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

D2.2

Mid-term report for WP2 programming models

Project and Deliverable Information Sheet

EoCoE-II	Project Ref:	INFRAEDI-824158
	Project Title:	Energy oriented Centre of Excellence: towards exascale for energy
	Project Web Site:	http://www.eocoe2.eu
	Deliverable ID:	D2.2
	Deliverable Nature:	Report
	Dissemination Level:	PU*
	Contractual Date of Delivery:	M18 30/06/2020
	Actual Date of Delivery:	10/04/2020
	EC Project Officer:	Andréa Feltrin

* - 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 :	Mid-term report for WP2 programming models
	ID :	D2.2
	Available at:	http://www.eocoe2.eu
	Software tool:	LaTeX
Authorship	Written by:	Mathieu Lobet (CEA)
	Contributors:	Ani Anciaux-Sedrakian (IFPEN), Bibi Naz (FZJ), Brian Wylie (FZJ), Chantal Passeron (CEA), Dorian Midou (CINES), Edoardo di Napoli (FZJ), Emily Bourne (CEA), Frederic Blondel (IFPEN), Garrett Good (Fraunhofer IEE), Georg Hager (FAU), Gerhard Wellein (FAU), Helen Schottenhamml (FAU, IFPEN), Herbert Owen (BSC), Jan Eitzinger (FAU), Jaro Hokkanen (FZJ), Jose Fonseca (CEA), Julien Bigot (CEA), Judit Gimenez (BSC), Marie Cathelain (IFPEN), Michel Mehrenberger (AMU), Philipp Franke (FZJ), Sebastian Achilles (FZJ), Stefan Kollet (FZJ), Thierry Gautier (INRIA), Ulrich Ruede (FAU), Virginie Grandgirard (CEA), Yanick Sarazin (CEA)
	Reviewed by:	PEC, PBS

Document Keywords: Performance, optimization, programming model, speed-ups, scalability, MPI, OpenMP, OpenACC, Cuda, kokkos, CPU, GPU, super-computer, ARM, task

Contents

List of Figures

List of Tables

1 Executive summary

2 Acronyms

Table 1: Acronyms for the partners and institutes therein.

Acronym	Partner and institute
AMU:	Aix-Marseille University
BSC:	Barcelona Supercomputing Center
CEA:	Commissariat à l'énergie atomique et aux énergies alternatives
CERFACS:	Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique
CIEMAT:	Centro De Investigaciones Energeticas, Medioambientales Y Tecnologicas
CoE:	Center of Excellence
EDF:	Électricité de France
ENEA:	Agenzia nazionale per le nuove tecnologie, l'energia e lo sviluppo economico sostenibile
FAU:	Friedrich-Alexander University of Erlangen-Nuremberg
FSU:	Friedrich Schiller University
FZJ:	Forschungszentrum Jülich GmbH
IEA:	International Energy Agency
IBG-3:	Institute of Bio- and Geosciences Agrosphere
IEK-8:	Institute for Energy and Climate Research 8 (troposhere)
IEE:	Fraunhofer Institute for Energy Economics and Energy System Technology
IFPEN:	IFP Énergies Nouvelles
INAC:	Institut nanosciences et cryogénie
INRIA:	Institut national de recherche en informatique et en automatique
IRFM:	Institute for Magnetic Fusion Research
NEWA:	New European Wind Atlas
MdlS:	Maison de la Simulation
MF:	Meteo France
MPG:	Max-Planck-Gesellschaft
POP:	Performance Optimization and Productivity Center of Excellence
PRACE:	Partnership for Advanced Computing in Europe
R-CCS:	RIKEN Center for Computational Science
RWTH:	Rheinisch-Westfälische Technische Hochschule Aachen, Aachen University
UBAH:	University of Bath
UNITN:	University of Trento

Table 2: Acronyms of software packages

Acronym	Software, codes and libraries
PDAF:	Parallel Data Assimilation Framework
PDI:	Parallel Data Interface
EFCOSS:	Environment For Combining Optimization and Simulation Software
ESIAS:	Ensemble for Stochastic Intergration of Atmospheric Simulations
EURAD-IM:	EURopean Air pollution Dispersion-Inverse Model
GISELA-X:	GYrokinetic SEmi-LAgrangian in 5D
HYPERstreamHS:	Dual-layer MPI large scale hydrological model including Human Systems
ICON:	Icosahedral Nonhydrostatic model
MDFT:	Molecular Density Functional Theory
MELISSA:	Modular External Library for In Situ Statistical Analysis
MESO-NH:	Mesoscale non-hydrostatic model

Nemo5:	NanoElectronics MOdeling Tools 5
neXGf:	non-equilibrium eXascale Green's functions
OpenFOAM:	Open Source Field Operation and Manipulation
OpenMP:	Open Multi-Processing
ParFlow:	PARallel Flow
PPMD:	Performance Portable Molecular Dynamics
ReaxFF:	Reactive Force Field
SHEMAT:	Simulator of HEat and MAss Transport
SOWFA:	Simulator fOr Wind Farm Application
SPS:	Solar Prediction System
TELEMAC:	TELEMAC-MASCARET system
TerrSysMP:	Terrestrial Systems Modeling Platform
WaLBerla:	A Widely Applicable Lattice-Boltzmann Solver
WanT:	Wannier Transport
WPMS:	Wind Power Management System
WRF:	Weather Research and Forecast model

Table 3: Acronyms for the Scientific Terms used in the report.

Acronym	Scientific Nomenclature
ABL:	Atmospheric Boundary Layer
AD:	Automatic Diffentiation
AMR:	Adaptive Mesh Refinement
AOT:	Aerosol Optical Thickness
PBE:	Perdew-Burke-Ernzerhof functional
BLYP:	Becke-Lee-Yang-Parr functional
COT:	Cloud Optical Thickness
CLM3.5:	Community Land Model version 3.5
CPU:	Central Processing Units
CSP:	Concentrated Solar Power
DA:	Data Assimilation
DFT:	Density Functional Theory
DMC:	Dynamic Monte Carlo
FSI:	Fluid-Structure Interaction
GPU:	Graphical Processing Unit
HLST:	High Level Support Team
HPC:	High Performance Computing
ITER:	International Thermonuclear Experimental Reactor
KMC:	Kinetic Monte Carlo
LES:	Large Eddy Simulations
MD:	Molecular Dynamics
MPI:	Message Passing Interface
NEGF:	Non-Equilibrium Greens functions
NREL:	National Renewable Energy Laboratory
NWP:	Numerical Weather Prediction
OED:	Optimal Experimental Design
ODE:	Ordinary Differential Equations
PBC:	Periodic Boundary Conditions
PDAF:	Parallel Data Assimilation Framework
pdf:	probability density functions

PF-CLM:	Parflow-Community Land Model
QMC:	Quantum Monte Carlo
QM:	Quantum Mechanics
SHJ:	Silicon HeteroJunction
SOL:	Scrape-Off Layer
SpMV:	Sparse matrix-vector multiplication
TDP:	Thermal Design Power
WP:	Work Package

3 Introduction

This document is the mid-term report (M18) of Work Package 2 of the Centre of Excellence EoCoE-2. Work Package 2, called Programming Models, focuses on the computing performance of the project's applications. This Work Package plays an essential role in the improvement and accompaniment of codes towards Exascale technologies. This work is being achieved in coordination with the Exascale Co-design Group and other Work Packages.

The list of flagship codes in the project is shown in Table 4. We also list the so-called satellite codes in Table 5. Not all codes will be optimized with the same ambitions and therefore not all of them are concerned by the WP2. Some codes will be accelerated with the improved linear solver (see WP3). We show the state before the start of the project and the desired state at the end of the project.

Application name	State before the project	Targeted architectures
ALYA	Only optimized for CPU super-computers thanks to EoCoE-I (FORTRAN, MPI)	CPU and GPU with high-level load balancing to use both at the same time (OpenACC, CUDA)
EURAD-IM	parallelized on CPU only (FORTRAN, MPI)	CPU optimization (Hybrid parallelism) and GPU porting (OpenACC)
ESIAS	parallelized on CPU only (Python, KSH, FORTRAN, MPI)	No optimization work in the WP2
LIBNEGF	CPU only, not optimized (FORTRAN, MPI)	CPU optimization and GPU porting if possible
KMC/ DMC	CPU-only (Python, C/C++, MPI)	No optimization work in the WP2
QMCPACK	CPU and GPU (Python, C++, MPI, OpenMP, CUDA)	No optimization work in the WP2
PARFLOW	Partly optimized CPU-only implementation (C, MPI, OpenMP)	Optimized CPU implementation with AMR (py4est), GPU porting (CUDA)
SHEMAT-SUITE	single-node CPU only (FORTRAN + OpenMP) except for ensemble runs (handled via MPI)	MPI under development
GYSELAX	Well optimized on CPU thanks to EoCoE-I (FORTRAN, MPI, OpenMP)	Further optimized to work efficiently on ARM-based processors (collaboration with the RIKEN)

Table 4: Flagship applications in the project. Orange cells are for CPU-only optimized applications. Green cells are for GPU-optimized applications. When a cell remains white, it means that the code in this state is not ready to exploit the power of future machines.

The adaptation of the codes to future PRACE and pre-exascale machines is a complex issue because it depends a lot on the adopted technologies. There are several technologies envisaged to build exascale machines capable of respecting an electric power consumption envelope of about ten MW. The United States has already adopted the hybrid CPU and GPU combination to break the 100 petaflops barrier. Major computing centers today choose this combination of technologies for their flagship powerful super-computers. The TOP500 makes this choice clear [42]. The GPU concentrates exceptional raw computing capacity at a more affordable power cost (TDP) than traditional x86 CPUs. Nevertheless, this method is not the only one being considered, even if it is the most accepted and mature today. Japan, for example, has chosen to update the K-computer by developing its own processors based on the ARM architecture. This architecture, used massively in the mobile environment, is gradually finding its way into the world of servers and HPC. This solution is being explored as well in Europe through the Mont-Blanc project. The ARM technology allows, among other things, to obtain a greater technological independence than GPUs (dominated by US companies). Other solutions are being considered such as combining CPU and accelerators based on ARM technology. This has already been done in particular on some Chinese computers.

Application name	State before the project	Targeted architectures
waLBerla	Only optimized for CPU super-computers (C++, MPI)	CPU only, no further developments in term of optimization
TOKAM3X/SOLEDG	CPU-only (FORTRAN, MPI, OpenMP)	No optimization worl in the WP2, final performance will depend on WP3.
METALWALLS	CPU and GPU (FORTRAN, C++, Python, MPI, OpenMP, OpenACC), the code was extensively optimized during EoCoE-I	No optimization development in EoCoE-II
MDFT	CPU-only (FORTRAN, oepnMP, MPI), the code was apartly optimized during EoCoE-I	No optimization development in EoCoE-II
GENE [19]	CPU and GPU (FORTRAN, C, MPI, OpenMP)	No optimization development in EoCoE-II
WIND POWER MANAGEMENT SYSTEM	scientific models in Matlab and/or Java	No optimization development in EoCoE-II
SOLAR PREDICTION SYSTEM	scientific models in Matlab and/or Java	No optimization development in EoCoE-II
ICON	FORTRAN	No optimization development in EoCoE-II

Table 5: Satellite applications in the project. Orange cells are for CPU-only optimized applications. Green cells are for GPU-optimized applications. When a cell remains white, it means that the code in this state is not ready to exploit the power of future machines.

It is an alternative to the CPU + GPU method but requires development efforts (i.e. investment) to reach the same level of maturity. Moreover it is a niche market unlike the GPU market boosted by the world of video games and artificial intelligence. That said, some GPUs in the mobile world exploit the ARM technology [30] and the same dynamics as CPUs may emerge in a few years. There are even more exotic technologies for the HPC world, such as FPGAs commonly used in the embedded world. Projects coupling CPU + FPGA (accelerator) are mostly led by private companies such as INTEL or MAXELER. Although these projects target specific applications, it is not excluded that we may benefit from advances in the world of scientific computing in the future. There are initiatives such as the Exa2pro project [15] to explore this possibility, but the technology is not yet fully mature. FPGAs, although more difficult to program than CPUs, offer more raw power at a lower cost. Moving towards hybrid CPU and GPU parallelization is the safest choice today to be able to use the full power of tomorrow's leading super-computers. Several teams have chosen to port their code to the GPU as shown in table 4 even if the software choices are not necessarily the same.

Before starting to optimize a code today, it is important to identify and anticipate the needs in terms of computing power in order to choose the software solutions capable of meeting this need in the long term. There is often a trade-off between performance, portability, maturity, readability, legacy and available development time. One of the roles of WP2 is to guide developers towards the best choices to meet their needs. The used software methods are partly given in table 4. Most of the tools used in EoCoE-2 are mature and proven.

The 3 programming languages used here are FORTRAN, C and C++. Although the choice of a better language is still debated today, there is consensus that C/C++ is a better choice because most HPC libraries today primarily support these languages. FORTRAN, not yet widely used for numerical simulation, is less and less supported and increasingly abandoned. Nevertheless, many codes are still written in FORTRAN and the rewriting work is a significant challenge that requires skilled and up-to-date human resources on languages, time and methodology.

MPI is the standard of choice widely used in distributed computing as it is on all modern HPC machines. As processors condense more and more compute cores, it is more and more common and interesting to adopt hybrid thread parallelization at the node level. Here, only OpenMP is used for this. OpenMP uses the notion of threads to exploit the parallelism of recent processors and uses the notion of directives to simplify the development.

Programming on GPUs can be done using proprietary low-level programming language and its associated libraries. CUDA is the most widely used, but only for NVIDIA boards. This solution makes the most of the power of NVIDIA cards but is not portable. In order to be more portable, openACC allows GPUs to be addressed by directives like OpenMP does on the CPU. This solution has the advantage of not being tied to a specific type of GPU card in order to remain as portable as possible. In this project, we are using both solutions.

During a GPU porting, we generally want to minimize code duplication, to have a good memory management between the host processor and the device, to have a portable implementation to avoid rewriting algorithms at each technological leap, to be able to minimize the distinction between a code intended to run on CPU and a code intended to run on GPU.

Recently, new programming models have become fashionable because they make it possible to bring all these requirements together. This is the case of Kokkos [25] and RAJA [35], both developed in the United States. In particular, they make it possible to abstract the use of memory and thus allow the development of generic CPU/GPU algorithms. The use of Kokkos will be explored in this project.

In collaboration with Work Package 4, WP2 is involved in the development and use of the PDI API. The Parallel Data Interface (PDI) is not a library itself but an interface that enables users to decouple all these I/O processes from codes through a single API ([38, 9]). As shown in Fig. 1, the API supports read- and write- operations using various I/O libraries within the same execution and allows switching and configuring the I/O strategies without modifying the source (no re-compiling). However, it does not offer any I/O functionality on its own. It delegates the request to a dedicated library plugin where the I/O strategy is interfaced. In other words, PDI offers a declarative API for simulation codes to expose information required by the implementation of I/O processes. The latter are encapsulated inside plugins that access the exposed information.

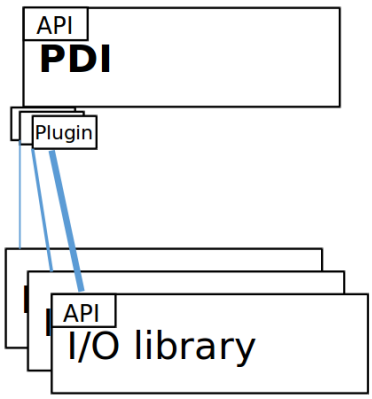


Figure 1: Conceptual scheme of the Parallel Data Interface (PDI).

The next sub-section describes the document structure.

3.1 How to read this document

Each section of this document represents one of the major task of WP2 as described in the proposal. The first task called performance evaluation and modeling is transverse to all the Scientific Challenges and concerns the missions of this Work Package. This first task is associated with the first section 4. The following tasks have been constructed to contain the work to be carried out in each Scientific Challenges respectively. As a result, the following sections are associated with each Scientific Challenges:

- section 5: Task 2.2 - Wind code optimization
- section 6: Task 2.3 - Meteorology code optimization
- section 7: Task 2.4 - Materials code optimization
- section 8: Task 2.5 - Hydrology code optimization
- section 9: Task 2.6 - Fusion code optimization

In each major task of this WP, we first remind the associated codes before describing the work carried out. This includes the members of each team and updates. We have in the proposal and then in the first deliverable divided major tasks into subtasks. Since the first deliverable, each code has had an action plan (simplified Gantt) that we update here according to the work progress, the difficulties and the encountered delays. The action plans are based on the explanatory model shown in the figure 2.

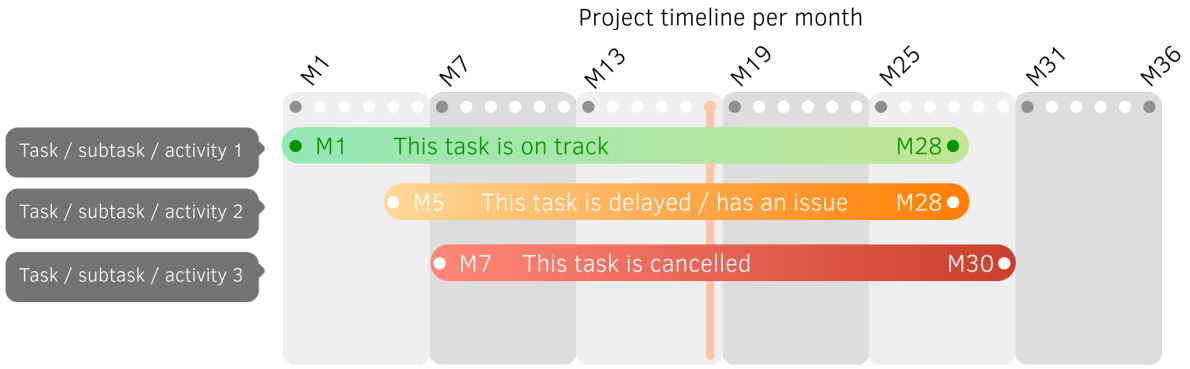


Figure 2: How to read our simplified Gantt chart.

A timeline provides approximate information on the start and end of each subtask. A green task does not present any difficulty. A task in orange has problems; it may possibly be delayed or extended. In red, the task is cancelled.

For each major task, a table of risks is shown at the end of the section.

3.2 Impact of COVID-19

Our project has been impacted by the health crisis due to COVID-19. The encountered difficulties and the impact on the project are described in the risk management sub-sections for this Work Package.

4 Task 2.1 - Performance evaluation and modeling

The goal of this task is to provide the required tools and resources to the project applications to ensure continuous and successful code optimization and performance improvement. This task is organized around several objectives:

- Performance evaluation process of the codes
- Performance bottleneck identification
- Optimization strategies on kernels and on full applications
- Workshops and hackathons to teach tools and guide optimizations with experts
- Knowledge benefit outside the EoCoE community

To achieve these objectives, task 2.1 contains several actions to perform:

- Support in performance evaluation, code optimization and code engineering through project experts
- Organization of workshop dedicated to performance evaluation and code optimization
- Communication around external training on code optimization (like PRACE trainings)
- Management of the computing resources

An active support is enabled thanks to the HPC experts connected to the project. Section 4.1 brings more details on support provided by our experts. The events organized by this Work Package are presented in section 4.2. The management of the PRACE computing resources is described in section 4.3.

4.1 Optimization support

Our experts are presented in Table 6.

People	Position	Role	Period
Georg Hager	FAU	Node-level optimization, LIKWID tools	M1-M36
Gerhard Wellein	FAU	Coordinator at FAU	M1-M36
Jan Eitzinger	FAU	Node-level optimization, Likwid tools	M1-M36
Thomas Gruber	FAU	LIKWID tools	M1-M36
Dominik Ernst	FAU	Node-level optimization, GPU optimization	M1-M36
Judit Gimenez	BSC	HPC expert, member of the POP COE, BSC tools	external to the project
Brian Wylie	JSC	HPC expert, member of the POP COE, JSC tools	external to the project
Thierry Gautier	CR INRIA	Expert in task-based programming model	M1-M36

Table 6: Performance and optimization experts for support in EoCoE-2.

Gerhard Wellein coordinates the FAU's activity within EoCoE. Georg Hager and Jan Eitzinger are part of the HPC expert panel available within the project to help application teams optimize their code. They are responsible for organizing tutorials and hackathons with a strong node-level component. Dominik Ernst is a GPU expert with in-depth experience on code optimization.

Thomas Gruber is the main developer of the LIKWID tool suite, which is taught and used during the workshops and for most performance-centric work on application code.

Team members are available as points of contact for code optimization. EoCoE developers can work with them in close collaboration.

Judit Gimenez is an HPC expert at BSC and a member of the POP COE team. She participates in the workshop organization on performance analysis and optimization.

Brian Wylie is an HPC expert and a member of the HPC application support at FZJ. He is actively involved in the POP COE and is a representing FZJ tools at the workshops.

Thierry Gautier is computer scientist and an expert in HPC with a strong expertise in asynchronism and task-based methods. He has joined the project specifically to provide some support in task parallelism especially for the development of GyselaX. Thanks to EoCoE resources, Thierry Gautier has improved the tools he is working on for the community (libKOMP [17], Tikki). In 2019, he leads work that results in pushing two patches of the LLVM OpenMP in the master branch to improve performance in the management of task in the runtime. It also includes the development of a performance monitoring module using the tracing method for OMPT (a first-party API for third-party performance and monitoring tools in OpenMP-5.0) called TiKKi. The module should be available end 2020.

Thomas Gruber and Dominik Ernst have been working with Alya developers to analyze the node-level performance properties of the Alya code. Using LIKWID markers and running the code under the control of the `likwid-mpirun` tool, it became possible to get an insight into load balancing and resource utilization issues. Due to the particular code structure (master-worker style), this activity posed an interesting and unusual case study that can be included in future optimization tutorials. As an important result of the analysis, memory bandwidth could be shown to be a marginal issue for the code's performance. This means that in-core performance and load balancing are the most viable optimization targets.

4.2 WP2 events

The WP2 organizes workshops dedicated to the performance analysis and the code optimizations. Our calendar of events is given in Fig. 3. At the beginning of the project (as described in the D2.1), we had in mind to organize two types of workshop:

- A performance evaluation workshop to teach the tools and helps the team to determine their application bottlenecks. This first session was to ensure that all code developers, and particularly developers involved in code refactoring and optimization, are on the same level of knowledge.
- At least two hackathon workshops dedicated to work in the codes, developers and HPC experts together. Hackathons should therefore gather HPC experts and application developers to work on specific optimization issues during approximately 3-day. They enable to overcome strong performance bottlenecks or complex optimization challenges for application developers. They also help to track the optimization progress and update performance-aware code development strategies.

The first performance evaluation workshop was held in Erlangen (Germany) from October 7 to the 10 2019 [28] (M10). It was co-organized and hosted by the FAU University. We have as well partnered with the POP COE [34] to propose the tools developed at BSC. The workshop was planned as follow:

- Two days were dedicated to the presentations of the CPU core architecture and the performance evaluation tools developed at FAU (LIKWID) alternating lectures and hands-on.

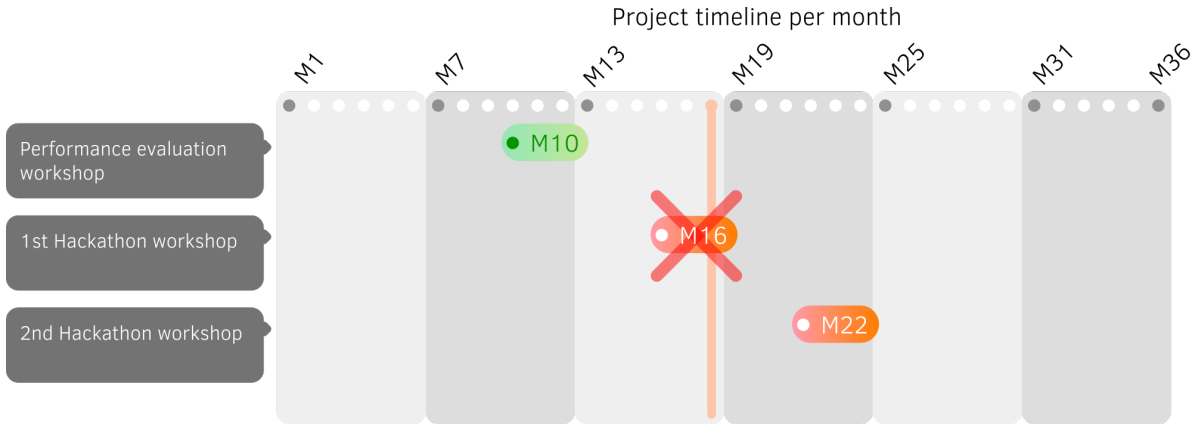


Figure 3: Events organized by the WP2.

- The third day was dedicated to the POP COE tools (Paraver, Scalasca) alternating as well lectures and hands-on.

Figure 4: WP2 performance evaluation workshop held in Erlangen (Germany) from October 7th to the 10th 2019.

The workshop proved to be a great success. A picture of the training room is shown in Fig. 4. We have welcomed 14 attendees from 8 different institutions. They were representing 12 different applications with 6 being EoCoE applications or libraries. The workshop general presentations (not including the hands-on) have been recorded and put online [20].

The second workshop should have been the first hackathon. Many application developers within EoCoE were waiting for it to start the close collaboration with HPC experts. It should have been from March 31 to April 3 2020. Because of COVID-19, it was cancelled a few weeks before it was to take place. Today we cannot say when this workshop will be postponed and if we will maintain two such events within the project. As this workshop was eagerly awaited by multiple teams to start the search for blocking points and code optimization, we invited the EoCoE developers to start remote point-to-point studies with the experts. As the crisis is not over yet, we are also relaying the PRACE online and local training courses to meet the needs. Maintaining our hackathon workshop as an online event appeared to be an option but was not retained due to lack of time.

We had in mind to organize the second hackathon at the M22. At the moment we cannot guarantee this date. We are waiting to see how the current crisis and the end of containment will evolve.

4.3 PRACE computational resources

PRACE offers some computational times to the Center Of Excellence at every Calls. The WP2 manages the computing resources allocated for the whole project. Every 6 months in March and September, PRACE updates the amount of hours for Center of Excellence. The new batch is divided between all Centers of Excellence depending on their needs. To evaluate our needs, the PRACE proposition is first scattered toward all our members. Then all members indicate what they need and a common proposition is therefore sent to PRACE for examination.

Table 7 summarizes the amount of hours granted to EoCoE per super-computers and what we have consumed.

