EuroHPC-01-2019

DEEP-SEA

DEEP – Software for Exascale Architectures

Grant Agreement Number: 955606

D1.1

Initial application co-design input

Final

Version:	1.0
Author(s):	JH Meinke (FZJ), A. Kreuzer (FZJ)
Contributor(s):	J. Amaya (KUL), D. Caviedes Voullieme (FZJ), P. Carribault (CEA), D. Grünewald (FhG), N. Jansson (KTH), D. Mancusi (CEA), S. Markidis (KTH), O. Marsden (ECMWF), J. de la Puente (BSC), J.E. Rodriguez (BSC), O. Castillo-Reyes (BSC)
Date:	31.07.2021

Project and Deliverable Information Sheet

DEEP-SEA Project	**Project ref. No.:**	955606
	Project Title:	DEEP – Software for Exascale Architectures
	Project Web Site:	https://www.deep-projects.eu/
	Deliverable ID:	D1.1
	Deliverable Nature:	Report
	Deliverable Level: PU*	**Contractual Date of Delivery:** 31.07.2021
		Actual Date of Delivery: 31.07.2021
	EC Project Officer:	Daniel Opalka

*– The dissemination levels are indicated as follows: **PU** - Public, **PP** - Restricted to other participants (including the Commissions Services), **RE** - Restricted to a group specified by the consortium (including the Commission Services), **CO** - Confidential, only for members of the consortium (including the Commission Services).

Document Control Sheet

Document	**Title:** Initial application co-design input	
	ID: D1.1	
	Version: 1.0	**Status:** Final
	Available at: https://www.deep-projects.eu/	
	Software Tool: LaTeX	
	File(s): DEEP-SEA_D1.1_Initial_application_co-design_input.pdf	
Authorship	**Written by:**	JH Meinke (FZJ), A. Kreuzer (FZJ)
	Contributors:	J. Amaya (KUL), D. Caviedes Voullieme (FZJ), P. Carribault (CEA), D. Grünewald (FhG), N. Jansson (KTH), D. Mancusi (CEA), S. Markidis (KTH), O. Marsden (ECMWF), J. de la Puente (BSC), J.E. Rodriguez (BSC), O. Castillo-Reyes (BSC)
	Reviewed by:	P. Carpenter (BSC) I. Schmitz (ParTec)
	Approved by:	BoP/PMT

Document Status Sheet

Version	Date	Status	Comments
1.0	31.07.2021	Final version	EC submission

Document Keywords

Keywords:	DEEP-SEA, HPC, Exascale, Software, Applications, Co-design

Acknowledgements:

The DEEP Projects have received funding from the European Commission's FP7, H2020, and EuroHPC Programmes, under Grant Agreements n° 287530, 610476, 754304, and 955606. The EuroHPC Joint Undertaking (JU) receives support from the European Union's Horizon 2020 research and innovation programme and Germany, France, Spain, Greece, Belgium, Sweden, United Kingdom, Switzerland.

EuroHPC
Joint Undertaking

SPONSORED BY THE
Federal Ministry
of Education
and Research

Swedish
Research
Council

Contents

List of Figures

List of Tables

Executive Summary

The applications from work package 1 (WP1, Co-Design Applications) provide use cases and requirements as co-design input to the other work packages. They demonstrate the capabilities of the DEEP-SEA software stack and evaluate its performance and usability for a wide range of scientific applications from molecular dynamics to space weather. The results feed back into the development cycle of the project in close collaboration with all work packages. To cover as many aspects of the software stack as possible WP1 also maintains a set of additional benchmarks.

To initiate the interaction between the work packages and obtain relevant input from the application, we organised a weekly seminar. Each week an application and an API or program model were introduced and discussed. The seminars have been attended typically by more than 50 people and have helped to clarify the kind of input needed.

This deliverable summarizes the requirements of the different applications.

1 Introduction

The DEEP-SEA project is developing the software stack for European exascale computers based on the Modular System Architecture (MSA) developed in DEEP-EST. A core component of the project is the co-design process with application from 7 highly relevant application areas. This deliverable provides the initial co-design input from the applications. The weekly DEEP-SEA seminar series has provided a forum to introduce and discuss the applications and a selection of Application Programming Interfaces (APIs) and programming models. The seminars have been very well attended and will continue.

The following chapters introduce the applications and their workflow. Each application presents the parallelization scheme used and presents its requirement in terms of interface, libraries, and hardware.

2 Space Weather

2.1 Introduction

KU Leuven is an autonomous university. It was founded in 1425. KU Leuven is a research-intensive, internationally oriented university that carries out both fundamental and applied research. It is strongly inter- and multidisciplinary in focus and strives for international excellence. To this end, KU Leuven works together actively with its research partners at home and abroad.

The team from KU Leuven (KULeuven) is affiliated to the Centre for mathematical Plasma Astrophysics (CmPA), founded 24 years ago as a division of the Mathematics Department of KU Leuven. The team is also associated with the Leuven.AI Institute and the Leuven Centre for Aero & Space Science, Technology and Applications (LASA). The CmPA includes four permanent staff members, two part-time affiliated staff member and around 30 research fellows, postdocs and PhD students. The CmPA employs applied mathematics, high performance computing and fundamental plasma physics models. The CmPA is also involved in multiple space missions (e.g., Proba2, IRIS, MMS, Parker Solar Probe, BepiColombo and Solar Orbiter). The CmPA has expertise in areas of space and plasma physics with a strong experience in supporting observational and experimental research. Examples are the support of fusion energy experiments and the strong participation in the ESA Space Situational Awareness (SSA) initiative. One of the three core components of the SSA program is the Space Weather (SWE) service dedicated to the study, surveillance and forecasting of the space environment from the Sun to the Earth.

Space weather studies the physics of the energetic activity of the Sun, the propagation of solar plasma through the solar system, its interaction with the plasma environment of the planets and the effects on the Earth's atmosphere, on human life and technology. Space weather has a high societal relevance for its impact on multiple industries (satellite operations, telecommunications, geolocation, electric distribution, space exploration, among others). During the DEEP projects, KU Leuven has developed the particle-in-cell code xPic, which studies the effects of solar plasma on the environment of different planets in the solar system. A suite of Data Analysis (DA) and Machine Learning (ML) techniques are employed around xPic to set up the initial conditions of the code and its boundary conditions and to analyse the generated data. This space weather workflow uses already the MSA and runs on the DEEP-EST prototype hosted by J?lich Supercomputing Centre.

The code xPic has two solvers: a field solver for the computation of electromagnetic fields (which runs on the Booster module), and a particle solver for the transport of plasma ions and electrons (which runs on the Cluster module). Terabytes of data are produced in the particle solver. The code AIDApy creates initial conditions for the simulation (using Deep Learning or Self-Organising Maps techniques) and analyses the velocity distribution data of the particles (using Gaussian Mixture Models).

2.2 xPic: plasma physics HPC code

The code xPic is based on the Particle-in-Cell (PIC) algorithm. The physical phenomenon that xPic studies is the dynamics and transport of space plasmas. Plasma is the fourth state of matter:

when very high pressures or temperatures are applied to matter, the electrons are peeled from the ions. These two types of particles (ions and electrons) carry opposite electric charges (positive and negative). The movement of each individual particle is then governed by electromagnetic forces. At the same time, the movement of the particles causes changes in the total electric charge of the environment and can produce currents that modify in turn the electric and magnetic fields. In a plasma the charged particles and the electromagnetic fields are tightly coupled.

Figure 1 shows the general structure of a PIC code. The algorithm is divided in four main phases: a) a *field solver* calculates the evolution of Maxwell's equations of electromagnetism using a numerical solver; b) a *particle solver* transports billions (or trillions) of individual and independent particles of different electric charge (ions and electrons), c) the movement of the particles depends on the forces interpolated from the field solver to the particle solver, d) and finally statistical information is gathered from the particles by integration of different moments of the velocity distribution function.

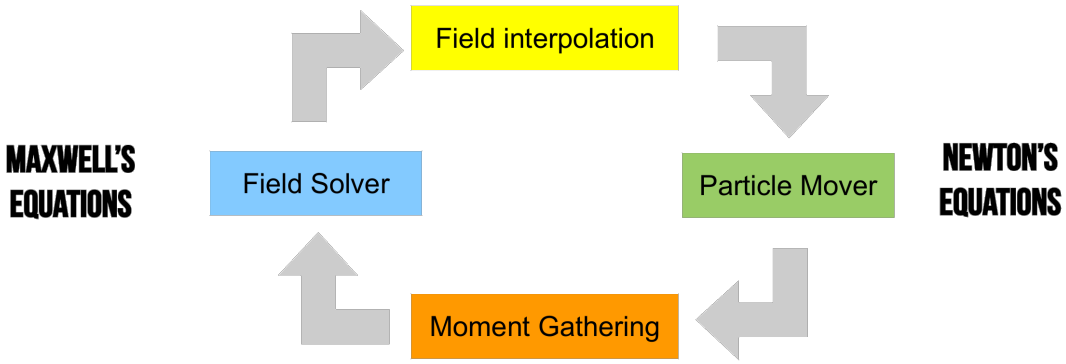


Figure 1: Four phases of the PIC algorithm used in the code xPic (KU Leuven).

2.2.1 Programming languages

The code xPic has been programmed in C++ mainly using the standard C++14. Some pre- and post-processing tools have been implemented in Python.

2.2.2 Libraries and other dependencies

The field solver of xPic uses PETSc. Files are read and saved in parallel using HDF5.

2.2.3 Parallelization

The code xPic uses three levels of parallelism: inter-node message passing using MPI, intra-node multi-threading using OpenMP and thread vectorization using OpenMP. This design was proposed during the DEEP-ER project in order to have optimal performance on Intel Xeon Phi accelerators. However, in the DEEP-EST project we had to restructure the code in order to use GPU accelerators

instead. Testing showed better performance when one thread per MPI process was used, and multiple MPI processes were requested for each node. Offloading to the GPUs has been implemented using OpenMP 5.0. Data transfers between GPUs is performed using GPU-aware MPI communications.

Figure 2 shows the dominant MPI communication pattern in xPic for a 2D example running in: 2 nodes x 24 cores/node = 48 cores. In the field and the particle solver the computational domain is divided in subdomains that are processed independently by each MPI process. In the field solver, a linear iterative method is used to minimize the residual of a linear system. This iterative method requires to gather the residual error at each iteration performing a global MPI reduction.

However, the dominant pattern is imposed by the particle solver. Particles that exit the MPI subdomain are transferred to the neighbouring MPI process. In the case used to extract Figure 2 a two-dimensional problem was used. There are four dominant diagonals, corresponding to the right, left, top and bottom MPI neighbours of each MPI process.

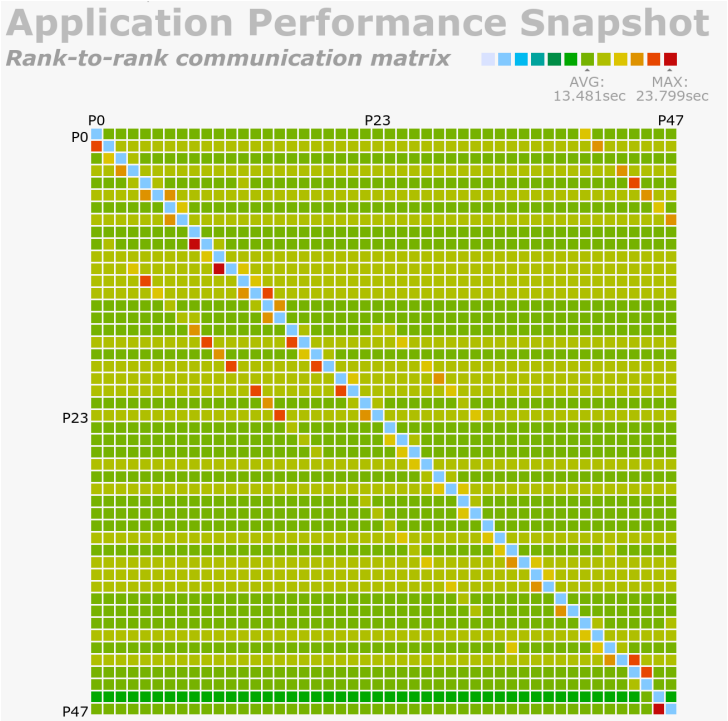


Figure 2: Dominant MPI communication patterns in xPic. Figure extracted with Intel APS. (KU Leuven).

Accelerators

The code xPic has been adapted to use GPUs. All particle computations can take place in the GPU on which the particle resides. We currently allocate all particle memory in the GPU itself, avoiding any time of transfer to the corresponding host.

To test the performance of the GPU section of the code, we deactivated all the different parts of the code, excep the particle mover. Figure 3 shows the weak scaling obtained when only the particle

mover is used (no field resolution, no particle moments gathering, no field interpolation). This test has been used to verify that the GPU offloading using OpenMP 5.0 is producing acceptable performance.

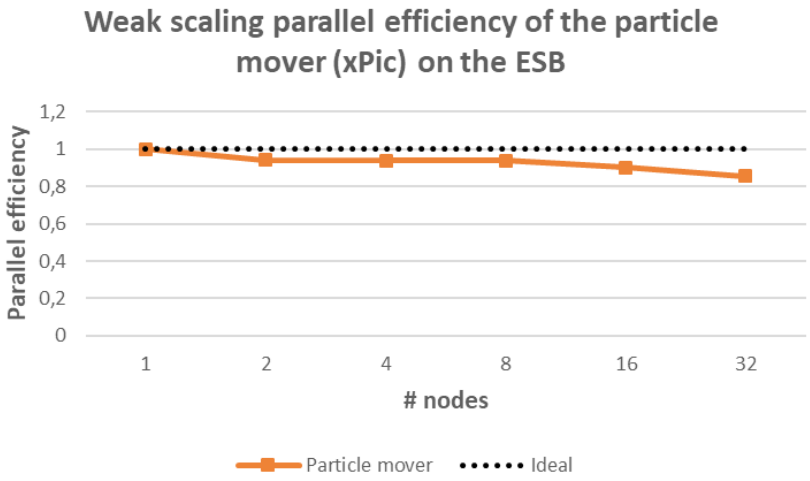


Figure 3: Weak scaling of the particle solver of xPic using the GPUs of the DEEP-EST Booster. (KU Leuven).

We are aware that there are still some serial sections in the code that can hinder the scalability of this section of the code. We will be working on the optimization of this particular section.

2.2.4 Typical use case

On most cases we perform simulations of 2D plasma setups, using a resolution of around 1024 x 1024 cells, and 32 particles per cell. For the tests presented in the sections above, we have set a 2D case with 512 x 768 cells and 1024 particles per cell. This is a problem size that is large enough to perform tests in a few computational nodes. Ten (10) iterations of this case are executed in around four (4) minutes, using a total of 48 cores.

During this project we will set up a few other tests of shorter duration with the goal to stress different phases of the code and find the best compromise between mesh size and number of particles.

2.2.5 Target use case

Our goal is to perform a full 3D simulation of the interaction of the solar wind with a planetary magnetosphere. This problem requires at least the following configuration: 1024 x 1024 x 2048 cells, with 2048 particle per cell. The simulation will need to run for at least 100000 iterations. This problem is 4 orders of magnitude larger than the tests used in the previous sections. Such problem would be a perfect candidate for an exascale machine and would produce significant scientific results.

2.2.6 Application requirements

Scaling

Our goal is to have at the same time a good weak scaling of the code and a very high value of the number of particles computed (also called as *particles pushed*) per second. The later is used as point of comparison with other existing PIC codes, while the former is needed in order to perform higher resolution and more detailed simulations that reproduce the real physics.

Compute

Figure 4 shows a roofline analysis of the code xPic using the case described in the sections above. This plot shows that many functions are approaching the optimal corner of the graph. However, this test was run on a CPU-only cluster, and the version of the code does not include detailed vectorization routines. When the code is running in a CPU machine the critical sections of the code needs explicit vectorization to achieve better performance.

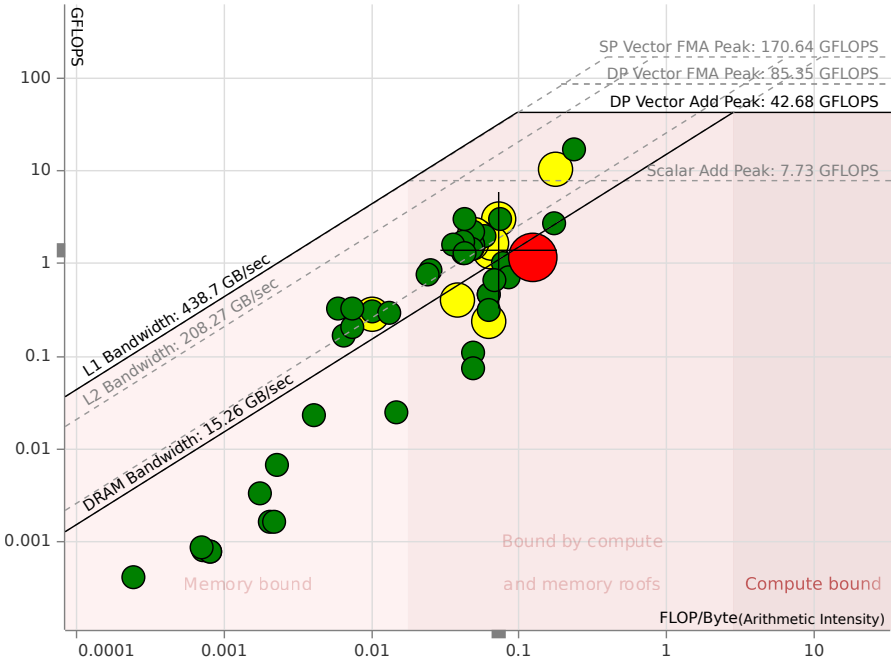


Figure 4: Roofline analysis of the code xPic running with the basic configuration, without optimizations, using the *monolithic* mode on the DEEP-EST CPU Cluster Module.

We require good compilers that can produce the best optimization calls and that can give hints to the developers about the sections that can be further improved. In particular we also require tools to profile and debug GPU code offloaded with OpenMP.

