EuroHPC-01-2019

DEEP-SEA

DEEP – Software for Exascale Architectures

Grant Agreement Number: 955606

D1.3
Applications use of DEEP-SEA software stack

Final

Version:	1.0
Author(s):	U. Sinha (FZJ), M.E. Holicki (FZJ)
Contributor(s):	J. Amaya (KULeuven), M.I. Andersson (KTH), D. Caviedes Voullieme (FZJ), G. Tashakor (FZJ), P. Carribault (CEA), D. Grünewald (FhG), N. Jansson (KTH), M. Karp (KTH), D. Mancusi (CEA), S. Markidis (KTH), O. Marsden (ECMWF), J. de la Puente (BSC), J.E. Rodriguez (BSC), O. Castillo-Reyes (BSC))
Date:	31.03.2022

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:	D1.3
	Deliverable Nature:	Report
	Deliverable Level: PU*	**Contractual Date of Delivery:** 31.03.2022
		Actual Date of Delivery: TBD
	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:** Applications use of DEEP-SEA software stack	
	ID: D1.3	
	Version: 1.0	**Status:** Final
	Available at: `https://www.deep-projects.eu/`	
	Software Tool: LaTeX	
	File(s): DEEP-SEA_D1.3_Applications_use_of_DEEP-SEA_software_ stack.pdf	
Authorship	**Written by:**	U. Sinha (FZJ), M.E. Holicki (FZJ)
	Contributors:	J. Amaya (KULeuven), M.I. Andersson (KTH), D. Caviedes Voullieme (FZJ), G. Tashakor (FZJ), P. Carribault (CEA), D. Grünewald (FhG), N. Jansson (KTH), M. Karp (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:	Stefano Markidis (KTH) Jennifer Lopez Barrilao (ParTec)
	Approved by:	BoP/PMT

Document Status Sheet

Version	Date	Status	Comments
1.0	31.03.2022	Final Version	EC Submission

Document Keywords

Keywords:	DEEP-SEA, HPC, Exascale, Software, Applications, Benchmarks, 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.

Contents

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.

This Deliverable focuses on identifying the programming paradigms and tools suited for each of the applications and describes the areas where they are providing their co-design input to the developers. It provides a detailed description of the topics on which these applications are collaborating with the other work packages of the DEEP-SEA project and their current status regarding each optimisation cycle mentioned in the Deliverable D3.1 [1].

1. Introduction

The previous Deliverables [2, 3] reported the application requirements in terms of libraries, hardware, and software in addition to their integration with the Jülich benchmarking environment (JUBE) [4] and instrumentation with performance analysis tools like Score-P, Extrae, Paraver, and Scalasca [5, 6, 7, 8]. At the moment, all these applications have been successfully ported into the DEEP system and are working to explore the Modular Supercomputer Architecture (MSA). The integration with JUBE has enabled them to perform automated benchmarking tests and facilitate easy reproducibility. Furthermore, the applications have successfully generated traces for their typical use-cases that has helped them to identify the performance bottlenecks and explore the regions that have potential for improvements.

In this Deliverable, each task presents the tools, software components, and programming models for their applications. Furthermore, a detailed description of their inter-work package and cross-SEA collaboration is presented. The tasks have identified the programming paradigms and tools from WP2-5 for their applications and are working together to adapt them to the MSA. These tasks are actively interacting on a variety of topics with all the DEEP-SEA work packages, the IO-SEA project, and the RED-SEA project to improve their applications and are providing their input for the co-design process.

2. Space Weather

2.1. Introduction

Space weather and its implications on society and technology has been discussed in detail in the Deliverable D1.2 [3]. The particle-in-cell code xPic, developed at KU Leuven, studies the effect of solar plasma on the planets of our solar system. We employ Data-Analysis (DA) and Machine Learning (ML) techniques to determine the initial and boundary conditions of the simulation and analyse the generated data. This simulation framework has been successfully run using the MSA on the DEEP system at Jülich Supercomputing Centre (JSC).

KU Leuven is also developing ML tools to analyse spacecraft and simulation data. The type of data generally encountered in these problems include time-series, remote images by solar telescopes, and multi-dimensional arrays captured by spacecraft instruments and by particle simulations. One example of the ML applications used in this project is concerned with the analysis of solar active regions. Among the different tools in the AIDApy package we have included algorithms to characterize solar active regions, i.e. zones of increased magnetic activity on the sun that can suddenly eject mass and radiation dangerous to our planet. Being capable of characterizing active regions, these can be used to classify the type of active region, and their level of risk. Our long term plan is then to use this automatic characterization to forecast solar energetic eruptions by following the evolution of the characteristic features over time.

We are constantly adding new Artificial Intelligence (AI) and ML tools to the AIDApy package. In addition to the use of ML for the study of solar active regions described in the previous paragraph, during the DEEP-SEA project we will also implement a second algorithm that analyses the very large amounts of data produced by the particle solver of the code xPic. We are using the Gaussian Mixture Model (GMM) to classify plasma regions of different characteristics in different zones simulated by the code xPic. Large exascale simulations of space plasmas can produce more than one TiB of particle information. This data cannot be stored and analysed after the execution of the code, it needs to be processed on-the-fly. We have developed an algorithm that transfers data from the particle solver of xPic to the GMM algorithm of the AIDApy package. This model is then executed in a Data Analytics Module.

2.2. Tools, software components and programming models

2.2.1. Application xPic

A detailed description of the code xPic and its internal components has been presented in Deliverables D1.1 [2] and D1.2 [3]. In this section, we describe the processor architectures and operating systems that xPic is compatible with and its requirements in terms of software libraries and compilers. In addition, the tools used for the performance analysis and optimisation of xPic are also described.

xPic has been developed under the DEEP projects and hence the programming models employed during the course of its development has been according to the selected processor architectures.

Initially, it was designed to run on a cluster-booster architecture where the booster module was made up of Intel Xeon Phi accelerators. Later, during the DEEP-EST project, a more generic approach was adopted to be able to run on boosters comprising of Nvidia GPUs. More recently, work has been done to make xPic compatible with the new generation of AMD processors and accelerators. These developments are inline with our goal to enable xPic to run on a variety of architectures.

One of our main goals has been to develop a code that can be ported to any general purpouse architecture. We understand that this can come with drawbacks, *e.g.* performance can be slightly compromised by the use of non-specific hardware operations. KU Leuven has decided to sacrifice a small part of the performances in exchange of portability.

xPic can be ported in to POSIX systems running on Linux, Unix, and macOS. It has also been ported to the HPC systems DEEP and JUWELS at JSC. To take advantage of the MSA, xPic creates two types of files at different intervals: 1) moment files (containing information about the plasma density, current, electric and magnetic fields, pressure tensors, etc.) with values at the grid points of Cartesian mesh, and 2) particle files (containing information about the particle position, velocity, and charge). As the number of particles in the simulation far exceed the number of grid points, the size of particle files are a few orders in magnitude larger than the moment files. We use the general I/O systems to obtain our simulation data and in case of check-pointing use the fast local storage.

The libraries required to compile the code are: a) C/C++ compiler that provides support to OpenMP 5.0, b) MPI with multi-threading support, c) HDF5 compiled with support for parallel I/O [9], d) H5hut (optional high level interface to HDF5) [10], e) PETSc [11]. Additional libraries are used in case of increased functionality. The code xPic has been equipped with optional interfaces to SIONlib and SCR for fast parallel I/O and checkpoint/restarting.

In addition to these libraries, we use a set of python scripts to initialise the fields and particles. These scripts use h5py to write initialisation files in HDF5 format [12]. The initialisation and simulation data files can be read, analysed, and post-processed using Python scripts. A post-processing tool, provided with xPic, converts these files in the Xdmf format compatible with Paraview [13] and Visit [14].

The code performance and bottlenecks are carefully monitored by using a series of clocks to measure the execution time of the different sections. This data is stored and shared between all processors to generate a statistical report at the end of each simulation run. Extrae/Paraver [6, 7] and Intel VTune [15] are used to instrument xPic for performance measurements. Analysis of load balancing and execution of the code has been done using Extrae/Paraver in collaboration with BSC. The roofline analysis has been done using Intel VTune. Recently, we have generated profiles and traces to measure the performance of the GPU part of the code. In addition to these tools, we plan to use Dimemas [16] and Score-P [5] with the aim to achieve high performance on future architectures. These developments will support the co-design process through interactions via optimisation cycles.

We have also used Intel VTune to understand the behaviour of our code. We have used this tool in particular to do roof-line analyses. This is an extremely useful tool that we would like to see available in other profiling tools. We hope to use during this project the Score-P profiler and other tools for the analysis of the memory use. We will support the co-design of this software during our interactions in the optimisation cycles.

There is still room for optimisation of the xPic code. We currently use OpenMP 5.0 to offload computation to the GPUs. Different algorithmic strategies could improve the efficiency here. We have also worked in the efficiency of communications between MSA modules. Until now we have not seen strong bottlenecks in the transmission of data, but we will continue our investigations during this project. We are also making use of GPU direct communications between GPUs [17]. At the moment we assume that a single node has access to only one GPU, and we connect one MPI process in the Cluster with one MPI process in the Booster. It would be possible to increase the performances of the code if one CPU connects to multiple GPUs. Using multiple GPUs can accelerate our computations and produce faster runtimes, but we can alternatively use the added power to include more particles in our simulations, thus reducing the inherent noise in PIC simulations, improving the associated statistics, increase dramatically the accuracy of our code.

2.2.2. Application AIDApy

AIDApy is a group of python modules used to analyse the data provided by spacecrafts and those obtained from simulations to study the solar activity, forecast solar wind conditions near earth, predict geomagnetic effects, and study the physics of solar plasmas. These modules employ a combination of DA, ML, and statistical tools for a rigorous analysis. As we mentioned in the introduction section 2.1 we propose to use two specific cases in the context of the DEEP-SEA project: a) the automatic parametrization of solar active regions, and b) the characterisation of particle distribution functions from xPic simulations.

Of these two cases, the first one, the parametrization of solar images, was originally deployed on a laptop featuring a 1 TiB SSD disk, an Intel Core i9-10885H CPU, and an Nvidia GTX 2070 Max-Q. Before performing any optimisation, we wanted to compare this desktop hardware with the un-optimized code in the DEEP system. The same code has been ported to the DEEP system, where we tested the algorithms using the Data Analytics Module (DAM) module. Although the accelerators installed in the DEEP system are more powerful, the performances of the GPUs were comparable in both systems. This performance penalty still needs to be investigated and corrected. However, a more important bottleneck was detected in the reading of large number of images in the ML algorithm used for this test (a deep convolutional neural network). The test running in the laptop with an SSD, was showing much better run-times than a single node in the DAM module of the DEEP system. To solve this issue, in our SLURM scripts we implemented an initial copy of all the files into the local NVMe drive of the DAM node. For this particular test we are using 140 MiB of data, and the copy from the login node to the DAM node takes only a few seconds at the beginning of the job. The I/O bottleneck is then averted by relying on the local disks in the DAM. We caution other users of the importance of using fast disks for any AI/ML application that requires the reading of large amounts of data.