Super-computer	Granted core hours	Consumed core hours	Usage Percentage
Marenostrum 4	891667	1056890	118.53%
SuperMUC	150000	0	0 %
SuperMUC NG	302500	0	0 %
Juwels	167500	10000	5.97 %
Joliot-Curie KNL	160000	0	0 %
Joliot-Curie SKL	571667	36156.87	6.32%
Joliot-Curie ROME (AMD)	1060000	0	0 %
Piz Daint	2366644	2481.1	0.1 %
Marconi Broadwell	155000	78027	50.34 %
Marconi KNL	770000	528109	68.59 %
Marconi 100	180000	0	0 %
Hawk	1100000	0	0 %

Table 7: PRACE resources for EoCoE-2.

So far, we have been granted a total amount of around 8 million core hours (sum over all super-computers). We have consumed around 1.7 million core hours (close to 22 %). If we go into detail, not all machines are used at the same level. Some of them are rapidly used at the maximum of their capacity like Marenostrum. On the contrary, some machines are just requested for testing new implementation and optimization. It happens that the amount has been overestimated and not totally used. In any case, this computational time is extremely useful for the project.

Note that this table represents a small part of the whole available resources since it does not take into account local resources at institution scale and PRACE or national access to super-computers external to the project.

4.4 Risks, warning points and mitigation

The risks and warning points for task 2.1 are detailed in Tab. 8.

Risks / Warning points	Who	Impact	Mitigation
Cancellation of the first hackathon workshop and organization of the next one due to COVID-19	Developers	Developers needing detailed analysis of performance issues and help in overcoming these obstacles find themselves working alone. This can lead to misinterpretations or less successful solutions.	Teams can work remotely on a point-to-point basis for the time being, make use of local resources and participate in online workshops.
Some members alerted on how PRACE resources are allocated: few hours scattered over many supercomputers. This limits the possibility of large-scale testing, particularly in the context of pre-exascale preparation.	Developers	Simulations are limited in size and duration. This is still useful for medium-scale tests.	Using classical PRACE Calls

Table 8: Risk management in Task 2.1.

5 Task 2.2 - Wind code optimization

5.1 Task overview

Task leader : BSC

Participants: BSC, FAU, IFPEN

The wind objective is to bring the Large Eddy Simulation (LES) formulation for wind farm simulation to the Exascale. In term of numerical simulation, a typical production runs should reach a resolution of 10^{10}-10^{11} grid points on unstructured grids with approximatively 1 day time-to-solution on a Exascale machine. In term of scientific purpose, the goal is to perform multiscale LES modelling of fluid-structure interactions in turbine blades and model entire wind farms with complex terrains (see WP1). For this aim, a full rotor model where the actual geometry of the wind turbine is modelled exactly should be implemented.

In the WP2, the wind challenge involves the flagship code Alya and 2 satellite codes waLBerla and Meso-NH. A brief summary of application properties and purposes is respectively given in the following sections section 5.1.1, 5.1.2 and 5.1.3.

The work to be done in these codes has been divided into 3 subtasks:

- Task 2.2.1 - Alya code refactoring and optimization for Exascale
- Task 2.2.2 - waLBerla actuator line code extension
- Task 2.2.3 - Performance comparison between waLBerla, Alya and Meso-NH (replacing SOWFA)

The detailed content of these tasks and the progress achieved so far is described in sections 5.2, 5.3, 5.4.

5.1.1 Flagship code ALYA

ALYA [1] is a high-performance computational mechanics code that solves complex coupled multi-physics problems, mostly coming from the engineering realm. The code is developed at BSC (ALYA website).

The main goal for ALYA is to bring the code to Exascale to tackle the simulation of full wind farm over complex terrain with up to 100 wind turbines. Within WP2, ALYA 's developers with HPC experts are refactoring and optimizing the code to be able to address heterogeneous computing nodes with maximal efficiency. They will implement a full rotor model where the actual geometry of the wind turbine is modelled.

Table 9 shows the team members of ALYA involved in EoCoE. Herbert Owen is a senior researcher at BSC. He has been leading the Wind Scientific Challenge since EoCoE-I. He coordinates wind energy developments of ALYA and represents this code in EoCoE. Guillaume Houzeaux is the manager of the Physical and Numerical Modelling group at BSC and one of the main developers of ALYA.

People	Position	Role	Period
Herbert Owen, PhD	Senior researcher at BSC	Responsible for the ALYA team within EoCoE and developer of the code	M1-M36
Guillaume Houzeaux, PhD	Physical and numerical group manager at BSC	Main Code developer	M1-M36

Table 9: Team Members for ALYA within EoCoE.

The work in ALYA is described in task 2.2.1 (see section 5.2) and task 2.2.3 (see section 5.4).

5.1.2 Satellite code WALBERLA

WALBERLA is a fluid simulation code that uses the lattice Boltzmann method (WALBERLA website). WALBERLA is developed at the Friedrich-Alexander University of Erlangen-Nuremberg (FAU). In WP2, WALBERLA developers will implement an actuator line model. The final goal is to be able to simulate wind turbine with the lattice Boltzmann method and to compare the results with the flagship code ALYA and the code MESO-NH (replacing SOWFA).

Table 10 shows the team members of WALBERLA involved in EoCoE. Ulrich Ruede is the code Coordinator at FAU. Helen Schottenhamml has been hired at M9 at FAU as a research assistant to work on WALBERLA for a duration of 8 months (until end of March 2020). She is in charge of the work in WALBERLA described in task 2.2.2 She was supposed to then move to IFPEN in France on April 1st, 2020 (M16) until M27, but the COVID-19 crisis prevented her to change her location. She worked from Erlangen for IFPEN until she could move to France in June 2020.

WALBERLA is concerned by task 2.2.2 (see section 5.3) and task 2.2.3 (see section 5.4).

5.1.3 Satellite code MESO-NH

MESO-NH is the non-hydrostatic mesoscale atmospheric model of the French research community (Meso-NH Website) dealing with scales ranging from synoptic (1000 km scale) to large eddy scales (meter scale). It has been jointly developed by the Laboratoire d'Aérologie (UMR 5560 UPS/CNRS) and by CNRM (UMR 3589 CNRS/Météo-France).

People	Position	Role	Period
Ulrich Ruede, PhD	FAU	Responsible for the WALBERLA code	M1-M36
Ani Anciaux-Sedrakian, PhD	IFPEN	Code optimization and development	M1-M33
Frédéric Blondel, PhD	IFPEN	Code optimization and development	M1-M33
Helen Schottenhamml, M.Sc.	PhD student at FAU and Engineer at IFPEN (M16-M27)	Code optimization and development	M1-M33

Table 10: Team Members for WALBERLA within EoCoE.

MESO-NH substitutes SOWFA that was the code originally given in the proposal for task 2.2.3 at IFPEN. They are several reasons that have motivated this choice. First, although MESO-NH is a LES code like SOWFA, it is more advanced from a meteorological point of view. MESO-NH can model more thermo-dynamical phenomena such as radiation, deep and shallow convection. It embarks advanced physical parameterizations for cloud and precipitation representation. It can be coupled with different modules for chemistry (aerosol...) or complex surface (vegetation, cities, ocean...) for instance. Then, MESO-NH is more advanced in term of HPC (Good scalability, vectorization) and is actively supported. The last argument to use MESO-NH is the size of the benchmarks. Simulated domains for EoCoE have a size of 40 km by 40 km much higher than the size usually considered in MESO-NH simulation at IFPEN.

Table 11 shows the team members of MESO-NH involved in EoCoE. Marie Cathelain is engineer at IFPEN in charge of coordinating the work in MESO-NH for the task 2.2.3.

People	Position	Role	Period
Marie Cathelain, PhD	Engineer at IFPEN	Responsible for the MESO-NH code	M1-M36

Table 11: Team Members for MESO-NH within EoCoE.

MESO-NH appears in task 2.2.3 (see section 5.4).

5.2 Work progress on task 2.2.1

Task 2.2.1 corresponds to the refactoring and the optimization of the code ALYA. It aims at optimizing ALYA for Exascale to run complex terrain and full rotor with the required accuracy. It contains the following subtasks:

- PDI or Sensei integration for in-situ visualization in WP4 and WP5
- ALYA general code optimization: Code cleaning, node-level optimization and vectorization, Dynamic load balancing (DLB package), MPI overlapping between communication and computation, hybrid GPU implementation, coexecution on heterogeneous cluster (CPU + accelerators), Fast and scalable geometric mesh partitioning based on Space Filling Curve, Dynamic coupling between rotating meshes that following turbine blades and fixed mesh for the rest
- Scaling to Exascale: Running real cases on exascale or pre-exascale machines: complex terrain and full rotor (rely on the speedups reachable with optimizations).

Although it was not originally mentioned in the proposal, we include in this task the work performed with the code MESO-NH reserved for code comparison in subtask 2.2.3.

Fig. 5 describes the current work plan for task 2.2.1.

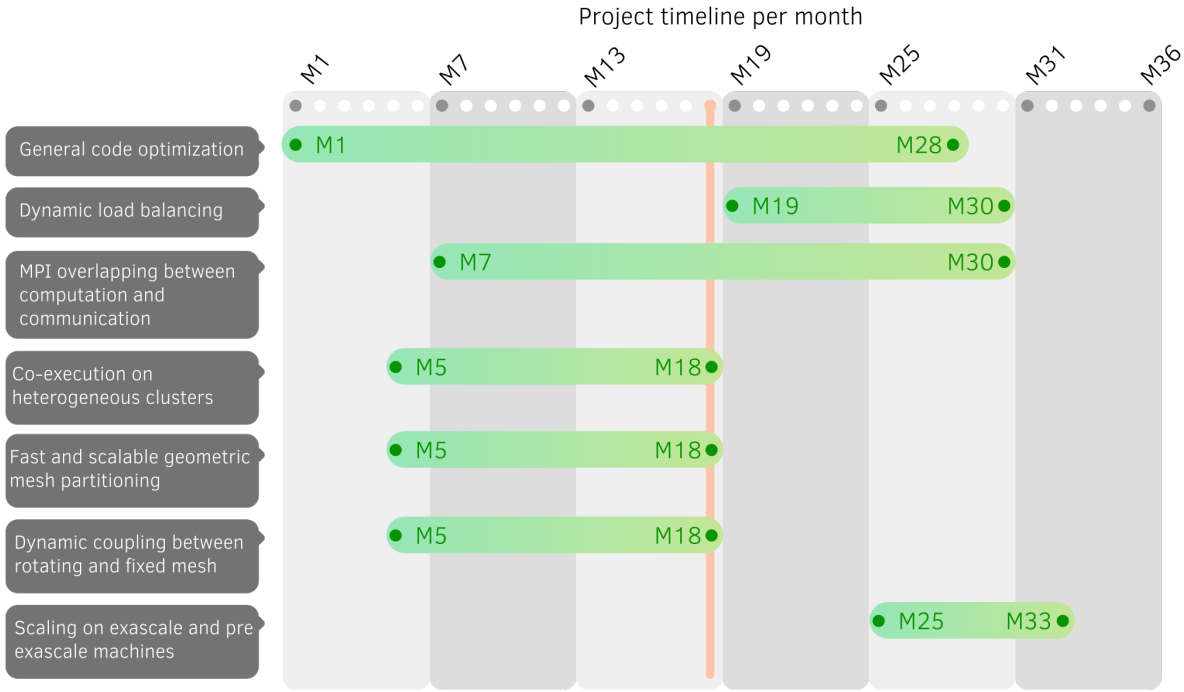


Figure 5: Breakdown (simplified Gantt chart) of the task 2.2.1 for ALYA.

5.2.1 Work progress in ALYA

General code optimization Guillaume Houzeaux and Herbert Owen attended to the EoCoE Performance Evaluation Workshop at Erlangen in October 2019 to learn about the code optimization tool to be used within EoCoE-II. It has allowed the BSC team to interface ALYA with the MPI version of Likwid. Some small difficulties appeared, since the MPI version of Likwid is typically less used than the version for shared memory. Once those problems were sorted out, BSC was able to perform a run with Likwid on a wind farm case. A finite element code has two main kernels, the construction of a matrix or right-hand side (RHS) vector and the solution of a linear system. The optimization will concentrate mainly on the construction of the momentum equation, which is treated explicitly for Large Eddy Simulation wind problems. The velocity and the pressure are uncoupled using a fractional step scheme. The pressure is solved implicitly since we are dealing with the incompressible Navier Stokes equations. The matrix for the pressure remains fixed during the whole simulation, which involves thousands of time steps. Therefore, the optimality of the matrix assembly phase is not essential. Finally, we will rely on the linear algebra packages provided within EoCoE for the solution of the linear system. BSC has identified the part of the code corresponding to the RHS vector construction, and the FAU team is currently analyzing its performance. The advance of this task has been somehow disturbed by COVID-19 restrictions. BSC had been organizing Hackathon in Barcelona for the first week of April. Its objective was to get together the HPC experts and the code developers from the different Scientific Challenges to start working on the codes. For the whole ALYA team, this would have been an excellent opportunity to interact with the HPC experts. Work on both CPUs and GPUs would have started. Due to the cancelation of the Hackathon, we have concentrated only on the CPU version for the moment, but we expect we can start with the GPU version shortly. Moreover, significant effort has been put into ALYA's

test suite to make the code more robust. On the other hand, it has been decided to change from an SVN repository to GIT and start using GitLab for a more professional workflow. All of this change is taking some time. The timing was quite unfortunate since the change from SVN to GIT took place a few days before the COVID-19 confinement. A course on GIT had been planned, but it had to be postponed due to COVID-19. Finally, it has been decided to try to separate Alya into different libraries. This will imply working on Alya's Separation of Concerns (SoC) to make the different parts of the code more independent from each other. We are currently working on this with the hope that it will make Alya more professional, easy to test, and robust.

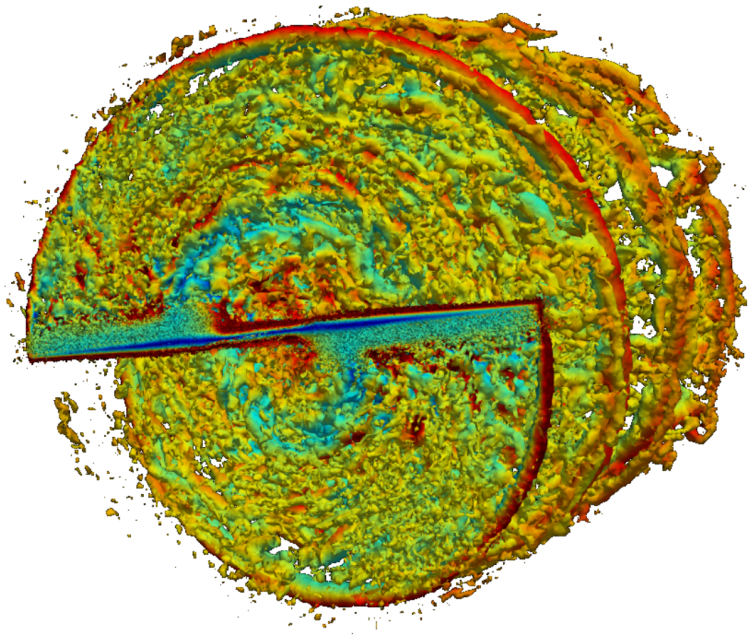


Figure 6: Instantaneous Q criterion isosurfaces coloured by velocity magnitude for the NREL VI wind turbine.

MPI overlaping between communication and computation Even though Alya has proven its scalability for up to hundreds of thousands of CPU-cores, some expensive routines could affect its performance on exascale architectures. One of these routines is the conjugate gradient (CG) algorithm. CG is relevant because it is called at each time step to solve a linear system of equations. Collective communications can create bottlenecks. The preconditioned CG (PCG) already implemented in Alya requires two collective communications. A pipelined version of the PCG (PPCG) algorithm, which allows to half the number of collectives, has been implemented. Moreover, non-blocking MPI communications were used to reduce the waiting time during message exchange even further. The resulting implementation was analyzed using Extrae/Paraver profiling tools. Several tests were performed using different number of processes/workloads to study the improvement in the scaling obtained with the implemented algorithms. The new PPCG algorithm is numerically equivalent to the PCG algorithm but, by reordering the operations, reductions are grouped and can be overlapped with the Sparse matrix-vector multiplication operations. It has been verified the new implementation produces the same convergence of the previous CG algorithm. It was planned to present the application of the Pipelined CG for wind farm problems in the ISC 2020 conference, which was canceled due to COVID-19. We are currently working on including these results on a

paper on Wind energy towards the exascale with Alya.

Coexecution on heterogeneous cluster (CPU + accelerators) and Fast and scalable geometric mesh partitioning based on Space Filling Curve Significant progress on co-execution on heterogeneous clusters (CPU + Accelerators) has been made. The Alya code has been adapted to work not only on CPUs but also on GPUs for Computational Fluid Dynamics problems, particularly Large Eddy Simulation cases. For such problems, a semi-implicit approach is used where the momentum equation is solved explicitly while the continuity equation is solved implicitly. The pressure matrix remains constant during all of the simulation, which involves tens of thousands of time steps. Thus, the computation time for its creation is negligible. Therefore, when a fractional step scheme is used, the two most expensive kernels are the right-hand side vector calculation for the momentum equation and the solution of a linear system for the pressure at each time step. For the right-hand side vector calculation, OpenACC has been used to adapt the code to GPUs. For the solution of the linear system, Alya's linear solvers have been ported to CUDA. We are currently working in WP3 to use EoCoE provided linear solvers that can run on GPUs. Considering that most of Alya can run in either CPUs or GPUs, we have decided to develop a co-execution approach that makes better use of current pre-exascale supercomputers, which typically blend GPUs and CPUs. The method is schematically described in Fig. 7. In this way, we make full usage of both GPUs and CPUs. CPUs are usually underused in such machines. A Fast and scalable geometric mesh partitioning based on Space-Filling Curve (SFC) has been key to enable the co-execution with a correct load balance between the GPUs and CPUs. At the beginning of the simulation, the SFC partitioning is called several times iteratively until an optimum partitioning of the mesh is obtained. In the first iteration, each MPI task (be it CPU or GPU) receives a specific portion of the mesh according to some initial weights. With this partition, it calculates a couple of time steps. Based on the computational time taken by each MPI task, it adapts the weights and repartitions again. After a couple of iterations, each processor receives the correct amount of work so that they all take nearly the same time. GPUs obviously receive a more significant chunk of the mesh than CPUs. In this way, the work done by the CPUs is spared in comparison to a pure GPU calculation.

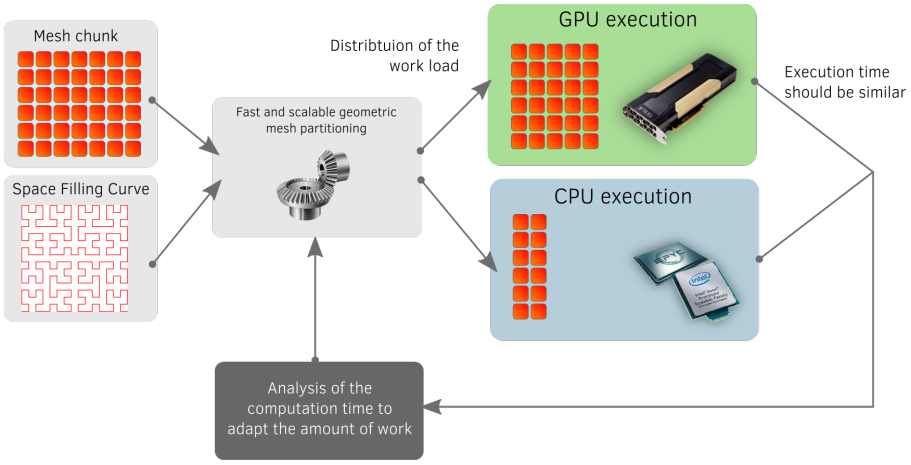


Figure 7: Description of the method used in Alya to distribute the load between the CPU and the GPU of a node.

Tests for wind energy problems are currently being performed on the MareNostrum POWER9 supercomputer formed by three racks of last IBM POWER technologies (POWER9 CPUs plus Volta GPUs) with a peak performance of 1.5 petaflops. We expect also to be able to perform larger tests on the Swiss Supercomputer Piz Daint. It was planned to present these results at the ISC

2020 conference, which has unfortunately been postponed. We are currently working to include them in a paper on Wind energy towards the exascale with Alya.

Dynamic coupling between rotating meshes and a fixed mesh. Alya counts with a parallel coupling library that allows to couple Alya to other codes. It also provides a coupling between two or more instances of Alya. This coupling library can be used, for example, to solve Fluid-Structure Interaction problems where one Alya solves for the Fluid part and the other one for the Solid part. The coupling allows the interchange of forces and displacements between both instances. The coupling library can also be used for problems in where one part of the domain rotates while the other one is fixed using a different instance of Alya on each part. Preliminary testing of the methodology for incompressible flow problems with complex geometry has provided positive results. Some robustness issues have been identified and solved. A rotating NREL Phase VI wind turbine has been simulated, using an unstructured mesh with 50 million tetrahedral, prismatic and pyramidal linear elements. We are currently running with a mesh of 400 million elements to analyze the effect of the mesh on the solution. These results, plus those for a case where the interaction between three wind turbines is studied, will be included in a presentation in the ParCFD congress. Figure 6 shows results for the wake behind the rotating wind turbine.

Figure 8 shows the decrease of the elapsed time per time-step when increasing the resources from 16 nodes (768 CPU-cores) up to 128 nodes (6144 CPU-cores) for three configurations: i) NO COUPLING: A case without rotation where a single mesh is used for the whole domain and there is no coupling; ii) STATIC COUPLING: A case without rotation solved with two meshes that do not match at the interface; iii) SLIDING MESHES: case where one of the meshes is fixed and the other one is rotating.

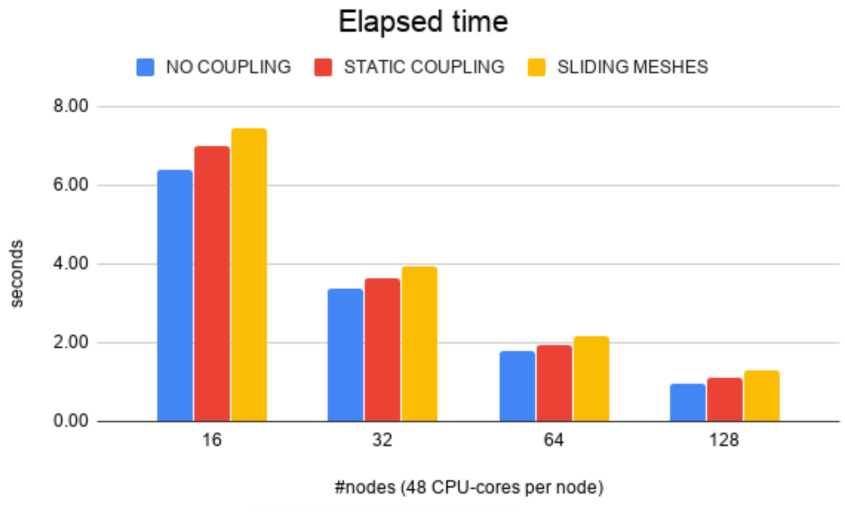


Figure 8: Elapsed time per time-step for three different configurations (find description in the text)

Scaling on exascale or pre-exascale resources Although this task is supposed to start in month 25, taking advantage of the excellent results we have obtained in WP3 while coupling to external linear algebra packages (PSBLAS/MLD2P4 and AGMG), important steps have already been given. Preliminary tests on complex terrain cases using unstructured tetrahedral grids have been performed, with mesh sizes of up to 2000 million elements and 300 million nodes. Good weak scalability results have been obtained up to 12k cores in Marenostum IV. We have a high expectation that we could probably run successfully up to 16000 million elements and 2400 million nodes using,

possibly, up to 96k cores. However, running such huge cases, we have rapidly run out CPU hours in our PRACE-EoCoE account in Marenostrum IV. A single test with 12k cores has used up 50k CPU hours, while our total resources in Marenostrum IV for these last six months are 200k CPU hours. They have already run out. We will have to move to Piz Daint or SuperMUC-NG where EoCoE still has some CPU time. Moving to Piz Daint is interesting because it will allow us to start testing the GPU implementation in both Alya and PSBLAS/MLD2P4. On the other hand, having to move from Marenostrum IV to SuperMUC-NG, which are very similar machines, is probably not the best solution. Moving to a new machine takes time for reading the user documentation, finding the correct modules to use, and recompiling the codes. Moreover, it implies having the runs spread among different machines. It would be much more comfortable to have all the resources concentrated in just one CPU supercomputer and one GPU supercomputer. Although the objective of CoEs is to try to run as close to the exascale as possible, we have found that it is not easy to perform tests with 100k cores in Marensotrum IV, Spain's Tier0 supercomputer. Such simulations are allowed once a year, coinciding with the electrical revision of the machine in August. We are currently trying to find out the situation in other Supercomputers.

5.2.2 Work progress in Meso-NH

Simulation work with MesoNH has started but the comparison of scientific, numerical and performance results has not yet been carried out. Details of the first simulation are in deliverable D1.2.

5.3 Work progress on task 2.2.2

The main objective of this task is to test an actuator line model in waLBerla. The work plan for this task was first updated in D2.1. Following the proposal and the first deliverable, this subtask can be divided into the following points:

- waLBerla code preparation for wind turbine
- Integration of the actuator line model
- First performance results on a single wind turbine
- Extension of the waLBerla models from a single wind turbine to wind farms

Fig. 9 describes the current work plan for task 2.2.2.

5.3.1 waLBerla code preparation for wind turbines

After the successful implementation of the wind turbine models (actuator line and disk models) in waLBerla [7, 21], the code-base was adjusted and extended to fit some special needs of the performed Lattice Boltzmann (LB) simulations.