Memory

We used Intel APS to extract the memory footprint of our test. It shows that there is a peak use of 50 GiB per node and 2.1 GiB per MPI rank. The memory footprint is directly correlated with the number of particles in the simulation. This value can thus be changed to accommodate any given architecture. The PIC algorithm works better when the maximum possible number of particles are included in the simulation.

The movement of the particles uses a very simple set of operations. The particle mover thus benefits from a very high bandwidth memory to ingest and process as many particles as possible at the same time.

IO

Files containing information about the electromagnetic fields are saved often in order to perform data analysis. Each one of the files has a size of only a few megabytes. On the other hand, the file containing particle information can be extremely large. We have performed simulations that produce files up to 1 TiB large.

All files are be written in parallel using two possible libraries: HDF4 and SIONlib.

Malleability and resilience

The application can run in two different modes: a) in the *monolithic* mode the code runs as defined in the previous sections, using an iterative cycle between the field solver and the particle solver, b) in the *modular* mode the code is divided in *Cluster* and *Booster* sections that run in different modules of the MSA. Figure 5 shows the sequence of steps in the main loop of the code for the monolithic and the modular modes.

The code can run fully on CPUs, and the particle solver has the additional option to use GPU offloading. The running modes and the offloading options allows to perform simulations in different architectures using different setups.

In case of an unexpected crash, the application uses SCR for restart. SCR uses SIONlib as a back-end keeping in the local storage the necessary data to restart a failed simulation.

2.3 AIDApy: space data analytics python package

AIDApy is the main code developed by the Horizon 2020 project AIDA (Artificial Intelligence and Data Analysis, Grant ID: 776262). The goal of this python package is to bring together three critical aspects in the study of heliophysical data: computer simulations of space plasmas, data from space missions, and machine learning algorithms. The goal of the package is to improve our scientific knowledge of the plasmas in the Solar Systems using AI and ML on simulations and observations.

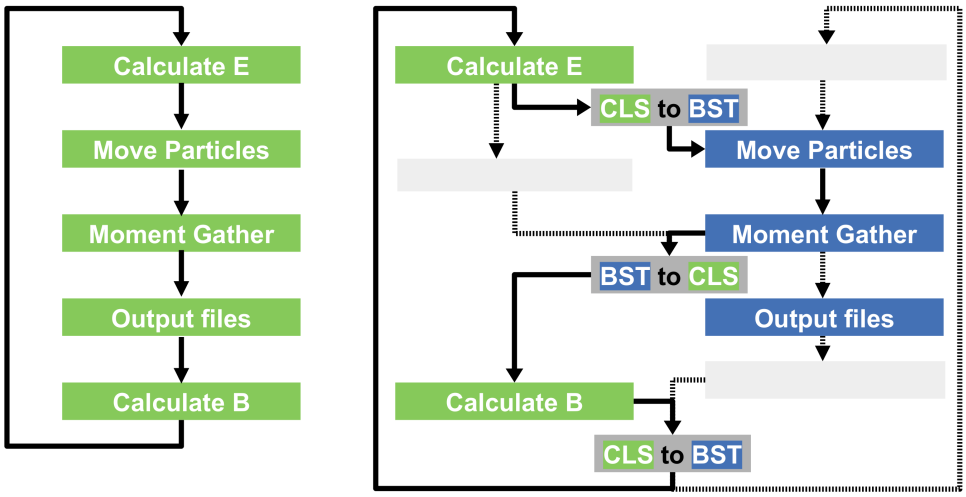


Figure 5: Left: sequence of steps of the main loop of the code xPic in the monolithic mode. Right: same sequence in the modular mode where the steps have been split in the Cluster (CM) and Booster (ESB) modules, and data transfers between the two modules have been added.

The package AIDApy is then composed of a multitude of python modules with different functionalities. For the DEEP-SEA project we will use three particular utilities of AIDApy: a) the data client that downloads data from multiple missions, b) the unsupervised Self-Organizing Map (SOM) method for the classification of spacecraft data, and c) the Gaussian Mixture Model (GMM) for the identification and analysis of particle distributions generated by the code xPic.

We are expecting to use the AIDApy package in two different ways: a) analysis of solar wind information to forecast the plasma conditions to inject in xPic simulations, and b) the automatic classification and analysis of particle distributions generated by the particle solver of xPic.

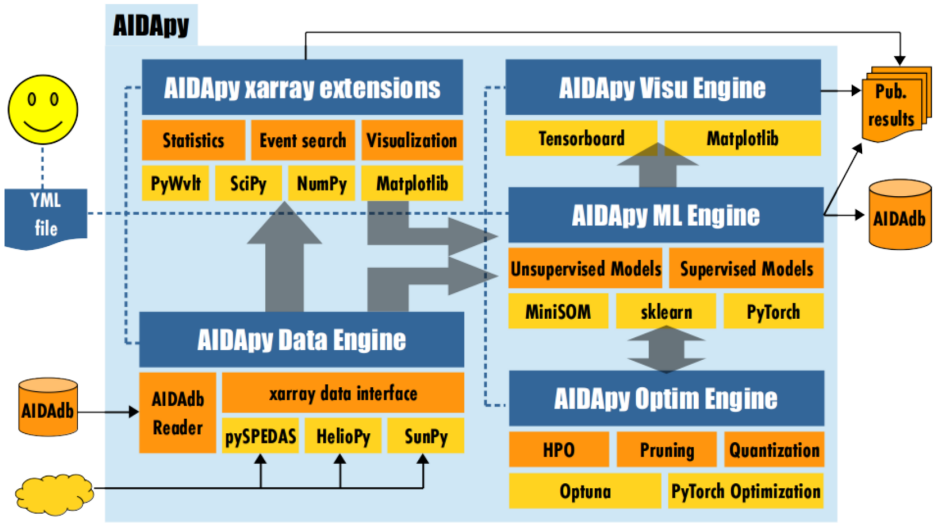


Figure 6: Internal structure of the AIDApy package (KU Leuven).

Figure 6 shows the internal organization of the AIDApy package. This is a package build on top of already existing python packages, like HelioPy, SunPy, PyWavelets, etc. The goal of this software is not to re-write existing software but to coordinate their use in a central package that allows interoperability.

From AIDApy, the Data Engine and the ML Engine will be used in the DEEP-SEA project. We use the former to download spacecraft data for analysis, and the later to perform the Self-Organizing Map and the Gaussian Mixture Model classifications.

2.3.1 Programming languages

The package has been entirely developed using Python 3.

2.3.2 Libraries and other dependencies

There is a large number of dependencies of this package, but all of them are included in the requirements file and the setup script. However, there are a few packages that require special care in the installation in the MSA. These are all the packages related to the machine learning framework and to the handling of parallelism. In the case of AIDApy we rely in particular on PyTorch, scikit-learn, mpi4py and Horovod.

2.3.3 Parallelization

The SOM has not been parallelized yet. However it uses an automatic optimizer to select the model hyper-parameters. This optimization can be performed in parallel, testing different parameters at the same time. A fully parallel version of the SOM algorithm exists and we will explore the possibility of deploying it in our code.

The GMM algorithm has been parallelized in AIDApy. However this is an embarrassingly parallel algorithm: each MPI process calculates the GMM of the particles in a subdomain of the computational domain. There is no requirement for data transfers among the MPI processes. Only one gathering of data is performed at the end of the processing to create the output results. Currently this data gathering step is extremely inefficient: all processors transfer their data to a single master processor, which then performs the IO. In DEEP-SEA we will update the IO routines to avoid this bottleneck.

Other algorithms based on the PyTorch framework use the routines implemented by the developers for GPU offloading and parallelization.

Accelerators

Accelerators are used by PyTorch in some of the algorithms implemented in AIDApy. However, the SOM and the GMM have not been implemented for use in the GPUs. We will explore the possibility of implementing the SOM on GPUs. However, we expect that accelerators will play a much interesting role on the execution of large neural network models with PyTorch.

We are currently developing new algorithms that might be used for the analysis of solar images and particle data. These models make extensive use of PyTorch and we hope that we will be able to test them during the DEEP-SEA project. Even if our current goal is to use the SOM and the GMM algorithms, the work of ML/AI is rapidly changing and we will keep updating our ML tools.

2.3.4 Typical use case

Current simulations of our PIC codes produce particle files of around 50 GiB. On the test cases used during the DEEP-EST project we used 2 nodes of the DEEP-EST Cluster (2 Intel Xeon 'Skylake' Gold 6146 processors per node, with 12 cores per processor) to analyze such data in a few minutes. This type of use case is perfect for debugging and testing of the DEEP-SEA systems.

2.3.5 Target use case

We are planning to perform simulations that produce more than 1 TiB of data. This data will not be saved to disk but will be transferred directly to the *Data Analytics Module* of the MSA where the GMM code will perform the analysis in parallel on each of the subdomains of the simulation.

2.3.6 Application requirements

Scaling

The two different ML models that we are testing, the SOM and the GMM, have different performance metrics. We would like to verify the strong scaling of the SOM algorithm, because the size of the problem (the amount of data) is in general known and constant. The goal would be to analyse large amounts of data in a much shorter time using the MSA. On the other hand, the GMM algorithm will be able to demonstrate weak scaling, following the weak scaling targeted by the xPic application. In principle this scaling should be straightforward, as there is no communication or synchronization among processors until the final step of the code.

Compute

The SOM and the GMM models have been coded using routines that rely heavily on the CPUs. We expect to make use of CPU nodes for these applications until GPU versions will be available. These models depend on single thread performance. On the other hand we will test the use of advanced neural networks with PyTorch, which allows to perform fast calculations using GPUs.

Memory

The SOM and the GMM models are applied mainly to data that does not require large amounts of memory to store. However, other applications using neural networks, images of the Sun, and 6D satellite data, might require larger and faster memory, as the one provided by the current generation of GPUs.

Let us use the example of the study of images of solar active regions: in this application a large catalog of images of the Sun is used to train a neural network autoencoder. Each image is on average of size 512 x 512. These scientific images can contain up to 10 different channels. Each image is then a matrix of 512 x 512 x 10. We have a catalog of multiple thousands of images. In general the training of such systems is performed using mini-batches in order to fit in memory the least amount of images for the training. These batches have in general a size of around 128 images. This means that each batch of images, occupying a total of 2.68 GiB, has to enter and exit the system memory as fast as possible, so the multiple training iterations of the neural network can be calculated.

IO

For data IO the main algorithms SOM and GMM do not require special attention, unless the techniques are applied to much larger and higher-dimensional data sets. In the current use case scenarios we are not expecting to stress the IO system with our algorithms.

This can change for applications where images or 6D data will be used. In such context the reading time of the data used during the training of the ML techniques might be a bottleneck and might require the use of specialized memory. In the use case explained in section 2.3.6, using neural network autoencoders to study the Sun, we use a very large catalog of images that can occupy multiple TiB of disk space. The data contained in these catalogs needs to be loaded from disk into memory as fast as possible. This can require the implementation of multiple levels of data handling.

Malleability and resilience

The code AIDApy is currently not very malleable. In its current state it relies heavily on CPU architectures. GPU versions of the models are slowly emerging but is more possible that malleable applications will emerge from the complementary applications that use neural networks. The PyTorch framework can be launched both in CPU and GPU architectures, using NVIDIA or AMD hardware.

3 The IFS weather forecasting software

3.1 Introduction

The European Centre for Medium-Range Weather Forecasts (ECMWF) is an international organisation supported by 34 states: 23 members (Austria, Belgium, Croatia, Denmark, Estonia, Finland, France, Germany, Greece, Iceland, Ireland, Italy, Luxembourg, the Netherlands, Norway, Portugal, Serbia, Slovenia, Spain, Sweden, Switzerland, Turkey and the United Kingdom) and 11 co-operating members (Bulgaria, Czech Republic, Hungary, Israel, Latvia, Lithuania, Montenegro, Morocco, North Macedonia, Romania and Slovakia). ECMWF's principal objectives are the preparation, on a regular basis, of medium-range and long-range weather forecasts for distribution to the meteorological services of the member states; scientific and technical research directed to the improvement of these forecasts; the collection and storage of appropriate meteorological data.

The Integrated Forecasting System (IFS) is the weather forecasting suite jointly developed by ECMWF and Météo-France. The suite is currently available to the National Meteorological Services of its member states, and research collaborations are pursued with a range of European research institutions, both via participation in EU framework projects and via specific projects with individual institutions (e.g. ESA, EUMETSAT).

The IFS software has been developed for over 30 years and has been run on various hardware architectures, such as shared and distributed memory machines, vector and scalar multi-core processors.

It is run operationally at ECMWF, currently on a Cray XC40 system. Products generated and disseminated to member states include medium-range forecasts updated twice-daily, and longer-range monthly and seasonal forecasts updated at a lower frequency.

3.2 The IFS

Coupled global forecasts carried out with the IFS suite consist of 3 main components, namely the atmosphere, covered by the IFS itself, the ocean, represented with NEMO, ("Nucleus for European Modelling of the Ocean"), and the ocean-atmosphere interface, covered by WAM, the ECMWF WAve Model. Dependencies between the three components are illustrated in Figure 7.

The IFS dynamical core is based on a spectral semi-Lagrangian algorithm with semi-implicit time stepping. The spectral transform method involves discrete spherical harmonics transformations between physical (grid-point) space and spectral (spherical-harmonics) space. Figure 8 illustrates the operations carried out during a time step of the atmospheric model, and their implementation in either grid-point or spectral space.

Spectral transformation capabilities are provided by an internal library. Grid-point space computations include in particular the physical parameterizations that contribute tendencies from subgrid-scale processes.

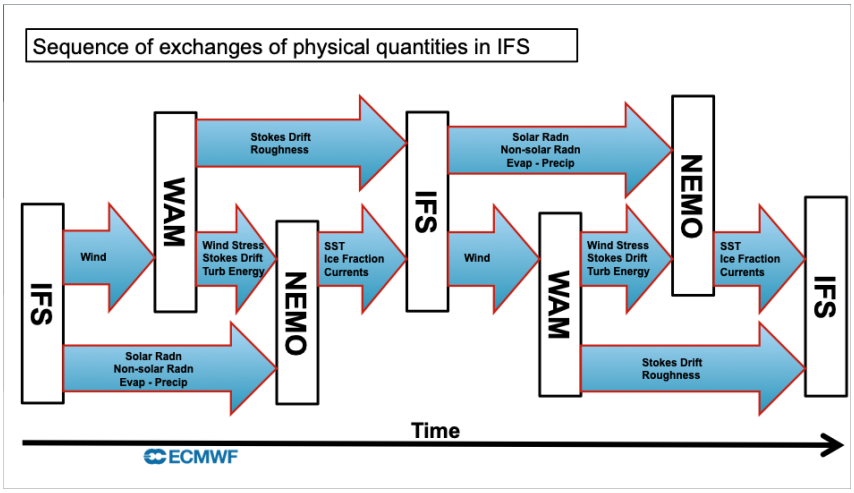


Figure 7: ECMWF Two time-step view of coupling sequence and exchange of physical quantities between the atmospheric, ocean wave and ocean models

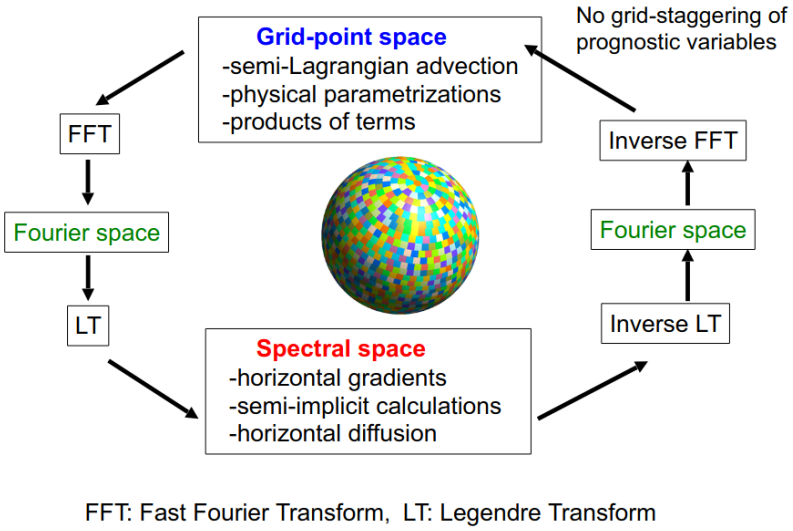


Figure 8: Schematic of a time step of the atmospheric model

3.2.1 Programming languages

The IFS is a large software package, written mostly in Fortran (90-2008), but also including some C. Some ECMWF dependencies are written in C++.

It is a large code base, containing more than 3 million lines of Fortran, distributed in over 7,000 source files.

3.2.2 Libraries and other dependencies

The IFS stack has a moderate list of third-party software requirements. Software requirements are as follows:

- Compiler suite with support for F2008, C, C++11, OpenMP.
- MPI-3
- NetCDF 4
- HDF5
- BLAS / MKL
- Python3

3.2.3 Parallelization

The code is distributed both in physical space and spectral space thanks to MPI. A combination of point-to-point and global communications are used, and buffered or not, blocking or non-blocking communication modes can be chosen at runtime.

Shared-memory parallelism is exposed at a very coarse granularity with OpenMP, and atmospheric quantities are stored in cache-friendly blocks allowing efficient multi-core usage, and thereby reducing the maximum number of MPI tasks required to fill a machine.

A critical communication pattern in the IFS is found in the data transpositions, required to go from gridpoint space to spectral space, and back again. They are implemented via MPI alltoallv calls, which constitute an intrinsic limit to scaling.

Accelerators

The IFS is undergoing considerable effort in order to target x86 architecture alternatives.

As part of this effort, the spectral transforms library has been ported to NVIDIA GPUs with a combination of OpenACC directives and CUDA library calls for linear algebra (CUBLAS) and fast Fourier transforms (CUFFT). An OpenMP-offload port of the transforms will be undertaken independently from, but during the course of, the DEEP-SEA project.

In a similar fashion, an OpenACC port of the radiation component of the IFS is currently being undertaken, and should become usable within the first year of the project.