The second test case, the analysis of particle distribution functions from xPic, has already been ported to the DEEP system. It can be executed in two modes: 1) reading particle information from disk, 2) receiving particle information directly from xPic, during execution, using MPI messages. We have already deployed and tested scripts for the launch of the xPic+AIDApy/GMM codes using the three modules of the DEEP system. The test is still under development and will require further refining and analysis.

The AIDApy tools have been developed in Python and use the PyTorch [18] and Scikit-Learn [19] ML frameworks. In addition we make use of typical data analytic tools like NumPy, Pandas, Xarrays. We also use external packages that grant access to data from spacecraft missions stored by the National

Aeronautics and Space Administration (NASA) and the European Space Agency (ESA). This data is freely available to anyone on the online data services of the respective agencies. Basic parallelism is obtained using mpi4py. On the DEEP system we make use of the pre-compiled PyTorch libraries, but the remaining packages are installed in a virtual Pip environment. The environment is saved in a directory that can be accessed by the compute nodes to be activated and give access to the pre-installed python packages. We are also evaluating the possible use of Singularity containers to pre-package all the required Python software into a single file. Containerization for the MSA is being studied in WP3 of the DEEP-SEA project.

For the two different tests described above, the data has two different origins. For the analysis of particle distributions, the data is generated by the simulation code xPic. Information about billions of particles from the simulation is traditionally stored in a large files. For the second case, the data consists of thousands of images of solar patches, each of size 526x256x3, where the third dimension is the number of channels used, in this case corresponding to three components of the magnetic field of the surface of the sun. We are planning to extend the dataset for this second case with many more channels per image, adding information of more wavelengths of the electromagnetic spectrum. This can multiply the amount of data used for the training of our second test case.

We use internal clock calls to optimise the AI/ML operations, including the use of multi-threading and GPU processing, in AIDApy. We are working on further improvements by making appropriate use of the underlying APIs and the transfer and location of the data.

2.3. Collaborations

KU Leuven is actively collaborating with the rest of the consortium by providing access to xPic and AIDApy. With an aim to improve our understanding of the physics of space plasmas, we are working towards implementing tools and algorithms that can make use of the upcoming exascale supercomputers. In addition, we have been developing Python tools to analyse large data generated by simulations and those provided by spacecrafts. In the following sub-sections, we describe our collaboration with other work packages of the DEEP-SEA project in detail. KU Leuven provides source-code access to xPic and to certain Python scripts integrated in the AIDApy codebase.

Our primary goal is to create an HPC software that will help to improve our scientific knowledge of the physics of space plasmas. This requires the implementation of algorithms and tools that can be deployed on exascale supercomputers. Our secondary goal is to develop all the Python tools necessary for the analysis of the large amounts of data generated by exascale simulations and spacecraft missions.

With these goals in mind we started our work in the DEEP-SEA project by profiling and understanding the bottlenecks in our codes. We describe in the following subsections how we are working in collaboration with all the WPs of the project to achieve our project objectives. In each subsection we also describe which are our requirements, or what are we expecting from other WPs.

The two applications described above, xPic and AIDApy, have been developed with different tools. While xPic is a C/C++ highly parallel HPC software, AIDApy is a Python-based data-intensive analysis tool. As such we have the opportunity to collaborate on multiple fronts with different WPs and with different partners in the DEEP-SEA project.

We are collaborating with IO-SEA to take advantage of the latest I/O developments for generation, reading, and analysis of data and for the training of AI/ML methods in AIDApy. Furthermore, to benefit from better interconnection technology for parallel efficiency and inter-module communications in MSA, we are collaborating with RED-SEA. The details of our collaboration with the DEEP-SEA workpackages is described below:

- **WP2**: We started our work in this project with the profiling of xPic using the Extrae/Paraver tools developed by BSC. We have followed different tutorials, but we have also contacted the developers at BSC directly to better understand the results obtained during our profiling. We want to continue this collaboration in order to have clearer extrapolations of the future performances of our code for different exascale supercomputers. We also gave feedback on the user experience during a one hour meeting with the developers at BSC. We used also the VTune tools from Intel to do roof-line analysis of xPic. The Score-P tools have not yet been used, but we want to gather as much information as possible to improve the efficiency of our codes. As a requirement/request we would like to see in Paraver and in Score-P a global visualization similar to the roofline analysis of VTune. A detailed analysis of GPU performances is also very useful.

  KU Leuven is also looking at the possible analysis of memory-access patterns that can be obtained using the MemAxes optimisation cycle. The code xPic is extremely sensitive to memory-access patterns, as such we require better analysis tools and better memory handling as proposed in WP2. This is one of many other possible optimisations that we will explore in this project, if we have enough time.

- **WP3**: Work on using containers for our two codes, xPic and AIDApy, is ongoing for two major reasons: a) the compilation of xPic can be difficult and cumbersome for users not familiar with HPC environments, and b) the management of Python packages and virtual environments for the AIDApy code is not straightforward, in particular when the reproducibility of experiments is important. Containerization using Singularity has already been tested on these two cases. In our interactions with WP3 we will formalize and setup two solutions following the recommended best practices.

  KU Leuven also has a long tradition of version control, and in particular, GitLab has been a used for a few years now. CD/CI tools have already been deployed in our workflow, but only following the most basic procedures. We plan to improve our use of the CI tools included in GitLab for our codes. CI tools are complementary to the work done in WP1 with the support of benchmarking cases using JUBE. The CI tools will be used to automatically launch specific JUBE cases and keep track of the evolution of our codes.

  Finally we will test the newest scheduling strategies installed in SLURM for the execution of MSA jobs. The codes xPic and AIDApy can be coupled and used simultaneously in all the three modules of the DEEP system's MSA prototype. We plan to demonstrate the use of the newest developments in job scheduling by the end of the project using a fully coupled xPic-AIDApy run for the analysis of space weather. We have already tested some of the new MSA features of SLURM, including the launch on three modules and the use of gateways to transfer data between models. We still need to perform more in depth studies of these interactions.

We will also test in the future the use of a differed job starting for the AIDApy/GMM code. We will test the corresponding options in the SLURM script but also, if we have time and resources available, the execution of the SLURM API from within the code xPic.

- **WP4**: Possibilities to use Processing In Memory (PIM) in the code xPic are being explored. We find that there are some basic operations that could be carried completely with a PIM procedure, like reductions or sorting. We have expressed that we can collaborate in the co-design and testing of the PIM libraries.expressed that we can collaborate in the co-design and testing of the PIM libraries.

  Our two codes xPic and AIDApy require an update of their I/O routines. At the very large scale, with thousands of processors, we have noticed the parallel writing of particle files in the code xPic is slow. We are seeking help from our colleagues to find possible solutions to this particular issue. The DEEP system is a very good development platform, but in this particular case we need to perform tests on larger systems, like JUWELS, to find and correct any large-scale I/O issues.

- **WP5**: In addition to the requirements expressed in the previous subsection, xPic already uses SIONlib [20] and SCR [21] for the checkpoint/restart procedures in case of a fail or crash. This system was included some years ago, but needs to be update with the tools developed by WP5.

  We will take advantage of the better file-system data structures implemented in WP5 to minimize data reading bottlenecks in the training of AI/ML techniques requiring large amounts of data. We have already shown that data locality and access times plays a critical role in the training of AI/ML. We will collaborate on the testing of these new systems.

2.4. Co-design input

The performances of our applications depends on the optimisation of the libraries used. The applications of KU Leuven require the coordinated packaging and building the software stack, containing the optimized versions of al least the following basic libraries: ParaStation MPI, SIONlib, HDF5, H5hut, and PETSc. In addition, our AI/ML applications require the installation of Python frameworks like PyTorch and scikit-learn. This is particularly challenging because new functionalities are added and released constantly at a higher pace than the updating cycles of libraries in an HPC centre. To minimize the impact on our compilation environments we want to explore the use of containers, so we require the deployment of Singularity.

We are also co-designing the tools for the analysis techniques proposed in WP2. We find that these tools are critical for any application that wants to progress towards exascale. KU Leuven has participated in multiple calls with BSC to work on the Extrae/Paraver tools. We require the support of experts with these tools to make use of more detailed analyses and in the longer term to perform extrapolations of performances with additional tools like Dimemas. We require also support in the analysis of Score-P profiles and traces. We will be comparing the results of both tools to identify their differences, their strengths and their weaknesses.

KU Leuven is also cooperating on the implementation of efficient parallel I/O using libraries like SIONlib. Up until now we have prioritized the use of HFD5 because such files can be directly analysed by post-processing tools like h5py in Python and ParaView. We will evaluate if it is still better to use HDF5 with its performance issues or SIONlib with its incompatibility with post-processing tools. We need support on the comparison of performances between SIONlib and HDF5. We have seen a degradation of performances on runs with very high number of processors. We will ask for support on the actualization of the SIONlib interface in the code xPic.

We need clarification on the interactions between JUBE scripts and GitLab runners for the automatic execution of tests, in order to trace the evolution of our performance. In addition to the existing tutorials and documentation, a working example of a simple JUBE/GitLab setup would be helpful for our own implementations.

Much of the progress described above will be achieved using some of the optimisation cycles proposed in Deliverable D3.1. We have provided a list of the optimisation cycles we want to implement, the ones that we find interesting and will explore, and those that we will evaluate if our time and budget allows. KU Leuven requires clarification on the details of these optimisation cycles: what are their clear objectives, what are the inputs, and which are the expected outputs. Toy examples would also help to implement the cycles on our own codes. We also need to understand if the cycles are automatic or if will be manual procedures. A wiki entry for each one of the optimisation cycles, including a clear indication of the interfaces, and a set of examples, would be extremely helpful. We are in particular interested in the use of the profiling and optimisation cycles, i.e. the "MSA-related Optimisation", the "Energy Optimisation", the "Memory Performance Analysis", and the "Analysis of Data Transfers and Memory Behaviour".

We also hope that the optimisation cycles will provide automatic "hints" or "indications" during the code compilation or execution, which can be used to detect optimisation opportunities and heavy bottlenecks. We have participated in the definition of the optimisation cycles, and we are constantly informed about their progress thanks to calls and seminars. We are working on their co-design and we will be implementing the different tools during this project.

2.5. Conclusion

The codes xPic and AIDApy have been ported to the MSA architecture in the DEEP system. We are improving the parallel performances of the two codes and improving the efficiency of the GPU use. In collaboration with other WPs, and following the optimisation cycles proposed in this project, we have the goal to produce MSA ready codes that can be projected to future exascale architectures.

We have a long history of collaborations with multiple European partners in the DEEP projects. KU Leuven has been working on the co-design of the MSA concept by providing access to our scientific applications and by testing different new technologies. We are strengthening this collaboration during the DEEP-SEA project. We have already started working together in the analysis of our codes and we are eager to implement some of the optimisation cycles proposed by the different WPs.

KU Leuven has already ported the code xPic to the MSA architecture and also tested mockup codes running on the three modules of the DEEP system, exchanging information between C/C++ codes and Python scripts. Our codes have already all the necessary intercommunicators and they only require improvements and optimisations. Our next goal is to optimize the parallel efficiency of these applications using the collaborations described above and the optimisation cycles proposed in the project.