Domain configuration Using the existent waLBerla boundary conditions, the implementation supports three major setups for the virtual environment of the wind turbine:

- a fully periodic flow for testing purposes,
- a wind tunnel setup (no-slip walls on the sides of the simulation domain, velocity inlet and pressure outlet in flow direction),

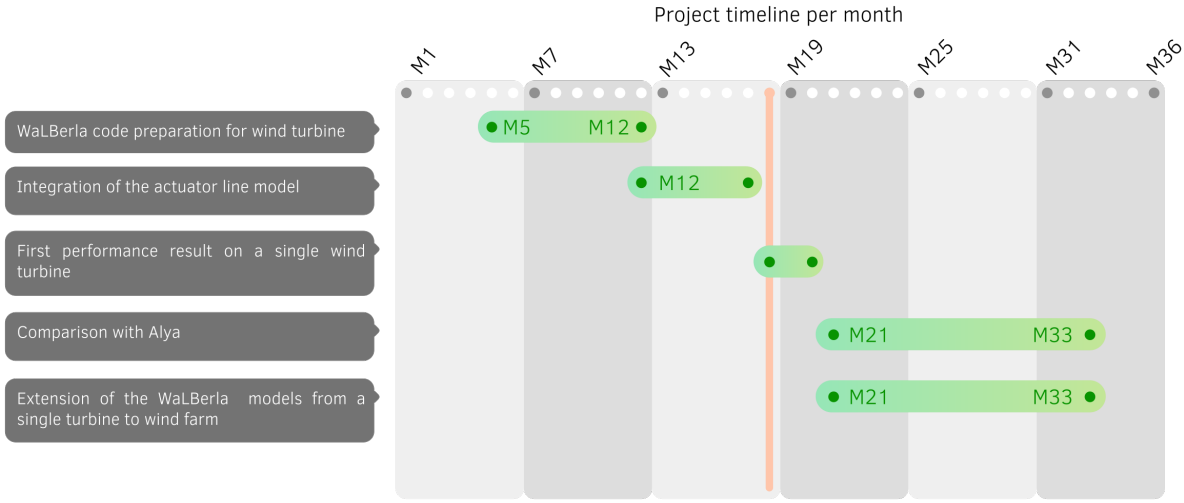


Figure 9: Breakdown (simplified Gantt chart) of the task 2.2.2 for waLBerla.

- an atmospheric flow configuration (no-slip wall on the bottom, periodic inlet/outlet, and slip-walls at the rest of the domain).

Numerical instabilities There are some factors in coupled Lattice Boltzmann and Actuator Line simulations that have an impact on the numerical stability of the simulations. First, the actuator line model itself introduces some numerical constraints. On the one hand, the mesh around the rotor must be fine enough to capture the build-up of the wake. A common rule consists in using 30 to 60 cells across the rotor diameter (Jah et al. [23]). However, this may not be sufficient to correctly resolve the tip vortex. On the other hand, the time step of the simulation must be fine enough to limit the actuator line tip motion to pass through no more than one mesh cell (according to Churchfield et al. [10]). For the Lattice Boltzmann method (LBM) this criterion is shown to be automatically fulfilled when respecting the typical operating conditions of a wind turbine - independent of the cell size. We have:

$$u_{LU}^{\max} = \frac{1}{\lambda},$$

where u_{LU} is the lattice velocity, λ is the tip-speed-ratio of the wind turbine. With typical lattice velocities of $u_{LU} \approx 0.05$, the criterion is fulfilled for tip-speed-ratios up to 20, which is beyond the operating conditions of a wind turbine. Moreover, some rule of thumb has also been derived regarding the width of the interpolation/projection Kernel. For the case of an isotropic Gaussian kernel, Troldborg [43] recommends using a Gaussian width greater than twice the local grid cell length $\epsilon > 2\Delta x$ to avoid artificial oscillations in the flow.

In this context, the Reynolds number $\mathcal{R}e = \frac{uD}{\nu}$ (u: wind speed, D: diameter of the rotor, ν: kinematic viscosity) are elevated compared to classical LBM simulations. In LBM, the kinematic viscosity is closely related to the so-called relaxation rate τ. Hence, to obtain these high Reynolds number, there are three choices: Increasing the number of cells per diameter, i.e. the resolution, increasing the lattice velocity, decreasing the relaxation rate. While increasing the resolution can easily be done, it also increases the computational effort and should be avoided where possible. The last two points influence each other and there is no strict rule to choose them. It can be shown that the lattice velocity can only be increased up to a certain point without causing numerical instabilities. However, it also holds that a decrease in the relaxation rate for $\tau \to 0.5$ is followed by a decrease of the maximal lattice velocity. These relations make the realization of high-Reynolds flows in LBM difficult.

One possibility to enhance the numerical stability and the correct representation of physics is the usage of turbulence models. Additional to the Smagorinsky model that is already implemented in waLBerla, the so-called WALE model was implemented [13]. Still, these turbulence models did not stabilize the simulation sufficiently for standard moment-based Lattice Boltzmann methods (SRT/TRT/MRT). Despite these methods being rather simple and fast, they show some limitations concerning their stability properties. These limitations make the usage of more advanced collision models inevitable.
Two other families of collision operators were investigated in terms of numerical stability and performance:

- *Cumulant Lattice Boltzmann methods (CLBM).* A different approach for the collision operator is the family of cumulant Lattice Boltzmann methods proposed by Geier et al. [18]. Cumulants are a good choice for high-Reynolds regimes and stability issues as they ensure Galilean invariance and the decoupling of mutually independent degrees of freedom. Nevertheless, these superior properties come at the price of reduced performance as more computations have to be performed.

- *Regularized Lattice Boltzmann methods.* As an alternative to the costly CLBM, we further investigated another family of LB methods, the Regularized LBM [27]. The idea behind the RLBM is to reduce approximation errors between the numerical scheme and the theoretical framework of Lattice Boltzmann methods. After a regularization step, a simple moment-based collision operator can be applied. The major advantage of RLBM is its performance close to these of the corresponding moment-based methods.
  Even though the RLBM increased the stability of simple benchmark problems, some spurious oscillations started to emerge from the wind turbine despite the use of adequate interpolation and distribution kernels. It seems that lowering the lattice velocity, u_{LU}, and therefore the Mach number Ma considerably improves the results. Although some discussions about the influence of the Mach number can be found in [6], we do not have a clear explanation to provide yet as the lattice velocity does not affect the stability of actuator line models (as was shown before).

Performance considerations and large scale. To improve the computational time when simulating large scale wind farm applications, the application was augmented and enhanced to optionally support local coarsening of the mesh. The refinement/ coarsening uses the work of Schornbaum [40]. Currently, this feature is not yet used extensively since the domain sizes are still moderate for single wind turbines. However, this will ease the later simulation of wind farms without the need to rewrite the LBM application.

5.3.2 First performance results on a single wind turbine

In our application, where LBM and the wind turbines only communicate via the force field, one can easily decouple these two parts for performance considerations. For a fast and performant simulation, both the Lattice Boltzmann and the wind turbine setup need to be optimized. Investigations concerning the single CPU [26] and the multi-core, multi-socket performance [12] of waLBerla already showed good results. There are, however, some factors to be considered to exploit the full potential - especially when coupling with other frameworks and methods. In addtion, the scalability of the application on supercomputers is crucial in HPC.

Scalability. Albeit the general scalability of waLBerla using the mesh refinement technology was already shown in [39], we also need to ensure this property within the wind turbine application.

As a first study of the scalability potential, a numerical test was performed on IFPEN cluster. The configuration consists of two *Intel(R) Xeon(R) Gold 6126* Skylake processors per node for a total of 36 cores. The considered domain consists of $360 \times 180 \times 180$ cells, uses periodic boundary conditions. A single wind turbine is placed in the center of the domain. The D3Q27 stencil with the entropic Smagorinsky TRT method was used. The results are shown in Figure 10.

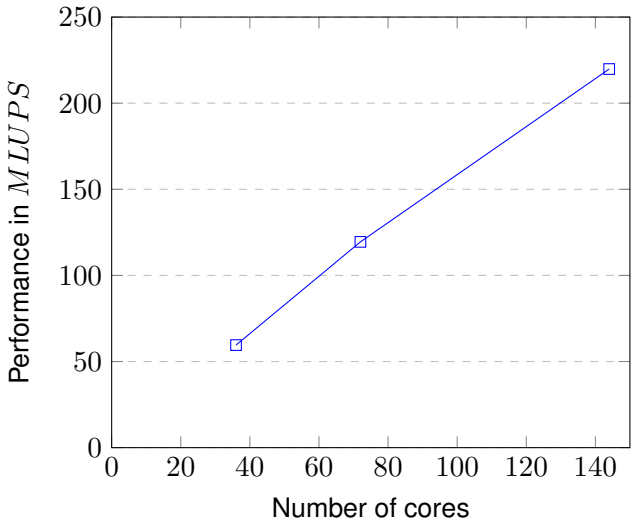


Figure 10: Performance of a D3Q27 entropic Smagorinksy LB model.

This early performance study did not yet use the LBM configuration as needed for wind turbine modeling but it marks an essential intermediate step to ensure scalability and performance within systematic, performance-driven software development.

LBM performance investigations. Good scaling properties are crucial for handling massive workloads by increasing computational resources. In large scale-applications, they are therefore indispensable. The key factor for a reasonable time to solution on any scale, however, is often the node-level performance of a codebase. It is not only affected by the architectural features of the processors but also by the code itself and how it adapts to the system topology and affinity.
For the generation of the corresponding LBM kernels, we use the code generation framework LBMPY [8]. This allows for the embedding of highly optimized LBM code in our wind turbine application that is tailored to the underlying hardware. Starting with the LBM setup, several factors need to be taken into account: the choice of the lattice Boltzmann method, the generation of corresponding efficient LBM kernels, the setup of the waLBerla domain (uniform grid or grid refinement), etc.

Some benchmark runs are performed to investigate LBMPY's abilities and to identify performance bottlenecks. Table 12 shows the results in $MLUPS$ (Mega Lattice Site Updates per second), where

$$MLUPS = \frac{n_{\text{cells}} * n_{\text{timesteps}}}{walltime \cdot 10^6}.$$

The reference domain consists of $480 \times 192 \times 192$ cells and uses periodic boundary conditions. For benchmarking, 6 cores of a *Intel Core i7-9850H* processor at 2.6 GHz were used. In the current configuration, the memory bandwidth is not yet saturated. Hence, the observations made in the following refer to the core performance. At a later stage, we will also need to consider the entire node-level performance, including bandwidth saturation, to exploit the full potential of one processor node. The rows of *Simple kernel* show the results for plain LBM methods that do not consider force or turbulence models. These runs were performed to investigate the general ability of LBMPY

	Performance in $MLUPS$	
	Uniform Grid	Refined Grid
Simple kernels		
D3Q19 SRT	239.28	66.41
D3Q27 SRT	170.96	58.45
D3Q19 Cumulant	95.98	55.05
D3Q27 Cumulant	5.39	5.17
Full kernel		
D3Q19 Cumulant	49.70	46.82
Turbine setup		
D3Q19 Cumulant	49.74	31.69

Table 12: Performance results for different lattice models using 6 cores on a *Intel Core i7-9850H* processor.

to generate efficient code. Whereas the SRT kernels show a decent performance on the uniform grid, some restrictions emerge for the refined grid. One explanation is the greater communication overhead in refined simulations due to the increased number of ghost layers. However, one must keep in mind that the time step size in the refined case is larger than for the uniform grid. So even though the performance in terms of $MLUPS$ is lower, the computational runtime may still decrease with grid refinement, particularly for larger domains.
For the case of uniform grids, we use the results of Bauer et al. [8] as reference values. Note that even though a different architecture was used, it still allows for the assessment of the generated lattice models. Both SRT models show good accordance with [8]. We even observe that the D3Q27 stencil performs with an expected decrease by a factor of 19/27. The cumulant LB kernels still exhibit more severe performance leaks. In the case of a D3Q19 stencil, the current version of lbmpy can optimize the kernel and reduce the computational expense. In [8], the cumulant LBM performs as well as the pure SRT models once the code saturates the bandwidth. For runs with 6 cores only, this is not the case. Hence, in our current test case, we observe a reduced performance for the cumulant LBM as compared with the SRT model for D3Q19 stencils.

In the case of a D3Q27 stencil, the numerical expressions become too complex for the symbolic optimization procedures currently realized in the current version of the lbmpy code generator. This is reflected in the major deterioration of the performance. While the effectivity and potential of the lbmpy code generator are underlined and emphasized by these results, it identifies a clear further research need. The generation of compute kernels for cumulant LBM versions in lbmpy requires improvement.

We point out in this report that in our approach expertise on advanced LBM methods must go hand-in-hand with in-depth knowledge of computer science technology, specifically in compiler construction and symbolic manipulation algorithms. The given funding situation within EoCoE-II provides only for exploring the potential of the novel co-design methodology and meta-programming technology, but the resources available are insufficient to conduct the needed in-depth interdisciplinary development. Substantial work on the code-generation technology is estimated to be in the order of 3-5 person-years beyond the funding available through EoCoE-II.

For further preliminary investigations therefore only the D3Q19 stencil was used to show the influence of other factors more reliably. The *full kernel* rows show the performance results of LBM methods with force and turbulence models. Adding more complexity to the kernel further decreases the performance, as was expected. Lastly, a full wind turbine setup with one turbine in the center of the domain was run. These results are preliminary, but they indicate the high potential of the methodology. Nevertheless, they primarily underline that the described combination of advanced co-design technologies can only leverage its benefits with substantial research effort outside

EoCoE-II funding.

Work in progress As pointed out in Section 5.3.1, two families of collision operators were investigated in terms of stability and performance. Although the Regularized Lattice Boltzmann methods show performance close to the standard moment-based LBM, the occurrence of spurious oscillations prohibits the physical simulation of wind turbines. As mentioned above, the reduction of the lattice velocity impedes these oscillations. Nevertheless, note that reducing the lattice velocity by e.g. a factor of 2 doubles the computational runtime for the same physical timespan. This, again, results in computationally expensive simulations. Therefore, we decided to primarily work with the cumulant Lattice Boltzmann methods. However, as outlined above, the kernels for cumulant LBM methods are not yet generated with full optimization in the current version of LBMPY as could be seen in the node-level performance considerations. Cumulants have good potential to simulate high-Re flows. The current code-generation pipeline of WALBERLA, however, is yet only optimized for the D3Q19 variant of cumulants [8]. It will be necessary to first identify the needed code transformations and simplifications manually and then additionally to implement the necessary automatic restructuring technologies. There are indications that we can reduce the performance loss compared to standard SRT/MRT schemes to less than 30%. Considering the good numerical stability, even for higher lattice velocities and therefore time step sizes, this performance loss is tolerable. Furthermore, the implementation of the wind turbines and the actuator line models have to be further investigated and optimized. Once this is done, we can proceed with proper benchmark runs to compare to other solvers.

5.4 Work progress on task 2.2.3

The main objective of this task is the performance comparison of the three codes using the flow over flat surface with wind turbine as a simulation case.

Fig. 11 describes the current work plan for task 2.2.3. The code comparison exercise has not yet started and will be performed in the second part of the project.

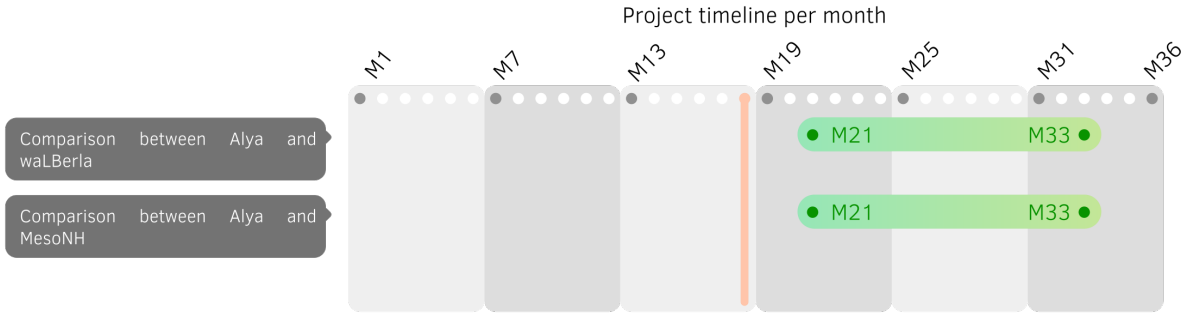


Figure 11: Breakdown (simplified Gantt chart) of the task 2.2.3 concerning the code performance comparison.

5.5 Risks, warning points and mitigation

6 Task 2.3 - Meteorology code optimization

6.1 Task overview

Task leader : FZJ

Risks / Warning points	Who	Impact	Mitigation
Due to COVID19, we could not obtain so far PRACE commutation hours on the TGCC-CCRT facility	MESO-NH	2-month delay impacting the MESO-NH simulation campaign.	Hopefully the situation should be solved. A small amount of computation time can be obtained using a preparatory access.
Tasks from Figure 5 that were supposed to end in M18 will extend up to M22 due to Covid-19	ALYA	Home working under abnormal conditions - kids at home	Not clear, will depend on when kids go back to school. Probably ask for extension of the project.
Cancellation of the Barcelona workshop on code optimization	ALYA	This workshop was essential to start the close collaboration between FAU and BSC on the CPU optimization of Alya.	The collaboration was initiated remotely and is currently ongoing. Further work on the GPU part of Alya has been postponed.
The PRACE resources allocated to CoEs are small and scattered over many machines. They do not allow for testing that meets the exascale objectives.	ALYA	Large scale simulation tests are limited in terms of number of cores and time at the risk of consuming an entire allocation at once..	While waiting to get more hours through traditional channels, it is possible to migrate to other machines. This solution is not satisfactory and requires additional adaptation work.

Table 13: Risk management in Task 2.2.

Participants: FZJ, FAU, CEA

The goal of the Meteorology scientific challenge is to improve weather forecasts (wind properties, cloud coverage, aerosols) for electricity production from solar and wind. Solar and Wind power prediction is performed using a framework gathering multiple codes working together. These codes, WRF (Weather Research Forecasting model [44] for meteorological analyses, and EURAD-IM for air quality assessments (with aerosol focus for EoCoE), are offline coupled and capable to perform large ensemble simulations of the order of 1000 members. The ensemble system is integrated into ESIAS. As the meteorological model WRF is a community code that is mainly maintained by NCAR (National Center for Atmospheric Research, USA) only the code EURAD-IM is concerned in WP2. The code and the related work is described in the following section.

6.1.1 Flagship code EURAD-IM

EURAD-IM simulates the formation and transportation of atmospheric chemical species and particles (aerosols) on the regional to continental scale. It is offline coupled with the regional meteorological model WRF. An advection-diffusion-reaction equation, with multiple solvers for chemistry and aerosols, is used. In EURAD-IM, the stiff solver for gas phase chemistry is one of the main performance bottlenecks and most time consuming part. The objective of WP2 is to improve the codes efficiency to address the Meteorology simulation challenges with main items:

- PDI integration (with CEA PDI experts) for IO optimization in WP4 and ensemble runs in WP5,
- Code refactoring (with FAU) including change of data structure for vectorization and memory management,
- Node level optimization (with FAU) and vectorization of the stiff gas phase ODE solver,

People	Position	Role	Period
Hendrik Elbern, PhD	Senior scientist at University of Cologne (RIU)	Former Scientific coordinator for Meteorology	Retired
Garrett Good, PhD	Scientist at Fraunhofer Institute for Wind Energy Systems (Fraunhofer IEE)	Scientific coordinator for Meteorology	M1-M36
Philipp Franke, PhD	Postdoctoral fellow at FZJ	EURAD-IM code expert	M1-M36
Carl Burkert	Master student at FZJ	Performance analysis and GPU porting of EURAD-IM	not paid by EoCoE, M0 - M20

Table 14: Team Members for EURAD-IM within EoCoE.

- Hybrid parallelization MPI + OpenMP/OpenACC to improve the parallelization on large-scale CPU machines first and leverage the possibility of GPU usage.

Table 14 shows the team members of EURAD-IM involved in EoCoE. Hendrik Elbern was the Meteorology Scientific Leader at the beginning of the project. He has retired at the end of 2019. Garrett Good is the new leader of the Meteorology SC. Philipp Franke, postdoctoral fellow at FZJ, is now coordinating activities around EURAD-IM in WP2. Carl Burkert is a student in applied mathematics and computer science writing his master thesis at FZJ.

6.2 Work progress on task 2.3

Fig. 12 describes the current and updated work plan for task 2.3. The optimization work has been divided into subtasks in deliverable D2.1. Compared to the provisional dates provided in the first deliverable, we have postponed certain tasks by a few months, partly due to the health crisis. Optimization work at the node level is the most general. In the second part of the project, efforts will also be focused on the GPU port of the ODE solver. This work was initiated in the first part of the project through training in particular. The EURAD-IM development teams are in contact with an expert on these issues at FZJ. The porting of this type of solver has already been carried out on GPU independently of EURAD-IM but the method can be easily reused. Parallelism hybridization (MPI + OpenMP) will be performed in parallel with the GPU porting. The integration of IDPs will be left to the second year of the project.

Performance analysis have been conducted in a simplified setup to evaluate the codes performance under real conditions. The simplifications comprise a reduced number of time steps and iterations per simulation. These simplifications were necessary in order to limit the memory and compute time required for the analysis. Anyway, the results of this analysis can be extrapolated to full simulations with more time steps and iterations.
The strength of EURAD-IM is the ability to performed four dimensional data assimilation (4D-var) analysis for atmospheric constituents. Using 4D-var, the observation-model discrepancy within a time window (assimilation window) can be projected onto initial values and emission rates of chemical species, which are two of the key drivers of forecast uncertainty. This data assimilation method includes the use of the adjoint code of the forecast model. In EURAD-IM, the adjoint code is designed for each routine separately to ensure a modular code setup. The adjoint code

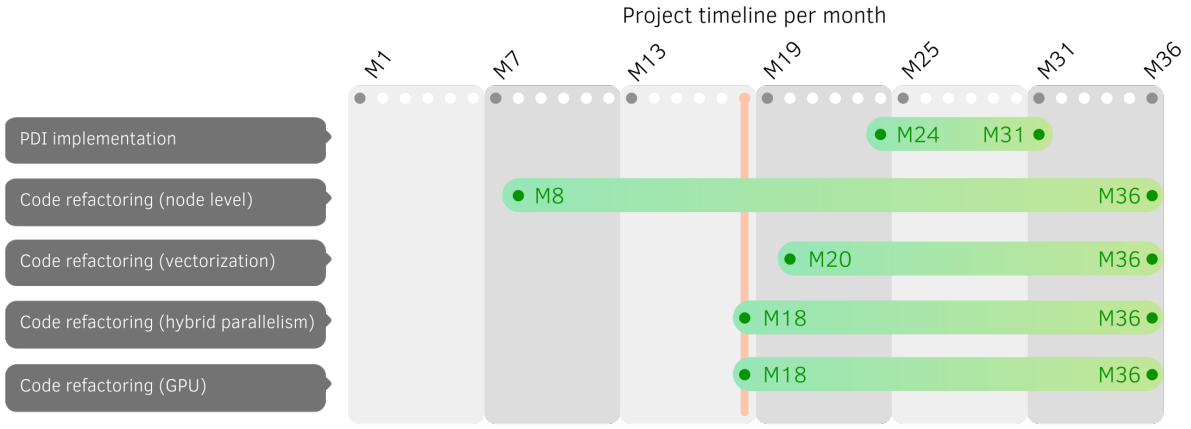


Figure 12: Breakdown (simplified Gantt chart) of the task 2.3 for EURAD-IM.

includes the forecast model to calculate the model state at which to linearize the model, essentially at each line of the code. In total, the 4D-var model calculation takes about 3-4 time longer per iteration than the pure forecast model.
By definition, there are certain similarities in the code structure and layout of the forecast model and the adjoint code. From the performance optimization point of view, these similarities lead almost to an effective doubling of the speedup gained from model improvements. Each speedup gained from optimizations of the forecast code lead to similar speedup of the adjoint code.
The performance analysis was performed for the forecast code and its adjoint separately using 239 cores on JUWELS. The simulation included two iterations and three simulation hours (54 time steps) for the European model grid (15 km horizontal resolution, 348x289 grid boxes, 30 vertical layers) on January, 01, 2016. Real analyses comprises 24 simulation hours and 15-20 iterations. Besides the stiff solver for gas phase chemistry, further performance bottlenecks have been identified (see also Tab.15, lower left). These main performance bottlenecks were the

- adjoint code of the stiff gas phase chemistry solver (ADCHEM in Tab.15);
- adjoint of the aerosol module for secondary inorganic aerosols (AD_EQL5);
- adjoint implicit solver for vertical diffusion (ADVDIFFIM);
- writing and reading of intermediate model states to/from file for later use in the adjoint code (TRAJ_IO);
- MPI parallelization, mainly the separation of the master (IO operations) from the workers (model calculation);
- serial netCDF from the master forcing the workers to wait at the next MPI exchange;
- load imbalances in multiple modules.

It is emphasized that the relative shares of CPU-time may differ between simulated days because of the differences in the simulated chemical regime. Nonetheless, the key bottlenecks of the codes performance stay the same.
During the performance analysis, first code improvements by refactoring have been done, which had also a positive effect on the performance. These improvements were:

- separation of the horizontal and vertical advection: it was recognized that in some vertical columns the CFL criterion was violated, which led to a halving of the time step for advection. After the separation, only the respective vertical column is affected by this halving of the

Routine	forecast	adjoint
MPI_BCAST	554 s / 10.9 %	16,797 s / 49.3 %
MPI_GATHER	—	2,310 s / 6.7 %
MPI_ALLGATHER	365 s / 7.2 %	584 s / 1.7 %
CHEM	1,102 s / 21.7 %	789 s / 2.3 %
ADCHEM	—	6,527 s / 19.2 %
WADVEC	1,211 s / 23.9 % (857 s / 18 %)	1,672 s / 4.9 % (848 s / 2.6 %)
ADCWADVEC	—	1,678 s / 4.9 % (839 s / 2.6 %)
EQL5	237 s / 4.7 %	237 s / 0.7 %
AD_EQL5	—	988 s / 2.9 %
VDIFFIM	126 s / 2.5 %	242 s / 0.7 %
ADVDIFFIM	—	537 s / 1.6 %
TRAJ_IO	634 s / 12.5 %	464 s / 1.4 %
MEGAN_GAMMA_VALUES	153 s / 3.0 %	—

Table 15: Accumulated exclusive time for selected modules and its relative contribution to the total accumulated run time of the performance analysis of EURAD-IM in Task 2.3. Exemplarily, results for 18 time steps ($\hat{=}$ 1 simulation hour) are shown. Large accumulated exclusive times for MPI modules indicate load imbalances between the MPI threads in other modules. The EURAD-IM modules listed are: CHEM: stiff ODE solver for gas phase chemistry; ADCHEM: adjoint of CHEM; WADVEC: advection scheme (horizontal and vertical); ADCWADVEC: adjoint of WADVEC; EQL5: solver for secondary inorganic aerosols; AD_EQL5: adjoint of EQL5; VDIFFIM: implicit solver for diffusion; ADVDIFFIM: adjoint of VDIFFIM; TRAJ_IO: IO of intermediate model states for use in the adjoint code. MEGAN_GAMMA_VALUES: calculator for biogenic emissions; For the advection modules the accumulated exclusive time after the code refactoring is given in parenthesis.

time step while the remaining vertical columns and the horizontal advection keep the full time step.