The physical parametrizations of the IFS, which represent a significant fraction of the overall cost of a timestep, are not ideally suited to a one-off GPU port, as was done for the spectral transforms. This comes from the number of lines of code involved, the number of scientists involved, and the frequency at which individual parametrizations are modified.

In light of this, a different avenue is being pursued, with the goal of being able to generate tailored architecture-specific implementations of the parameterizations, at compile time. This approach relies on compiler-like source-to-source tooling being developed at ECMWF, which allows deep analysis of the existing code, and implementation of transformation recipes, such as array and loop order flipping, acceleration directive insertion, and more.

3.2.4 Typical use case

The typical use case for the IFS in DEEP-SEA will be a coupled forecast with a TCo639L137 atmospheric grid, corresponding to an average horizontal grid spacing of 20km and 137 vertical levels. This is large enough to offer good scaling properties up to $\mathcal{O}(100)$ nodes, yet can fit in the memory of 12 DEEP-EST CPU nodes.

3.2.5 Target use case

If resources become available to do so during the course of the DEEP-SEA project, our exascale-class test case would be a coupled forecast on a TCO7999 grid. This would correspond to an average horizontal grid spacing of 1.4km, and would require of the order of 300 TB of RAM.

3.2.6 Application requirements

Scaling

Thanks to the combination of numerical accuracy and stability demonstrated by the semi-implicit semi-Lagrangian spectral dynamical core, the IFS serves as a reference in terms of global forecasting efficiency. Scaling studies have been performed on a number of large GPU-enabled machines. As an illustration, Figure 9 shows scaling plots established on Titan, Summit, and Piz Daint, at very high

spatial resolutions of 5 km and 1.4 km, or 2 times and 6 times the current ECMWF operational HRES forecast resolution.

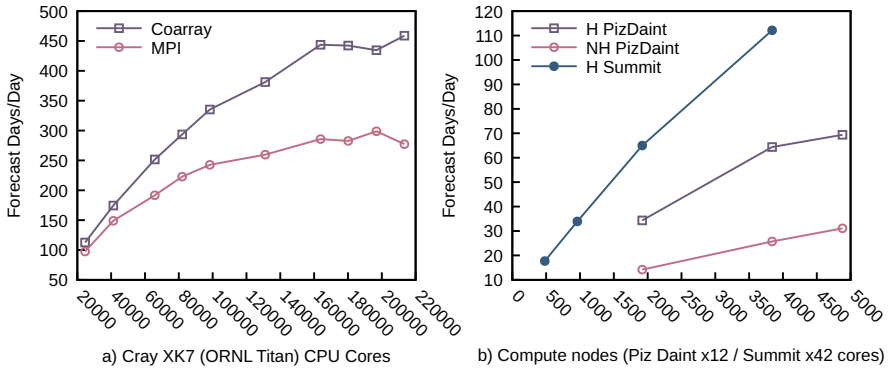


Figure 9: Left: IFS model scaling runs on Cray XK7 Titan at 5 km horizontal resolution (TCo1999L137) comparing baseline (MPI) and Fortran2008 co-arrays (coarrays) implementations
Right: Speed-up on Summit and on Piz Daint for the 1.4 km, 62-vertical level (TCo7999) hydrostatic (H) and non-hydrostatic (NH) IFS

As mentioned in Section 3.2.3, the principal factor limiting the scaling properties of the IFS is the global communications required in the spectral transforms. Figure 10 illustrates strong scaling characteristics of the transforms, established on the Summit machine consisting of IBM Power9 CPUs and NVIDIA Volta GPUs. The GPU results were obtained with the GPU port based on OpenACC directives, and CudaDGEMM and CudaFFT library calls. Despite the influence of the global communication, performance continues to increase up to nearly 2000 CPU nodes, and for the GPU version, 11520 GPUs. This underlines the dependency on having a high-bandwidth, low-latency interconnect.

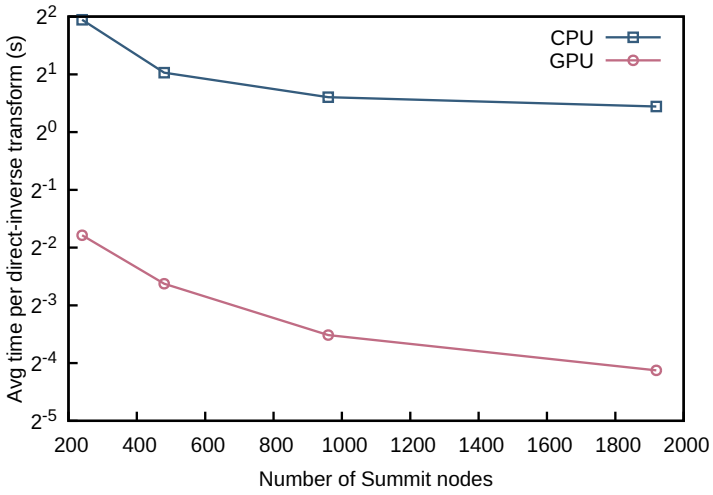


Figure 10: Scaling of IFS spectral transform on Summit using a hybrid OpenMP/OpenACC/MPI configuration, GPUdirect for MPI_alltoallv and CudaDGEMM/FFT libraries

Compute

Despite efforts to make IFS run efficiently on accelerators, as mentioned in Section 3.2.3, many optimizations explicitly aimed at execution on x86 architectures are present in the source. This includes data structures arranged in cache-optimized block layouts and vectorization-friendly loop orders. Various components of the IFS exhibit different performance characteristics. Although communication latency in the spectral transforms determines the strong scaling limit, before that there is no single component responsible for the majority of runtime.

Memory

The typical use case should be able to run on a minimum of 12 DEEP-EST CPU nodes, while the exascale-class problem would require $\mathcal{O}(300)$ TB of RAM.

IO

Application IO is performed by the ECMWF ECCODES library, in (compressed) grib format. To reduce the impact of IO latency, dedicated MPI ranks, either co-located with compute ranks or on separate nodes, are used to aggregate distributed field data before IO processing. Total size of input files for our typical use case on a TCo639 grid is about 8 GB and output is in the order of 25 GB per simulated day. For the target use case on a TCo7999 grid this increases to more than 3 TB per simulated day.

Malleability and resilience

Although some GPU offload capabilities are already available in the IFS, the software is currently not able to adjust to available resources at runtime. The IFS is able to write (bespoke) checkpoint files at a user-configurable frequency. This gives the application a degree of fault mitigation sufficient for operational requirements, which mandate a relatively short overall runtime and justify only limited resiliency overhead compared to re-running a simulation.

4 Seismic imaging

4.1 Introduction

Seismic depth migration algorithms calculate images of the Earth's subsurface from the measured and pre-processed seismic reflection data. These images deliver important pieces of information to the geoscientist for discovering oil and gas reservoirs. The method of Reverse Time Migration (RTM) stands out by high imaging quality even in case of high geological complexity. Solving the wave equation (rather than high-frequency approximations to this equation) realistically simulates the propagation of waves through the subsurface and allows the exact imaging of structures with strongly contrasting seismic velocities as they occur, e.g., for salt bodies.

With seismic imaging algorithms, one essentially solves a forward problem, i.e. one models the full seismic wavefield in the presence of a given subsurface model which needs to be provided as input and which is initially generated from some geophysical intuition. However, ultimately, the geophysicists actually want to solve the inverse problem, i.e. one generates the subsurface properties directly from the preprocessed reflection data.

The full waveform inversion (FWI) solves such an inverse problem and uses the adjoint method to pose an iterative improvement of the physical model, typically parameterised in terms of wave velocity and density. As such, during the optimization process several RTMs are performed. While the Fraunhofer Reverse Time Migration (FRTM) is implementing a RTM, the Barcelona Subsurface Imaging Tools (BSIT) implements a FWI using different parallelization strategies and optimization approaches.

4.2 FRTM

FRTM implements the RTM method. By using proprietary HPC tools, FRTM is a massively parallel and robust implementation of RTM that combines the necessary degree of accuracy with ultimate efficiency.

The seismic reflection data comes as a set of single measurements, so called shots illuminating different possibly overlapping subsurface regions. The typical input survey size is of the order of 10000s of shots. For a single shot computation, one has the source signal and the pre-processed seismic reflection data as input whose typical size is of the order of a few megabytes. The source signal is propagated forward in time. Afterwards, the receiver wave field is propagated backward in time. For every physical time step, the zero-lag cross correlation of the source and receiver fields has to be determined to form the final single shot image. Since the source and the receiver wave fields get computed in different directions in time, one usually uses a checkpointing approach in order to store the source wave field at certain time steps. These checkpoints are used to compute the source wave field on the fly at intermediate time steps to be able to compute the cross-correlation. The wave equation modeling for the propagation of the source and receiver signals is based on a Time Domain Finite Difference (TDFD) discretization on a regular grid. The final image is constructed from single shot results by stacking.

4.2.1 Programming languages

FRTM is completely written in C++.

4.2.2 Libraries and other dependencies

FRTM uses GPI-2, the Global Address Space Programming Interface (GASPI) reference implementation and pthreads.

4.2.3 Paralleliation

Along the computation of the final result, there are two levels of parallelism involved. The upper level computes a set of shots concurrently and evaluates partial sums, i.e. stacks the final result. This is trivially parallel. On the lower level a single shot computation is distributed over several compute nodes by domain decomposition. The non-local structure of the stencil operator requires boundary data to be exchanged. This introduces a tight coupling between the subdomains and makes this part of the parallelization non-trivial. Using a fine-granular decomposition of the single shot migration output volume in combination with dynamic load balancing across several threads on the CPU level, we yet achieve a high parallel efficiency. As communication library GPI-2 has proven to be best suited in order to maximally overlap the communication by the computation with minimal overhead. Scaling out to many thousands of compute cores nearly ideal scalability of the migration can still be observed.

Shared memory parallelization The shared memory parallelization is done using the Asynchronous Constraint Execution (ACE)-framework. ACE is an industry proven C++ library for the description and efficient execution of task based algorithms for x86 CPU based shared memory systems inside of a NUMA domain.

Figure 11 shows the components of ACE. The API provides an abstract interface for the definition of tasks and for the construction of a task graph. The task graph describes the data dependencies between the tasks of a given algorithm.

A pthread-based task scheduler manages the cores provided by the CPU. A thread is started once, at the beginning of the application, for every core. ACE provides different pinning strategies in order to increase data locality. The task scheduler extracts the tasks which are currently executable and assigns them to free compute resources. The topology of the task graph is static along execution. There is no dedicated scheduling thread. The check for preconditions to execute a given task and the synchronization between threads is lightweight and does not produce significant overhead.

ACE is especially suited for iterative methods in which one and the same task graph is executed over and over again for every iteration. In ACE, the task graph is stored for a single iteration only and every task keeps separately track on its iterations by an internal state variable. This allows to get rid of a global synchronization point terminating every iteration. Different iteration states can be represented within the same task graph.

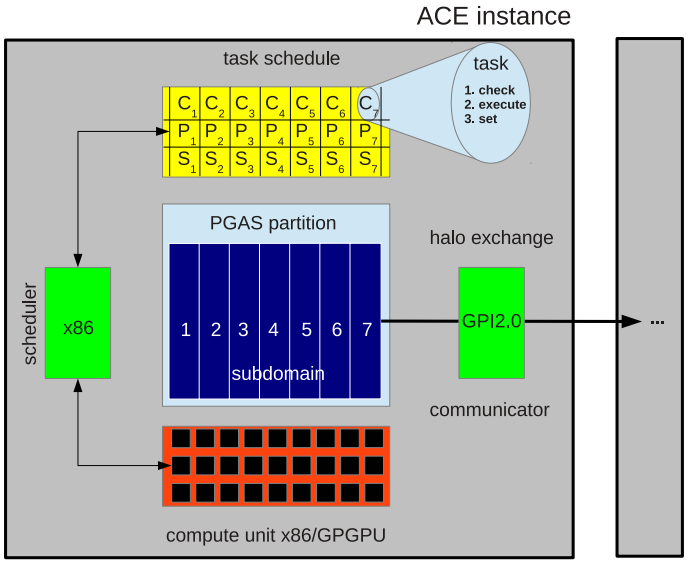


Figure 11: ACE for task based data dependency driven execution.

Distributed memory parallelization The distributed memory parallization is done using the Global Address Space Programming Interface (GASPI) API. GASPI provides single-sided communication primitives which are supplemented by a lightweight synchronization mechanism within a Partitioned Global Address Space (PGAS). As such, GASPI allows for a fine-granular control of the actual data transfer and the subsequent data synchronization.

The finite difference discretization of the wave equation finally ends up in a stencil update. This introduces dependencies among the sub-domains which are generated along the partitioning of the problem. For the update of a boundary point remote information is required. The three-dimensional data structures provided by FRTM contiguously integrate halo regions which incorporates the required remote information. The data structures are allocated directly within the local partition of the global address space. Communication along the slowest memory direction is done directly, i.e. without any additional memory copies. For the other directions, the data to be communicated is non-contiguous in memory. Here, it is advantageous to provide explicit communication buffers within the global address space to which the data can be packed before the transfer and unpacked after the transfer. This generates additional overhead, however, the data can be communicated within a single message which is more optimal with respect to communication latency.

Double buffering is used for the actual 3D data structures and communication buffers, i.e. the data structures are duplicated and used in an alternating fashion between the iterations. This, in combination with the symmetric communication pattern generated by the symmetry of the stencil operator allows to get rid of a lot of explicit synchronizations along the communication like e.g., the synchronization that the target or source buffer for communication can be overwritten, such that the critical path for communication and synchronization in FRTM is a single transfer. These explicit synchronizations can be avoided since they are implicitly guaranteed by the inherent dependencies provided by the algorithm.

Accelerators

Currently, there is no explicit accelerator support in FRTM.

4.2.4 Typical use case

A typical use case migrates a selection of $O(10000)$ shots. The typical grid sizes are of the order of 1000^3 grid points resulting in a minimal memory footprint of 24 GB for a single shot computation in single precision.

4.2.5 Large use case

With seismic imaging algorithms, one essentially solves a forward problem, i.e. one models the full seismic wavefield in the presence of a given subsurface model which needs to be provided as input and which is initially generated from some geophysical intuition. However, ultimately, the geophysicists actually want to solve the inverse problem, i.e. one generates the subsurface properties directly from the preprocessed reflection data. Evaluating the structural properties using RTM is typically done in a manual optimization process of the subsurface model. This includes several cycles of model building and subsequent imaging using RTM. The image produced by RTM yields feedback for improvement in the next modeling cycle. For the seismic expert who is adjusting the model parameters, it is desirable to have updated results available immediately. The same holds for semi-automatic generation of the subsurface model via Full Waveform Inversion (FWI) which essentially minimizes the least square misfit between measured and modeled data. Ordinary optimization methods are used to achieve that. In order to get to the final solution, a lot of RTMs ($\sim$100) have to be performed. In combination with an ever-increasing demand for more detailed spatial subsurface resolution this yields easily a factor of 1000 in required compute power in comparison to today's petascale systems on which current RTMs are running. In RTM, the resolution is adjusted by an upper limit of the frequency which is contained in the simulation. The time to solution for a single-shot migration scales with the fourth power of the target frequency. For example, increasing the spatial resolution by a factor of two results in an increase of a factor of 16 in the required computational effort.

4.2.6 Application requirements

Scaling

A representative benchmark which mimics industry use cases has been performed on SuperMUC at LRZ on Sandy-Bridge utilising up to 4096 compute nodes.

Figure 12 shows the speedup achieved for a single shot computation using the TTI kernel. The time to solution is dominated by the wave propagation kernel, which turns out to be bandwidth limited. The speedup is measured in units of shots computed per hour. Each compute node includes 2 sockets with 8 cores each, making that a total number of 65536 cores used at maximum. One process with 8 threads is launched on each socket using GPI-2 for inter-socket communication. The single shot

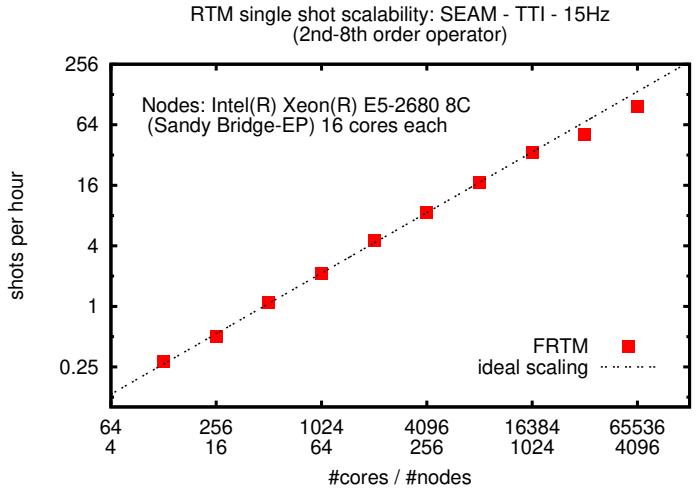


Figure 12: TTI single shot scalability.

compuation scales over almost three orders of magnitude and shows the efficient parallelization of the FRTM single shot computation.

Compute

FRTM uses fully roofline model optimized finite difference stencils of various orders for solving the wave equation for isotropic, VTI, and TTI velocity models in order to maximize the throughput measured in stencil operations in seconds. A SIMD abstraction layer provides a common interface to several SIMD instruction set backends, including SSE and AVX. The number of stencil operations which have to be performed on a single process is proportional to the number of process-local grid points, $N = N_x \cdot N_y \cdot N_z$, times the number of time steps to be performed. This typically depends on the maximum depth to be imaged and may differ for the source wave field as well as the receiver wave field.

Memory

The memory consumption is proportional to the number of process-local grid points, $N = N_x \cdot N_y \cdot N_z$, multiplied by the number of arrays which are required for the simulation. This is a combination of the number of simulation fields N_{field} and the number of model parameters N_{model}. This has to be supplemented by the memory required for the communication buffers, $N_{halo} = 2r\,(NxNy + NxNz + NyNz)$. Hence, the total memory consumption is given by $N_{memory} \propto (2 \cdot N_{field} + N_{model})\,N + \cdot N_{halo}$.