3. The IFS weather forecasting software

3.1. Introduction

The Integrated Forecasting System (IFS) suite is run operationally at ECMWF on a daily basis. In this operational context, optimized time-to-solution is critical to achieving high-quality forecasts. 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 written mostly in Fortran90, with some more recent Fortran (F03, F08) as well. It depends on a number of mostly C++ ECMWF libraries, as well as a small number of external scientific libraries (netCDF, HDF5, FFTW, BLAS). The IFS is currently undergoing significant work in order to prepare it for future supercomputers, in terms of both GPU compatibility, as well as modularity and flexibility. These efforts will work in synergy with DEEP-SEA. Investigation of IFS dwarfs and components (communication-heavy spectral transforms dwarf, compute-heavy CLOUDSC IFS micro-physics dwarf) via DEEP-SEA optimisation cycles will provide inputs and data-points to GPU offload developments, and reciprocally, more accelerator-enabled code will become available during the duration of the DEEP-SEA project. New dwarfs are being created where appropriate, in order to investigate mapping of IFS components to the MSA system. This is the case for example with the newly developed PAPADAM mini-app, which couples compute-heavy and communication-heavy IFS components in a small synthetic code-base, ideal for DEEP-SEA testing.

3.2. Tools, software components and programming models

The DEEP-SEA software stack includes a number of tools and developments aiming to allow improved use of hardware resources. A combination of these tools will be used to study IFS performance, to target improvements, and to monitor and asses their effects on overall application performance. This is described in the optimisation cycles below.

3.2.1. Optimisation cycles

The different optimisation cycles developed in DEEP-SEA are discussed below in relation to the IFS, and intended IFS improvements during the project.

- **Monitoring optimisation cycle:** The PAPADAM mini-app performance will be regularly monitored, thus enabling single-metric evaluation of performance evolution with both system stack updates, and improvements to the application itself. It will serve as an early warning when code changes have unexpected performance impacts, allowing deeper investigations to be carried out.
- **MSA-related optimisation:** The PAPADAM mini-app has been instrumented with Opari2, allowing Score-P usage. This will be regularly done during the project.

- **Application mapping toolchain:** During the DEEP-SEA project, two complementary avenues of improvements currently underway are expected to be completed. Firstly, an improved mechanism for MPI setup and communications between different coupled components, will enable better usage of job resources by the different components. Secondly, offload capability of the spectral transforms component of PAPADAM and of the IFS to GPUs, is expected to be completed. Both of these developments will be assessed within this optimisation cycle.
- **Malleable optimisation:** Investigation of dynamic load balancing and malleability is being carried out with modifications to the CLOUDSC mini-app to allow the use of DLB. Given the relatively small load imbalance observed in this mini-app, further work will be carried out to test DLB-based resources sharing in other IFS grid-point code which exhibits more load imbalance than CLOUDSC.
- **Memory management optimisation:** Data movement is known to be a bottleneck in the IFS physics. This cycle will be deployed on the CLOUDSC mini-app, with the aim of analysing memory access patterns in the IFS physics.
- **High-level programming interfaces:** The CLOUDSC IFS physics mini-app is representative of a large fraction of the computation carried out in grid-point space in the IFS. In previous HPC2020 projects (EPiGRAM-HS, EuroEXA) it has been used as a platform for prototyping a GPU porting strategy suitable for deployment on all of the physics parametrization code. Following these efforts we have an optimized OpenACC-Fortran version of this mini-app which runs efficiently on GPUs, and a source-to-source toolchain which is able to generate this version from our baseline Fortran-CPU implementation. These different versions will be used to assess the production of GPU-capable code with the DaCe framework. The baseline dwarf version will be parsed with DaCe, and GPU-targeted code will be generated, before being compared to our existing GPU port.
- **Energy Optimisation:** Energy-to-solution of the IFS as a whole will be assessed during the project. We expect that CPU frequency could be reduced in areas of code that are IO-heavy, or make significant use of GPUs, with little detriment to code performance. The BDPO tool will enable the testing of these assumptions, and the assessment of their impact on the overall energy-to-solution metric.
- **Memory system performance analysis:** It is expected that the CLOUDSC miniapp will be investigated with the memory system performance analysis cycle, simultaneously with work on the memory management optimisation cycle.
- **Multi-level simulation approach:** Compatibility of this tool with the IFS will be investigated.
- **Analysis of data transfers and memory behaviour:** As with the two other memory-related cycles, the MemAxes tool will be deployed on the CLOUDSC mini-app, in order to obtain the most complete view of memory usage characteristics in the IFS physics.

3.2.2. Benchmarking

We have created a JUBE benchmarking script for the PAPADAM mini-app and will create additional JUBE scripts for the IFS mini-apps as we use them during the project. The JUBE script for PAPADAM compiles the mini-app with optional Opari2 instrumentation for ScoreP, runs the binary and validates the results. The JUBE script has been integrated into GitLab runners for the DEEP cluster. Additional JUBE scripts will be developed for other IFS components, as they are investigated in the project. Full IFS test cases have been defined, which will be used to assess full application impact, once improvements have been demonstrated in IFS mini-apps.

3.3. Collaborations

We are collaborating with other work packages and expect to continue this throughout the project. Following is a detailed description of our inter-WP collaboration.

- **WP 2** The development and deployment of the JUBE script for the PAPADAM mini-app was carried out with assistance from WP2. During this work, Opari2 developers were made aware of the problems encountered in instrumenting PAPDAM. This collaboration is expected to continue when other JUBE scripts for other IFS components are developed.
- **WP 3**We have a prototype version of the spectral transforms which leverages persistent collectives to improve scalability, and would be keen to test this in the DEEP environment in conjunction with WP3. Therefore, we are ready to use the new standard MPI-4 persistent collectives as soon as they become available in the DEEP software stack during the DEEP-SEA project.
- **WP 4** Along with active collaboration with developers of the DLB library, interaction with developers of other tools is expected throughout the project.

3.4. Co-design input

We are working closely with the developers of various tools (e.g. Opari2 and Score-P) to streamline the Fortran support for the IFS use-case. Along with this we expect to provide feedback on the tools' application to IFS mini-apps, in terms of usability and of performance. We will thus help in ensuring compatibility of the tools with the modern Fortran standards present in the IFS codebase.

3.5. Conclusion

The PAPADAM mini-app has been developed under the DEEP-SEA project, with the aim of focusing on MSA-related improvements. A Continuous Integration (CI) method has been adopted to automate correctness- and performance-assessment of code changes. Optimisation work will be commencing, on top of this initial version of PAPADAM as well as on the other IFS mini-apps that will be used during the project. We have deployed the DLB load-balancing library in the CLOUDSC mini-app and generated performance profiles associated with this mini-app. We will further investigate possibilities of load-imbalance mitigation in the IFS based on DLB. The components of the DEEP-SEA software stack will help in focussing on areas of code where improvements will be most beneficial, as well as enable the monitoring of correctness and of performance as improvements are implemented.

4. Seismic imaging

4.1. Introduction

The Reverse Time Migration (RTM) method images seismic data and is based on the discretization of the full wave equation. It is employed for oil and gas exploration and allows for accurate imaging of complex subsurface structures.

The Fraunhofer Reverse Time Migration (FRTM) implements the RTM method in a robust and massively parallel way by using proprietary HPC tools (c.f section 12.1).

In Full Waveform Inversion (FWI), 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 difference between data and synthetics as sources for the simulation and runs backward in time. The wave fields 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. Correlated wave fields are written to disk and then read on a posterior kernel execution stage. Only at the end of the kernel run, a clean-up step removes those temporal correlated wave fields. 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. However, this is not the only simulation the FWI require. A minimization of a misfit functions $E(\mathbf{m})$ [22], where **m** is the current model, need to be done. The computation of E requires modelling accurately the response of our current model **m** by means of a forward simulation, also called test. Together with a few test runs (between two and six typically) misfit is accurate enough and so the model is updated: we have performed a single iteration of FWI. Optimisation requires about 15-30 iteration steps per frequency band used and typical runs use 3-10 frequency bands.

The Barcelona Subsurface Imaging Tools (BSIT) is a proprietary code with a high level of maturity and several other configurations besides FWI. Therefore, within DEEP-SEA all experiments and optimisation approaches use a BSIT analogue 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. The Mockup is easy to configure and modify, thus providing an ideal environment to investigate different optimisation cycles in the context DEEP-SEA project.

4.2. Tools, SW Components & programming models

4.2.1. FRTM

FRTM employs two levels of parallelism to compute the final result. The concurrent computation and aggregation of shots is trivially parallel and managed by the GPI-Space runtime system using an appropriate workflow implementation. The FRTM single-shot computation is a hybrid parallel GASPI-based application using a static two-level domain decomposition. On the first level, the simulation domain is partitioned on the distributed memory level across the processes and data is exchanged using GASPI. On the second level, the process local partitions are divided to allow for a task-based execution on the shared memory level using the pthread Asynchronous Constrained Execution (ACE) task scheduler. On the second level, the decomposition is very fine grained due to which many compute tasks are generated. The dynamic scheduling is then able to equilibrate any potential imbalances on the node level.

The kernels used to compute the propagation of the pseudo-acoustic wave fields are usually memory bound for most of the different available material classes. As such memory locality is crucial in order to achieve a good performance and FRTM may profit from the memory placement tools developed in the project. The concurrent shot computation is trivially parallel and is suitable to evaluate the malleability features developed along the project. Resources might be easily removed or added at that level of parallelism along the computation of the final result.

FRTM uses the JUBE benchmarking environment [4] to create a set of benchmarks, run those on the DEEP system and evaluate the results. The provided JUBE script instruments the FRTM proxy application. It comes as part of the FRTM GitLab repository, which contains all the required files. Tracing and profiling are performed using Score-P [5].

4.2.2. BSIT

BSIT distinguishes different parallelism paradigms. First, the workflow manages to run thousands of independent simulations (shots). Based on a master–worker approach, its natures allows the scalability to be almost linear. However, the performance of BSIT is obtained from the kernels, which simulates the wavefield simulation for each shot. Kernels, as in FRTM, are usually memory bound. In order to solve that limitation, BSIT kernels currently implement two levels of parallelism. First, the distributed memory parallelization. For those shots that do not fit in a computational node in terms of memory, it uses domain decomposition and MPI routines are used for inter-domain communication, including data exchange and gather and scatter operations when needed. Second, the shared memory parallelization. The PDE solver consists in two loops which are collapsed and parallelized using OpenMP leaving to each thread a number of consecutive memory regions to update.

BSIT also uses JUBE environment. The test script manages the download, compilation, and execution of the test. The code is instrumented so that Extrae can be used to analyse the obtained traces. This first test produces one inversion iteration and is set up to use OpenMP parallelism only. In its current configuration, this first test performs the computation of a gradient (one forward and a backward run) and single line search or test run (forward run) for a single shot. JUBE benchmark are on the DEEP-SEA GitLab repository.

