward Gauss-Seidel smoother at the intermediate levels and a coarsest solver based on the FCG method preconditioned by a block-Jacobi preconditioner with ILU(1) on the diagonal blocks; a stopping criterion relying on the reduction of the relative residual of 3 orders of magnitude or a maximum number of iterations equal to 10 is applied for this coarsest solver. The multilevel hierarchies were built by applying the new parallel coupled aggregation scheme implemented in AMG4PSBLAS, relying on the *coarsening based on compatible weighted matching* [4]. We built hierarchies characterized by aggregates of size at most 16; the corresponding method is referred in the following as *MLVSMATCH4*. The coarsening procedure was stopped when a maximum number of 4 levels was reached. As in the past experiments, we analyze parallel efficiency and convergence behaviour of the linear solver for 20 time steps after a preprocessing phase so that we focus on the solver behaviour in the simulation of a fully developed flow for all the mesh sizes but the largest one, where we were not able to skip the transient phase due to long simulation time. In this last case we considered a total number of 1379 time steps and analyzed solver performance in the last 20 time steps. Note that, increasing mesh size imposes to decrease time step due to stability constraints of the explicit time discretization that is preferred for LES simulations, therefore, the total simulated time depends on the mesh size. We also observe that in the Alya code a master-slave

[4] `https://www.top500.org`

approach is employed, where the master process is not involved in the parallel computations. Furthermore, in order to reduce observed operating oscillations associated to the full node runs, we used only a total of 46 cores per node.

6.1.1 Weak Scalability

As already mentioned, we analyze weak scalability of the solvers, i.e., we look at the behaviour of the solvers when we fix the mesh size per core and increase number of cores. Indeed, the 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 considered a mesh size per core equal to $1.2e5$ and used a scaling factor of 8 for going up to the largest mesh size. In the same way we scaled the number of cores for our weak scalability analysis. We can limit our analysis to observe the average number of linear iterations of the solver per each time step and to analyze execution times and scaled speedup for the solve and AMG preconditioner setup phases. We compare the results obtained by using the PSCToolit's solver against the Alya's Conjugate Gradient solver (hereby *AlyaCG*). In these experiments we also tried to use the Deflated CG implemented in Alya, but it does not work for the two larger test cases and *AlyaCG* appears better in the case of smaller size meshes. In Fig. 10, we report the average number of iterations per each time step. We can observe a general increase, ranging from 133 to 179 for increasing number of cores, when the original *AlyaCG* is employed, while a very good algorithmic scalability, with an average number of linear iterations per each time step ranging from 4 to 6, when the PSCToolkit's solver is applied. In Figs. 11-12 we can see the total solve time and the corresponding scaled speedup. We can observe that

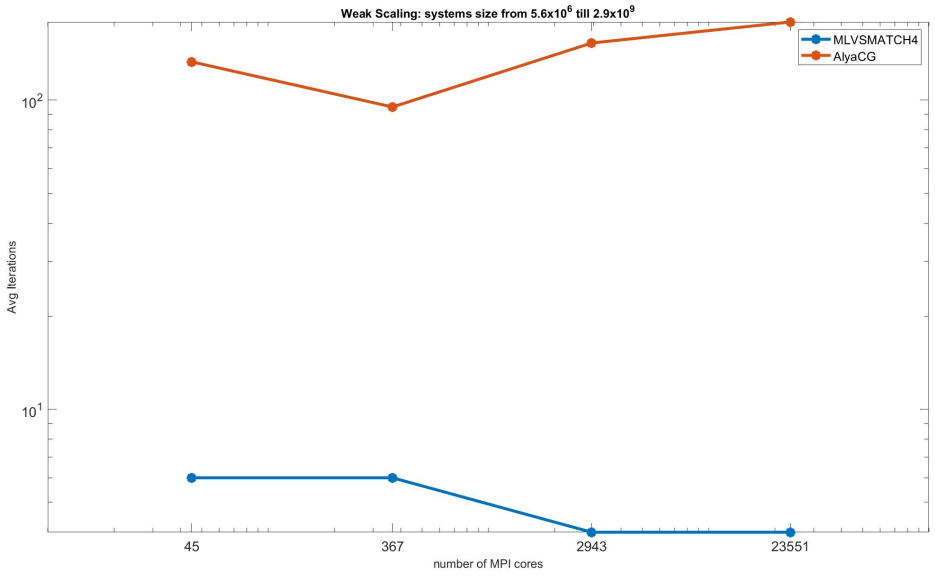


Figure 10: Weak scalability (Alya/PSCToolkit): average number of linear iterations per time step

the good algorithmic scalability of *MLVSMATCH4* leads to an almost flat execution time for solve when the first 3 mesh sizes are employed, while a decrease is observed for the simulation carried out with the largest mesh size, depending on a smaller average number of iterations per time step. On the contrary, the original *AlyaCG* generally shows a very large increase for increasing number of cores and problem size, but the second one, where a decrease of the average number of iterations per time step is observed. Then we look at the scaled speedup, defined as $scalfactor * T_{45}/T_p$, where $scalfactor = 1, 8, 64, 512$, for increasing number of cores, T_{45} is the total time for solving linear systems when 45 cores are involved in the simulation, and T_p is the total time spent in linear solvers for all the increasing number of cores. We observe that for the 2 largest mesh size, *MLVSMATCH4* has a super-linear scaled speed-up, showing that

its very good algorithmic scalability is coupled with an excellent implementation scalability of all the basic computational kernels. This scalability is very promising in facing the exascale challenge. In Figs. 13-14