IO

FRTM uses custom designed random perturbation boundary conditions which make the software free of any IO of intermediate results that would typically be of multi-terabyte scale for each of the 10000s of shots. This IO is quite often a major reason why other implementations fail in computing large scale problems. In contrast to that, FRTM only needs to read the input data at the beginning of the computation of a shot and needs to write the final image at the end. For this, FRTM does not use any special IO library. The file format which is used is SEG-Y and/or SU.

Malleability and resilience

The application can run on x86 resources. It can dynamically include or release compute nodes after it has been launched, i.e. while it is running. A failed shot computation is automatically rescheduled up to three times. After the third failure, the shot is marked as failed and is no longer rescheduled. If the job fails completely, FRTM can be restarted automatically from the last consistent state available.

4.3 BSIT

For geophysical inverse problems, BSIT includes fully operational seismic inversion capabilities. These are based upon the full-waveform inversion (FWI) algorithm which uses the adjoint method to pose an iterative improvement of the physical model, typically parameterized in terms of wave velocity and density, see Figure 13. BSIT is a production-ready package that has been tested on many hardware platforms (with both CPU and GPU kernels).

Seismic data comes from active surveys, where artificial sources (shots) generate seismic waves which interact with the subsurface and are recorded at a group of receivers. Each simulation of a single shot, which could be a forward modelling or inverse modelling case, and is independent from each other. Thus shots are executed by binaries which we call kernels. A typical run can involve hundreds of thousands of kernel calls, which take more than 90% of the compute time. The efficiency of kernels and the distribution of kernels among the available computational resources is thus paramount to the proper execution of BSIT.

In FWI, or inversion mode, a kernel performs two wavefield simulations per shot, such as in FRTM. The forward run simulates a shot and records the synthetic traces at receiver locations. The backward run uses the differences between data and synthetics as sources for the simulation and runs backward in time. The wavefields of forward and backward simulations must be correlated in order to produce the main output of the FWI kernel: a 3D gradient of the parameter inverted. The wavefields correlated need to be previously stored and read inside the kernel execution and later can be removed before the kernel finishes. In this regard, the behaviour of a single FWI kernel is similar to that of FRTM. The gradients from each kernel need to be stacked, or summed pointwise for all shots. The gradient provides the direction of change in the model within the inversion process. Together with few test runs (between two and six typically) the model becomes updated: we have performed a single iteration of FWI. Optimization requires about 15-30 iteration steps per frequency band used and typical runs use 3-10 frequency bands.

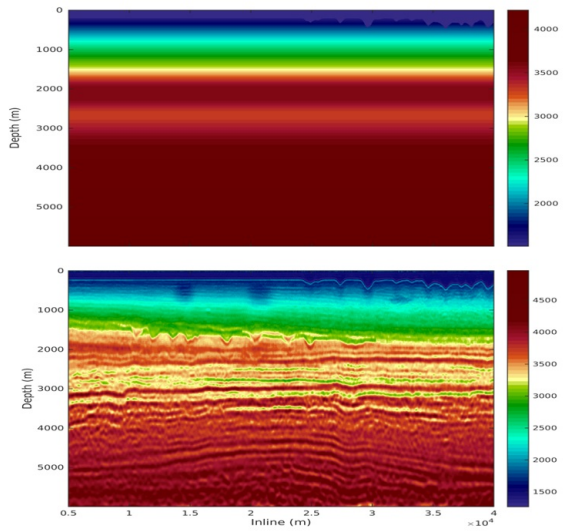


Figure 13: Top: initial model of velocity. Bottom: Result of BSIT's FWI

BSIT is a proprietary code with a high level of maturity and several other configurations besides FWI. Therefore, within DEEP-SEA all experiments and optimization approaches will use a BSIT analog referred to as BSIT Mockup. This code was built to emulate the characteristics of BSIT (e.g data workflow, numerical schemes, parallel paradigms, etc) without the functional capability of the original BSIT code. Furthermore, the functions of task distribution use a different paradigm from BSIT, which is based on a master–worker approach. The Mockup is easily configurable and modifiable, thus providing an ideal environment to investigate different optimization and porting strategies. Specifically, during the project we will explore storage and memory usage technologies on novel architectures such as Xeon Phi, Post-K Fujitsu; Cascade Lake with Optane or servers with heterogeneous memory.

Numerical scheme Seismic waves are simulated solving a partial differential equation, in general elastodynamics. BSIT uses a first-order version of the PDE which reads as follows [15].

$$\begin{pmatrix} \partial_t \sigma_{xx} \\ \partial_t \sigma_{yy} \\ \partial_t \sigma_{zz} \\ \partial_t \sigma_{yz} \\ \partial_t \sigma_{xz} \\ \partial_t \sigma_{xy} \end{pmatrix} = C \begin{pmatrix} \partial_x u \\ \partial_y v \\ \partial_z w \\ \partial_z v + \partial_y w \\ \partial_z u + \partial_x w \\ \partial_x v + \partial_y u \end{pmatrix} \tag{4.1}$$

$$\begin{aligned} \rho \partial_t u &= \partial_x \sigma_{xx} + \partial_y \sigma_{xy} + \partial_z \sigma_{xz} \\ \rho \partial_t v &= \partial_x \sigma_{xy} + \partial_y \sigma_{yy} + \partial_z \sigma_{yz} \\ \rho \partial_t w &= \partial_x \sigma_{xz} + \partial_y \sigma_{yz} + \partial_z \sigma_{zz} \end{aligned} \tag{4.2}$$

The variables are six different stress components (three compressional stresses σ_{xx}, σ_{yy} and σ_{zz}, and three shear stresses σ_{yz}, σ_{xz} and σ_{xy}). Stresses induce material instantaneous strains that are

quantified in terms of displacements, in particular the displacement time derivatives or velocities u, v, and w in the x-, y- and z- directions, respectively. The parameters of the equations are C, which is a symmetric 6×6 matrix, referred to as the stiffness matrix, and the density ρ. Such parameters are constant but may vary spatially in heterogeneous media.

Symmetry of the stiffness matrix C results in 21 potentially independent values. When additional symmetries are present, fewer values are needed to populate C. For example materials known as monoclinic, orthorhombic and transversely isotropic, display 13, 9 and 5 independent parameters, respectively. A frequent case is that of isotropy, where two parameters suffice to populate C or even acoustic behaviour (e.g. the one used in FRTM) where a single parameter is used, and further simplifications can be made in the PDE. In this work, we will use a full anisotropic assumption, often referred to as triclinic, of which all other material types are particular cases. We remark that using such an assumption we can also include realistic topography in our simulations, as explained in [16].

As a solver we use explicit time-domain staggered-grid finite-differences (FD). The spatial finite differences use a fully-staggered grid (FSG, see e.g. [17]) approach, which builds FD cells that include 4 full stress representations and 4 full velocity representations (i.e. a total of 36 variables per cell) and one full set of parameters (i.e. density and C, totalling 22 parameters per cell) as depicted in Figure 14. Materials are stored in a single vertex of the FSG cell albeit they must be present at all vertices at compute time. With this strategy we save memory, trading storage per computation. Our time scheme is order 2 in time, of the leapfrog type. This means that stresses and velocities are defined half a time-step away from each other. In this way central differences are used to solve the PDE with accuracy. Regarding the spatial integration, our solver uses order 8 finite differences, which means that our stencil reaches 4 cells in each direction, both forwards and backwards. Within ABC buffer layers we revert to order 2 accuracy and, whenever a free-surface condition is present, we use specific asymmetric stencils known as mimetic stencils (see [16]).

4.3.1 Programming languages

BSIT is exclusively written in C.

4.3.2 Libraries and other dependencies

BSIT uses multithreading-enabled MPI, OpenMP, OpenACC and CUDA. There have been forks using OmpSs as well. Furthermore, for FFT calculations BSIT uses either FFTW3 or MKL depending on the platform.

4.3.3 Parallelization

There are several parallelization levels in BSIT, at different scales of granularity. These are summarized in the following.

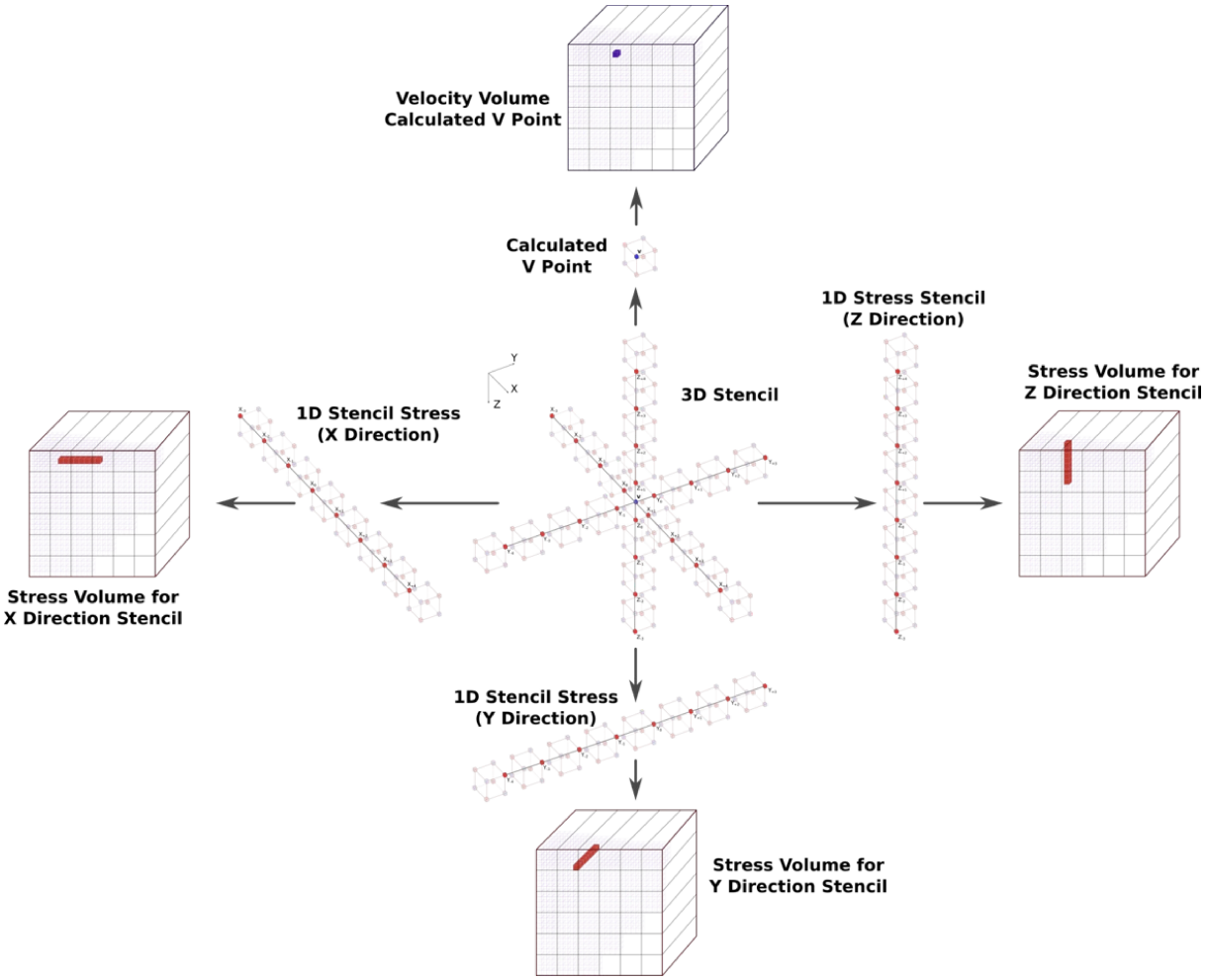


Figure 14: 3D finite-different stencil for each FSG cell.

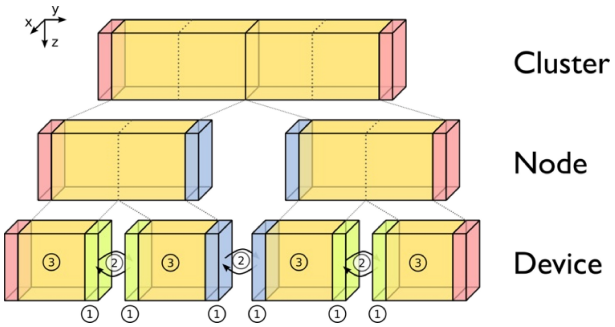


Figure 15: Domain decomposition strategy at kernel.

Shared memory parallelization Starting at the innermost part of the computation, the kernel updates velocities using 12 different 3D stencils plus another 12 3D material interpolations for each grid cell. On the other hand, the computation involves 24 3D-stencils for stresses plus 84 3D interpolations for the material properties to update stresses. We remark that material interpolations are 2nd order accurate and hence much less expensive than the variable stencils, which are high-order accurate. This, in turn, results in both velocities and stresses calculations being typically dominated by accesses to main memory to retrieve the data needed to update the corresponding values. The resolution of the PDE consists in two outermost loops which are collapsed and parallelized using OpenMP, leaving to each thread a number of consecutive memory regions to update. Using this approach, each thread is pinned to each CPU core. Physically, CPU cores are divided in two different sets, each set belonging to one CPU socket. Logically, this division can also be seen as different NUMA nodes. By adding one more level of parallelism using MPI and allocating one MPI task per socket (or NUMA node) we are able to pin each OpenMP thread to the core closer to the memory being used by this thread. Also it worth mentioning that in most program implementations, developers leave the default OpenMP loop scheduling algorithm set. In OpenMP, a static scheduling has not necessarily reported the best performance values. Our previous studies, including static, dynamic and guided scheduling strategies, point out that dynamic scheduling is the best option to achieve the highest performance in our case.

Distributed memory parallelization For the scenarios where a shot (i.e. a kernel) does not fit in a computational node in terms of memory, we use domain decomposition. As occurs with the FRTM application, in those cases, the global domain of the kernel is divided into small sub-domains and computed separately onto different computational nodes or accelerators. MPI routines are used for such inter-domains communication, including exchange, gather and scatter data when needs. To a naive domain decomposition strategy, we have added several improvements towards better scalability. At each simulation time step, we split all compute functions into two stages, which are functionally identical but update different regions of each domain. In particular we divide regions whose update depends only in local values (i.e. memory in the current domain or NUMA node) and regions that need updated information from neighbouring domains in order to be updated. Moreover, we restrict domain decomposition to a single dimension (namely y in our case), hence the topology of communications is much simpler and each buffer region relates to one neighbour at most. Such an approach allows for overlapping memory transfers and computation when using a domain decomposition strategy. Phase 1 computations perform the update of cells which belong at the buffer regions. Then, at Phase 2,

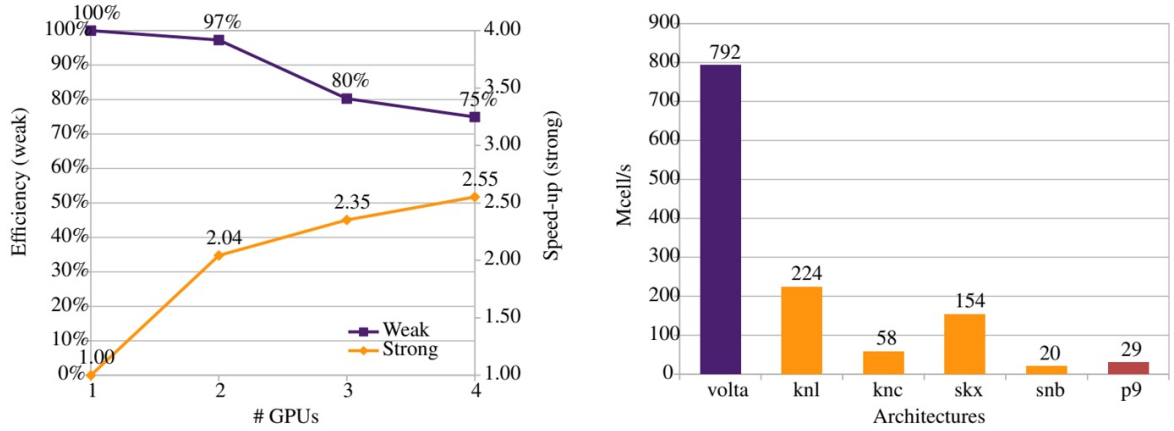


Figure 16: Weak efficency in GPUs (left) and efficiency in Mcells/s for different architectures (right).

these updated values can be transferred to neighbouring domains simultaneously (asynchronously) while updating the rest of the cells in each domain (see e.g., Figure 15). Moreover, as part of the snapshot correlation step, we need to either dump the value of the snapshots to disk (at the forward stage) or read the previously stored snapshots from disk (at the backward stage). Such temporary data storage can result in large I/O bottlenecks on the overall kernel execution for large runs. Our baseline version of the code overlaps both I/O and compute operations which mitigates the problem and speeds up the code execution significantly.

We remark that BSIT, and not its Mockup, includes a workflow manager based upon a master–worker paradigm with full checkpointing/resilience capabilities. This is not present in the Mockup in its current version.

Accelerators

Currently, the BSIT mockup supports computing on accelerators, using both CUDA and OpenACC libraries. Tests include KNL, KNC as well as Volta and P9 NVIDIA GPUs, see Figure 16.

4.3.4 Typical use case

A typical use case inverts a model with a selection of $O(1000)$ shots. Notice that, contrary to migration, FWI is a complex iterative process where more than 100 iterations are expected to do and where each iteration executes a migration-like simulation with the whole set of shots and between 2 and 6 modelling simulations (that do not perform intensive I/O), also with the complete set of shots.

FWI executions boast an outermost loop that controls frequency, typically cycling from low- to high-frequencies. Wavefield simulations as those present in BSIT Mockup's kernel increase by $\times 8$ in memory and $\times 16$ in compute time and temporary storage when we double the frequency. BSIT's Mockup implements an adaptative grid-resize mechanism that take care of projecting results and

interpolating models between frequencies. A typical case results in a footprint between $\sim$15 GB to more than 40 GB for a single-shot computation in single precision.