4.3. Collaborations

We are collaborating with the other workpackages. The details of our inter-workpackage collaboration are the following:

- **WP2**: Integrate Score-P into the FRTM proxy app and Extrae/Paraver assimilation and tuning for the BSIT benchmark generation using JUBE.

- **WP3**: Work on GPI and GPI-Space and the memory management tools. BSIT is looking at the malleability tools, the energy consumption libraries, and the mutli-level simulation approach using MUSA. We are evaluating the memory management and performance optimisation cycles for BSIT.
- **WP5** : FRTM is planning to have a look into the malleability and resiliency features in addition to the standard monitoring cycle.

4.4. Co-design input

FRTM aims to provide input to the malleability features developed in WP5, especially for the interaction of GPI-Space with SLURM and *vice versa*. A tight coupling would allow to dynamically shrink or increase the used MSA resources in dependence of e.g. a preconfigurable job priority in combination with an optimal and/or minimal resource demand which is balanced with the current availability of resources. That would allow SLURM to shrink or increase the resources in order to allow for other jobs of higher priority on one hand, while FRTM could run at the maximum number of currently available resources at any time on the hand.

BSIT is a real-world application with high-memory requirements that serves as a model for obtaining useful feedback while evaluating novel memory hardware and libraries. Furthermore, the new DEEP-SEA programming models may be integrated for assessing malleability, resiliency, and energy consumption as well as the use of heterogeneous execution models, such as those provided by MUSA.

4.5. Conclusion

BSIT and FRTM are highly mature application that comes with a mockup version that allows collaborating within the optimisation cycles that require the evaluation of novel execution models and programming paradigms. Nevertheless, BSIT's main collaboration relies on integrating the most recent memory hardware and building a reference benchmark. FRTM is an excellent test case for the malleability features developed for GPI-Space.

5. GROMACS: Molecular Dynamics

5.1. Introduction

GROMACS is a massively parallel open source[1] Molecular Dynamics (MD) simulation code [23]. It is written in C++ and uses MPI and OpenMP libraries for parallel execution. It studies the physics of atoms and molecules by using the interaction forces between them to solve the equations of motion. MD is widely used by researchers in diverse fields such as drug discovery, protein folding, and material science. There is also extensive support for GPU acceleration for the whole application, and it has shown to be a cost effective way to run MD simulations [24, 25]. GROMACS uses CUDA for acceleration on Nvidia GPUs and OpenCL for GPU acceleration on AMD devices (both GPUs and APUs) and Intel integrated GPUs.

The Multiple-Program, Multiple-Data (MPMD) structure of GROMACS allows it to use the hardware in multiple configurations. The MPMD design separates the Particle-Mesh Ewald (PME) and the Particle-Particle (PP) calculations into a pipeline parallelism. This is described in detail in D1.1 and D1.2 [2, 3] and creates a natural load-balancing problem which will be of key importance when optimizing GROMACS for MSA systems. The PME calculations can use acceleration in different ways in a heterogeneous jobs. A job on a single node with a single GPU has four ways of using GPU acceleration, all useful for maximizing the utilization of the available hardware and maximizing throughput.

Due to the nature of MD we can optimize for several different types of jobs, including large single-trajectory simulations and multiple short-trajectory simulations for different types of enhanced sampling methods. It is possible to increase throughput by running batched simulations on the same hardware.

It is clear from these examples that GROMACS is a malleable code with many opportunities for optimisation within a MSA-system.

5.2. Tools, SW Components and programming models

The DEEP software stack contains various tools for performance monitoring and performance enhancement to support the transition towards Exascale HPC. We will in this section describe the most suitable tools that DEEP-SEA provides to solve critical challenges that exascale computing will present for GROMACS.

5.2.1. Optimisation Cycles

There is a set of optimisation cycles developed within DEEP-SEA WP3. These optimisation cycles represent typical workflows for application development on the DEEP system. In this section, we describe their potential usefulness for GROMACS and the current state of use.

[1] https://gitlab.com/gromacs/gromacs

- **Monitoring Optimisation Cycle** The monitoring infrastructure in DEEP will be used to track performance development and as an entry point into the other optimisation cycles.
- **MSA-related Optimisation** This cycle is based on the performance monitoring tool Score-P. Profiling and tracing with Score-P is now supported by GROMACS after a light modification of GROMACS. We have verified compatibility between Extra-P and GROMACS by using input data from Score-P profiling of GROMACS in Extra-P.
- **Application Mapping Toolchain** This optimisation cycle is useful for GROMACS to choose the best configurations suited to the available modules/resources. This cycle will work together with the two previous optimisation cycles to create an optimal mapping between the application's configuration and the available resources.
- **Malleable Optimisation Cycles** There are several potential use cases where GROMACS can utilise this optimisation cycle, for system wide malleability and on a node level. Although there are no concrete specifications for this cycle yet, some key aspects of GROMACS that relates to this cycle: GORMACS has support for check-pointing, it should therefore be possible to adapt to a malleable MSA-systems without any re-write of GROMACS code. The code has a malleable MPMD design with many configurations and can utilise heterogeneous hardware in multiple ways, for example with different combinations of off-loading PME calculations to GPU. GROMACS can be used for ensemble jobs and have support for multiple simulations on shared resources which allows for a flexible use on a system level.

  We will, with the DLB library, explore node-level malleability. The DLB library has a potential to increase throughput, peak performance and usability for application users. We aim to use DLB together with Extrae found in other optimisation cycles.
- **Memory Management Optimisation** This optimisation cycle is based on profiling from Extrae, we will based on the results of the profiling evaluate what optimisations within this cycle are useful.
- **High-level Programming Interfaces Cycle** We see the possibility of using DaCe to solve the PME calculation as a data-flow problem. This has been initialized by implementing a 1D FFT, which is the basis of the 3D FFT algorithm which is an integral part of PME. The PME calculation is the part of the code that limits strong scaling.
- **Energy Optimisation** We will investigate the use of BDPO for GROMACS.
- **Memory System Performance Analysis** We will analyse the performance of the heterogeneous memory system using Extrae, Paraver and PROFET. From this analysis we will evaluate the need for optimisation within the optimisation cycle.
- **Multi-Level Simulation Approach** We will investigate the compatibility of the software towards the end of the project after investigating other optimisation cycles.
- **Analysis of Data Transfers and Memory Behaviour** We would like to look into it after we have completed work with the other optimisation cycles.
- **Analysis and Debugging of MPI-RMA Communications** GROMACS does not utilize one-sided MPI; however, there are potential use-cases for one-sided MPI in GROMACS, such as for the 3D FFT communication. This optimisation cycle will be investigated as an optimisation tool towards the end of the project if our analysis with the other optimisation cycles are complete.

5.2.2. Benchmarking

We have several use-cases described in D1.2 [3] included in JUBE for automatic testing. There are more test-cases planned to better exploit the malleable system and showcase the many use-cases of GROMACS. We will also integrate GitLab Runners for remote execution of CI/CD tasks.

5.3. Collaborations

The following are planned and completed collaborations between WPs.

- **WP2** We have had collaborations with WP2 in implementing JUBE, modifying GROMACS to work with Score-P and designing meaningful benchmarks.
- **WP3** We expect future collaboration with WP3 when implementing the optimisation cycles.
- **WP4** We are developing kernels in DaCe for GROMACS and Nek. We have initiated contact with the developers of DLB library in the malleable optimisation cycles for node-level malleability.
- **WP5** We aim to collaborate with WP5 in optimisation of MPI collectives relevant to GROMACS.

5.4. Co-design input

At the present stage we have no co-design input.

5.5. Conclusions

We have set up an environment for performance testing and profiling GROMACS on the DEEP system, with JUBE and Score-P. Work has been initiated in identifying and testing relevant software from the DEEP software stack used in the optimisation cycles, such as Score-P, Extra-P, Extrae, Paraver, PROFET and DLB Library. The High-level Programming interface is investigated through DaCe with the aim of solving the main bottleneck of the current code, but also to make a unified portable solution.

6. Computational Fluid Dynamics: Nek

Within the DEEP-SEA project, the highly scalable computational fluid dynamics (CFD) solver Nek5000 [26] has been chosen as a suitable candidate to utilize the DEEP-SEA software stack. Nek5000 has been used extensively for production runs to obtain high-fidelity turbulence simulations and has scaled to over a million MPI ranks on CPUs. Nek5000 is based on the spectral-element method and uses a so-called matrix-free approach to evaluate linear systems, yielding minimal communication between MPI-ranks and obtaining high-order accuracy in space [27]. These aspects make the spectral-element method particularly appealing as we approach exascale.

However, Nek5000 in its current form uses a legacy codebase in Fortran 77 and static memory allocation. This has limited its use of accelerators, such as GPUs. At KTH a modernized version of Nek5000 has therefore been under development which uses modern Fortran 08 and an object-oriented approach to accommodate different computer architectures. This solver is called Neko and was recently made public[1]. In our efforts to utilize the DEEP system and its booster modules in conjunction with the DEEP-SEA software stack we are integrating Neko into our workflow to accommodate more computer architectures and for easier integration of high-level programming approaches such as DaCe. When referring to Nek5000 and its successor Neko we will use the common term "Nek".

6.1. Tools, software components and programming models

The DEEP-SEA software stack offers several different opportunities for us to explore for Nek. In part, we have the different software developed by our collaborators in other work packages as well as how they synergize in each optimisation cycle.

6.1.1. Benchmarking

For benchmarking we have developed initial JUBE scripts for the mini-app for Nek5000 - Nekbone as well as a wider range of tests for Neko. The JUBE script for Neko compiles the solver for CPUs and GPUs and runs a relevant simulation of the Taylor-Green vortex at Re=1600 with 32 000 elements on 2-16 CPU nodes as well 2-8 Booster nodes on the DEEP system. With these simple tests, the performance improvements that we will achieve through the different optimisation cycles will be clear and also illustrate the correctness of the solver as it is optimized. Integration of the JUBE script into Gitlab runners that run on the DEEP system will also be evaluated.

We have also made a larger scaling study with Neko on 16-64 V100 GPUs on the DEEP system. The very same test has been run on several different supercomputers and computer architectures. With this reference data, we will be able to compare the scalability and performance of Neko over large time scales.

[1] https://github.com/ExtremeFLOW/neko

6.1.2. Optimisation cycles

In this section, we will go through the different optimisation cycles and their applicability to Nek. We will briefly discuss how we can organize our workflow and which ones are most interesting to us.

