EUPEX Deliverable D3.1
Application Analysis Report

Document Properties

Contract Number	101033975
Contractual Deadline	M12 (December 31, 2022)
Dissemination Level	Public
Nature	Report
Edited by	Daniele Cesarini, CINECA
Authors	Daniele Cesarini, Fabio Pitari, Federico Ficarelli, Piero Lanucara, CINECA, Gabriele Cavallaro, FZJ, Emanuele Casarotti, HPC4NDR, Igor Piljić, UNIZG, Olivier Marsden, Ioan Hadade, ECMWF, Luca Tornatore, INAF, Fotis Nikolaidis, Angelos Bilas, FORTH, Erwan Raffin, Antoine Morvan, Atos, Ondrej Meca, Riha Lubomir, VSB-TUO, Theophile Lohier, Cybeletech, Valérie Brenner, Luigi Genovese, Maxime Delorme, CEA
Reviewers	Maike Gilliot, Sergio Saponara
Date	30/12/2021
Keywords	Workflow, Applications, Performance, Analysis, Codesign, Report
Status	Submitted
Release	1.0


This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101033975. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, Germany, Italy, Greece, United Kingdom, Czech Republic, Croatia.

History of Changes

Release	Date	Author, Organization	Description of Changes
0.1	31/03/2022	Daniele Cesarini, CINECA	Content definition
0.2	01/12/2022	Gabriele Cavallaro, FZJ, Fabio Pitari, Federico Ficarelli, Piero Lanucara, CINECA, Emanuele Casarotti, HPC4NDR, Igor Piljić, UNIZG	Added FZJ, CINECA, HPC4NDR, UNIZG contribution
0.3	05/12/2022	Olivier Marsden, Ioan Hadade, ECMWF, Luca Tornatore, INAF, Fotis Nikolaidis, Angelos Bilas, FORTH, Erwan Raffin, Antoine Morvan, Atos	Added ECMWF, INAF, FORTH, Atos contribution
0.4	08/12/2022	Ondrej Meca, Riha Lubomir, VSB-TUO, Theophile Lohier, Cybeletech, Valérie Brenner, Luigi Genovese, Maxime Delorme, CEA	Added VSB-TUO, Cybeletech, and CEA contribution
0.5	19/12/2022	Daniele Cesarini, CINECA	Finalization of the document
1.0	30/12/2022	Daniele Cesarini, CINECA	Final release

Index

Executive Summary

This document reports activities accomplished in the WP3 in the first year of the project. The work done in this document is related from the M0 to the M12 and it is focused on the applications that compose the EUPEX benchmark suite. In particular, we focused the work on the analysis of the use cases that will be used to benchmark the EUPEX pilot platform when it becomes available to the WP3.

For each application that composes the EUPEX benchmark suite, we reported a detailed description and the impact of production and scientific fields. Most of our applications have been identified from the European strategic objectives in the research and innovation activities like the digital twins for: the Destination Earth Initiative, Industry 4.0, the European digital strategy against pandemics, the square kilometres array project and others. In particular, we want to valorise and link of EUPEX project with the centre of excellence founded by EuroHPC Joint Undertaking, which represent several European scientific fields.

During the WP3 work, each application proponent partner has selected a specific use-case designed to be representative of a real production workload, which often represent strategic digital twins for the European Union. Differently from other EuroHPC projects, in EUPEX we target not single applications but entire workflows. An application workflow is often composed as a pipeline of applications and tools that are used in production systems to discover real scientific results. For each application workload, we carefully describe the entire workflow and how different workflow step might be mapped to a modular supercomputing architecture platform such as the EUPEX platform, making good use of the available hardware heterogeneity.

As described in the DoA part B document, the objectives of this deliverable are:

1. Describe the applications and the selected use-cases, as well as a set of mini applications, which implement specific kernels considered relevant to benchmark the EUPEX platform.
2. Report the porting effort of the applications to run on the modular software development vehicles of the project (Fujitsu A64FX and Nvidia V100/A100 as described in the D6.1) and show the learned lessons.
3. Analyse the performance bottlenecks of the initial version of the ported applications.
4. Provide co-design outputs for the EUPEX platform.
5. Provide co-design outputs for the GPU technology selection as outcomes for milestone 3.

In future deliverables, we will optimize the applications for our target SDVs leveraging the modular supercomputing architecture of the EUPEX platform.

Introduction

The success for an HPC architecture of new generation is the capacity to exploit the performance of large-scale and data-centric HPC applications from several scientific fields. For this reason, the applications that composed the EUPEX benchmark suite has been identified to valorise scientific fields that the European Commission has defined strategic to address global problems.

A subset of applications that composed the EUPEX benchmark suite has been selected in the context of the Destination Earth Initiative[1] (DestinE). DestinE aims to develop a highly accurate digital model of the Earth to monitor and predict the interaction between natural phenomena and human activities. In this content, we selected three applications: i) Integrated Forecasting System (IFS) proposed by ECMWF, ii) Forestering Precision Agriculture proposed by Cybeletech, and iii) Machine Learning for Earth Observation (AI4EO) proposed by Forschungszentrum Jülich. The IFS weather forecasting suite is the codebase used at ECMWF for the operational production of medium-range global weather forecasts, Forestering Precision Agriculture is an application that comprises various modules aiming at reproducing the main processes governing plant growth, and to conclude, the AI4EO aims to produce time-series of large-scale and consistent classification maps through the process of learning from multi-source Earth Observation to enable the monitoring of land and ocean processes. In EUPEX, these applications represent some use cases of the Digital Twin of the Earth in the context of DestinE.

Similarly to the DestinE, the second digital twins valorised by EUPEX applications faces the consequences of the effects of future catastrophic earthquakes in the European Union[2]. The use case proposed by HPC4NDR represents a digital twin for seismic resilience, which aims to realize a seismological/engineering probabilistic simulation workflow to describe the effect of an earthquake on civil buildings. HPC4NDR proposed a workflow application mainly based on SPECFEM3D application.

In the context of the European strategy for the Industry 4.0[3], EUPEX project selected an application proposed by VSB-TUO to link this object with the EUPEX project. In detail, VSB-TUO proposed ESPRESO FEM, a highly parallel open-source framework for finite element methods applicable for a wide range of complex engineering simulations for mechanical engineering. Usually, FEM applications are used in Industry 4.0 to predict, analyse, and understand the behaviour or potential behaviour of a product or material under various physical conditions to realize a digital twins of manufacturing systems.

In the wake of the coronavirus crisis, the European Commission's Digital Strategy renewed the importance of digital tools to research and develop diagnostics, treatments, and vaccines[4]. For this reason, a couple of EUPEX applications was selected to link this digital strategy against COVID pandemic with the EUPEX project. We selected two representative applications: i) LiGen proposed by CINECA and ii) BigDFT proposed by CEA, both applications tackle the COVID virus using virtual screening of biochemical compounds to find active molecule against the virus. LiGen uses docking methodology in which the interaction between a three-dimensional pose of a given molecule called "ligand" and a given target site of a protein called "pocket" are evaluated to find possible active interaction, instead BigDFT is used Electronic Structure Calculations of biological systems to find the possible positive effect of a protein against the virus.

The European Commission is making an important contribution supporting the development of the Square Kilometre Array (SKA) telescope[5]. The SKA is widely considered as a human capital development instrument and an innovation driver in many areas of interest to industry and society as a whole.

---

[1] https://digital-strategy.ec.europa.eu/en/policies/destination-earth

[2] https://cordis.europa.eu/article/id/436470-getting-down-to-the-core-of-future-earthquakes-in-europe

[3] https://www.europarl.europa.eu/RegData/etudes/STUD/2016/570007/IPOL_STU(2016)570007_EN.pdf

[4] https://commission.europa.eu/strategy-and-policy/coronavirus-response/digital-solutions-during-pandemic_en

[5] https://ec.europa.eu/research-and-innovation/en/horizon-magazine/ska-time-machine-will-be-able-detect-formation-first-stars-galaxies

EUPEX project want to contribute to this initiative supporting a benchmark use case in the astrophysics context related to the SKA initiative. INAF, which is also involved in the SKA project, proposed OpenGadget3 application that could be effectively coupled to the SKA science domain thanks to its capability of simulating large-scale magnetic fields (which can be used to simulate and interpret SKA-low results from large objects at low frequencies) and the non-equilibrium chemistry of Hydrogen and Helium that offers the unique capability of simulating the evolution of HI regions, that are also a target for the SKA-low and SKA-mid frequencies, gaining valuables insights on the evolution of cosmic structures in the post-reionization era. EUPEX project will use OpenGadget3 to effectively codesign the EUPEX platform and to optimize the application workload.

In the Table 1, we summarize the EUPEX benchmark suite proving: the application name, the scientific field covered, and the proponent partner.

TABLE 1: EUPEX BENCHMARK SUITE

Application	Scientific Field	Proponent partner
Integrated Forecasting System	Climate/Weather forecast	ECMWF
Forestering Precision Agriculture	Climate/Agriculture	Cybeletech
Machine Learning for Earth Observation	Climate/Earth	FZJ
SPECFEM3D	Engineering/Seismological	HPC4NDR
ESPRESO FEM	Engineering/Mechanic	VSB-TUO
LiGen	Biochemical/Pharmacophore	CINECA
BigDFT	Biochemical/Material	CEA
OpenGadget3	Astrophysics	INAF

The strategic objects of the European Commission in the form of digital twins were not only the main characteristics that we used to select the EUPEX applications, but we also take in consideration other two important factors to select representative HPC applications:

- **Exascale readiness:** is the capacity of the application to scale on a larger number of nodes, this was a key factor during the selection. The applications of the EUPEX benchmark suite are capable to use thousand to hundreds of thousands of computing elements, in fact they have been run on the fastest supercomputers into the world. The EUPEX platform will be composed of a relatively small set of nodes, for this reason the applications have been identified with high-scalable characteristic leaving the work in EUPEX to be more focused on the workflow aspects and the architectural efficiency of the EPI processors with respect to the scalability.
- **Application workflow**: real large-scale applications are today composed to several and complex heterogenous steps, which combined, can realize custom workflows to achieve different scientific goals. Each step of the workflow usually required to be executed in a specific order and on different computing architectures to efficiently perform the overall computation. So, the workload characteristic of each workflow step can impact differently on the computing elements of the platform. For this reason, the EUPEX platform has been defined under the design concept of supercomputing modular architecture, but to leverage the MSA characteristics of the platform the application workflow must be mapped in the most efficient way on the system.

On top of the applications that composed the EUPEX benchmark suite, we also selected a couple of mini applications that are interesting for specific porpoises.

The first mini application is Bolt65 proposed by UNIZG. Bolt65 is an implementation of HEVC codec video to test and analyse the video processing capabilities of EUPEX platform. Video processing workload is particular important for large-scale AI applications (Chang 2022) that often need to decode and to encode a video frame respectively in the first and in the last steps of a complex workload before to apply AI methodologies. For this reason, we will use Bolt65 to test and analyse the specific characteristic of the EUPEX platform to process video workloads.

The second mini application is the Dyablo-Whole Sun proposed by CEA. Dyablo-Whole Sun is a simulation code for the modelling of magneto-hydrodynamics on Adaptive Mesh Refinement grids. Adaptive

Mesh Refinement is a numerical method of adapting the accuracy of a solution on a certain important region of the mesh. The code relies on the portability performance library Kokkos (Trott, et al. 2021), which make this mini application doubly interesting for the project. Moreover, Adaptive Mesh Refinement is used on several HPC codes making more interesting the evaluation and optimization of this code even for a large spectre of other HPC codes.

The mini applications don't have the characteristics of the full applications of EUPEX that consider complex workflow with specific scientific use cases, for this reason the effort given by the project on the mini application is less with respect full applications. In the Table 2, we summarize the mini application of EUPEX.

TABLE 2: EUPEX MINI-APPLICATIONS

Mini-Application	Scientific Field	Proponent partner
Bolt65	Video	UNIZG
Dyablo-Whole Sun	Astrophysics	CEA

In the next Chapter, we will present the work done for each application and mini application in order to describe: i) the overall application, ii) the selected use cases that will be used as benchmark in EUPEX, iii) the application workflow, iv) the effort spent on the code porting on ARM architecture, and v) the analysis of the workload characteristics of the application.

Application Analysis

IFS weather forecasting suite

Application Description

The IFS weather forecasting suite is the codebase used at ECMWF for the operational production of medium-range global weather forecasts. It is developed jointly by ECMWF and Meteo-France.

It is a so-called spectral model, its dynamical core being based on the spectral transform method. One part of the dynamics-related computations is performed in spherical-harmonics space, while the rest of the dynamics, along with the physical parameterisations of the atmosphere, are carried out in physical, or grid-point, space. Spectral transforms are used to move from one space to the other. The advection scheme used in the dynamics is semi-Lagrangian, which, combined with a semi-implicit time-stepping scheme, allows for the use of large time steps. The use of such large time steps is critical to the IFS's operational efficiency for medium-range forecast generation.

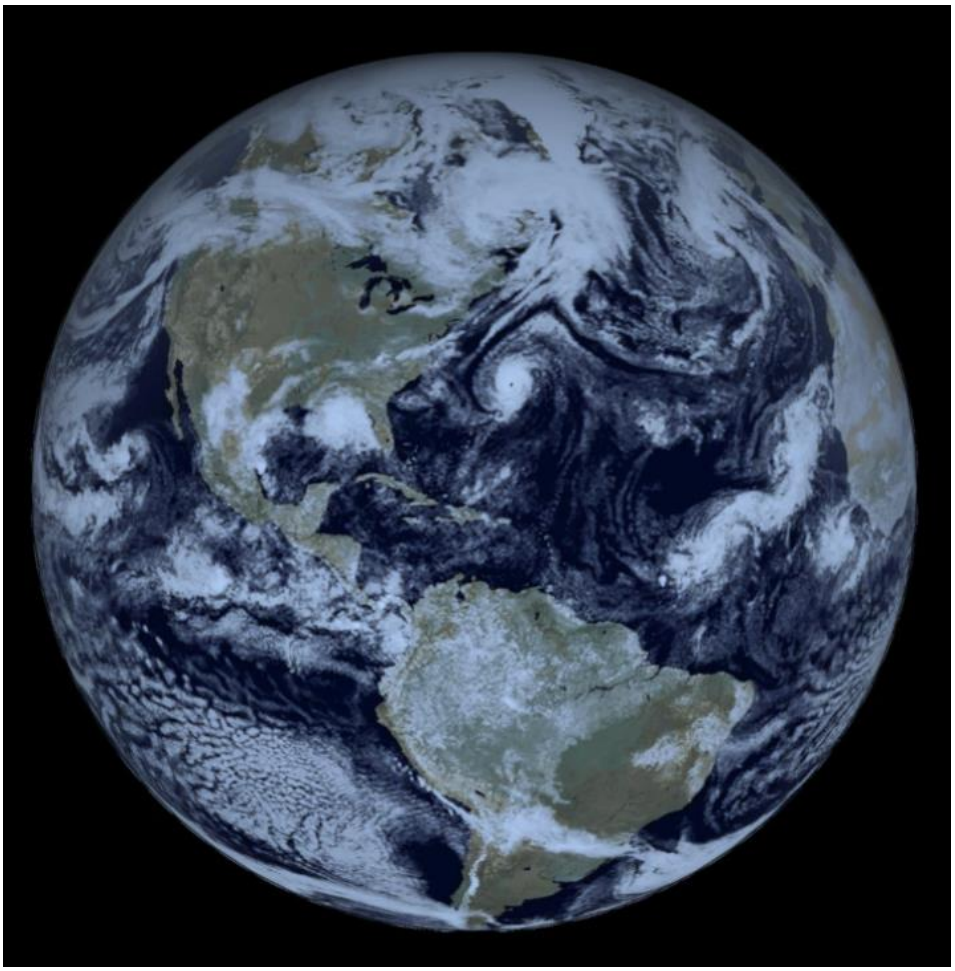


FIGURE 1: TCO1279L137 ENS, 51 MEMBERS, 20200913 00 UTC + 41H

ECMWF's operational global forecasts are used by a number of national weather services to initialise and constrain boundary conditions for their limited area forecasts. They thus contribute to minimising the effect of meteorological conditions on European citizens.

Readying the IFS for future generations of heterogeneous supercomputers is critical to ensuring its position as.the reference application for global weather forecasting.
This means increasing flexibility of the full weather forecasting workflow, in order to make use good use of platform abilities, but also means increasing the portability of the main IFS forecasting task across CPU architectures and accelerators.

The demonstration of a full forecast optimized for the EUPEX platform will be carried out during the project. Along the way, detailed performance assessments and targeted optimisation of IFS components on the EPI GPP chip will be performed.

CLOUDSC mini-app

This mini-app models the effect of cloud micro-physics on temperature and moisture, as well as evaluating the proportion of moisture that is in different possible physical states, e.g. cloud liquid, ice, rain, snow. The cloud microphysical processes are modelled as a one-dimensional problem that happens only in the vertical direction. This assumption allows each atmospheric vertical profile, or column, to be handled entirely independently, with no horizontal dependencies or communication. The assumption of horizontal independence is one that maps to all of the physical parameterisations in the IFS for the foreseeable future of global grid resolutions. Thanks to this, the CLOUDSC mini-app is provides a proxy in terms of compute and memory patterns for a large amount of IFS code, that represents between a quarter and a third (depending on grid resolution) of the cost of an atmospheric forecast.

CLOUDSC has been packaged as a stand-alone mini-app and ported to a variety of platforms including NEC vector processors, GPUs via OpenACC, OpenMP-offload and CUDA, and FPGAs during the EuroEXA H2020 project.

ecTRANS mini-app

This mini-app performs the spectral transforms that transform atmospheric fields from a gridpoint representation to one in spherical harmonics space, and vice versa. This algorithmic step is both a key enabler in IFS's leading time-to-solution capabilities, as well as a communication hotspot due to the sparse global communications it requires.

A direct, or gridpoint-to-spectral, transform starts by the transposition of the global fields, distributed both in the latitudinal and the longitudinal directions, into a collection of latitudes, each of which is owned by a specific MPI task. Each latitude undergoes a spectral transform to obtain its Fourier representation. The resulting set of spectral wavenumbers for each latitude then undergoes the second algorithmic transposition step, so that each wavenumber is owned by a single MPI task. A Legendre transformation is then applied to each wavenumber set, yielding altogether the full spherical representation. The inverse transformation from spectral space back to gridpoint follows the same steps in reverse order.

The global data transpositions are implemented thanks to sparse AlltoAllV calls. These communication calls are highly sensitive to the communication fabric and topology in an HPC system, and constitute the key limiting factor for IFS strong scaling.

The mini-app performs a number of pseudo-timesteps, each consisting of a direct spectral transform of gridded data to spectral space, followed by an inverse spectral transform of data back to gridpoint space.

RAPS IFS forecast

The RAPS package is ECMWF's procurement benchmark suite. It contains the IFS code base, as well as other code needed to perform benchmark-style runs on a machine. This includes software dependencies, a GNU-make-based build system, machine-specific and compiler-specific configurations, and scripts and datasets to run the benchmark configurations themselves.

Application Use Case

The main IFS use case targeted in the EUPEX program is a global forecast on a high-resolution TCO399 spatial grid. This Cubic Octahedral grid has the ability to represent spectral decompositions up to a maximum wavenumber of 399 at the equator, corresponding to an average grid spacing of† around 30

km. Meteorological output from the forecast will be handled by the meteorology-specific object store FDB stack.

The chosen TCO399 grid is sufficiently large to enable scaling demonstrations up to a large number of nodes, while being able to fit in tightly packed mode into a small number of nodes. Although ECMWF's current operational high-resolution deterministic forecast is performed at a higher grid resolution than TCO399, it is the resolution that is used operational in the data assimilation pipeline, meaning it is of direct relevance for current meteorological production.

Application Analysis

Analysis has been undertaken on individual IFS components packaged as mini-apps, as well as on the full RAPS IFS forecast model. An assessment of performance of the mini-apps as well as of the full forecast on the A64FX-based IRENE system is presented in what follows.

CLOUDSC

The CLOUDSC mini-app has been run in its non-distributed (MPI off) threaded (OpenMP) version. Performance assessments on a full as well as an underpopulated socket have been carried out.

Performance is compared that obtained to AMD EPYC 7742 cpus, a Maxeler MAX5 dataflow engine, and an Nvidia A100 GPU. Results for single precision are reported in fig. 32. A similar situation is observed for double precision. EPYC Rome and the A64FX runs were executed with the OpenMP variant of the mini-app, the Maxeler run was based on a manual port to the MaxJ ecosystem, and the A100 GPU run used an OpenACC implementation based on source-to-source translation of the original fortran thanks to the in-house Loki source-to-source tooling. Each architecture was run with tunable parameters (vector length, number of OpenMP threads) set to their best observed value for the platform. The Intel compiler (2021.2.0) was used on AMD, NVHPC 22.3 on Nvidia, and on the A64FX the Fujitsu 4.3.1 compiler suite was used.

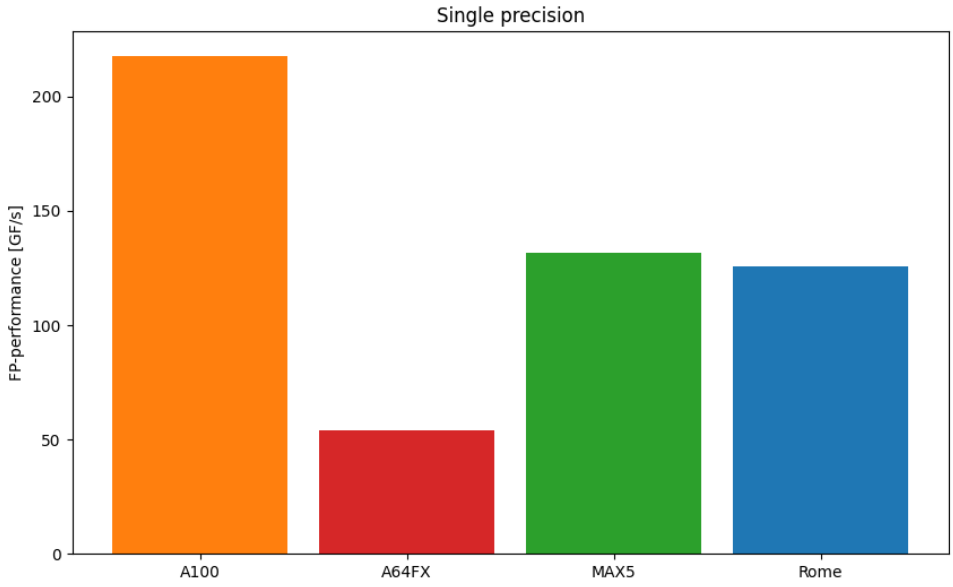


FIGURE 2: CLOUDSC PERFORMANCE ON DIFFERENT HARDWARE

Results for the A64FX fall substantially behind those of the other architectures tested here. Given that the AMD chip and A64FX have very similar theoretical peak performance (3.48 vs 3.38 GFlop) the efficiency observed on the A64FX CPU is 2.4 times lower than that on the AMD EPYC CPU. Preliminary investigations, based on the Fujitsu Fipp profiler, of possible reasons for this relative poor performance suggest that vectorization achieved is lower than on the X86 platform. These results are not reported here as the hardware counter information is currently not limited to the compute kernel but also includes the heavy setup phase. Nevertheless, the early Fipp data, combined with the observation of only small

performance incidence of compiling without the SVE instruction set, suggest that increasing achieved vectorization will be crucial to improving performance.

ecTRANS mini-app

The ecTRANS mini-app has been run on the TCO1279 grid which is currently used for operational weather forecasting at ECMWF. A scaling study up to the maximum node count on IRENE has been undertaken.

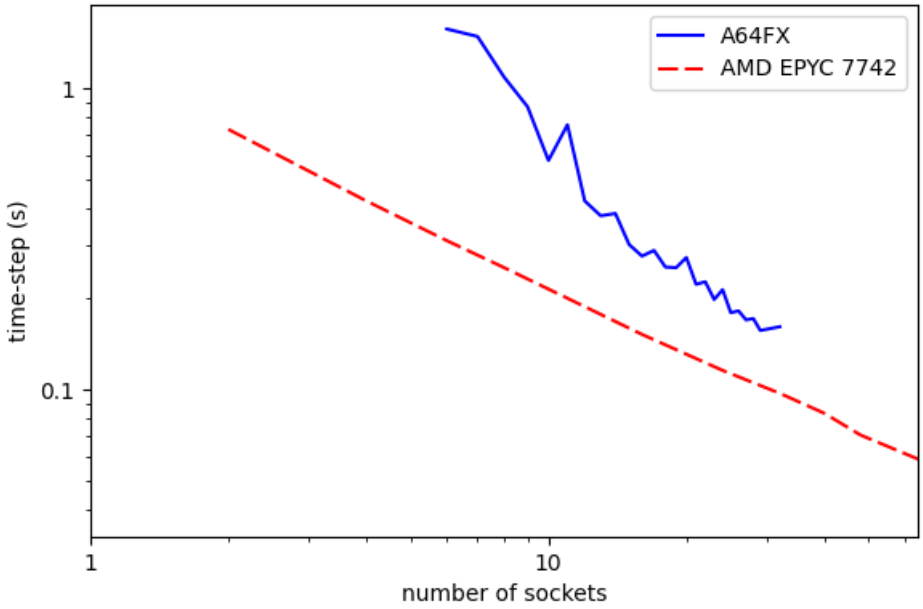


FIGURE 3: ECTRANS RUNNING ON TCO1279 GRID

RAPS IFS forecast

The full IFS forecast model based on release CY47R3 which is currently operational at ECMWF has been compiled and benchmarked on the IRENE system. For compilation, we used the following modules provided by the system: fortran/fujitsu/1.2.0, mpi/openmpi/4.0.5, hdf5/1.12.0, netcdf-c/4.7.4, netcdf-fortran/4.5.3, ssl/1.2.0, cmake/3.20.0 and sqlite/3.33.0. The compilation process was relatively straight forward due to previous experiences in using the Fujitsu compiler on the Fugaku system with only minor code modifications required such as use of TINY and HUGE intrinsics for very large or very small constants along with some issues regarding the use of associate statements. Compilation on an A64FX compute node was very slow and the whole build process took approximately two hours using 12 cores in an interactive session.

We started the performance analysis process by focusing on single core and single node performance. The configuration used for this purpose was a global forecast based on 132 km horizontal grid-point discretization (TCO79) with 137 vertical levels which was small enough to fit into the very tight 24GB memory limit of an A64FX node and large enough to exercise all of the code paths and operations that are found in higher resolutions forecasts currently performed in operations at ECMWF.

The first experiment was to instrument a single task run using the FIPP profiler (Fujitsu Instant Performance Profiler) with the following options: *fipp -C -d $FIPP_DIR -Icall -Inompi -Icpupa ifsMASTER* and parsed the results presented below via: *fipp -A -d $FIPP_DIR -Icall -Inompi -Icpupa*.

```
Performance monitor event:statistics

********************************************************************************
```

```
    Application - performance monitors
*******************************************************************************

  Execution                       Floating-point  Mem throughput  Mem throughput
         time(s)          GFLOPS   peak ratio(%)          (GB/s)   peak ratio(%)
-------------------------------------------------------------------------------
       3014.5771          1.2745          2.2126          0.4134          0.0404   Application
-------------------------------------------------------------------------------
       3014.5771          1.2745          2.2126          0.4134          0.0404   Process      0

  Effective       Floating-point  SIMD inst.      SVE operation
     instruction       operation         rate(%)         rate(%)
-------------------------------------------------------------------------------
    4135861826444    3841964954158         15.5759         85.8298                  Application
-------------------------------------------------------------------------------
    4135861826444    3841964954158         15.5759         85.8298                  Process      0

             IPC            GIPS
-------------------------------------------------------------------------------
          0.7633          1.3720                                                   Application
-------------------------------------------------------------------------------
          0.7633          1.3720                                                   Process      0
```

From the above listing, we can observe that the single task run achieved an average performance of 1.27 GFLOPS with a FP peak ratio of 2% and 0.4 GB/s memory throughput. Since this was a single task run, these figures are not as important as the following which show the % of SIMD instructions issued which is only 15%. However, 85% of those SIMD operations were SVE which leads to an IPC of 0.76. We were not able to use FIPP on runs using multiple MPI ranks and OpenMP threads to drive all of the cores of the A64FX chip however the single core tasks results highlight the fact that the Fujitsu Fortran compiler was only able to vectorize at most 15% of the floating point operations. To achieve good performance on the A64FX architecture, it is imperative that the % of SIMD floating-point instructions is as high as possible versus their scalar counterpart.

The listing below shows the application profile of the same single task run. The highest performance bottleneck is dominated by math library functions (e.g., dexp) which would indicate that a large performance improvement can be obtained by either using a more performant math library or also by trying to vectorize them. From the application side, the largest bottleneck was the radiation calculations followed by the cloud microphysics parameterization (cloudsc) and advection (lascaw).

```
Procedures profile (Total thread cost basis)

*******************************************************************************
    Application - procedures
      Application and Process outputs the total value of the cost of each thread.
      Procedure outputs the total value of the procedure cost of each thread.
*******************************************************************************

          Cost          %  Operation (s)        Barrier          %    Start      End
-------------------------------------------------------------------------------
```

10227	100.0000	1022.6319	3	0.0293	--	--	Application
852	8.3309	85.1943	0	0.0000	--	--	__g_dexp
792	7.7442	79.1947	0	0.0000	--	--	__g_arxr
601	5.8766	60.0960	0	0.0000	--	--	__g_exp
477	4.6641	47.6968	0	0.0000	404	552	radiation_two_stream.calc_
445	4.3512	44.4970	0	0.0000	--	--	dl_gmcpbt_
415	4.0579	41.4972	0	0.0000	68	337	radiation_aerosol_optics.add_aero
403	3.9405	40.2973	0	0.0000	--	--	dl_gmwn1_general_
366	3.5788	36.5976	0	0.0000	--	--	__f_sum
318	3.1094	31.7979	0	0.0000	5	4256	cloudsc_
260	2.5423	25.9983	0	0.0000	2	1187	lascaw_

We have validated the above metrics with perf-stat which showed the same IPC of 0.7 and confirmed that the application is very much memory bound with more than 60% of instructions being stalled in the backend.

```
 Performance counter stats for '/ccc/scratch/cont005/epxt310/hadadeio/RAPS20/flexbuild/bin/ifsMASTER.SP':

 3,747,156,485,580      instructions              #    0.78  insn per cycle
                                                  #    0.77  stalled cycles per insn
 4,813,535,770,988      cycles
 2,899,466,526,164      stalled-cycles-backend    #   60.24% backend cycles idle
   440,236,894,984      stalled-cycles-frontend   #    9.15% frontend cycles idle

    2683.925338695 seconds time elapsed

    2672.906213000 seconds user
       1.339767000 seconds sys
```

The application profile exhibited by the FIPP-instrumented single task run was also confirmed when we profiled a run using 12 MPI tasks that were evenly spread across the 4 CMGs and instrumenting each task with perf record.

```
#         Overhead  Command       Shared Object            Symbol
# ................  ............  .......................  ..................................................................
#
     7.02%   6.47%  ifsMASTER.SP  libfj90f.so.1            [.] __g_dexp
     6.79%   5.55%  ifsMASTER.SP  libfj90f.so.1            [.] __g_arxr
     5.08%   4.87%  ifsMASTER.SP  libfj90f.so.1            [.] __g_exp
     4.44%   2.46%  ifsMASTER.SP  ifsMASTER.SP             [.] dl_gmcpbt_
     4.22%   4.12%  ifsMASTER.SP  ifsMASTER.SP             [.] cloudsc_
     4.09%   3.14%  ifsMASTER.SP  ifsMASTER.SP             [.] radiation_two_stream.calc_reflectance_transmittance_sw_
     3.86%   4.84%  ifsMASTER.SP  ifsMASTER.SP             [.] radiation_aerosol_optics.add_aerosol_optics_
     3.63%   7.36%  ifsMASTER.SP  ifsMASTER.SP             [.] dl_gmnn_kernel_
```

```
    3.35%   2.68%  ifsMASTER.SP  libfj90f.so.1          [.] __f_sum
    1.85%   1.35%  ifsMASTER.SP  libfj90f.so.1          [.] __g_dacos
    1.64%   2.64%  ifsMASTER.SP  ifsMASTER.SP           [.] larmes_
    1.50%   0.43%  ifsMASTER.SP  ifsMASTER.SP           [.] radiation_adding_ica_sw.adding_ica_sw_
    1.47%   2.15%  ifsMASTER.SP  libfj90i.so.1          [.] __jwe_xalc_intrp
    1.43%   1.41%  ifsMASTER.SP  ifsMASTER.SP           [.] cuadjtq_
    1.38%   1.81%  ifsMASTER.SP  ifsMASTER.SP           [.] radiation_two_stream.calc_no_scattering_transmittance_lw_
    1.29%   0.50%  ifsMASTER.SP  ifsMASTER.SP           [.] radiation_ifs_rrtm.gas_optics_
    1.27%   2.54%  ifsMASTER.SP  libc-2.28.so          [.] __memcpy_generic
    1.27%   1.26%  ifsMASTER.SP  ifsMASTER.SP           [.] laitri_weno_
    1.23%   0.70%  ifsMASTER.SP  ifsMASTER.SP           [.] larche_
    1.18%   1.05%  ifsMASTER.SP  libfj90f.so.1          [.] __g_dasin
    1.17%   0.78%  ifsMASTER.SP  ifsMASTER.SP           [.] random_numbers_mix.initialize_random_numbers_
    1.16%   1.57%  ifsMASTER.SP  libuct.so.0.0.0        [.] uct_mm_iface_progress
    1.06%   0.53%  ifsMASTER.SP  ifsMASTER.SP           [.] updecaec_
    1.02%   0.57%  ifsMASTER.SP  ifsMASTER.SP           [.] lascaw_
    0.97%   0.69%  ifsMASTER.SP  ifsMASTER.SP           [.] cubasen_
```

We then began to explore the performance of various combinations of MPI tasks and OpenMP threads to use up all of the available 48 cores on the A64FX node. The Table 3 shows the runtime in seconds and throughput as FDPD (Forecast Days per Day) for each configuration for a 10 day TCO79 L137 (132 km horizontal resolution) forecast. NPROMA is a tuning parameter used in the grid-point space computations which defines the length of the inner loop of those calculations and the blocking factor of the associated data structures.