4.3.5 Large use case

A large case is typically the same as the typical case, only with larger frequencies at the latest iterations. In this case the memory footprint can reach 200 GB per shot.

4.3.6 Application requirements

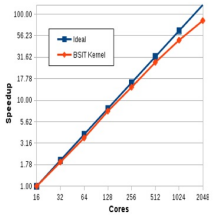
Our application expects a 3D model provided in binary format and data provided in SEG-Y format, as well as metadata related to the geoposition of the model. A global filesystem is expected for IO.

Scaling

At system level there exists embarrassing parallelism between shots or kernels, hence the scalability at this level is almost perfect. It domain decomposition is mandatory, BSIT's Mockup can be efficient if the ratio between inner cells and buffer cells is about 8 or higher. Otherwise the kernel loses parallel efficiency due to communication overheads (see Figure 17).

KERNEL

# cores	Runtime (s)	speedup	efficiency
16	450.88	1.0000	100%
32	236.31	1.9080	95%
64	123.86	3.6403	91%
128	60.49	7.4541	93%
256	31.84	14.1607	89%
512	16.38	27.5310	86%
1024	9.06	49.7816	78%
2048	5.36	84.0430	66%



SYSTEM

# cores	Runtime(s)	speedup	efficiency
128	109303	1	100%
256	54518	2	100%
512	27360	3.99	99%
1024	13716	7.97	99%
2048	6860	15.93	99%

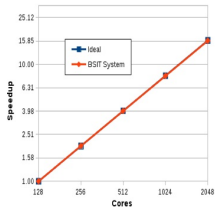


Figure 17: Scalability at Sandy Bridge, both at kernel level and system level.

Compute

Most of the computing time happens at kernel level. The system performs best if kernels can be kept inside a single NUMA node. The system relies on a global access file system to store persistent data and also a scratch disk to dump temporary data.

Memory

The memory consumption is proportional to the process local number of (single precision floating point) grid points $N = N_x \cdot N_y \cdot N_z$ multiplied by the number of arrays which are required for the simulation. This is a combination of the number of simulation fields N_{field} and the number of model parameters N_{model}. This has to be supplemented by the memory required for the communication buffers $N_{halo} = 2r\,(NxNz)$. Hence, the total memory consumption is given by $N_{memory} \propto ((2 \cdot N_{field} + N_{model}) \cdot N + N_{halo}) \cdot float32$.

IO

BSIT is not using any external software library for IO management. For each of the thousands of shots that are executed, 2D (time-space ordered) and 3D ($[N_z, N_x, N_y]$ models) RAW data is read from disk. During each, or few, computed time steps, a snapshot is written to disk that will be read at a future stage. The size of the snapshot is approx. $N_z \cdot N_x \cdot N_y$. The size of snapshots depends on the domain decomposition and the number from the simulation' time steps. The output of the simulation are several 3D (of size $[N_z, N_x, N_y]$) binary files. Additional outputs are possible only for QC purposes.

Malleability and resilience

BSIT has been deployed in many different environments, mainly with x86 architectures but also with IBM Power and NVIDIA GPU accelerators. Domain decomposition cannot be configured automatically and must be set prior to launch the simulation, manually. However, the baseline code can detect NUMA hardware on the fly and optimize the execution setup in run-time to attain a higher performance. Regarding resiliency and malleability (i.e. the dynamic allocation of compute resources), BSIT provides fault tolerance mechanisms at workflow level, including restarts, kernel execution retries, assigning nodes to a black list on-the-fly when those fails and also adding/removing resources on the fly. However, the Mockup (BSIT like-version), given its R&D use, focuses on performance rather than on dependability. Hence no resilience mechanisms are implemented.

5 GROMACS - Molecular Dynamics

5.1 Introduction

GROMACS is an open source framework for Molecular Dynamics (MD) simulations of systems ranging from a few hundred to millions of atoms [18]. GROMACS has been widely used in several biological systems, including recently the COVID-19 virus.

Recent GROMACS versions are written in C++ and have more than a million lines of code. GROMACS is capable of running in heterogeneous computers consisting of multiple nodes with CPU and GPUs. When compared to many other Molecular Dynamics simulation frameworks such as Amber and NAMD, GROMACS exhibits relatively good performance. However, one of the computational bottlenecks, especially towards exascale computing, remains the 3D Fast-Fourier Transform (FFT) used for the Particle Mesh Ewald (PME) calculations.

Within DEEP-SEA, our goal is to improve the performance of the electrostatic solver based on 3D FFT. In particular we aim at integrating into GROMACS a parallel and accelerated 3D FFT library resulting from the work in DEEP-SEA WP4 on the DaCE framework. As the first step of this work, we conduct a profiling study and investigate the scaling behavior, hot spots, and time spent on different modules focusing on the 3D FFT. For the initial profiling study, we have used the Tetralith, a Swedish supercomputer with 1,908 compute nodes (each with two Intel Xeon 16-cores CPUs). The profiling analysis has been carried out using the ARM-MAP software and using the built-in performance counters. For the scalability analysis, we study two systems: one system with approximately 30,000 atoms and one system with 0.85 million atoms. In addition, the performance of different FFT modules has been evaluated as this is one of the significant computing and data transfer intensive segments of the code. In particular we profile the performance of three HPC FFT libraries: FFTW, FFTpack, and MKL-FFT libraries. This performance study will provide a performance baseline we will use for comparing the performance of 3D FFT developed in WP4.

5.2 Molecular Dynamics

MD simulations mimic the dynamics of the particles in a self-consistent field. In a naive implementation of MD codes, the field calculation requires N^2 calculations where N is the number of particles in the system. For this reason, MD calculations can mainly be classified as compute-driven. However, for systems with large sizes, e.g., with millions of atoms, parallel communication also becomes an equally demanding segment. However, improvement of the MD algorithm can be achieved. For instance, it is possible to divide the field calculation into short-range field calculations and long-range field calculations. For the short-range computation, we can consider only the field contribution of particles within a cut-off distance, reducing the number of short-range interactions drastically. For the long-range field calculations, we can use the PME approach. In the PME approach, a grid is introduced in the simulations, and an electrostatic potential is calculated on it. For this, the FFT is used. The PME usage makes the cost associated with electrostatic calculations $\mathcal{O}(N_g \log N_g)$ where N_g is the number of grid points. In this way, the actual computational cost is not of the order of N^2. For the calculations of short-range interactions, the computing nodes need to know the coordinates of the

particles' positions and those within a cut-off distance (usually 15–20 Å). However, the calculations of the long-range part of the interactions require coordinates of all the particles. The computing of such contributions requires all-to-all communications.

In parallel field computation, GROMACS splits the processes into processes to compute the short-range interaction (called Particle-Particle or PP) and processed to compute the PME and the long-range calculations. In DEEP-SEA, we focus on improving the performance of the PME calculations requiring the usage of 3D FFT. The all-to-all type communication in the 3D FFT drastically affects GROMACS performance and the parallel scaling, especially going towards to exascale. Typically, the PME calculations are carried out on one-third of the total number of ranks available. The typical size of the data generated during MD (positions of all the particles and velocities as a function of time stored for carrying out various analyses of various structural and dynamical properties) can be from a few 10s to a few hundred GB depending upon the time interval defined for storing the coordinates/velocities.

5.2.1 Programming languages

The GROMACS software is entirely written in C++ and uses SIMD acceleration on a wide range of architectures. It also supports GPU offloading acceleration and both OpenMP and MPI parallelism within and between nodes.

5.2.2 Libraries and other dependencies

Within DEEP-SEA, we focus on PME calculations with 3D FFT. For computing the long-range part of the electrostatic interactions, the PME method is used, which involves FFT. GROMACS allows the selection of different FFT libraries: FFTW3, FFTPACK, MKL-FFT. To use a different FFT library in GROMACS, an adaptor needs to be developed in GROMACS to factor the GROMACS data layout into the FFT library layout.

The visualization of trajectories is usually performed by using various software such as VMD, PYMOL, and Chimera. The time evolution of various properties (namely energies, RMSD, RMSF) can be viewed using xmgrace and Matlab.

5.2.3 Parallelization

The parallelization in GROMACS is achieved using both OpenMP and MPI on homogeneous super-computers. The particle positions within a cutoff distance are required for the short-range interaction calculations. These data are shared using point-to-point communication between the ranks (referred to as PP ranks). The positions and charges of all particles in the simulation box are required to compute long-range interaction using the PME method. These data are communicated using all-to-all type communication between the PME ranks.

Accelerators

GROMACS has been ported to several accelerator architectures, including NVIDIA and AMD GPUs and IBM Cell accelerator. Different programming systems have been used for programming GROMACS on accelerators, including CUDA. OpenCL, and SYCL. In general, accelerators provide a considerable performance enhancement (mainly on the PP part): previous benchmarking studies show three times faster performance of GPU version when compared to CPU version. GROMACS can offload the computer heavy direct PP interaction calculation to GPUs. However, the PME electrostatics calculation is performed on CPUs. As part of this work, we plan to evaluate the PME performance also on accelerators offloading 3D FFT calculations to GPUs and FPGAs. GROMACS developers at KTH are also working on a version that can be compiled using OpenACC, OpenCL, and HIP programming languages.

5.2.4 Typical and Target Use cases

We study two systems with different numbers of particles and grid cells for the typical and target use cases. The typical use case employs 30,000 atoms, while the target use case employs 0.85 million atoms. These simulation setups are designed to study virus capsids, virus particles (capsid with genome), viral spike proteins in complex with mammalian cell receptors. The system size in these cases can easily reach a few million atoms. Given that more serious viruses appear in nature, it will be important to study such larger-sized systems for designing novel drugs to treat viral infections. So, we study such systems with the help of GROMACS in HPC with the DEEP-SEA architecture. We will investigate the viral spike protein in complex with full human-ACE2 receptor as the target system.

5.2.5 Application requirements

Scaling

We performed scaling tests on the Tetralith supercomputer with approximately 1,700 nodes, Intel Xeon GOLD6130 CPU with 32 cores. In the typical use case (30,000 atoms), the peak performance was observed for 512 cores. In the target use case (0.85 million atoms system) a near-linear scaling in performance with the number of cores was observed up to 8,000 cores. The scaling behaviour in smaller systems should be attributed to the dominant contribution from the communication segment rather than the computation.

Further, the GROMACS installation with FFTW (the default FFT library used during compilation) showed very poor scaling. However, the scaling behaviour and performance of GROMACS with other FFT libraries (such as FFTpack, MKL-FFT) exhibited better performance and scaling properties in the use cases.

Compute

The GROMACS particle mover phase, responsible for the particle trajectory calculations, is compute-bound and can easily take advantage of accelerators, such as GPUs and FPGAs. However, the 3D FFT operation is a memory-bound operation (as it requires transpositions) and is typically carried on CPUs. In this deliverable, we focus on profiling different 3D FFT packages in GROMACS on CPUs as this is the typical usage. However, we plan to develop and test 3D FFT in GROMACS also on NVIDIA and AMD GPUs. Possibly, we will also investigate 3D FFT porting to FPGAs.

Workflow

The workflow involved the following main tasks: (i) testing the scaling behaviour of GROMACS, (ii) evaluating the computational cost involved in the different subprocesses of long-range interaction energy calculations using the PME method, and (iii) evaluating the performance of various FFT libraries, namely FFTW, FFTpack, MKL-FFT for the PME computations.

We have prepared systems of two sizes with approximately 30,000 atoms and 0.85 million atoms, and the systems are heterogeneous in nature where biomolecules, namely lysozyme and spike protein: human ACE-2 complex embedded in water. The short-range interactions are computed by dedicated particle–particle (PP) ranks which compute the interactions using domain-decomposition approach. In this, the whole simulation box (can be cubic, orthorhombic, triclinic) is divided into 3D grids, and these cells (referred to as domain decomposition cells) are assigned different PP ranks. The integration of the equation of motion of the particles within the domain is fully handled by a specific PP rank. The required particle positions from the neighbouring cells are transferred through point-to-point communication. For testing the performance of different FFT libraries, we have compiled GROMACS with various FFT libraries (FFTw, FFTpack, MKL-FFT). The profiling of GROMACS has been done with in-built timers in GROMACS itself. In addition, we have used a profiler, Arm-MAP/21.0.1. The load imbalance due to domain decomposition and PP/PME workload imbalance has been investigated along with the analysis of wall time on different computing segments.

Profiling results

We perform scaling tests for the typical and target use cases. Figure 18 shows the performance of GROMACS with an increasing number of cores. We have used 1–256 nodes, and the number of cores involved in computing was in the range of 32–8,192. The analysis used the in-built timer of GROMACS, which provides a time scale that can be accomplished in a day. The presented results are based on five independent MD simulations. We used 10,000 MD steps for doing this analysis. As can be seen, the system with a smaller size (having 30,000 atoms) showed peak performance for 512 cores. With the use of more cores, the performance goes down as the communication segment of the program starts dominating over the computing segment. So, when the system size is smaller, it is not recommended to use more than eight nodes. However, for the large-sized system, the performance increases monotonically until up to 8,192 cores which shows the efficient parallelization of this GROMACS implementation. However, a poor scaling behaviour is observed for the default FFT library (FFTW), which showed peak performance for 512 cores (as it was found for the typical use case). This study clearly shows the role of FFT libraries in the scaling behaviour, and one has to

Table 1: Performance dependence on the number of grids used in FFT.

Fourier Spacing, nm	nx	ny	nz	Performance, ns/day	Mflops	% Flops
0.12	60	60	60	175.9	440065	2.6
0.06	120	120	120	86.5	5729424	29.5
0.03	240	240	240	27.0	52471581	78.8

be careful about this at the time of compilation. Our goal is to improve the performance of PME and FFT using the DaCE-based FFT library.

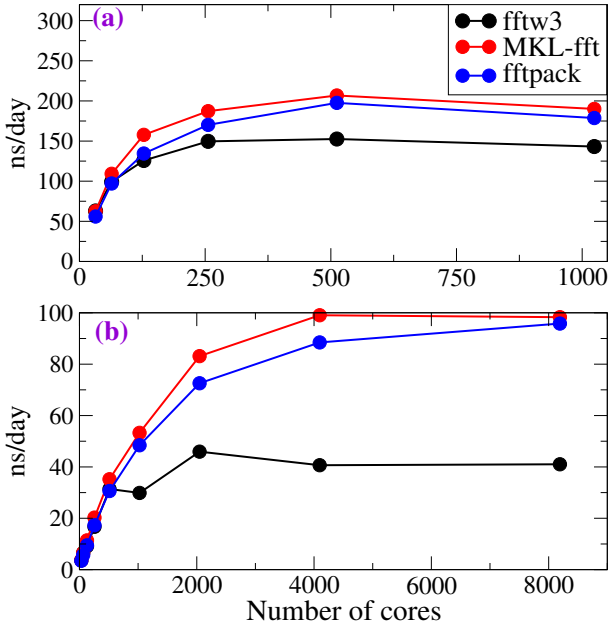


Figure 18: GROMACS parallel scaling on the Tetralith supercomputer.

Given that the computing associated with the long-range part of the interaction is playing a major role in performance, we analysed the cost associated with the number of grid points and CPU time spend on different parts of the PME scheme using FFT. In particular, the long-range interaction involves setting up a 3D grid controlled by the input spacing parameter. The default value in GROMACS is 0.16 nm. Table 1 shows the performance in ns/day and %Flops. As can be seen, the grid spacing parameter plays a major role in dictating the performance and scaling behaviour of GROMACS. When the grid spacing was reduced from 0.12 nm to 0.06 nm, the performance went down by 50%, and when the spacing was reduced to 0.03, the performance went down to 15%. The study shows that one has to be very careful in choosing the grid spacing, e.g., the number of grid points for the FFT calculation. In fact, reducing the grid spacing from 0.12 nm to 0.06 nm does not necessarily improve the accuracy of the electrostatic energy computation.

Table 2: Wall time spent on different segments of PME calculation.

FFT	Send X to PME	PME mesh	PME wait for PP	Wait+Recv. PME F
FFTW	43.5	5.3	8.8	1.7
MKL-FFT	2.1	2.0	22.5	2.2
FFTPACK	0.1	1.7	23.2	1.8

We have also tabulated (refer to Table 2) the wall time spent on data transfer and computing, and the results are presented for three different GROMACS installations. As we have discussed above, the FFTW performed poorly. If we analyse the wall time spent on PME force calculation (presented as PME mesh in Table 2), we can observe that the FFTW version consumes twice more time than the other two cases. The most interesting part is that the wall time spent on data transfer from PP ranks to PME ranks is 44% for the FFTW, while in the other two cases, it is significantly low (in the range of 0.1 to 2.1%). In contrast, the waiting time for PME ranks to receive data from PP ranks was significantly higher in these two cases (22.5 to 23.2) when compared to 9% corresponding to FFTW version. Interestingly, the performance of different FFT libraries in the scaling behaviour seems to have origin in the time spent on data transfer from PP to PME ranks and PME ranks waiting for data from PP ranks. The time spent on computing itself is comparable for the three versions.

The profiling analysis also shows that for the large-sized system, the load imbalance was about 25%. In this, about 16% was due to domain decomposition, and the remaining was due to load imbalance between PP/PME ranks. The former performance slow-down can be improved by using fewer cells (during domain decomposition) along the direction where the simulation box inhomogeneity is higher. The load imbalance in PP/PME ranks can be improved by accommodating more processes for doing PME.

Memory

Depending upon the system size, the memory usage can be from a few hundred MB to few tens of GB. In GPUs (using MPI-CUDA version of GROMACS), effective memory management can be done by allocating PME tasks to GPUs while doing particle–particle interaction calculations within the CPUs.

IO

GROMACS supports several file formats, including GROMACS-specific formats and other file formats such as PDB. The master node is responsible for IO operation for IO. Parallel IO libraries, such as HDF5 and NetCDF, are not used. The typical output file size depends on the simulation parameters and in particular on the number of particles in use. Simulations with 30,000 and 0.85 million particles require approximately 1.5MiB and 40MiB (in double precision) output.