- **Monitoring optimisation cycle:** Neko will continually be evaluated with LLview and DCDB and compared to historic data. With this information, we will be able to address performance bottlenecks and see how performance relates to the underlying hardware. While not an immediate focus of our work, this optimisation cycle will always be one of the foundations our optimisations rely on.
- **MSA-related optimisation:** Due to issues with Score-P and mpi_f08 the applicability of this optimisation cycle is currently limited. However, by integrating Neko, instead of Nek5000, we now provide full GPU and CPU support. The utilization of different modules concurrently is not fully supported, but might be of interest going forward, even if it is not a priority. The main issue with utilizing different modules is the load balancing between different computer architectures as the code provides no natural way to offload certain tasks to a CPU or GPU, the only difference will be how many elements each node gets.
- **Application mapping toolchain:** Neko currently scales well on both CPUs and GPUs so the mapping to a given allocation mostly revolves around using as many resources as possible. For CPUs, the scaling is linear and the strong scaling limit is usually hard to reach. As long as the number of elements on the GPUs is more than 4000 the scalability is also larger than 70%. To run on a mix of CPU and GPU nodes the load balancing must be addressed before we optimize the mapping.
- **Malleable optimisation cycles:** As for malleability, Neko provides support for checkpoints and a simulation could thus be restarted when more computing resources are available. It also provides support for OS signals so another possible approach would be to rebalance the mesh as more nodes join the job. As the software details of the malleable online monitor are limited we do not yet know which approach will be most effective.
- **Memory management optimisation pipeline** Nek uses large arrays with very few complicated memory patterns. In addition, no significant amount of data is moved between the host and device except during initialization and when performing I/O. However, to establish if this can be easily investigated and whether the performance gains will be deemed significant requires further scrutiny.
- **High-level programming interfaces** In this part of the project we will investigate using DaCe to generate highly optimized kernels for suitable parts of Nek. In particular, we are interested in the formation and evaluation of the linear system Ax in the spectral element method. Efficient evaluation of Ax is essential for high performance in the spectral element method. We are interested in evaluating how DaCe may unveil non-trivial optimisations for the system Ax and the linear solvers used to solve the system.
- **Energy Optimisation** We will evaluate whether BDPO can be used to regulate the energy consumption of an allocated node. Nek is generally memory-bound so it might be possible to reduce, for example, clock speeds to great effect without sacrificing any significant performance.

- **Memory system performance analysis** We might use the memory-system performance analysis to improve the data locality in Nek. We know from experience that memory bandwidth is a key bottleneck and are interested in ways to remedy this.
- **Multi-level simulation approach** Simulating scaling-up our application is interesting, but is highly dependent on the problem size. It is therefore unclear at this moment how much insight multi-level simulations may provide.
- **Analysis of data transfers and memory behaviour** MemAxes appears to be an appealing tool to track suboptimal memory behaviour. MemAxes might help us improve the data locality and communication pattern within Neko
- **Analysis and debugging of MPI-RMA communications** We have previous experience looking at one-sided communications and see the clear potential in improving the strong scalability in Neko. Right now, however, Neko does not make use of one-sided communication. If we extend the communication in Neko to use MPI-RMA we will evaluate how we can use the RMA-Analyser to find errors and bugs in our code.

6.2. Collaboration

Our collaboration with the other workpackages and the topics of collaboration are listed below:

- **WP2**: We are collaborating to instrument our application with Score-P and will work to integrate the Gitlab runners for our JUBE script for Neko.
- **WP4**: Utilize DaCe for computationally demanding tasks in Nek.

We are also connected to the IO-SEA project as our research group participates in that development too. We hope that this collaboration will help us improve the IO behaviour of Nek and other applications.

6.3. Co-design input

We need support for mpi_f08 to profile Neko. As Neko is the variation of Nek5000 which has the largest possibility to accommodate future heterogeneous systems, mpi_f08 support is essential for us to be able to use the tools within DEEP-SEA.

6.4. Conclusion

We have integrated and tested out Neko, our modern version of Nek5000, on the DEEP system to a large extent. We are now aiming to make initial use of DaCe and to evaluate the performance of Neko with the different profilers used in the optimisation cycles. As Neko is evolving fast, we are optimistic that we will be able to use the DEEP software stack to provide a high-performance and malleable CFD solver that is compatible with a variety of heterogeneous computer architectures.

7. Neutron Monte-Carlo Transport for Nuclear Energy

7.1. Introduction

The PATMOS code developed by 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 fuel depletion, as well as thermo-hydraulics and thermo-mechanical feedbacks.

The application is written in C++ and relies on the hybrid MPI+OpenMP programming model to express massive parallelism. In large depletion calculations, the amount of memory required for each MPI rank may be 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 rank on each compute node.

7.2. Tools, software components and programming models

This section details the developments and evolutions that will be made inside the PATMOS application regarding the software stack developed in the DEEP-SEA project. It is organized around the tools, the software components (including programming models) and the optimisation cycles.

7.2.1. Tools

Regarding tools, PATMOS relies on debugging and profiling software to enable further advanced developments. In the DEEP-SEA project, we will use tracing tools, like Score-P [5], and trace visualization/analysis tools, like Scalasca [8]. Furthermore, PATMOS has been integrated into the JUBE system to enable benchmarking and validation.

7.2.2. Software components and programming models

The upcoming main new developments in PATMOS will be relying on the DEEP-SEA software stack and are organized around the intra-node and inter-node features and capabilities of the DEEP system:

1. **Intra-node developments**

   The main new developments of PATMOS in intra-node configuration will be related to memory storage and heterogeneous computing. As mentioned in the introduction of this chapter, PATMOS is written in C++ and relies on MPI and OpenMP parallel-programming models to exploit homogeneous clusters, *i.e.* CPU-only supercomputers with 1 level of memory. The main goal of this development is to add the possibility to exploit the different resources of memory and compute units:

- **Memory-level support**

  Regarding memory storage, the first enhancement will be related to the exploitation of additional memory levels available on recent compute nodes. Because of the overall memory footprint, which can be very large, the ability to store data inside a fast memory tier with a large capacity may lead to performance improvements.

- **GPU support**

  Regarding compute-unit support, the main evolution of PATMOS inside the DEEP-SEA project will be regarding leveraging of GPU-based architectures. This step is mandatory to exploit future generations of supercomputers, especially in MSA clusters. Some parts of PATMOS are compute-intensive enough to port them onto GPU devices in an incremental fashion.

2. **Inter-Node Developments**

   Regarding its inter-node configuration, PATMOS will try to exploit simultaneously a variety of different compute nodes through MSA. One of the previously-described goals related to GPU support will help motivate the leveraging of modular architectures, which expose different kind of compute units. For this purpose, PATMOS will rely on the ability of the MPI layer to adapt to different architectures inside a cluster, for example using MPI gateways. Based on this new feature and the support of the job manager, it will be possible to deploy PATMOS on various compute nodes with different compute-unit and memory-storage configurations.

7.2.3. Optimisation Cycles

This section details the interaction between PATMOS and the various optimisation cycles describes in DEEP-SEA Deliverable D1.3 [1] according to the available features in PATMOS and planned future developments for PATMOS as part of the DEEP-SEA project.

- **Monitoring Optimisation cycle** This is one of the major fundamental cycles. PATMOS will likely use it to help start monitoring information and metrics to feed into the other cycles. For example, this is mandatory to help execute the cycle regarding application mapping related to memory storage.
- **MSA-related optimisation** One goal of PATMOS is to exploit MSA, this cycle will help us achieve this. The cycle analyses communication patterns made by applications and aids in understanding the behaviour of applications on MSA and the corresponding possible optimisations to apply.
- **Application Mapping Toolchain** Exploiting MSA is something that can be optimized from a communication point of view thanks to the previous cycle, but being sure to map the right parts of an application to the most suitable architecture is another important point. This cycle will help in deciding which part should be allocated on a specific architecture. This can be useful after enabling MSA support, memory-level support and GPU support in PATMOS.

- **Malleable Optimisation cycles** As PATMOS is based on Monte Carlo and statistics, the direct application of malleability can be complex, especially during the project lifetime. Indeed, dynamically adapting or redistributing the number of MPI ranks can lead to change in final statistics which would require some modifications in the algorithm and source code.
- **Memory management optimisation pipeline** The exploitation of multiple memory levels can be complex especially when checking where allocations would benefit from being allocated to another memory tier. This cycle will help detect dynamic allocation and porting issues to a dedicated programming model like OpenMP.
- **High-level programming interfaces** PATMOS does not currently exploit task parallelism and it does not include kernels in the Nablab language. Furthermore, because of the Monte Carlo approach, the main physics might not be the most suitable target for Nablab. Therefore, this cycle will not be studied during this project.
- **Energy optimisation** the ability to monitor energy consumption and the possibility to drive such consumption down would be helpful. Unfortunately, this study might be too complex to conduct depending on time availability during the project.
- **Memory system performance analysis** This in-depth analysis would be complementary to the previous cycle dedicated to memory levels, but it can be complex to schedule such a study during the lifetime of this project.
- **Multi-level simulation approach** This cycle could be of interest, but there might be no time during the project.
- **Analysis of data transfers and memory behaviour** This cycle could be of interest, but there might be no time during the project.
- **Analysis and debugging of MPI-RMA communications**: PATMOS does not currently rely on one-sided MPI communications. Therefore, this cycle might not be applicable.

7.3. Collaborations

This section describes the collaborations between PATMOS and the other work packages.

- **WP2** The collaborations will be based on the experience of using the different tools and the benchmark environment of the project through the usage of JUBE for benchmarking PATMOS input sets.
- **WP3** The main collaborations with WP3 will be related to the low-level implementation of important components of the DEEP-SEA software stack. Indeed, the main mechanisms for exploiting MSA and memory-level support will be designed and developed inside this work package of the project. Even if the interface with the application will not be directly trough WP3 there will be interactions to help debug and optimize WP3 approaches.

- **WP4** This work package will provide a functional implementation of the OpenMP compiler/runtime with support for multiple memory tiers. PATMOS will use this specific interface to express memory allocation to be done into others levels of memory. The collaboration will be based on the usage and the optimisation of this interface.
- **WP5** This work package proposes the high-level implementation of all mechanisms related to the best exploitation of MSA. This is one of the major goals of PATMOS in this project: being able to run and leverage the different architectures available inside a cluster. Therefore, this collaboration will focus on the MPI interface for gateways and the exploitation of different compute nodes (including GPUs).

7.4. Co-design input

PATMOS will provide the test cases detailed in Deliverable 1.1 [2] and will fully include them inside the JUBE systems to help extracting metrics and inclusion inside optimisation cycles. During PATMOS development to use the new component provided by the DEEP-SEA project (e.g., OpenMP memory levels or MPI gateway support), PATMOS will update the source code and test cases to exploit these components, while giving feedback to other developers.

On the flip side, PATMOS will expect help to exploit MSA on the technical side and on the main approach to drive the decisions on porting a specific part and/or a specific test case to a dedicated architecture (different compute units and/or memory storage).

7.5. Conclusion

PATMOS provides an existing implementation that already support multiple parallel-programming models (MPI and OpenMP) and test cases that have different requirements in terms of memory consumption and compute power. The main PATMOS developments that will be done to exploit the DEEP-SEA software stack will rely on the ability to exploit the MSA including the communications between the different architectures and the performance inside a cluster (either homogeneous or heterogeneous). This will provide us with several scenarios to help the other work packages when designing and optimizing their components.