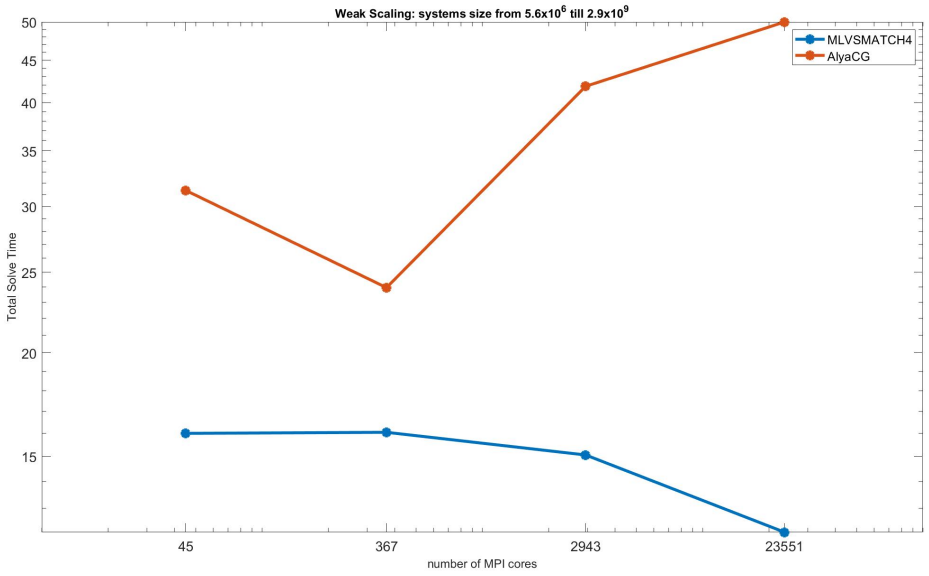


Figure 11: Weak scalability (Alya/PSCToolkit): total solve time of the linear solvers.

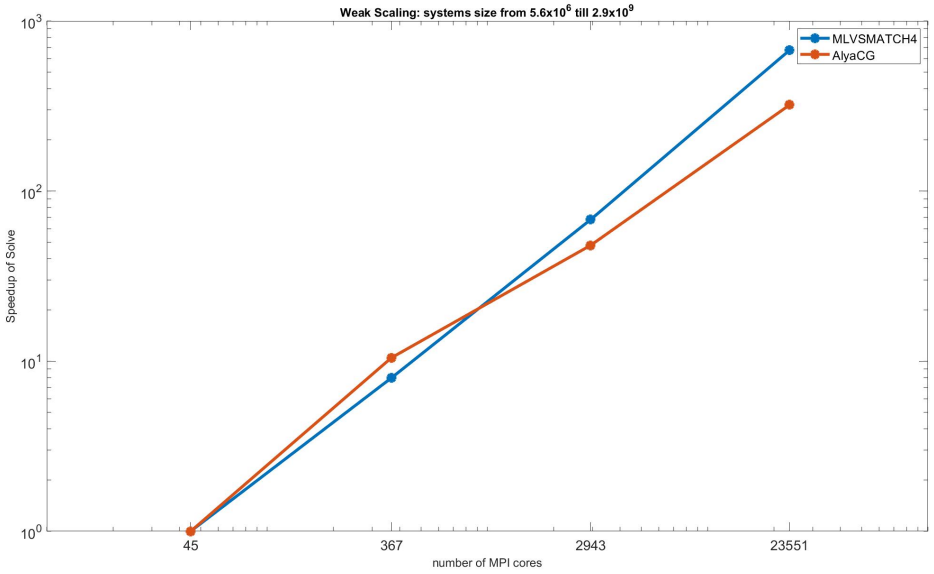


Figure 12: Weak scalability (Alya/PSCToolkit): scaled speedup of the linear solvers.

we can see the *MLVSMATCH4* setup time and the corresponding scaled speedup. We observe that the setup time of the preconditioner is about 7.2 sec. for the largest mesh size. This time is absorbed in a few time steps of the overall simulation, whose average solve time per time step is about 0.64 sec. against an average solve time per time step of about 2.5 sec. obtained with *AlyaCG*.

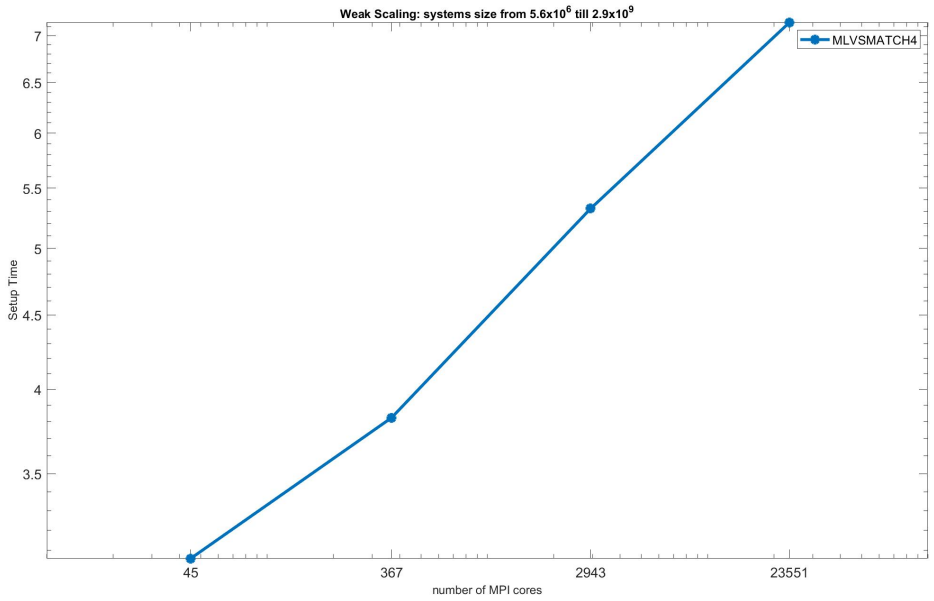


Figure 13: Weak scalability (Alya/PSCToolkit): MLVSMATCH4 preconditioner setup time.