Malleability and resilience

The GROMACS has the potential to run on laptops, desktops, workstations, supercomputers with GPUs, and in any computers with heterogeneous architectures. On most occasions, the software itself identifies efficient ways to distribute the tasks. For example, the PP and PME tasks are allocated to different groups of ranks in multiple CPU machines. Similarly, the task allocation between the CPUs and GPUs is also efficiently handled by the software. The domain decomposition scheme is also implemented efficiently, and this takes care of distributing the particle–particle interaction calculations to various ranks within the PP group of ranks.

GROMACS supports checkpointing / restart capabilities using the .cpt format.

6 Computational Fluid Dynamics

6.1 Introduction

Computational Fluid Dynamics (CFD) is at the heart of modern engineering and an indispensable tool for a qualitative and quantitative analysis of the flow of fluids in areas such as automotive, aerospace, energy, weather and climate. A significant topic here is turbulence, as about 10% of the energy use in the world is spent overcoming turbulent friction [11]. Improvements in this area have thus a clear environmental and societal impact. Furthermore, with a virtually unbounded need of computational resources, CFD is a natural driver for exascale computing [12].

6.2 Nek5000

Nek5000 is an open-source computational fluid dynamics code based on the spectral element method. Nek5000 solves the incompressible Navier–Stokes equations, together with a number of additional physics (heat transfer, magneto-hydrodynamics, low Mach number, electrostatics) on general hexahedral spectral elements. Special focus is laid on single-core efficiency via fast tensor product operator evaluations. For high-order methods, assembling either the local element matrix or the full stiffness matrix is prohibitively expensive. Therefore a key to achieving good performance in spectral element methods is to consider a matrix-free formulation, where one always works with the unassembled matrix on a per-element basis. Gather–scatter operations are used to ensure continuity of functions on the element level, operating on both intra- and inter-node element data.

6.2.1 Programming languages

Nek5000 is mainly written in Fortran 77, with communication (and some I/O) routines implemented in C as a separate library gslib.

6.2.2 Libraries and other dependencies

The application can be built with a set of optional libraries, ParMETIS to enable graph-based mesh partitioning, p4est providing adaptive mesh refinement, HYPRE for algebraic multigrid preconditioners and VisIt libsim or ParaView Catalyst for in-situ analysis.

6.2.3 Parallelization

The code uses distributed memory parallelization, following the matrix-free numerical formulation and distributes the workload, whole spectral elements, based on a recursive spectral bisection or dual graph-based bisection of the computational mesh.

Communication in Nek5000 is mainly due to collective operations from inner-products, norms, and gather–scatter operations for ensuring continuity between elements. The current gather–scatter library in Nek5000 utilities three different communication strategies: nearest neighbour exchange, message aggregation, and collectives, to efficiently perform communication on a given platform.

Accelerators

Nek5000 is undergoing a considerable development effort to efficiently support accelerators. The entire solver runs on NVIDIA GPUs using a combination of OpenACC directives for the bulk of the application and CUDA for performance-critical kernels. Support for AMD accelerators is currently being developed using a similar strategy with OpenMP target offloading for most of the code and native HIP for performance-critical kernels.

6.2.4 Application requirements

6.2.5 Typical use case

Typical use cases for Nek5000 are high fidelity flow simulations of turbulent flow using 6–9th order polynomials on meshes with 100,000–500,000 hexahedral elements, resulting in hundreds of millions of grid points.

6.2.6 Large use case

Large use cases often use more than one million elements, with the same 6–9th order polynomials, thus billions of grid points. For example, the current pre-exascale use cases are in the order of 50 billion grid points (50 million elements) and require close to a terabyte of data per checkpoint.

Scaling

The scalability of Nek5000 has been investigated in several studies [14, 13], primarily focusing on traditional scalar processors. On these platforms, the code is mainly limited by latency and synchronization costs and will scale strongly down to 10–30 elements per Processing Element (PE), depending on polynomial order.

To illustrate the scalability of a typical use case, we consider the Taylor–Green Vortex (TGV) problem at a Reynolds number of 5000. In Figure 19 (left), the average time per time-step is shown for solving TGV on a Cray XC40, using a mesh with 262,144 elements and ten quadrature points in each direction (ninth order polynomials). For this configuration, Nek5000 scales well down to 8–16 elements per PE, before communication costs starts to dominate and affect scalability.

Weak scalability studies are often performed using Nekbone, Nek5000's proxy application. Nekbone solves the Poisson's equation and has a code structure that resembles the basic structure of the full code. Due to its smaller size, Nekbone is often used to evaluate the performance on new architectures

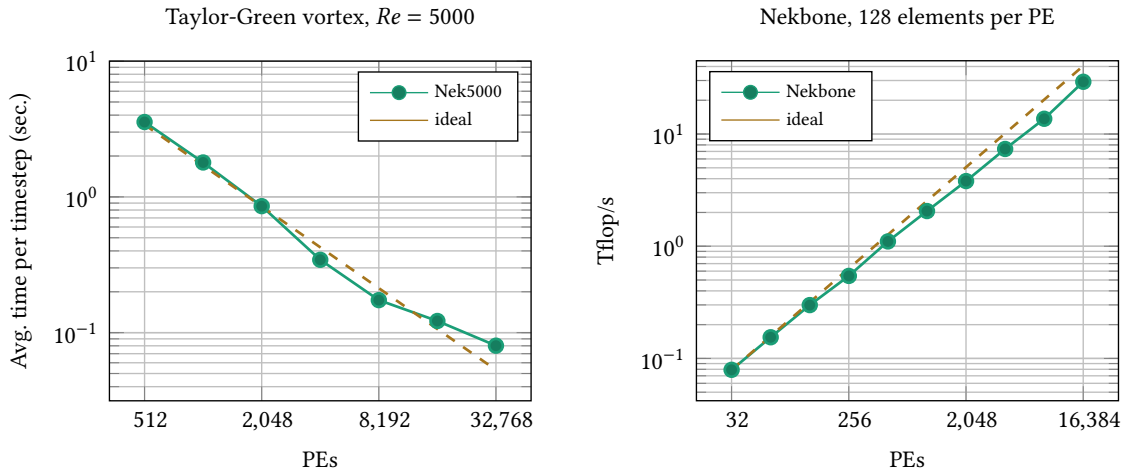


Figure 19: Strong scaling results for Nek5000 (left) for solving the Taylor–Green vortex problem, and weak scaling results for Nekbone (right), both running on a Cray XC40.

and as a testbed for code porting and kernel tuning. Figure 19 (right) gives weak scaling results for Nekbone on a Cray XC40.

Compute

Memory

Memory usage in Nek5000 depends on the number of elements E and the polynomial order N. The code uses roughly 400 8-byte words per degree of freedom (dof), where each element hold $(N+1)^3$ dofs (assuming the same polynomial order in each direction).

IO

Application IO is performed using MPI-IO, in Nek5000 own file format.

Malleability and resilience

The main flow solver in Nek5000 is not able to utilize different resources nor adjust to available resources at runtime. However, in-situ analysis can be co-scheduled on different resources as well as ensemble runs.

For resilience, Nek5000 periodically write a set of checkpoint files, allowing for a proper (high-order time integration) restart.

7 Neutron Monte-Carlo Transport for Nuclear Energy

7.1 Introduction

The French Alternative Energies and Atomic Energy Commission (CEA) is a large French Research and Technological Organisation (RTO). CEA is a key player in research, development and innovation in four main areas: low carbon energies (nuclear and renewable energies), technological research for industry, fundamental research in the physical sciences and life sciences, defence and security.

The solution of the linear Boltzmann equation by the Monte Carlo method is based on the simulation of a great deal of random particle trajectories within the considered system. The ensemble averages of the simulated trajectories provide estimates for the physical observables of interest. The Monte Carlo trajectories describe random walks whose mathematical properties are chosen to take into account the physical laws of particle–matter interaction, to the best of our knowledge. Since the method does not require any space, energy or angle discretization, Monte Carlo has always been considered as the *golden standard* for the study of nuclear reactors. This desirable property comes at the price of a somewhat slow statistical convergence, with uncertainties scaling as the inverse of the square root of the number of simulated histories. In turn, the basic Monte Carlo algorithms lend themselves well to massive parallelization, which can help mitigate the slow convergence.

Until very recently, the large CPU requirements of Monte Carlo simulation have almost exclusively limited its use to the study of stationary problems within systems with a predetermined set of temperatures and compositions (i.e. without any physical feedback loop). However, the computing power and available storage have now grown to the point where it is possible to run true “numerical experiments” in realistic configurations using Monte Carlo.

Today, the goal of Monte Carlo neutron transport is the simulation of full-core configurations, in non-stationary conditions (quasi-static depletion calculations and true dynamic calculations with physical feedback). This requires a deep revision of the architecture of Monte Carlo codes; indeed, it is crucial to efficiently exploit the massive parallelism and vectorization which characterize modern machines.

The PATMOS code, developed at CEA, has been designed to explore the requirements of next-generation Monte Carlo neutron transport, in terms of code architecture, parallelism, algorithms, memory and CPU time. The goal of PATMOS is to demonstrate the feasibility of Monte Carlo calculations for the depletion of a full-scale pressurized water reactor, taking into account thermo-hydraulics and thermo-mechanical feedbacks.

7.2 PATMOS

PATMOS prototypes the next generation of applications used for nuclear reactor physics, nuclear safety and radiation shielding (see Figure 20). It is written in C++ and relies on the hybrid MPI+OpenMP programming model to express massive parallelism.

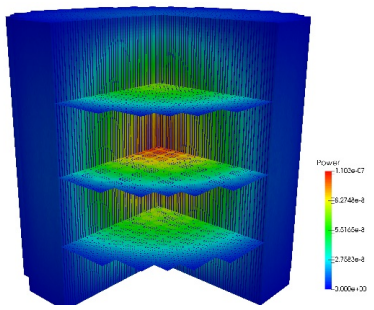


Figure 20: A three-dimensional representation of the fission rate calculated by PATMOS in the Hoogenboom-Martin benchmark.

```
for i_sample ← 1 to n_sample do
    source ← InitSourceParticles();
    for particle ∈ source do
        while IsAlive (particle) do
            material ← FindMaterial(particle.position);
            Σ ← 0;
            for nuclide ∈ material do
                σ ← ComputeCrossSection(nuclide, material.temperature);
                density ← nuclide.density;
                Σ ← Σ + σ × density;
            end
            particle.position ← SampleNextInteractionPoint(Σ);
            nuclide ← SampleCollidedNuclide(material);
            reaction ← SampleReaction(nuclide);
            particle ← UpdateParticleState(reaction);
        end
    end
    AccumulateScores (i_sample);
end
```

Algorithm 1: Pseudo-code for the innermost PATMOS loop.

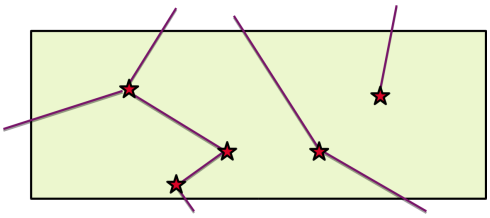


Figure 21: A schematic representation of the Monte Carlo particle-transport algorithm. Lines represent particle flights, stars represent collisions.

Algorithm 1 describes, through a pseudo-code, the structure of the innermost Monte Carlo loop in PATMOS. Particle transport is simulated by sampling flights and collisions from suitable probability distributions (see Figure 21). The parallelization of this algorithm is relatively straightforward since the iterations of the outermost loop are easily decoupled. Each iteration in fact represents an independent Monte Carlo sample. Parallelization of this loop requires the following (standard) actions to be taken:

1. The pseudo-random number generators need to be seeded in such a way that they may be considered independent.
2. The scores accumulated by different parallel units need to be reduced at the end of the simulation by taking their statistical averages.

With these provisos, PATMOS can be efficiently parallelized across multiple MPI ranks (in order to exploit multiple compute nodes) and OpenMP threads.

In large depletion calculations, the amount of memory required for each copy of the scores is very large (up to 1 TB). For this reason, it is typical to maximize sharing and minimize the memory footprint by running only one MPI process on each compute node. Concurrent score updates from different threads are synchronized using atomic operations.

7.2.1 Programming languages

PATMOS is written in C++14 and exposes a set of Python bindings (through pybind11).

7.2.2 Libraries and other dependencies

PATMOS can be coupled to external solvers for depletion calculations.

7.2.3 Parallelization

As detailed in the previous section, PATMOS is based on MPI and OpenMP parallel programming models to exploit the parallelism available in the outermost loop of Algorithm 1. Because of the update of scores, communication patterns rely on reduction operations in MPI.

Accelerators

Currently, PATMOS does not support accelerators.

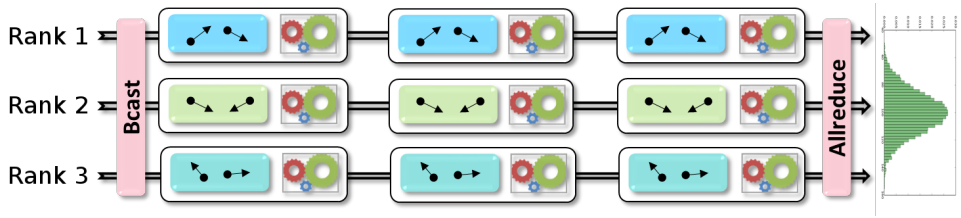


Figure 22: Illustration of PATMOS typical use case.

7.2.4 Typical use case

The PATMOS typical use case follows the steps described in Algorithm 1 and is illustrated in Figure 22. Each MPI rank processes a set of samples that are further updated through OpenMP parallel regions. Thus, each line of the figure represents one MPI process, starting with a broadcast operation to distribute the required input data. Then each rank applies one iteration of the outermost loop from Algorithm 1. At the end a collective operation is performed to reduce all scores and to broadcast the main result to every MPI rank.

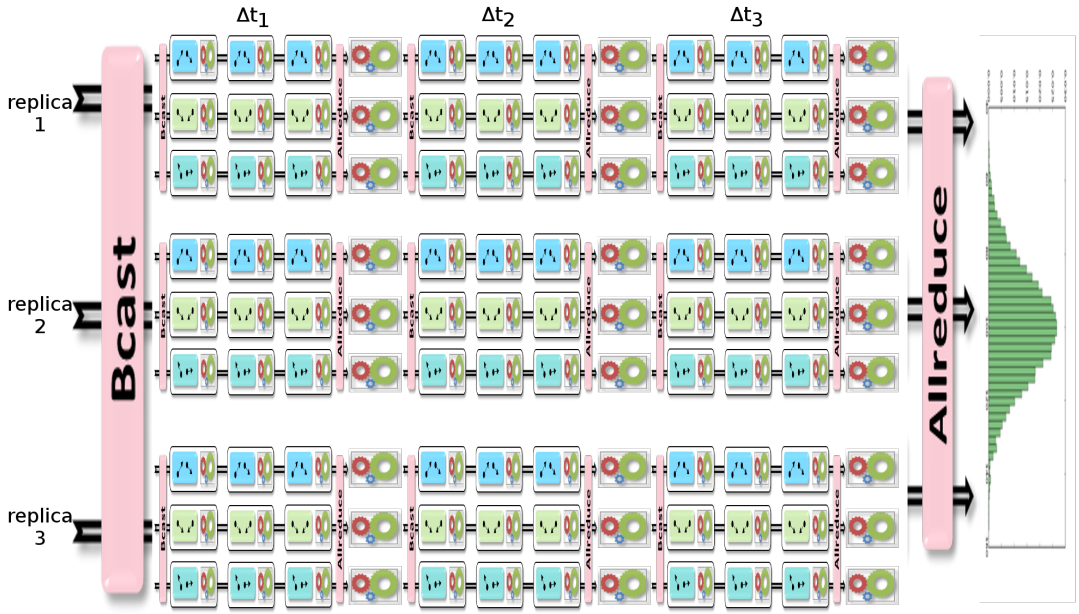


Figure 23: Illustration of PATMOS target use case.

7.2.5 Target use case

The target use case models the irradiation cycle of a full nuclear reactor core. This kind of simulation requires several hundreds of thousands of cores. As shown in Figure 23, the structure of the target use case is an intercalation of transport loops (as presented in the Typical Use Case in Figure 22) and calls to an external depletion solver, represented by the gears in the diagram. The depletion

solver takes the scores computed by the transport step as input, and outputs material compositions for the following transport step. This use case could be a candidate to exploit an exascale system.

7.2.6 Application requirements

This section details some information about the PATMOS typical use case and target use case described above.

Scaling

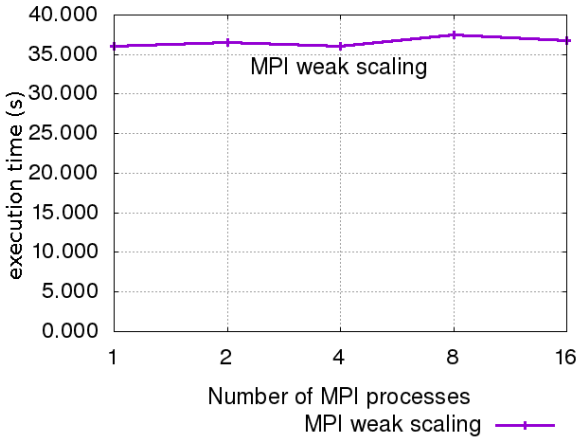


Figure 24: PATMOS MPI Weak Scaling

Figure 24 depicts the execution time of typical use case with increasing number of MPI ranks in a weak scaling manner. This experiment was conducted on one node of dual-socket 8-core Intel Sandy Bridge processors. The amount of work is increased as the number of MPI ranks is getting larger. With a perfect parallel efficiency, the execution time should remain the same. Because the MPI communication pattern is pretty simple in the typical use case, the overall parallel efficiency is good. However, the efficiency on the target use case is expected to be lower because of intermediate communications within each replica.

Figure 25 highlights the OpenMP efficiency through a weak scaling study on the same compute node with increasing amount of work, leading to weak scaling runs. Because the time to process each particle could vary and could lead to creation of new batches of particles, the OpenMP parallelism is not well balanced. This phenomenon has an impact on the parallel efficiency illustrated on the figure through an increase in the execution time.