8. Earth System Modelling: TSMP

8.1. Introduction

The Terrestrial Systems Modelling Platform (TSMP) is a fully coupled, scale-consistent, highly modular, and massively parallel regional Earth System Model. TSMP is capable of simulating 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 the land surface, the subsurface, and the atmosphere [28]. TSMP is maintained by the Simulation and Data Laboratory Terrestrial Systems (SDLTS) at JSC, and is an open source software publicly available on GitHub [1]. Technically, 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 (v3.9) hydrological model. Coupling is done through the OASIS3-MCT coupler. These *component models*, which are mostly developed by third parties, span different programming languages, parallelization and acceleration solutions, which motivates the modular nature of TSMP. TSMP is flexible, allowing someone to build with one, two or all three component models. TSMP is capable of Data Assimilation through the Parallel Data Assimilation Framework (PDAF). Aditionally, ongoing TSMP developments will interact with DEEP-SEA activities. The most relevant of these will be the introduction of the ICON [29] atmospheric model, which will replace COSMO as an operational system in many European institutions, and the upgrade from CLM3.5 to CLM5. These experimental branches will soon be part of the stable TSMP release, and therefore will also be incorporated into DEEP-SEA activities.

8.2. Tools, software components and programming models

DEEP-SEA offers a number of solutions and tools which are of interest for TSMP to explore. We have identified three key challenges that TSMP faces towards exascale, which are the conceptual framework with which we assess DEEP-SEA's tools, software and programming models.

The first challenge for TSMP is to improve its inter-node parallelism and scalability. A large part of these efforts are around asynchronous communications, I/O, host-device memory exchange, and optimal GPU performance, among others. Process (inter-component model) synchronization overheads, and to a lesser extent intra-process (MPI tasks within a single component) synchronization need to be minimized in order to reduce the costs associated with blocking.

Second, although memory constraints are not currently an issue of large concern in TSMP, there is likely benefit to be found on programming models reducing intra-node and inter-node data moving costs, both in terms of time and energy. TSMP needs support from these new programming models and new technologies for managing memory access overheads which have an increasing impact at scale and Software-Managed Memories to provide more support for runtime and application management of memory. For example, The Mitos wrapper feeds the MemAxes [30] analyses and

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

visualizes outputs that can identify the problematic sections and bottlenecks. These outputs can be the primary source to pursue optimisations or using software-managed memories for TSMP.

The third challenge is evident from TSMP runtime results (see section 16.3 in [3]). These show the static behaviour of mapping model components and their processes to physical resources and clearly show that the different loads of each of the component models can produce long idle times, mostly on the resources allocated to CLM. This leads to the idea of leveraging on a dynamic strategy, enabling dynamic load balancing, higher resource use efficiency, and possibly higher resilience. Moreover, within the MSA concept, different job configurations may lead to very different loads, which are difficult to predict. Consequently, dynamically re-balancing jobs to better satisfy optimal performance (both in terms of runtime and resource use efficiency) is highly relevant for TSMP. Additionally, resource use efficiency translates also into energy use efficiency, a matter of particular concern for Earth system models. Consequently, TSMP expects that solutions such as malleability, which may not necessarily translate into runtime reduction, will increase TSMP's efficiency on the use of the HPC resources. Additionally, enabling malleability in TSMP, will contribute to system-level management (e.g., adaptive power management, transient faults, maintenance), by adaptively remapping logical program constructs to physical resources as well as respond to elastic resources.

8.2.1. Benchmarking

TSMP has been setup within a JUBE workflow for benchmarking, together with two standard cases. One of these cases aims towards testing scalability, albeit on an idealised problem. The second test is a short-runtime version of a production job, ill-suited for extensive scalability tests, but well suited for performance analysis. Both of these tests will be continuously assessed with the tools which are encompassed by the monitoring optimisation cycle and the MSA-related optimisation cycle. The workflow is flexible enough that it will allow to explore performance changes on homogeneous, heterogeneous and modular TSMP jobs.

8.2.2. Optimisation cycles

The DEEP-SEA optimisation cycles are a way to deploy and integrate different solutions in the DEEP-SEA software stack. Herein, we briefly discuss the possible optimisation cycles which are of interest to explore with TSMP, also highlighting priorities and the relationship to different software components within DEEP-SEA.

- **Monitoring optimisation cycle** The monitoring infrastructure collects the first level of information from TSMP runs. Inspecting LLview reports is already standard practice for both production and development runs. As imagined in this optimisation cycle, this information is already used for basic optimisation approaches for TSMP. In particular, in heterogeneous (and in the future, modular) jobs LLview has provided already specific insights for a basic understanding of the different loads and usages that the TSMP component models demand (especially for production-level jobs, which are not strictly benchmarked within DEEP-SEA activities, and for which detailed tracing is very costly).

- **MSA-related optimisation** This optimisation cycle is particularly relevant to understand inter-module communications and load balancing, which is key to the current and foreseeable state of TSMP. Currently, of the three component models only ParFlow is ported to GPUs. Consequently, the expected default modular production jobs have ParFlow running on accelerators, while COSMO and CLM run on CPU modules. To achieve this, it is key to optimise inter-module traffic and inter-module load balancing. So far, profiling and tracing of TSMP via Score-P and Scalasca has been performed and the results are currently being studied to understand wait-state instance statistics and load balancing. Call path profiles are visualized with CUBE. With this information, and the deployment of these activities on the DEEP system, possible optimisations will be evaluated. Moreover, in the future larger jobs in which the inter-module load balancing may need to be re-assessed and re-optimized will also be evaluated.
- **Application Mapping Toolchain (AMT)** The runtime information collected for TSMP in the two previous optimisation cycles can be used together with Extra-P to model possible alternative configuration in the MSA. It could also serve as a guideline for dynamic load balancing, dynamic resource allocations and job malleability limits. Implementing this cycle is planned.
- **Malleable Optimisation cycles** Malleability is of high interest for TSMP. There are several possible levels of malleability. Firstly, when TSMP is restarted, jobs could be resized. TSMP is often run for very long simulation times, which implies a large number of checkpoint restarts (typically following system administrator policies). Full malleability of the jobs could also be of interest. This is particularly true for ensemble runs. These are parallel jobs, all with identical resource allocation, and solving the same problem with small variations. However, it is often the case that some ensemble members (single job) require a longer runtime than the mean of the ensemble. This variability is unpredictable. Since often there is a periodic reduction operation, this hinders the entire ensemble. Consequently, shrinking fast(er) jobs while simultaneously enlarging slow(er) jobs could also be of benefit. Currently, it is still difficult to assess the technical aspects and the effort required to implement this, as malleability in DEEP-SEA is still in early development.
- **High-level programming interfaces cycle** This cycle not of interest for TSMP. We do not support or require Dace or NabLab, nor do we plan reformulations of component model kernels to support Dace or NabLab.
- **Energy optimisation** TSMP aims to be energy efficient. Consequently, documenting and optimizing energy usage is of interest. Particularly, it will be of interest to explore whether heterogeneous/modular jobs are energetically more efficient than homogeneous jobs, aside from potential performance gains, as well as optimising heterogeneous and modular configurations for energy efficiency. Energy monitoring may also lead to possible alternative energy-reducing techniques, *e.g.* underclocking CPU cores assigned to CLM, as these processes are not limiting computations. Implementing this optimisation cycle is not currently a high priority, but will be addressed in the future.
- **Memory-system performance analysis and prediction** Memory performance has not been a main issue for TSMP. However, it is of interest to evaluate the performance within a heterogeneous and modular system (making use of Extrae, PROFET, and Paraver). These metrics are considered in order to reliably evaluate the performance of the model by behaviour-oriented memory-related monitoring. This is not of high priority at the moment, but will be considered later on.

- **Memory management optimisation pipeline** Continuing from the previous cycle, the outputs of the Extrae profiler are fed to the Heterogeneous Memory Advisor (HMem Advisor) to decide the final destination of the allocation, leveraging different allocation calls. This is also not of high priority at the moment but will be considered later on.
- **Multi-level simulation approach** This cycle may allow for further insights in performance analysis and optimisation of a heterogeneous memory system, which is relevant for TSMP. From our point-of-view, a combination of using Extrae, PROFET, Paraver, and Dimemas may be more compatible than MUSA. The BSC software stack is application behaviour-oriented not syntactic or semantic oriented. Although it is not possible for developers to act in real-time on mapping the delays or imbalances to the related functions or part of the code, the performance analysis which collects many computations and MPI parameters seems enough and comprehensive. This is currently not of high priority, and will be re-assessed in the future.
- **Analysis of data transfers and memory behaviour** This cycle is not a priority for TSMP. Nonetheless, identifying potential bottlenecks from collected memory analysis samples may offer further possible optimisation. This will be addressed, if time allows.
- **Analysis and debugging of MPI-RMA communications** RMA is not currently implemented in TSMP. Its implementation in the component models is not foreseen. However, it may be of interest for inter-component communications (i.e., exchanging information between model components), in which case this cycle will be of interest to optimize the use of RMA. This is not currently a priority and will only be addressed if time allows.

8.3. Collaborations

- **WP1** A strong internal collaboration was initiated within WP1 in order to set up the JUBE benchmarking workflows, as reported in Deliverable 1.2.
- **WP2** Ongoing collaborations with WP2 are in place, concerning monitoring, profiling and tracing tools. TSMP is already using Score-P and Scalasca, and we are in the process of implementing workflows with Extrae and Paraver. We expect to have a more intense collaboration when attempting to monitor and instrument modular jobs (so far only heterogeneous jobs have been tested in the JUWELS system). The initial setup and use of selected elements in the software stack and their integration with TSMP is our first priority. the interoperability and composability of the MSA-related optimisation and the detailed acquisitive analysis reports extracted from the multi-level simulation approach is the second plan.
- **WP3** We currently plan to collaborate with WP3 (and WP4) in the continuous integration tasks seeking to leverage on GitLab runners for automated deployment of TSMP on the DEEP prototype, leveraging also on similar plans for automated building and deployment of the DEEP-SEA software stack. This will also require deploying TSMP through virtual machines. We will continue to follow discussions and developments in WP3, in particular those which integrate the SLURM scheduler and malleability.
- **WP4** We will continue to follow the developments in WP4, on future automatic load-balancing between multiple MPI+OpenMP processes in WP4, task-based performance models, and

specifically data placement and flow (T4.1) performance models (T4.2) which may contribute to simplify the analysis of TSMP's inter-component data transfers and memory behaviour.

- **WP5** in the immediate future we will initiate interactions with WP5 around moldability and malleability support. We expect this will require intense exchange, early on to gauge expectation of what is feasible and assess the workload. Later on, intense collaboration will be necessary to implement malleability with TSMP. Additionally, we will continue to follow developments in WP5 about allocation control deliverables as the future solutions of resources orchestration of interest for TSMP including (i) developments in ParaStation MPI towards a more generalized hierarchy awareness with the possibility to consider different information and information sources. (ii) Atos developments on integration of GPU communication in Open MPI hierarchical collective communication. (iii) BSC's developments in the PMIx Tools and the discussions between WP3 and WP5 to define the malleability support.

Importantly, TSMP offers a significant link between DEEP-SEA and IO-SEA as it participates in both projects, with full overlap of participating members. This has already been particularly fruitful in setting up benchmarking workflows (as describe above) which are useful for both DEEP-SEA and IO-SEA. In these JUBE workflows the same benchmarking cases have been implemented, but of course focusing on different benchmarking metrics.