TABLE 3: RUNTIME IN SECONDS AND THROUGHPUT AS FDPD (FORECAST DAYS PER DAY) FOR EACH CONFIGURATION FOR A 10 DAY TCO79 L137 (132 KM HORIZONTAL RESOLUTION) FORECAST

RESOLUTION	FCLEN	MPI#	OMP#	NPROMA	FDPD	RUNTIME	COMPLETED	REASON
TCO79	10 day	4	12	16	537	1634	YES	N/A
TCO79	10 day	8	6	16	784	1126	YES	N/A
TCO79	10 day	12	4	16	913	1000	YES	N/A
TCO79	10 day	18	3	16	N/A	N/A	NO	OOM
TCO79	10 day	24	2	16	N/A	N/A	NO	OOM

As one can observe from the table above, the best performance is obtained by using as few OpenMP threads as possible. Unfortunately, we were not able to try a full flat MPI configuration with 48 tasks per node due to running out of memory. The highest number of MPI tasks that we could run on one node was 12 tasks and 4 OpenMP threads whereas other combinations such as 18 MPI tasks and 3 OpenMP threads or 24 MPI tasks and 2 OpenMP threads also got OOM killed. The reason for the degradation in performance when using a hybrid configuration vs flat MPI was unclear and experimenting with setting the OpenMP wait policy from passive to active did not make any difference. We instrumented with perf record a run that used 8 MPI tasks and 6 OpenMP threads and compared this profile to an MPI-only run. This is presented in the listing below.

```
# Samples: 840K of events 'anon group { cycles, instructions }'
# Event count (approx.): 5474504012605
#
#         Overhead  Command       Shared Object           Symbol
# ................  ............  ......................  ..................................................................
#
     9.30%   8.92%  ifsMASTER.SP  [unknown]               [k] 0xffff0000109c7e08
```

```
4.92%   5.12%  ifsMASTER.SP  libfj90f.so.1          [.] __g_arxr
4.87%   7.22%  ifsMASTER.SP  libfj90f.so.1          [.] __g_dexp
3.99%   4.87%  ifsMASTER.SP  libfj90f.so.1          [.] __g_exp
3.55%   0.72%  ifsMASTER.SP  [unknown]              [k] 0xffff000010096000
3.03%   3.91%  ifsMASTER.SP  ifsMASTER.SP           [.] cloudsc_
2.77%   2.64%  ifsMASTER.SP  ifsMASTER.SP           [.] radiation_two_stream.calc_reflectance_transmittance_sw_
2.68%   4.59%  ifsMASTER.SP  ifsMASTER.SP           [.] radiation_aerosol_optics.add_aerosol_optics_
2.35%   2.54%  ifsMASTER.SP  libfj90f.so.1          [.] __f_sum
2.22%   2.59%  ifsMASTER.SP  libfj90i.so.1          [.] __jwe_xalc_intrp
1.65%   1.89%  ifsMASTER.SP  libc-2.28.so           [.] __sched_yield
1.57%   1.77%  ifsMASTER.SP  libc-2.28.so           [.] __memcpy_generic
```

The run using 6 OpenMP threads for each MPI task contains two large bottlenecks in the kernel which could not be resolved by perf due to lack of permissions. Running addr2line on these two addresses gives the following:

```
addr2line -f -e ./debug/usr/lib/modules/4.18.0-240.15.1.el8_3.aarch64/vmlinux 0xffff0000109c7e08
__raw_spin_unlock_irq
/usr/src/debug/kernel-4.18.0-240.15.1.el8_3/linux-4.18.0-240.15.1.el8_3.aarch64/./include/linux/spinlock_api_smp.h:169
addr2line -f -e ./debug/usr/lib/modules/4.18.0-240.15.1.el8_3.aarch64/vmlinux 0xffff000010096000
arch_static_branch
/usr/src/debug/kernel-4.18.0-240.15.1.el8_3/linux-4.18.0-240.15.1.el8_3.aarch64/./arch/arm64/include/asm/jump_label.h:31
```

We then continued with a parametric study of finding out the optimal NPROMA value as seen in the Figure below. In contrast to other x86 processors where the optimal NPROMA value is usually twice that of the underlying SIMD register width or equal to the L1 cache line size (e.g., 32 on AMD Rome 7742), on the A64FX, the best performance was obtained with an NPROMA of 128. This is likely due to the much larger cache line size on the A64FX which is in fact a factor of 4 larger than on the AMD Rome 7742.

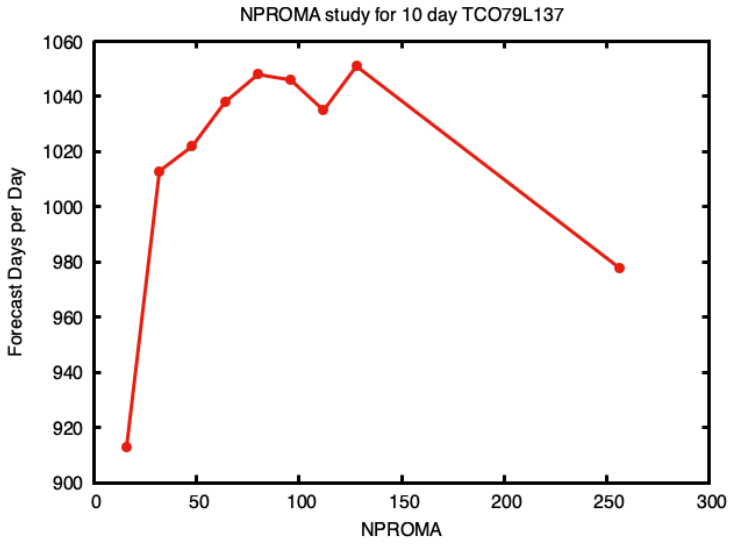


FIGURE 4: NPROMA PARAMETERIC STUDY

The pie charts in Figure 5 present both the overall runtime of a 10 day TCO79 forecast as well as the application profile of each run. The top left pie chart is a reference run performed on a two socket AMD Rome 7742 system (128 cores, 256 GB DDR4 RAM) in a 32 MPI tasks and 4 OpenMP threads configuration. The top right, bottom left and bottom right pie charts show the runs on an IRENE A64FX node in a 4x12, 8x6 and 12x4 configuration where the first number of the number of MPI tasks and the second,

the number of OpenMP threads. When compared to an x86 node, the best performing configuration on the A64FX is still almost a factor of six slower than the AMD Rome node. Unsurprisingly, on the x86 node we manage to obtain an IPC of 1.44 (2x that of A64FX) whilst 30% of floating point instructions issued are SIMD which is also a factor of two higher than on the A64FX.

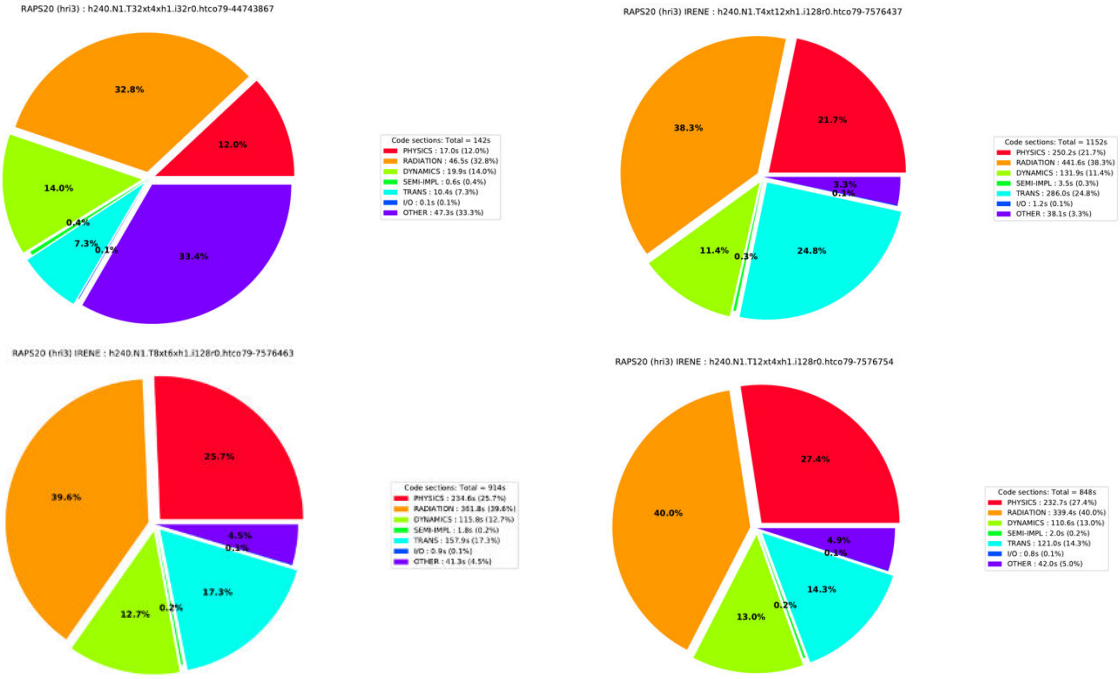


FIGURE 5: OVERALL RUNTIME OF A 10 DAY TCO79 FORECAST AND THE APPLICATION PROFILE

We end our performance analysis with multi-node runs using a TCO399L137 configuration (32 km horizontal discretization) where we can see that throughput scales linearly with the number of nodes. However, due to the limited memory available on the A64FX nodes, we required at least 32 nodes to run this configuration whereas on an x86 host with 256GB of memory per node, such run can be performed on as little as 4 nodes. This further highlights the issues posed by the fact that the A64FX comes with relatively small amount of main memory be it HBM.

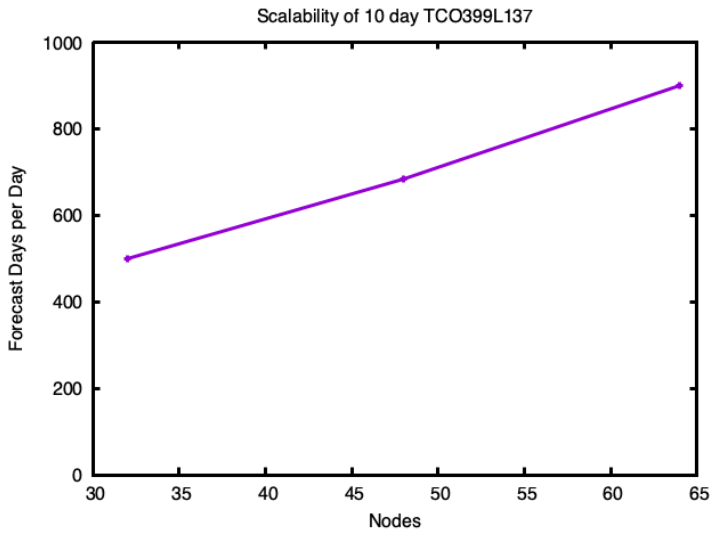


FIGURE 6: TCO399L137 MULTI-NODE SCALABILITY

Conclusion of the Analysis

Two IFS mini-apps, CLOUDSC and ecTRANS, as well as the full RAPS/IFS forecast, have been ported and run on the IRENE A64FX platform. The porting exercise did not hit any particular language issues, the Fujitsu compiler suite handling most of the ECMWF software stack without problem, but CMake handling of the compiler suite and Fujitsu performance libraries, was not ready out of the box.

Performance results observed on the mini-apps was mixed. CLOUDSC performance suggests that vectorization will be a crucial focus of optimization, and OpenMP SIMD behaviour will be investigated. Depending on the results of this investigation, targeted use of intrinsics, not an easy task as not natively available in the Fortran ecosystem, may also be trialled. The ecTRANS spectral transforms mini-app had better relative performance, presumably in large part due to the heavy numerical lifting (FFT, BLAS) being handled by Fujitsu performance libraries. The platform is insufficient in size to conduct a large scaling test, but the scaling performance observed up to 32 nodes was good.

We can draw the same conclusions for the full IFS runs. The inability of the Fujitsu Fortran compiler to vectorize the grid-point calculations (radiation, cloud microphysics) had a very large impact on performance. The largest bottlenecks were the Fujitsu math library functions. A first step in the optimisation process should be to try and vectorize these or trial other math library implementations that support SVE. All of these can and should be first attempted in mini-apps and then ported to the full model. The low amount of memory available on the A64FX node is the biggest concern though when it comes to full model runs. This is especially true at larger resolutions where we are effectively limited by the storage requirements of the Legendre polynomials.

Fostering Precision Agriculture

Application Descriptions

Plant growth is determined by environmental conditions experienced by the plant and by intrinsic plant properties such as its ability to capture and use resources or to endure stress.

Environmental conditions are the combination of experienced aerial climate which is different from one year to another and can vary a lot in a brief time and of pedological context which is relatively stable during the growth season. These two elements constitute the growth environment and decide to a large extent of the plant growth dynamic and thus of the maturity time and of the final yield of the plant, which are usually the targets of growth models.

However, plant genetics also impacts a lot the response of plants to their environment. Two varieties can have very different behaviours when exposed to a stress or combination of stresses resulting in a high variability at harvest. For instance, if a maize crop experiences a drought right during the flowering stage, which is common in the southern European areas, one variety can overcome the stress and maintain an average yield while another will be impacted (fertility reduced, less grains, lower grain weight…).

Globally, the main universal processes governing plant growth can be separated in two groups:

- resource uptake: all the processes allowing plants to take advantage of their immediate environment and gather what they need. The main processes here are photosynthesis, evapotranspiration and root uptake. The first two are associated so the plant can capture what it needs in the atmosphere while maintaining an equilibrium (water vapour, carbon dioxide, etc).
- plant development: all the processes implying a development and a growth of the plant in reaction to its environment. It can be by consuming the previously captured resources or by reaching new phenological stages. The succession of phenological stages results in an evolution in the allocation of the resources to the main growing organs, for instance at the beginning of the plant life leaves are favoured and then the flowers and the fruits.

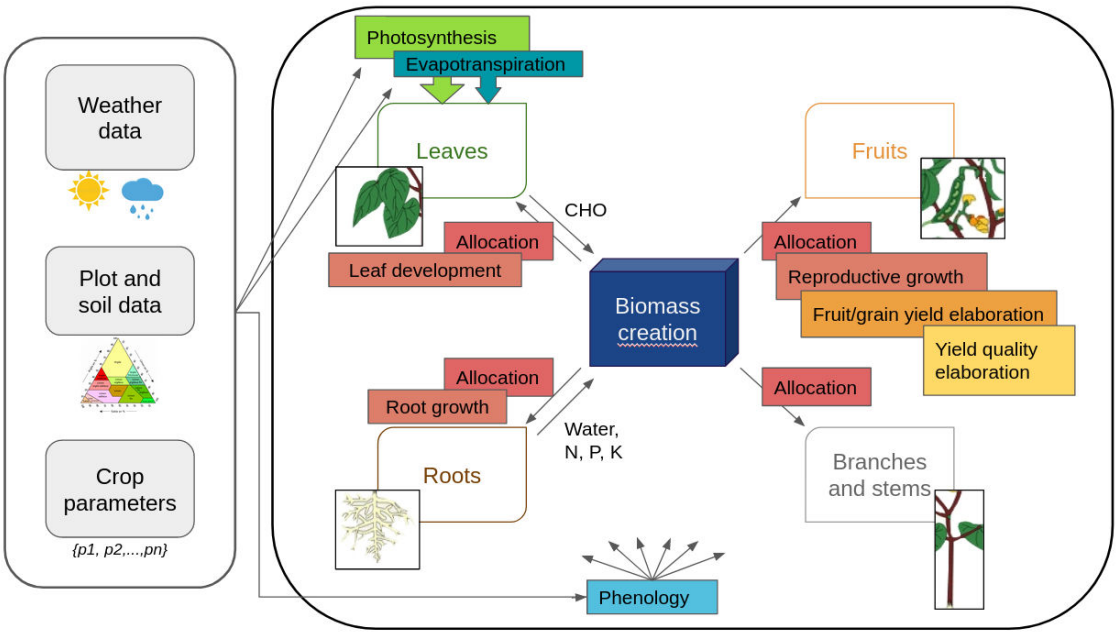


FIGURE 7: PHYSIOLOGICAL PROCESSES AND ENVIRONMENTAL VARIABLES INCLUDED IN PLANT GROWTH MODELS OF CYSIM

The Cybeletech plant growth simulation suit CySim is comprised of various modules aiming at reproducing the main processes governing plant growth. Although there is a huge variety in physiology and growth context amongst the cultivated crops of interest (open field vs greenhouse, annual vs perennial crop, cereals vs oleaginous vs vegetables, etc.), they all rely on the same basic processes to develop, with little subtleties, and the CySim suit thus aims at simulating a wide range of crops and growing contexts.

The CySim suit modules and main variables simulated are the following:

- Root development: root development must be estimated in order to evaluate the amount of water accessible for the plant. This module outputs variables such as the depth reached by root front, soil temperature over the different soil layers as well as a potential hydric stress endured by roots.
- Evapotranspiration: evapotranspiration is the combination of the two main processes leading to water loss from the soil as a system. The first part of it, evaporation, is the passive loss of water to the atmosphere and the second, transpiration, results of a more active mechanism involving plant roots uptake. Evaporation is thus directly linked to daily atmospheric conditions and to irradiance. Knowing these data and the current plant/crop state a potential evapotranspiration is computed and will be used in the water balance module.
- Water balance: to know what amount of water is going to be available for plant uptake on a certain day the water balance must be calculated all over the cycle. The ingoing water flow is due to precipitation and irrigation mainly. The outgoing flow is due to daily plant uptake combined with evaporation (i.e., evapotranspiration) and to drained water leaving to deeper soil layers inaccessible to the plant root system. The main variables simulated with this module are the amount of water available for plant uptake, the amount of drained water and a hydric stress index describing what proportion of its needs the plant will be able to fulfil.
- Phenology: phenology is extremely important as it is the equivalent of a timer over the plant growth cycle, it describes how the plant goes from germination to emergence to vegetative phase to reproductive phase and eventually to senescence. It decides when it is time to pass to the next phenological state and what organs must be focused on at each stage. It has to be calibrated properly as plants do not react in the same way to their environment and to stresses over the different stages. For instance, maize is extremely sensitive to drought during the flowering period so this flowering period must be detected correctly in order to apply the right level of stress in case it happens.
- Leaf development: estimating leaf development and leaf surface is necessary to evaluate the fluxes between the plant and its environment: how much light will be captured; how much stomatal surface is there to realize CO2 and H2O exchange.
- Biomass growth: biomass accumulation can be a truly complex module in terms of computation but it can be simply described as the conversion of all previously gathered resources (water, nutrients, light) into new biomass through photosynthesis mainly.
- Biomass partitioning: the newly created biomass must be allocated to the different sink organs according to their needs. The surplus is kept in memory and stocked to be reused a next day.
- Reproductive growth: reproductive growth is mostly used for all the crops where the organs of interest for yield are grains or fruits. This module outputs the biomass of these reproductive organs and some specific variables like the number of grains or the number of fruits. For more complex models the repartition of the fruits on the plant is also simulated.

Each module takes as input environment variables, state variables estimated by other modules and parameters that describes plant intrinsic properties. Some of these parameters are determined by empiric observations such as phenological stages which can be estimated through multi-environment trials. Some are determined from genomic characteristics given by breeders.

However, several model parameters cannot be estimated directly, and parametric estimation must be performed to accurately simulate plant development. This procedure should be run each time a new species or variety must be simulated. In order to estimate these parameters from in situ measurements CySim integrates parametric estimation tools. These tools work from ground data including in situ plant measurements and corresponding environment historic including climate conditions, plot management data, and soil characteristics.

The CySim suit is used in the Cybeletech greenhouse management decision support system, for instance in tomato greenhouse where it helps to define management policies that reduces CO2 injection.

Another application of the CySim suit is in open field and involves estimating a maturity date and state. It is for instance used in maize to estimate an optimal harvest date as silage maize must be ideally harvested between 32 and 35 % of overall dry matter. Usually, it takes two to three days for a maize crop to gain one dry matter point but during very dry and hot years like in 2022 dry matter evolves much faster and one dry matter point can be taken each day. The optimal harvest period thus comes much faster and pass in only 3-4 days and CySim was useful to help farmers detect it. The silage maize module can use weather forecast and thus also allows to anticipate the optimal harvest period from 2 to 3 weeks which is a real gain in terms of organization for the farmers.

Application Use Case

In Eupex we will focus on the parametric estimation module of Cysim, applied to the tomato plant growth model. The tomato model is a mechanistic model aiming at simulating the growth and development of a tomato plant from seed until all over the harvest season in greenhouse. It is one of the most complex and resource-demanding models included in Cysim. The global architecture of the model is organised around the next main steps:

1. Estimation of intercepted radiations given greenhouse climate data
2. CO2 assimilation through photosynthesis, depending on the amount of light intercepted and of the gaseous environment of the plant.
3. Estimation of stomatal conductance and of leaf resistance to get intrafoliar CO2 rate.
4. Estimation of biomass produced during photosynthesis
5. Biomass allocation to organs (roots, fruits, leaves, stems…)
6. Generation of new organs
7. Implementation of cultural practices (harvest frequency, harvest criterium, leaf stripping…)

The model complexity arises on one hand from the detailed description of physiological processes underlying photosynthesis, and on the other hand because position and characteristics of each organ is described by a L-system. The model thus tends to reproduce the architecture of a real tomato plant and enables to come very close to the reality in reproducing tomato plants and cultural practices surrounding them in greenhouse. It allows to reproduce the weekly harvest in detail like whether the operators criterium to harvest a truss is to wait until the last berry is turning red or until the first fruit has reached a certain size. It also allows to reproduce the weekly leaves stripping criterium like whether a stripping is made when there 15 or 18 leaves on the plant. This point is important because it allows a much finer simulation of the foliar surface and thus of the amount of radiations that will be intercepted and of the eventually produced biomass.

Given the necessary complexity of the tomato model, the number of parameters and their direct and indirect impacts on the different plant variables, the parametric estimation procedure is most demanding in terms of computational power and time.

Cybeletech greenhouse management decision support system is based on Cysim suit and is currently used in tomato greenhouse to help growers to reduce CO2 injection. Indeed, few tools exist to allow greenhouses operators to manipulate and adjust CO2 injection during the day or across the seasons and CO2 can be in largely oversupplied. CO2 is used by the plant for its photosynthesis as a basic material to synthetize more complex molecules. Through photosynthesis, CO2 consumption is thus directly linked to the quantity of light available for the plant. The photosynthesis module included in Cysim tomato growth model enables to identify an optimal CO2 rate to set in the greenhouse in relation to the light available (Figure 8).

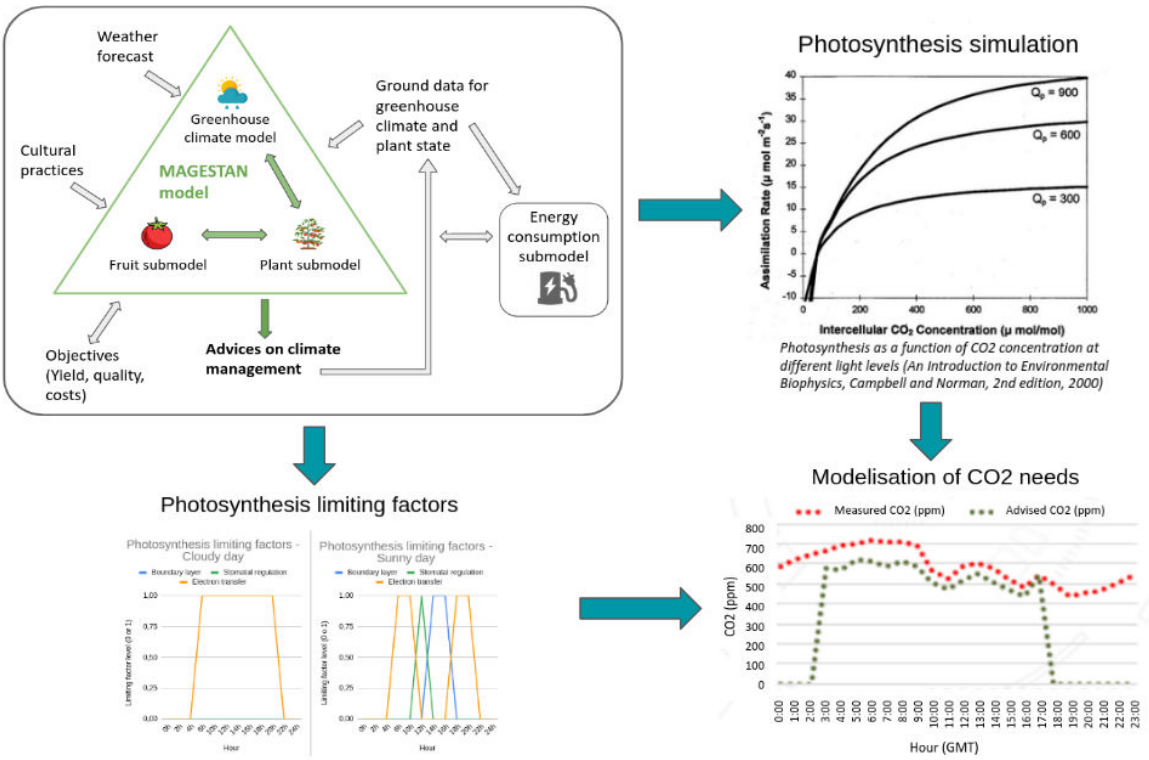


FIGURE 8: OPTIMISATION OF CO2 MANAGEMENT IN GREENHOUSE

The Cysim suit is deployed and run using docker containers on Cybeletech servers, and the setup of parametric estimation is adapted to the computational capacities of these servers. The resulting calibrated model is encouraging with good consistency between observed and simulated plant properties across the season (Figure 9).

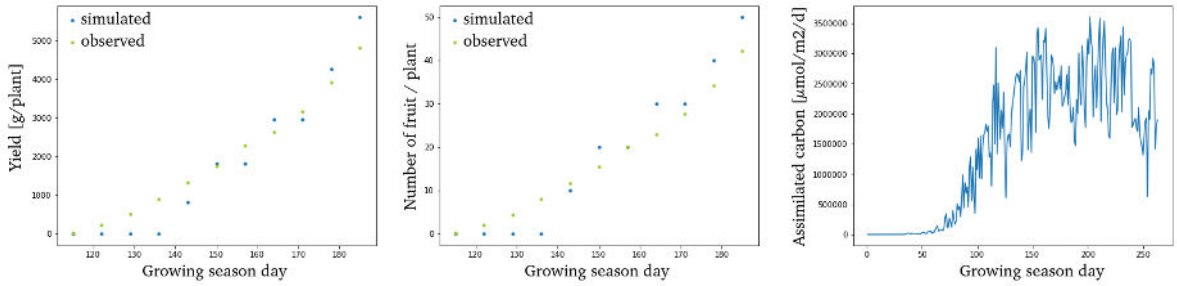


FIGURE 9: RESULT OF CALIBRATED MODEL

During the 2021 season a reduction from 300 g CO2/kg tomato to 75 g CO2/kg tomato has been achieved using the management policies derived from simulation performed with Cysim. This reduction represents a significant economy for the producer as well as a reduction of the environmental impact. However due to computational resource limitations, some important aspects of the plant growth have been overlooked.

The ground measures used to calibrate the model are the yield, the number of fruits harvested, their weight, the number of fruits on the plants and the foliar surface (LAI). Measures are provided at a weekly rate all over the growing season, from January until the beginning of September. Fitting a model on all of these variables while getting the right trajectory over the cycle is needed to support further projects aiming at reducing the environmental cost of such a crop.

Fitting on all these variables implies that most of the processes of the model are involved and so that many parameters must be included in the calibration. To keep the running time reasonable on our servers, a maximum of about 10 000 simulations are run during a calibration with emphasis on 4 parameters. There are about 15 parameters to include in a full calibration to address the processes of interest and we estimate that we would need to run over 1 million simulations to efficiently explore the parameters space.

Workflow Analysis

The workflow for parametric estimation consists of four main steps detailed in Figure 10.