- improved MPI exchange of the CFL criterion for horizontal advection: before, the local CFL criterion of each MPI task was send to the MPI master. The master calculated the global CFL criterion and sent the results to the workers. This was replaced by a global MPI exchange. Interestingly, the use of MPI_ALLGATHER in combination with local calculation of the maximum wind speed for each worker showed a better performance than the use of MPI_ALLREDUCE(). The reason for this needs too be further investigated.

- mitigation of load imbalances due to the calculation of biogenic emissions in later iterations. The biogenic emissions are calculated in the first forecast run of EURAD-IM accompanied with writing the calculated values to a file, which is retrieved in the adjoint part and now at later iterations, too.

EURAD-IM comprises a module for load balance optimization, which is outdated and not available for adjoint calculations. It needs to be updated for the use in the current model setup and further tests are required to analyze its applicability for adjoint calculation. Especially the treatment of stored model states need to be analyzed carefully when applying the load balance optimization module to the adjoint run. Nonetheless, the run time of the performance bottlenecks, especially the solvers for the chemical and aerosol state, will reduce the load imbalances as well. Thus, the modules need to be optimized before applying additional load balance optimization.
Fig. 13 shows the speedup and the run time of the advection routine (WADVEC) and its adjoint (ADCWADVEC) before (black) and after optimization (blue). For the run time, the full run time including MPI communication for local domain boundaries is included (dotted lines). Both

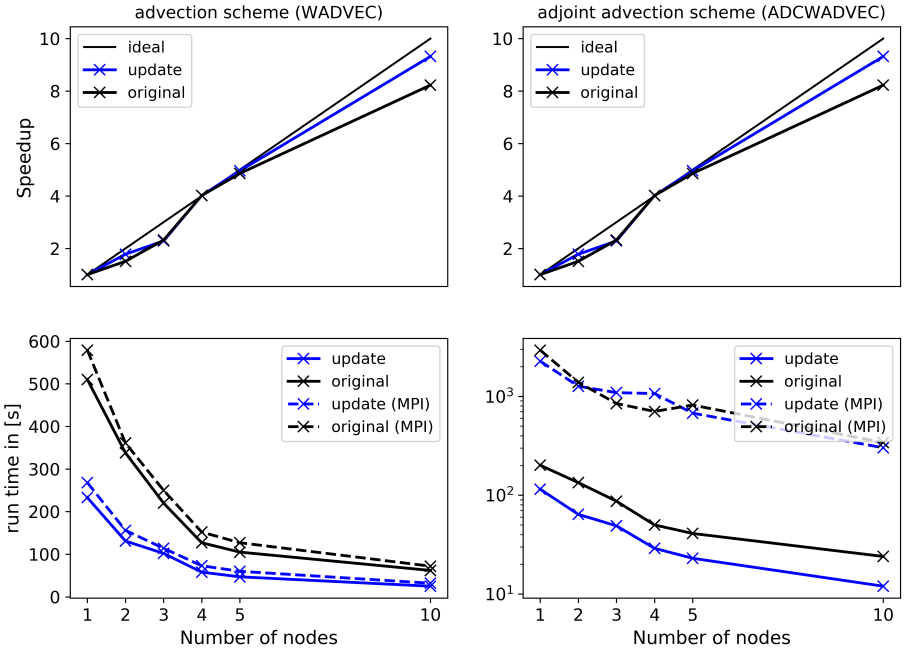


Figure 13: Speedup (top) and run time (bottom) of the advection schemes WADVEC (left) and its adjoint (right) of EURAD-IM. The speedup and run time is given for the codes before (black) and after (blue) the optimization. Additionally, the run time of the full routines including MPI communication is shown (dotted lines).

routines show almost perfect scaling behavior in terms of speedup with slight improvements after the optimization using 10 nodes. The run time in both routines can nearly be halved due to the updates. By considering the routines including MPI communication the load balance especially in the adjoint advection routine becomes obvious. While the communication does not significantly influenced the run time of the advection routine (WADVEC), it dominates the run time of the adjoint routine (ADCWADVEC) eliminating the improvements gained by the updated adjoint routine. The speedup test was performed for January, 06, 2016. As the load imbalance results mainly from the stiff ODE solver for gas phase chemistry, it also does depend on the simulated day. This can be seen in the differences in run time of the advection schemes in Tab. 15 and in Fig. 13. As a next step, the load imbalance will be approached by implementing OpenMP for hybrid parallelization for the main expensive routines.

6.3 Risks, warning points and mitigation

Risks / Warning points	Who	Impact	Mitigation
COVID-19 lockdown	EURAD-IM	delay in finalizing code performance analysis and optimization	shift of tasks

Table 16: Risk management in Task 2.3.

7 Task 2.4 - Materials code optimization

Task leader : FZJ

In the Materials scientific challenge, one of the goal focuses on improving the modeling of solar cell device at atomic scale and use our high-end numerical tools to determine the properties of new materials for photovoltaic. Table 17 shows the team members of the task 2.4 involved in the WP2 of EoCoE. Additional members of the team are involved in the WP1 and are not listed in the table.

People	Position	Role	Period
Edoardo Di Napoli	Senior scientist at the Jülich Research Center (Forschungszentrum Jülich – FZJ)	Supervises and coordinates the libNEGF activity	M1-M36
TBA in substitution of ~~Paul Baumeister~~	Experienced programmer at FZJ	Involved in porting the refactored code to distributed multiple GPU architectures	M24-M36
Sebastian Achilles	Research Scientist at FZJ and PhD student at RWTH	HPC expert: In charge of the refactoring and the parallelization	M1-M30
Alessandro Pecchia	Lead scientist at CNR	In charge of development of new functionalities and code validation	M12-M36
Gabriele Penazzi	Research Scientist	In charge of development of new functionalities and code validation	M12-M36
Georg Hager	Senior Scientist at FAU	Expertise and advisor in node-level code optimization	M25-M28

Table 17: Team Members for task 2.4 within the WP2.

Task participants : FZJ, CNR, FAU

7.1 From PVnegf to libNEGF

The simulation of quantum transport is at the core of the Materials for Energy Scientific Challenge. Initially, the flagship code of choice was a developed within the IEK-5 institute part of the FZJ partner. PVnegf provides photocarrier dynamics (generation, transport and recombination) of nanostructured regions and at complex interfaces. It solves the steady-state non-equilibrium Green's function for charge carriers coupled to photons and phonons. Both interactions are treated on the level of self-consistent Born self-energies. Despite the non-ballistic formalism in PVnegf is at a very advanced stage of development, that same cannot be said for the type of physical systems it can simulate.

PVnegf advanced functionalities have been developed based on a simplified geometry. Namely, PVnegf targets quasi one-dimensional (1D) systems in a typical simplified tight-binding approximation with only two bands. In practice, all solids are mapped to an atomic chain with 2 orbitals per site in the chain. In order to simulate interfaces between amorphous and crystalline silicon (as stated in the task T1.3.1-3 of D1.1), what is needed is a full 3D multiband treatment. Bringing PVnegf to achieve such target together with its refactoring and parallelization was part of the overall aim of the combined effort in WP1 and WP2 of EoCoE. Six months after the start of the projects a number of obstacles became evident.

1. To go from 1D, 2-band to 3D, multiband requires a full reworking not only of the code but also of the mathematical formalism behind it.

2. Once the multi-band is in place, bringing the code towards large scale simulations will require a new parallelization scheme that cannot take advantage of the work currently under way on the single band code.

3. Even in the unlikely case the first two points could be achieved by the end of the project, we realized the base of users would be restricted to EoCoE. From the point of view of impact and technology transfer this is and undesirable result.

4. Once becoming a full 3D multiband code, PVnegf would require validation. Such validation could take many months if not years, and need a qualified scientific lead. Unfortunately, the main developer and scientific lead of the code, Dr. Urs Aeberhard, just left FZJ to work for the industry. Carrying out validation without the scientific lead is unthinkable.

In order to address the obstacles above and still bring to fruition the tasks established at the beginning of the project, it has been decided to change flagship code. The search for the right candidate took up to the end of 2019.

7.1.1 The path to a new Quantum Transport code

The methodology of non-equilibrium Green's Functions (NEGF) has seen a large development during the 90s, particularly with applications to mesoscale physics and 1-dimensional devices such as III-V semiconductor heterostructures (quantum-wells, Resonant Tunneling devices, QCLs, etc.). The most advanced code in terms of performances was NEMO, based on an empirical tight-binding (TB) formulation for the electronic Hamiltonian and phenomenological electron-phonon coupling.

Since the early 2000 several codes solving NEGF equations with density-functional approaches have been developed throughout the world starting from the original TranSIESTA implementation. The experience matured into different projects, most of which commercial (ATK, NanoDSim). Interestingly, beside few exceptions, there are currently no open-source (or readily available) packages for large-scale quantum transport simulations. Several DFT codes, although highly specialized, suffer from severe scalability bottlenecks currently limiting the problem size that can be solved.

Full DFT+NEGF codes

- Smeagle/Gollum (U. Lancaster, Free for Academic),
- ATK/Quantum Wise (Commercial)
- WanT (Wannier Transport for Quantum Espresso or VASP, possibly discontinued)
- NanoDSim (LMTO, Closed or commercial)

Empirical Tight Binding + NEGF codes

- TB_Sim (CEA, closed)
- OMEN (U. Purdue and ETH, closed)
- NEMO 5 (U. Purdue, Academic license with several limitations)
- libNEGF (University of "Tor Vergata" and CNR, LGPL license)

Based on their level of development and license status only one code was considered a viable alternative for PVnegf: libNEGF

The new flagship code: the LIBNEGF project LIBNEGF is a LGPL project seeded in 2008 at the University of 'Tor Vergata' and CNR, hosted on github (`https://github.com/libnegf`). It is a general purpose non-equilibrium Green's function library to compute the density matrix and transport in open quantum systems such as nano and molecular devices. The library is developed as a general-purpose tool that can handle any input Hamiltonian, from most diverse problem formulations. Indeed it has been interfaced to several different codes such as,

- Density-Functional Tight-Binding (DFTB) code (`https://gihub.com/dftbplus`)
- Finite element code (TiberCAD) for both k.p and effective mass Hamiltonians (proprietary code)
- Empirical Tight-Binding Hamiltonians (within TiberCAD)
- Hessian matrices for phonon transport (development branch of dftb+)

Besides being integrated in other academic codes, LIBNEGF is embedded in the proprietary package suite "Materials Studio", formerly developed by Accelrys, acquired by Dassault Systems and renamed as Biovia. Biovia has extended the interface of LIBNEGF also to the ab-initio software DMol3.

7.2 Roadmap and milestones for LIBNEGF

Having moved from PVnegf to LIBNEGF brought a new partner in this task, namely the institute that is behind the main development of LIBNEGF: the "istituto per lo studio dei materiali nanostrutturati" of the CNR. The latter is already a partner within the EoCoE project. To support the acquired partnership, FZJ has transferred a small part of its budget (3PMs) to CNR starting from beginning of June 2020.

7.2.1 The exascale potential of LIBNEGF

The main motivation behind the joint effort within EoCoE is to fill the gap of available tools for quantum transport simulations on large supercomputing facilities, especially in the perspective of exascale computing facilities. The NEGF formalism is a highly computing intensive method that provides an excellent example of exa-scale applications. Scaling of the method up to 100,000 cores have been demonstrated, at least within OMEN, thanks to 3 levels of parallelism obtained by distributing **k**-points, energy-points and a domain decomposition.

The goal set on EoCoE is to increase the level of parallelization to execute the code on the entire cluster of one of the largest supercomputers in EU by the end of the project. The challenges to be solved are multiples, especially concerning data transfer bottlenecks that might require smart strategies of data distribution. A notoriously problematic bottleneck of the NEGF method, both in terms of computation as well as memory consumption, is the scaling with lateral supercell size.

Scaling up the lateral dimension poses several challenges. In order to perceive the problem one should consider that for instance a layer of Si crystal the size of 10x10 nm comprises 3200 atoms, involving $m = 12800$ basis sets in the simplest sp^3 approximation. This alone requires a memory storage of 2.6 Gb in double precision complex numbers required by the Green's function. Assuming to accomodate this on a single node and the typical need of about $6Mm^2$ of memory in order to compute the Density Matrix gives a limiting upper bound in the current implementation of about 10 nm length on a 96 Gb node. Simple transmission calculations in equilibrium require less memory (Mm^2). This is for a single energy and **k**-point, which are logically distributed over cluster nodes. Inelastic scattering involves an energy convolution making the problem non-local in

energy. Internode communications of such a large amount of data can be a severe bottleneck, hence recalculation on the fly might be a better option. The above considerations naturally bring into play mixed precision arithmetics. The OMEN-DaCe code, for instance, heavily resorts to mixed precision. Other strategies involging tensor products for efficient **k**-point summations have been tested on the PVnegf code and can give substantial efficiency boosts. However in this case the **k**-grid and energy-grid are distributed over a cartesian grid of nodes.

In order to illustrate the promise of LIBNEGF as an exascale candidate we report below a number of tests we have already performed on the code, include a performance profile and several scaling experiments. In particular the profile already allowed us to carry out a small but important optimization which improved performance substantially with respect to its adoption by the EoCoE project.

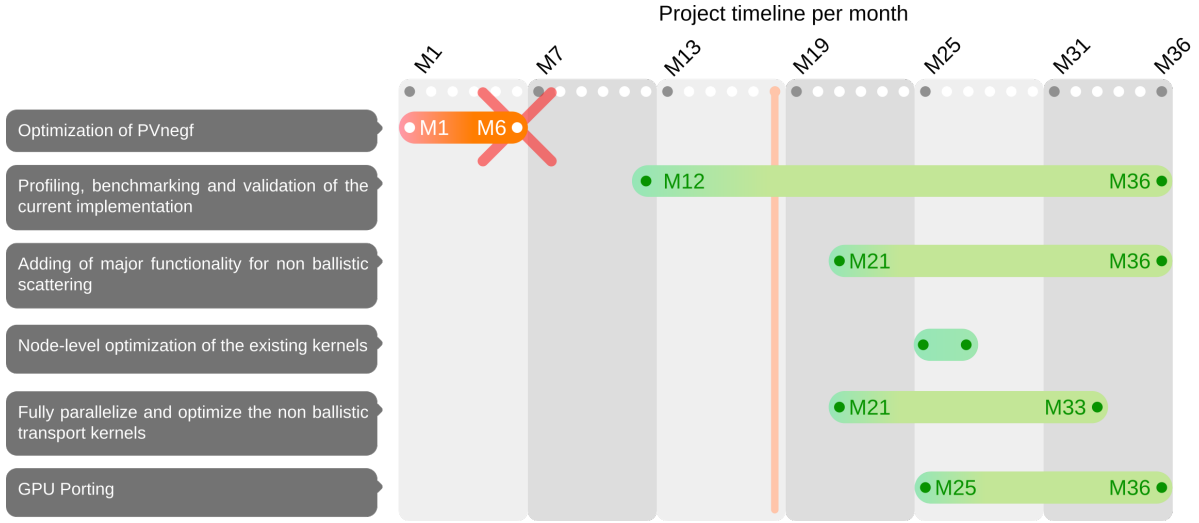


Figure 14: Breakdown (simplified Gantt chart) of the task 2.4 for LIBNEGF.

7.2.2 Performance Analysis

In order to profile LIBNEGF, we created a sequence of test input files that allow us to perform a short profiling run, but at the same time are close to the actual physical systems that will be simulated later on. The physical system of the test input files is a Silicon layer supercell, where any given number of layers can be generated. This allows us to easily generate any physical system of arbitrary size. The input files generated are executed only with ballistic transport. Once the additional functionalities, which will extend LIBNEGF by including non-ballistic scattering, will be developed within the scope of T1.3.1-3 of WP1, we will extend the input files to cover these functionalities as well and re-evaluate the performance.

To simplify the profiling during the project and increase the efficiency of repeated profiling and performance evaluations, we decided to create a JUBE[16, 29] script. JUBE is a framework that allows one to write a recipe on how to automate each individual step that normally would have to be performed by hand, such as configuration, compiling, running all benchmark suites, postprocessing and result verification and analysis. In addition the JUBE script is portable across several platforms. For the performance evaluation, we wrote a JUBE script for different tools: Intel APS [**APS**], Intel VTune [**VTune**], and Score-P [24]. While there was initial overhead in the creation of these scripts, The usage of JUBE script automating all steps, allows us to re-run the profiling at any point by just executing one command on a continuous basis—e.g. after every improvement to the code.

The most time consuming routines of the profiling of the 2x2 silicon input are given in table 18.

Routine	time (sec)	percentage
Total Runtime	3468	100%
`inversions_MP_zinv_`	1687	48.64%
`contselfenergy_MP_decimation2_`	621	17.91%
`integrations_MP_integrate_el_`	161	4.66%

Table 18: Profiling of LIBNEGF with the 2x2 silicon test input. The table lists the top three most time consuming functions.

The profiling was executed on JURECA cluster hosted by the Jülich Supercomputing Center one 1 node with 4 MPI ranks per node and 12 OpenMP threads per MPI rank. 48.64% was spend in the inversion routine `zinv`. In this routine two LAPACK routines are called: `zgetrf` and `zgetri`. In the specific, we are linking against the Intel MKL, which is closed source library. SCORE-P uses source instrumentation which is not able to measure what is happening inside library functions. This would only be possible with EBS sampling methods. In the next subsections we analysed both functions in detail.

7.2.3 Code Optimization and Improvements

To analyze the node level performance within the routine `inversions_MP_zinv_` we switched to a different tool. As described before there are only two LAPACK function calls within this routine, `zgetrf` and `zgetri`. In order to get a better understanding of the performance of these functions we used the ELAPS Framework: Experimental Linear Algebra Performance Studies [33]. This tool allows to benchmark every linear algebra routine for varying input or varying number of threads. With the help of this tool we could explore a large parameter space flawlessly.

The inversion routine is part of the Dyson equation, where we need to compute the inverse of an complex matrix inv(A). The two LAPACK [2] function do the following: `zgetrf` computes an LU factorization of a general double complex M-by-N matrix using partial pivoting with row interchanges. The factorization has the form

$$A = P \times L \times U$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if $m > n$), and U is upper triangular (upper trapezoidal if $m < n$). This is the right-looking Level 3 BLAS version of the algorithm. `zgetri` computes the inverse of a matrix using the LU factorization computed by `zgetrf`. This method inverts U and then computes inv(A) by solving the system inv(A) $\times$ L = inv(U) for inv(A).

With ELAPS we benchmark `zgetrf` and `zgetri` for increasing number of cores on one node on JURECA. In figure 15 the runtime is plotted as a function of the used core. For both x and y axis logarithmic scale is used. Two things can be identified: 1) The LU factorisation `zgetrf` is scaling well with increasing cores. 2) The inversion `zgetri` shows almost no scaling. The combined scaling of functions is plotted as well: The scaling behavior of both functions combined is almost similar to the one of `zgetri`. This means that the resources on the node level are not used fully utilized, only 1 core is used, the others cores are idling. This implies bad node level performance.

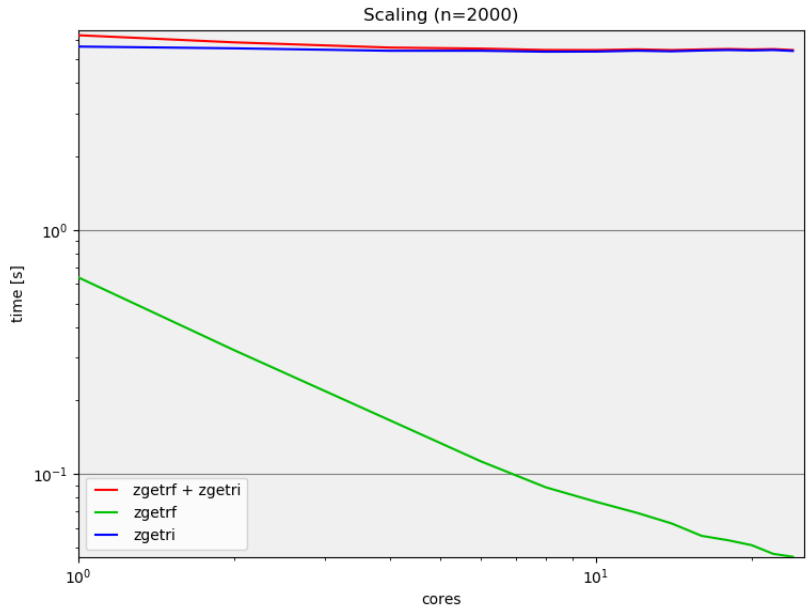


Figure 15: ELAPS Benchmark Result of the LAPACK routines used within the function zinv of LIBNEGF. For an exemplary matrix fixed size of $n = 2000$ the run time of each function is shown as a function of the cores used. A log-log scale is used to put the emphasis on the scaling behavior. The combined runtime of zgetrf+zgetri is shown in red, the runtime of zgetrf in green and the runtime of zgetri in blue. We can see that zgetrf (green) is scaling well with increasing number of cores, while zgetri and also the both combined are scaling quite badly with increasing number of cores.

7.2.4 Performance Model for Inversion

There are multiple ways to solve an inversion. Within Lapack there are also different routines. An other one is called `zgetrs` which solves a system of linear equations

$$A \times X = B, A^T \times X = B, \text{ or } A^H \times X = B$$

with a general N-by-N matrix A using the LU factorization computed by `zgetrf`. While `zgetri` is a tridiagonal solver which is inherently a iterative sequential algorithm, the algorithm of `zgetrs` can be parallelized.

We repeated the experiment with ELAPS and we benchmark in addition `zgetrs`. In figure 16 the runtime is plotted as a function of the used core. For both x and y axis logarithmic scale is used. The following things can be identified: 1) `zgetrs` does scale well with increasing number of cores compared to `zgetri`. 2) The combined runtime of `zgetrf` and `zgetrs` is also scaling well with increasing number of cores compared to the combined run time of `zgetrf`+`zgetri`.

Based benchmark on JURECA with ELAPS with varying number of cores p and matrix size n we created a multi-parameter performance model for the functions. We tried first well known automated performance modelling tools for this task, but due to lack of functionality in the end we performed the performance modelling manually using PYTHON. This are the models we calculated:

- `zgetrf` performance model:

$$-2.59 \times 10^{-4} - 2.90 \times 10^{-11} n^3 \log_2(p) + 4.05 \times 10^{-10} n^{2.75}$$

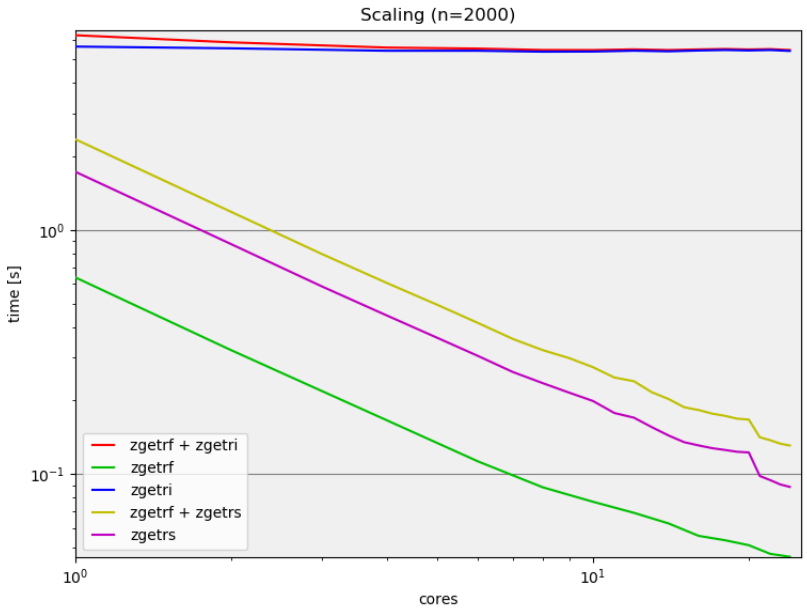


Figure 16: ELAPS Benchmark Result of the LAPACK routines zgetrs. For an exemplary matrix fixed size of $n = 2000$ the run time of each function is shown as a function of the cores used. A log-log scale is used to put the emphasis on the scaling behavior. The combined runtime of zgetrf+zgetri is shown in red, the runtime of zgetrf in green, the runtime of zgetri in blue, the runtime of zgetrf+zgetrs is shown in yellow and the runtime of zgetrs in pink.

- `zgetri` performance model:

$$1.70 \times 10^{-3} + 4.30 \times 10^{-10} n^3$$

- `zgetrs` performance model:

$$2.50 \times 10^{-4} - 5.27 \times 10^{-11} n^3 \log_2(p) + 1.90 \times 10^{-10} n^3$$

In table 19 we compared the with our performance model predicted run time of the old approach with `zgetrf` + `zgetri` compared with the new approach `zgetrf` + `zgetrs`. The matrix sized used correspond to our test input files for benchmarking, namely the 2x2, 3x3, 4x4, 5x5 and 6x6 silicon supercell. We can conclude the following: 1) `zgetrf` + `zgetrs` is always faster compared to `zgetrf` + `zgetri` 2) `zgetrf` + `zgetrs` shows a much better scaling behavior. Compared to `zgetrf` + `zgetri` we can see a speedup of 46x on 1 JURECA node with 24 cores for the largest matrix size $n = 2592$.

7.2.5 Scaling of LIBNEGF

The evaluate the node-level improvements described above and to check the scaling behavior of LIBNEGF we did a couple of scaling experiments on JUWELS. In table 20 and Figure 18 we did a node-level sweetspot analysis for the 2x2 silicon supercell input for varying configurations of MPI ranks and OpenMP thread, where the product of both is equal to the number of CPU core on one JUWELS node. If the MPI parallelization and the node-level parallelization would have a similar efficiency we would expect an almost similar runtime. We preformed this test for

	zgetrf + zgetri			zgetrf + zgetrs			Speedup		
n	1 c [s]	12 c [s]	24 c [s]	1 c [s]	12 c [s]	24 c [s]	1 c	12 c	24 c
288	0.015	0.012	0.012	0.009	0.002	0.002	1.69	7.42	6.68
648	0.146	0.118	0.121	0.085	0.010	0.009	1.73	12.36	13.33
1152	0.794	0.633	0.651	0.459	0.045	0.033	1.73	13.94	19.60
1800	4.134	3.502	3.548	1.703	0.171	0.098	2.43	20.49	36.19
2592	14.978	13.156	13.153	5.016	0.481	0.281	2.99	27.36	46.86

Table 19: Performance Model of the Inversion of LIBNEGF for increasing matrix size for zgetrf+zgetri and zgetrf+zgetrs.