8.4. Co-design input

- **WP2** Concerning Extra-P, we must consider how processes from the multiple component models can be mapped to processes in different computational modules without mixing them (i.e., it is not admissible to the ParFlow processes and run half in a CPU cluster module and half on a GPU module). These processes must be somehow treated as blocks in the mapping and in the performance model. Additionally, since TSMP is theoretically capable of modular jobs, it can be quickly used as a test bed for GPU and modular job monitoring developments in the different tools (Score-P / Scalasca, Extrae, Paraver).
- **WP3/WP5** Co-design activities regarding moldability and malleability. It is very relevant to establish and understand and establish the implications in terms of implementation and the levels at which it must occur (e.g., scheduler level, or in the application itself). Additionally, the decision making workflow needs to be co-designed to address questions such as, which malleability decisions depend on the job owner (application) and which ones depend on system (administrator). Along the same lines, applications should be capable of specifying limits for malleable job sizes, including (but not necessarily limited to) minimum resources and maximum resources beyond which enlarging the job would be inefficient). At the system management level, it is necessary to understand how will priorities within criteria be established to enforce malleability (e.g., based on runtime, resource use efficiency, energy efficiency, rewards system for non-priority shrunken jobs), while complying with the needs of the application.
- **WP4** discuss and understand implications of task-based programming models for MPI+X applications, asynchronous communications, etc. Assess necessary actions from TSMP side.

8.5. Conclusion

TSMP has interest in exploring several of the solutions proposed in the DEEP-SEA software stack. In the shortest term, the use of monitoring and tracing tools (considered within the Monitoring and MSA-related optimisation cycles) are the most relevant, in order to build sufficient know-how, experience and workflows that will allow to assess the modular performance of TSMP. Malleability and dynamic resource allocations are of high interest, and are expected to require a large investment from Task 1.9 and a strong co-design and collaboration effort with WP3 and WP5. Other solutions are also of interest, but of lower priority. Their implementation with TSMP will be continuously re-evaluated as more insights are available from the continuous monitoring and benchmarking TSMP, and new information is available from the solution design teams in other WPs.

9. Summary

This Deliverable describes in detail the tools, software components, and programming model of the applications and outlays the work being done to adapt them to the MSA.

The applications are actively working on the optimisation cycles described in D3.1 [1] to improve their performance. Each of these applications have presented their current status and future plans for each of the optimisation cycles and are actively collaborating with the concerned work packages. In addition, they have identified and listed the tools from WP2-5 that are suited to their requirements. Co-design input on the tools like parallel libraries, programming languages, data recording software, performance analysis tools, and optimisation cycles used in the applications that may improve their performance have been identified and are being worked on in close collaboration with the developers.

Work on a common benchmarking framework comprising of JUBE scripts and GitLab runners, latest I/O developments for fast writing of data files, and use of ephemeral data services to offload I/O operations is being carried in collaboration with IO-SEA. The space weather team is collaborating with RED-SEA to better the inter-connection technology for improved inter-module communications in the MSA.

The applications have now clearly identified the programming paradigms and the tools suited for them and have set up an active collaboration to exploit them to their benefit.

Appendices

A.1. Inter-workpackage and cross-SEA collaboration

Here, we describe our collaboration with different work packages of the DEEP-SEA project and with other SEA projects i.e IO-SEA and RED-SEA. Table 1 describes the topics on which each of the WP1 tasks are collaborating with other DEEP-SEA work packages. Table 2 describes the areas on which WP1 is collaborating with IO-SEA and RED-SEA.

Table 1.: Collaboration between WP1 tasks and other DEEP-SEA work packages.

Tasks	WP2	WP3	WP4	WP5
Space Weather	Profiling studies using Extrae/Paraver and Score-P, extrapolation of application performances to exascale architectures, analysis of memory patterns and accesses using Mem-Axes, and use of novel memory stacking to speed-up xPic. Tracing and modelling the performances of our GPU code.	Using containers for rapid deployment on different systems and for reproducibility. Optimisation of memory management. Tracing of changes and automatic testing using CI. Use of the latest MSA scheduling strategies.	Performance analysis and improvements of our parallel I/O procedures. Exploring the possible use of PIM to accelerate basic operations in xPic.	Updating of our checkpoint/restart procedures using the new available tools. Making use of better data structures in our file systems to reduce AI/ML I/O bottlenecks.
Weather Forecasting	JUBE and performance analysis	MPI-4 and spectral transforms	DLB library	-
Seismic Imaging	Exploit monitoring and tracing tools	Follow memory management toolchain	Look at memory management API	GPI-Space malleability
Molecular Dynamics	JUBE implementation and Score-P	Future collaborations regarding Optimisation Cycles	Using DaCe and collaboration on dlb	Optimizing MPI collectives for GROMACS
Computational Fluid Dynamics	JUBE, Score-P, and GitLab runners	-	Utilize DaCe for Nek	-
Neutron Monte-Carlo Transport	Tools and benchmarking	Low-level mechanisms for MSA	OpenMP memory-level interface	MPI gateways and MSA exploitation
Earth Systems Modelling	Exploit monitoring and tracing tools on modular jobs	GitLab runners, deploying TSMP on virtual machines with DEEP-SEA software stack	Task-based performance models and data placement	Engage in co-design activities for prototype malleability

Table 2.: Cross-SEA collaboration of WP1 with IO-SEA and RED-SEA.

	IO-SEA	RED-SEA
WP1	• Share a common benchmarking framework of JUBE scripts and GitLab runners with task 1.2. • Ongoing collaboration on Space Weather studies to take advantage of the latest I/O developments for fast writing of large files in xPic and for fast reading of data from disk for the training of AI/ML methods in AIDApy. • Collaboration on Computational Fluid Dynamics. • TSMP is actively collaborating with IO-SEA. Synergies have already been exploited in setting up the benchmarking workflows and cases. Plans which involve activities in both DEEP-SEA and IO-SEA in TSMP include making use of ephemeral data services to offload IO operations and allow a higher level of asynchronicity among the different computational modules in the MSA system and TSMP's component models.	• Space weather team collaborates to better interconnection technology in the highly parallel sections of the xPic code and on the inter-module communications in the MSA.

B.2. Optimisation cycles

In this section, we describe the current status of the WP1 applications with regard to the optimisation cycles.

👁 : Looking at using
O : Interested
✓: Will try to implement/Implementing
- : No time/ after the project
N/A: Not Applicable

Table 3.: Optimisation Cycles: Expressions of Interest.

Opt. Cyc.	**xPic**	**AIDApy**	**IFS**	**FRTM**	**BSIT**	**GROMACS**	**Nek5000**	**PATMOS**	**TSMP**
Monitoring	✓	✓	✓	👁	👁	👁	👁	👁	✓
MSA	✓	👁	👁	N/A	-	✓	👁	👁	✓
Map. Tool.	-	👁	👁	N/A	-	👁	👁	👁	O
Malleability	-	👁	👁	👁	👁	✓	👁	👁	✓
Mem. Alloc.	O	👁	O	👁	O	👁	👁	✓	👁
DaCe, Nablab	N/A	N/A	N/A	N/A	N/A	✓	✓	N/A	N/A
Ene. Opt.	✓	✓	👁	👁	👁	👁	👁	👁	👁
HMS	✓	✓	✓	👁	O	👁	👁	👁	👁
MUSA	✓	👁	O	N/A	👁	👁	👁	-	👁
Mem-Axes	👁	✓	O	👁	-	👁	👁	-	👁
RMA Ana.	N/A	N/A	N/A	N/A	-	👁	👁	N/A	O

List of Acronyms and Abbreviations

A

ACE Asynchronous Constraint Execution, a pthread based task scheduler used by the FRTM application

AFSM The All-Flash Storage Module (AFSM) is a purely flash-based storage module of the DEEP system

AI Artificial Intelligence

AIDApy Python package for the analysis of space data developed by the AIDA project

API An Application Programming Interface (API) allows for a software to communicate with other software that support the same API

ACE Asynchronous Constraint Execution framework. A task based scheduling system for heterogeneous architectures

B

BDGS Big Data Generator Suite efficiently generates scalable big data, such as a petabyte (PB) scale, while employing data models to capture and preserve the important characteristics of real data during data generation

BN Booster Node (functional entity)

BoP Board of Partners for the DEEP-SEA project

BSC Barcelona Supercomputing Centre, Spain

BSIT Barcelona Subsurface Imaging Tools is a software platform, designed and developed to fulfill the geophysical exploration needs for HPC applications.

C

CI/CD Continuous Integration/Continuous Deployment (CI/CD) is an automated system for the testing, integration and deployment of software

CLM Community Land Model

CM The Cluster Module (CM) is one of the DEEP-system modules. It consists of 50 CNs.

CN A Cluster Node (CN) consists of two high-end general-purpose CPUs

CORDEX	Coordinated Regional Climate Downscaling Experiment
COSMO	Atmospheric model - Consortium for Small-scale Modelling
CPU	Central Processing Unit
CUDA	The Compute Unified Device Architecture is a parallel computing platform as well as an API that allows for the communication with certain types of graphics-processing units
CEA	French Alternative Energies and Atomic Energy Commission, France

D

DA	Data Analysis
DAM	Data Analytics Module: with nodes (DN) based on general-purpose processors, a large 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
DEEP-SEA	DEEP – Software for Exascale Architectures
DEEP system	Prototype Modular Supercomputer deployed within the DEEP-EST project. It consist of three compute modules and two storage modules.
DN	Nodes of the DAM
DaCe	Data Centric Parallel Programming is a parallel programming framework that takes code in Python/NumPy and other programming languages, and maps it to high-performance CPU, GPU, and FPGA programs, which can be optimized to achieve state-of-the-art.

E

EC	European Commission
ESA	European Space Agency
ESB	Extreme Scale Booster provides an additional 75 power-efficient nodes to the DEEP-EST Prototype, each hosting one Intel Xeon CPU and one NVIDIA V100 GPU, to address the needs of highly scalable codes and adapt them to the computer architectures likely to be used in the Exascale era.
EU	European Union
Exascale	Computer systems or Applications, which are able to run with a performance above 10^{18} Floating point operations per second

Extrae Extrae is a tool that uses different interposition mechanisms to inject probes into the target application to gather information regarding the application performance.

ECMWF European Centre for Medium-range Weather Forecasts, headquartered in Reading, UK

F

FFT Fast Fourier Transform

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

FRTM Fraunhofer RTM is a RTM software package developed by the Fraunhofer Institute for Industrial Mathematics

FWI Full Waveform Inversion (FWI) is a technique for estimating the medium parameters inside a medium of interest by means of the adjoint method. Obtaining gradients, the building block of FWI, is achieved by means of wave equation modelling. In terms of application design, gradient computation is similar to computing one FWI gradient.

G

GitLab GitLab is a platform for software development and information-technology operations. In this project it is used to organize developed software.

GitLab Runner A GitLab Runner is software that connects to GitLab servers for remote execution of CI/CD tasks.