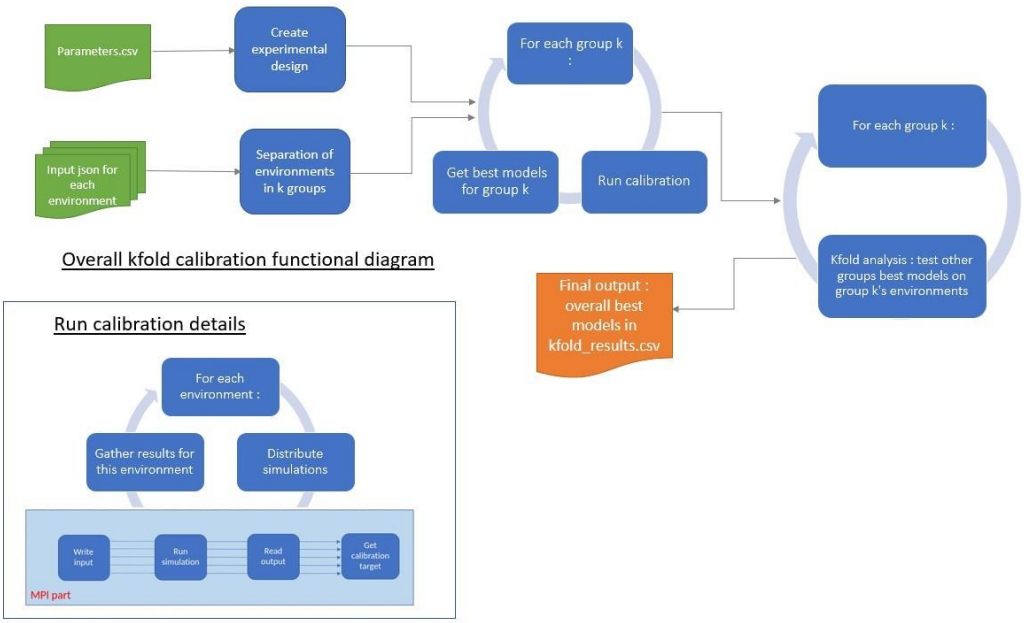
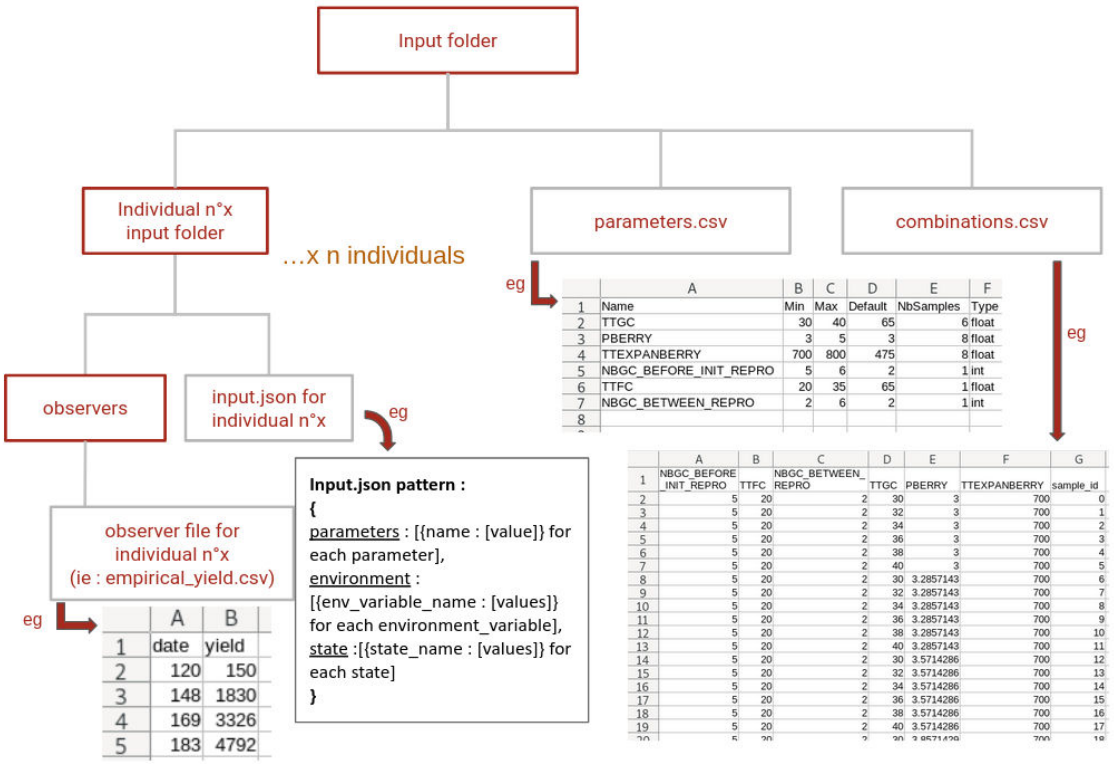


FIGURE 10: WORKFLOW FOR PARAMETRIC ESTIMATION

First, several data sources are gathered to construct the input file describing the plant and its environment. The climate is described by ECMWF atmospheric reanalysis of the global climate covering and/or local sensors, soil characteristics are estimated from the LUCAS (Land Use and Coverage Area frame Survey) survey, and several parameters of the model are extracted from Cybeletech database on crop species properties (Figure 11). This step is implemented in Python and requires standard Python library (e.g., numpy, pandas, pymongo) as well as Cybeletech internal libraries.



	A	B	C	D	E	F
1	Name	Min	Max	Default	NbSamples	Type
2	TTGC	30	40	65	6	float
3	PBERRY	3	5	3	8	float
4	TTEXPANBERRY	700	800	475	8	float
5	NBGC_BEFORE_INIT_REPRO	5	6	2	1	int
6	TTFC	20	35	65	1	float
7	NBGC_BETWEEN_REPRO	2	6	2	1	int

	A	B
1	date	yield
2	120	150
3	148	1830
4	169	3326
5	183	4792

	A	B	C	D	E	F	G
1	NBGC_BEFORE_INIT_REPRO	TTFC	NBGC_BETWEEN_REPRO	TTGC	PBERRY	TTEXPANBERRY	sample_id
2	5	20	2	30	3	700	0
3	5	20	2	32	3	700	1
4	5	20	2	34	3	700	2
5	5	20	2	36	3	700	3
6	5	20	2	38	3	700	4
7	5	20	2	40	3	700	5
8	5	20	2	30	3.2857143	700	6
9	5	20	2	32	3.2857143	700	7
10	5	20	2	34	3.2857143	700	8
11	5	20	2	36	3.2857143	700	9
12	5	20	2	38	3.2857143	700	10
13	5	20	2	40	3.2857143	700	11
14	5	20	2	30	3.5714286	700	12
15	5	20	2	32	3.5714286	700	13
16	5	20	2	34	3.5714286	700	14
17	5	20	2	36	3.5714286	700	15
18	5	20	2	38	3.5714286	700	16
19	5	20	2	40	3.5714286	700	17

FIGURE 11: ARCHITECTURE AND EXAMPLES OF INPUTS FOR PARAMETRIC ESTIMATION

The data from the Cybeletech database on crop species properties are also used to define the range of the parameters to estimate. The experimental design is built based on these extrema and user specifications such as the target number of simulation and the sampling method. The experimental design generation is implemented in Python using scipy.

The experimental design is spitted according to the number of CPU available and the simulation are distributed using MPI. The split and parallelisation are implemented in Python using the library mpi4py while the plant growth model is implemented in C++. The runs of the models are triggered by the Python script using the precompiled .exe. The C++ library for plant growth simulation relies on standard C++ libraries.

The simulation results are gathered and accuracy metrics such as RMSE or DTW distance are computed to evaluate each set of parameters across the different environmental conditions. These results are analysed to identify the parameters set or group of parameters set describing the more accurately the plant response to its environment. Accuracy estimation and model selection is implemented in Python and has standard dependencies.

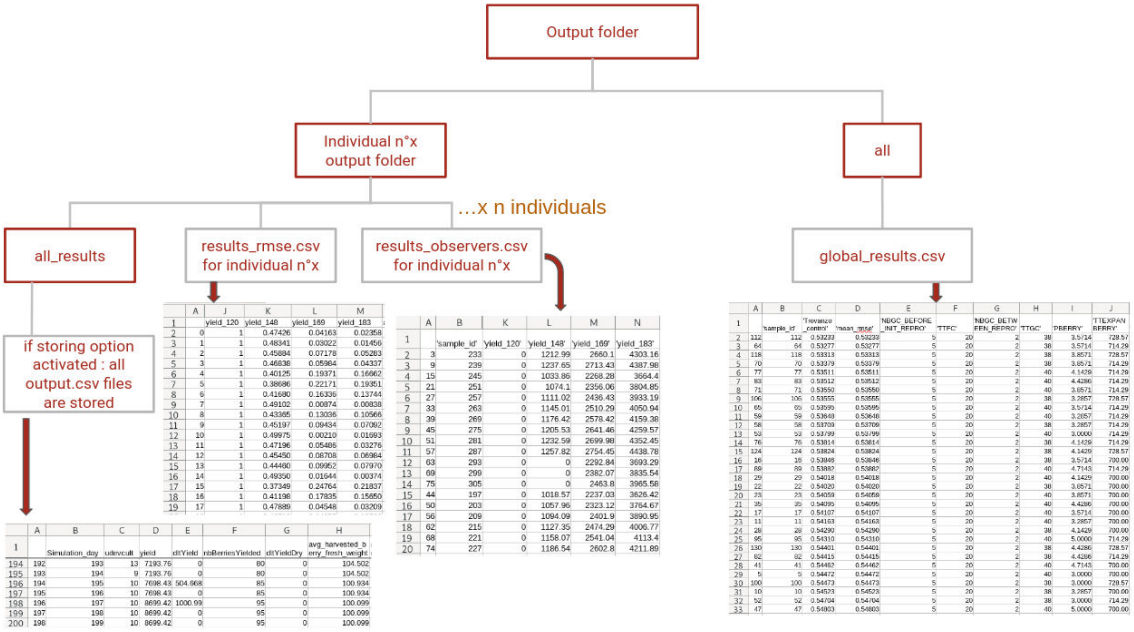


FIGURE 12: ARCHITECTURE AND EXAMPLES OF OUTPUTS FOR PARAMETRIC ESTIMATION

If there are enough individuals, they can be separated in subgroups and calibrations will be performed separately on each of them. The best models of each group will then be tested on other groups in order to identify the statistically most valid among them. If there are not enough individuals a single calibration is performed, and the best models are selected.

The workflow is usually deployed and run using a Docker image embedding both C++ precompiled libraries and executable files for plant growth simulation and Python libraries require to run the workflow.

A Docker image for ARM has been built and deployed on Cybeletech servers. First runs of the workflow have been performed and monitored.

The construction of the model inputs and experimental design generation are straightforward and do not require high computational power and do not consume a lot of memory. Simulations is the step of the workflow having the highest computational demand and runtime (Figure 12). For a batch of simulation, the RAM usage remains very low (< 20MB) and will only marginally increase with the number of target variables.

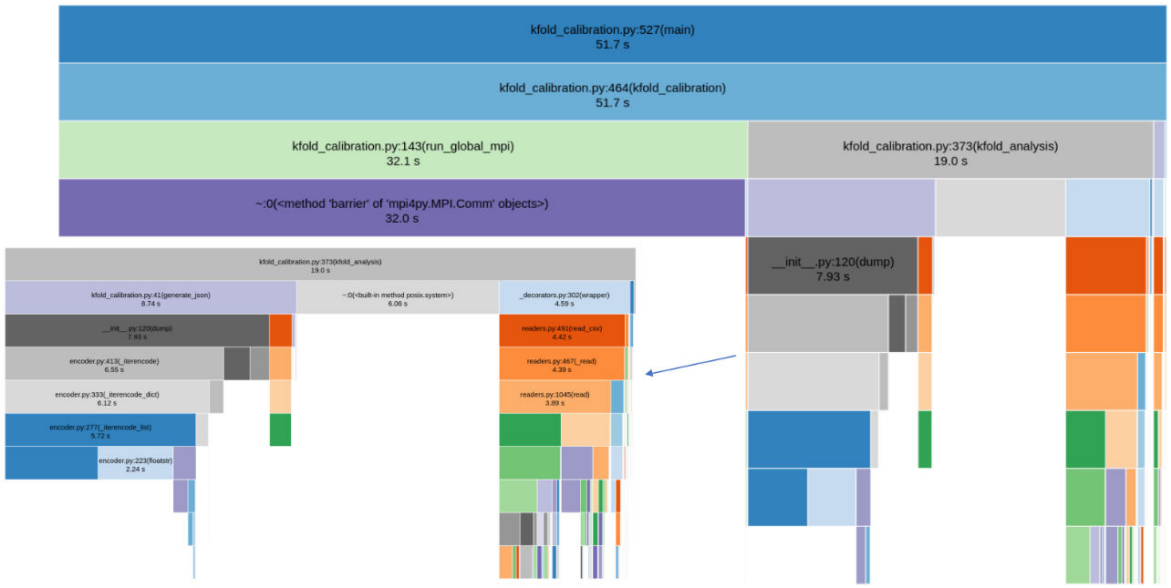


FIGURE 13: RUNTIME AND CPU CONSUMPTION OF THE PARAMETRIC ESTIMATION WORKFLOW

Runtime and CPU consumption scale linearly while RAM usage slightly increases with the size of the experimental design. The speedup is very close to the ideal speedup with an increasing offset due to MPI communication from one hand and to the increase in the simulation analysis step runtime which is not distributed (Figure 13).

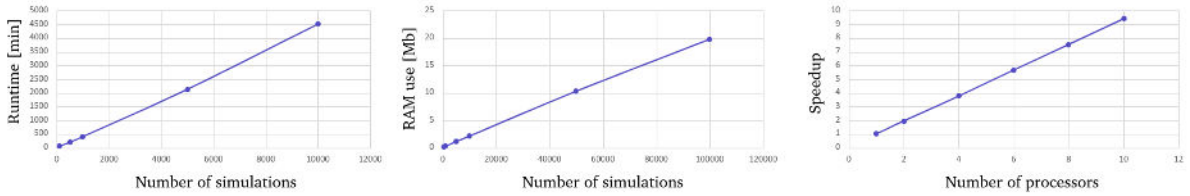


FIGURE 14: RUNTIME, MEMORY USE AND NODE SCALABIITY

Docker container cannot be run directly on the IRENE cluster of TGCC. The ARM docker has then been adapted using Apptainer/Singularity.

Conclusion of the Analysis

The parametric estimation workflow included in the Cysim suit has been profiled with a particular emphasis on execution time in relation to the number of parameters sets explored, which is the number of simulations. Runtime, CPU consumption and memory use scales almost linearly with the number of simulations. The offset is due to the analysis of the simulation results, that according to the method selected may induce extra-consumption of resources. On the other hand, the parametric estimation scales very well with MPI parallelization and the memory consumption per node remains relatively low. Therefore, we don't foresee any resource limitation when running on Eupex architecture.

The docker container currently used to deploy the workflow has been adapted to ARM architecture and to HPC system using Apptainer/Singularity. A first attempt to run the application on A64FX has been made. However, some technical issues remain and will be resolved in the next steps of the project.

Machine Learning for Earth Observation (AI4EO)

Application Description

This application is about the production of time-series of large-scale and consistent classification maps through the process of learning from multi-source Earth Observation (EO) data. It builds upon a classification system (Paris, Bruzzone e Fernández-Prieto 2019) that was deployed in the “S2-4Sci Land and Water - Multitemporal Analysis” project that was supported by ESA in the context of the “Scientific Exploitation of Operational Missions” (SEOM) program. It was used to process a large volume of Satellite images to generate a land cover map of 10m spatial resolution for the whole Italian country. Since 2020, Forschungszentrum Jülich (FZJ) started to operate this processing workflow on the HPC systems of the Jülich Supercomputing Centre (JSC). Furthermore, it initiated several research activities which include the optimization of each processing step with more advanced AI algorithms and more scalable algorithms. This processing workflow is selected for EUPEX as it is both data and compute intensive. It is an application that is not only challenged by the increasing volume and variety of the EO data but also by the high computational complexity of the algorithms involved.

EO is one of the main applications of the Space industry as it enables us to monitor land and ocean processes, to analyse the dynamics at work and in the end monitor the system of the planet. Due to the advent of modern EO programs such as Copernicus of the European Space Agency (ESA) (Aschbacher 2017), a wide variety of open high-resolution and multi-temporal Remote Sensing (RS) data are available at the global level and can be exploited by research communities, agencies and industry. The fleet of satellites of Copernicus (i.e., Sentinels) includes a wide variety of RS instruments (i.e., active and passive sensors with different resolutions) and can acquire more than 12TB of data per day.

The Sentinel-2 mission comprises a constellation of two polar-orbiting satellites that can re-observe a specific geographical area every 5 days under cloud-free conditions. Its data represent an important source of information to better understand the environment and to continuously monitor dynamic processes on Earth's surface. Indeed, one of the most important applications in RS is the production of frequently updated thematic products. Most of them are maps that represent the spatial distribution of identifiable Earth surface features classified in classes (Canty 2014). In particular, they can be associated to land-cover classes (i.e., physical objects that occupy the surface of the Earth, e.g., crop types, kinds of buildings, types of water bodies, etc.) or to land-use classes (i.e., to describe the use of the land surface by humans, e.g., farming fields, industrial areas, etc.).

Classification maps are relevant for several purposes as they allow studying environmental phenomena that affect the Earth' surface (see an example in Figure 15). They can be employed to better understand how human activities influence the environment (Yifang, Gong e Gini 2015), to improve performances of ecosystem, hydrological, and climate models at the global scale, and also essential to understand the global spread of diseases from natural causes.

FIGURE 15: TEMPORAL ANALYSIS OF FLOOD EVENT OF FEBRUARY 2021 AT MÜNSTER, GERMANY. THE BASE LAYER (EARTH'S SURFACE) IS ACQUIRED BY COPERNICUS SENTINEL-2, AND THE FLOOD LAYER IS BASED ON COPERNICUS SENTINEL-1 DATA[6]

The use case represented by this application "Machine Learning for Earth Observation (AI4EO)" provides a processing workflow able to automatically generate classification maps at large scale (i.e., country scale) in an unsupervised way (i.e., assuming that no new annotated training data will be collected). It leverages the availability of: (i) publicly available land-cover maps, (ii) high resolution RS data, and (iii) street-level crowdsourcing geo-tagged images. The complementary information provided by the RS images and the street-level images are used to increase the level of detail of the desired classification scheme as well as to reduce the probability of selecting noisy training samples.

To combine the information provided by the satellite and the street-level images, the workflow leverages the capability of Machine Learning (ML) and Deep Learning (DL) models to extract highly semantic features from both the RS data and crowdsourcing images in order to automatically detect reference samples belonging to the same land-cover class. To enable the production of classification maps at large scale, the ML and DL models that are part of the workflow are based on parallel algorithms that can scale on High Performance Computing (HPC) systems.

Application Use Case

The generation and updating of classification maps can be accurately achieved with supervised or semi-supervised classification techniques (Bruzzone e Demir 2014). In the last decades, many global thematic products have been successfully generated using satellite RS data (Hansen, et al. 2010), (Zhang, et al. 2021). Nevertheless, these approaches require large volumes of annotated samples for training the ML and DL models. In EO, the collection of reliable annotated samples is one of the most demanding issues at operational level. Firstly, the gathering of a sufficient number of high-quality annotated samples is challenging when considering ground surveys or photo-interpretation techniques. While in-situ surveys are expensive and highly time consuming, photo-interpretation is constrained by limited budget for manual annotation. Secondly, collection campaigns might not be feasible in large-scale EO applications (e.g., at country, continental, or global scale). Furthermore, as land-cover classes are related to physical phenomena characterised by specific properties (i.e., spatial variability of classes, non-stationary spectral signatures, etc.), the quality of training samples is always conditioned by the acquisition process and the ground conditions (e.g., atmospheric effects, different viewpoints (Bruzzone e Demir 2014)).

This means that data collected over different locations and time periods results in largely different data distributions (i.e., data shift (Yang e Wu 2006)). This results in real-world application problems: for example, monitoring the same crop field in Germany over a month (temporal difference) can result in entirely different data distribution due to different atmospheric conditions during the acquisition, rendering the learning process challenging (the available labelled datasets become quickly obsolete as new unlabelled data become available to researchers in an unprecedented pace).

In this context, the detailed information provided by the street level images help to understand the properties of the considered training samples in order to better interpret the spectral information associated with the pixel. Figure 16 reports an example of street-level images where rocks are present in the scenes. However, because of the presence of the grassland (see Figure 16 (a)), the spectral signature associated with these samples can be extremely different. A similar issue can be encountered for the samples belonging to crop fields Figure 16 (c-d). Although the street-level-images represent crop fields, the spectral signature associated with these acquisitions will be extremely different because of the different phenological stages of the crops.

[6] Source is Copernicus Emergency Management Service: https://emergency.copernicus.eu/mapping/list-of-components/EMSN092

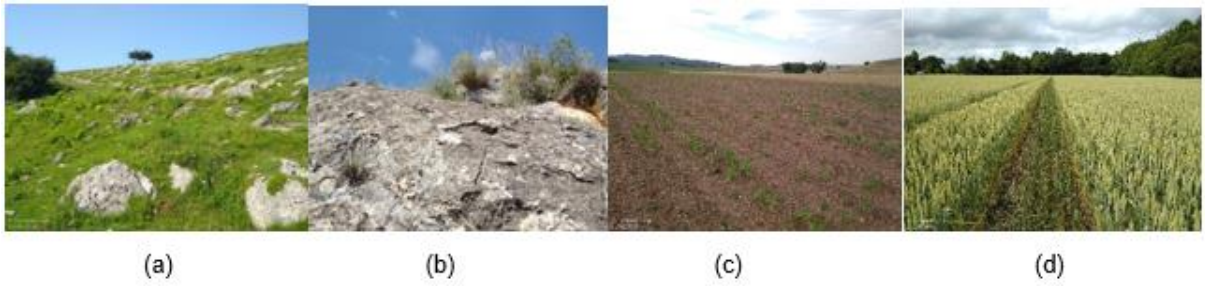


FIGURE 16: EXAMPLES OF STREET LEVEL IMAGES WITH ROCKS PRESENT IN THE SCENE (A-B) AND CROP FIELDS IN DIFFERENT PHENOLOGICAL STAGES (C-D).

Workflow analysis

The workflow is made of different sequential steps, which can be grouped into four overall processes: (A) data retrieval, (B) preprocessing, (C) samples extraction and model training and (D) evaluation and production. These processing steps are detailed below:

Data retrieval: The time series of Sentinel-2 images are retrieved with the sentinelsat Python API. It is a library that allows querying data from the Copernicus Open Access Hub. The CORINE land cover map is gathered from the Copernicus website under the land monitoring service programme.

Preprocessing: The Sentinel-2 images and the CORINE land cover map are reprojected, resampled (i.e., the Sentinel-2 images are resampled to 10m resolution and data composites are generated) and overlapped. The Sentinel-2 images are also atmospherically corrected and clouds are removed through the Sen2Cor tool provided by the European Space Agency (ESA). Furthermore, the semantics of the CORINE legend (i.e., the nomenclatures of the land cover classes) are converted into a set of classes that can be discriminated according to the spectral and temporal properties of Sentinel-2 data. All these tasks are supported by the GDAL library, which is a translator library for raster and vector geospatial data formats (available as a command-line tool and Python API).

Samples extraction and model training: To extract annotated training samples from the preprocessed data, this step uses clustering to identify the samples that have the highest probability of being accurately associated with their land cover classes. This is a necessary step since when adopting publicly available land-cover maps (i.e., CORINE) as the main source of knowledge, inconsistent labels may be chosen for a variety of reasons (Congalton, et al. 2014). The output of this step is the annotated dataset that is used to train a classifier. This can be either a classical ML method (e.g., Random Forest (RF), Support Vector Machine (SVM) or a DL one (Convolutional Neural Network (CNN), Long short-term memory (LSTM), Transformers).

Evaluation and production: this final step includes production of the final classification map (i.e., by feeding to the trained classifier a new time series of Sentinel-2 images) and several validation steps to minimise classification errors.

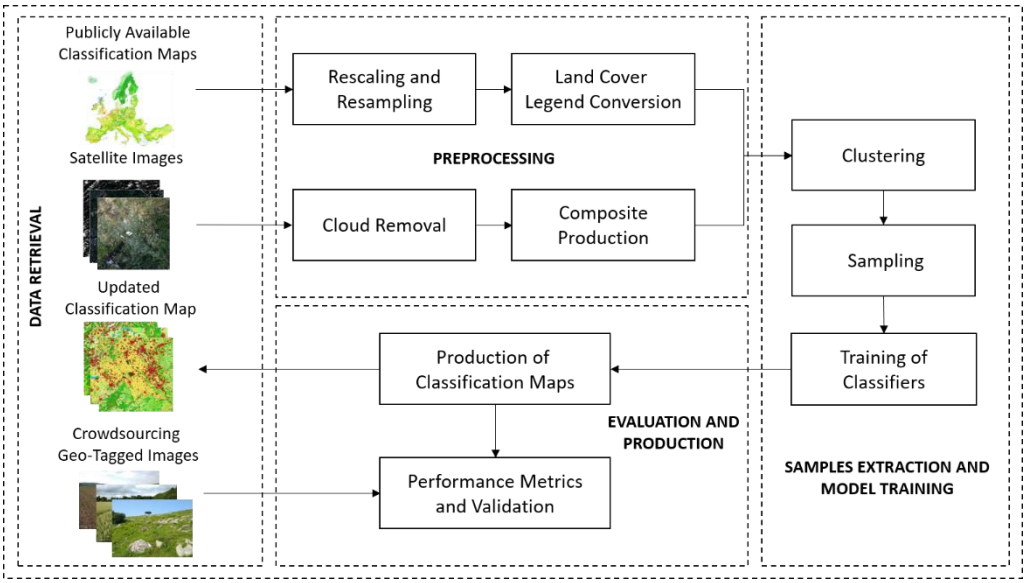


FIGURE 17: AI4EO WORKFLOW

In this deliverable we report the results obtained in EUPEX for the algorithms used in the "samples extraction and model training" step, namely the clustering and the training of the DL model. We focused on these two as they represent the most compute-intensive algorithms of our framework.

Workflow analysis of Clustering

In this section, we consider two clustering algorithms, namely K-Means (Lloyd 1982) and Density-based spatial clustering of applications with noise (DBSCAN) (Ester, et al. 1996). The input dataset to be clustered consists of 1836040 samples, with each sample defined by 5 features.

Clustering using the K-Means algorithm of scikit-learn

The K-Means clustering algorithm is one of the simplest clustering and computationally efficient clustering algorithms. It is based on recursively computing the distance between each point in the dataset with the centroids of K clusters and then assigning the point to the closest centroid. We used the cluster module from the scikit-learn package for clustering using the K Means algorithm.

Porting of application to IRENE at TGCC

We ported the "preprocessing" and "samples extraction and model training" steps of our application to IRENE. Hereon we focus our attention on clustering, as it is one of the key components of the application, and a time-consuming one. We disregarded the "data retrieval" step and transferred the input dataset from Jülich Storage Cluster (JUST), as there is no direct internet connection between the machines hosted at TGCC and the outside. For the same reason, we requested the system administrators to install a number of Python packages that were crucial to the application, among which GDAL, geopandas, scikit-learn, scipy. The Python packages and ScoreP were installed on IRENE as a module and can be loaded via:

```
$ ml load python3/3.8.10
```

The code snapshot for the usage of the KMeans module of scikit-learn is given below.

```
import KMeans module
from sklearn.cluster import KMeans
```

```
kmeans_model = KMeans(
            n_clusters=clusters_for_class,
            random_state=configuration.SAMPLES_KMEANS_RANDOM_STATE,
            max_iter=configuration.SAMPLES_KMEANS_MAX_ITERATIONS)


kmeans_model.fit(indexes_matrix_normalized[clc_indexes_for_class])
```

Note that in the code snapshot, the 'indexes_matrix_normalized[clc_indexes_for_class]' contains the spectral indices for each point in the dataset.

The input and output data of the application are saved in the $STORAGE partition and copied also to $SCRATCH for a faster execution.

The algorithm took about 84 seconds to cluster the dataset of 1836040 entries on the A64fX partition of IRENE. As the clustering process was carried out only in one node, we could not make use of the parallelism offered by the A64FX cluster. Hence, we decided to employ a different clustering algorithm (i.e., DBSCAN) for which we have a parallel implementation available.

HPDBSCAN

Introduction

Highly parallel DBSCAN (HPDBSCAN) (Götz, Bodenstein e Riedel 2015) is a highly parallel implementation of the established DBSCAN clustering algorithm developed at the FZJ. It takes points of an arbitrary dimension as one of its input arguments and correctly labels each point to a cluster ID or identifies it as noise. The DBSCAN algorithm determines this through the use of input parameters: the minimal number of points required to form a cluster and the maximum neighbourhood search radius. HPDBSCAN is used in this workflow to scale the clustering on multiple nodes. We originally used k-means in an embarrassing parallel fashion, clustering each tile independently on one node. However, in the original approach to scaling lies the intrinsic limitation of the number of tiles of interest. The number of tiles is in fact limited, and an embarrassing parallel approach would stop scaling soon (f.e., the Netherlands are covered by 10 Sentinel-2 tiles, implying that we would scale clustering on 10 nodes with the original approach).

The proposed approach using HPDBSCAN is based on a fusion of the pre-processed data of each tile. The clustering can then be scaled on an arbitrary large number of nodes, independently on the number of tiles that are actually considered.

The correctness of the clustering procedure is based on two required input values: the minimal number of points required to form a cluster and the maximum neighbourhood search radius. Thus, the HPDBSCAN's executable, DBSCAN, has three required input parameters:

- minpoints, the minimum amount of points required to form a cluster,
- epsilon, the maximum neighbourhood search radius for each point,
- filename, a file reference to the input dataset.

Installation and application setup

The algorithm is written in C++ (GNU version 4.9.x and up) and is optimised for high parallelism, using both shared and distributed memory parallelism via OpenMP and MPI respectively. Further parallelism is also employed by exploiting the HDF5 API when the input data is read and the output data is written. The application is expected to scale well and is able to practically operate on any number of nodes, cores and threads. It mostly uses standard C/C++ libraries but in addition is also requires:

- HDF5 C/C++ library
- OpenMP
- MPI

- ArrayFire

Input files for HPDBSCAN must adhere to the HDF5 format. The HDF5 API is used by each MPI process to access the input data in parallel, where the data is divided into equally sized chunks corresponding to the number of MPI processes, thus enabling each MPI process to read just the chunk it is going to process. HPDBSCAN labels each point to a cluster ID and writes the labels for each point into the input file near the end of its execution.

Additionally, it writes the execution time of each part of the application and its status to the standard output, as can be seen in the Figure 18.

```
Computing cell space
        Computing dimensions ... [OK] in 0.007702
        Computing cells ...      [OK] in 0.006389
        Sorting points ...       [OK] in 0.104052
        Distributing points ...  [OK] in 0.132429
Clustering ...
        DBSCAN ...               [OK] in 5701.464165
        Merging halos ...        [OK] in 0.071922
        Appyling rules ...       [OK] in 0.000512
        Recovering order ...     [OK] in 0.669348
Summary ...
        Clusters:       2
        Cluster points: 1836031
        Noise points:   9
        Core points:    1836031
```

FIGURE 18: HPDBSCAN´S STANDARD OUTPUT, USING THE RS DATASET RUNNING ON THE IRENE'S A64FX PARTITION WITH N CORES

Intermediary time-measurements are provided for each relevant part of the application, as well as the whole execution.

To summarise, the HPDBSCAN´s execution is divided into six steps:

- The entire RS dataset is divided into equal-size chunks and loaded by all processors.
- The data is pre-processed where each point is sorted into a virtual, unique, spatial cell corresponding to their location within the data space.
- The sorted data workload is balanced via a heuristic and redistributed to a distinct number of cells from the hyper grid.
- Clustering of the assigned cells is performed locally by each processor.
- Local results are merged into one global result.
- Cluster relabelling rules from the previous step are broadcasted and applied locally.

The single program, multiple data (SPMD) approach is used for the computation. The application consists of a single program that distributes tasks to multiple processors that run simultaneously. As previously mentioned, HPDBSCAN uses both MPI and OpenMP to communicate across nodes and cores, respectively. The communication profile is heavily dependent on the makeup of the dataset that is being processed. Specifically, data is distributed among all nodes using MPI_Alltoall and gathered with MPI_Gather.

The memory requirements depend on the initial execution parameters, size and make of the dataset. In general, a “standard amount” of memory and bandwidth should suffice. More specifically, the actual memory footprint is O(log(n)) for the indexing step plus O(n/p) for redistribution where n is the total number of points in the dataset and p is the number of processors used.

HPDBSCAN uses datasets stored in files with the HDF5 format. The size of these files can range from a few hundred MBs to tens of GBytes, depending on the number of dimensions, size and resolution of the data. Each point in the dataset has 8 bytes per dimension (F64), i.e. 40 bytes are required for a single point in 5 dimensions. At the end of its execution, the application appends the cluster ID labels to the dataset file.

HPDBSCAN is rather a rigid application and will not be able to utilise additional resources that become available during its execution. Moreover, a reduction of cores will be fatal for its execution. However, it can be executed with an arbitrary number of resources with no practical limit.

HPDBSCAN is not a resilient application. It expects the hardware and its access to it to remain static throughout its execution. The cores that are assigned to HPDBSCAN during the start of its execution must remain unchanged until the application finishes its execution, otherwise an error will occur and the allocated resource time will be wasted.

Application setup

HPDBSCAN requires the following dependencies in order for successful compilation of the code.