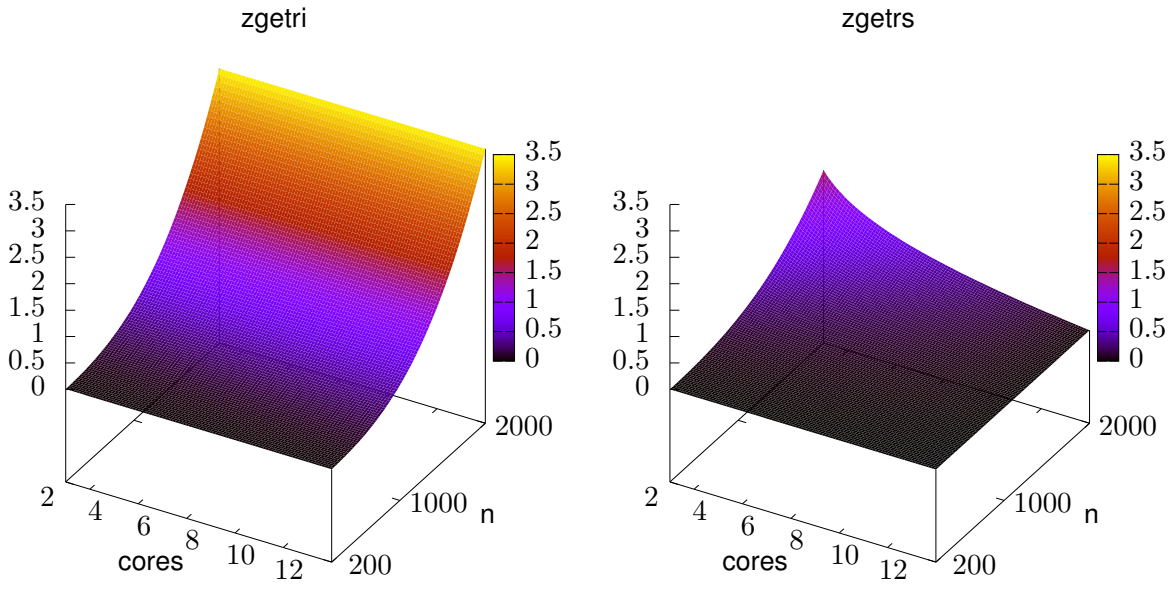


Figure 17: Performance Model for the two LAPACK function that could be used for the Inversion.

Input	Nodes	tasks p. n.	threads p. t.	time origin [sec]	time inversion [sec]	Speedup
2x2	1	1	48	1544.5	937.04	1.65
2x2	1	2	24	640.57	398.78	1.61
2x2	1	4	12	347.64	230.93	1.51
2x2	1	6	8	251.45	176.69	1.42
2x2	1	8	6	199.20	145.20	1.37
2x2	1	12	4	153.46	120.72	1.27
2x2	1	24	2	103.79	91.61	1.13
2x2	1	48	1	92.61	86.36	1.07

Table 20: Node-level sweetspot analysis of the original version and the optimized version of LIBNEGF one 1 JUWELS node for the 2x2 silicon test input comparing different combinations of MPI ranks and OpenMP threads.

the original source code as well as the version with the optimized inversion. We can see that the configuration of 48 MPI ranks and 1 OpenMP thread gives the best performance. This implies two things: the small input case has not that much computational work, which harms the node-level performance. We can also see that the improvement of the inversion is always faster compared

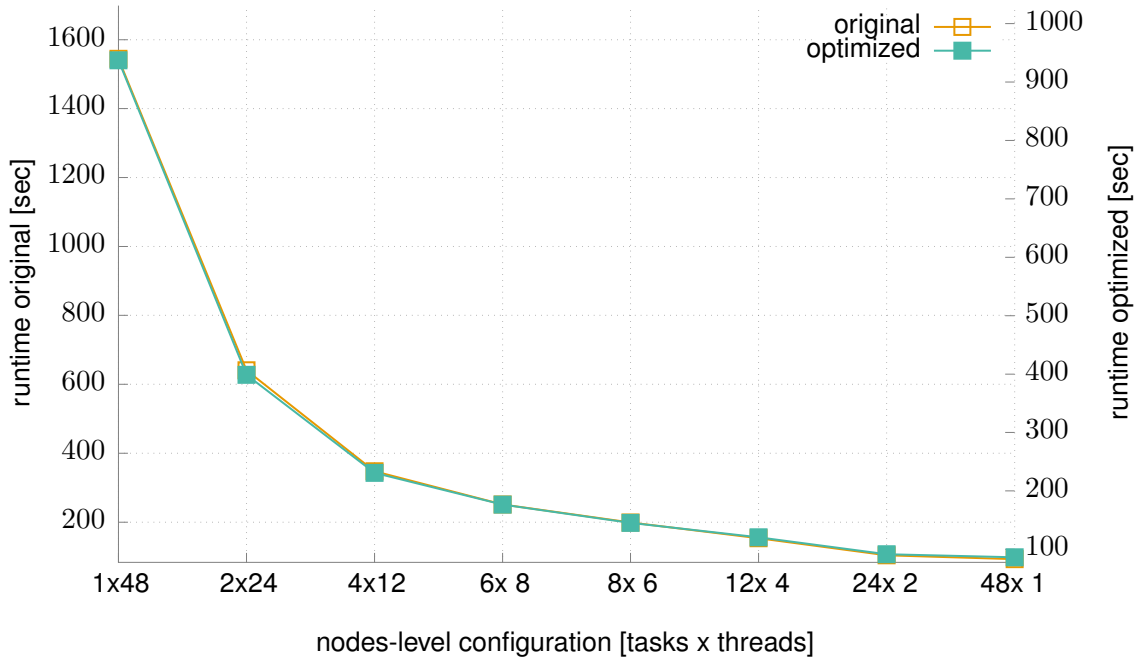


Figure 18: Node-level sweetspot analysis of the original version and the optimized version of LIBNEGF one 1 JUWELS node for the 2x2 silicon test input comparing different combinations of MPI ranks and OpenMP threads.

to the original version. In increasing speedup factor shows that the node-level performance of the optimized version is better compared to the original version.

Input	Nodes	tasks p. n.	threads p. t.	time origin [sec]	time inversion [sec]	Speedup
6x6	10	1	48	N/A[1]	13204.06	N/A
6x6	10	2	24	59847.20	6393.42	9.36
6x6	10	4	12	35895.52	4951.13	7.25
6x6	10	6	8	25822.66	4523.72	5.71
6x6	10	8	6	21210.44	4478.50	4.74
6x6	10	12	4	15502.92	4363.53	3.55

Table 21: Node-level sweetspot analysis of the original version and the optimized version of LIBNEGF one 10 JUWELS node for the 6x6 silicon test input comparing different combinations of MPI ranks and OpenMP threads.

Since the 2x2 input is quite small, we repeated the experiment with the 6x6 silicon supercell input. Table 21 and Figure 19 show the result of this experiment. Due to memory constrains we couldn't run the case with 24 and 48 MPI ranks per node, as each MPI rank required more memory. To be able to run this larger test, we used 10 nodes, and distributed all 10 K points across the nodes. The energy points were distributed when more MPI ranks per node were used. In that way the experiment is comparable to the one before. We can identify that the curve of the runtime is flattening out earlier. That implies that the node-level performance is better for a large input size (e.g the matrix size that is used on each MPI rank). Starting from around 4 MPI tasks per node, we can see almost similar timings. This shows that we should for this input size use at last 4 MPI tasks per node. However it also shows the there is further room for improvement regarding the node-level performance. In general we would like to use the large but still efficient number of OpenMP threads

[1] Max Walltime limit of 24h was exceeded, no measurement possible.

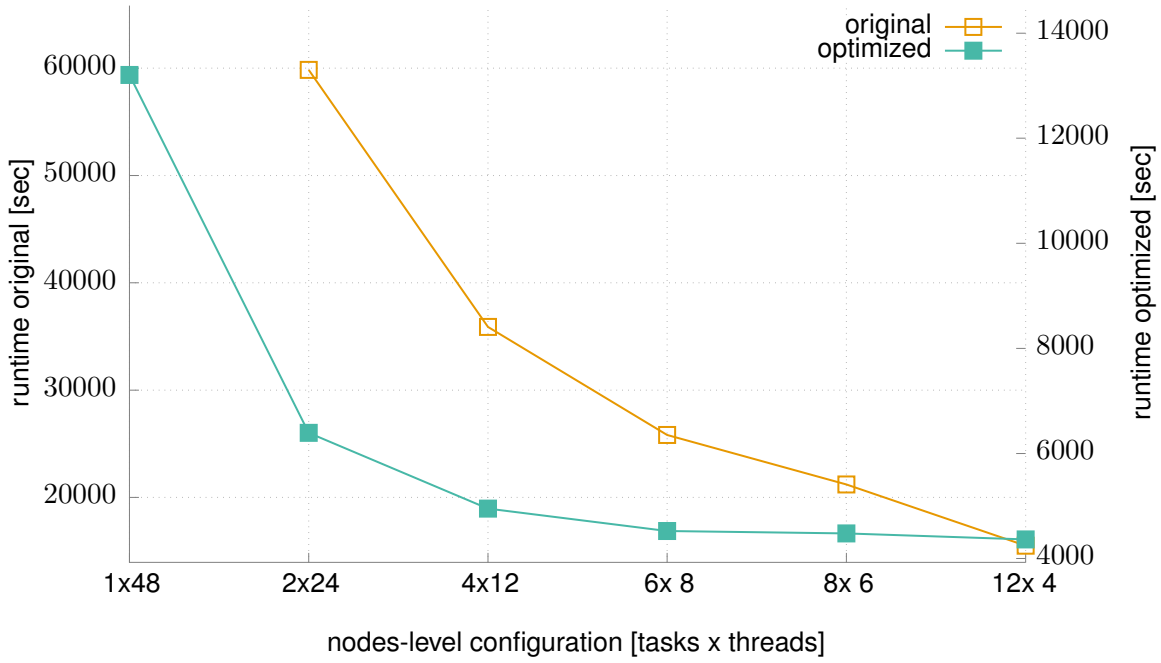


Figure 19: Node-level sweetspot analysis of the original version and the optimized version of LIBNEGF one 10 JUWELS node for the 6x6 silicon test input comparing different combinations of MPI ranks and OpenMP threads.

per tasks. This has two reasons: 1) This allows us to solver bigger problems efficiently. 2) More parallelism relaxes the memory constrains, which again allows bigger simulations.

Input	Nodes	tasks p. n.	threads p. t.	time origin [sec]	time inversion [sec]	Speedup
2x2	10	4	12	38.61	25.02	1.54
3x3	10	4	12	399.62	149.81	2.67
4x4	10	4	12	2190.27	615.98	3.56
5x5	10	4	12	9949.08	1836.85	5.42
6x6	10	4	12	33125.51	5015.48	6.60

Table 22: Comparison of the different supercell silicon test inputs for the original version and the optimized version on 10 JUWELS node with 4 MPI ranks per node and 12 OpenMP threads per rank.

Table 22 shows the a runtime comparison of the different input cases for the original and the optimized version of the code. For this experiment we have used 10 nodes and distributed the k points across these. 4 MPI ranks per node have been used to distribute the energy points. And we used 12 OpenMP threads per tasks. The interesting observation here is the Speedup between the two code versions. The result shows that with increasing problem size the speedup is increasing as well. We gained a speedup of up to 6.6x for the 6x6 input.

Table 23 and Figure 20 show the scaling behavior with the K parallelism. Therefor we increased the number of nodes and distributed the K points across the nodes. The focus of this test was on the MPI parallelization. Therefore we used the smallest input case. We can see that the efficiency of the parallelization is varying. Only good performance can be achieved when the number K points is dividable by the number of nodes, which make sense as the smallest unit of distribution is 1 K point. For the numbers that are dividable we achieve good performance.

Similar to the previous experiment, we distributed the energy points across nodes. Table 24 and

Branch	Input	Nodes	tasks p. n.	threads p. t.	time [sec]	Par. Efficiency
origin	2x2	1	4	12	348.32	1.00
origin	2x2	2	4	12	173.96	1.00
origin	2x2	4	4	12	105.46	0.83
origin	2x2	5	4	12	72.6	0.96
origin	2x2	6	4	12	72.93	0.80
origin	2x2	8	4	12	70.49	0.62
origin	2x2	10	4	12	38.25	0.91
inversion	2x2	1	4	12	230.13	1.00
inversion	2x2	2	4	12	115.9	0.99
inversion	2x2	4	4	12	69.45	0.83
inversion	2x2	5	4	12	48.02	0.96
inversion	2x2	6	4	12	48.21	0.80
inversion	2x2	8	4	12	47.18	0.61
inversion	2x2	10	4	12	25.28	0.91

Table 23: Scaling with K parallelism of the 2x2 silicon supercell test inputs for the original version and the optimized version on up to 10 JUWELS node with 4 MPI ranks per node and 12 OpenMP threads per rank.

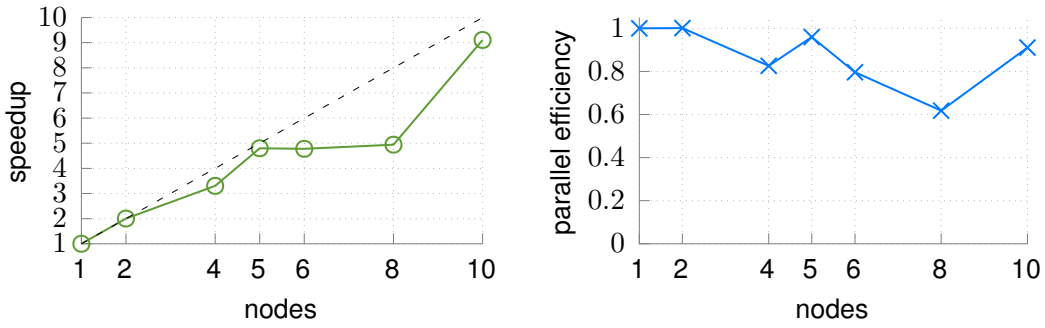


Figure 20: Scaling with K parallelism of the 2x2 silicon supercell test inputs of the optimized version on up to 10 JUWELS node with 4 MPI ranks per node and 12 OpenMP threads per rank.

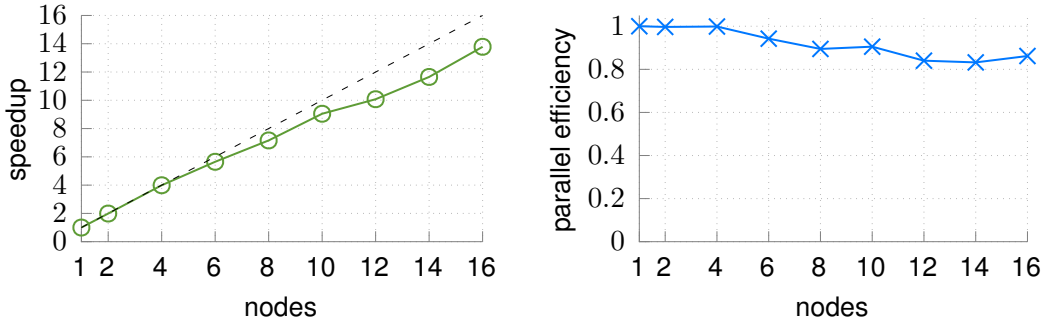


Figure 21: Scaling with E parallelism of the 2x2 silicon supercell test inputs of the optimized version on up to 16 JUWELS node with 4 MPI ranks per node and 12 OpenMP threads per rank.

Figure 21 show the scaling behavior with E parallelism. In this case we didn't distribute the K points, but only distribute energy point across the MPI ranks. The scaling of the E parallelism shows good performance. On 16 nodes and 4 tasks per node we can still measure a parallel efficiency of 86%.

In the Table 25 and Figure 22 we used the previous results and preformed a strong scaling exper-

Branch	Input	Nodes	tasks p. n.	threads p. t.	time [sec]	Par. Efficiency
origin	2x2	1	4	12	346.58	1.00
origin	2x2	2	4	12	173.86	1.00
origin	2x2	4	4	12	86.75	1.00
origin	2x2	6	4	12	61.29	0.94
origin	2x2	8	4	12	48.43	0.89
origin	2x2	10	4	12	38.3	0.90
origin	2x2	12	4	12	34.39	0.84
origin	2x2	14	4	12	29.75	0.83
origin	2x2	16	4	12	25.14	0.86
inversion	2x2	1	4	12	230.4	1.00
inversion	2x2	2	4	12	116.99	0.98
inversion	2x2	4	4	12	59.31	0.97
inversion	2x2	6	4	12	40.88	0.94
inversion	2x2	8	4	12	32.33	0.89
inversion	2x2	10	4	12	25.52	0.90
inversion	2x2	12	4	12	22.95	0.84
inversion	2x2	14	4	12	19.92	0.83
inversion	2x2	16	4	12	17.01	0.85

Table 24: Scaling with E parallelism of the 2x2 silicon supercell test inputs for the original version and the optimized version on up to 16 JUWELS node with 4 MPI ranks per node and 12 OpenMP threads per rank.

Branch	Input	Nodes	tasks p. n.	threads p. t.	time [sec]	Par. Efficiency
inversion	6x6	1	4	12	48008.77	1.00
inversion	6x6	2	4	12	23655.21	1.01
inversion	6x6	3	4	12	16068.58	1.00
inversion	6x6	5	4	12	9892.71	0.97
inversion	6x6	6	4	12	8122.23	0.99
inversion	6x6	10	4	12	5012.17	0.96
inversion	6x6	15	4	12	3270.75	0.98
inversion	6x6	25	4	12	2000.96	0.96
inversion	6x6	30	4	12	1664.67	0.96
inversion	6x6	50	4	12	999.52	0.96
inversion	6x6	75	4	12	676.82	0.95
inversion	6x6	125	4	12	415.41	0.92
inversion	6x6	150	4	12	354.94	0.90
inversion	6x6	250	4	12	214.3	0.90
inversion	6x6	375	4	12	167.26	0.77
inversion	6x6	750	4	12	92.64	0.69

Table 25: Scaling of the 6x6 silicon supercell test inputs of the optimized version of LIBNEGF on up to 750 JUWELS node with 4 MPI ranks per node and 12 OpenMP threads per rank.

iment with the largest 6x6 silicon input case. We used 4 MPI tasks per node, since this yields good node-level performance. We distributed as many K points as possible for the given number of overall tasks. Since the input case used 10 K points and 300 energy points, the maximum number of tasks is 3000. Since we used 4 tasks per node, the largest possible number of nodes is 750. In this case each tasks has just one K and one Energy point. The parallel efficiency is above 90% for up to 250 nodes. On 750 nodes we achieved 69% parallel efficiency.

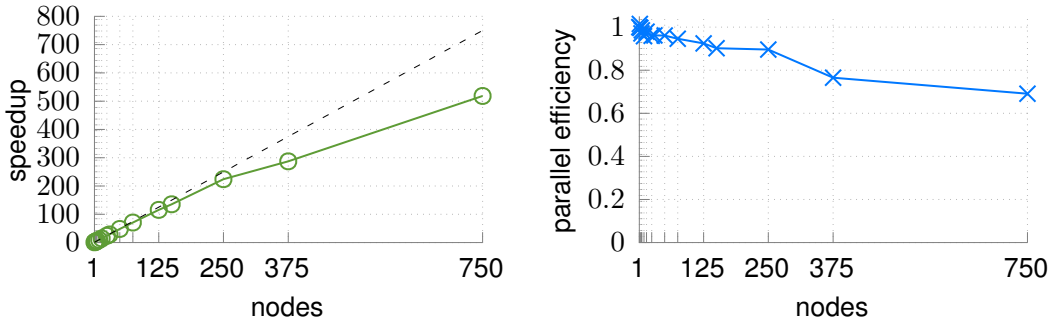


Figure 22: Scaling of the 6x6 silicon supercell test inputs of the optimized version of LIBNEGF on up to 750 JUWELS node with 4 MPI ranks per node and 12 OpenMP threads per rank.

7.3 Metrics definition and performance tools

To analyze the code performance we used the performance metrics developed within the EoCoE-I project. The definition of all global performance metrics is given in table 26. Several tools are used to extract them:

- The UNIX *time* command is used to measure total application wall time and the memory footprint of the first MPI rank of the application.
- Darshan[2] provides all metrics concerning IO
- Scalasca[3] provides all metrics concerning MPI, OpenMP and load balancing
- PAPI[4], used through Scalasca, provides all performance counters

Metrics Global.1, Global.2 and Global.3 might exhibit some inconsistencies as these three measures are extracted from three different runs performed with different binaries. This should not change the global picture as long as similar run times are observed for these three runs.

The MPI time (Global.3) is measured by Scalasca. But Scalasca will also measure MPIIO calls as part of the MPI time measurement, so this MPIIO time is subtracted from MPI time during the metric extraction process.

The IO time (Global.2) is measured by Darshan. The IO time itself within Darshan is separated into POSIX and MPIIO time. The POSIX IO handling is a subset of the MPIIO handling, so typically it would be enough just to use the MPIIO timings (if available) to represent the total IO time. Of course there are also applications which use MPIIO and POSIX file IO at the same time. In such a case the maximum of both will be selected to represent the IO time metric.

Memory vs Compute Bound metric (Global.4) is computed with the runtime coming out of two dedicated runs. The two runs use the same amount of MPI ranks and threads but on twice the number of nodes. This leads to depleted resources, and, by using specific deployments, one has the chance to observe memory bandwidth effects. Typically on current dual socket systems, a compact and a scatter run are performed. The compact run packs all the MPI processes and threads on a single socket, whereas the scatter run distributes them evenly on the two sockets. Going from the compact run to the scatter one, the available computing power is kept constant while doubling the available memory bandwidth. As a consequence, if both runs exhibit the same wall time, this means that the memory bandwidth available has no impact on the application. So the code is

[2] http://www.mcs.anl.gov/research/projects/darshan/
[3] http://www.scalasca.org/
[4] http://icl.cs.utk.edu/papi/

		Metric name	Definition	Tool
Global	1	Total Time (s)	Total application wall time	*time*
	2	Time IO (s)	Average time spent in doing IO for each process	Darshan
	3	Time MPI (s)	Average time spent in MPI for each process	Scalasca
	4	Memory vs Compute Bound	1.0 means strongly compute bound, 2.0 means strongly memory bound	cf text
	5	Load Imbalance	Ratio of the load imbalance overhead towards the critical path duration	Scalasca
IO	1	IO Volume (MB)	Total amount of data read and written	Darshan
	2	Calls (nb)	Total number of IO calls	Darshan
	3	Throughput (MB/s)	IO.1 / Global.2	Computed
	4	Individual IO Access (kB)	IO.1 / IO.2	Computed
MPI	1	P2P Calls (nb)	Average number of peer to peer communications per MPI rank	Scalasca
	2	P2P Calls (s)	Average time spent in peer to peer communications per MPI rank	Scalasca
	3	P2P Message Size (kB)	Average message size in peer to peer communications per MPI rank	Scalasca
	4	Collective Calls (nb)	Average number of collective communications per MPI rank	Scalasca
	5	Collective Calls (s)	Average time spent in collective communications per MPI rank	Scalasca
	6	Collective Message Size (kB)	Average message size in collective communications per MPI rank	Scalasca
	7	Synchro / Wait MPI (s)	Average time spent in synchronization per MPI rank	Scalasca
	8	Ratio Synchro / Wait MPI	MPI.7 / Global.3	Computed
Node	1	Time OpenMP (s)	Time spent in OpenMP parallel region	Scalasca
	2	Ratio OpenMP	Ratio of the time spent in OpenMP parallel region towards the total calculation time	Scalasca
	3	Time Synchro / Wait OpenMP	Average time spent in synchronization/OpenMP overhead per thread	Scalasca
	4	Ratio Synchro / Wait OpenMP	Node.4 / Node.1	Computed
Mem	1	Memory Footprint	Average memory footprint of an MPI process	ldrMem/ Slurm
	2	Cache Usage Intensity	Cache Hit / (Cache Hit + miss) in Last Level Cache	PAPI
Core	1	IPC	Total number of instructions executed / Total number of cycles	PAPI
	2	Runtime without vectorization	Total application wall time compiled with vectorization disabled	*time*
	3	Vectorisation efficiency	Global.1 / Core.2	Computed
	4	Runtime without FMA	Total application wall time when compiled with FMA disabled	*time*
	5	FMA efficiency	Global.1 / Core.4	Computed

Table 26: Global performance metrics definition

strongly compute bound and the ratio run time compact / run time scatter is 1.0. On the other hand, if the scatter run is twice as fast, the ratio is than 2.0 and this means that the code is strongly memory bound.

The load imbalance metric (Global.5) gives the potential for code improvement if the load imbalance would be perfectly fixed. Thanks to the trace analysis, Scalasca is able to compute the critical path of the application and the overhead due to load imbalances between ranks/threads. The metric used here is simply the ratio overhead / critical path. For instance, if a 20% load imbalance is measured, fixing perfectly this load imbalance would improve the performance of the code by 20%.

Synchro / Wait MPI (MPI.7) is calculated by gathering the communication overhead except the pure communication time. This metric sums up the average waiting time per process (e.g. because of a MPI barrier operation) and the synchronisation time to start collective operations.

Metrics Mem.2 and Core.1 use the PAPI counter interface. The implementation of this interface and the available metrics are highly platform specific. Because of that not all applications might allow the extraction of these two metrics.

7.4 Automated metrics extraction process

The generation of the binaries as well as the execution of all necessary runs to generate the metric overview has been automated by using the JUBE environment. Specific metrics as well as a full metric overview can be created with a single JUBE execution.

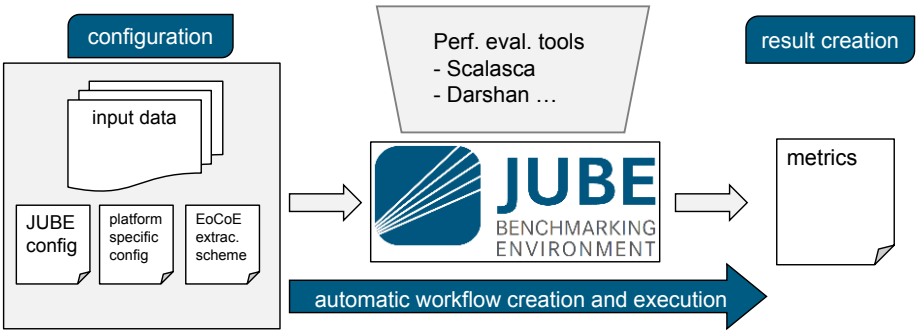


Figure 23: General JUBE workflow for the EoCoE metric extraction process.

Figure 23 shows the main workflow by using the JUBE environment. The application build and run procedure is included into a JUBE configuration file. This part is application specific. Platform specific configuration datasets and the EoCoE specific execution scheme is added together with the relevant input data for the different benchmarking cases of the application. Within the JUBE environment, different runs are performed as written below. Different metric extraction tools like Scalasca and Darshan are called from within the JUBE environment. The final outcome of the execution is the set of metrics as shown in table 26.