Compute

The main computational part of PATMOS is illustrated in Algorithm 1. The `ComputeCrossSection` function can be implemented with different methods. One possibility is through table lookup, leading to large memory storage. The other approach relies on computational kernel called `Sigma1` that

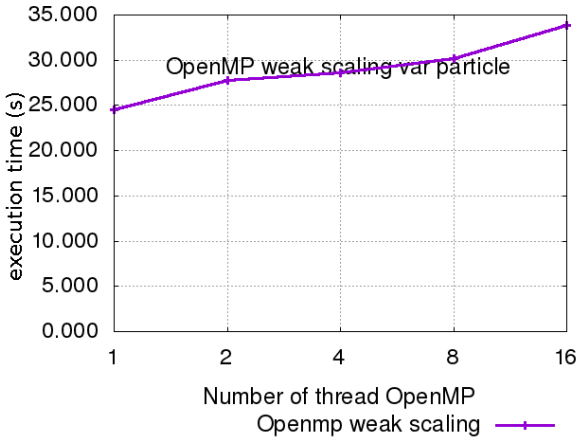


Figure 25: PATMOS OpenMP Weak Scaling

re-calculates cross section information. This kernel can be a good candidate to evaluate GPU offloading, even if this would require some refactoring of the overall algorithm to enable computing cross sections on batches of particles.

Memory

The typical use case requires about 1 GB of memory while the target use case may consume up to 1 TB of memory.

IO

Regarding input files, PATMOS stores cross sections into serialized C++ objects through a header-only library called `cereal`. For output information, the code prints ASCII outputs (which is reliable when the output is small). For the target use case, the main goal is to rely on an external standard library. This work will be realized outside of the DEEP-SEA project, but its outcome will be available for use with the target use case within the timeframe of the project.

Malleability and resilience

PATMOS has no restriction on the number of resources to exploit and the distribution among MPI processes and OpenMP threads. This has to be set when launching the code and cannot be dynamically modified during the execution of the application. Furthermore, there are no specific developments for fault tolerance nor snapshots. Because the main algorithm is based on Monte Carlo approach, it can automatically adapt to external failures (assuming the underlying software stack is able to handle this failure).

8 Earth System Modelling: TSMP

8.1 Introduction

The Simulation and Data Lab Terrestrial Systems (SDLTS) is a research and support unit at the Jülich Supercomputing Centre (JSC). SDLTS acts as a contact point for the Terrestrial System research community to HPC, and in particular to the resources and expertise at JSC. SDLTS further links other regional partners working at the interface between geosciences and HPC. One of the key responsibilities of SDLTS is the maintenance of the Terrestrial System Modelling Platform (TSMP), and supporting its user and developer community.

Terrestrial Systems are conceptualised as a subset of the Earth System integrating interactions and feedbacks among processes occurring on the land surface, the subsurface and the lower atmosphere. Processes in these Earth System compartments are typically non-linear and span a large range of spatial and temporal scales. Moreover, the physical processes typically controlled by small scale variability tend to manifest in emergent behaviour in larger scales. This multiscale nature implies that it is complex and difficult to predict how the emergent behaviours arise from a multitude of small scale interactions. Consequently, physics-based representations of single processes, which are well-understood at small scales are key to develop an integrated understanding of Terrestrial Systems. In the context of Earth System Science, the modelling of Terrestrial Systems is key to assess the effects of environmental change on the soil-vegetation-atmosphere continuum, and the effects of land-use changes on regional climate.

Physics-based modelling of Terrestrial Systems poses significant computational challenges. Firstly, it is an intrinsically multi-physics problem, with various and very different domains (porous media, land surface, atmosphere). Fluxes in these domains are described by systems of partial differential equations of different nature and thus requiring different numerical and computational solutions. Additionally, the spatial scales range from the process scales and the scales of heterogeneity (of land use, soils, and topography) in the orders of meters, all the way to continental scales in the order of thousands of kilometres. The temporal scales range again from the process scales in the orders of seconds to climate and ecological time scales of decades and centuries.

Spanning such a diverse computational problem requires smart modelling techniques, robust and efficient numerical solutions, and reliable, efficient, and scalable computational implementations. This requires a strong community effort, and indeed, several community models targeting each of the components of interest for Terrestrial System modelling. The Terrestrial Systems Modelling Platform (TSMP) leverages on a set of existing community models to provide an integrated modelling platform, which allows to couple the subsurface, land and lower atmosphere domains through well-defined physically based models, while ensuring HPC applicability and software sustainability.

Many challenges still exist and are arising in regional Earth System modelling and Terrestrial Systems modelling. There is a strong need to increase model resolution to enable convection-permitting scales in atmospheric models, to better resolve spatial heterogeneities on the land surface and subsurface and to capture key topographical features such as streams. Additionally, further physics are likely to enter in these models, attending to fundamental needs (such as addressing biogeochemical cycles) and those arising from resolving smaller scales. Moreover, there is a push of enabling global scale

terrestrial system simulations. Finally, the inherent uncertainties in modelling Terrestrial Systems requires to run ensembles. All these aspects together constitute a strong pressure to take Earth and Terrestrial System models to exascale capabilities.

8.2 Terrestrial System Modelling Platform

The Terrestrial Systems Modelling Platform (TSMP) is a fully coupled, scale consistent, highly modular and massively parallel regional Earth System Model. TSMP (v1.2.3) is a model interface which couples three core model components: the COSMO (v5.01) model for atmospheric simulations, the CLM (v3.5) land surface model and the ParFlow (v3.7) hydrological model. Coupling is done through the OASIS3-MCT coupler. TSMP is also enabled for Data Assimilation (DA) throughParallel Data Assimilation Framework (PDAF). TSMP allows to simulate complex interactions and feedbacks between the different compartments of terrestrial systems. Specifically, it enables the simulation of mass, energy and momentum fluxes and exchanges across land surface, subsurface and atmosphere [10]. TSMP is maintained by SDLTS (JSC), and is an open source software publicly available in GitHub [1].

The coupling design is inherently modular, allowing to build all combinations of component models, or only build one of them. This design also leads to a Multiple-Program Multiple-Data (MPMD) execution model and operational flexibility, allowing the different model components to run at different spatial and temporal resolutions. In fact, within TSMP different versions of the component models are supported for both legacy and experimental purposes.

TSMP is undergoing significant developments, which are not part of DEEP-SEA but are ocurring in parallel and will be deployed during the lifetime of DEEP-SEA. Experimental development branches include the coupling of the ICON atmospheric model and the upgrade from CLM3.5 to CLM5. These implementations are at different stages of development but are expected to become operational within a year. Consequently, they will interact with co-development efforts in DEEP-SEA.

The component models—developed by third parties—are written using different programming languages, use different parallelization and acceleration schemes (can exploit different hardware), and show different scaling behaviour. The diversity in features and responses is partly what motivates the modular design of TSMP.

TSMP is mostly a compute-driven application. The core computational effort comes from solving large sets of partial differential equations. The computational requirements are higher for COSMO/ICON and ParFlow, and considerably lower for CLM. Additionally DA ensemble runs are somewhat data-driven, as observational data needs to be assimilated into the workflows.

The core TSMP workflow is illustrated in Figure 26. The core component models are initialised, and all three component models are run concurrently. The different complexities and numerical approximations result in different runtimes to a particular time level. In particular CLM will typically finish first. Once the two other components run until the desired time level, information is exchanged from COSMO/ICON and ParFlow into CLM, which can start advancing again, and then exchange the information back again, followed by triggering the other two component models to continue marching forward. This process is repeated throughout the simulation.

[1] https://github.com/HPSCTerrSys/TSMP

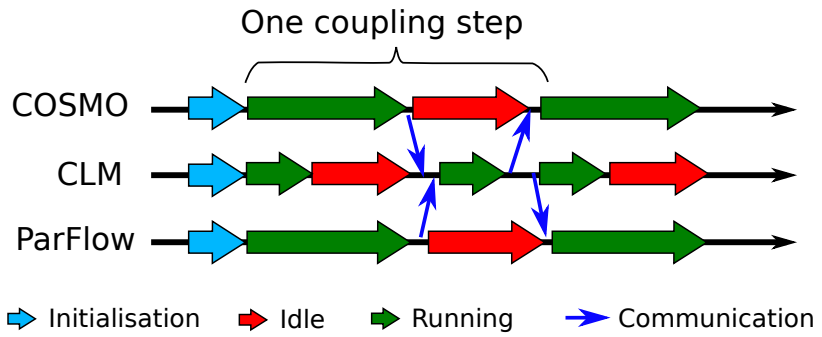


Figure 26: Computational workflow within TSMP. Adapted from [2].

Preprocessing workflows are necessary to setup initial conditions (initialization), and to prepare boundary forcing data (in particular for the atmospheric component) which will be used throughout the simulation. Postprocessing workflows are used for different types of visualization, analytics, etc. Data Assimilation workflows are also frequent with TSMP. DA workflows (see Figure 27) rely on ensembles of concurrent runs as the one illustrated in Figure 26 and are discussed further as a parallelization feature in Section 8.2.3.

8.2.1 Programming languages

TSMP can be conceptualised as a two-part system: an interface between the different components models and a build workflow which creates a common environment, patches component model source codes and compiles the model system.

The interface itself is constructed mostly over the OASIS3-MCT coupler which is written in Fortran90. The build workflow relies on shell (kash), Tcl and Perl. TSMP's coupling and wrapper codes files match the component models: Fortran (COSMO, CLM3.5, PDAF) and C (ParFlow). The expected new components of TSMP (ICON and CLM5) are also written in Fortran.

8.2.2 Libraries and other dependencies

The dependencies for TSMP are mostly those required by the component models. These include curl, GRIBAPI HDF5, HYPRE, BLAS/LAPACK, eccodes, NetCDF, Silo, zlib.

Visualization is not supported within TSMP, but its output follows the NetCDF standard and can therefore be visualised through ncview and Paraview, or using Python and R.

8.2.3 Parallelization

TSMP has different levels of parallelization, illustrated in Figure 27. TSMP strongly relies on point-to-point MPI communicators managed through OASIS, coupling the component models. These

communicators are also used within the component models to deal with domain decomposition. Collective communicators are also used within component models for selected operations.

The first level of parallelism is a consequence of the MPMD execution model of TSMP, where the three component models (COSMO/ICON, ParFlow, CLM) run concurrently on different computational resources (compute nodes). The global communication space is provided by TSMP via OASIS3-MCT, leveraging on MPI (MPI_COMM_WORLD). The distribution of these resources is typically done based on experience and is known to potentially strongly impact performance. For the typical use case (see Section 8.2.4) the resource distribution among components is well established from empirical experience.

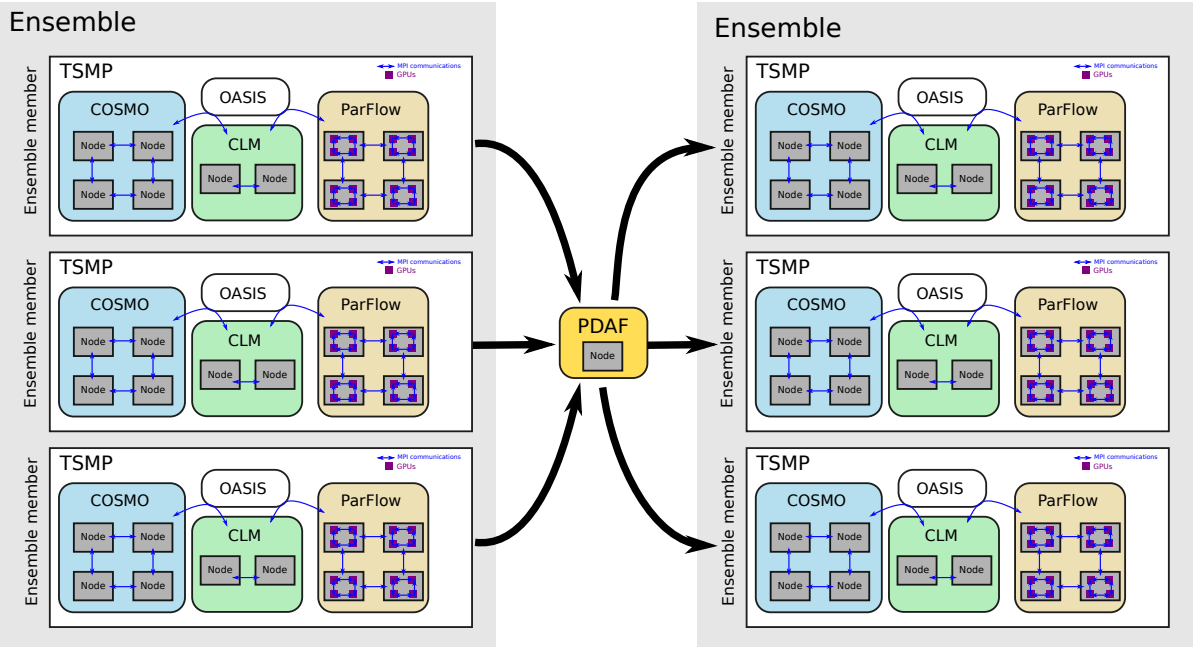


Figure 27: TSMP parallelization levels with a Data Assimilation workflow, and a heterogeneous computing configuration.

Within a model component, the second level of parallelism is implemented via MPI for all component models (COSMO/ICON, CLM, ParFlow). Intra-node parallelism also relies (in the default configuration) on MPI. Node-level multiGPU ParFlow execution is enabled through MPI.

A third level of parallelism is enabled in ParFlow and ICON (experimental in TSMP) via OpenMP, and through GPU accelerators (see details in Section 8.2.3).

Additionally, workflows using ensembles for DA allow one higher level of parallelism. Ensembles consist of a set of independent members (or realizations), all of which have the same problem size and computational resources. They are effectively repetitions of the same base computational problem. These ensemble members are independent and can be run concurrently. Once all realizations are finished, information is collected from all of them by PDAF. Together with observational data, PDAF then constructs DA operators and creates new configurations for a new ensemble, which launches a new set of concurrent runs. This process is repeated until the Data Assimilation process is completed. The process-blocking reduction of ensemble runs into PDAF is a potential bottleneck.

Empirical observations show that a few ensemble members may have significantly longer runtimes than the average of the ensemble. This is unavoidable, since the simulation setup (parameters, initial conditions, forcing, etc.) of each member has a stochastic nature, and it is difficult to predict if an excessive runtime will be required.

Accelerators

ParFlow v3.7 is the only component model in TSMP which is ported into CUDA for NVIDIA GPU acceleration [4]. The next version of ParFlow (v3.8, not yet supported in TSMP) also enables Kokkos for acceleration. ICON (which remains experimental in TSMP) also provides GPU support. ICON (currently experimental within TSMP) also has an OpenACC backend for accelerator portability.

8.2.4 Typical use case

TSMP's typical use case is the so-called EUR11-CORDEX problem. This case simulates mass, energy and momentum fluxes across land surface, subsurface and atmosphere in a continental scale pan-European domain, conforming to the specifications of the EUR-11 model grid, as defined by the World Climate Research Program's Coordinated Regional Downscaling Experiment (CORDEX) [5]. This domain is roughly 5450$\times$5300 km in extension, and includes Europe (and parts of northern Africa, and western Russia and Asia). It is discretized with a resolution of 0.11° (approximately 12.5km), resulting in a horizontal grid of $436 \times 424 = 184864$ elements. Each of the component models requires different vertical resolutions. ParFlow is discretized with 15 vertical elements($\sim 2.8 \times 10^6$ total elements), CLM with 10 ($\sim 1.9 \times 10^6$ total elements) and COSMO with 50 ($\sim 9.2 \times 10^6$ total elements) [6, 7]. Parametrization, initial conditions and forcing for this typical use case are well established and publicly available. They can be retrieved with automated scripts available at TSMP's GitHub. The duration of the simulation depends on its intended use, which may range for a few hours (typically for scaling and benchmarking studies) to months and years (for forecasting), and to decades (for reanalysis). A short simulation configuration is available together with TSMP, for a 12-hour long simulation.

8.2.5 Target use case

The next generation of simulations of continental scale water cycle and land-atmosphere interactions will require a significant increase in spatial resolution, to achieve *convection-permitting* scales, ideally with resolutions approaching the km-scale. This will lead to improved simulation of, for example, local precipitation [1] which in turn affects water redistribution on and below the surface. Additionally, higher spatial resolutions will better resolve dynamics induced by different land uses, soil types and topography on both hydrodynamics and atmosphere dynamics. Experimental setups exist for sub-continental regional simulations [9], and in the near future the push is to take convection-permitting resolutions to continental domains [3].

Consequently, the European CORDEX domain will be re-discretized to a resolution of 0.0275° (approximately 3×3 km), which implies $1600 \times 1552 = 2435200$ horizontal grid points. This new case is called *pEU3km* and will maintain the vertical discretization of the EUR-11 CORDEX case (Section

8.2.4). This translates into a computational problem roughly $13.2\times$ larger than the EUR-11 CORDEX case. Additionally, increased sharp gradients in topography may affect the stability and convergence rates of non-linear solvers in TSMP's component models, making it hard to predict a-priori what will be the increase in computational cost. Scaling the computational problem is not a matter only of increasing the spatial grid resolution, but the entire parametrization and forcing of the system needs to be downscaled. This is an ongoing project with partners at the Institute of Bio- and Geosciences: Agrosphere (IBG-3/FZJ), which is still being configured and experimented with [3, 8], and which will be made available for DEEP-SEA as the target use case.

8.2.6 Application requirements

Scaling

Detailed weak scaling studies [2] have been performed in the past in systems which are now decommissioned (JUQUEEN, IBM Blue Gene/Q system). This study showed that CLM had a 98% parallel efficiency, with near-zero communication overhead. COSMO (v4.11) showed a 92% parallel efficiency, and the higher computational load, thus governing runtime. ParFlow (v3.1) showed a lower efficiency of 82%. Although this serves as a rough guideline, the current version of TSMP implements more recent versions of the component models, notably including a GPU-accelerated ParFlow, for which dedicated studies exist [4] showing excellent scaling. A detailed scaling study for the current version of TSMP (and its components) needs to be retaken, also considering current hardware, and in particular under heterogeneous configurations. This will be carried out within DEEP-SEA.