- CMake 3.10+
- C++11 compliant compiler (e.g. g++ 4.9+)
- OpenMP 4.0+ (e.g. g++ 4.9+)
- HDF5 1.8+

The following modules were loaded in IRENE to meet the requirements for installation of HPDBSCAN.

```
$ ml cmake/3.20.0
$ ml hdf5/1.12.0
$ ml mpi/openmpi/4.0.5.2
```

The input dataset which was originally in CSV format was converted to HDF5 format using the python packages of numpy and h5py.

```
import h5py
import numpy as np
data = np.genfromtxt('data.csv', delimiter=',')[:,:].astype(np.float32)
   with h5py.File('iris.h5', 'w') as handle:
      handle['DBSCAN'] = data
```

To run the application without MPI (on a single node), the following command is used.

```
./hpdbscan -t <THREADS> <PATH_TO_HDF5_FILE>
```

To run the application using MPI

```
mpirun -np <NODES> ./hpdbscan -t <THREADS> <PATH_TO_HDF5_FILE>
```

Application run and profiling results

The performance metrics were evaluated on a reference run which was a single threaded run on a single node. Here are the performance metrics for the reference (single node, single threaded) run. The density threshold and the spatial search radius were given the default values of 2 and 0.1 respectively.

- Execution time: 8056.70s
- FLOPS:124.44 Mflops
- Cache Hit/Miss ratio is 0.013 i.e 1.3% of the L1 data cache references were misses.

The execution time and FLOPS were measured using the high level PAPI application program interfaces.

```
if((retval=PAPI_flops(&realTime,&procTime,&flpins,&mflops)) < PAPI_OK) {
   std::out<<"Your platform may not support floating point operation
            event"<<endl;
   exit(1);
}
PAPI_flops(&realTime,&procTime,&flpins,&mflops);
PAPI_ipc(&realTime,&procTime,&ins,&ipc);
/*
 *Call to HPDBSCAN
```

```
*/
PAPI_flops(&realTime,&procTime,&flpins,&mflops);
PAPI_ipc(&realTime,&procTime,&ins,&ipc);
```

Thread scalability

The thread scalability of HPDBSCAN application on a single node of the A64FX cluster is illustrated in Figure 19. In order to study the thread scalability, only one node has been used and no MPI tasks were created. The number of OpenMP threads was increased from 1 to 48. As shown in the plot, there is a linear increase in the speedup when using more than 8 threads. The speedup for the scalability plot was measured as t(i) where t(i) is the time taken for 'i' threads and 't' is the time taken for a single threaded execution which is taken as the reference run.

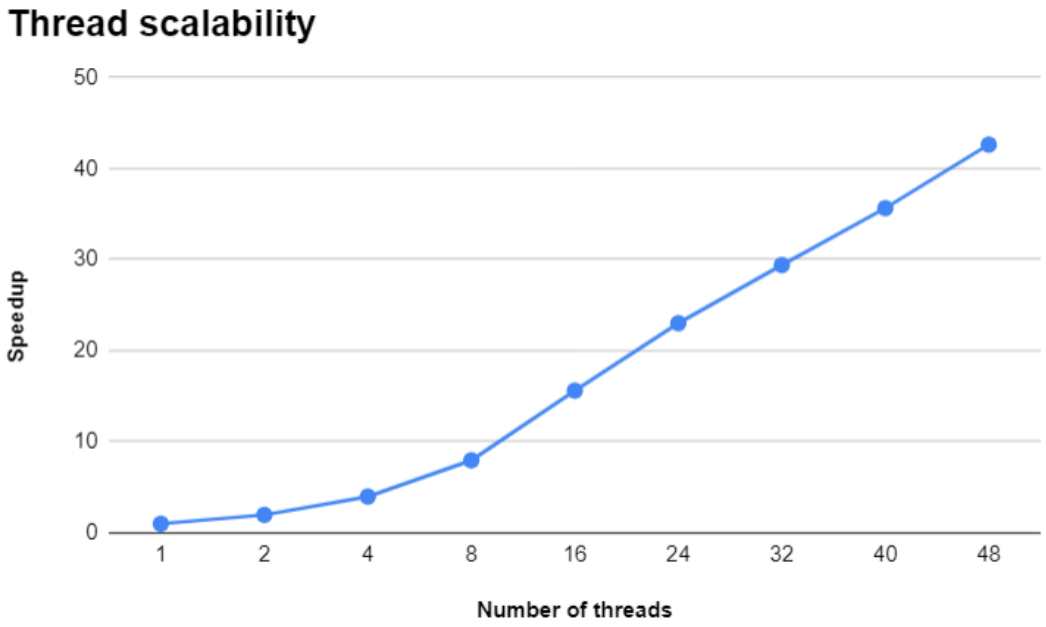


FIGURE 19: AI4EO - THREAD SCALABILITY OF HPDBSCAN

Node scalability using MPI

In order to study the scalability using fully distributed MPI, we spawned a process for each core and increased the number of cores in steps of 48 as each node in the A64FX partition consisted of 48 cores. In order to make the application on multiple nodes, the number of MPI tasks need to be a multiple of the number of cores per node which is 48.

The application scales well up to 8 nodes (8*48 = 384 MPI tasks) after which there is a gradual break-down in scaling which can be attributed to a significant MPI communication overhead with the problem size being constant.

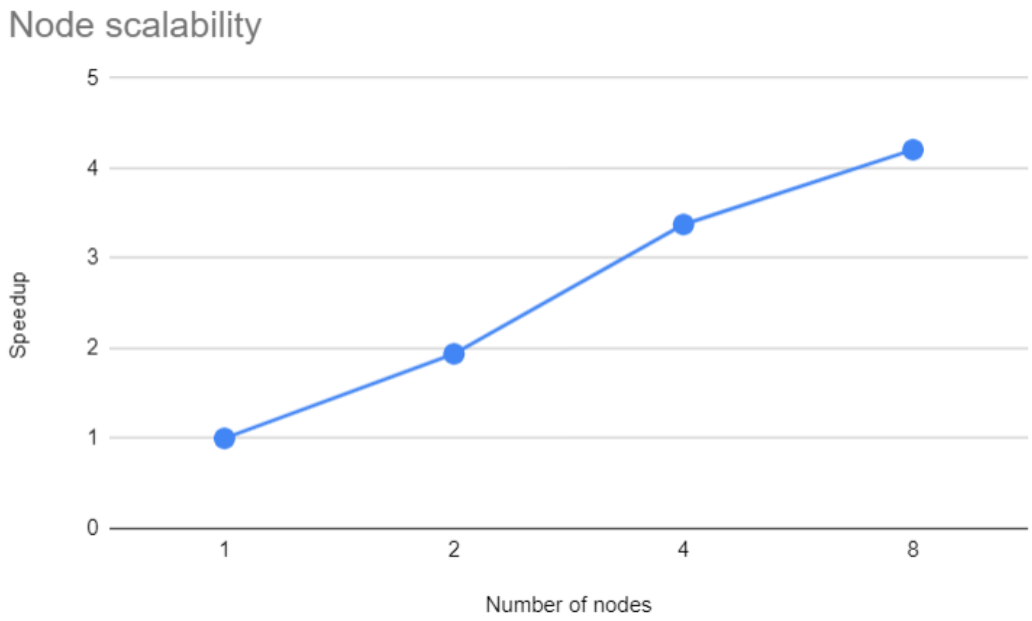


FIGURE 20: AI4EO - NODE SCALABILITY OF HPDBSCAN

Figure 20 illustrates the scalability observed when run on multiple nodes. Note that each node corresponds to 48 cores and the number of threads per MPI task was kept at 1. The reference implementation in a single node run with 48 MPI tasks.

In order to study the breakdown in scalability when the number of nodes were increased beyond 8 (not shown in the figure), we ran HPDBSCAN on a dataset which was twice the size of that used in the previous experiment. As expected, we see that application scales up to 16 nodes before the scalability gradually begins to breakdown. This is illustrated in Figure 21.

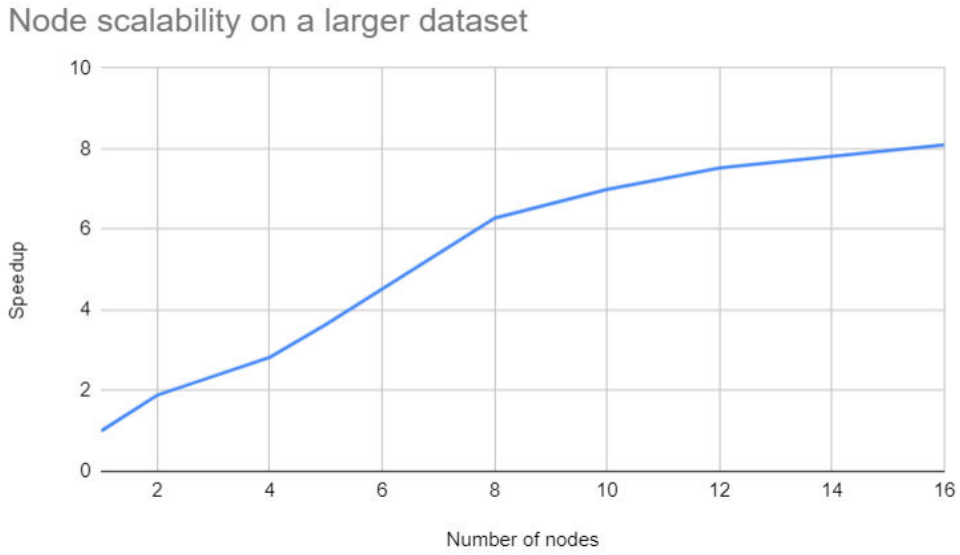


FIGURE 21: AI4EO - NODE SCALABILITY OF HPDBSCAN ON A LARGER DATASET

Next, we study the scalability due to hybrid implementation of HPDBSCAN using both OpenMP and MPI. With the hybrid implementation, the scalability observed was much poorer and for a given task, internal parallelizing it with OpenMP threads did not yield any gains.

Workflow analysis of Classification

Porting of Transformer to JURECA-DC at FZJ

For the DL step we rely on the software stack Stages/2022 and a list of modules that are first loaded via:

```
$ ml Stages/2022 GCC/11.2.0  OpenMPI/4.1.2
$ ml PyTorch/1.11-CUDA-11.5
$ ml matplotlib
$ ml GDAL/3.3.2
```

A virtual environment is also created to enable the installation of further Python packages not included in the pre-installed modules. Differently from IRENE, on JURECA-DC the login node has access to the internet. We used the package installer pip, and we created and sourced via:

```
$ python3 -m venv <env_name>
$ source <env_name>/bin/activate
```

The training of the DL model is performed in parallel thanks to the adoption of PyTorch DDP. Due to an open issue[7] of the distributed runner, a specific configuration is currently used.

PyTorch/1.11 does not detect the correct IB utilised Internet Protocol (IP) addresses. This issue arises from the distributed communication package c10d[8] of PyTorch which connects the communication library with DDP by selecting one of the compute nodes as the host to handle the parallel communication. In the current version of PyTorch 1.11.0, an external Python library sockets[9] included in c10d fails to fetch the IB's IP address of the host compute node. PyTorch then appoints the IP addresses from a slower ethernet connection which is not optimised for node-to-node communication. At the time of writing this document, the PyTorch developers only released a manual fix to this problem which modifies the torchrun source file:

```
import re, sys
from torch.distributed.run import main
from torch.distributed.elastic.agent.server import api as sapi


def new_get_fq_hostname():
    return _orig_get_fq_hostname().replace('.', 'i.', 1)


if __name__ == '__main__':
    _orig_get_fq_hostname = sapi._get_fq_hostname
    sapi._get_fq_hostname = new_get_fq_hostname
    sys.argv[0] = re.sub(r'(-script\.pyw|\.exe)?$', '', sys.argv[0])
    sys.exit(main())
```

[7] https://github.com/pytorch/pytorch/issues/73656

[8] https://pytorch.org/docs/stable/distributed.html

[9] https://docs.python.org/3/library/socket.html

This modified file converts the ethernet to an IB IP address. However, the torchrun command requires the following additional flags with new definitions to detect which compute node is the host:

```
rdzv_conf=is_host=\$(((SLURM_NODEID)) && echo 0 || echo 1)
rdzv_endpoint='$(scontrol show hostnames "$SLURM_JOB_NODELIST" | head \
  -n 1)'i:29500
```

These flags include environment variables available to the Slurm job scheduler. Therefore, further changes are necessary in case a different job scheduler is used.

A common parallelization strategy to train DL models with large datasets is to distribute the input dataset to separate GPUs, where the trainable parameters between the GPUs are exchanged occasionally. This method is called Distributed Data Parallelization (DDP) and greatly reduces the training time. Depending on the data exchange rate between the workers (in this case GPUs), this type of parallelized training can scale to many workers. Currently, the GPUs require the CPUs to access the input data, therefore the transfer of the input data to the workers is, in this case, the bottleneck of such training. Alternatively, only CPUs could be employed to train DLmodels but the GPU architecture enables shorter training durations due to much faster DL-related matrix operations.

In this deliverable, we report the work of porting and analysing the performance of the DDP module (PyTorch package (Li, et al. 2020)) on the Jülich Research on Exascale Cluster Architectures (JURECA) system, (Forschungszentrum Jülich), in particular on the Data Centric (DC) module which includes 192 accelerated compute nodes that have 512 GB of DDR4 RAM and four NVIDIA A100 GPUs with four 40 GB HBM2e.

PyTorch is an open-source ML framework, mainly developed by Facebook AI Research. The PyTorch-DDP module features a built-in way to run distributed training of neural networks on multiple workers. The PyTorch-DDP library uses an AllReduce operation for gradient reduction. Additionally, the gradients are not synchronised individually. Instead, several gradients are collected in ‘buckets’ and are only communicated when the buckets are sufficiently filled. This reduces the total number of communication operations. PyTorch-DDP uses a heuristic for determining the reduction order on a local worker level: The gradients are bucketed in the reverse order of their computation in the forward pass of the network. This is motivated by the fact that the last layers of a network are likely the first to finish computation during the backward pass. To further speed up the training process, PyTorch-DDP offers a no_sync option to only execute an AllReduce operation every few iterations.

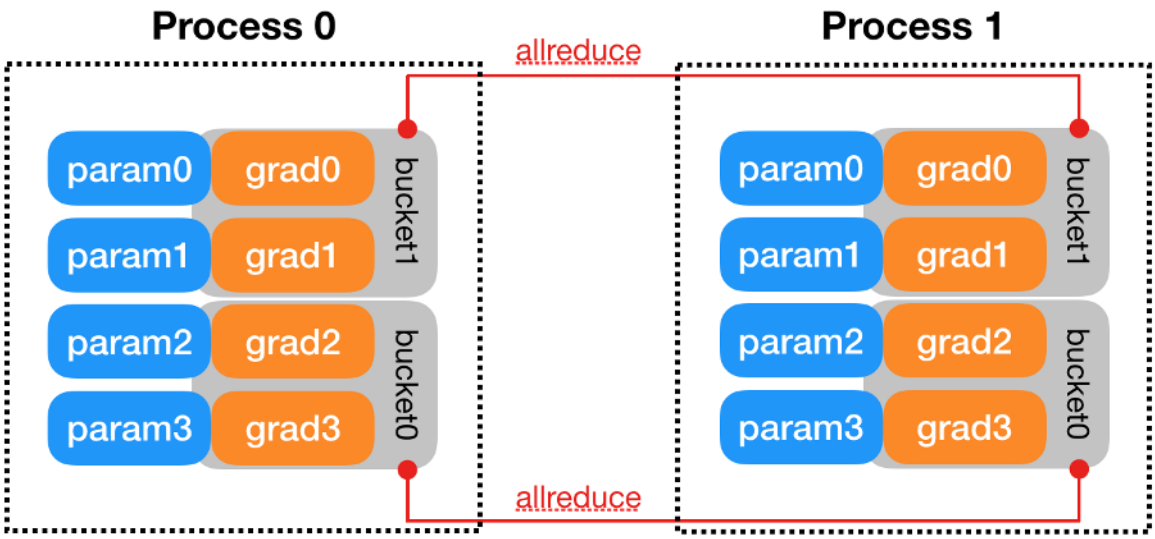


FIGURE 22: ALLREDUCE APPLIED TO THE LOCAL GRADIENTS OF TWO PROCESSES IN ORDER TO COMPUTE THE GLOBAL GRADIENTS[10]

DDP can significantly speed-up the training of the model (as shown in the scalability plot reported below), however it comes with some challenges. As each GPU is set to a fixed mini-batch size, increasing the number of GPUs yields an increase in the total batch size. It has been observed that large batch sizes can hinder the generalisation of the model on unseen data. This limits the scaling performance of the ML training, as the training accuracy becomes the decisive factor when investigating the scaling performance of a trained model. A common strategy is to tune other hyperparameters keeping the accuracy of the training at an acceptable level which has intensively been addressed in the literature (Goyal, et al. 2017), (You, Gitman e Ginsburg 2017). For this purpose, we plan to adopt Ray Tune, which provides a Python API to streamline the hyperparameter tuning of DL models (Liaw, et al. 2018).

Profiling results

The PyTorch profiler was used to measure the time and memory consumption of the model's operators. PyTorch profiler is enabled through the context manager and accepts a number of parameters.

```
with profile(activities=[ProfilerActivity.CPU, ProfilerActivity.CUDA], record_shapes=True,
                        profile_memory=True) as prof:
    with record_function("transformer"):
    #DDP application
```

ProfilerActivity.CPU - Profiles Pytorch operators, TouchScript functions and user defined labels (code segments can be labelled using record_function)

A snapshot of the execution times of the different operators or modules are given in Table 4. Note that the column Self CPU corresponds to the percentage of time spent only in that operator and it excludes the time spent in the child operators whereas the CPU total corresponds to the time spent in the operator including those of the child operators.

[10] https://pytorch.org/docs/stable/notes/ddp.html

TABLE 4: EXECUTION TIMES OF THE DIFFERENT OPERATORS OR MODULES

Name	Self CPU %	CPU total %	CPU time avg
fit_pred_Transformer	17.47%	91.18%	417.622s
enumerate(DataLoader)#_MultiProcessingDataLoaderIter...	50.34%	50.35%	295.643ms
aten::copy_	0.03%	10.70%	5.024ms
Optimizer.step#AdamW.step	2.32%	8.00%	11.104ms
DistributedDataParallel.forward	1.16%	6.60%	8.917ms
aten::to	0.00%	5.38%	7.467ms
aten::_to_copy	0.00%	5.38%	7.464ms
cudaStreamSynchronize	5.32%	5.32%	2.887ms
cudaLaunchKernel	4.88%	4.88%	5.658us
autograd::engine::evaluate_function:torch::autograd...	0.47%	1.89%	39.656us
autograd::engine::evaluate_function:MmBackward0	0.12%	1.58%	104.662us
Optimizer.zero_grad#AdamW.zero_grad	0.25%	0.98%	1.358ms
cudaMemcpyAsync	0.97%	0.97%	19.164us

Similarly, Table 5 illustrates the GPU execution times of some of the operators as reported by the profiler.

TABLE 5: GPU EXECUTION TIMES OF SOME OF THE OPERATORS

Name	Self CUDA	Self CUDA %	CUDA total	CUDA time avg
ncclKernel_AllReduce_RING_LL_Sum_float(ncclDevComm*	49.343s	73.74%	49.343s	14.952ms
fit_pred_LSTM	0.000us	0.00%	10.044s	10.044s
Optimizer.step#AdamW.step	0.000us	0.00%	6.452s	1.955ms
void at::native::vectorized_elementwise_kernel<4	3.251s	4.86%	3.251s	3.454us
aten::add_	2.543s	3.80%	2.543s	3.510us

Distributedata-Parallel.forward	0.000us	0.00%	2.475s	730.055us
aten::mul_	2.353s	3.52%	2.353s	3.572us

Performance metrics

In order to understand the slight deviation from the theoretical scaling for higher number of GPUs, the PyTorch profiler was used to study the GPU execution times on runs using 32, 48 and 96 GPUs each. The percentage of time spent in the module 'ncclKernel_AllReduce_RING_LL_Sum_float' is given in Table 6 for each of the runs.

TABLE 6: PERCENTAGE OF TIME SPENT IN THE MODULE 'NCCLKERNEL_ALLREDUCE_RING_LL_SUM_FLOAT' FOR EACH RUN

Number of GPUs	Average of the time spent in NCCL kernel by a GPU	Standard deviation of the time spent in the NCCL kernel by a GPU.	Average % of time spent in NCCL kernels by a GPU	Standard deviation of the % of time spent in the NCCL kernel by the GPUs.
32	73.46s	6.03s	76.15%	1.73
48	55.69s	5.15s	78.3%	1.63
96	39.24s	4.78s	83.14%	1.76

Higher the number of GPUs, higher is the percentage of time spent in the NCCL kernels which is most likely to offset the performance gains.

Scalability

Here we discuss a scalability plot of the Transformer, scaling the training from 1 node / 4 GPUs to 24 nodes / 96 GPUs (hard threshold of maximum number of nodes that can be allocated on JURECA DC). The Figure 23 shows that as the number of GPUs increases, the actual scalability factor is increasing sub linearly. The decrease in scaling efficiency reflects the values of the metrics retrieved for the NCCL kernels and can be explained by a larger portion of communication vs computation time as the number of GPUs increases.

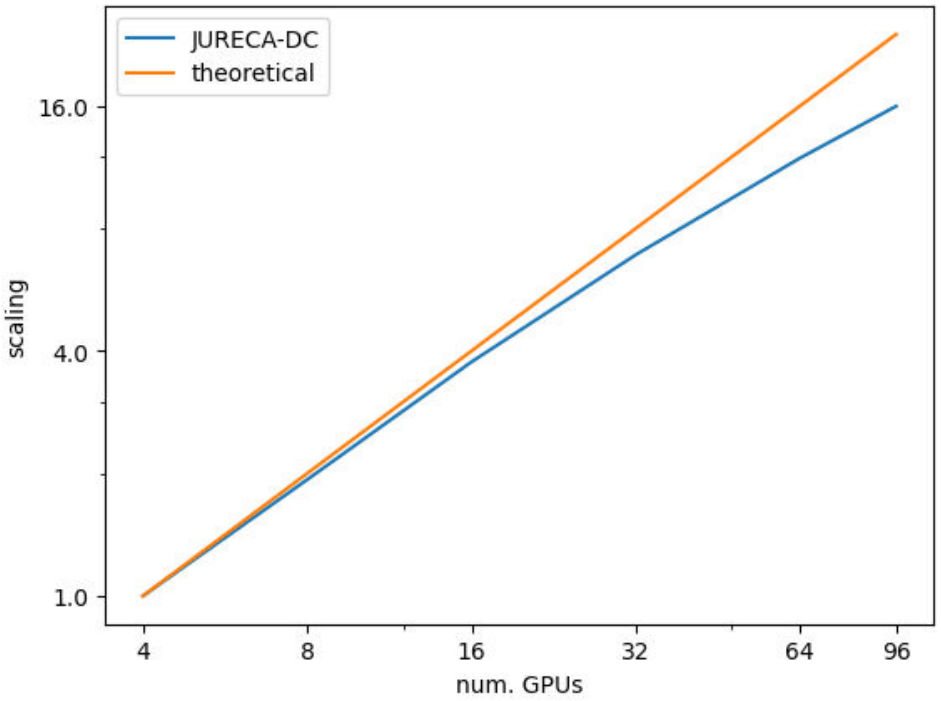


FIGURE 23: SCALABILITY PLOT FROM 4 TO 96 GPUS

Conclusion of Analysis

In this study, we studied the scalability of HPDBSCAN clustering algorithm and analysed the profiling results of the DL step. We observed that, HPDBSCAN scales very well with OpenMP parallelization but the breakdown observed in scaling in the hybrid approach on the MPI only approach on the A64FX partition is to be studied. Regarding the DL step, with the help of tables and a figure we reported several metrics computed on the training of the Transformer. We presented the scaling capabilities of the code and the findings that motivate its current limitations. The profiling results will come at aid to explore and implement strategies that can address such shortcomings.

From Faults to Building – SPECFEM3D & OPENSEES

Application Description

The application promises a systematic and structured combination of several different existing and state-of-the-art seismological and engineering High-Performance Computing (HPC) applications in a novel scalable hybrid-platforms workflow capable of enhancing effectiveness and performance portability. This application aims to create a digital twin for seismic resilience in Central Italy and in particular a seismological/engineering probabilistic simulation workflow resulting from the interaction between realistic ground motion acceleration probabilistic scenarios and the corresponding effects at surface level, specifically on strategic buildings (i.e., from earthquake to buildings).

Computational seismology has historically been one of the most active HPC communities. Current exascale-ready numerical community codes (i.e., SPECFEM3D that we adopted in EUPEX) can provide accurate synthetics seismograms incorporating many of the physical processes occurring during wave propagations (Peter, et al. 2011), (Lee, et al. 2009), (Magnoni, et al. 2014), (Mazzieri, et al. 2013), (Paolucci, Mazzieri e Smerzini 2015). The computational requirements of wave simulations depend on the frequencies that we need to resolve, requiring space-time scales of meters and fractions of seconds. The ultimate goal is to provide seismic hazard analysis insights: for example, CyberShake created a probabilistic seismic hazard map using a physics-based computational approach with full 3D seismic wave simulations (Graves, Seyhan e Stewart 2011). This crucial task requires the contribution of monitoring networks specialists, data analysts, and physics-based model developers, but also the optimization of the workflow (i.e., I/O management, data staging, computational optimizations) of the whole processing to gain an advantage of the forthcoming exascale, modular and heterogeneous computational. resources. The seismological community focuses on this effort as evidenced by national and European initiatives like EU Horizon DARE[11], the Centre of excellence for solid Earth CHEESE[12], eFLOW4HPC[13], Open Data L'Aquila project[14], and DT-GEO[15].

Geotechnics aims to forecast the seismic motion at the civil structure scale depending on shallow deposits' properties and might deal with permanent deformations (e.g., soil liquefaction). Real applications might involve complex subsoil geometries resulting in challenging numerical analyses that may require high computation capability. Structural engineers are in charge of analysing structures dependent on the construction materials, functional specifications, and the related performance level. The latter is associated with accepted damage levels activated by seismic actions. In EUPEX, we plan simulate the ground motion scenarios with the SPECFEM3D codes and their impact on the buildings using the OPENSEES framework through time-histories associated with the ground motion probabilistic scenarios.

Application Use Case

The Workflow is described in Figure 24 and it is composed of 4 steps: A) creation of a geological databases of seismic sources in Central Italy and relative uncertainties (SPECFEM3D), B) simulations and storing of seismic wavefield simulations, C) creation of probabilistic ensemble of ground motion parameters, D) structural and geotechnical engineering simulations (soil-building interaction) and creation of the fragility curve (OPENSEES)

[11] http://project-dare.eu

[12] https://cheese-coe.eu

[13] https://eflows4hpc.eu/

[14] https://www.opendatalaquila.it/

[15] https://dtgeo.eu/

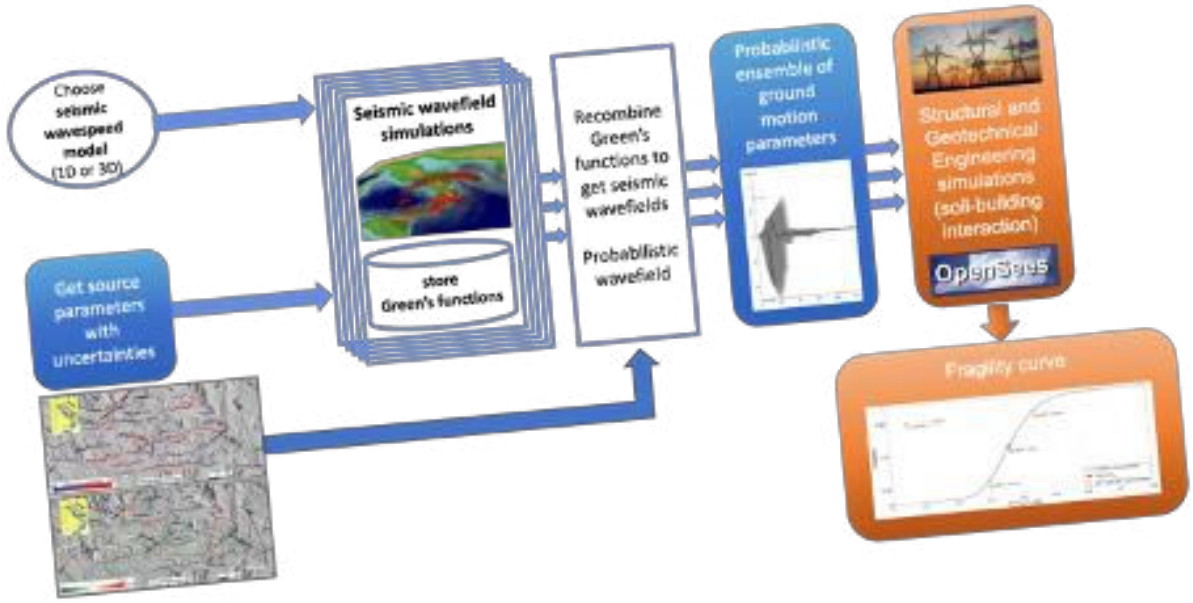


FIGURE 24: WORKFLOW DESCRIPTION OF FROM FAULTS TO BUILDING APPLICATION

SPECFEM3D code[16] (Peter, et al. 2011) solves linear seismic wave propagation on heterogeneous 3D models. It is based on a high-order spectral-element discretization for unstructured hexahedral meshes. Required input data includes topography, 3D wave speed, density, and attenuation fields. Scalable performance is exascale-ready and the code runs on the largest supercomputers worldwide, including NVIDIA A100 GPU clusters. The solver has been actively used in the seismological community over many years as it supports MPI, OpenMP, CUDA and HIP GPU acceleration. SPECFEM3D is open-source software and licensed under the GNU GPLv3. It is one of the seismological codes selected as flagship code in the EU Horizon ChEESE center of excellence in Solid Earth for pre-exascale and exascale infrastructures.

Physical-based simulations will not exceed 2 Hz of frequency resolution both for lacking of geological information and for the large computational power request. Such frequencies are too low for direct engineering applications. To overcome this problem, we will apply a hybrid approach that involves the combined use of physics-based simulations for low frequencies and stochastic and / or artificial intelligence methods for high frequencies. In EUPEX, we will use an open-source software written in FORTRAN, called EXSIM (Extended SIMulation, Motazedian & Atkinson 2005), capable of simulating ground motion time series at high frequencies. To consider the complex geometry of the fault and the non-homogeneous rupture mechanism, the fault plane is divided into N sub-elements, each of which basically acts as a point source. Many others geological details (e.g site amplification, geometrical spreading, path duration and quality factor) can be included to improve the reliability of the results and the matching with the deterministic portion of the spectrum.

The combination of stochastic approach and statistical construction of a probabilistic ground motion scenario will provide the possibility to study the Figure 25[OBJ]).