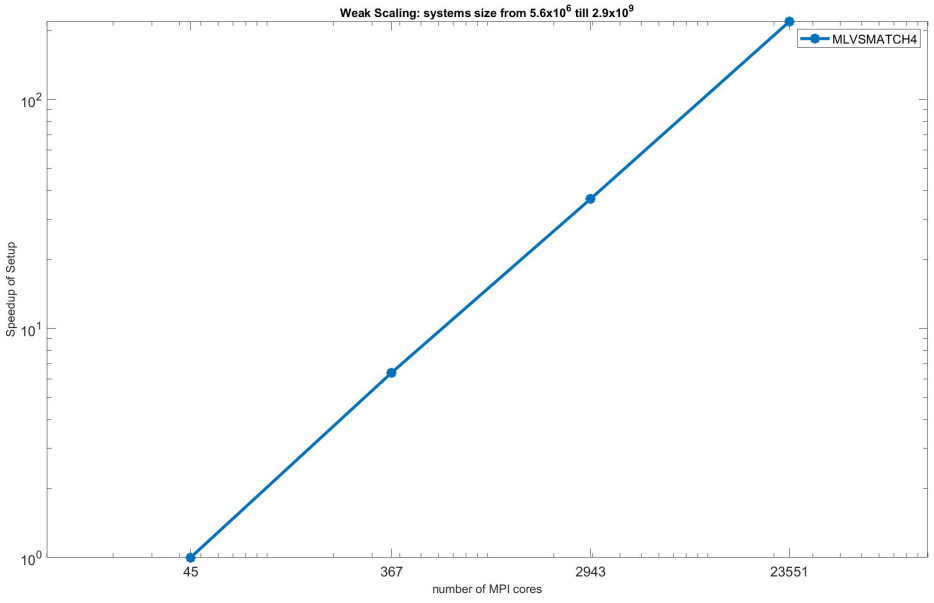


Figure 14: Weak scalability (Alya/PSCToolkit): scaled speedup of MLVSMATCH4 preconditioner setup.

7. Task 3.5: Transversal activities

7.1 PSCToolkit: PSBLAS and AMG4PSBLAS

Partners: CNR, UNITOV
Software packages: PSCToolkit, PSBLAS, AMG4PSBLAS

As already widely reported in Deliverable D3.3 [16], during the duration of EoCoE-II, we have revised and improved the PSBLAS linear algebra package, we have started a novel package AMG4PSBLAS, which is a substantial evolution of the previous MLD2P4 package, and we have defined a combined toolkit PSCToolkit containing both packages, together with some other support tools.

During the last phase of the project we have released versions 3.7.1 and 3.8 of PSBLAS and versions 1.0.1 and 1.1.0 of AMG4PSBLAS.

The main updates include:

1. the new index-owner identification (already reported in [16]);
2. a new facility to allow accumulation of remote contributions.

The latter feature is related to common practice in finite elements applications; each process will generate contributions to the matrix coefficients from its own elements, and those contributions may need to be added to data residing on a different process to complete the assembly.

In the course of the EoCoE-II project, and specifically in the WP3 transversal activities, it was decided among the linear algebra experts that this behaviour ought to be supported by all packages; accordingly, we have now full support for remote contributions in PSBLAS 3.8.0.

This has required some changes in the API interface; these changes appear in the optional arguments, and therefore the default behaviour has not changed since the previous releases. Nevertheless, applications need to be recompiled, on account of the interface changes, thus we changed the version number, and also we changed AMG4PSBLAS version number to reflect the alignment with the new capabilities.

Using the last release of the libraries we did some experiments on the GPUs of the *Marconi-100 supercomputer* (18^{th} in the November 2021 TOP500 list [5].) All the code was compiled with the `gnu/8.4.0` suite, `CUDA 11.0` for GPU kernels, and linked against the `openmpi/4.0.3` and `openblas/0.3.9` libraries. We used only inner functionalities of the `PSBLAS 3.8`, `PSBLAS-ext 1.3.0`, and `AMG4PSBLAS 1.1` libraries, with no use of optional third party libraries. We compared our preconditioners with that available in the NVIDIA AMGX library (rel. 2.2.0) installed on Marconi 100.

As benchmark test case, we consider algebraic systems arising in the solution of the Poisson equation in 3D:

$$-\nabla \cdot (K \nabla u) = \mathbf{f}, \quad \text{in } [0,1]^3,$$

with homogeneous Dirichlet boundary conditions, $K = 1$ and unitary right-hand side. The discretization of this problem is obtained by the classic 7-points finite-difference stencil for the left-hand side operator (the Laplacian operator), which results in the following system of algebraic equations:

$$\frac{6u_{i,j,k}}{h^2} - \frac{u_{i-1,j,k} + u_{i+1,j,k} + u_{i,j-1,k} + u_{i,j+1,k} + u_{i,j,k-1} + u_{i,j,k+1}}{h^2} = f_{i,j,k} \equiv 1,$$