Specifically, for the purpose of automation four separate code binaries are initially needed:

- Normal (ref)
- scalasca instrumented (scalasca)
- Normal plus "no-vectorization" (no-vec)
- Normal plus "no-fma" (no-fma)

If needed a separate executable could be created for the Darshan or the memory instrumentation.

Next, 9 runs are performed:

1. ref ⇒ reference run
2. ref ⇒ memory footprint run
3. ref + Darshan ⇒ IO metrics
4. scalasca profile run ⇒ CPU counters
5. scalasca trace analyse ⇒ Global, MPI, OMP
6. (no-vec) ⇒ Core, vectorization efficiency
7. (no-fma) ⇒ Core, FMA efficiency
8. ref compact run ⇒ mem vs comp. bound
9. ref scatter run ⇒ mem vs comp. bound

The dependencies between the different runs are also shown in Figure 24.

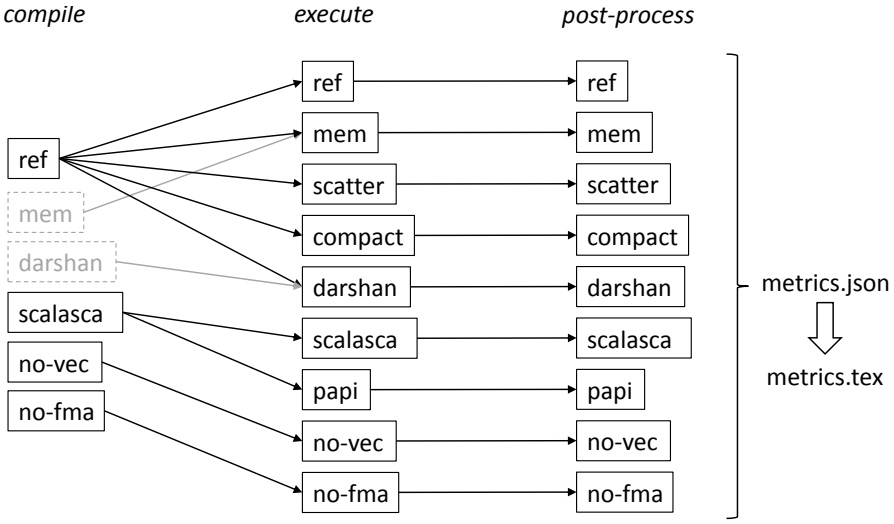


Figure 24: Steps in the automated JUBE workflow for the EoCoE metric extraction process.

All metrics paths could also be executed separately if needed.

7.4.1 Performance Metrics

Table 27 show the result of the performance metrics for LIBNEGF.

7.5 Risks, warning points and mitigation

The main risks, their impact and relative mitigation strategies are described in Table 28. As described in Sec. 7.1, this task, as well as the corresponding subtask T1.3.1-3 of WP1, had to be planned anew after it was decided to change flagship code from PVnegf to libNEGF. At the same time, human resources were either unavailable or lost in the course of the change. These losses and changes have been replaced with new committed participants to the team of developers and scientists. Overall, despite the delays, the task in on the right path to achieve its goal by the end of the project.

	Metric name	Original	Inversion
Global	Total Time (s)	203	176
	Time IO (s)	0.08	0.02
	Time MPI (s)	0.59	0.65
	Memory vs Compute Bound	1.00	1.01
	Load Imbalance (%)	83.41	83.41
IO	IO Volume (MB)	0.95	0.95
	Calls (nb)	3994	3994
	Throughput (MB/s)	11.18	46.52
	Individual IO Access (kB)	0.28	0.28
MPI	P2P Calls (nb)	0	0
	P2P Calls (s)	0.00	0.00
	P2P Calls Message Size (kB)	0	0
	Collective Calls (nb)	3	3
	Collective Calls (s)	0.48	0.53
	Coll. Calls Message Size (kB)	1237	1237
	Synchro / Wait MPI (s)	0.00	0.00
	Ratio Synchro / Wait MPI (%)	0.05	0.03
Node	Time OpenMP (s)	33.89	28.98
	Ratio OpenMP (%)	16.67	16.67
	Synchro / Wait OpenMP (s)	0.01	0.01
	Ratio Synchro / Wait OpenMP (%)	29.37	29.85
Mem	Memory Footprint	361952kB	361604kB
	Cache Usage Intensity	0.78	0.55
Core	IPC	1.88	1.85
	Runtime without vectorisation (s)	202	173
	Vectorisation speedup factor	1.00	0.98
	Runtime without FMA (s)	201	145
	FMA speedup factor	0.99	0.82

Table 27: Performance metrics for LIBNEGF measured on JUWELS with 1 node running 8 MPI ranks and 6 OpenMP threads each.

Risks / Warning points	Impact	Mitigation
Main developer of PVnegf (Dr. Aeberhard) moved to industry	Loss of Knowledge and Expertise. The lack of scientific lead was among the main reasons to change flagship code	Substitution of PVNEGF with LIBNEGF and involvement of LIBNEGF developers as expert
Maternity leave of Dr. Aguilera, main contributor to the development of WP1 functionalities	Optimization tasks in WP2 depended on development in WP1	The order of tasks in WP1 and WP2 has been changed: Optimization of the current version of LIBNEGF has been prioritized, Optimization of new functionalities dealing with non-ballistic scattering will be implemented at a later stage.
Loss of Dr. Baumeister, one of the main developer to the 2.4 task of WP2	Delays in the progress of task 2.4 by approximately 3 months	Search for new personnel is ongoing. A shared resource could be secured by Fall 2020. In the meantime some work has been reassigned to Sebastian Achilles, currently part of the task 2.4.
Shutdown of all PRACE supercomputer due to IT security incident. JSC machines were offline for 1 months.	Delays in the development of HPC optimization, profiling of the code and benchmarking	Development switched to laptops as far as possible, applying for compute time call on other supercomputer (CLAIX which was not affected by the security incident and allows continuous submission of proposal) and getting all team member familiar with the different environment on other supercomputer.
Decreased efficiency and productivity due to COVID-19 pandemic and subsequent lockdown during the last months.	Delay of progress	Weekly video confcall to compensate for the usual in-person meetings.

Table 28: Risk management in Task 2.4 for Materials.

8 Task 2.5 - Hydrology code optimization

8.1 Task overview

Task leader : FZJ

Participants: FZJ, FAU, CEA, RWTH

The goal of the hydrology scientific challenge is to enable high-resolution (down to 100m) continental-scale hydrological simulations with mixture of active and inactive regions to make prediction of hydropower supply more accurate.

Two flagship codes are concerned by the WP2 technical challenge: ParFlow and SHEMAT-Suite. They are respectively and briefly describe in the following sections 8.1.1 and 8.1.2.

The work in this code is divided into 3 different subtasks:

- Task 2.5.1 - ParFlow code optimization
- Task 2.5.2 - SHEMAT-Suite code optimization
- Task 2.5.3 - Unified platform to unify the physics of both code

The detailed content of these tasks and the progress achieved so far is described in sections 8.2, 8.3, 8.4.

The risk management for the second part of our project concerning the hydrology scientific challenge is presented in 8.5.

8.1.1 Flagship code ParFlow

ParFlow is a parallel, integrated hydrologic model, which simulates surface and subsurface flow (ParFlow website). It is based on the shallow water equations coupled with the three dimensional Richard's equation. The code provides a solver for the latter based on a cell-centered finite difference scheme on regular Cartesian meshes. Time integration is performed with an implicit Euler method. The resulting system of nonlinear algebraic equations is solved by a multigrid-preconditioned Newton-Krylov method.

Table 29 shows the team members of ParFlow involved in EoCoE. Stefan Kollet is the Scientific Leader of the hydrology scientific challenge. He is as well the scientific coordinator of the ParFlow code. Jose A. Fonseca is a postdoctoral fellow in computer science and mathematics modeling at MdlS, CEA. He is working on AMR aspects in ParFlow. Jaro Hokkanen is a postdoctoral fellow in computer science at FZJ. He is focusing on porting and optimizing ParFlow on GPU. Mathieu Lobet is coordinating the ParFlow at MdlS and is working on PDI aspects in this code.

People	Position	Role	Period
Stefan Kollet, PhD	FZJ	Scientific coordinator	M1-M36
Bibi Naz, PhD	FZJ	PDI aspects	M1-M16
Jose A. Fonseca, PhD	Postdoc at MdlS, CEA	HPC expert for AMR aspects of ParFlow	M5-M29
Jaro Hokkanen, PhD	Postdoc at FZJ	Computer Scientist on code optimization and GPU aspects	M9-M33
Mathieu Lobet, PhD	Research-engineer at MdlS, CEA	coordinator and PDI aspects	M1-M36

Table 29: Team Members for ParFlow within EoCoE.

8.1.2 Flagship code SHEMAT-Suite

The SHEMAT-Suite is a code for simulating single- or multi-phase heat and mass transport in porous media (SHEMAT-Suite code repository). It solves coupled problems including heat transfer, fluid flow, and species transport. SHEMAT-Suite can be applied to a range of hydrothermal or hydrogeological problems, be it forward or inverse problems.

Table 30 shows the team members of SHEMAT-Suite involved in EoCoE. Johanna Bruckmann, Research Associate at RWTH Aachen University, is coordinating the SHEMAT-Suite activities in the different WPs of EoCoE and is responsible for the scientific challenge related to SHEMAT-Suite. Berenice Vallier has been recruited as a postdoctoral fellow to work on SHEMAT-Suite related tasks in WP2, WP3 and WP4.

People	Position	Role	Period
Johanna Bruckmann, M Sc	Research Associate at RWTH Aachen University	SHEMAT-Suite coordinator; scientist for WP1	M1-M36
Berenice Vallier, PhD	postdoctoral fellow at RWTH Aachen University	PDI implementation and PDAF	M10-M23

Table 30: Team Members for SHEMAT-Suite within EoCoE.

8.2 Work progress on task 2.5.1

Subtask 2.5.1 corresponds to the ParFlow code optimization. The programming model followed by ParFlow's developers is described in Fig. 25.

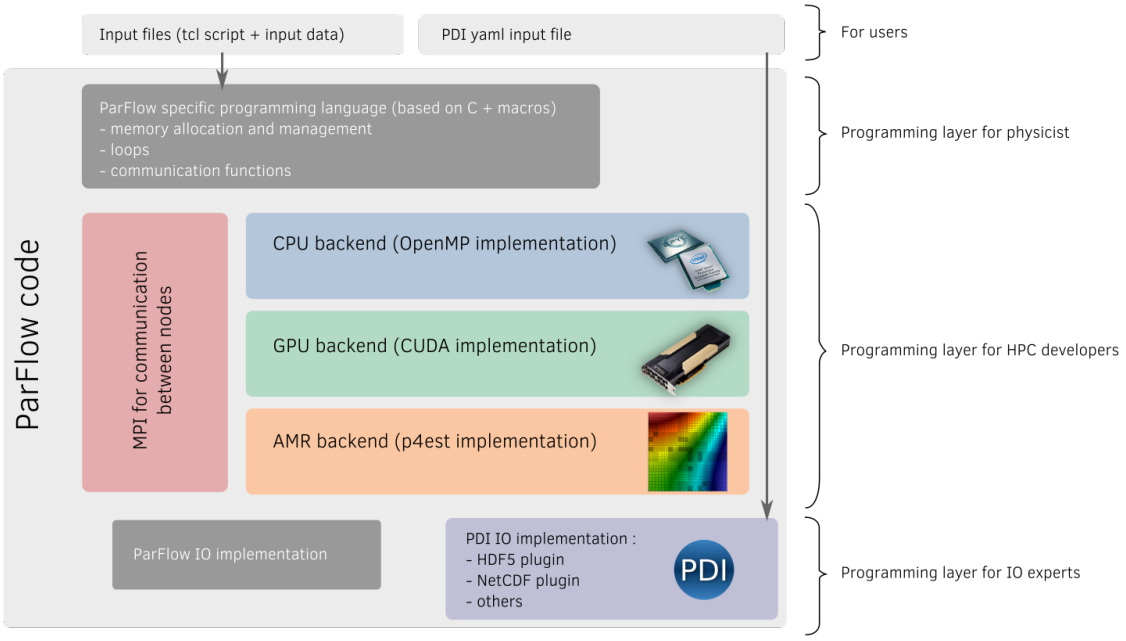


Figure 25: Description of the ParFlow programming model.

ParFlow is being developed and optimized in two different directions:

- At the beginning of the project, ParFlow was only working on CPU architectures and discretizing the computational domain with a uniform structured grid. The first axis consists in improving the CPU implementation by adding an Adaptive Mesh Refinement (AMR) module so that simulation with non-uniform grid becomes possible. The main interest in using AMR is the possibility to use a wide range of different spatial resolutions at a reduced computational cost. With the upstream version of ParFlow, the minimum required spatial resolution is used to solve the whole domain with the same accuracy even where a lower resolution would be sufficient. AMR enables multiple scales in the same simulation saving computational resources for high-resolution regions. Furthermore, the spatial discretization can be refined dynamically to adapt in time to the physical parameter evolution. A former project leaded by Carsten Burstedde at the University of Bonn and Stefan Kollet at FZJ, funded by the German Research Foundation (DFG) has led to the integration of ParFlow and the AMR library p4est without explicitly exploiting the AMR capabilities provided by this library. The current work extends the existing ParFlow 's integration with p4est by using the AMR routines of the latter and allowing the use of locally refined meshes. The progress is described in section 8.2.3. Such task requires several changes in the ParFlow core code that we have divided into the multiple subtasks described in the D2.1 and updated in the corresponding updated work plan displayed in Fig. 30.
- The second optimization axis is the GPU porting of the code. This work is currently handled at FZJ by Jaro Hokkanen. The progress is presented in section 8.2.2. The work plan for GPU adaptation is presented in Fig. 27.

These two axis toward the modernization of the code are performed independently so far and should be merged at the end of the project. This will be partly the subject of subtask 2.5.3. In addition

to the code optimization, WP2 is involved in the PDI implementation in ParFlow. This work is described in section 8.2.1.

8.2.1 PDI implementation

A first PDI implementation has been done in ParFlow that will be certainly improved after more intensive tests. ParFlow currently has three solvers implemented. Each solver use its own physical parameters and therefore has a specific output process. However, each solver calls the same generic output functions. All of them have been updated for PDI.

The ParFlow output code structure is shown in Fig. 26. It includes how PDI is plugged. They are three kinds of output format that can be used: ParFlow binary files (PFB), SILO, NetCDF. The PFB format is a home-made binary type of output. Each format has its own generic functions used by the solvers. Output parameters and period can be controlled in a similar way via the tcl input script. PDI has been implemented as a 4th possible output format for each solver. It therefore respects the same formalism and the same implementation structure. Similarly, PDI options in the tcl input script are similar to what has been done for other output formats. Currently, parameters available via PDI are the same as for the PFB format.

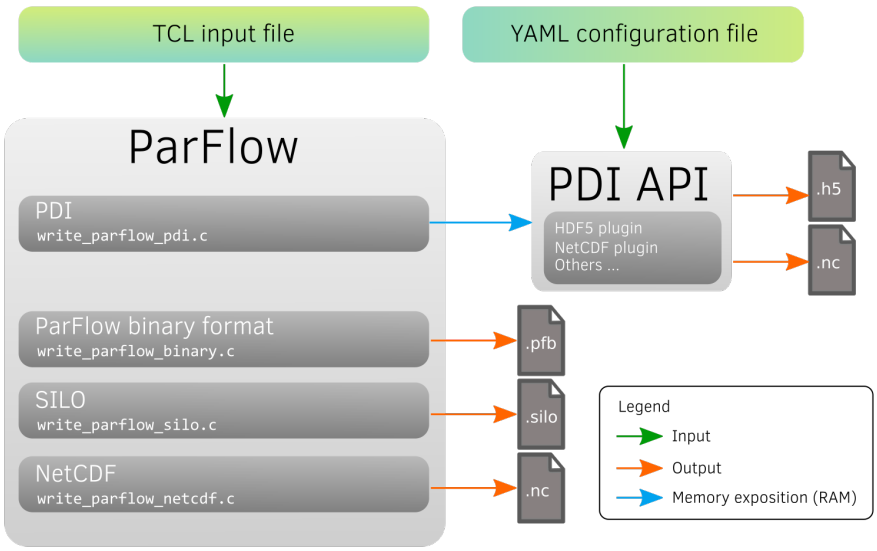


Figure 26: PDI integration in ParFlow.

An advantage of PDI in the future is the possibility to make all output options in ParFlow uniform. Indeed, all physical quantities cannot be written on disk in all formats. Some of them are only available with a specific format. Thanks to PDI, this limitation would be solved easily. Once a physical quantity can be exposed to PDI, it can then be written using any PDI plugin.

Contrary to the format currently used by ParFlow that requires a pre-process of the data (for instance, for the PFB files, the content of the physical data vectors is treated), PDI uses the full data structures. The treatment of the data is let to PDI and depends both on the YAML configuration file and the plugin. The YAML is now provided with the code.

The PDI implementation has been validated using the `default_single` and the `default_richards` test cases. In our tests, we have used the HDF5 plugin. We have developed a Python script that compares the data on disk in `.pfb` files and `.h5` files.

The ParFlow version with PDI is not officially released and is only accessible within the project. When this version will be validated with WP4 and PDI experts, this new feature will be proposed

to ParFlow's developers for integration in the master version of GitHub.

8.2.2 GPU porting

The work done on the GPU porting of ParFlow is described in this section. The developments started at M8 when Jaro Hokkanen was hired at FZJ. As shown in Fig. 27, we have subdivided the work into micro-tasks. ParFlow is now fully functional on GPU. The new version is now integrated to the master one. The following describes the integration details and performance results.

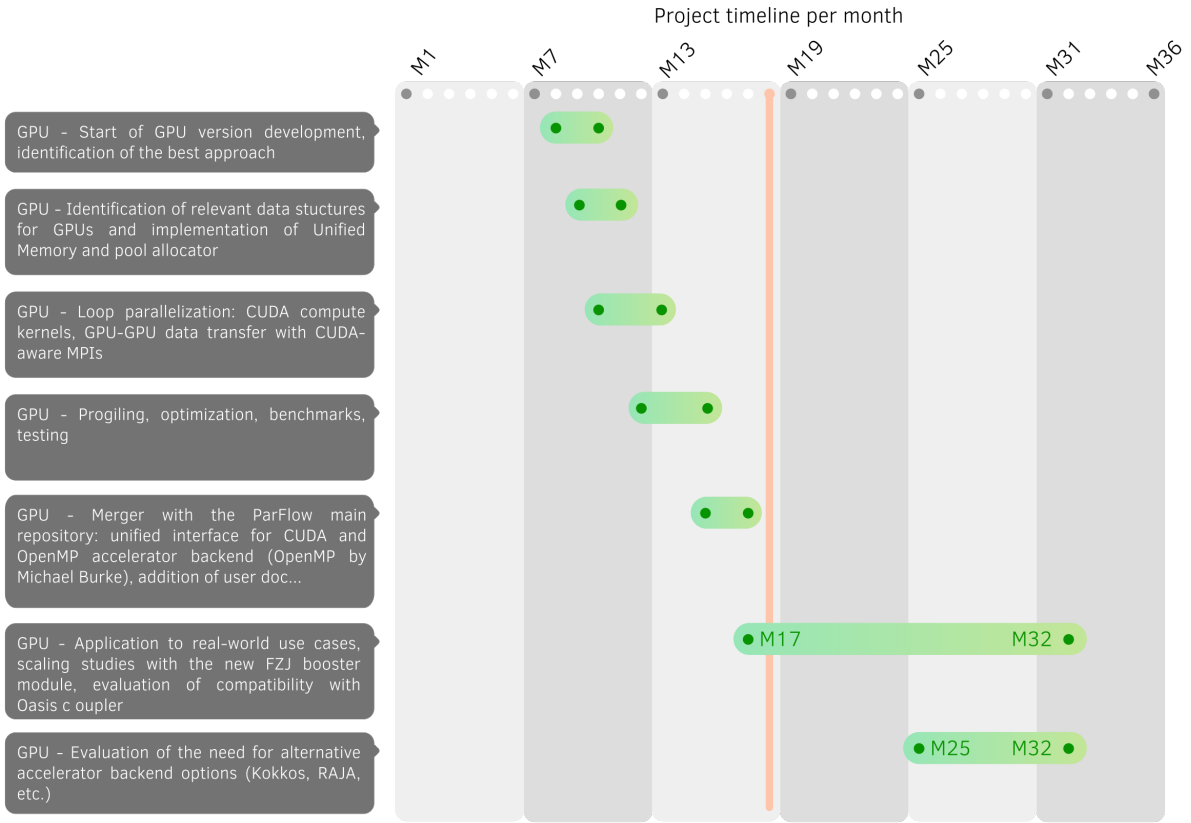


Figure 27: Breakdown (simplified Gantt chart) of the task 2.5.1 concerning GPU in ParFlow.

The GPU acceleration is built directly into the ParFlow embedded domain-specific language (ParFlow eDSL) headers such that, ideally, parallelizing all loops in a single source file requires only a new header file. This is possible because the ParFlow eDSL provides an interface for looping, allocating memory, and accessing data structures. The decision to embed GPU acceleration directly into the eDSL layer resulted in a highly productive and minimally invasive implementation.

The first accelerator backend supporting GPUs is based on CUDA. Features provided by CUDA C++ such as Unified Memory (with a pool allocator) and host-device lambdas were extensively leveraged in the ParFlow implementation in order to maximize productivity and codebase maintainability in the long-term. Efficient intra- and inter-node data transfer between GPUs rests on a CUDA-aware MPI library and newly developed application side GPU-based data packing routines.

The current, moderately optimized ParFlow GPU version runs a representative model up to 24 times faster on a node with 2 Intel Skylake processors and 4 NVIDIA V100 GPUs compared to the original version of ParFlow, where the GPUs are not used (see Figure 28). Furthermore, Figure 29 shows the weak scaling behavior for the same benchmark problem. The relative performance

multiple appears to approach 15-16 when the number of nodes is increased suggesting good scaling across multiple nodes.

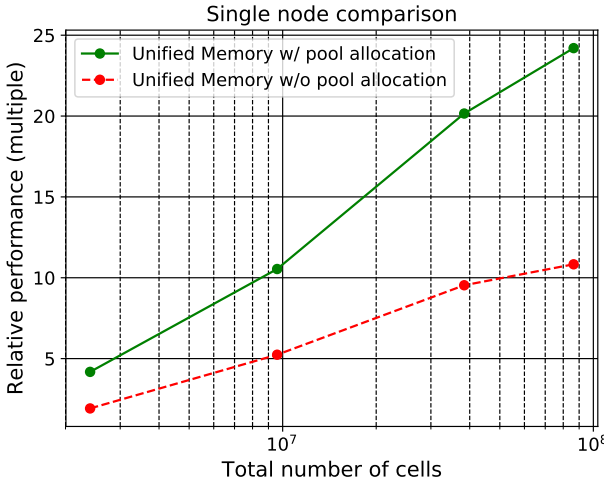


Figure 28: Single node performance comparison.

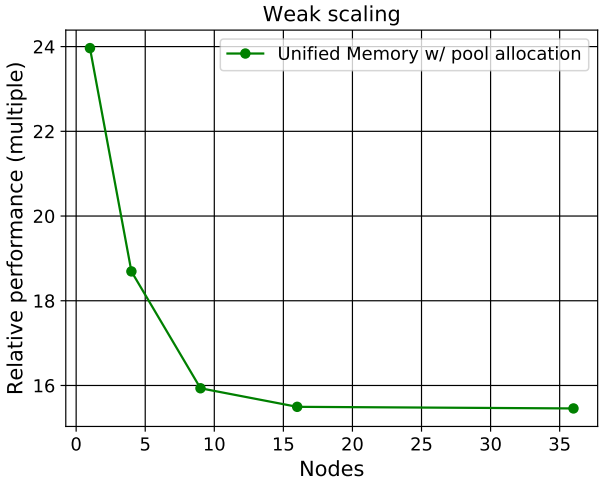


Figure 29: Weak scaling comparison.

The future work involves the application of GPU capabilities to real-world problems and potentially leveraging GPU support for more features, such as alternative preconditioners. The GPU usage with Oasis coupler for TerrSysMP and the need for additional accelerator backends to support more architectures are also evaluated. Overall, the GPU accelerator architectures enable faster simulations for larger models.

8.2.3 AMR implementation

Converting a mature and a complex code like ParFlow to AMR is a challenging task because two fundamental parts of the code are constructed under the assumption of a uniform mesh: first, the way in which information is communicated between processes, i.e., the parallel partition and second, the mathematical operators that represent the underlying PDEs the code aims to solve. We have identified several tasks to follow in order to solve these challenges and summarized them in Figure 30. In the rest of this subsection we discuss the progress done in each of these steps.

We have completed the first task in table 30. As in many codes based on finite difference discretizations, ParFlow stores an additional strip of degrees of freedom at the boundary of each process in order to perform parallel updates, see Figure 31(a). If a locally (2:1 balanced) refined mesh is enforced, we need to provide additional storage for the situation in which a parallel update of information occurs between two or more different size subgrids, see Figure 31(b).

We are currently working on number two in table 30. This task has been changed with respect to the previous report. Previously, our aim was to replace ParFlow's native discretization for a mixed finite element one. Nevertheless, taking this approach proved to translate disruptive modifications in the code that we have not envisioned before. We chose instead, to follow an approach similar to [31]. In this work, the authors observed that in order to obtain a globally second order discretization of the Laplacian, one may use discretizations that are only first order accurate at locally refined points but reduce to second order accurate when applied at locally uniform points.