[16] www.specfem.org

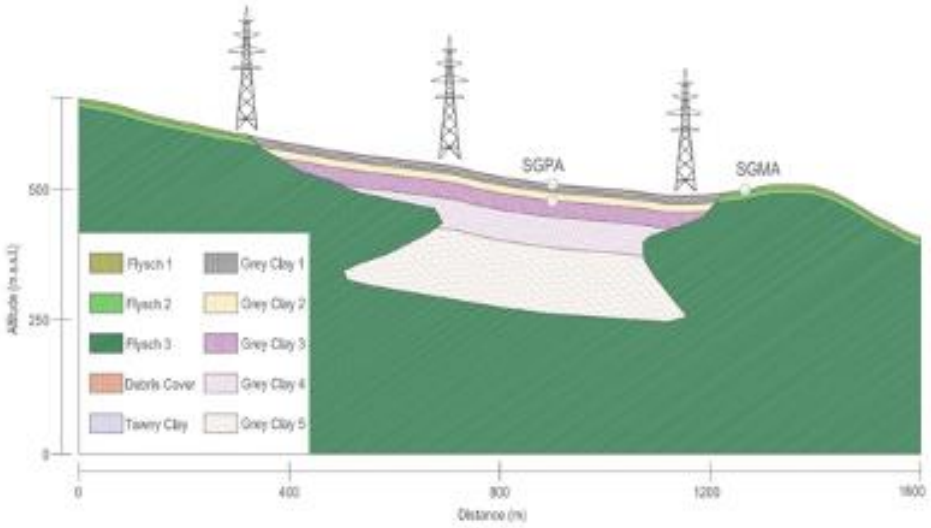


FIGURE 25: EXAMPLE OF GEOTECHNICAL DOMAIN

We adopt OpenSEES, an open-source Tcl/C++ framework for sequential and parallel finite element applications, to simulate the response of structural and geotechnical systems under seismic loadings. The ensemble motion obtained through SPECFEM3D+hybridization will be applied at the base of the soil model. Nonlinear behaviour of soils will be modelled through advanced constitutive model and an integrated soil-structure model will be developed in OpenSEES properly included the soil-structure interaction. Probabilistic analyses will be carried out in order to provide fragility curves (i.e., the probability of reaching or exceeding a specific damage state under a given earthquake excitation parameter) associated with selected limit state (a condition of a civil construction beyond which it no longer fulfils the relevant design criteria) at structural and geotechnical level.

One of the critical challenges of complex software integration and parallel coordination is mapping the resulting highly parallel program onto the target system, supporting scalability and overall execution speed and reliability. We plan to adopt a modular approach to ease the design and development of efficient, scalable, and reliable pipelines of highly-specialized parallel components. This approach this approach will be realized by leveraging parallel coordination patterns offered by the Streamflow Workflow Manager System and the scalable "Cross-Application Programmable I/O" (CAPIO) layer. Streamflow+CAPIO makes it possible to handle large datasets, increase the exploitation of the available parallelism at different levels (i.e., from hardware accelerators to Cloud-based clusters), and reuse existing software components to address complex problems with reduced programming effort.

Workflow analysis

We expand the analysis of the workflow steps described in the previous paragraph and in Figure 24:

A. Creation of a geological databases of seismic sources in Central Italy and relative uncertainties. Starting from the available geological dataset (i.e. DISS) will be create a dataset of the seismic sources in Central Italy: mainly 300 finite fault discretized in patches of 2x2km for a total of 9000 files of point-like seismic sources (N). Another initial step is the physical domain's discretization (hexahedral mesh). It would be one of the most challenging and underrated tasks in seismic simulations. More than 80% of human time needed for simulating seismic events is spent creating a reliable geometric shape of collected geological information and applying meshing algorithms to obtain unstructured grids. In this project, we take advantage of the significant amount of high-quality data currently available for the selected regions to create hexahedral meshes. Building a new geological model is beyond the resource available in this project, and the geological structure does not change significantly in times comparable with human life. Nevertheless, rock and soil properties do (e.g., for fluid migration). Within this framework, the grids will

be decoupled by the material definition. While the meshes will be kept constant to reduce computational domain discretization effort, the material properties will be improved thanks to new findings or data. The mesh is composed of 6 ASCII files for total size of approximately 1GB. The model is one ASCII files of 1.5 GB.

B. Simulations and storing of seismic wavefield. For each seismic source a SPECFEM3D simulation will be performed and the output will be a number of ASCII files each one corresponding to wavefield time snapshot. The number of files is variable depending on the simulation time step but the total size will be approximately 200 MB for each source. For the specific geotechnical region, a large number of seismograms describing the ground motion will be saved at the bedrock for S station. The adoption of Adaptable Seismic Data Format (ASDF) to store the seismogram output, significantly reduces the number of files to O(N*S). the size of the singular station could reach around 10*N MB.
C. Creation of probabilistic ensemble of ground motion parameters. e seismograms and the wavefield snapshots will be combined in an ensemble ground motion scenario. Each point of the wavefield will be describe by statistical distribution of the peak motion (And other engineering parameters) while the ensemble seismograms will be stored as peak envelop, average seismogram, standard deviation.
D. Structural and geotechnical engineering simulations. The seismograms will be feed into OPENSEES together with domain discretization, material properties and constitutive law to obtain the fragility curve for each building desired.

Workflow analysis of SPECFEM3D

At this stage of EUPEX we analysed the performance of the solver SPECFEM3D and in particular after the porting on IRENE@TGCC. In particular, the application analysis was focused to port SPECFEM3D miniapp (by CHEESE CoE) on A64FX and testing the performance with Scalable Vector Extension SVE (or without, NOSVE) and CHEESE optimizations (OPT/NOOPT).

Furthermore, we compare the results with Intel x86_64 machine (CINECA Galileo100).

The profiling has been performed using Likwid[17]. The metric are: total MFLOPS, IPC (instructions per cycle), VECT. RATIO (vectorization ratio) and runtime.

The primary goal is to test the impact of SVE

Figure 26 shows the importance of the vector optimization in ChEESE and the consequently good performance of SVE. The general performances are still far from the ones obtained on x86_64 infrastructure.

[17] https://github.com/RRZE-HPC/likwid

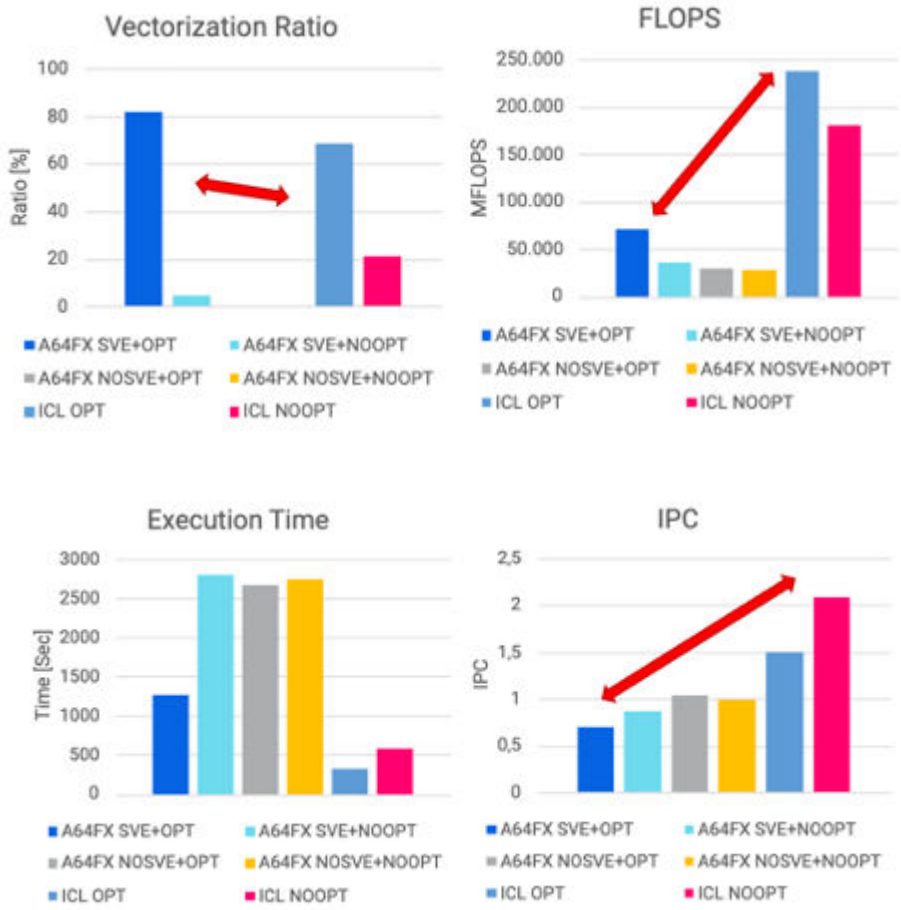


FIGURE 26: HISTOGRAM OF THE METRIC FOR SPCFEM3D SIMULATION ON IRENE (A64FX) AND GALILEO100 (ICL) WITH OR WITHOUT SVE AND CHEESE OPTIMIZATIONS.

Conclusion of the Analysis

SPECFEM3D is by far the largest load of the workflow and it has been porting on IRENE (A64FX) without particular difficulties in term of compilation and execution.

We ported both the regular version and the version optimized inside ChEESE COE in order to study the importance of SVE and HBM. We compared the results with Galileo100 (x86_64 machine). we used ARM compiler on IRENE and OneAPi on Galileo100.

The impact of HBM and SVE on A64FX is promisingly relevant but only if the code is restructured. The performances are still slightly lower comparing on production machine based on x86.

ESPRESO FEM

Application description

ESPRESO (numbox.it4i.cz 2022) is a massively parallel framework based on the Finite Element Method for Engineering Application. The main goal of the ESPRESO framework is to create a robust and highly parallel open-source package applicable for a wide range of complex engineering simulations in areas such as mechanical engineering, civil engineering, bio-mechanics, and energy industry.

The package contains several logical units, which can be combined to reflect requirements for a given problem solution. These units include entry data pre-processing, computational mesh processing, numerical model construction, specific physical problem solution using a massively parallel sparse linear solver in combination with available nonlinear solvers, and final preparation of output data for visualization and online monitoring of achieved results.

From the very beginning all modules within the package are developed as a parallel code based on massively scalable numerical methods and algorithms. Thereby we can ensure optimal utilization of state-of-the-art supercomputers. The added value of the package is our implementation of a FETI (Finite Element Tearing and Interconnecting) domain decomposition-based massively parallel linear solver, which can solve problems of hundreds of billions of unknowns.

Unstructured mesh processing

ESPRESO input/output modules are designed to perform operations with unstructured mesh databases that are commonly used in the finite element method. It includes highly parallel loading and storing from / to a user favourite format and performing high-quality domain decomposition. The modules are conceptually developed as a massively parallel tools designated for the HPC environment including high-performance storage systems.

The input module provides direct usage of tools for the creation of complex engineering models as ANSYS, HyperMesh, ANSA, and ABAQUS with highly parallel solvers. This direct interconnection allows usage of robust commercial pre-processing tools for the preparation of complex sets in cases where excessive demands on computational mesh quality are required. With this module, users can use a database file saved in his/her favourite format and use it for multiple calculations. Every calculation can be performed using a different number of compute nodes without further restrictions as our approach does not require running on a fixed number of MPI processors and domains as in the case of a parallel binary file. This solution allows different amounts of computational resources to be used based on the needs or availability of computational resources.

Physical modules

One of the main parts of our product is a Finite Element Method (FEM) module for description of physical processes. Development of this module follows from IT4Innovations collaboration with industrial partners as well as work on national and international projects. This module is designed to solve complex engineering problems while covering all physical aspects such as non-linear material models, convection, ration, etc. to system of linear equations. Currently, ESPRESO FEM supports non-linear and time-dependent solution of heat transfer, structural mechanics, and acoustics physics.

Massive parallel FETI solver

The module with massively parallel sparse solver contains several highly scalable solvers based on the FETI (Finite Element Tearing and Interconnecting) methods which allows the computational capacity of the state-of-the-art supercomputers to be fully utilized and thus solve problems with billions of unknowns. FETI is a domain decomposition method in which a whole problem is decomposed into smaller non-overlapped parts, subdomains, that are glued together by Lagrange multipliers (reaction forces between subdomains). Then, the solution is computed by a combination of iterative and direct solvers - subdomains solutions are computed by the direct sparse solver and optimal Lagrange multipliers are computed by the iterative solver.

Solving Engineering Problems by FETI solver

The ESPRESO FEM library provides a full workflow to solve engineering problems starting from loading an input description from an external database to storing the results for visualization. An example of a possible input and output can be seen in Figure 27, where a drill bit is described by an unstructured mesh. Our library loads this description together with a configuration file that describes boundary conditions, initial conditions, material properties, etc. Then, the library decomposes whole mesh into domains and builds a system of linear equations that is solver by the internal FETI solver. Then the solution is stored to an external database that can be visualized by common post-processing tools. The workflow of the library is described in more details in the sections below.

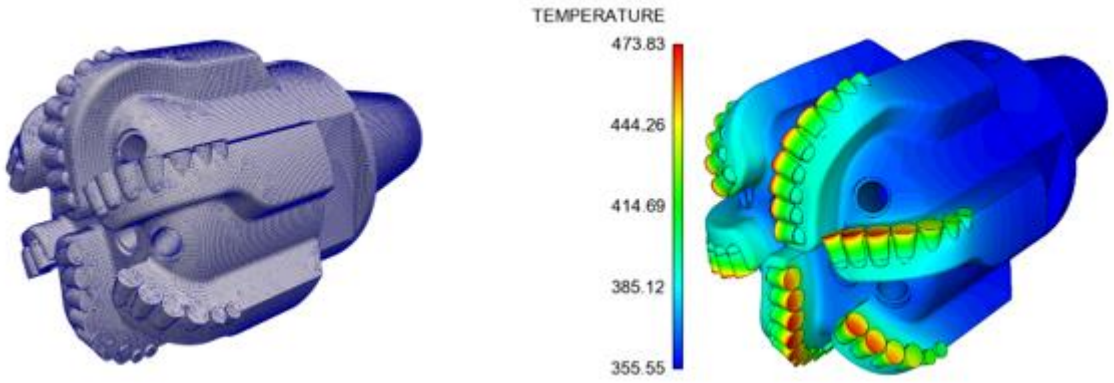


FIGURE 27: EXAMPLE OF A POSSIBLE INPUT AND OUTPUT OF ESPRESO FEM

Scalable mesh processing

The first step of the library is to load a description of the solved problem. This step is performed by our algorithm for highly parallel loading and processing of unstructured mesh databases in a distributed memory environment of large HPC clusters without collecting data into a single process (Meca Ondřej 2022). The algorithm accepts arbitrarily scattered data across all processes and produce a parallel mesh description with elements sorted according to Hilbert's space filling curve (SFC) (Sagan 1994). This ordering assures that the mesh is well load-balanced, and the number of neighbouring processes is sufficiently small for good scalability. Hence, we can efficiently compute the dual graph and pass it to ParMETIS to get the final decomposition. The reason why we use ParMETIS is its decomposition quality. The quality of the spatial clusterization produced by the SFC does not assure the quality needed by our parallel FETI solvers. In particular, improper domain shapes in the case of complex geometries negatively affect the numerical scalability of the solver. This can lead to a longer solver runtime.

The first step of the algorithm is to read and understand data in an input database. Most of a database consists of raw mesh data (coordinates, elements connectivity, etc.) that are determined by keywords in order to allow parsing. There keywords must be found in a database and synchronized by the all-gather operation. At the beginning of the algorithm, each process reads a piece of a database and searches it for keywords. Based on the position and parameters of all keywords, each process understands the meaning of its local data and parses them.

After the parallel parsing, we have arbitrarily distributed data across all processes. To process these data, the parallel sorting of nodes and elements according to ids is applied. The sorting moves all nodes and elements to a known location (process) where they can be directly addressed. Then, we apply second parallel sort according to SFC to move spatially close elements to the same processes. After this, processes hold uniformly distributed and spatially close nodes and elements.

The result of the previous steps is a mesh decomposed by Hilbert's SFC. Then, a loaded mesh is processed using standard methods. In our case, the mesh processing contains computation of the dual graph, re-indexing of elements nodes from ids to local offsets, and synchronization of regions data across processes (computing global offsets, the number of elements within regions, etc.), and calling ParMETIS and METIS.

FETI solver

For more than two decades, the FETI method (for more detail see, e.g., in (Farhat C. 1994), (Gosselet 2006) has been successfully used in the engineering community, primarily for very large problems arising from the discretization of partial differential equations (PDE). The classical FETI algorithm is based on eliminating the primal unknowns so that the resulting linear system, in terms of the Lagrange Multipliers (LM), can be solved iteratively with the projected conjugate gradient method. Over the years, numerous variants of the basic FETI concept have been developed. In ESPRESO FEM, we have implemented the algorithm based on T(otal)FETI method (Dostal Zdenek 2006). This algorithm is briefly sketched to better understand steps in analysis.

Let us consider a partitioning of the global domain into s subdomains. To each subdomain corresponds stiffness matrix $\boldsymbol{K}^s$, the nodal load vector $\boldsymbol{f}^s$, matrix $\boldsymbol{R}^s$ that is kernel of $\boldsymbol{K}^s$, and a signed boolean matrix $\boldsymbol{B}^s$ defining connectivity of the subdomain s with neighbour subdomains. Using proposed notation, the discretized equilibrium equation for s-th subdomain is $\boldsymbol{K}^s\boldsymbol{u}^s = \boldsymbol{f}^s - (\boldsymbol{B}^s)^T\boldsymbol{\lambda}$. Collecting $\boldsymbol{K} = diag(\boldsymbol{K}^1, \dots, \boldsymbol{K}^s)$, $\boldsymbol{R} = diag(\boldsymbol{R}^1, \dots, \boldsymbol{R}^s), \boldsymbol{B} = [\boldsymbol{B}^1, \dots, \boldsymbol{B}^s], \boldsymbol{u} = [\boldsymbol{u}^1, \dots, \boldsymbol{u}^s], \boldsymbol{f} = [(\boldsymbol{f}^1)^T, \dots, (\boldsymbol{f}^s)^T]^T$ together with $\boldsymbol{Bu} = \boldsymbol{o}$ the equation can rewritten as

$$\boldsymbol{Ax} = \begin{pmatrix} \boldsymbol{K} & \boldsymbol{B}^T \\ \boldsymbol{B} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{u} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \boldsymbol{f} \\ \boldsymbol{o} \end{pmatrix}.$$

This system can be rewritten to (please see e.g., (Farhat C. 1994))

$$\begin{pmatrix} \boldsymbol{F} & \boldsymbol{G}^T \\ \boldsymbol{G} & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\alpha} \end{pmatrix} = \begin{pmatrix} \boldsymbol{d} \\ \boldsymbol{e} \end{pmatrix},$$

where $\boldsymbol{F} = \boldsymbol{B}\boldsymbol{K}^+\boldsymbol{B}^T, \boldsymbol{G} = -\boldsymbol{R}^T\boldsymbol{B}^T, \boldsymbol{d} = \boldsymbol{B}\boldsymbol{K}^+\boldsymbol{f}, \boldsymbol{e} = -\boldsymbol{R}^T\boldsymbol{f}$, and $\boldsymbol{\alpha}$ are amplitudes of rigid body motions. Vector $\boldsymbol{\lambda}$ in $\boldsymbol{G\lambda} = \boldsymbol{e}$ can be split into $\bar{\boldsymbol{\lambda}} + \tilde{\boldsymbol{\lambda}}$, where $\tilde{\boldsymbol{\lambda}}$ satisfies $\boldsymbol{G}\tilde{\boldsymbol{\lambda}} = \boldsymbol{e}$ and $\boldsymbol{G}\bar{\boldsymbol{\lambda}} = \boldsymbol{o}$, where $\boldsymbol{G}\bar{\boldsymbol{\lambda}} = \boldsymbol{o}$ can be enforced by the orthogonal projector $\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{Q}$ onto the null space of $\boldsymbol{G}$, where $\boldsymbol{Q} = \boldsymbol{G}^T(\boldsymbol{G}\boldsymbol{G}^T)^{-1}\boldsymbol{G}$ is the orthogonal projector. This projector is called coarse problem and its assembling is the most time-consuming part during FETI solver pre-processing. Then equations can be solved with an arbitrary iterative linear system solver e.g., by projected conjugate gradient (PCG) that works as follows:

Set $\bar{\boldsymbol{\lambda}} = \boldsymbol{P}\overline{\boldsymbol{\lambda}_0}, \overline{\boldsymbol{\lambda}_0} \in \mathbb{R}^n, \varepsilon > 0, i_{max} > 0$

$\boldsymbol{g} = \boldsymbol{F}\bar{\boldsymbol{\lambda}} - \tilde{\boldsymbol{d}}, \boldsymbol{w} = \boldsymbol{P}\boldsymbol{g}$

for $i = 0,1, \dots, i_{max}$ **do**

$\boldsymbol{v} = \boldsymbol{F}\boldsymbol{p}$ *(apply F in the measurements)*

$\boldsymbol{w}_{prev} = \boldsymbol{w}, \boldsymbol{w} = \boldsymbol{P}\boldsymbol{g}$ *(apply P in the measurements)*

$\rho = -(\boldsymbol{q}, \boldsymbol{w})/(\boldsymbol{p}, \boldsymbol{v}), \bar{\boldsymbol{\lambda}} = \bar{\boldsymbol{\lambda}} + \boldsymbol{p}\rho, \ \boldsymbol{g}_{prev} = \boldsymbol{g}, \boldsymbol{g} = \boldsymbol{g} + \boldsymbol{v}\rho$

if $\sqrt{(\boldsymbol{g}, \boldsymbol{P}\boldsymbol{g})} < \varepsilon$ **then break**

$$\beta = \frac{(\boldsymbol{g}, \boldsymbol{w})}{(\boldsymbol{g}_{prev}, \boldsymbol{w}_{prev})}$$

$\boldsymbol{p} = \boldsymbol{w} + \boldsymbol{p}\beta$

$\boldsymbol{\lambda} = \tilde{\boldsymbol{\lambda}} + \bar{\boldsymbol{\lambda}}$

Note that apply dual operator $\boldsymbol{F}$ and projector $\boldsymbol{P}$ are the most expensive operations as the rest of operations manipulates with vectors only. Their performance determines the performance the FETI solver.

Performance of ESPRESO FEM on A64FX

This section presents the performance of the ESPRESO FEM library on ARM A64FX partition on the IRENE cluster. This partition consists of 80 nodes (A64FX Armv8.2-A SVE @1.8Ghz, 48 cores, 32GB HBM) interconnected with a 100Gb/s network. The cluster contains Lustre storage with a theoretical throughput of 300 GB/s. The theoretical peak performance of a single node is 2.8 TFLOP/s.

We compare absolute performance with CPU partition on the Karolina cluster at IT4Innovations National Supercomputing Center. Karolina CPU partition consists of 720 compute nodes (2 x AMD EPYC 7H12, 64-cores, 2.6 GHz; 256 GB DDR4) interconnected by InfiniBand HDR 200 Gbps network. The theoretical peak performance of a single socket is 2.66 TFLOP/s that is similar to IRENE.

The performance of GPU acceleration was measured on Karolina GPU partition consists of 72 compute nodes (2 x AMD EPYC 7763, 64-cores, 2.45 GHz, 1024 GB) with 8 x NVIDIA A100 (40 GB HBM2).

ESPRESO FEM library on IRENE

ESPRESO FEM is entirely written in C/C++ (standard c++11). The parallelization is based on a combination of MPI and OpenMP for CPU-only machines. The common usage is 1 MPI process per NUMA domain and OpenMP threads to utilize all cores. The most time-demanding parts can be accelerated by GPU.

Originally, the library was written and optimized for x86 architectures with the utilization of third-party libraries for basic mathematical operations provided by Intel-MKL. Since Intel-MKL is not available on A64FX, the solver was rewritten for SuiteSparse which is an open-source package already installed on the IRENE cluster. It provides basic operations with sparse matrices that play a key role w.r.t. the performance of the library. Besides the SuiteSparse library, we also utilized BLAS for operations with dense matrices and METIS and ParMETIS during the mesh pre-processing phase.

On IRENE, the library was compiled with the Fujitsu compiler. The list of all used modules is the following:

- **fujitsu, mpi** - a compilation of core modules, due to '-SSL2' parameter also BLAS routines,
- **metis/5.1.0, parmetis/4.0.3** - high-level domain decomposition required by our FETI solver,
- **suitesparse/5.8.1** - sparse routines required by our FETI solver.

The following flags were used for compilations:

```
-march=armv8.2-a+sve -Kfast -KA64FX -KSVE -Kopenmp -SSL2 -std=c++11 -O3 -g
```

Performance was tested with 2 benchmarks. The first benchmark was manifold (see right image in Figure 28). It was used to test the input module. The model was stored in an Ansys CDB database file with a size 1.9 GB. The database contains 9,211,936 nodes and 6,105,544 quadratic tetrahedrons (tetrahedrons with mid-points). The second benchmark was cube (see left image in Figure 28). It is an artificial example that can be generated by the ESPRESO FEM according to a user's needs (also in 2D as a square). It was used to test the solver since it is easy to generate a wide set of examples to thoroughly test solver behaviour.

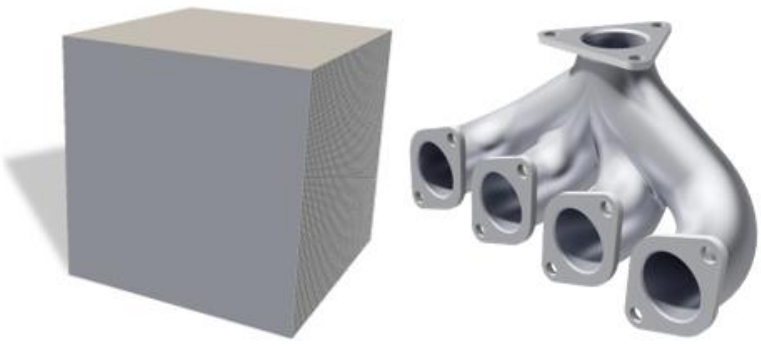


FIGURE 28: EXAMPLES USED FOR TESTING ESPRESSO FEM LIBRARY ON IRENE

Scalable mesh processing

This section describes the behaviour of loading an input database with an unstructured mesh description. As this step is the pre-processing phase, the main goal is to prepare the mesh for the solver as fast as possible with given environment settings (usually, the number of threads and MPI processes is set according to the solver). Hence, we measure the strong scalability of the manifold benchmark with an increasing number of MPI processes. The results are in Table 7.

TABLE 7: STRONG SCALABILITY OF ESPRESO FEM LOADER ON IRENE

# MPI	load [s]	parse [s]	builder [s]	mesh pre-processing [s]
6	2.26	5.73	25.29	66.37
12	1.53	3.44	13.83	34.53
24	1.71	1.51	7.98	18.40
48	2.00	0.82	4.62	12.72
96	1.21	0.43	3.25	6.97
192	0.80	0.25	2.21	3.11

We divided this step into 4 intervals: (a) loading the whole mesh database into the main memory, (b) parallel parsing of input database, (c) mesh builder, and (4) mesh pre-processing. The performance of the loading is determined by the throughput to the storage (Lustre). Achieved throughput is depicted in Figure 29.

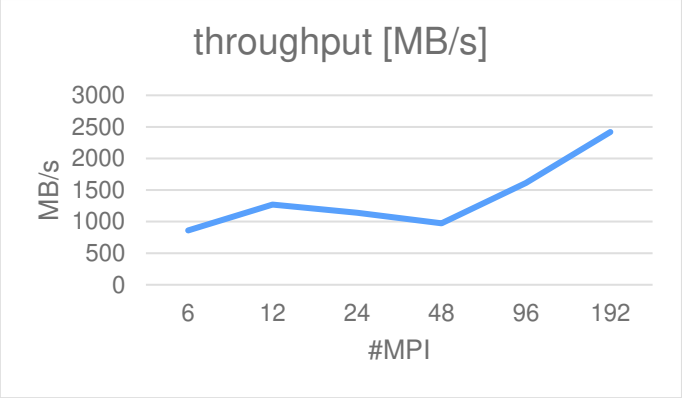


FIGURE 29: THROUGHPUT OF ESPRESO FEM LOADER ON IRENE

The behaviour of the rest intervals is depicted in Figure 30. The performance of parallel parsing is dominated by converting numbers from ASCII to binary format. Mesh builder contains the all-to-all communication pattern and re-indexation of nodes and elements to local indices (offsets into local arrays). During mesh pre-processing ParMETIS and METIS libraries are called to compute domain decomposition used by our FETI solver. The communication pattern in this interval should be nearest-neighbour due to SFC decomposition computed by the mesh builder. The expected overall behaviour is almost linear speedup up to the point where the overhead of the all-to-all communication pattern (in mesh builder) starts to dominate the run-time. Linear scaling is confirmed by the graph in Figure 30.

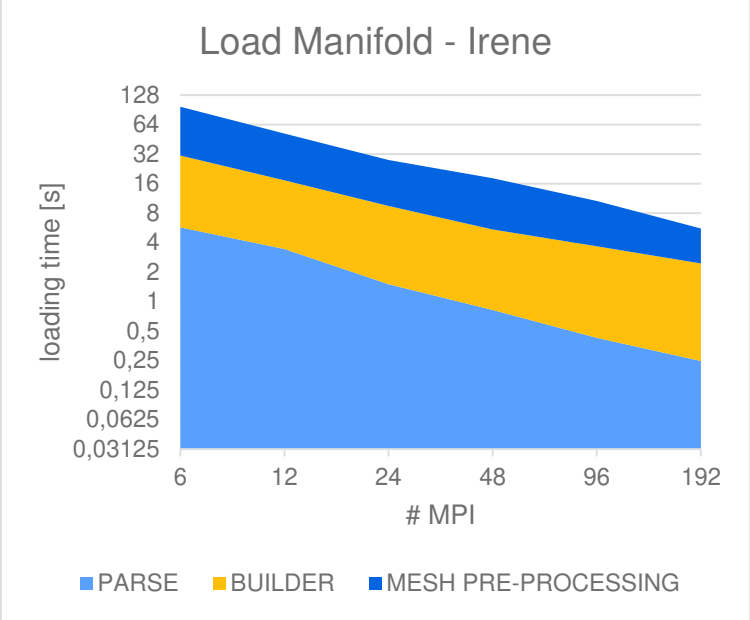


FIGURE 30: STRONG SCALING OF ESPRESO FEM LOADER ON IRENE

Despite expected scalability, absolute performance is about 5x slower compared to the Karoline cluster as can be seen in Table 8 if we compare chip to chip performance.

TABLE 8: COMPARISON OF ESPRESO FEM LOADER ON KAROLINA VS. IRENE

CPUs	Karolina [s]	IRENE [s]	ratio [AMD EPYC/A64FX]
1/8	18.58	97.39	0.19
1/4	9.12	51.80	0.18
1/2	4.73	27.89	0.17
1	3.12	18.17	0.17
2	1.82	10.65	0.17
4	1.35	5.56	0.24

FETI solver

This section describes the performance of our FETI solver. It is the key module that performs the required simulation. In the case of non-linear and time-dependent simulations, the solver is called more times. Hence, it is usually the most time-demanding part of the ESPRESO FEM library. Its performance is dependent on the performance of third-party libraries that provide key functionalities: solving a sparse system of linear equations and applying a dense matrix to a vector. Dense operations are provided by BLAS. For sparse operations, the SuiteSparse module was used.