for $i, j, k = 1, \ldots, nd$, and $h = 1/(nd+1)^2$, and with $u_{i,j,k} = u(ih, jh, kh) \equiv 0$ on $\partial[0,1]^3$. The above linear systems are characterized by a s.p.d matrix of coefficients and represent the computational kernel of many traditional scientific and engineering applications. The linear system is distributed by means of a block 3D distribution: the MPI ranks are arranged in a Cartesian grid similar to the cubic grid induced by the discretization, and the unknowns in each discretized sub-block are then assigned to the corresponding process. Note that in our case, the numbering of the unknowns follows the usual lexicographic ordering on the global mesh. We investigated the weak scalability properties of the preconditioners, therefore the tests were performed by a number of subsequent refinements of the mesh in such a way that, as the number of MPI ranks is increased, the number of unknowns per rank is fixed equal to about 6×10^6. We assigned 4 MPI ranks per compute node, so that each MPI rank is associated to 1 GPU. Taking into account previous experiments on Piz Daint (see D3.3 [16]), we limit our analysis to the preconditioners relying on the coarsening based on maximum weight matching, where a maximum size of aggregates is fixed equal to 8. The library default is applied to stop the coarsening procedure. We used both unsmoothed and

[5] See the relative list on the website www.top500.org/

smoothed prolongators, and the resulting AMG hierarchy is applied as V-cycle in both cases. In the following we named the corresponding preconditioners as *PSCTVA8U* and *PSCTVA8S*, respectively. The unsmoothed version of the preconditioner is used for comparison aims with the analogous preconditioner supported by the AMGX library. The highly parallel ℓ_1-Jacobi method is used both as smoother and as coarsest solver: 4 iterations of the method are applied as pre/post smoothing steps at the intermediate levels, while 40 iterations are applied at the coarsest level. In both cases we use the AMG methods as preconditioners for a Flexible Conjugate Gradient (FCG) algorithm. A similar AMGX preconditioner, based on matching of unknowns with maximum size of aggregates equal to 8, has been selected. Library default are used for stopping the coarsening procedure and the resulting hierarchy is applied in the same conditions of our preconditioners. This preconditioner is referred in the following as *AMGXVA8U*. For the sake of completeness, we also selected the AMGX preconditioner based on classic C/F coarsening with D-2 interpolation, as reported in the library documentation as best practice. Also in this case, the resulting hierarchy is applied as V-cycle with ℓ_1-Jacobi both as smoother and as coarsest solver, in the same conditions of our preconditioners. This last preconditioner is referred as *AMGXCD2*. We observe that the AMGX library makes available an example test for solving the 3D Poisson equation discretized as explained above, however, the 3D data distribution is applied by replicating the same cube on the different process; in this way the resulting ordering of the unknowns follows the lexicographic approach locally per each cube (process), changing the sparsity pattern of the global matrix and then impacting on the data communication during the parallel computations. The preconditioners were coupled with the Conjugate Gradient (CG) method available in AMGX. In our experiments, due to machine operating constraints, we were able to use up to 256 nodes of Marconi-100, for a total of 1024 GPUs, solving systems with up to 6.1×10^9 unknowns.

7.1.1 Weak Scalability and Comparison with AMGX

To measure the overall performance of the Krylov methods with parallel AMG preconditioners, we focus on both *algorithmic* and *implementation* scalability. Perfect algorithmic scalability is achieved when $\rho(B^{-1}A) \approx 1$ independently of the global size n of the linear system; implementation scalability on the other hand corresponds to having an optimal application cost for B^{-1} that is $\mathcal{O}(n)$ flops per iteration while achieving parallel speedup proportional to the number of processes employed. Algorithmic scalability can be analyzed by looking at the number of iterations needed by the CG/FCG to achieve a relative residual norm of 10^{-6} as the size n grows; implementation scalability is analyzed by considering both the total solve time for the procedure as well as the average time per iteration. We also analyze the timings for the preconditioner setup with increasing number of processes; all the timings reported in the figures are in seconds.

Regarding the memory footprint of the proposed multigrid hierarchies as well as the cost of the application of a V-cycle, a quantitative measure is given by the operator complexity

$$\text{opc} = \frac{\sum_{l=0}^{\text{nl}-1} \text{nnz}(A_l)}{\text{nnz}(A_0)} > 1.$$

We remark that the *AMGXCD2* preconditioner produces a memory error on 32 and 1024 GPUs, therefore we missed results for those configurations.

In Fig. 15 we show the operator complexity of the AMG preconditioners. We can observe that *PSCTVA8U* has the best value of about 1.14 for all the number of GPUs, showing a very low memory footprint and V-cycle computational complexity, due to the use of unsmoothed aggregation and then very sparse prolongators. In the same way, *AMGXVA8U* has a small operator complexity equal to about 1.3 for all the number of GPUs. Looking at the smoothed version of our preconditioner, we observe that *PSCTVA8S* has a small increase in the complexity which still remain under 2. On the contrary, *AMGXCD2* shows a very large complexity of about 4.5 producing a large memory footprint and large computational complexity in the V-cycle application. Despite the smaller operator complexity, our *PSCTVA8S* produces the smallest number of iterations up to 128 GPUs, while a small increase is observed for increasing number of GPUs, where *AMGXCD2* has the best behaviour (see Fig. 16). On the other hand, if we look at