Non-exhaustive scaling studies of TSMP are periodically run on JSC systems with a short duration run of the typical EUR11-CORDEX problem, mainly serving for compute-cost estimations. Figure 28 shows the latest strong scaling results in JUWELS.

It is quite clear the main sources of scaling inefficiencies in TSMP come from load imbalancing and idle times of computational resources. As illustrated in Figure 26, the CLM resources are idle for large times, and these resources are not used. Additional idle times can occur between COSMO/ICON and ParFlow, although in a significantly lower magnitude.

No scaling studies are available yet for the target use case pEU3km. Experimental runs with singular components (CLM or ParFlow) have been carried out in the JURECA system [8] without performance studies.

Compute

A typical setup of computational resources for the EUR-11 CORDEX case in JUWELS (JSC) is as follows. COSMO requires 384 MPI processes, ParFlow 144 MPI processes and CLM 48 MPI processes. In terms of compute nodes in JUWELS, this setup requires 12 nodes, with an allocation of 8 for COSMO, 3 for ParFlow nodes and 1 for CLM. Standard requirements for running this case under a heterogeneous setup, with ParFlow GPU are under investigation. Recent jobs in JUWELS show average inter-node data transfer rates in the order of 1 GB/s, and peaking at 7 GB/s.

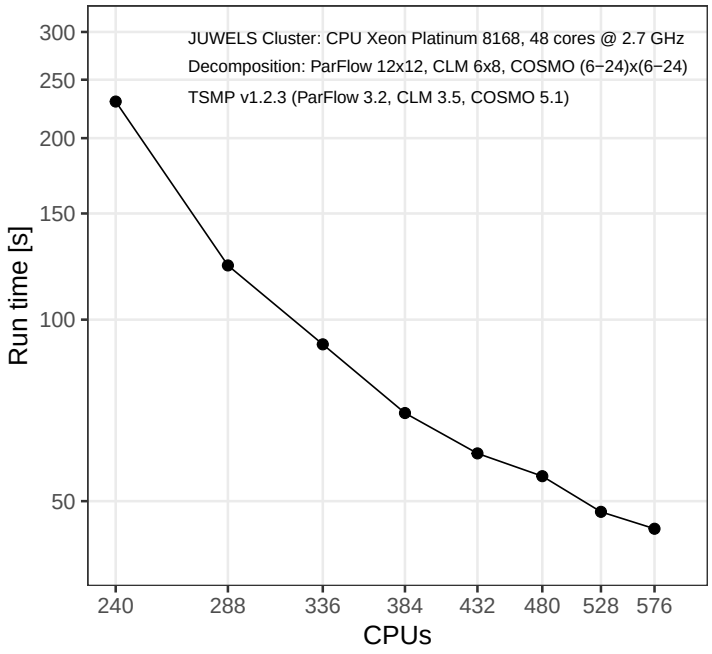


Figure 28: Strong scaling of TSMP in JUWELS, for the EURO CORDEX case.

The required computational resources for the target case are yet to be determined. A blunt extrapolation from the current setup is likely to be inaccurate since the change in resolution also produces changes in non-linearities in the system. This affects numerical stability and convergence, potentially increasing computational effort non-linearly.

Memory

The memory requirements per node depend on which component model is being executed in a given node, and of course on the allocated computational resources. For the typical EUR-11 CORDEX with the resource detailed in Section 8.2.6, the total memory used is 295 GB, and the average node memory use is approximately 25 GB, peaking nearly at 30 GB.

For the target test case there is no measured memory footprint. Total memory can be expected to increase proportional to the increase in the number of elements, resulting in a total footprint of 3894 GB. For a rough comparison, with the current available memory in JUWELS nodes this target test case would require around 41 standard nodes (or 21 large memory nodes), considering only memory limitations.

IO

IO is not controlled by TSMP itself, but rather its component models. The key dependency for IO is the NetCDF library. Model initialization requires a set of files for each model component, from namelists (a few MBs) to NetCDF files (in the order of 100 MBs). Boundary forcing data is in the form

of NetCDF files, usually obtained from global atmospheric simulations, downscaled, and interpolated into the typical EUR-11 domain. Boundary data volumes depend on the frequency at which forcing is imposed on the model. Approximately 3 GB of input data are required per hour of model time. All component models support the NetCDF standard for output. COSMO additionally supports GRIB and ParFlow also supports a native, non-standard binary output format. In practice, mostly NetCDF files are used in TSMP jobs. Total output volumes depend on the output frequency. For the typical case, a run would require approximately 15 GB per model day (with hourly COSMO and CLM output, and daily ParFlow output). During runs in JUWELS of the typical EUR-11 case show IO rates peaking at ~ 850 MB/s, for both read and write.

For the target use case the output would increase roughly to 198 GB per model day if the same frequency is kept. However, higher output frequencies may become relevant in such problem.

Malleability and resilience

TSMP has been successfully ported to all operating JSC HPC systems (JUWELS, JURECA, JUSUF) and the experimental DEEP-EST system. Additionally, it has also been ported to other Tier-1 and Tier 2 HPC systems hosted by other institutions. An automated workflow is also available to port TSMP to generic x86 machines.

TSMP runs can be executed with ad-hoc resource allocations depending on availability, although designing the resource distribution for optimal performance is non-trivial, system-dependent, and case-dependent. TSMP does not currently allow a dynamic reallocation of computational resources. In theory the MPMD model would allow for a good degree of malleability. Moreover, malleability would be especially desirable for Data Assimilation ensemble runs, in which resources from fast ensemble members are then transferred to slower members.

TSMP can be viewed as fault tolerant, insofar any snapshot written to disk (normal simulation output) can be used to restart a simulation. This mechanism is commonly used in TSMP jobs at JSC for very long simulations broken into shorter jobs which comply with the maximum job time policies.

9 Summary

This deliverable describes the co-design applications and their requirements. The applications cover a wide range of scientific fields, programming languages, and parallelization strategies. They include pure MPI applications, MPI+X and global address space (GASPI) approaches. They take advantage of accelerators using pragma-based as well as language-based programming models. Some focus on point-to-point communication, but most make use of collective MPI calls. Several of the applications can generate large data sets and one even has dedicated IO ranks. Many focus on computational aspect, but at least two applications rely on large input sets and at least one uses machine learning techniques both to improve the calculations and interpret the results. Together these applications provide a cross section of HPC codes used on European supercomputers and an excellent testing ground for the programming models, tools, and API developed in the other work packages of the DEEP-SEA project

List of Acronyms and Abbreviations

A

AIDA	Artificial Intelligence and Data Analysis. H2020 project.
AIDApy	Python package for the analysis of space data developed by the AIDA project.
API	Application Programming Interface

B

BN	Booster Node (functional entity)
BoP	Board of Partners for the DEEP EST project
BSC	Barcelona Supercomputing Centre, Spain

C

CLM	Community Land Model
CM	Cluster Module: with its Cluster Nodes (CN) containing high-end general-purpose processors and a relatively large amount of memory per core
CN	Cluster Node (functional entity)
CORDEX	Coordinated Regional Climate Downscaling Experiment
COSMO	Atmospheric model - Consortium for Small-scale Modeling
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture

D

DAM	Data Analytics Module: with nodes (DN) based on general-purpose processors, a huge amount of (non-volatile) memory per core, and support for the specific requirements of data-intensive applications
DEEP	Dynamical Exascale Entry Platform (project FP7-ICT-287530)
DEEP-ER	DEEP – Extended Reach (project FP7-ICT-610476)
DEEP/-ER	Term used to refer jointly to the DEEP and DEEP-ER projects
DEEP-EST	DEEP – Extreme Scale Technologies

DN Nodes of the DAM

E

EC European Commission

ESB Extreme Scale Booster: with highly energy-efficient many-core processors as Booster Nodes (BN), but a reduced amount of memory per core at high bandwidth

EU European Union

Exascale Computer systems or Applications, which are able to run with a performance above 10^{18} Floating point operations per second

F

FFT Fast Fourier Transform

FHG-ITWM Fraunhofer Gesellschaft zur Foerderung der Angewandten Forschungs e.V., Germany

Flop/s Floating point Operation per second

FP7 European Commission 7th Framework Programme

FPGA Field-Programmable Gate Array, Integrated circuit to be configured by the customer or designer after manufacturing

G

GPU Graphics Processing Unit

GROMACS A toolbox for molecular dynamics calculations providing a rich set of calculation types, preparation and analysis tools

H

H2020 Horizon 2020

HPC High Performance Computing

HPDA High Performance Data Analytics

HW Hardware

I

ICON Icosahedral Nonhydrostatic Weather and Climate Model

Intel Intel Germany GmbH, Feldkirchen, Germany

IO Input/Output. May describe the respective logical function of a computer system or a certain physical instantiation

J

JURECA JURECA (Jülich Research on Exascale Cluster Architectures) Supercomputer at FZJ

JUWELS JUWELS (Jülich Wizard for European Leadership Science) Supercomputer at FZJ

K

KNL Knights Landing, second generation of Intel® Xeon Phi (TM)

KU Leuven Katholieke Universiteit Leuven, Belgium

L

LLNL Lawrence Livermore National Laboratory

M

MoU Memorandum of Understanding

MPI Message Passing Interface, API specification typically used in parallel programs that allows processes to communicate with one another by sending and receiving messages

MPMD Multiple-Program-Multiple-Data

MSA Modular Supercomputer Architecture

N

NUMA Non-Uniform Memory Access

O

OmpSs BSC's Superscalar (Ss) for OpenMP

OpenCL Open Computing Language, framework for writing programs that execute across heterogeneous platforms

OpenMP Open Multi-Processing, Application programming interface that support multi-platform shared memory multiprocessing

P

ParFlow Hydrological model

PDAF Parallel Data Assimilation Framework

Phi see Xeon Phi

PIC Particle-in-Cell algorithm.

PME Particle mesh Ewald

PMT Project Management Team of the DEEP-EST project

Q

R

RAM Random-Access Memory

S

SCR Scalable Checkpoint/Restart. A library from LLNL

SIMD Single Instruction Multiple Data

SIONlib Parallel I/O library developed by Forschungszentrum Jülich

SW Software

T

Tk Task, Followed by a number, term to designate a Task inside a Work Package of the DEEP-EST project

TSMP Terrestrial System Modelling Platform

U

V

W

WP Work package

X

x86 Family of instruction set architectures based on the Intel 8086 CPU

Xeon Non-consumer brand of the Intel® x86 microprocessors (TM)

Xeon Phi Brand name of the Intel® x86 manycore processors (TM)

xPic eXascale ready Particle-in-Cell code for space plasma physics.

Y

Z

Bibliography

[1] N. Ban, C. Caillaud, E. Coppola, E. Pichelli, S. Sobolowski, M. Adinolfi, B. Ahrens, A. Alias, I. Anders, S. Bastin, D. Belušić, S. Berthou, E. Brisson, R. M. Cardoso, S. C. Chan, O. B. Christensen, J. Fernández, L. Fita, T. Frisius, G. Gašparac, F. Giorgi, K. Goergen, J. E. Haugen, Ø. Hodnebrog, S. Kartsios, E. Katragkou, E. J. Kendon, K. Keuler, A. Lavin-Gullon, G. Lenderink, D. Leutwyler, T. Lorenz, D. Maraun, P. Mercogliano, J. Milovac, H.-J. Panitz, M. Raffa, A. R. Remedio, C. Schär, P. M. M. Soares, L. Srnec, B. M. Steensen, P. Stocchi, M. H. Tölle, H. Truhetz, J. Vergara-Temprado, H. de Vries, K. Warrach-Sagi, V. Wulfmeyer, and M. J. Zander. The first multi-model ensemble of regional climate simulations at kilometer-scale resolution, part i: evaluation of precipitation. *Climate Dynamics*, apr 2021.

[2] F. Gasper, K. Goergen, P. Shrestha, M. Sulis, J. Rihani, M. Geimer, and S. Kollet. Implementation and scaling of the fully coupled terrestrial systems modeling platform (TerrSysMP v1.0) in a massively parallel supercomputing environment – a case study on JUQUEEN (IBM blue gene/q). *Geoscientific Model Development*, 7(5):2531–2543, oct 2014.

[3] K. Goergen, A. Belleflamme, A. Ghasemi, C. Hartick, B. S. Naz, L. Poshyvailo, W. Sharples, N. Wagner, and S. J. Kollet. First results from a convection-permitting pan-European fully coupled TSMP simulation. In *AGU Fall Meeting Abstracts*, volume 2020, pages A096–0005, Dec. 2020.

[4] J. Hokkanen, S. Kollet, J. Kraus, A. Herten, M. Hrywniak, and D. Pleiter. Leveraging HPC accelerator architectures with modern techniques — hydrologic modeling on GPUs with ParFlow. *Computational Geosciences*, may 2021.

[5] D. Jacob, C. Teichmann, S. Sobolowski, E. Katragkou, I. Anders, M. Belda, R. Benestad, F. Boberg, E. Buonomo, R. M. Cardoso, A. Casanueva, O. B. Christensen, J. H. Christensen, E. Coppola, L. D. Cruz, E. L. Davin, A. Dobler, M. Domínguez, R. Fealy, J. Fernandez, M. A. Gaertner, M. García-Díez, F. Giorgi, A. Gobiet, K. Goergen, J. J. Gómez-Navarro, J. J. G. Alemán, C. Gutiérrez, J. M. Gutiérrez, I. Güttler, A. Haensler, T. Halenka, S. Jerez, P. Jiménez-Guerrero, R. G. Jones, K. Keuler, E. Kjellström, S. Knist, S. Kotlarski, D. Maraun, E. van Meijgaard, P. Mercogliano, J. P. Montávez, A. Navarra, G. Nikulin, N. de Noblet-Ducoudré, H.-J. Panitz, S. Pfeifer, M. Piazza, E. Pichelli, J.-P. Pietikäinen, A. F. Prein, S. Preuschmann, D. Rechid, B. Rockel, R. Romera, E. Sánchez, K. Sieck, P. M. M. Soares, S. Somot, L. Srnec, S. L. Sørland, P. Termonia, H. Truhetz, R. Vautard, K. Warrach-Sagi, and V. Wulfmeyer. Regional climate downscaling over europe: perspectives from the EURO-CORDEX community. *Regional Environmental Change*, 20(2), apr 2020.

[6] J. Keune, F. Gasper, K. Goergen, A. Hense, P. Shrestha, M. Sulis, and S. Kollet. Studying the influence of groundwater representations on land surface-atmosphere feedbacks during the european heat wave in 2003. *Journal of Geophysical Research: Atmospheres*, 121(22):13,301–13,325, nov 2016.

[7] S. Kollet, F. Gasper, S. Brdar, K. Goergen, H.-J. Hendricks-Franssen, J. Keune, W. Kurtz, V. Küll, F. Pappenberger, S. Poll, S. Trömel, P. Shrestha, C. Simmer, and M. Sulis. Introduction of an experimental terrestrial forecasting/monitoring system at regional to continental scales based on the terrestrial systems modeling platform (v1.1.0). *Water*, 10(11):1697, nov 2018.

[8] B. S. Naz, W. Kurtz, C. Montzka, W. Sharples, K. Goergen, J. Keune, H. Gao, A. Springer, H.-J. H. Franssen, and S. Kollet. Improving soil moisture and runoff simulations at 3 km over europe using land surface data assimilation. *Hydrology and Earth System Sciences*, 23(1):277–301, jan 2019.

[9] E. Pichelli, E. Coppola, S. Sobolowski, N. Ban, F. Giorgi, P. Stocchi, A. Alias, D. Belušić, S. Berthou, C. Caillaud, R. M. Cardoso, S. Chan, O. B. Christensen, A. Dobler, H. de Vries, K. Goergen, E. J. Kendon, K. Keuler, G. Lenderink, T. Lorenz, A. N. Mishra, H.-J. Panitz, C. Schär, P. M. M. Soares, H. Truhetz, and J. Vergara-Temprado. The first multi-model ensemble of regional climate simulations at kilometer-scale resolution part 2: historical and future simulations of precipitation. *Climate Dynamics*, 56(11-12):3581–3602, feb 2021.

[10] P. Shrestha, M. Sulis, M. Masbou, S. Kollet, and C. Simmer. A scale-consistent terrestrial systems modeling platform based on COSMO, CLM, and ParFlow. *Monthly Weather Review*, 142(9):3466–3483, sep 2014.

[11] M. Guest, G. Aloisio and R. Kenway. The scientific case for HPC in Europe 2012-2020. *PRACE*, 2012.

[12] J. P. Slotnick, A. Khodadoust, J. J. Alonso,D. L. Darmofal, W. D Gropp, E. A. Lurie, D. J. Mavriplis. CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences. *NASA Technical Report*, NASA/CR-2014-218178, 2014.

[13] N. Offermans, O. Marin, M. Schanen, J. Gong, P. F. Fischer, P. Schlatter, A. Obabko, A. Peplinski, M. Hutchinson, E. Merzari. On the Strong Scaling of the Spectral Element Solver Nek5000 on Petascale Systems. *Proceedings of the Exascale Applications and Software Conference*, 2016.

[14] P. F. Fischer. Scaling limits for PDE-based simulation. *22nd AIAA Computational Fluid Dynamics Conference*, 2015.

[15] Keiiti, A. and Richards, P.G. Quantitative Seismology. University Science Books, 2003

[16] de la Puente, J., Ferrer, M., Hanzich, M., Castillo, J.E. and Cela, J.M. Mimetic seismic wave modeling including topography on deformed staggered grids. *Geophysics*, **79**(3), T125–T141, 2014

[17] Davydycheva, S., Druskin, V. and Habashy, T. An efficient finite-difference scheme for electromagnetic logging in 3D anisotropic inhomogeneous media. *Geophysics*, **68**(5), 1525–1536, 2003

[18] Van Der Spoel, David and Lindahl, Erik and Hess, Berk and Groenhof, Gerrit and Mark, Alan E and Berendsen, Herman JC. GROMACS: fast, flexible, and free. *Journal of computational chemistry, 26, 16*, 2005.