TABLE 9: PERFORMANCE OF FETI SOLVER ON IRENE

DIM	MPI	subdo-mains	K^s size	Analysis	Build	symbolic factoriza-tion	assembler	update	solve	itera-tions
2D	4	9216	961	1.07	11.45	0.49	0.26	5.63	2.03	48
2D	**4**	**4608**	**1891**	**0.97**	**11.85**	**0.40**	**0.26**	**1.18**	**2.66**	**76**

2D	4	2304	3721	0.92	12.17	0.40	0.26	0.68	4.80	67
2D	4	1152	7381	0.90	12.58	0.39	0.26	0.65	7.26	105
2D	4	576	14641	0.90	13.01	0.38	0.26	0.69	6.32	92
2D	4	288	29161	0.90	13.50	0.37	0.26	0.78	9.62	141
2D	4	144	58081	0.93	13.91	0.37	0.26	0.85	8.36	122
2D	8	18432	961	1.09	11.45	0.53	0.27	16.61	4.00	79
2D	**8**	**9216**	**1891**	**0.96**	**11.84**	**0.42**	**0.26**	**3.10**	**2.62**	**73**
2D	8	4608	3721	0.92	12.17	0.42	0.26	0.90	7.91	109
2D	8	2304	7381	0.90	12.58	0.40	0.26	0.68	7.25	102
2D	8	1152	14641	0.90	13.00	0.38	0.26	0.70	10.42	150
2D	8	576	29161	0.89	13.50	0.38	0.26	0.78	9.61	137
2D	8	288	58081	0.93	13.91	0.37	0.26	0.86	13.85	198
2D	16	36864	961	out of memory						
2D	16	18432	1891	0.97	11.85	0.45	0.26	8.70	3.03	75
2D	**16**	**9216**	**3721**	**0.92**	**12.17**	**0.43**	**0.26**	**1.51**	**5.02**	**67**
2D	16	4608	7381	0.91	12.60	0.87	0.27	0.85	7.44	106
2D	16	2304	14641	0.89	13.01	0.38	0.26	0.73	6.55	93
2D	16	1152	29161	0.89	13.50	0.39	0.26	0.78	10.19	147
2D	16	576	58081	0.92	13.91	0.37	0.26	0.86	8.74	126
2D	32	73728	961	out of memory						
2D	32	36864	1891	0.97	11.85	0.53	0.26	29.02	3.54	73
2D	32	18432	3721	0.92	12.17	0.48	0.26	4.63	8.23	109
2D	**32**	**9216**	**7381**	**0.90**	**12.59**	**0.41**	**0.26**	**1.07**	**7.24**	**101**
2D	32	4608	14641	0.89	13.01	0.39	0.26	0.78	10.56	152
2D	32	2304	29161	0.89	13.50	0.38	0.26	0.80	9.75	138
2D	32	1152	58081	1.03	13.92	0.39	0.26	0.86	14.49	208
3D	4	4096	729	0.58	10.10	0.50	0.25	5.68	1.54	30
3D	4	2048	1377	0.54	10.46	0.40	0.25	1.10	1.68	39
3D	**4**	**1024**	**2601**	**0.52**	**10.81**	**0.35**	**0.25**	**0.68**	**1.49**	**37**
3D	4	512	4913	0.52	11.13	0.32	0.25	0.99	1.28	33
3D	8	8192	729	0.58	10.10	0.53	0.25	12.21	1.95	37
3D	8	4096	1377	0.54	10.46	0.42	0.25	3.07	1.46	33
3D	**8**	**2048**	**2601**	**0.53**	**10.80**	**0.36**	**0.25**	**0.86**	**1.88**	**47**
3D	8	1024	4913	0.53	11.13	0.33	0.25	1.01	1.59	41

The performance of the solver was tested with the cube benchmark (both 2D and 3D). The solver runtime is summarized in Table 9. We divided overall time into six intervals. During the analysis, the solver analyses the user's settings. According to the settings, the pattern of sparse matrices is built (matrices $\boldsymbol{K}$ and $\boldsymbol{B}$). With the known pattern and type of matrices, the FETI structures are initialized (coarse problem $\boldsymbol{GG}^T$, symbolic factorization of $\boldsymbol{K}$). The output of these three steps is dependent on unstructured mesh and its domain decomposition. Hence, we call these three steps only once at the begging of the computation. Then, the assembler fills the values for all matrices and vectors ($\boldsymbol{K}, \boldsymbol{R}, \boldsymbol{B}, \boldsymbol{f}$). With evaluated matrices, we update the FETI solver by calling numerical factorization of $\boldsymbol{K}$ and computation of explicit inversion $(\boldsymbol{GG}^T)^{-1}$. Then, the final solution is computed. Whereas assembler and solve intervals together

with numerical factorization of $\boldsymbol{K}$ must be called at every non-linear and time step, recalculation of the explicit inversion can be skipped depending on the user's settings.

The performance of the update and solve steps is mainly dependent on the size of subdomain matrices $\boldsymbol{K}$ and the total number of subdomains that denotes the size of the orthogonal projector $\boldsymbol{P} = \boldsymbol{I} - \boldsymbol{G}^T(\boldsymbol{G}\boldsymbol{G}^T)^{-1}\boldsymbol{G}$. Increasing the number of subdomains decreases their size and vice versa. Hence, with the fixed problem size the optimal number of subdomains can be found. To demonstrate this, we ran weak scalability tests of the solver with a different number of subdomains. As can be seen in the above table, the optimal number of subdomains (the minimal overall time) varies with the size of the benchmark (the optimal setting w.r.t. overall run time for a given benchmark size is highlighted). This behaviour is typical for FETI solver. The number of subdomains also influence the number of iterations of CPG. It can be seen in the last column of the table that decreasing the number of subdomains increase the number of iterations. Note that the number of iterations is not increasing only since we change the shape of the subdomains (from square to rectangle). When we compare the number of iterations for subdomains with the same shape only (even or odd rows), the number of iterations always increases. We can also see in the table that we were not able to run some settings due to insufficient amount of memory. This fact limits the size of a problem that can be solved on IRENE.

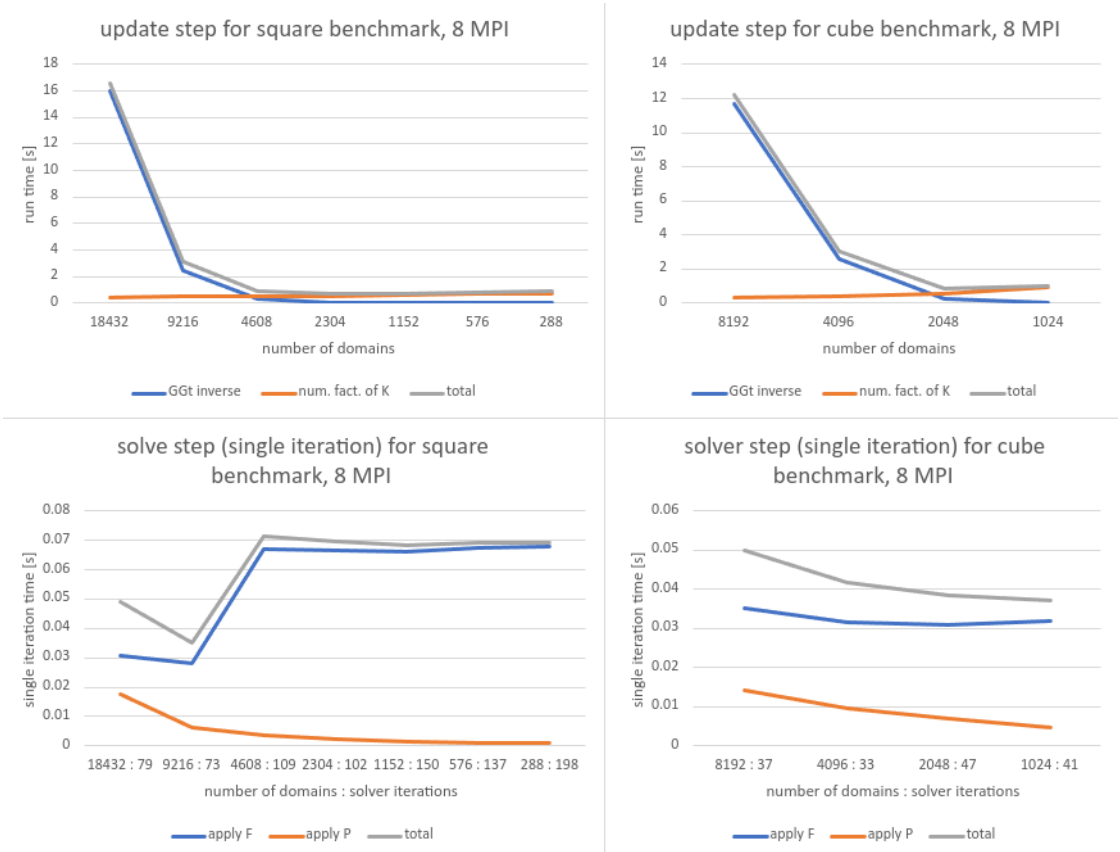


FIGURE 31: PERFORMANCE OF UPDATE AND SOLVE STEPS ON IRENE

The overall performance and the optimal settings of the ESPRESO FEM library is strongly tied with performance of the third-party sparse solver (SuiteSparse) as we will show by more detailed analysis of steps update and solve. Figure 31 shows the performance of these two steps on 8 MPI processes (for other combinations the performance is similar). The performance of update step is dictated by computation of $(\boldsymbol{G}\boldsymbol{G}^T)^{-1}$ and numerical factorization of $\boldsymbol{K}$. Time for numerical factorization of a single subdomain matrix increases with its size. However, time for numerical factorization of all subdomain matrices increases slightly since there is a smaller number of subdomains. On the other hand, we must keep an appropriate size of $\boldsymbol{G}$ since evaluation of $(\boldsymbol{G}\boldsymbol{G}^T)^{-1}$ significantly increase with its size. When the size of $\boldsymbol{G}$ is doubled (by doubling the number of subdomains) we double both the size of $\boldsymbol{G}\boldsymbol{G}^T$(the system that is

factorized by a sparse solver) and the size of $(\boldsymbol{GG}^T)^{-1}$. Usually, there is a limit on the number of subdomains that are manageable before $(\boldsymbol{GG}^T)^{-1}$ starts to be the bottleneck of the FETI method.

In the solve step, the most time demanding is $\boldsymbol{F}$ operator (as can be seen in the lower parts in Figure 31) that is performed by a sparse solver. This operator can be also accelerated by GPU as is described in section below.

Table 10 compares absolute performance of the ESPRESO FETI solver on IRENE and Karolina clusters. We compare three intervals: computation of $(\boldsymbol{GG}^T)^{-1}$, apply $\boldsymbol{F}$, and all other operations together (analysis, build, symbolic factorization, and assembler). As we can see, the performance of $(\boldsymbol{GG}^T)^{-1}$ and apply $\boldsymbol{F}$ is dependent on the size of matrices. Operations with small matrices are faster on Karolina. Operations with large matrices are faster on IRENE. In general, this behaviour is advantageous as FETI method focus on solution of large examples. The explanation of this behaviour is provided by the next section.

TABLE 10: COMPARISON OF ESPRESO FEM FETI SOLVER ON KAROLINA VS. IRENE

	IRENE [s]			Karolina [s]			ratio [IRENE / Karolina]		
subdomains	$(\boldsymbol{GG}^T)^{-1}$	apply F	other	$(\boldsymbol{GG}^T)^{-1}$	apply F	other	$(\boldsymbol{GG}^T)^{-1}$	apply F	other
18432	16.058	0.031	13.332	20.417	0.019	4.073	**0.786**	1.651	3.274
9216	2.489	0.028	13.487	0.750	0.020	4.226	3.321	1.448	3.192
4608	0.294	0.067	13.769	0.176	0.082	4.361	1.671	**0.816**	3.158
2304	0.039	0.067	14.144	0.012	0.085	4.751	3.351	**0.784**	2.977
1152	0.009	0.066	14.538	0.003	0.085	4.857	2.898	**0.784**	2.993
576	0.002	0.067	15.026	0.001	0.089	5.078	3.518	**0.758**	2.959
288	0.001	0.068	15.473	0.000	0.092	5.301	3.408	**0.734**	2.919
8192	11.686	0.035	11.458	23.033	0.025	3.296	**0.507**	1.438	3.476
4096	2.587	0.032	11.664	2.958	0.023	3.415	**0.874**	1.390	3.415
2048	0.225	0.031	11.940	0.277	0.029	3.307	**0.814**	1.067	3.610
1024	0.024	0.032	12.232	0.019	0.050	3.347	1.310	**0.632**	3.655

Hardware counters

In this section we show measurements of hardware counters. We measured the same intervals as in previous chapters. Counters were measured through PAPI interface. Metrics significantly vary for each region as each region contains different kind of algorithms. Table 11 contains metrics for regions that are independent on the number of domains and size of $\boldsymbol{K}$ matrices.

The first three regions represent the performance of mesh processing. Other regions are from FETI solver. The highest IPC has database parser that converts ASCII data to binary format and assembler where mesh elements are transformed into a system of linear equations by local matrix kernels. Other intervals have low throughput and cache miss ratio around 20%. It can be caused by indirect addressing in these intervals.

TABLE 11: HARDWARE COUNTERS FOR ESRPESO FEM LIBRARY

	IPC	throughput [GB/s] / NUMA domain	Cache miss ratio
database parser	1.15	3.69	27.19%
mesh builder	0.89	4.34	22.83%
mesh pre-processing	0.69	7.21	19.93%
analysis	0.81	3.29	35.07%
build system (small mat.)	0.86	0.43	3.95%

build system (large mat.)	0.89	2.16	21.82%
sym. fact. (small matrices)	1.01	6.33	6.06%
sym. fact. (large mat.)	1.11	16.23	24.07%
assembler	1.44	29.38	43.29%

Table 12 and Table 13 show metrics for most computationally intensive parts of the FETI solver. When $(GG^T)^{-1}$ is large (high number of domains) the matrices do not fit into caches. It increases throughput as the matrix need to be reloaded from the main memory. In that case, throughput has the highest impact on the performance. The similar behaviour has the apply $\boldsymbol{F}$ operator. When matrices in external sparse solver fit into the caches, the operator reaches high IPC. On the other hand, throughput and cache miss ratio are lower. Ratio between amount of data that fit into caches and amount of data that are reloader from the main memory is dependent on the size of matrices $\boldsymbol{K}$ and their pattern. Hence, there are differences between 2D and 3D examples. In general, matrices for 2D examples are sparser than for 3D examples as the pattern is denoted by the mesh connectivity and elements in 3D have more neighbors.

These metrics explain the speed comparison of Karolina and IRENE cluster in Table 10. Since FETI solver in ESRPESO FEM is memory bound, the speed in influenced by the speed of memory. Karolina has faster caches than IRENE (Karolina: L2=512KiB/core, hit latency=12+ cycles + L3=16MiB/4 cores, hit latency=~39 cycles; IRENE: L2=8MiB/12 cores, hit latency=37-47 cycles). IRENE has higher memory bandwidth (due to HBM). When majority of data fit into caches IRENE is slower than Karolina. When data are loaded from the main memory IRENE is faster. Hence, IRENE is faster that Karolina when matrices are large and need to be loaded from main memory as is confirmed by tables Table 12 and Table 13. With increasing the matrices size, cache miss ratio and memory throughput are increasing. On the other hand, throughput is still far from maximum (256GB/s for each NUMA domain).

TABLE 12: HARDWARE COUNTERS FOR $(GG^T)^{-1}$

$(GG^T)^{-1}$	time [s]	domains	cache miss ratio	throughput [GB/s] / NUMA domain	IPC
2D	16.068	18432	14.3%	96.1	0.18
2D	2.493	9216	16.8%	136.8	0.29
2D	0.294	4608	1.1%	15.7	0.58
2D	0.039	2304	7.5%	15.9	1.44
2D	0.009	1152	7.4%	6.5	1.48
2D	0.002	576	10.2%	7.2	1.4
2D	0.001	288	18.0%	11.0	1.12
3D	11.682	8192	12.4%	55.8	0.11
3D	2.588	4096	12.6%	55.7	0.1
3D	0.225	2048	0.3%	3.3	0.25
3D	0.025	1024	0.3%	3.8	0.46

TABLE 13: HARDWARE COUNTER FOR APPLY F

apply $\boldsymbol{F}$	time [s]	do-mains	K size	cache miss ra-tio	throughput [GB/s] / NUMA do-main	IPC
2D	0.03	18432	961	28.90%	29.25	0.96
2D	0.03	9216	1891	32.60%	38.58	1.02
2D	0.07	4608	3721	42.50%	24.79	0.59

2D	0.07	2304	7381	44.40%	27.96	0.59
2D	0.07	1152	14641	43.50%	31.29	0.59
2D	0.07	576	29161	43.30%	35.01	0.59
2D	0.07	288	58081	43.50%	38.08	0.58
3D	0.04	8192	729	31.40%	33.61	0.75
3D	0.03	4096	1377	34.20%	44.26	0.73
3D	0.03	2048	2601	36.00%	62.87	0.67
3D	0.03	1024	4913	37.70%	87.04	0.62

GPU acceleration

The solve step executes the PCG iterative solver and its execution time is therefore given by the number of iterations and the iteration time. As can be seen in Figure 31, the most time-consuming part is executing apply $\boldsymbol{F}$ (forward and backward substitution) of the sparse direct solver using the Cholesky decomposition of the $\boldsymbol{K}$. This operation takes up to 95% of the iteration time.

To reach the full potential of a GPU architecture a dense representation of the data that were originally sparse is the key task. In the original FETI solver implementation operator $\boldsymbol{F}$ is evaluated by converting the vector of dual variables $\bar{\boldsymbol{\lambda}}$ to primal variables using gluing matrix $\boldsymbol{B}^T$. The result of the solve routine of $\boldsymbol{K}$ (primal variables) must be again converted into dual variables using $\boldsymbol{B}$. This operation in the algorithm of the projected conjugate gradient method appears in each iteration. Instead of executing these three operations on sparse matrices, we can directly assemble the operator $\boldsymbol{F}$, that is a dense matrix of size equal to the number of rows of $\boldsymbol{B}$. Explicit assembling of $\boldsymbol{F}$ is performed during the update step by sending the factors of $\boldsymbol{K}$ and matrix $\boldsymbol{B}$ to GPU.

We compared times on the IRENE cluster with times measured on 8 x GPU on Karolina GPU partition (one node). The performance of this modification is summarized in Table 14. Explicit assembling of $\boldsymbol{F}$ is needed only for GPU acceleration. It runs after the numerical factorization. Hence, time for assembling $\boldsymbol{F}$ increases FETI solver pre-processing time. On the other hand, iteration time is significantly speed up due to faster applying $\boldsymbol{F}$ operator. The speed up is achieved by higher memory throughput of GPU and reducing the size of matrices that we manipulate with. The reduction of the matrix size can be seen in the last column of the table. Even the reduction ratio is higher for 2D example, in both examples (2D, 3D) reduction increases with increasing the subdomains size. Hence, benefit of GPU acceleration is higher for large subdomains as can be seen in the last column in the table. Unfortunately, possibility to solve large examples is significantly limited by 32 GB / node memory on IRENE.

TABLE 14: PERFORMANCE OF UPDATE AND SOLVE STEP USING GPU

DIM	MPI	GPU	domains	DOFs	num. fact.	assemble F for GPU	apply F	apply F, GPU	surface-volume ratio	apply F speedup
2D	8	8	18432 : 79	961	0.41	5.52	0.031	0.045	0.125	0.7
2D	8	8	9216 : 73	1891	0.51	4.09	0.028	0.031	0.095	0.9
2D	8	8	4608 : 109	3721	0.49	3.66	0.067	0.016	0.064	4.3
2D	8	8	2304 : 102	7381	0.53	3.37	0.067	0.008	0.049	8.5
2D	8	8	1152 : 150	14641	0.59	2.82	0.066	0.004	0.033	15.4
2D	8	8	576 : 137	29161	0.68	2.87	0.067	0.002	0.025	27.5

2D	8	8	288 : 198	58081	0.75	2.86	0.068	0.001	0.017	46.0
3D	8	8	8192 : 37	729	0.33	5.59	0.035	0.033	0.513	1.1
3D	8	8	4096 : 33	1377	0.38	3.58	0.032	0.018	0.447	1.7
3D	8	8	2048 : 47	2601	0.56	3.19	0.031	0.010	0.373	3.1
3D	8	8	1024 : 41	4913	0.91	2.96	0.032	0.006	0.290	5.1

Conclusion of the Analysis

We tested the ESPRESO FEM library on the A64FX partition on IRENE cluster and compared performance with Karolina cluster. The performance is summarized in Table 15, where we compared performance of a single CPU (A64FX vs. AMD EPYC 7H12). The table contains measurement for mesh preprocessing, solver pre-processing, and solution of a single time step that contains operations that must be performed for each time step. Mesh and solver pre-processing can be performed only once. These steps are only slightly influenced by domain decomposition. In general, their performance is worse on A64FX. It can be caused by indirect addressing that is required during these steps.

The performance of operations that are performed each time step is influenced by domain decomposition. Operations with small matrices that fit into caches are faster on Karolina. Operations with large matrices that do not fit into caches are faster on IRENE due to higher bandwidth of HBM memory. This behaviour is advantageous for solving large instances. On the other hand, IRENE has only 32 GB of available memory per node, that limit solving very larger instances by ESPRESO FEM library.

TABLE 15: OVERALL PERFORMANCE COMPARISON OF ESPRESO FEM LIBRARY

	IRENE [s]	Karolina [s]	speedup [Karolina / IRENE]
mesh preprocessing	18.17	3.12	0.17
solver preprocessing	13.48	4.18	0.31
time step small matrices	3.84	2.29	0.60
time step large matrices	9.21	11.99	1.30

Ligand Generator (LiGen)

Application Descriptions

LiGen (Ligand Generator[18]) is an application designed for the virtual screening of biochemical compounds in an HPC environment. It mainly relies on GPU and it's written in CUDA, but it can be also efficiently run on a CPU-based framework. The application itself is a key component of the Exscalate drug discovery platform[19].

Finding new drugs is a challenge which largely stood out with the unfortunate event of recent COVID-19 pandemic. The drug discovery process is long and demanding, and usually involves two different stages: a computational approach - the so called "virtual screening" - in which a restricted selection of candidate molecules are selected from a wide database, and an experimental stage in which effectiveness and toxicity of such selected molecules is tested in real world.

High throughput virtual screening played a central role in recent drug discovery applications, and growing interest bought to deployment of novel applications.

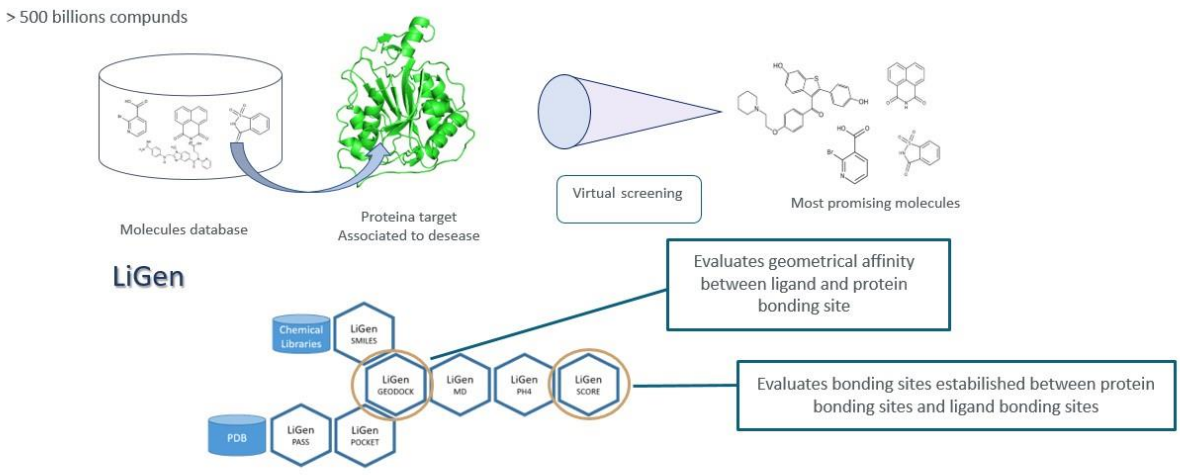


FIGURE 32: LIGEN WORKFLOW

The main phase of virtual screening is called "docking", in which the interaction between a three-dimensional pose of a given molecule called "ligand" and a given target site of a protein called "pocket" are evaluated, in terms of energy (and thus chemical stability). Several implementations of this mechanism are possible: efficiency of this computational step is a key element since an efficient implementation allows to test a wider range of possible candidates.

[18] https://www.ligateproject.eu/workflow.html

[19] The Exscalate Drug Discovery Platform, home page: https://exscalate.com

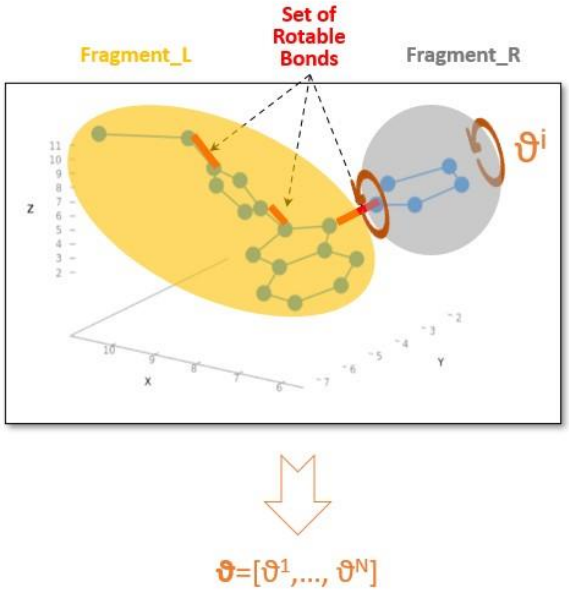


FIGURE 33: DOCKING STEP IN THE LIGEN WORKFLOW

Different conformations of the proteins act as input for the docking step in the workflow: each of them is characterized by a different conformation of the binding site, i.e., the portion of the protein which interacts with the ligand; each rotatable bond inside the ligand is rotated in order to obtain new conformations. For each of these ligand conformations the bonding affinity with the protein is evaluated by an estimate of the actual energy of the interaction, considering several factors like:

- Atomic interactions (e.g., Van Der Waals, Lennard-Jones...)
- Metal bonds
- Hydrogen bonds
- Solvation effects

Application Use Case

The LiGen use case we exploit in Eupex is the one related to the EXSCALATE4CoV European project; its aim is to discover new potential drugs against Covid 19 virus.

LiGen has been co-designed by Dompé S.p.A. and CINECA; in order to keep up with the evolving HPC technologies; it has been kept continuously updated during the years and endured massive code interventions with a strong co-design focus in order to be ready for any urgent computing needs.

The first large-scale application of Excalate platform happened in 2016, when Zika virus pandemic arised and spreaded in several areas, mainly from South America to North America, beside Southeast Asia and Pacific islands. In the context of the Antarex project, LiGen has been largely redesigned and underwent a large improvement in terms of performances.

A more recent large-scale application has been in the context of the above-mentioned EXCALATE4Cov project[20], in the unfortunate recent pandemic occurrence. During the project, LiGen has been massively employed on Marconi 100 cluster at CINECA and HPC5 cluster at ENI Green Data Centre; with a combined throughput of 81 PFlops, the EXCALATE platform allowed to rank a chemical library of more than 70 billion ligands against 15 binding sites of 12 viral proteins of Sars-Cov2. The main simulation involved

[20] The Exscalate4CoV project, home page: https://www.exscalate4cov.eu

both the clusters in full, for a total time length of 60 hours and a trillion of docking operations, resulting in the largest virtual screening simulation so far (50x more ligands and 7.5x more targets with respect to previous record). The outcome of the simulation suggested 40 molecules to be the best candidates as potential drugs against SARS-Cov2, with Raloxifene emerging as the most promising and being employed in clinical trials[21].

The use cases we will employ in the EUPEX project are taken from the ones employed in the EXCALATE4Cov runs, in order to have comparisons with performances on different platform, and to exploit the know-how acquired by such large simulations.

Workflow Analysis

The workload is intrinsically embarrassingly parallel, because every molecule can be tested independently of the others; this feature, joined with the possibility of relying on urgent computing when the unfortunate event of pandemic arises, makes HPC the optimal framework for virtual screenings workflows.

The input is the target pocket and the initial pose of the ligand, and the output is a score assigned to the evaluated pose.

An outer loop checks all the possible poses; for each one of them, inside this outer loop, the docking proceeds in two steps:

i. Rotation of the entire ligand is performed via two nested loops on the angle between ligand and pocket; score is evaluated for each orientation (12 degrees step)
ii. Ligand is reshaped via rotation of its fragments to find the angles which maximize scoring function (without overlapping)

As a compromise between accuracy and performances, this two for cycles are not nested, since it would be too computationally expensive.
At the end of this process, the most promising poses are filtered removing the ones which are too similar to each other; the remaining ones are re-scored.

With such algorithm, parallelization can be applied on rotations, validity checks and scoring, since all these operations are independent.

To have a more qualitative exploration of the code, we can proceed with a deeper look to the details of the algorithm.
The aim of the outer loop is to increase the possibility to avoid local minima starting from different initial configurations. Inside this outer loop, we can distinguish three different main stages:

i. Alignment (i.e., the rigid rotations of the ligands); all these rotations can be tested in parallel, and a reduction is performed at the end of the kernel in order to select the best pose. There's no need to store the tested poses, since they are needed only to evaluate the score, which is computed on the fly.
ii. Optimization of the pose, where each fragment is rotated and evaluated independently; the loop on the fragments cannot be parallelized since some atoms might be included in more than one fragment; however, the scoring phase of each fragment are independent one to each other, thus the inner operations can be run in parallel retrieving the score on the fly, just like in the alignment phase
iii. Rescoring is preceded by a "filtering" step in which too similar poses are removed in order to avoid unnecessary computational effort; at this point only different (enough) configurations are selected, and each atom-atom between ligand and pocket is evaluated for each configuration; differently with respect to previous stages, atoms are not moved here.

[21] The *One Trillion Dock* experiment, home page: https://1trilliondock.exscalate4cov.eu

While the entire workflow of LiGen is currently up and running on A64FX architecture, our first analysis mainly focuses on the GPU implementation of the docking kernel to show the efficiency on GPU architecture). We also ported a vanilla implementation of the same computational kernel on the IRENE A64FX cluster to test the entire workflow without leveraging on GPU accelerators.

Workflow analysis on GPU

We target NVIDIA A100 GPUs due it's our acceleration SDV, and we use the CUDA language to exploit the maximum potential of the architecture. In the next section, we will provide some general considerations on these architectures.

LiGen algorithm and GPU architecture

The algorithm explained in the workflow analysis, has been developed taking in consideration the levels of parallelism needed to fully exploit the GPU. For this reason, we target the SIMT (Single Instruction Multiple Threads) approach, where different threads are executing the same operation on different inputs. This has been done since threads are the inner parallelism level in the CUDA hierarchy. They are organized in blocks (of max 1024 threads), which are themselves organized in grids. A block is mapped on a single streaming multiprocessor (SM), while the grid is distributed across the SMs on the GPU. This organization is visible in Figure 34.

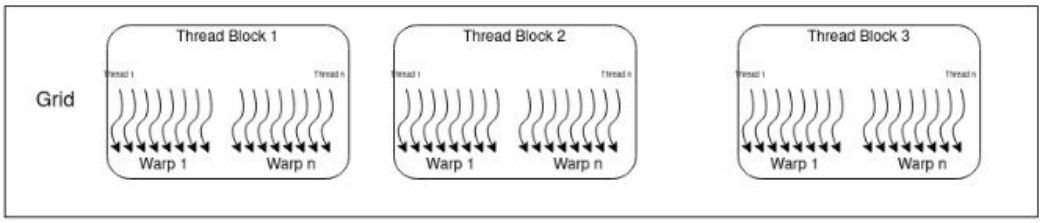


FIGURE 34: GPU THREADS LAYOUT

An important detail in organizing the code is the attention to warps: a warp is a unit of thread scheduling in the GPU and consists of 32 threads. A warp executes one common instruction at a time, so full efficiency is achieved when all threads in the warp execute the same instruction. If the execution paths of threads in a warp diverge via a conditional branch, the warp executes each path disabling the threads that are not on that path (inactive threads). That introduces an overhead since the instructions on the two paths are not executed in parallel. The most common instructions that can cause warp divergence are conditional instructions, such as loops, if statements, and so on. Since warp divergence can negatively impact the performance of a code, it is essential to minimize the number and (more importantly) the length of divergent execution paths within the warps as much as possible.