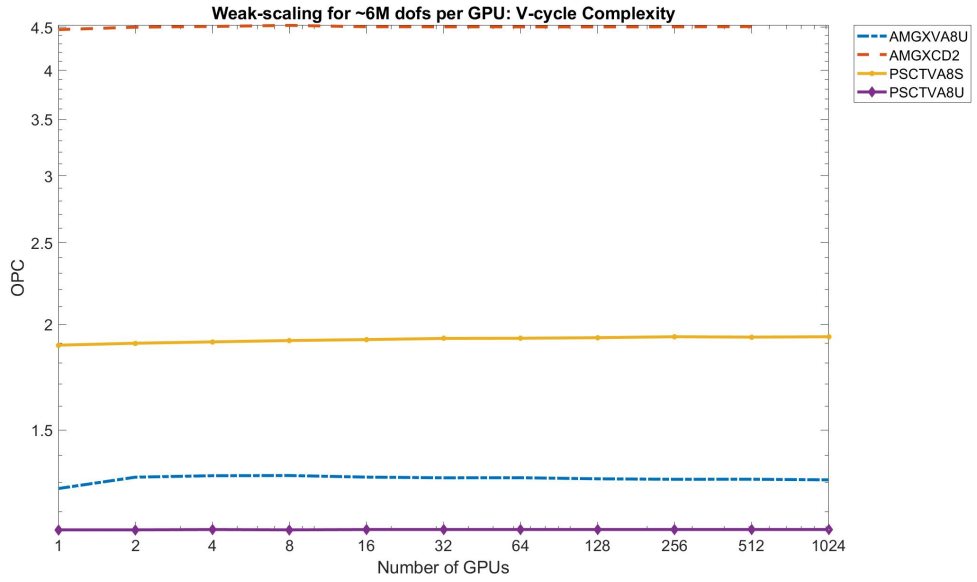


Figure 15: Weak scalability (PSCToolkit). Operator Complexity of the AMG preconditioners

the unsmoothed aggregation versions of the preconditioners, we see that our *PSCTVA8U* shows a smaller number of iterations w.r.t. the corresponding *AMGXVA8U*. This behaviour confirms that the coarsening based on maximum weight matching is very effective in producing good quality AMG hierarchies with no use of heuristics based on the concept of strenght of connections among the unknowns. Furthermore, as expected from the theory, the smoothed version of our preconditioner has largely better algorithmic scalability w.r.t. the unsmoothed version, at a very low increase of the operator complexity, also using very simple and highly parallel smoothers, such as weighted Jacobi. In Fig. 17 we show the solving time of the systems. According with the convergence behaviour and the operator complexities discussed above, we see that our *PSCTVA8S* shows the best time to solve for all number of GPUs except 256, where an increase in the number of iterations and an anomalous increase in the time per iteration (see Fig. 18) seems to produce an increase in the solve time. On the other hand, a new decrease in the solve time for increasing number of GPUs is measured. If we look at the AMG preconditioners based on the unsmoothed aggregation, we see that *PSCTVA8U* has generally better solve time w.r.t. *AMGXVA8U* up to 128 GPUs, while a small increase is observed for increasing number of GPUs. This behaviour is consistent with the time per iteration per each one of the considered AMG preconditioners (see Fig. 18). Finally, in Fig. 19, we show the time for the setup of all the preconditioners. We see that, as expected, the setup of our preconditioners requires a larger execution time w.r.t. the AMGX preconditioners. This is essentially due to the fact that our setup is run on CPUs, indeed current version of our libraries does not yet support GPU kernels for main computational building blocks of the setup phase, such as computation of maximum weight matching and sparse matrix-sparse matrix multiplication needed in the triple matrix Galerkin product. This development as well as the development of OpenMP support for the AMG preconditioners setup is work in progress and will be main objective of future projects.

7.2 MUMPS integration into HPDDM

The goal of this work is to investigate the efficient use of the MUMPS sparse direct solver within the HPDDM domain decomposition solver, two linear algebra packages of the EoCoE II project. In particular, we study the potential computational gains that MUMPS and its BLR feature enhanced by the recent advances in multithreading optimization may provide. In our experiments, HPDDM is used as a preconditioner for a GMRES solver through the PETSc interface.

HPDDM is a high-performance unified framework for domain decomposition methods. One of its