Employing finite differences on a (2:1 balanced) locally refined mesh will require values of the function to differentiate at locations where such values are not available. Using linear interpolation to derive the required missing values will add new terms in the corresponding Taylor analysis that

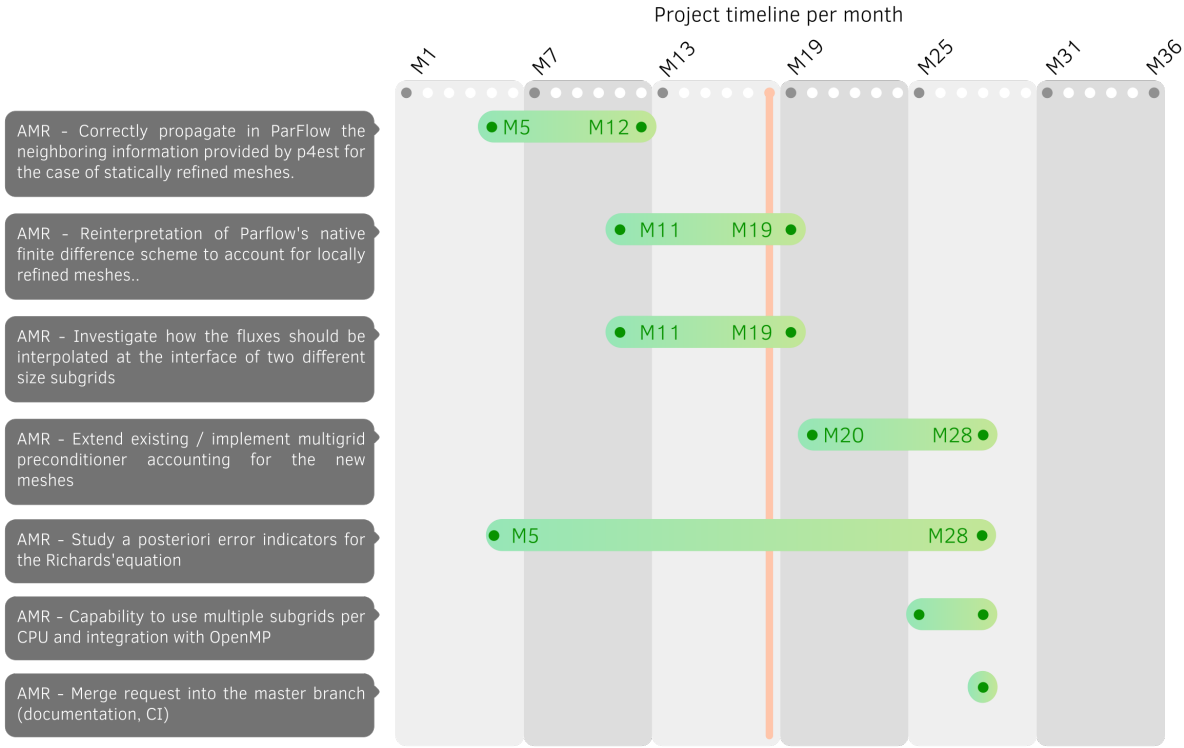


Figure 30: Breakdown (simplified Gantt chart) of the task 2.5.1 concerning AMR in ParFlow.

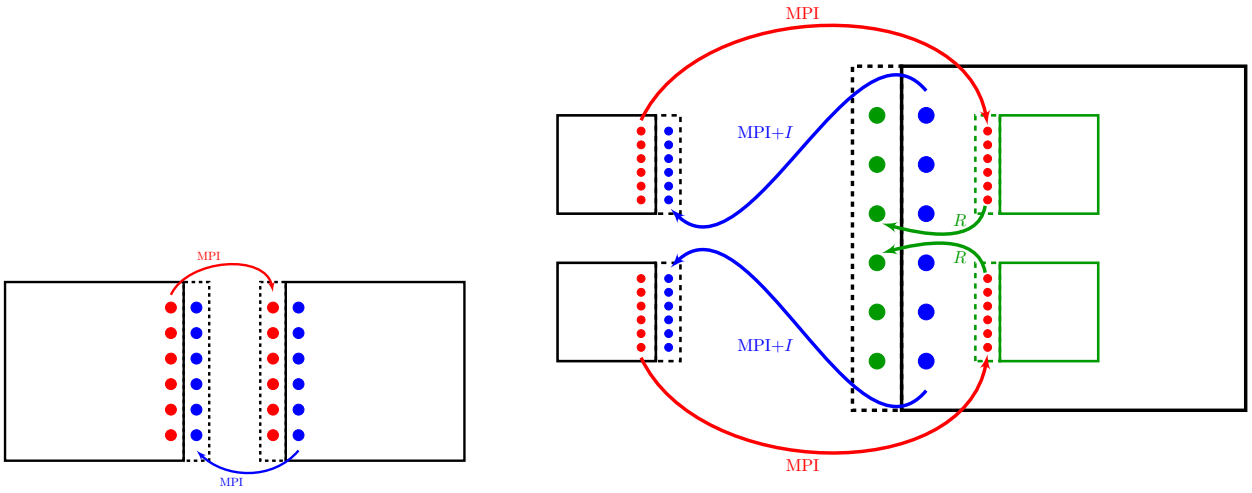
may degrade the accuracy of the approximation compared to the uniform case. The idea proposed in [31], is that it is possible to tweak the approximations to regain the accuracy one should obtain in the uniform case. See Figure 32.

With this approach we are already able to run simple test cases, for example, by appropriately choosing the parameters appearing in the Richard's equation we can reduce it to a Poisson equation and thus, use ParFlow to solve the later. In Figure 33 we display an example of a numerical solution obtained in this way.

Please note, that the task number four in Table 30 has been slightly modified. The previously mentioned change in task two, implies that the envisioned SPAMG preconditioner won't be a suitable choice since it is explicitly build for mixed finite element discretizations. Hence, we aim for extending the existing multigrid preconditioner.

Simulations of large physical cases will take place in the second part of the project starting around month 26. We envision idealized simulations displaying discontinuities in the conductivity tensor which typically require high mesh resolutions to be resolved correctly. Such cases are encounter with frequency in practical applications and offer a perfect example in which the benefit of a locally refined mesh becomes clear: less degrees of freedom to solve Richards's equation with at least the same accuracy as if a fine uniform mesh resolution is imposed in the whole domain.

The work on the AMR will not be finalized before month 25. A first step will be to merge the OpenMP capabilities developed in the master branch of ParFlow with the multiple-subgrid functionalities developed in the AMR branch. In the master branch, each MPI rank is currently limited to hold a single subgrid. The AMR developments unlock this limitation by allowing multiple of subgrids per MPI rank. Hence, a combination of both features adds flexibility to the user regarding the amount of computational work to be carried by a single tread. This might be highly beneficial to further investigate threading optimization in a hybrid MPI/OpenMP environment and enlarges the class of architectures that the code may take advantage.



(a) (b)

Figure 31: Left, default communication pattern of a stencil propagating information in the x–coordinate direction. The values that need to be exchanged are displayed in red and blue dots and the ghost layer where these are written to is enclosed in dotted lines. Right, schematic representation of the approach taken to propagate numerical information for a locally refined mesh in ParFlow. We impose a 2:1 balance condition on the mesh such that this is the only relevant case to treat. A coarse subgrid requires to share information with two neighboring finer subgrids. We create additional ghost subgrids internal to the coarse one, in the figure displayed in green. We conveniently call them "inner ghost subgrids". The arrows clarify the flow of information, MPI denotes communication, I interpolation and R restriction.

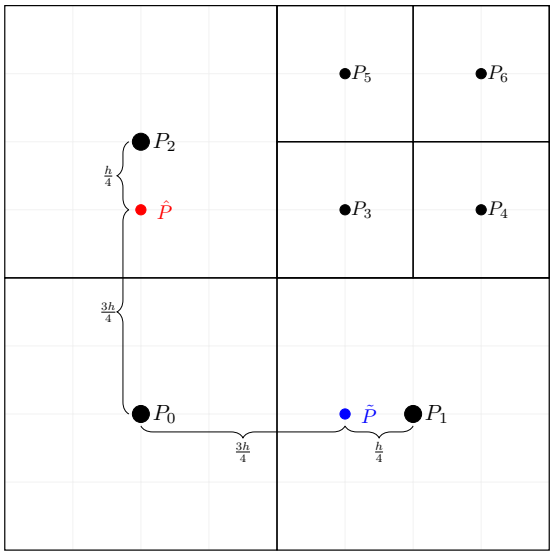
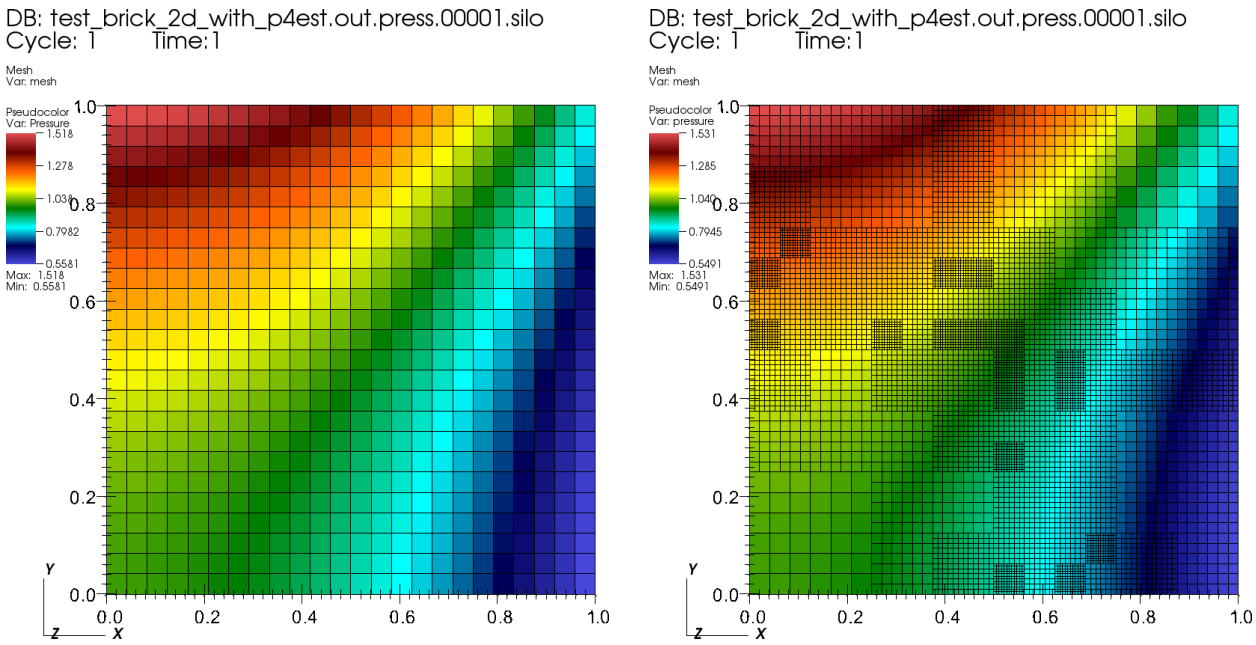


Figure 32: We wish to approximate the Laplacian at P_3 using a standard 5 point stencil using the values at $P_3, \hat{P}, P_4, \tilde{P}$ and P_5. The function values at $\hat{P}$ and $\tilde{P}$ are obtained by linear interpolation using the available data. This will yield an approximation of the Laplacian at P_3 of the form $L_x + L_y$, which lacks the order of accuracy one would obtain in a uniform mesh. The idea in [31] is that it is possible to pick an adequate weighting $aL_x + bL_y$, for some constants a and b to recover the desired accuracy. For this particular example one can choose $a = b = 5/6$.

Once the AMR has been tested and validated on a large scale, the final step will be to merge this work with the main branch. This is the last subtask in Fig. 30.



(a) (b)

Figure 33: Approximate solution of the Poisson equation on the unit square. The right hand side and boundary conditions are selected such that $p(x, y) = \cos(x)\cosh(y)$ is the analytical solution. Left, we show the solution computed by ParFlow on uniform mesh with 24 cells peer coordinate direction. Right, we display ParFlow's computed solution on a randomly refined mesh, allowing up to three levels of refinement with respect to the uniform case.

8.3 Work progress on task 2.5.2

Subtask 2.5.2 is dedicated to the SHEMAT-Suite application. The task goal is to improve code performance for ensemble runs (see WP5) by integrating PDI and the Parallel Data Assimilation Framework (PDAF). Ensemble runs will be used for stochastic parameter estimation and uncertainty quantification within geothermal reservoirs. Berenice Vallier has been hired for 12 months to complete the PDI and PDAF activities in WP2 and WP4.

The work on the PDI integration began in November 2019 (M11). The first steps for the integration such as the installation on the RWTH compute cluster CLAIX-18 were finalized; the PDI integration is ongoing. PDAF integration has not started yet. This deliverable focuses on the integration of PDI into the SHEMAT-Suite and once this will be finalized, our main interest will be the integration of PDAF into SHEMAT-Suite. PDAF provides parallel data assimilation approaches, such as the Enseble Kalman Filter (EnKF) and will thus improve the performance of stochastic inversion and data assimilation in SHEMAT-Suite. The PDAF integration is closely related to our tasks in WP4.

8.3.1 Integration of PDI into SHEMAT-Suite

Many I/O processes are associated to scientific simulations such as in the SHEMAT-Suite software. Some are enhanced before the actual simulation, such as the initialization of data. Others occur during the simulation, such as the intermediate or final checkpoints to account for execution failures. Finally, after simulation, post-processing, diagnostics, storage to the disk and visualization of the results are additional I/O processes associated with SHEMAT-Suite. All these I/O processes can be managed individually thanks to libraries like HDF5, MPI I/O.

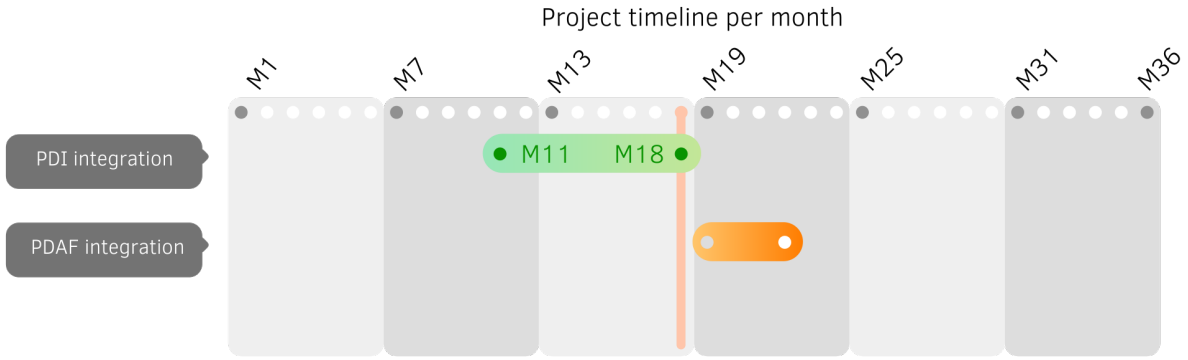


Figure 34: Breakdown (simplified Gantt chart) of the task 2.5.2 for SHEMAT-Suite.

By integrating PDI into SHEMAT-Suite, we aim to minimize the changes required in SHEMAT-Suite along the I/O processes. The main goals are to: (i) increase efficiency of the I/O processes; (ii) decouple I/O from the simulation; (iii) facilitate the usage of different I/O libraries; (iv) integrate PDI into SHEMAT-Suite will enable to make use of functionalities like in-situ visualization or big data and ensemble handling in the future.

First of all, the preliminary work of the PDI integration has been the installation of PDI on the RWTH cluster CLAIX-18 and the realization of the tutorial explained on the official website of PDI [9]. Thanks to the help of the developers of PDI, Julien Bigot and Karol Sierocinski, the installation and the tutorial have been conducted successfully. A documentation of the preliminary steps as well as the integration is written in parallel of the task 2.5.2 as guidelines for the future users of PDI in SHEMAT-Suite.

The implementation of PDI mainly focuses on a declarative API, the few changes required in SHEMAT-Suite code itself. Indeed, we define a unique Yaml file called specification tree supporting the calling of libraries. Each library call described in the specification tree relies on: (i) Data storages for data transfer referring to the list of parameters allowing this transfer. (ii) Event subsystem for control transfer called by example when a new parameter is made available in the store.

The typical structure of the specification tree is described in Fig. 35. The `data` and `metadata` sections specify the type of the data in buffers exposed by the application. For `metadata`, PDI keeps a copy while it only keeps references for `data`. The `plugin` section specifies the list of plugins to load and their configuration.

The code annotation API is the main interface to use in the SHEMAT-Suite source code. The initialization of PDI is called by the `PDI_init` function, the configuration file is parsed and the `decl'H5` plugin is loaded. This plugin initialization function is called and analyzes its part of the configuration to identify the events to which it should react. For the finalization, the `PDI_finalize()` function is releasing all resources at the end of the simulation. Exposing and reclaiming data to PDI are called by the `PDI_share()`, and `PDI_reclaim()` or `PDI_expose()`. The `PDI_event()` function is a PDI notification in a specific location in SHEMAT-Suite, the plugins are then reacting to the event. For integrating PDI into SHEMAT-Suite, the subroutines dealing with I/O processes have been identified in SHEMAT-Suite. More details about the SHEMAT-Suite can be obtained in the references works of [11] and [36]. Fig. 36 lists the SHEMAT subroutines related to I/O processes. By example, the main SHEMAT-Suite subroutine containing the functions for reading the input files is called `read_model`. All the keywords referencing the readable parameters are included as metadata and the data are the arrays such as the temperature, pressure, head and concentration. The reading of the keywords should be done in the right order because some parameters depend on previous ones.

```
pdi:
  metadata:
  rank: int
  width: int
  height: int
  data:
  main_field: {type:, size:[$width, $height]}
  plugins:
  decl_hdf5:
        - file: output_${rank}.h5
        write: [rank, main_field ]
```

Figure 35: Structure of a typical specification tree in YAML format.

Different I/O processes	SHEMAT subroutines related to I/O processes
reading the input files	- read_array - read_bc - read_check - read_control - read_data - read_model - read_property - read_split - read_time
writing the output files	- write_data - write_dense_3d - write_logs - write_monitor - write_outt - write_status_log - write_tecdiff - write_tecplot - write_tecplots - write_text - write_vtk
reading/writing/opening/closing/modify files of hdf5 format	- close_hdf5 - closeopen_hdf5 - mod_hdf5_vars - mod_input_file_parser_hdf5 - open_hdf5 - read_hdf5 - read_hdf5_int - test_hdf5 - write_all_hdf5

Figure 36: List of I/O related SHEMAT-SUITE f90-subroutines for the forward code.

The same process of identifying data, metadata and writing the configuration tree is done for other files covering the reading or opening of input or output. Fig. 37 shows an example of configuration tree written in a YAML file for a SHEMAT subroutines related to I/O process. The reading or writing is triggered in the SHEMAT-SUITE source by events, the plugins are then reacting to the events. Subsequently, the output routines of SHEMAT-SUITE will be replaced by PDI calls.

This part of the PDI-integration has been completed and test models have been defined. We begin with a simple test case and will increase the test model complexity, i.e. the amount and complexity of I/O data, successively. First, it will be a simple steady-state 2D case with only one

```
pdi:
  metadata: # small values for which PDI keeps a copy
    i:    int
    j:    int
    filename:    character
    line:    character
    filename_data:    character
  data: # values for which PDI does not keep a copy
    isplit: { type: array , subtype: int, size: ['$nsplit'] }

  plugins:
    test:
    decl_hdf5:
      - file: ${filename}.h5
        on_event: event_1
        read:
          nsplit:
            dataset: '$filename_data'
      - file: ${line(${i}:${j})}.h5
        on_event: event_2
        read:
          nsplit:
            dataset: '$filename_data'
```

Figure 37: Example of configuration tree in yaml file for a SHEMAT-Suite subroutine.

active model state, i.e. variable input array. The tested output format will be HDF5. If this first test is successful, we will add other types of input arrays to the test model and add other input parameters, e.g. by switching from a stationary to a transient simulation. In a next step, the PDI-integration needs to be tested with a 3D model. Finally, the PDI integration will be extended to I/O processes related to advanced SHEMAT functionalities, such as inversion.

8.3.2 Integration of PDAF into SHEMAT-Suite

The concept of PDAF is explained in deliverable D4.1 of WP4. To summarize, PDAF is a software framework for parallel ensemble data assimilation. PDAF contains fully implemented and optimized algorithms ensemble based Kalman filters and nonlinear filters ([32]). Like PDI, PDAF also includes API which permits to combine these algorithms with SHEMAT-Suite. Then, only minimal changes in the model source code are required for the implementation. Fig. 38 corresponds to the logical structure of the assimilation system of PDAF. The latter is based on a consistent structure of the three components of the data assimilation system: (i) the model; (ii) the filter algorithm; (iii) the observations. The filter algorithms are part of PDAF, while the model routines and the ones handling observations are provided by the user. A standardized interface for all filter algorithms connects the three components.

SHEMAT-Suite includes parameter estimation and data assimilation approaches, both stochastic (Monte Carlo, ensemble Kalman filter) and deterministic. We aim to implement PDAF instead of the serial ensemble Kalman filter (EnKF) currently implemented in SHEMAT-Suite for the stochastic inversion. Then, PDAF will allow to optimize the process of stochastic inversion and data assimilation.

8.4 Work progress on task 2.5.3

Task 2.5.3 originally concerned the development of a common base for the ParFlow and SHEMAT-Suite codes, bringing together CPU/GPU multi-platform parallelization and AMR capabilities as partly described in the proposal and the D2.1.

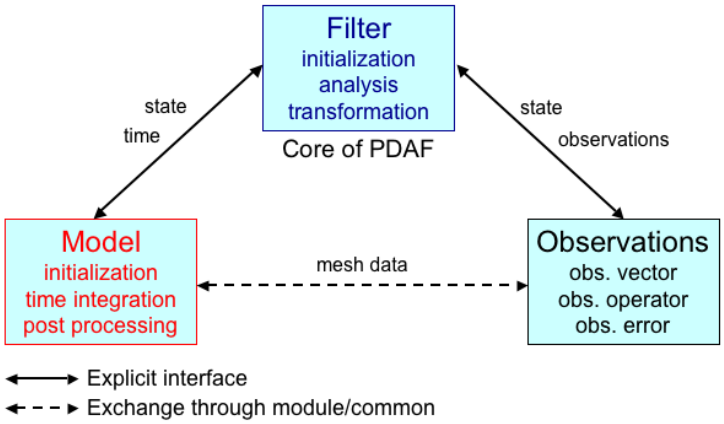


Figure 38: Logical separation of the assimilation system in PDAF. From the official website: `http://pdaf.awi.de/trac/wiki`

The platform called ExaTerr was to be based on Kokkos. This idea, although ideal for questions of stability and pooling of resources, proves to be far too ambitious in comparison to the needs solicited by tasks 2.5.1 and 2.5.2. The integration of AMR into ParFlow and GPU porting will take all the resources allocated to these activities.

Some of the work done during GPU porting is getting closer to the goals of this task. Indeed, alternative backends such as Kokkos or RAJA will be explored at the end of the project. This prospective activity is part of the preliminary work for this subtask. However, it is inconceivable to imagine a mature and functional platform at the end of EoCoE-II.

The needs of such a platform are also questionable and will not call into question the optimization of ParFlow for Exascale. The SHEMAT-Suite code could have benefited from this for its optimization but these are not the objectives of EoCoE-II for this code.

8.5 Risks, warning points and mitigation

Risks / Warning points	Who	Impact	Mitigation
Decreased efficiency and productivity due to COVID19 pandemic and subsequent lockdown during the last months.	SHEMAT-Suite	Delay of progress in task 2.5.2, which might result in a lack of time for finalizing the PDAF integration into SHEMAT-Suite.	PDAF integration will be split into subtasks, so that at least the basis for the parallel filter algorithm can be finalized and the integration can be extended in future projects or by other members of the SHEMAT-Suite team, if necessary.

Table 31: Risk management in Task 2.5.

9 Task 2.6 - Fusion code optimization

9.1 Task overview

Task leader : CEA-IRFM

Participants: CEA-IRFM, FAU, INRIA, MPG

The goal of the Fusion Scientific Challenge is to bridge the gap between gyrokinetic core transport modelling and edge plasma physics for reliable predictions of fusion performance, which will require a number of numerical and physics bottlenecks to be overcome. The Fusion SC is composed of a single flahship code GYSELA and satellite codes. Only GYSELA is concerned by the WP2. The objective is to develop a new numerical tool to address the core-edge issue, which will consist of refactoring and rewriting the flagship gyrokinetic code GYSELA [14, 22], targeting the disruptive use of billions of computing cores expected in exascale-class supercomputers. The new code after refactoring will be named GYSELAX.

The work in GYSELA in the context of the WP2 aims at modernizing and adapting the code for forthcoming super-computers including first pre-exascale prototypes and demonstrators. It is divided into 2 subtasks :

- Subtask 2.2.1 - Prototype of GYSELAX
- Subtask 2.2.2 - Advanced GYSELAX

Subtask 2.2.1 was originally dedicated to the prospecting of the best solutions in term of performance, readability and longevity for the prototype of GYSELAX. This substask was partly updated in the deliverable report at M6: D1.2. The choices and the work in subtask 2.2.1 is the subject of section 9.2.

Subtask 2.2.2 focuses on the main developments by extending the prototype. The last developments and results for this subtask are presented in section 9.3.

9.1.1 Flagship code GYSELA

GYSELA is a 5D full-f (regarding Vlasov equations) and flux-driven gyrokinetic FORTRAN parallel code that solves Vlasov (ions and electrons) and Poisson (electric potential) equations to simulate electrostatic plasma turbulence and transport in the core of Tokamak devices (GYSELA website). During EoCoE-II, it will progressively evolve towards an upgraded version, targeting exascale supercomputers and solving electromagnetic turbulence from the core to the far edge region in ITER-relevant magnetic geometry.

Table 32 shows the team members of GYSELA involved in EoCoE.

Yanick Sarazin is coordinating the Fusion activity within EoCoE. Virginie Grandgirard and Chantal Passeron are both supervising and working for the WP2 tasks. Julien Bigot is HPC advisor and focuses on PDI activities. Dorian Midou at CINES is external to the project and contribute to adapt the code on ARM architecture.

An important member has left at the beginning of the project, Guillaume Latu. He was HPC engineer coordinating and actively working on the refactoring of GYSELA. He was as well one of the main architect of the code structure with a long experience dealing with GYSELA performance issue. Therefore, the leave of Guillaume Latu has significantly impacted the project and particularly the WP2 tasks.

9.2 Work progress on subtask 2.6.1

The subtask 2.6.1 was originally dedicated to the development of a C++ GYSELAX prototype with a minimal set of optimized operators based on modern programming models and parallelization solutions. It was divided into the following items:

People	Position	Role	Period
Virginie Grandgirard, PhD	Researcher, CEA-IRFM	Numerical Analyst, GYSELA main developer	M1-M36
Chantal Passeron	Developer, CEA-IRFM	Support to GYSELA development	M1-M36
Julien Bigot	Researcher, CEA-MdlS	Computer Scientist, expert in HPC and I/O	M1-M36
Michel Mehrenberger	Researcher, AMU	Applied Maths., GYSELA developer	M1-M36
Emily Bourne	PhD student at CEA-IRFM and AMU	Computer Scientist, handling complex geometry	M10-M36
Dorian Midou	HPC engineer in CINES	external expert, optimization for ARM architecture	M12-M28
Yanick Sarazin	Researcher, CEA-IRFM	Physicist, coordination and reporting	M1-M36

Table 32: Team Members for GYSELA within EoCoE.