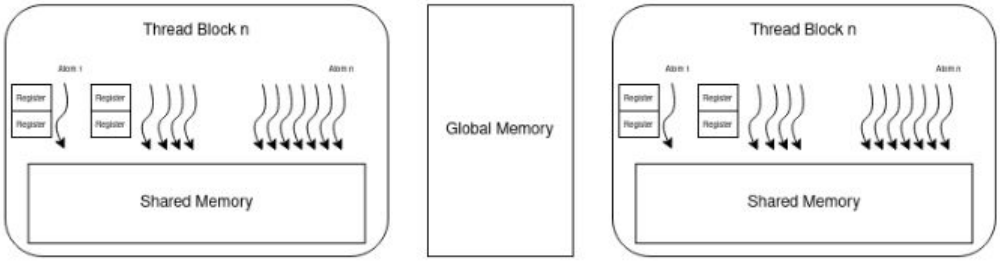


FIGURE 35: GPU MEMORY LAYOUT

Another important feature of the GPU that must be considered is the memory hierarchy, visible in Figure 35. It is really important that all the data are close to where they are needed, however, GPUs have a small cache memory and a different organization with respect to the CPU. There are 3 different levels of memory on the board:

- Global Memory: the slowest (2000 GB/s of band width on A100) and largest memory (up to 80GB on A100) available on the accelerator. This is the memory to which data from the host can be directly copied.
- Shared Memory: a small, low-latency, and high-bandwidth memory shared among the threads of a block, and accessible by all of them. Threads of a block can load shared memory with data from global memory and, by using thread synchronization, shared memory can be used as a scratchpad to implement user-defined data caches. SMs cannot run more blocks than the amount of shared memory they are equipped with allows, and thus shared memory usage of a kernel can limit occupancy.
- Register set: the fastest memory space available to threads. Each SM has 65536 32-bit registers that are used partitioned among the threads of the blocks running on it. In addition to the amount of shared memory required by each block, also the number of registers per thread that a compiled kernel requires determines how many blocks can concurrently run on an SM and, if too high, can limit the kernel occupancy.

Experimental results on GPU

In this section, we will characterize the computational workload in terms of execution profiles, applying the instruction roofline methodology (Ding e Williams 2019), on an input dataset constructed to be representative of different molecule categories from real-world datasets (Mediate: Molecular docking at home. s.d.).

Considering the dimension that affect workload's computational complexity, namely the number of atoms (including hydrogens) and the number of rotatable bonds, three molecule classes have been defined. For each of those classes, a sample molecule from the testing dataset has been randomly selected and then duplicated to produce a uniform input dataset allows for homogeneous execution paths across all warps involved in a single kernel grid, especially for the batched implementation where each warp takes care of different input ligands. The test molecule classes have been defined as:

- Small: (0, 64] atoms, 1 rotatable bond, batch of 1920 molecules;
- Medium: (64, 96] atoms, 12 rotatable bonds, batch of 1600 molecules;
- Large: (96, 160] atoms, 20 rotatable bonds, batch of 960 molecules.

We analyse how the computing resources are used by the kernel using roofline plots (Ding e Williams 2019) produced by measuring both implementations execution behaviours via NVIDIA NSight profiler (NVIDIA Nsight Compute Kernel Profiling Guide. s.d.) in Figure 36.

In particular, we report the instruction issued roofline, which is obtained by considering all kinds of warp-level instructions issued. From this graph, we can say that the kernel implementation of the application is not memory-bound. Moreover, since we are close to the roof, we can say that this implementation uses the GPU appropriately. If we look at the behaviour on the size of the different molecules the amount of issued instruction per second increase with the molecule size. This is expected due to respective scaling design the number of instructions increase because the efficiency of the kernel is constant, and the amount of data increases.

Another insight given by the plot is the cache reuse: the horizontal distance between points of the same molecule class represents the ability of the cache to satisfy a request. The larger the distance between two points, the highest the reuse of data present in the highest-level memory (i.e., distance between L1 and L2 caches represents the ability of the L1 cache to serve the read request). From the top plot of Figure 36, we can see a regular cache reuse across molecule classes, and we can notice that the reuse of the L2 cache increases with the size of the ligands. Larger molecule classes begin to rely heavily on L2 cache: this can be seen by the fact that the HBM arithmetic and instruction intensities are ~100× higher than L1 and L2 values.

Instead, the below plot of Figure 36, reports the shared memory roofline. This is obtained by measuring both warp-level load/store instructions issued and shared memory transactions performed). The x-axis indicates within the interval between no bank conflict and 32-way bank conflicts how efficient the kernel is in terms of shared memory access. The kernel shows little to no impact due to shared memory bank conflicts and thus an efficient access pattern, this also mean a good utilization of the GPU for small/medium ligands.

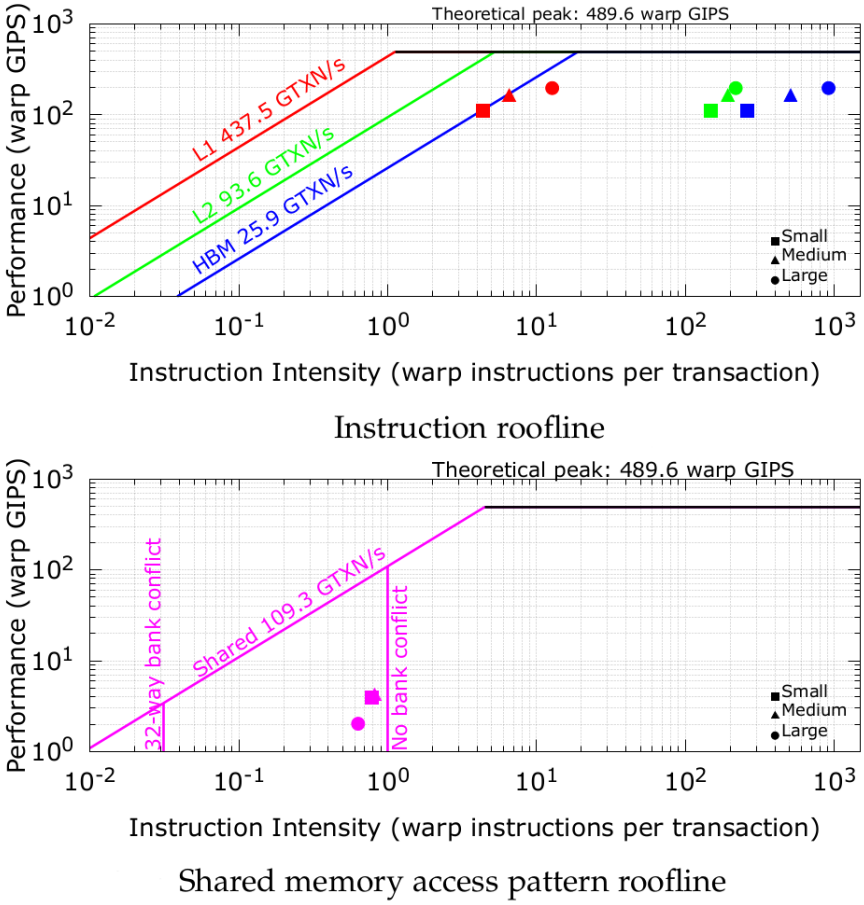


FIGURE 36: ROOFLINE ANALYSIS ON INSTRUCTION PERFORMANCE AND SHARED MEMORY ACCESS PATTERN

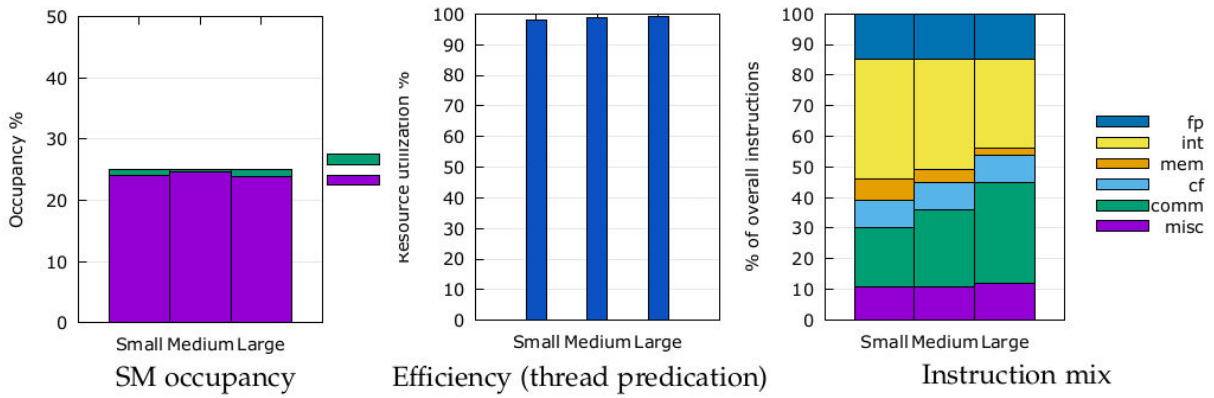


FIGURE 37: PEAK AND SUSTAINED ACTIVE WARPS (ON THE LEFT), EFFICIENCY (THREAD PREDICATION, IN THE CENTRE), AND INSTRUCTION MIX (ON THE RIGHT).

Moreover, provide an investigate the execution profile of the kernel. The results of this analysis are reported in Figure 37. The left image of Figure 37 reports the occupancy, defined as the ratio between sustained and peak percentage of active warps per SM (measured by *sm__maximum_warps_per_active_cycle_pct* and *sm__warps_active.avg.pct_of_peak_sustained_active* metrics respectively (NVIDIA Nsight Compute Kernel Profiling Guide. s.d.)). Occupancy is one of the factors that can be used to improve performances, but it's not the only one since it is possible to reach optimal performances by

decreasing the occupancy and having more registers per thread (Volkov 2010). It's interesting to notice that the occupancy behaviour on different size of molecule is uniform, which also means that the SM occupancy cannot be improved with small molecule.

The second image reports the efficiency (Figure 37), defined as the degree of thread predication across all the instructions executed in a single SM Sub Partition (or SMSP (NVIDIA Nsight Compute Kernel Profiling Guide. s.d.), measured by the *smsp__thread_inst_executed.sum* metric for thread-level instructions and *smsp__inst_executed.sum for warp-level* instructions). The kernel shows high degrees of execution efficiency and thus low degrees of thread predication. This plot demonstrates that the kernel is quite efficient in the use of resources.

Finally, the third image reports the instruction mix (Figure 37), defined as the percentage of instructions executed in a single SMSP grouped by instruction type:

- *fp*: floating point instructions (any precision, including scalar, FMA and tensor),
- *int*: integer instructions (any integer data type),
- *mem*: memory operations (load/stores),
- *cf*: control flow operations,
- *comm*: inter-thread communication and synchronization,
- *misc*: everything else including bit-wise operations and casts

There are two interesting pieces of information in this figure. The first one is that the largest part of the operation done is integer arithmetic. This is expected since they comprehend index calculations and the Score function used to select the best pose is a sum over integer values. Moreover, we can notice a large (20 to 40 %) of comm instructions that suggest to us that there is room for algorithm improvement

Workflow analysis on CPU

In the last section, we show the architectural efficiency of the main computational kernel of LiGen. Our GPU exploration show that even the most efficient implementation of this kernel on a conventional architecture won't never reach the performance on an accelerated HPC system (Nvidia A100 has 3x the peak performance of an A64FX and even of Rhea processor). But to test the entire workflow on IRENE cluster we tried to explore the performance and efficiency of a vanilla implementation of the computation kernel for conventional architecture. This allows us to test the entire LiGen workflow without GPUs.

We start our exploration using different compilers with different configurations to port the entire workflow on the IRENE cluster.

The GCC suite (ver. 11.0) works almost flawless with no particular interventions needed on the compilation tools beside minor interventions on the CMake files. Current results of the workflow obtained by such compilation will be presented below in this section.

As a rule of the thumb, we expect the best performances with the most optimized compiler on the A64FX microarchitecture, which is Fujitsu compiler; unfortunately, the current compilation with the LiGen vanilla code presents some incompatibilities between the standard C++ library included in the Fujitsu suite and the function calls currently present in the code, in particular the ones related to the file system modules; we intend to explore such issues more deeply in view of the next deliverable, and to address issues/workarounds/solutions to the other concerning EUPEX work packages if needed.

In addition, the main dependences for LiGen are Boost libraries, which presented several challenges both for Armclang and Fujitsu compiler concerning the modules needed by LiGen; we will investigate further also these paths in the context of the optimization of the performances in the future steps.

During the work in the WP3, we were able to port the entire workflow on the A64FX-based IRENE cluster at CEA; we analysed the kernel used in the GPU, the metrics are obtained using perf events via perf-based Likwid compilation, running on 48 cpus (full node).

Unfortunately, due the low memory of A64FX (only 32 GByte with respect 80GByte available on the Nvidia A100), we were only able to run the small case reported for the GPU exploration. The Table 16 shows the initial exploration results of the computational kernel.

TABLE 16: SHOWS RESULTS ARE OBTAINED WITH O3 OPTIMIZATION AND SVE VECTORIZATION ENABLED, WITH NO INSTRUMENTATION IN THE CODE

Metric	Value
Average IPC	1.06 avg per core
Vectorization ratio	27.05% avg per core
FLOPS	767.17 MFLOP/s avg per core (36823.97 MFLOP/s per node)
Cache hit/miss ratio	1.22e-5% avg per core
Memory bandwidth	65.29 MB/s avg per core (3.1 GB/s per node)

As we can see, these results are very poor in term of architectural efficiency with respect the GPU kernel.

As we can see for the GPU exploration, the kernel is mainly compute bound so if the vectorization level of the code is not perfectly tuning the kernel can suffer of severe performance degradation with a low utilization of the architecture (low IPC).

Furthermore, to have a very rough idea of the implications that the A64FX microarchitecture implied in the kernel performance, we also performed the same test on an environment base on Intel architecture (48x CascadeLake cpus), maintaining the same approach with the same limitations (e.g., no instrumentation), same level of optimization (O3), with AVX vectorization, same compiler (GCC 11.0).

The performances are quite low also on Intel architecture, but still far better with respect to what has been obtained on IRENE in terms of throughput: just to give a general idea, the DP+AVX flops on Intel cores are roughly a factor x4 w.r.t. DP+SVE flops on A64FX. We think that Intel architecture with higher cores' frequency and complex pipeline architecture are able to maximize the architectural efficiency even on poor-designed computational kernels.

Conclusion of Analysis

As highlighted in the previous section, the entire workflow is up and running on A64FX.

We also ported a vanilla implementation of the main kernel at the base of LiGen but, as expected, we had very poor performance on conventional architecture.

Taking into account the overall contribution to the net pipeline throughout given by the host kernels versus the accelerator kernels (as shown by our analysis), it's our conclusion that focusing on the GPU part of the workflow would be much more profitable than doing the same on the CPU part. Moreover, this prioritization would be in line with the design decisions and development/optimization work done so far on the pipeline itself by the development team.

Moreover, depending on the GPU technology used on the EUPEX platform, it would be necessary to port the CUDA kernels to an implementation that rely on open standard programming models (like SYCL and/or OpenMP) enabling different acceleration technologies for EUPEX platform.

BigDFT

Application Descriptions

Starting in 2005, the BigDFT EU project aimed to test the advantages of Daubechies wavelets as a basis set for DFT using pseudopotentials. This led to the creation of the BigDFT code, which has optimal features of flexibility, performance and precision. In addition to the traditional cubic-scaling DFT approach, the wavelet-based approach has enabled the implementation of an algorithm for DFT calculations of large systems containing many thousands of atoms, with a computational effort which scales linearly with the number of atoms. This feature enables electronic structure calculations of systems which were impractical to simulate even very recently.

BigDFT has rapidly become a mature and reliable package suite with a wide variety of features, ranging from ground-state quantities up to potential energy surface exploration techniques. BigDFT uses dual space Gaussian type norm-conserving pseudpotentials including those with non-linear core corrections, which have proven to deliver all-electron precision on various ground state quantities. Its flexible poisson solver can handle a number of different boundary conditions including free, wire, surface, and periodic. It is also possible to simulate implicit solvents as well as external electric fields. Such technology is employed for the computations of hybrid functionals and time-dependent (TD) DFT.

BigDFT is free and open-source software, made available under the GPL license. The code is developed by few individuals, ranging from 6 up to 15 people. The active code developers are, or have been, located in various groups in the world, including EU, UK, US, and Japan. Its conception and developments were conceived since the early stages to test new approaches oriented to the exploitation on massively parallel computing resources. For these reasons, in addition to production calculations aimed at scientific results, the code has been often employed as a testbed for numerous case-study in computer science and by hardware/software vendors, to test the behaviour of novel/prototype computer architectures in realistic runtimes.

Looking ahead to the exascale era, BigDFT developments have focused on both internal code aspects (compilation and modularity, parallel performance), as well as external usability, both of which are necessary for an efficient transition to ever-evolving architectures.

The compilation of the code suite relies on the splitting of the code components into modules, which are compiled by the bundler package. This package lays the groundwork for developing a common infrastructure for compiling and linking together libraries for electronic structure codes, and it is employed as the basis for the ESL bundle. It can be used to install the BigDFT suite, as well as a variety of optional packages such as python modules that can be used with PyBigDFT.

BigDFT is an award-winner DFT code, recipient of the first edition (2009) of the Bull-Fourier prize for its “the ground-breaking usage of wavelets and the adaptation to hybrid architectures,combining traditional processors and graphic accelerators, leading the path for new major advancements in the domain of new materials and molecules". It is parallelized using a combination of MPI and OpenMP and has support for GPU acceleration since the early days of GPGPU computing. Such supports involve both CUDA as well as OpenCL computing kernels, and can be routinely applied to large systems. For example, the calculation of a 12 000 atoms protein system requires about 1.2 hours of wall-time on 16 nodes of the IRENE-ROME supercomputer. This calculation can be further accelerated for systems composed of repeated sub-units using a fragment approach for molecules, or in the case of extended systems, a pseudo-fragment approach, both among the outcomes of MaX2 project.

To facilitate driving the calculations of dense workflow graphs involving thousands of simulations of large systems, the code suite includes a python package called PyBigDFT as a framework for managing DFT workflows. PyBigDFT is able to handle building complex systems or reading them from a variety of file types, performing calculations with BigDFT linked with AiiDA package, and analyzing calculation results.

Currently, BigDFT is being deployed on large HPC machines including Fugaku (RIKEN, JP), Archer2 (Edinburgh, UK), and IRENE-ROME (TGCC-CEA, FR).

Application Use Case

A challenging task of modern biochemistry is to propose reliable and systematic strategies to modulate the stability of protein/protein assemblies, like the increase of binding affinity with respect to a target antibody as much as desired.

Recent advances in BigDFT code have made possible to examine the systems' density matrices when large molecules - large enough to be biologically relevant - interact with each other.

For instance, the threat of new SARS-CoV-2 variants underscores a grim possibility: the emergence and spread of new strains that may escape currently effective antibodies. Here we would provide tools which would enable us to ask a simple yet critical question: can we predict, before spreading, which variants are likely to arise and prepare to counter them? Existing data suggests that high efficiency attachment to the host cell and avoidance of neutralising antibodies are the two main drivers of the emergence of the most pernicious SARS-CoV-2 variants. This is highlighted by the emergence and even fixation of SARS-CoV-2 variants with mutated spike proteins in a way that either increases binding to human ACE2 (hACE2) as the natural substrate, confers resistance to antibodies, or both.

In this use case, we wish to propose a workflow employing Electronic Structure Calculations of biological systems of many thousand atoms that is compatible with high throughput screening applications. In order to demonstrate the feasibility of our approach we will associate this code to highly accurate approaches of molecular dynamics (the Polaris polarizable force field code) in order to produce populations of data that are able to address for dynamical quantities (solvent and temperature effects) as well as entropic contributions.

While other LS codes such as ONETEP and Conquest are able to treat large systems, including biologically relevant materials, a complexity reduction framework has been implemented in BigDFT which allows an automatic and unbiased analysis of large systems, by identifying fragments and inter-fragment interactions using quantitative indicators which naturally arise from the localized approach inherent to LS-BigDFT. This approach has been applied to complex biological systems and offers the possibility of both offering new insights into their electronic structure and informing the setup of QM/MM without any need for prior knowledge of the system.

Most of the available theoretical tools devoted to it focus on investigating close contact antibody/antigen, AA, local regions and usually ignore the effect on affinity of more distant domains. Because of the size of AA assemblies (they refer to at least 500 protein residues) only standard pairwise molecular modelling force fields or empirical cost functions are traditionally used to quantify the strength of the interactions within them.

In 2020, in the context of the COVID-19 pandemic we proposed a sequential multi-scale Molecular Modelling/Quantum Mechanic simulation approach, whose accuracy was assessed by investigating the microscopic factors at the origin of the stability of SARS-CoV-2 main protease/inhibitor complexes, both peptidic and based on drugs. Such an approach, based on the BigDFT code combined with dynamics extracted from polarisable force fields MD simulations, has been awarded a SANOFI iTech Award and is able to simulate protein assemblies comprising up to 20k atoms, the atomic size of standard AA assemblies. We developed for this aim an original and synthetic representation of the mMM/QM simulation results as an interaction graph (IG) showing both the details and the magnitudes of the interactions (at the residue scale) responsible for the stability of a AA assembly.

We propose to employ large-scale ab initio quantum mechanics-based workflows to predict future SARS-CoV-2 variants and understand the interaction with respective neutralising antibodies. We develop an original ab initio in silico tool to accurately estimate the magnitude of protein/ligand or protein complex interaction energies by coupling a quantum chemistry (QM) approach (BigDFT) to readily perform computations on molecular systems comprising thousands of atoms, and a polarizable multi scale molecular modeling (MM) approach (Polaris) to efficiently sample the potential energy surface of large molecular systems in aqueous phase up to the million atoms scale. The MM approach is then used to generate statistically meaningful sets of molecular structures that are then post processed using the QM approach. Among other possible applications, by coupling both these approaches we can (1) define unequivocally the protonation states of key protein residues (like catalytic residues), (2) refine and/or assess MM polarizable force field to handle any ligand molecules; (3) identify the microscopic factors

modulating the catalytic activity of enzyme (like reaction energy barriers); (4) and decompose the interaction pattern of protein complexes on a set of “fragments” for instance.

As an example of application of our coupled approach, we are using it to investigate the interaction between new peptidomimetic -ketoamide inhibitors and the main SARS-COV-2 protease. That example shows that we are able to propose reliable starting structures for standard docking approaches as well as to assess the validity of docking solutions.

Our ambition is to propose a screening methodology that can serve as a reference and calibration for less time-consuming methods. This methodology would be more precise than what exists by combining the electronic structure aspect with the dynamic aspect of proteins.

This workflow, assisted by unprecedented abundance of data available on the SARS-CoV-2 spike, will allow us to verify the applicability of ab initio QM modelling to events of biological relevance. The shift to a first principles-based description of protein-protein interactions will have far-reaching impact in many biological research (both applied and basic science) beyond COVID-19.

Workflow Analysis

Our study is performed via a full Quantum Mechanical (QM) model, as implemented in the BigDFT computer program suite. The approach employs the formalism of Daubechies wavelets to express the electronic structure of the assemblies in the framework of the Kohn-Sham (KS) formalism of Density Functional Theory (DFT). The electronic structure is expressed by both the density matrix and the Hamiltonian operator in an underlying basis set of support functions – a set of localized functions adapted to the chemical environment of the system. Such functions are expressed in Daubechies wavelets, typically using one to four support functions per atom as the basis set. The electronic density matrices, as well as the Hamiltonian expressed in the BigDFT basis set, are analyzed to provide quantum observables of the systems. The code provides efficient and accurate QM results for full systems of large sizes, delivering excellent performance on massively parallel supercomputers. In the present study, we employ the PBE approximation corrected by dispersion D3 correction terms. Each of the calculations presented here requires about 2 h of wall-time on 32 compute nodes of the IRENE-Rome supercomputer, at the TGCC Supercomputing centre (France). A similar approach has been previously used, in conjunction with the other atomistic techniques described in the introduction, to investigate the interaction patterns of the SARS-CoV-2 main protease with natural peptidic substrates and to design peptide inhibitors tested in vitro.

Starting from a representative 3D model of the molecules as our input, we calculate the system's electronic structure, from which we extract various quantities. We draw a contact network to identify relevant chemical interactions among the target protein and the various interactors considered in this study. The strength of the inter-residue interaction is quantified by the Fragment Bond Oorder (FBO), calculated using the electronic structure of the system in proximity of a given residue.

We use the FBO to identify the interface residues, defined as the amino acids of the counter-ligand that have a non-negligible value, above a set threshold of the FBO, with the ligand. In contrast to a simple geometrical indicator like the RBD-ligand distance, the FBO provides a metric that enables a nonempirical identification of steric hotspot interactions. We here identify as chemical hotspot interface residues the amino acids, which exhibit a FBO value with the ligand larger than 7 × 10–3. Such a threshold is obtained by comparing the hydrogen bonding interaction network of the SARS-CoV-2 main protease to its natural peptidic substrates, derived from traditional FF analysis and the equivalent FBO network.

Once the chemical connection among amino acids is identified, we assign to each residue its contribution to the binding interaction between the two subsystems. We calculate these interaction terms from the output of the DFT code and interpret them as two parts. First, a long-range electrostatic attraction/repulsion term, defined from the electron distributions of each of the fragments (even when far apart, two fragments may still interact). The remaining term, which can only be attractive, is provided by the chemical binding between the fragments, and is nonzero only if the electronic clouds of the fragments superimpose (short-range). This term is correlated with the FBO strength, and we identify it as the chemical interaction.

By including long-range electrostatic terms, the decomposition enables us to single out relevant residues not necessarily residing at the interface. In this way, the model provides an ab initio representation of the RBD-ligand interactions as the final output.

For each protein/protein structure considered, the workflow will employ hundreds of DFT calculations extracted from equilibrated MD trajectories, and combine together the data extracted on a per-snapshot basis. The interaction fingerprint is derived by suitable manipulations the full systems' density matrices, expressed in the localised basis set of BigDFT approach, to identify de novo the relevant fragments that participate in the chemical bonds and network of inter- and intra- protein interactions. This realises a mechanistic, chemical (i.e., non-geometrical), first principles-based identification of the protein regions which are relevant for the interaction.

Because of its first-principle nature, this approach does not require a priori knowledge about the system under analysis and can be applied to any atomic system for which a all-atom structure is available.

We think this approach can inform drug design efforts and offer new insight about intermolecular interactions by granting access to variables otherwise impervious to observation.

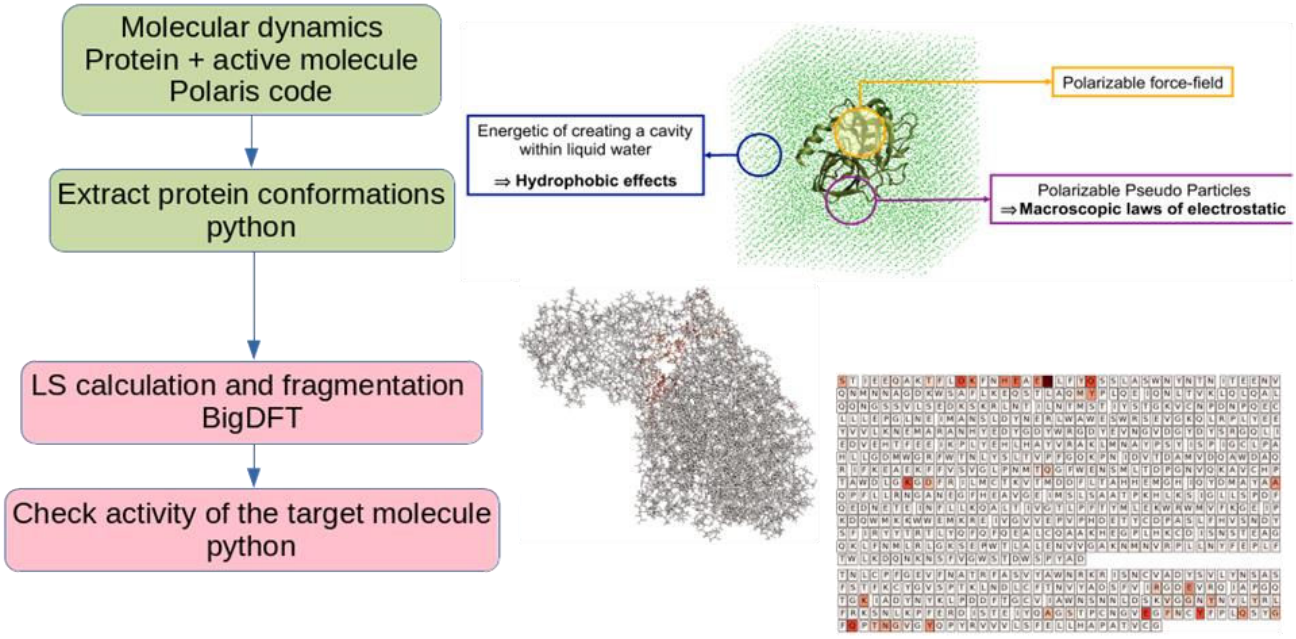


FIGURE 38: ONE MONOCLONAL ANTIBODY (1400 RESIDUES), 3H WALLTIME (32 NODES, 4096 AMD ROME CORES)

Conclusion of Analysis

We have analysed a number of calculations performed by BigDFT to quantify the resources employed on a per-atom basis. Being the approach O(N) in the number of atoms, it is interesting to notice the memory-per-atom and the CPU time per atom of a given calculation. This enables to quantify the computational cost of a workflow with respect to a provided system.

We see in the figures below that on IRENE-ROME, each atom takes about ¼ of CPU hour in the regime of the proteins employed for this workflow, and roughly 300 Megabytes of memory. Such performance figures enables us to dimension the calculation on a large-scale system and identify the number of nodes that have to be involved. Presently the limitations on IRENE-Rome come from the communication scheduling that has performance degradation for fine granularity of the parallelisation (few dozen atoms/per node). Improving the communication behaviour of the code (particularly the scheduling of the one-sided communications) will be the object of the forthcoming activities.

We have also measured the same data on Fugaku machine, which provide clearly the same amount of memory per atom and a CPU time per atom that is roughly three times slower. Yet, the data on the parallelisation and the scalability are much better on the Fugaku machine and enable for better decomposition of the system. On this architecture, the problem is mainly provided by the low amount of memory per node, which limits the size of the systems that can be treated to some twenty thousand atoms.

OpenGadget3

Application Description

GADGET (**GAl**axies with **D**ark matter and **G**as int**E**rac**T**) is an astrophysical numerical simulation application for performing cosmological simulations originally developed by Volker Springel at MPA. With a large user community and a number of slightly different custom versions, particularly in Europe, it allows the following of many physical processes.

GADGET works in both "physical" and "comoving" coordinates. This means that the code is well-suited for standard numerical and cosmological computations. An example of standard numerical computation is the evolution of a model galaxy whose initial conditions are devised to represent the Milky Way's properties as we observe it today. In contrast, Cosmological computations usually start from an early phase of the Universe evolution, as deduced, e.g., by the data we have on the properties of the Cosmic Microwave Background - almost 13 billion years ago - and follow the formation and evolution of structures in a fully cosmological context, as described by our models, reaching the present time.

OpenGadget3 is a fully OpenMP/MPI parallelized version of the non-public cosmological code P-Gadget3, developed by LMU and INAF with the aim of be publicly released it in its full-physics version (whereas other public versions only include Gravity and Hydrodynamics). It solves two major scientific problems. On the one hand, it solves the evolution of large-scale structures in an expanding cosmological background. On the other hand, it describes the coupled evolution of observed physical properties of gravitationally bound objects at smaller scales, like cluster of galaxies, galaxies, and local regions therein, shaped by non-gravitational effects.

Modelling

In these simulations, the universe is modelled as a set of particles interacting through gravity and hydrodynamics. Thus, the scientific problem consists of two distinct parts: (i) the calculation of long-range interactions affecting all of the computational elements of the chosen domain, i.e., the gravity force; and (ii) the calculation of short-range interactions that couple each particle with its neighbourhood. The latter interactions are almost local and only affect ordinary matter (in astrophysics, called "baryonic" matter). Among them, the most important of such interactions is hydrodynamics. A large variety of additional physical effects acting on baryonic physics - transport processes, sink particles and sub-grid models to follow chemical networks, stellar evolution, treatment of supermassive black holes and ore - are also included and interest local scales as well.

GADGET computes gravitational forces using a TreePM technique. For nearby regions of the computational domain, all particles interact among themselves; while (in first approximation) only the center of mass of far regions is considered by using the multipole expansion of the matter distribution within a given aperture angle. This means that the mean field approximation is used for large scale - called Particle-Mesh, PM - while a Treecode is used at smaller scales. In the latter case, the computational domain is partitioned using an oct-tree.

Hydrodynamics is solved using the so-called Smoothed Particle Hydrodynamics technique (SPH). In this case, a particle represents a fluid element whose thermodynamic properties, such as density, pressure, entropy, are obtained from those of neighbouring particles. Using a kernel with suitable characteristics, the obtained values are smoothed over a given physical scale (smoothing length). The smoothing length gives the resolution of the computation; only information on particles within such a scale is needed to calculate hydro forces.