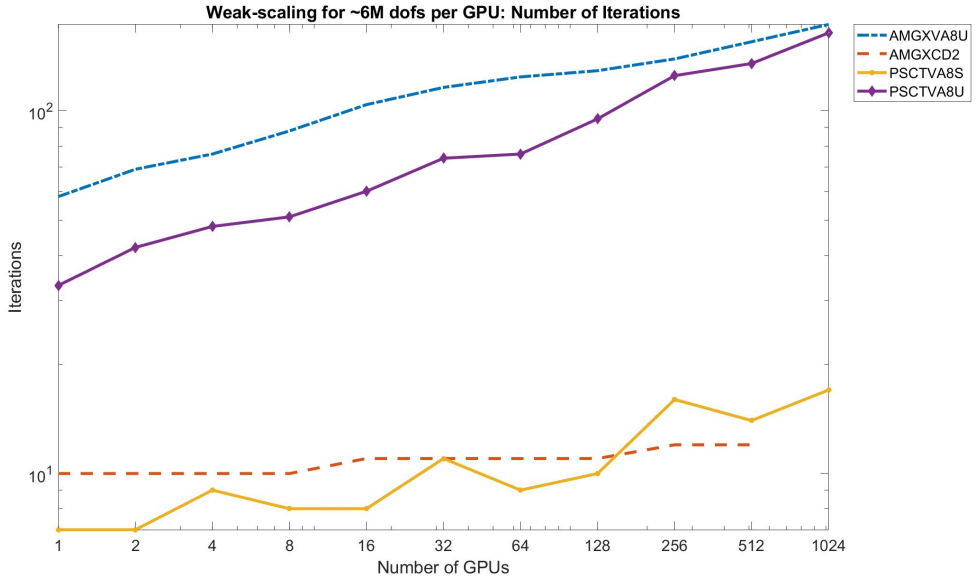


Figure 16: Weak scalability (PSCToolkit). Number of iterations for solve

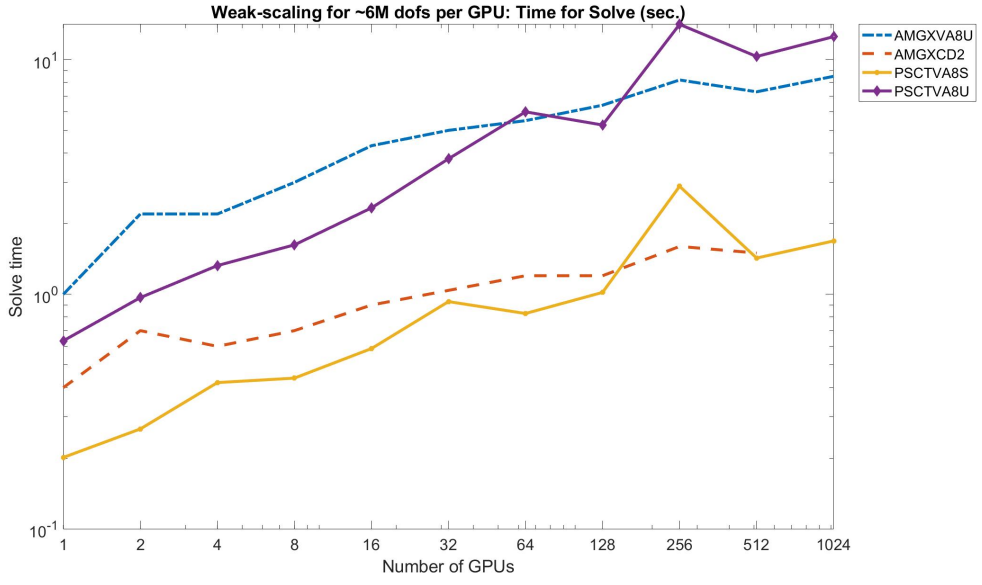


Figure 17: Weak scalability (PSCToolkit). Solve time

Figure 18: Weak scalability (PSCToolkit). Solve time per iteration

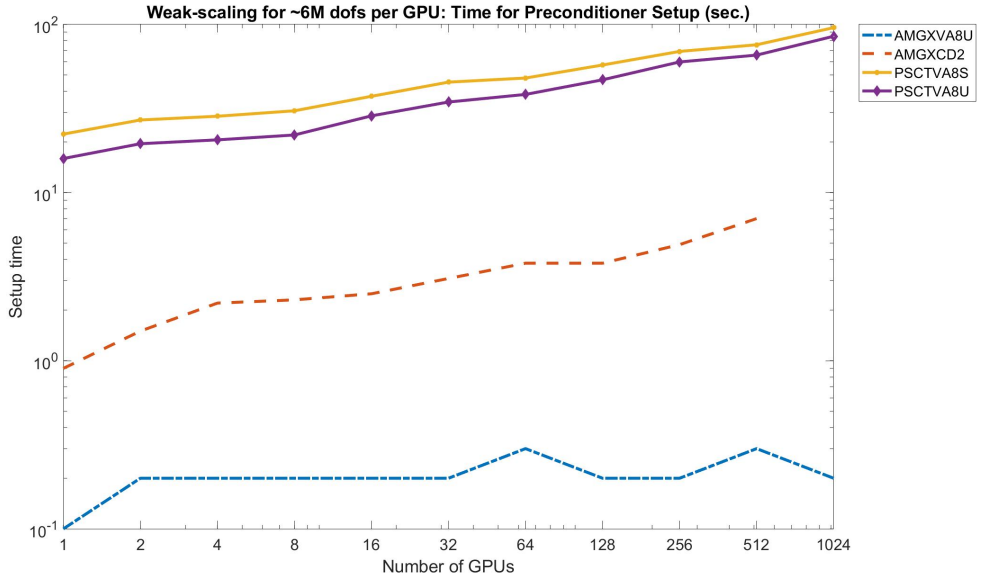


Figure 19: Weak scalability (PSCToolkit). Setup time for AMG preconditioners

main features is the construction, whenever possible, of an algebraic coarse grid by solving Generalized Eigenproblems in the Overlap (GenEO); this allows the domain decomposition method to be robust and scalable up to a very high number of domains. Within HPDDM, a sparse direct solver, namely MUMPS in our case, can be used in multiple stages:

1. local solves of the restricted problem;
2. local eigenvalue problems (GenEO) solved using SLEPc to compute the local deflation matrix;
3. one aggregated solve of the coarse problem when available.

Our objective throughout the EoCoE-II project was to evaluate the use of some novel features of the MUMPS solver and study whether and how they improve the behavior of the domain decomposition method. This was partially achieved on problems from the Alya application (see previous deliverables); here we report more general experiments on different classes of problems to asses the broader scope of our work.

In the previous deliverable we reported on the use of the MUMPS block low-rank (BLR) and tree multithreading (TreeMT) features within steps 1 and 2 above for a structural mechanics problem such as those issued from the blade design in the Alya application. Experimental results showed that, in this case, a moderate improvement of around 10% can be achieved. This is due to the fact that whenever it is possible to build a coarse correction problem, the convergence of the solver is not or only marginally affected by the number of subdomains; therefore, in this case, it is best to increase as much as possible the number of subdomains in order to reduce the cost of steps 1 and 2. As a result, the size of the subproblems solved in steps 1 and 2 is too small for the novel MUMPS features to be effective.

We have conducted new experiments on different classes of problems, namely, those where it is not possible to build a coarse correction. In this case, the number of iterations can severely increase if the number of subdomains grows (and, inversely, their size decreases). Consequently, a favorable trade off between the number of subdomains and their sizes has to be found in order to achieve the optimal number of iterations.