- New data structures with possible high-level memory abstraction using Kokkos
- Parallelized algorithms handling high concurrency
- First OpenMP task approach (asynchronism)

As explained in the first deliverable report D2.1, the long-term effort started before the beginning of EoCoE-2 to develop a C++ prototype has not shown enough advantages with respect to the main expectation.

Regarding parallel algorithms handling concurrency, it reveals limited efficiency of communication / computation overlap with less than typically 10% speedup. In addition, it has shown major and crippling drawbacks regarding both readability and maintainability. A paper is submitted on this issue [37]. This cannot fly given the need for physicists to be able to regularly access the code (modify equations, change boundary conditions, etc.). The same holds for some of the initially envisioned subtasks, namely strengthened task implementation and advanced runtime, and communication/computation MPI overlapping.

Regarding Kokkos implementations, tests have already been performed on C++ prototype applications, in particle-in-cell [3] and semi-Lagrangian schemes [4, 5]. The preliminary conclusion is that Kokkos reveals the most adequate framework for performance portability of a parallelized code over a broad spectrum of architectures. However, this implementation choice requires C++ language.

Therefore, the work done in this task has shown that the use of these previous technologies would require the complete rewriting of the code because the implementation would impose severe and deep rewriting with a lack of readability and maintainability for a production code mainly used by physicists like GYSELA. Adding to that the fact that the CEA-IRFM team loses one of the pillar developers of the code –with an expertise on computational science and high-level parallelism which cannot be replaced by non-permanent staff–, the initial option of rewriting from scratch the code in C++ was no longer a viable option. Taking into account all these constraints, it appears that the best solution is to keep a version based on a parallelization based on MPI + OpenMP loops. The objective is to successfully extract the most expensive computational kernels of the current code in order to be able to optimize them separately. In this idea of refactoring the code it is not yet excluded to rewrite some of these modules in C ++ to be able to benefit from the new technologies explored above (mainly Kokkos), but if this is the case it will be done at the end of the project.

Anyway, testing whether this alternative approach using FORTRAN with enhanced modularity is relevant for exascale applications, will already be a very interesting result in itself. In this framework, the PDI library likely offers a suitable solution. Indeed, successful results were obtained during EoCoE with respect to checkpoint-restart issues in GYSELA. PDI will be further implemented and its efficiency quantified on prototype versions before being deployed in GYSELAX if successful. PDI results is the subject of section 9.3.

Subtask 2.6.1 is considered as finished.

9.3 Work progress on task 2.6.2

Four main activities constitute the backbone of the GYSELAX development, some being backed by the outcomes of task T2.6.1. As described in the previous section, it was decided that the GYSELA code would stay the pillar of the new code GYSELAX by focusing our efforts on refactoring the code to add more modularity. This refactoring started with five months of complete cleaning of the code. This first step led to version 0 of GYSELAX in September 2010: while the release #31 (2019/04) contains 83 900 lines and 257 input parameters, the release #33 (2019/09) is much shorter with 65 800 lines (−22%) and 173 input parameters (−33%). As a matter of fact, this initial refactoring has greatly simplified –and even made it possible– the implementation of electromagnetic effects in the code. This was one of the important objectives of the project (see T1.5.1-1). The refactoring has then entered its second phase, which has started 6 months ago and will continue until the end of the project. The four activities described below are developed with a constant concern to increase this modularity.

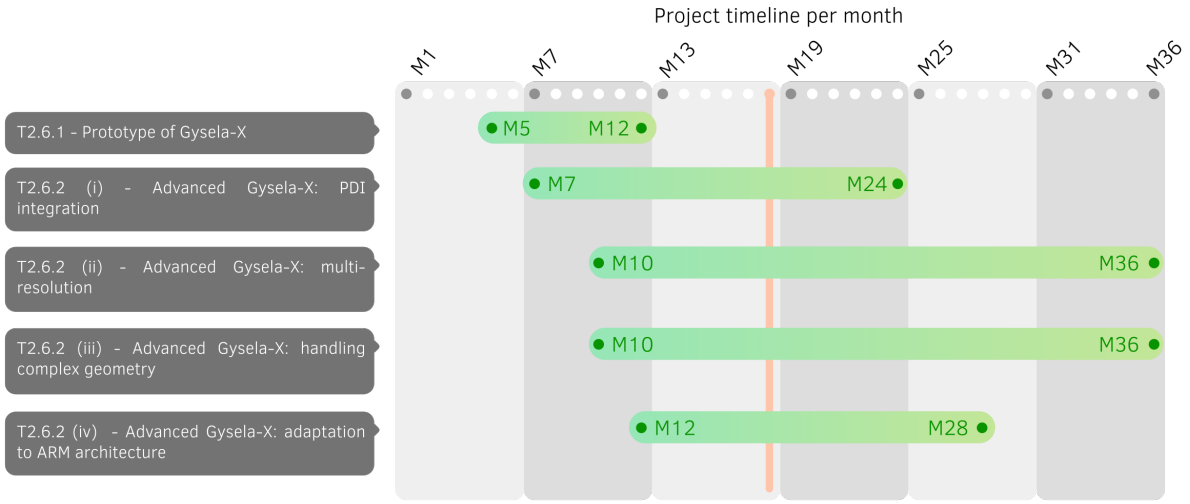


Figure 39: Breakdown (simplified Gantt chart) of the task 2.6 for GYSELAX.

9.3.1 PDI integration and enhanced modularity developments

A simplified version of PDI has been installed in GYSELA during EoCoE-I and successfully applied for the handling of the checkpoint-restart mechanism. This approach enabled us to evaluate and rely on the best checkpointing technology (HDF5, asynchronous HDF5, FTI, SIONlib, ...) depending both on the machine used and the type of run executed. The goal in GYSELAX is to go beyond this and use PDI as one of the technologies at multiple levels to increase code modularity. As of now, an extensive analysis has been completed, implementation choices have been made and the actual implementation phase proper has just started.

PDI will be used not only to handle the checkpointing process and I/O subroutines, but also to split the "diagnostics" from the main code in GyselaX. The diagnostics are parts of the Gysela code that apply data transformation and reductions on the main 5D particle distribution function to generate meaningful physical data fields that are then written to disk. These diagnostics are directly integrated in the code and not executed as post-processing because the amount of data to write before reduction would be unreasonable and make the code I/O bound by a large factor. This approach does however make writing these diagnostics complex, and limits the exploitation of the data generated by the code.

For GyselaX, we have designed a new approach where only the data field required to advance the simulation (mostly the distribution function and the electromagnetic fields) remain the responsibility of the simulation code. The exposure of those fields to PDI enable us to implement the diagnostics using existing and upcoming PDI in situ data analytics plugins. This approach will make the transition to Exascale possible by separating the question of the parallelization of the code simulation core from that of the diagnostics. We should thus be able to focus on the optimization of this core without impacting a huge difficult to maintain code-base.

The current status of the implementation is that the build-system has been upgraded to include the latest version of PDI instead of the old version originating from EoCoE-I. Initial work has gone on the use of PDI using the new more integrated approach for checkpointing in GyselaX. This will continue with the integration for result writings to disk and separation of the diagnostics.

Another aspect of the modularization work focuses on the Vlasov solver and more exaclty the spline interpolation subroutines of this solver. A new version of these subroutines has been implemented independently from GyselaX. Experiments have been conducted to implement these in C++ and integrating them in a FORTRAN code. The experiments seem successful for now and have helped us identify guidelines for the integration of independent modules in the code in different language. The experience acquired through this experiment will now be used to **a)** actually integrate this new spline interpolator in the code and **b)** adopt a similar approach to make most parts of the existing code-base as modular as possible.

9.3.2 Multi-resolution

The large temperature variation – typically by 2 orders of magnitudes – from the far edge to the very core of tokamak plasmas requires refined meshes. Multi-resolution and/or multi-patch approaches then reveal mandatory to avoid wasting large amounts of CPU time and memory resources. In the context of reduced manpower, we had to abandon the multi-patch strategy initially proposed because it would have required an almost complete rewriting of the code. Therefore, it has been decided to treat this intrinsic difficulty by using non-equidistant splines. This requires modularizing the existing equidistant splines and replacing this module with a new one. First coupling tests of a non-equidistant spline module developed in the SELALIB numerical library (collaboration CEA-IRFM Cadarache and MPG-IPP Garching) have been performed on a prototype code and reveal conclusive. The coupling to the GyselaX code is in progress and will still require several weeks of development due to the fact that the splines are used almost everywhere in the code. In a first step these new splines will be validated on the current equidistant mesh. This work is part of Emily Bourne PhD and is done in collaboration with Michel Mehrenberger (Aix-Marseille university). The transition to non-equidistant splines implies testing the semi-Lagrangian numerical schemes on non-equidistant meshes. As described in section 1.2.2, tests on simplified physical models were planned for this summer but will have to be postponed.

9.3.3 Complex Geometry

This major task requires rewriting most of the operators so as to handle generalized coordinates to be able to address ITER relevant D-shape magnetic geometries. So far, GYSELA can only cope with circular cross-sections. This task has started since the beginning of the year and will pursue all along the project. Several necessary steps have been now identified:

- The first step was to define an analytical equilibrium able to tackle D-shape magnetic configurations. The choice of Culham equilibriums has been made.
- The second step –directly related to the global refactoring strategy– was to extract the magnetic equilibrium initialization from the code. This new module was developed with the aim of being able to both initialize a circular equilibrium or a more general analytical equilibrium. This module has been validated for circular equilibrium. The covariant and contra-variant metric tensors associated to a Culham equilibrium transformation have been analytically derived. Their implementation in the new module is a work in progress.
- The third step is to take correctly into account this transformation in the Vlasov equations. At this stage some tests could be already done without modifying the Poisson solver to study the impact of the D-shape geometry on neoclassical transport.
- The fourth step consist in modifying all the diagnostics (around thirty diagnostics directly included in the code plus all the associated Python diagnostics). This step will be carried out by diagnostic group according to their priority levels. This coding can be performed separately from the previous step.
- The final step is to modify the Poisson solver. The quasi-neutrality equation is currently solved by projecting in Fourier space in the poloidal direction and by using finite differences in the radial direction. This strategy is no more applicable in the case of non-circular geometry. This will imply to extract the Poisson solver to replace it by one of the solvers developed for this purpose in task 1.5.1-2.

The numerical and computing work is part of Emily Bourne PhD and is done in collaboration with Xavier Garbet (IRFM/CEA) and David Zarzoso (CNRS/Aix-Marseille). The physical exploitation of the code with D-shape magnetic configurations will be explored by Kevin Obrejan (EoCoE-II post-doc hired for this task for 18 months since April 2019).

9.3.4 Adaptation to ARM architecture

In the D2.1 following the refactoring of the Fusion activity, there were 7 PMs left to be dedicated for this activity. Using additional funds, a job offer has been published at M9 for a duration of 1 year. At the same time, our aim was to reinforce and further develop our collaboration with the R-CCS in Kobe, Japan. Considering the lake of interesting candidate for the postdoctoral position and the positive exchange with R-CCS, it has been decided to cancel the position and use the found to strengthen the Japanese collaboration. Finally, CEA-IRFM is now collaborating with two external institutes on adapting the code on ARM architecture:

- CINES: since M12, Dorian Midou has been offering his expertise on ARM architecture to make GYSELA run on French ARM prototypes (ATOS prototype equipped of Thunder X2 ARM-based processors). He is as well strongly involved in the Japanese collaboration.

- R-CCS: The R-CCS collaboration is a good opportunity to access the pre-exascale Fujitsu computer Fugaku equipped of A64FX ARM processors [41]. It is active since M16. Tetsuya Odajima, Yuetsu Kodama and Kento Sato are proposing their expertise to first test GYSELA on the Fugaku machine. They will help to analyze the performance bottlenecks on this very recent computing architecture and optimize the code.

Comparison between HASWELL and ARM processors To evaluate the ARM cluster (INTI), the results of a reduced production case are compared with the results of this test case on OCCIGEN. OCCIGEN is a Tiers-1 HPC machine located at CINES (Montpellier - France). It integrates a partition of Haswell processors and a partition of Broadwell processors. The comparisons are made between ThunderX2 (INTI) and Broadwell (OCCIGEN). The table 33 summarizes the characteristics for one node of the two architectures.

Architecture	CPU/node	Vector register size [bits]	Freq [GHz]	Operation / cycle	Peak performance [TFlops]
Broadwell	28	256	2.6	4	1.1648
ThunderX2	64	128	2.4	4	1.2288

Table 33: Characteristics between Broadwell (OCCIGEN) and ThunderX2 (INTI) nodes. The peak performance is computed as CPU/node $\times$ Freq $\times$ Vector register size/64 $\times$ Operation/cycle.

The table 33 shows that it can be expected a gain of 5.5% in computation time between OCCIGEN and INTI.

The test case runs with 32 MPI processes and represents 796M degrees of freedom. The simulation of this test case is performed following two configurations on both clusters:

1. **configuration 1:** 4 MPI processes per node
2. **configuration 2:** 2 MPI processes per node

To compare the performance of the processors, only the compute intensive part of the GYSELA code is studied. The table 34 shows the compute time needed for both configurations on both platforms.

CASE	MPI/node	Thread/MPI	Nb of nodes	Compute Time [s]	Compute Time [node.hours]
OCCIGEN	4	14	8	71.85	0.16
ARM	4	32	8	66.83	0.15
OCCIGEN	2	28	16	50.4	0.22
ARM	2	64	16	44.83	0.20

Table 34: Total compute time in seconds and total compute time as node.hours for the test case on OCCIGEN and ARM clusters for both configurations

The table 34 shows that the speed up between OCCIGEN and INTI goes from 1.07 (4 MPI/node) to 1.12 (2 MPI/node). That means that if a production simulation needs 1 node.hours on OCCIGEN, it will need 0.93/0.90 node.hours on this ARM platform.

If the gains are not massive because the performance of one node of these clusters are similar, it is a first step towards the reduction of the compute resources. Moreover, it is to be noted that on the

ThunderX2 architecture the Scalable Vector Extension (SVE) is not available, only 128bits vectors are used (instead of 256bits vectors on Occigen). It is then expected greater gain when GYSELA can access this technology and hence benefit from a larger vector width.

It is also to be noted that in order to achieve the smallest elapsed time on INTI it is mandatory to properly bind the processes. This is performed through the following strategy:

- each MPI process is bind to a pre-defined number of cores within one NUMA node
- the OpenMP threads for each MPI process are bind to the cores allocated to a MPI process : **export KMP_AFFINITY="granularity=fine,compact,0,0,verbose"**

Without proper MPI biding the elapsed time can be multiplied by 20. Without proper thread binding the elapsed time can increase of 16%.

10 Conclusion

At this stage of the project, no code is entirely ready to run with all the planned optimizations and developments. Nevertheless, a number of applications are already very advanced (ALYA, PARFLOW, GYSELA) and already show good performances. Other applications have been a little late in their development schedule due to various reasons: late recruitment (EURAD-IM, GYSELA, SHEMAT-SUITE, PARFLOW), loss of staff (LIBNEGF, GYSELA), impact of the health crisis (LIBNEGF), problem of access to computing resources (MESO-NH).

Despite these hazards, most developments follow their road map. Most applications will be ready for the first pre-exascale machines. We can wonder when these machines will be available in Europe. The current PRACE allocation for CoEs does not allow to make very large simulations because of the limits in computation time.

References

[1] ALYA's *website*. URL: https://www.bsc.es/research-and-development/software-and-apps/software-list/alya (visited on 06/01/2020).

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Third. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999. ISBN: 0-89871-447-8 (paperback).

[3] Victor Artigues, Katharina Kormann, Markus Rampp, and Klaus Reuter. "Evaluation of performance portability frameworks for the implementation of a particle-in-cell code". In: *Concurrency and Computation: Practice and Experience* 32.11 (2020), e5640.

[4] Yuuichi ASAHI, Guillaume Latu, Virginie Grandgirard, and Julien Bigot. *Accelerating plasma simulation codes with portable frameworks: OpenACC and Kokkos*. (Visited on 06/01/2020).

[5] Yuuichi ASAHI, Guillaume Latu, Virginie Grandgirard, and Julien Bigot. "Performance portable implementation of a kinetic plasma simulation mini-app". In: (2020).

[6] Henrik Asmuth, Hugo Olivares-Espinosa, Karl Nilsson, and Stefan Ivanell. "The Actuator Line Model in Lattice Boltzmann Frameworks: Numerical Sensitivity and Computational Performance". In: *Journal of Physics: Conference Series* 1256 (July 2019), p. 012022. DOI: 10.1088/1742-6596/1256/1/012022. URL: https://doi.org/10.1088%2F1742-6596%2F1256%2F1%2F012022.

[7] Martin Bauer, Sebastian Eibl, Christian Godenschwager, Nils Kohl, Michael Kuron, Christoph Rettinger, Florian Schornbaum, Christoph Schwarzmeier, Dominik Thönnes, Harald Köstler, and Ulrich Rüde. "waLBerla: A block-structured high-performance framework for multiphysics simulations". In: *Computers & Mathematics with Applications* (2020). ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2020.01.007. URL: http://www.sciencedirect.com/science/article/pii/S0898122120300146.

[8] Martin Bauer, Harald Köstler, and Ulrich Rüde. "lbmpy: A flexible code generation toolkit for highly efficient lattice Boltzmann simulations". preprint on webpage at https://arxiv.org/pdf/2001.11806v1.pdf. Jan. 2020.

[9] Julien Bigot. PDI *website*. URL: https://pdi.julien-bigot.fr/0.5.1/ (visited on 06/01/2020).

[10] Matthew Churchfield, Scott Schreck, Luis MartÃnez Tossas, Charles Meneveau, and Philippe Spalart. "An Advanced Actuator Line Method for Wind Energy Applications and Beyond". In: *35th Wind Energy Symposium*. Jan. 2017. DOI: 10.2514/6.2017-1998.

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

[12] S. Donath, K. Iglberger, G. Wellein, T. Zeiser, A. Nitsure, and U. Rude. "Performance Comparison of Different Parallel Lattice Boltzmann Implementations on Multi-Core Multi-Socket Systems". In: *Int. J. Comput. Sci. Eng.* 4.1 (Nov. 2008), pp. 3–11. ISSN: 1742-7185. DOI: 10.1504/IJCSE.2008.021107. URL: https://doi.org/10.1504/IJCSE.2008.021107.

[13] F. Ducros, Nicoud Franck, and Thierry Poinsot. "Wall-Adapting Local Eddy-Viscosity Models for Simulations in Complex Geometries". In: *Numerical Methods for Fluid Dynamics VI* (Jan. 1998).

[14] D. Estève, Y. Sarazin, X. Garbet, V. Grandgirard, S. Breton, P. Donnel, Y. Asahi, C. Bourdelle, G. Dif-Pradalier, C. Ehrlacher, C. Emeriau, Ph. Ghendrih, C. Gillot, G. Latu, and C. Passeron. "Self-consistent gyrokinetic modeling of neoclassical and turbulent impurity transport". In: *Nuclear Fusion* 58.3 (Jan. 2018), p. 036013. DOI: 10.1088/1741-4326/aa6ebb. URL: https://doi.org/10.1088%2F1741-4326%2Faa6ebb.

[15] *EXA2PRO project*. URL: https://exa2pro.eu/ (visited on 06/01/2020).

[16] A. Galonska, W. Frings, P. Gibbon, D. Borodin, and A. Kirschner. “JuBE-based Automatic Testing and Performance Measurement System for Fusion Codes”. In: *Applications, Tools and Techniques on the Road to Exascale Computing / ed.: K. De Bosschere, E.H. D'Hollander, G.R. Joubert, David Padua, Frans Peters, Mark Sawyer, IOS Press, 2012, Advances in Parallel Computing, Vol. 22. - 978-1-61499-040-6. - S. 465 - 472*. Record converted from VDB: 12.11.2012. 2012. DOI: 10.3233/978-1-61499-041-3-465. URL: https://juser.fz-juelich.de/record/17928.

[17] Thierry Gautier. *LibKOMP*. URL: https://gitlab.inria.fr/openmp/libkomp (visited on 06/01/2020).

[18] Martin Geier, Martin Schönherr, Andrea Pasquali, and Manfred Krafczyk. “The cumulant lattice Boltzmann equation in three dimensions: Theory and validation”. In: *Computers & Mathematics with Applications* 70.4 (2015), pp. 507–547. ISSN: 0898-1221. DOI: https://doi.org/10.1016/j.camwa.2015.05.001. URL: http://www.sciencedirect.com/science/article/pii/S0898122115002126.

[19] *GENE*. Accessed: 2020-06-01.

[20] Judit Gimenez Georg Hager Jan Treibig. *EoCoE Performance Evaluation Workshop recorded videos*. URL: https://public.weconext.eu/eocoe2/2019-10-07/index.html (visited on 06/01/2020).

[21] Christian Godenschwager, Florian Schornbaum, Martin Bauer, Harald Köstler, and Ulrich Rüde. “A Framework for Hybrid Parallel Flow Simulations with a Trillion Cells in Complex Geometries”. In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. SC ’13. Denver, Colorado: Association for Computing Machinery, 2013. ISBN: 9781450323789. DOI: 10.1145/2503210.2503273. URL: https://doi.org/10.1145/2503210.2503273.

[22] GYSELA*’s website*. URL: http://gyseladoc.gforge.inria.fr/ (visited on 06/01/2020).

[23] Pankaj Jha, Matthew Churchfield, Patrick Moriarty, and Sven Schmitz. “Guidelines for Volume Force Distributions Within Actuator Line Modeling of Wind Turbines on Large-Eddy Simulation-Type Grids”. In: *Journal of Solar Energy Engineering* 136 (Aug. 2014), p. 031003. DOI: 10.1115/1.4026252.

[24] Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen Malony, et al. “Score-p: A joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir”. In: *Tools for High Performance Computing 2011*. Springer, 2012, pp. 79–91.

[25] *KOKKOS*. URL: https://github.com/kokkos/kokkos (visited on 06/01/2020).

[26] Carolin Körner, Thomas Pohl, Ulrich Rüde, Nils Thürey, and Thomas Zeiser. “Parallel Lattice Boltzmann Methods for CFD Applications”. In: *Numerical Solution of Partial Differential Equations on Parallel Computers*. Ed. by Are Magnus Bruaset and Aslak Tveito. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 439–466. ISBN: 978-3-540-31619-0.

[27] Jonas Latt and Bastien Chopard. “Lattice Boltzmann method with regularized pre-collision distribution functions”. In: *Mathematics and Computers in Simulation* 72.2 (2006). Discrete Simulation of Fluid Dynamics in Complex Systems, pp. 165–168. ISSN: 0378-4754. DOI: https://doi.org/10.1016/j.matcom.2006.05.017. URL: http://www.sciencedirect.com/science/article/pii/S0378475406001583.

[28] Mathieu Lobet and Georg Hager. *EoCoE Performance Evaluation Workshop registration website*. URL: https://indico.math.cnrs.fr/event/4587/ (visited on 06/01/2020).

[29] Sebastian Lührs, Daniel Rohe, Alexander Schnurpfeil, Kay Thust, and Wolfgang Frings. “Flexible and Generic Workflow Management”. In: *Parallel Computing: On the Road to Exascale*. Vol. 27. Advances in parallel computing. International Conference on Parallel Computing 2015, Edinburgh (United Kingdom), 1 Sep 2015 - 4 Sep 2015. Amsterdam: IOS Press, Sept. 1, 2015, pp. 431–438. ISBN: 978-1-61499-620-0. DOI: 10.3233/978-1-61499-621-7-431. URL: https://juser.fz-juelich.de/record/808798.

[30] *MALI ARM GPU*. URL: https://exa2pro.eu/ (visited on 06/01/2020).

[31] Chohong Min, Frédéric Gibou, and Hector D Ceniceros. “A supra-convergent finite difference scheme for the variable coefficient Poisson equation on non-graded grids”. In: *Journal of Computational Physics* 218.1 (2006), pp. 123–140.

[32] L. Nerger and W. Hiller. “Software for ensemble-based data assimilation systems-implementation strategies and scalability”. In: *Comput. Geosci.* 55.1 (2013), pp. 110–118.

[33] Elmar Peise and Paolo Bientinesi. “The ELAPS Framework: Experimental Linear Algebra Performance Studies”. In: *The International Journal of High Performance Computing Applications* 33.2 (2019), pp. 353–365.

[34] *POP Center of Excellence*. URL: https://pop-coe.eu/ (visited on 06/01/2020).

[35] *RAJA*. URL: https://github.com/LLNL/RAJA (visited on 06/01/2020).

[36] 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”. In: *Geophysical Journal International* 167.1 (2006), pp. 453–466.

[37] Jérôme Richard, Guillaume Latu, Julien Bigot, and Thierry Gautier. “Fine-Grained MPI+ OpenMP Plasma Simulations: Communication Overlap with Dependent Tasks”. In: *European Conference on Parallel Processing*. Springer. 2019, pp. 419–433.

[38] C. Roussel, K. Keller, M. Gaalich, L. Bautista Gomez, and J. Bigot. “PDI, an approach to decouple I/O concerns from high-performance simulation codes”. In: *hal-01587075* 1.1 (2017), pp. 1–12.

[39] Florian Schornbaum and Ulrich Rüde. “Extreme-Scale Block-Structured Adaptive Mesh Refinement”. In: *SIAM Journal on Scientific Computing* 40.3 (2018), pp. C358–C387. DOI: 10.1137/17M1128411. eprint: https://doi.org/10.1137/17M1128411. URL: https://doi.org/10.1137/17M1128411.

[40] Florian Schornbaum and Ulrich Rüde. “Massively Parallel Algorithms for the Lattice Boltzmann Method on NonUniform Grids”. In: *SIAM Journal on Scientific Computing* 38.2 (2016), pp. C96–C126. DOI: 10.1137/15M1035240. eprint: https://doi.org/10.1137/15M1035240. URL: https://doi.org/10.1137/15M1035240.

[41] *Supercomputer “Fugaku”*. URL: https://www.fujitsu.com/global/Images/supercomputer-fugaku.pdf (visited on 06/01/2020).

[42] *Top-ranked systems*. URL: https://www.top500.org/ (visited on 06/01/2020).

[43] N. Troldborg. “Actuator Line Modeling of Wind Turbine Wakes”. PhD thesis. Technical University of Denmark, 2009.

[44] *Weather Research and Forecasting model*. URL: https://www.mmm.ucar.edu/weather-research-and-forecasting-model (visited on 06/01/2020).