GPFS The General Parallel File System (GPFS) is an IBM developed high-performance clustered file system

GPU Graphics Processing Unit

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

GASPI Global Address Space Programming Interface is a Partitioned Global Address Space (PGAS) API. It aims at extreme scalability, high flexibility and failure tolerance for parallel computing environments.

H

H2020 Horizon 2020

HPC	High Performance Computing
HPCG	The High Performance Conjugate Gradient benchmark is a benchmark based on a conjugate-gradient kernel
HPL	The High Performance LINPACK (HPL) is a performant software package for solving linear system.

I

IB verbs	The API for communication using InfiniBand (IB), a communication hardware
IFS	Integrated Forecasting System
IOR	The Interleaved Or Random (IOR) benchmark is a benchmark for I/O
I/O	Input/Output. May describe the respective logical function of a computer system or a certain physical instantiation
IPC	Instructions Per Cycle

J

JUBE	The JÜlich Benchmarking Environment
JURECA	JURECA (Jülich Research on Exascale Cluster Architectures) Supercomputer at FZJ
JUWELS	JUWELS (Jülich Wizard for European Leadership Science) Supercomputer at FZJ
JSC	Jülich Supercomputing Centre

K

KNL	Knights Landing, second generation of Intel® Xeon Phi (TM)
KU Leuven	Katholieke Universiteit Leuven, Belgium

L

LINPACK	LINPACK is a software package for solving linear systems
LinkTest	LinkTest is a communication-API benchmark
LLNL	Lawrence Livermore National Laboratory

M

mdtest Included with the IOR benchmark, the mdtest benchmark is for benchmarking metadata creation

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

MPI-I/O MPI – Input/Output is an extentsion to MPI for I/O

MPMD Multiple-Program-Multiple-Data

MSA Modular Supercomputer Architecture

ML Machine Learning

MPDATA Multidimensional Positive Definite Advection Transport Algorithm, Smolarkiewicz 1988

MD Molecular Dynamics

MKL The Math Kernel Library is a mathematics library provided by Intel®

N

NASA National Aeronautics and Space Administration

NoSQL NOn relational SQL (NoSQL) databases are SQL databases that can efficiently handle huge amounts of unstructured rapidly changing data. NoSQL unlike SQL does not refer to a language and is generally an adjective

NUMA Non-Uniform Memory Access

NEMO Nucleus for European Modelling of the Ocean, ocean forecasting model used in coupled IFS forecasts

Nek5000 Nek5000 is an open-source computational fluid dynamics code based on the spectral element method.

Nekbone Proxy app for Nek5000. It solves the standard Poisson equation using a conjugate gradient iteration with a simple or spectral element multigrid preconditioner on a block or linear geometry.

O

OMB The Ohio State University (OSU) MicroBenchmarks (OMB) is a suite of MPI benchmarks that has been extended to other communication APIs

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

OTF2 Open Trace Format Version 2

P

PAPADAM **P**roxy **A**pp to **P**lay **A**round with **D**EEP-SEA **A**PI for **M**PI, ECMWF development mini-app

ParaStation MPI ParaStation MPI is the MSA-enabled MPI library and runtime of ParaStation Modulo. It contains a high-performance communication library especially designed for HPC supporting different communication transports concurrently, and offers a complete process management system integrated with the batch queuing system and job scheduler.

Paraver Paraver is a flexible performance analysis tool.

ParFlow Hydrological model

PATMOS Monte Carlo neutron transport parallel application developped by CEA

PDAF Parallel Data Assimilation Framework

PIC Particle-in-Cell algorithm.

PIM Processing In Memory

PME Particle Mesh Ewald

PMT Project Management Team of the DEEP-SEA project

Profile Recording of aggregated information, time measurements, and counts for function calls, bytes transferred, and hardware counters.

PSM2 Performance Scaled Messaging (PSM) 2 is the second generation of the PSM API for communication

R

RAM Random-Access Memory

RDBMS A Relational-DataBase Management System (RDBMS) is a management system for relational databases

RDSMS A Relational-Data–Stream Management System (RDSMS) is a management system for relational data streams

RTM Reverse Time Migration (RTM) is an imaging scheme that uses two related wavefields inside a medium of interest to image said medium. This is commonly achieved by zero-lag cross correlating the two wavefields in time, even though a temporal deconvolution of the two would be theoretically preferred. One of the used wavefields in the medium is commonly estimated from data recorded on the surface of the medium after wavefields were execited in the medium. The other wavefield is often estimated using an approximation of the excitation method that was dominantly responsible for the wavefield that was recorded on the surface of the medium which was used to estimate the other wavefield.

S

Score-P Scalable Performance Measurement Infrastructure for Parallel Codes. It is a software system that provides a measurement infrastructure for profiling, event trace recording, and online analysis of HPC applications.

SQL The Structured Query Language (SQL) is a domain-specific language for accessing data in a RDBMS or in a RDSMS

SSSM Scalable Storage Service Module – conventional, spinning-disk-based storage module of the DEEP system

SW Software

SDLTS Simulation and Data Laboratory Terrestrial Systems

T

TCP The Transmission Control Protocol (TCP) is one of the main communication protocols of the internet protocol

Trace Recording detailed information about significant events during execution of the program and save information using a timestamp, location, and event type.

TSMP Terrestrial System Modelling Platform

TDFD Time-Domain Finite Difference

U

UCP The Unified Communication Protocol (UCP) is an API aimed at unifying different communication APIs, similar in that sense to MPI

V

Vampir A framework that enables developers to quickly display and analyze arbitrary program behavior at any level of detail.

W

WP Work package

WAM WAve Model, used in IFS forecasts to predict the ocean-atmosphere interface

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 many-core processors (TM)
xPic	eXascale ready Particle-in-Cell code for space plasma physics.

Bibliography

[1] S. Pickartz, M. Marazakis, and N. Eicker. *DEEP-SEA Deliverable 3.1: Software Architecture*. Tech. rep. Nov. 2021.

[2] J. H. Meinke and A. Kreuzer. *DEEP-SEA Deliverable 1.1: Initial Application Co-Design Input*. Tech. rep. July 2021.

[3] M. H. J.H. Meinke. *DEEP-SEA Deliverable 1.2: Application use cases and traces*. Tech. rep. Dec. 2021.

[4] Jülich Supercomputing Centre. *JUBE — Jülich Benchmarking Environment*. http://www.fz-juelich.de/jsc/jube. 2008.

[5] A. Knüpfer et al. “Score-P: A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir”. In: *Tools for High Performance Computing 2011*. Ed. by H. Brunst et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 79–91. ISBN: 978-3-642-31475-9. DOI: 10.1007/978-3-642-31476-6_7. URL: http://link.springer.com/10.1007/978-3-642-31476-6_7.

[6] Barcelona Supercomputing Center. *Extrae*. https://tools.bsc.es/paraver. 2006.

[7] Barcelona Supercomputing Center. *Paraver*. https://tools.bsc.es/paraver. 2001.

[8] M. Geimer et al. “The Scalasca performance toolset architecture”. In: *Concurrency and computation: Practice and experience* 22.6 (2010), pp. 702–719.

[9] The HDF Group. *Hierarchical Data Format, version 5*. https://www.hdfgroup.org/HDF5/. 1997.

[10] M. Howison et al. “H5hut: A high-performance I/O library for particle-based simulations”. In: *2010 IEEE International Conference On Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS)*. 2010, pp. 1–8. DOI: 10.1109/CLUSTERWKSP.2010.5613098.

[11] S. Balay et al. *PETSc/TAO Users Manual*. Tech. rep. ANL-21/39 - Revision 3.16. Argonne National Laboratory, 2021.

[12] A. Collette. *Python and HDF5*. O’Reilly, 2013.

[13] J. Ahrens, B. Geveci, and C. Law. “Paraview: An end-user tool for large data visualization”. In: *The visualization handbook* 717.8 (2005).

[14] H. Childs et al. “VisIt: An End-User Tool For Visualizing and Analyzing Very Large Data”. In: *High Performance Visualization–Enabling Extreme-Scale Scientific Insight*. Oct. 2012, pp. 357–372.

[15] V. Tsymbal and A. Kurylev. “Profiling Heterogeneous Computing Performance with VTune Profiler”. In: *International Workshop on OpenCL*. IWOCL’21. Munich, Germany: Association for Computing Machinery, 2021. ISBN: 9781450390330. DOI: 10.1145/3456669.3456678. URL: https://doi.org/10.1145/3456669.3456678.

[16] Barcelona Supercomputing Center. *Paraver*. https://tools.bsc.es/dimemas. 2001.

[17] S. Potluri et al. “Efficient inter-node MPI communication using GPUDirect RDMA for InfiniBand clusters with NVIDIA GPUs”. In: *2013 42nd International Conference on Parallel Processing*. IEEE. 2013, pp. 80–89.

[18] A. Paszke et al. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[19] F. Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[20] W. Frings, F. Wolf, and V. Petkov. “Scalable massively parallel I/O to task-local files”. In: *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. 2009, pp. 1–11.

[21] A. Moody et al. “Design, modeling, and evaluation of a scalable multi-level checkpointing system”. In: *SC’10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2010, pp. 1–11.

[22] A. Tarantola. “Inversion of seismic reflection data in the acoustic approximation”. In: *Geophysics* 8 (Aug. 1984), pp. 1259–1266.

[23] M. J. Abraham et al. “GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers”. In: *SoftwareX* 1-2 (2015), pp. 19–25. ISSN: 2352-7110. DOI: https://doi.org/10.1016/j.softx.2015.06.001. URL: https://www.sciencedirect.com/science/article/pii/S2352711015000059.

[24] C. Kutzner et al. *Best bang for your buck: GPU nodes for GROMACS biomolecular simulations*. 2015.

[25] C. Kutzner et al. “More bang for your buck: Improved use of GPU nodes for GROMACS 2018”. In: *Journal of computational chemistry* 40.27 (2019), pp. 2418–2431.

[26] P. F. Fischer, J. W. Lottes, and S. G. Kerkemeier. *nek5000 Web page*. 2008. URL: https://nek5000.mcs.anl.gov/.

[27] M. O. Deville, P. F. Fischer, and E. Mund. *High-order methods for incompressible fluid flow*. Vol. 9. Cambridge university press, 2002.

[28] P. Shrestha et al. “A scale-consistent terrestrial systems modeling platform based on COSMO, CLM, and ParFlow”. In: *Monthly weather review* 142.9 (2014), pp. 3466–3483.

[29] G. Zängl et al. “The ICON (ICOsahedral Non-hydrostatic) modelling framework of DWD and MPI-M: Description of the non-hydrostatic dynamical core”. In: *Quarterly Journal of the Royal Meteorological Society* 141.687 (June 2014), pp. 563–579. DOI: 10.1002/qj.2378.

[30] A. Giménez et al. “MemAxes: Visualization and Analytics for Characterizing Complex Memory Performance Behaviors”. In: *IEEE Transactions on Visualization and Computer Graphics* 24.7 (2018), pp. 2180–2193. DOI: 10.1109/TVCG.2017.2718532.