Figure 21 shows experimental results achieved on a Helmholtz problem with discontinuous wavenumbers on the unitary cubic domain depicted in Figure 20. Here, the same number of cores (640) was used for all the points of the curves in Figure 21; this corresponds to the number of MPI tasks (reported on the X axis, equivalent to the number of subdomains) times the number of threads used within each MPI task (which also corresponds to the number of threads used within MUMPS). The blue curve shows the behavior of the solver (time and number of iterations) prior to our work in the EoCoE-II project. As the number of MPI tasks (and, thus, subdomains) increases, the size and complexity of the subdomains becomes smaller but the number of iterations grows up to a point where it is not beneficial to increase it further; the optimum is reached for 320 MPI tasks using two threads each. The use of the BLR and TreeMT features allows to push the optimum on the left and, most importantly, lower. This is because these features reduce the complexity of the subdomains and improve the efficiency of multithreading within MUMPS; as a result, it is beneficial to choose larger subdomains which allows to achieve convergence in fewer iterations. When both features are used at the same time, the optimum is achieved with 80 MPI tasks using 8 threads each which leads to an overall improvement of almost 20%.

Since the last deliverable we have, furthermore, studied the use of TreeMT to accelerate step 3, i.e., the solution of the coarse correction problem. This only makes sense for very large problems because, otherwise, the coarse correction matrix would be too small for TreeMT to be effective. As explained above, in the case where a coarse correction can be computed, it is best to have small size subdomains with a small complexity; this corresponds to a "pure MPI" setting where no multithreading is used on the fine grid. However, in order to use TreeMT, every time the coarse problem is handled, we must dynamically switch from a "pure MPI" setting to a hybrid MPI+OpenMP one. The latest MUMPS releases provide the `mpi_to_k_omp` feature that allows to do this: MUMPS can be launched with a "pure MPI" setting and be asked to convert some MPI tasks into OpenMP threads which are used by the TreeMT feature. Nevertheless, we have chosen to implement this feature externally through the PETSc API which provides access to some advanced MPI-3 features which allows for a better placement of threads. Figure 22 shows the

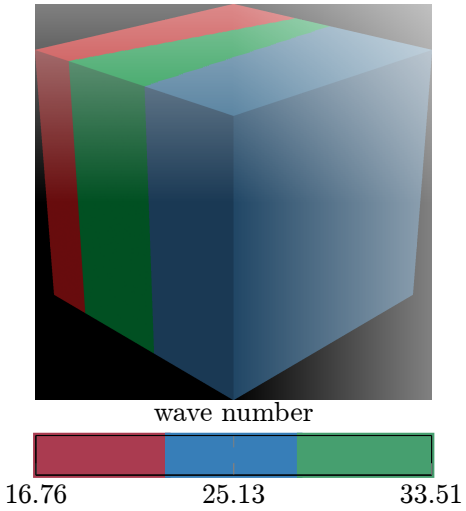


Figure 20: (HPDDM-MUMPS) Computational domain of the Helmholtz problem with discontinuous wavenumber

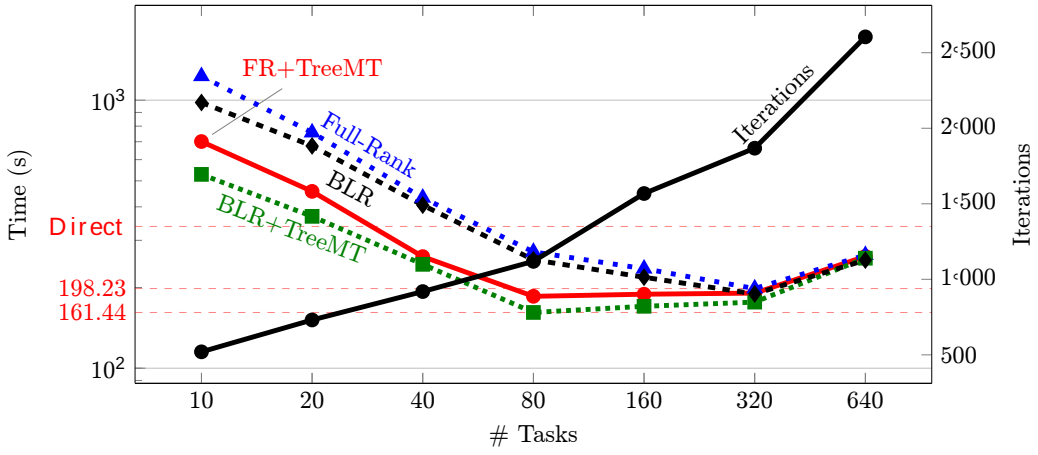


Figure 21: (HPDDM-MUMPS) Example of runs for a 3D heterogenous Helmholtz problem (unitary cube). $10\,793\,861$ dofs, $\omega = 8 \times 2\pi$, order of approximation = 2. We show here the configurations that provide the best timing for a given number of subdomains. The number close to each marker is the corresponding number of threads. As a reference, we show the best result obtained by solving the problem with a direct solver (with BLR and Tree MT activated and parallel analysis.). Additionally, we exhibit the reference timing before the introduction of the BLR and Tree MT feature (198.23) and the improved timing (161.44s).

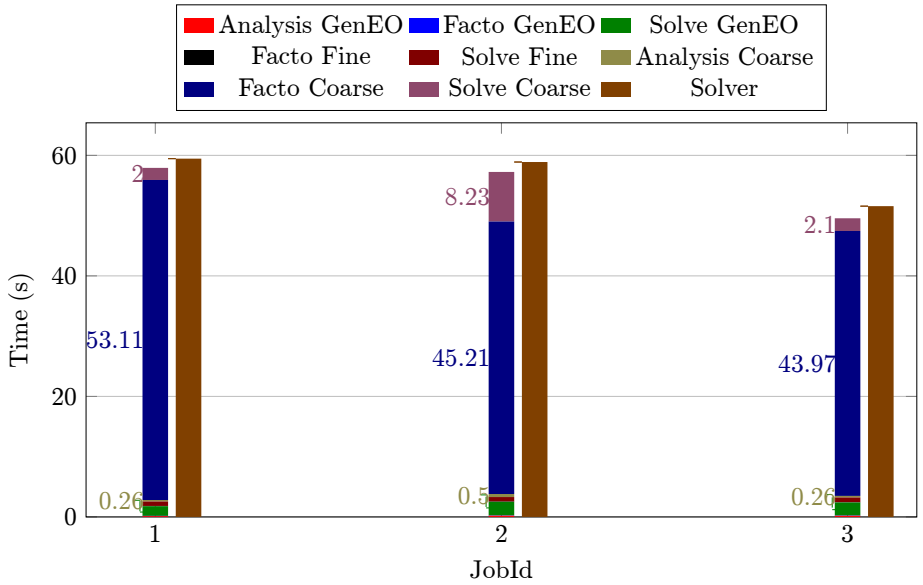


Figure 22: (HPDDM-MUMPS) Run time for the solution of a xyz DoF problem. Test 1 corresponds to the case where a pure MPI setting with 64 MPI was used on the coarse level. In Tests 2 and 3 the coarse matrix was assembled on 128 MPI tasks and 64×2 (MPI×OpenMP) where used in the direct solver; the transition is done using, respectively, the MUMPS `mpi_to_k_omp` feature and our own MPI-3 based implementation.

improvement achieved using our own implementation with respect to the MUMPS `mpi_to_k_omp` feature. In these tests the fine grid having a size of roughly 10^8 dofs is distributed over 25600 MPI tasks; the coarse problem is assembled on a subset of 128 MPI tasks and solved using a direct method in a hybrid 64×2 (MPI×OpenMP) setting. In this case, our MPI-3 based implementation achieves an improvement of roughly 10% is achieved with respect to the MUMPS `mpi_to_k_omp` feature.

References

[1] D. Bertaccini, P. D'Ambra, F. Durastante, and S. Filippone. Preconditioning Richards equations: spectral analysis and parallel solution at very large scale, 2021.

[2] R. Borrell, J.C. Cajas, D. Mira, A. Taha, S. Koric, M. Vázquez, and G. Houzeaux. Parallel mesh partitioning based on Space Filling Curves. *Computers & Fluids*, 173:264–272, 2018.

[3] 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.

[4] P. D'Ambra, F. Durastante, and S. Filippone. AMG preconditioners for linear solvers at extreme scale. *SIAM J. on Sci. Comp.*, 43(5), 2021.

[5] P. D'Ambra, F. Durastante, and S. Filippone. AMG4PSBLAS User's and Reference Guide, 2021. https://psctoolkit.github.io/amg4psblasguide/amg4psblas_1.0-guide.pdf.

[6] P. D'Ambra, S. Filippone, and P. S. Vassilevski. BootCMatch: a software package for bootstrap AMG based on graph weighted matching. *ACM Trans. Math. Software*, 44(4):Art. 39, 25, 2018.

[7] P. D'Ambra and P. S. Vassilevski. Adaptive AMG with coarsening based on compatible weighted matching. *Comput. Vis. Sci.*, 16(2):59–76, 2013.

[8] P. D'Ambra and F. Durastante. Preliminary results and performance evaluation of linear algebra solvers, 2020. Deliverable 3.2 of EoCoE II.

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

[10] S. Filippone and A. Buttari. PSBLAS User's Guide, rel. 3.7.0.1, 2021. https://psctoolkit.github.io/psblasguide/psblas-3.7.0.1.pdf.

[11] M. J. Kühn, C. Kruse, and U. Rüde. Energy-minimizing, symmetric discretizations for anisotropic meshes and energy functional extrapolation. *SIAM Journal on Scientific Computing*, 43(4):A2448–A2473, 2021.

[12] 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. *Journal on Scientific Computing*, 91(28), 2022.

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

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

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

[16] Y. Notay. Updated results and new releases of la solvers, 2019. Deliverable 3.1 of EoCoE II.

[17] Y. Notay, P. D'Ambra, M.J. Kuhn, 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.

[18] H. Owen, G. Houzeaux, F. Durastante, S. Filippone, and P. D'Ambra. AMG4PSBLAS Linear Algebra package brings Alya one step closer to exascale. In *Proceedings of the 32nd International Conference on Parallel Computational Fluid Dynamics (ParCFD)*, 2021. https://parcfd2020.sciencesconf.org/343142/document.

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

[20] J. Pierre, E. R. Jose, and S. Zampini. KSPHPDDM and PCHPDDM: Extending PETSc with advanced Krylov methods and robust multilevel overlapping Schwarz preconditioners. *Computers & Mathematics with Applications*, 84:277–295, 2021.

[21] PSCToolkit Team. PSCToolkit: Parallel Sparse Computation Toolkit, 2021. https://psctoolkit.github.io/.

[22] P. Vaněk, J. Mandel, and M. Brezina. Algebraic multigrid by smoothed aggregation for second and fourth order elliptic problems. *Computing*, 56(3):179–196, 1996. International GAMM-Workshop on Multi-level Methods (Meisdorf, 1994).