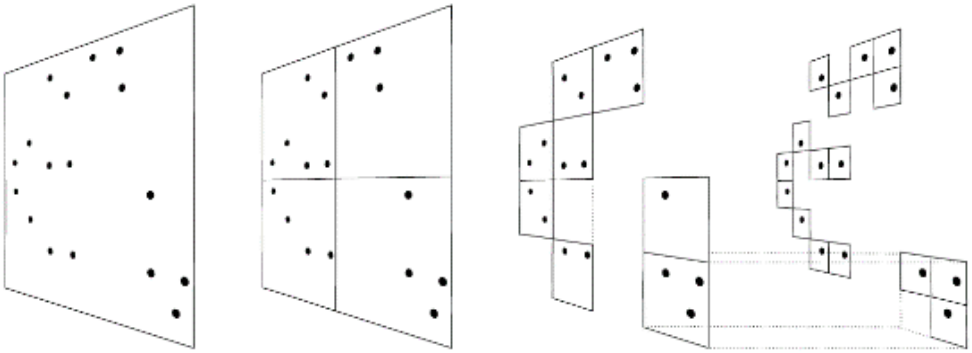


FIGURE 39: OCT-TREE-BASED SPACE DECOMPOSITION IN A 2D EXAMPLE. THE REPRESENTATION IS GLOBAL FOR DOMAIN DECOMPOSITION AND LOCAL FOR THE PARTICLE'S DISTRIBUTION (SPRINGEL 2011).

Simulation
Gadget works iteratively, with each iteration (in general) executing all the *simulation phases*. Each iteration is called a timestep, within which the whole simulated system goes through each simulation phase once. Depending on the state of the simulated system, a timestep may progress the simulation by a non-fixed amount of time. This amount of progress is called *simulated time*.

More specifically, the progress of a simulation can be summarized as follows:

1. The physical time range is subdivided into 2b time-bins, where b is either 29 or 61 depending on whether an integer or a long integer precision is adopted to represent a single timestep (the latter offering a much larger precision needed in some cases). The possible timesteps of the particles are then discrete, allowing the leap-frog integrator to be symplectic.
2. Upon the desired integration accuracy is reached, the simulator assigns to each particle an individual timestep that depends on its own physical conditions. Particles that suffer from more violent dynamics will have a smaller timestep to achieve accuracy on their trajectories, while particles that experience a less dramatic evolution (for instance, because they are in large voids outside the densest regions) will have a larger timestep.
3. At a given time t a particle will then have its own timestep Δ; at time t+Δ, the particle will be considered as active, its physical properties (e.g., position, velocity, and acceleration; or in the case of a gas particle, the density, energy, star formation rate, and cooling rate) will be updated, and it will receive a new timestep Δ depending on the same physical properties.
4. In a given code step, all the active particles are processed, with the simulator updating the current physical time and calculating the new code-step accordingly to the population in the lowest level of the time-bins hierarchy (i.e., the smallest timesteps).
5. Group finding is also a basic analysis task for finding virialized dark matter halos and their embedded gravitationally bound subhalos which can be run both on the fly and in postprocessing. Group finding in OpenGadget3 is supported through two main algorithms, the classic friends-of-friends (FOF) approach to find groups of particles of approximately virial overdensity, and the SUBFIND algorithm to identify gravitationally bound substructures in these groups in configuration space. **Note that the tree-walk is a very fundamental operation that happens very frequently to provide neighbours-finding in different scopes.**

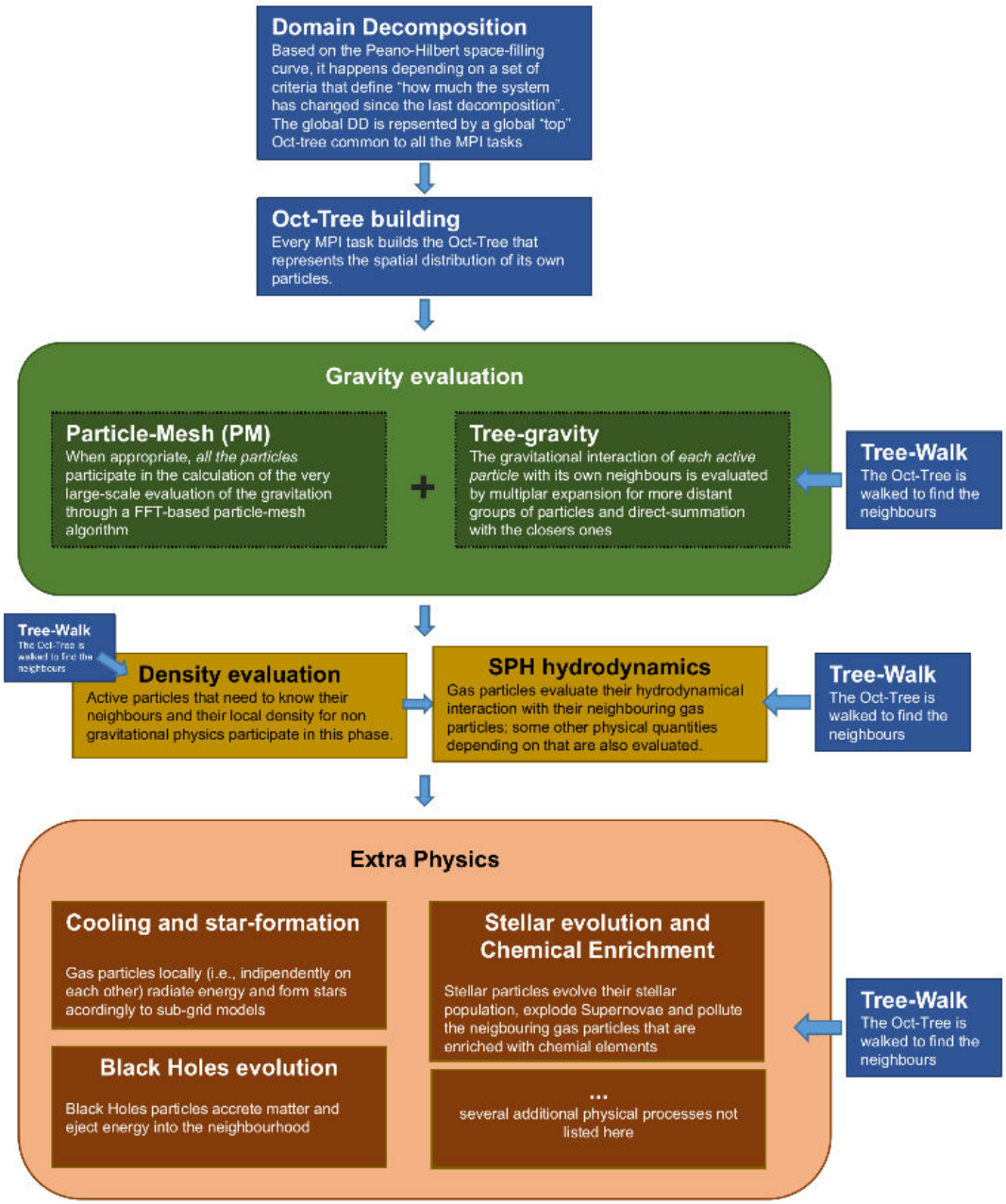


FIGURE 40: A SUMMARY OF THE MAIN OPERATIONS ACCOMPLISHED IN A SINGLE CODE-STEP. EVERY CODE-STEP CONSISTS OF DIFFERENT PHASES, WHICH CAN VARY FROM CODE-STEP TO CODE-STEP. IN THE PICTURE, BLUE ELEMENTS REPRESENT "STRUCTURAL" OPERATIONS INTERNAL TO THE SIMULATOR WHEREAS AS NON-BLUE ELEMENTS REPRESENT PHYSICAL PROCESSES.

With all the above, it becomes evident that OpenGadget3 is a large (approximately 250K lines of code) and sophisticated application with multiple execution phases, each performing different computations, implementing different algorithms, and hence changing its performance profile over time. (i.e., change from compute-bound to memory-bandwidth-bound, and vice versa). Currently, OpenGadget3 is being deployed mainly on large x86_64 machines hosted in several Tier-0 level computational centre (among others: LRZ in Germany and CINECA in Italy).

Workflow Analysis

Besides the simulator itself, OpenGadget3 involves numerous steps essential to successfully describe the formation of cosmological structures and the galaxies within. A typical workflow consists of sequential stages, which can be classified into five overall processes: (i) prepare the simulation environment, (ii) perform the particle simulation, (iii) perform post-processing for data analysis, (iv) event correlation, and (v) visualization. These processes are detailed below:

Environment Preparation: This is a generic stage that involves preparing the run folder for the simulation data, copy the appropriate grid's initial conditions (e.g., particle position) and parameter files (e.g., mass resolution) for the desired experiment, as well as use the appropriate config file to compile the simulation code.

Particle Simulation: OpenGadget3 generates three types of datasets: (i) log files that contain information about the internal state of the simulator and resource utilization statistics, (ii) snapshots of the universe that list all the particles' data (location, velocity, etc.) at different epochs, and (iii) FOF files that contain results from on-the-fly Friends-of-Friends finder.

Data Analysis: Running Gadget in SUBFIND mode to produce data on clustering and galaxies. This stage goes through the simulation data to save the IDs of particles that belong to each sub-structure and extract the global properties of substructures.

Event Correlation: Running the cross-indexer to ease the search of particle data from subhalos. This stage correlates the sub-structures produced by Data analysis with the particles' timelines produced by the simulator.

Data Visualization: Finally, scientists need to have accessible ways to understand trends, outliers, and patterns in data. Thus, they heavily depend on visualization tools that analyze massive amounts of data and generate human-friendly representations, like time-series plots and Project 2D maps.

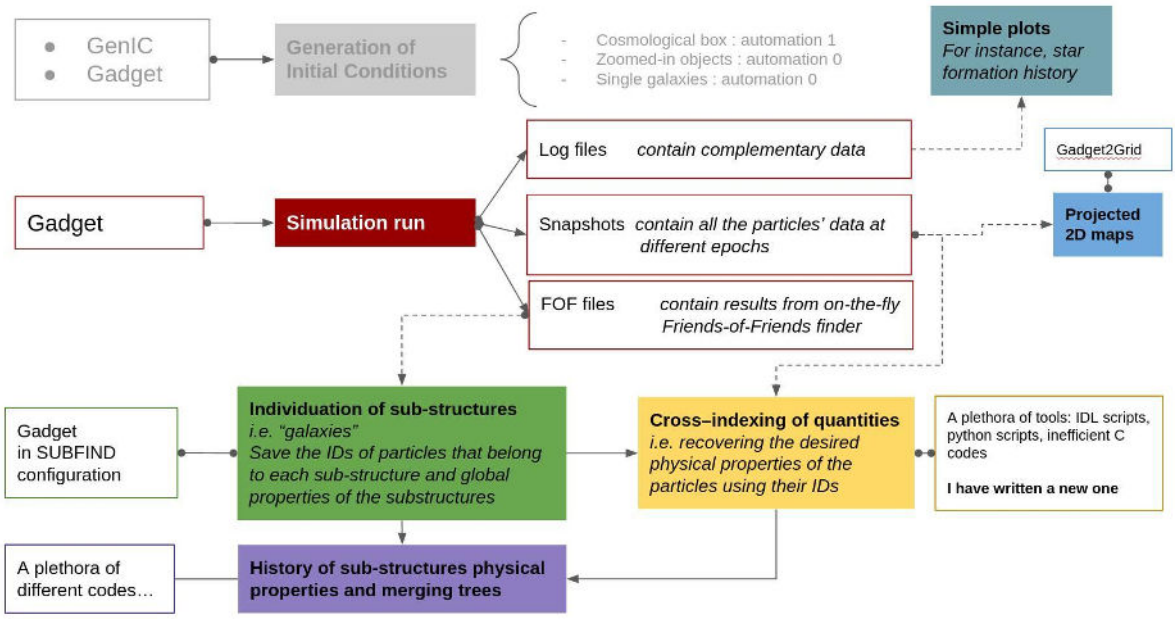


FIGURE 41: OPENGADGET3 WORKFLOW

Currently, most of these steps are maintained in an ad-hoc manner by scientists of different backgrounds, are written in custom Python and Bash scripts, and are tightly coupled to the underlying infrastructure (e.g., fixed paths, environment modules, libraries). Thus, the configuration and maintenance of individual components, and the workflow as a whole, is a complicated and time-consuming job.

Additionally, it is impractical to use large-scale supercomputers for DevOps and/or testing of workflows. This is because HPC resources have long batch queues, and dev time commonly delays as a result. In this context, it is desirable to leverage virtualization technologies so that complex applications can be rebuilt, layered, or shared across heterogeneous computing systems, from laptops to supercomputing resources.

Application Use-Case

In EUPEX, INAF (third party of INGV) and FORTH, will pursue the following two goals:

1. **Optimize the code**
2. **Enable modular and portable workflows.**

Code optimization

The fact that physical processes are applied independently to a huge number of particles gives OpenGadget3 great optimization opportunities through SIMD vectorization. One solution would be to rely on accelerators such as GPUs to deliver high arithmetic performance and memory bandwidth. However, offloading tasks to accelerators comes with its own challenges due to their programming models, which can be hard for applications to exploit. An alternative approach is to **use a new generation of general-purpose CPUs to deliver GPU-class performance while maintaining the ease of use of a traditional CPU.**

The selected CPU architecture used in EUPEX as an SDV for Exascale computing is Fujitsu A64Fx. Based on Arm64 architecture, A64FX can deliver up to 1 TB/s of memory bandwidth by using the same HBM2 technology found in top-end GPUs, and it offers 512-bit-wide vectors through the Scalable Vector Extension (SVE). It is the first CPU to integrate either HBM2 or SVE.

In this use-case, INAF and FORTH will adapt OpenGadget3 to the A64FX architecture, including system software, specialized debugging and performance tools, and will add support of technologies needed for multi-hierarchy architectures expected in the Exascale era (NUMA awareness, vector capabilities, improved modularity).

Enable Modular and Portable Workflows

The Cloud software ecosystem is rich in interactive tools that can significantly benefit HPC users. Among others, that includes (1) workflow management systems, (2) monitoring utilities, and (3) frameworks for scaling out code written in different languages. These tools are usually based on Microservices, packaged in portable containers, and managed by Orchestrators like Kubernetes.

However, the dissimilarity of supercomputing from cloud infrastructure makes it challenging for HPC users to benefit from such tools. Most current Cloud-HPC convergence approaches focus on running container-based HPC "virtual clusters" inside a Kubernetes-based environment (George Zervas e Bilas 2022), or bridging two separate setups (Cloud and HPC) via higher-level frameworks (Evangelos Maliaroudakis e Bilas 2022). We argue that the former approach induces significant overheads and lacks scalability, whereas the second has a great maintenance cost and cannot exploit resources or data sharing.

In this use case, we explore the viability of **"mini-Clouds" that can accommodate both Cloud and HPC workflows on the same hardware.** Such “mini-Clouds” should be easily deployable by HPC users (without root permissions), and provide a high-level of isolation among jobs of different users with negligible overheads. To achieve that, we environment an unprivileged, daemon-less layer that converts Kubernetes instructions to an HPC-compatible scheduler like Slurm. For the containerization, we have chosen the Singularity engine as it does not require escalated (root) privileges, as happens with other container engines like OpenVZ, Linux Container (LXC) or Docker.

Application Analysis

Generating efficient code for such a new architecture requires a good understanding of its performance features. In this deliverable, the systematic application analysis was focused on:

1. **Comparing the performance scalability on x86_64 and A64FX architectures**
2. **Revealing hidden patterns in the execution profile of the simulator.**
3. **Quantifying the performance overhead of containers against native execution.**

TABLE 17: SUMMARY OF INFRASTRUCTURES USED IN THE EXPERIMENTS

Platform	Processors	Memory	Compiler	Runtime
INAF	x86_64 (Xeon Gold 5118)	512 GB	gcc	Binary & Singularity
CEA	A64FX	32GB	gcc	binary
~~CEA~~	~~A64FX~~	~~32GB~~	~~fujitsu~~	~~binary~~

Infrastructure

For the evaluation, we used the following configurations:

- **Baseline:** measurements taken on the INAF cluster (x86_64 processors and 500GB DDR4 RAM) using GCC compiler version 11.2 and native execution.
- **Target 1:** measurements taken on the CEA cluster (A64FX processors and 32GB HMB RAM) using GCC compiler version 11.1 and native execution.
- **Target2:** measurements taken on the CEA cluster (A64FX processors and 32GB HMB RAM) using Fujitsu compiler and native execution. This target did not produce any valuable outcome because simulator was **consistently crashing** due to `*bus error*` and `*spmd_run_cmd killed*` error.
- **Target 3:** measurements taken on INAF cluster (x86_64 processors and 500GB DDR4 RAM) using GCC compiler version 11.2 and Singularity-based execution v3.10.4.

Experiments

In order to characterize the code's behaviour on the different target platforms, we have chosen to run simulations of different resolutions on cosmological boxes of different sizes, following the evolution of the Universe structure to the present time.

The following table summarizes the five simulation models (Cosmological Boxes) we used in the experiments.

- Boxes A to C have the same size and an increasing number of particles (and hence an increasing mass resolution).
- Boxes D and E have a larger size, two and four times, respectively, than box A, and are populated with eight times and 64 times more particles while retaining the same mass resolution.

TABLE 18: SUMMARY OF EVALUATED COSMOLOGICAL SIMULATIONS.

Box name	Box size	Nr. of particles	Mass resolution
A	*L*	32^3	*base*
B	*L*	64^3	*base* x 8
C	*L*	128^3	*base x 64*
D	2 x *L*	64^3	*base*
E	4 x *L*	128^3	*base*

For each box, we simulate its evolution with several different combination of «Nodes, MPI Tasks, OpenMP Threads», in order to assess:

the scaling behaviour of OpenMP for a fixed amount of tasks

i. the performance changes when the number of threads per MPI process varies
ii. the scalability of the code with the size of the problem and the amount of computational resources
iii. the fraction of run-time spent in different code regions

Errore. L'origine riferimento non è stata trovata. summarizes all the parameters and the rational for each combination we used. We point out to the reader's attention that the computational nodes on both INAF and CEA platforms have a remarkably similar configuration in terms of computational resources since both have 4 sockets (although the A64FX is physically a single socket it appears as it had 4 sockets and a 2-levels memory affinity matrix) with 12 cores each. Hence, or reference configuration aims to saturate the socket using 24 MPI tasks and 2 OpenMP threads.

TABLE 19: SUMMARY OF COMPUTE CONFIGURATIONS USED IN COSMOLOGICAL SIMULATIONS

Box	# Nodes	# MPI tasks	# OpenMP Threads per task	Goal
A	1	4	2 to 12	*To assess the scalability of OpenMP within a node*
A, B, C, D	1	24	2	*To assess the behaviour of the code under varying MPI/threads combination* *The 24x2 case is the reference case for the scalability study when using more than 1 node*
	1	12	4	
	1	6	8	
	1	4	12	
A, D	2	24	2	*To assess the impact of the network when more nodes are used with the same total computational resources*
	4	24	2	
A, B, D	2	48	2	*Together with the «1N, 24x2» to assess the strong scalability*
	4	96	2	

For the rest of the text, we will refer to the «Node, MPI, OpenMP» combinations as «nN, iT x jt» triplets (for instance: «1N, 24 x 2» translates as «1 Node, 24 MPI tasks x 2 OpenMP threads per task»).

Experimental Results

For brevity, we skip the installation details (e.g., transfer of dataset and path reconfiguration) and compilation efforts required to port OpenGadget3 from x86_64 to A64FX (e.g., compilation flags). Instead, we focus on gaining insights that will drive the co-design of OpenGadget3 in the context of EUPEX project.

Simulation Time Ratio between CEA/INAF platform

As a first-order approach, the overall execution time on different platforms is considered to initiate the benchmark campaign between Intel's x86_64 and Fujitsu's A64FX. Figure 23 shows execution times

normalized to the baseline experiments on the INAF cluster. We see that **CEA execution is consistently 3x to 6x times slower** than the baseline when the number of computational resources is kept fixed.

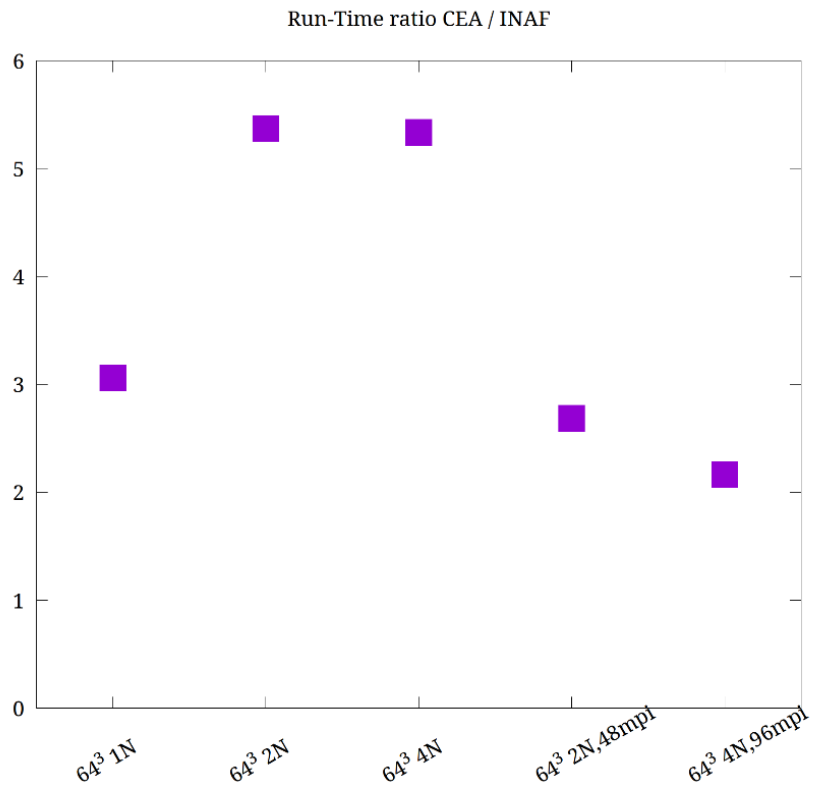


FIGURE 42: RUNTIME RATIO CEA/INAF. SIMULATIONS RAN ON CEA ARE CONSISTENTLY 3X TO 6X SLOWER THAN SIMULATIONS RAN IN INAF.

OpenMP Threads scalability

We assessed the multi-thread scalability by running boxes A and B with 4 MPI tasks and variable OpenMP threads in the range «2 ..12».

We have chosen to perform this test only for box A for two reasons. Firstly, the wall-clock time required to complete all six runs is limited compared to large boxes like B or C. Secondly, the small size of the test case allows the usage of an equally small number of MPI tasks and, consequently, to span the entire possible range of threads per task.

Although the very limited amount of memory with which the A64FX nodes are equipped on the CEA cluster is a severely limiting factor for larger runs, in this case the very large bandwidth and the significantly larger size of the cache lines play a fundamental role and confirm to be key factors for performance on the target architecture. Furthermore, we report that from our test allowing an MPI task to run threads outside of its socket is strongly discouraged by the degradation in memory bandwidth for inter-socket access.

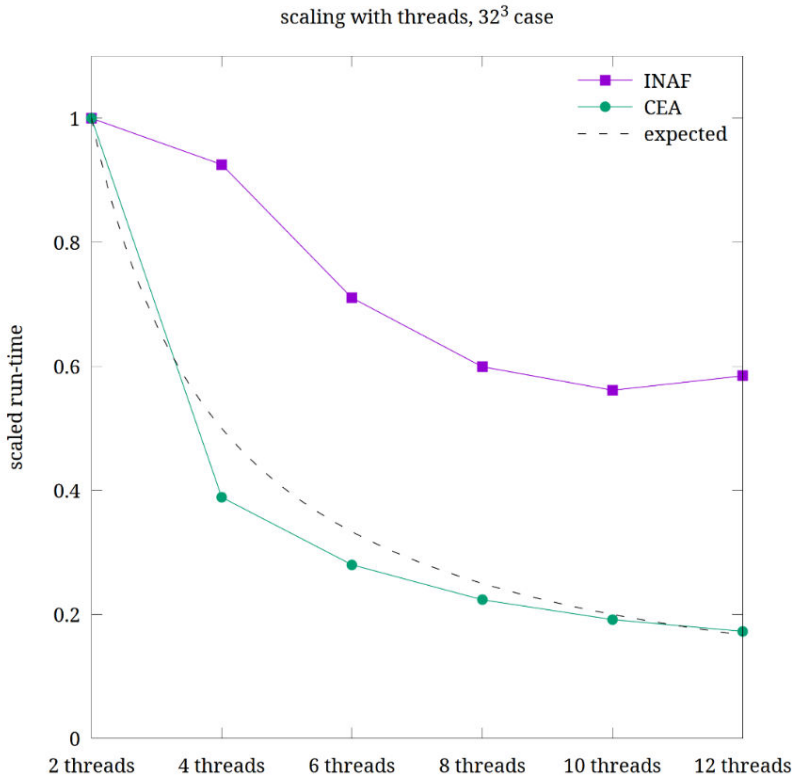


FIGURE 43: CLEARLY SHOWS THAT THE CODE SCALES PERFECTLY ON THE TARGET CEA PLATFORM, AT ODDS WITH THE BASELINE INAF CLUSTER WHERE THE THREADS SCALABILITY IS SIGNIFICANTLY POORER.

MPI Task scalability

In order to evaluate the strong scalability of the code, we run box C with 1283 particles (the largest resolution in our test suite) using 1, 2, and 4 nodes with 24, 48 and 96 total MPI tasks, respectively. The OpenMP threads are always 2 per task. In practical terms, this configuration decreases the workload per node while increasing the number of nodes. Since the number of particles per task scales down as $1/N_{nodes}$ the workload (the "pressure" on the memory and the computational load) scales down accordingly.

The target CEA cluster seems to respond significantly better to this strong scaling test. We attribute this behaviour to the frequent tree-walk operation of the simulator. Albeit the tasks are distributed, this operation implies a strong coupling among them, which according to Amdal's law, constraints the overall parallelization to the execution of the slowest task. Hence, the HBM memory of CEA nodes is crucial for improving the performance of the slowest task, and therefore supporting greater scalability of the tasks.

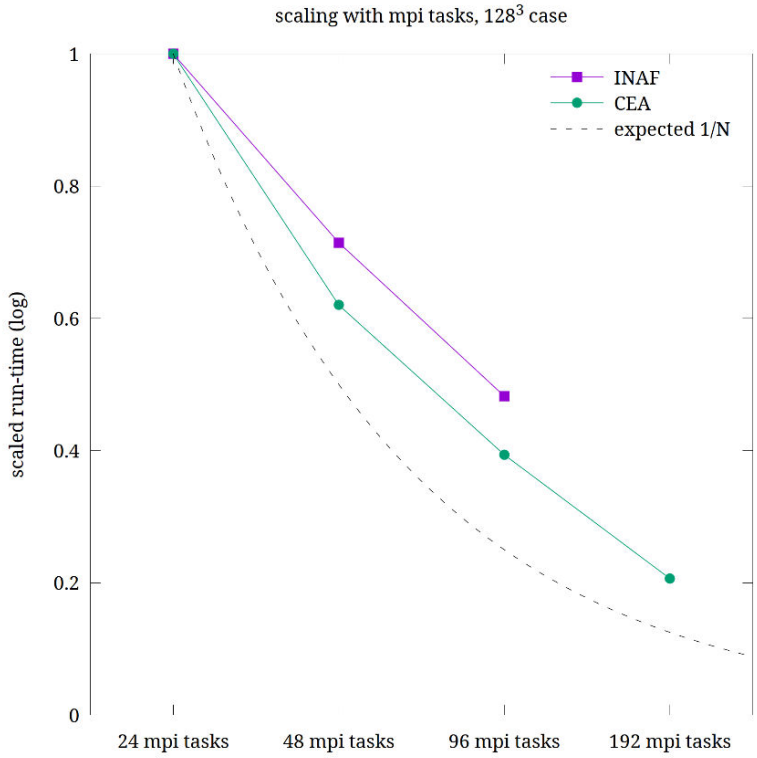


FIGURE 44: STRONG SCALING TEST FOR BOX C, USING FROM 1 U TO 8 NODES WITH 24 MPI TASKS PER NODE AND 2 OPENMP THREADS PER TASK.

Resolution study

At odds with the OpenMP experiment (323 box), where the memory performance was a dominant aspect, this test is shaped by the computation due to the larger number of particles involved.

On the left figure, we increase the workload per node, and since we have a single node, we increase the total workload. On the right figure, we keep the total workload constant, and by distributing it to four nodes, we decrease the workload per node. In all simulations, we used 24 MPI tasks per node and 2 OpenMP threads per task.

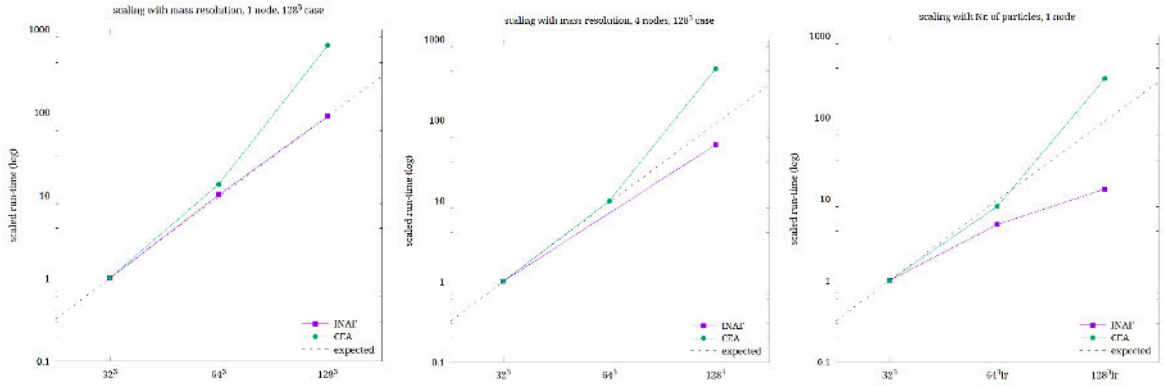


FIGURE 45: CODE SCALING WHEN THE MASS RESOLUTION INCREASES (BOXES A,B,C) USING 1 NODE (LEFT PANEL), AND 4 NODES (MIDDLE PANEL). IN THE RIGHT PANEL, THE MASS RESOLUTION IS KEPT FIXED, INCREASING BOTH THE NUMBER OF PARTICLES AND THE SIMULATION VOLUME (BOXES A,D,E) USING 1 NODE.

It must be signalled that the deviation from an otherwise perfect scaling in the CEA cluster (leftmost point in all the three plots, corresponding to 128^3 case) is due to a weird code behaviour that we are still inquiring. The anomaly is in the fact that, using exactly the same set-up for INAF and CEA as it was for all the runs, the simulation on the CEA cluster produced a much different time integration in that the chosen timesteps for the particles were much smaller than on INAF cluster. That caused a much larger number of code-steps to be computed (about 10 times) and hence a significant slowdown. At the moment of writing, we have not yet clarified this behaviour, and we suggest it may be due to some issue at code level, and to consider that the scaling will be as good as on the INAF cluster once it will be fixed.

Impact of threads per MPI task

We also studied how the run-time is affected when the number of OpenMP threads per MPI task changes while keeping fixed the total number of cores used. We simulated boxes A and B on a single node, using all 48 cores. MPI tasks range from 4 to 24, with careful mapping to sockets and binding of threads on the same sockets of their master process. The timings are normalized to our reference case, 24 MPI Tasks with 2 OpenMP threads per task (leftmost point in both plots).

From our preliminary tests, we infer that on A64FX architecture, **a process whose threads run on a different socket degrades the performance critically more than on our reference x86 architecture**. For this reason, we never place more than one MPI Task per socket in our tests.

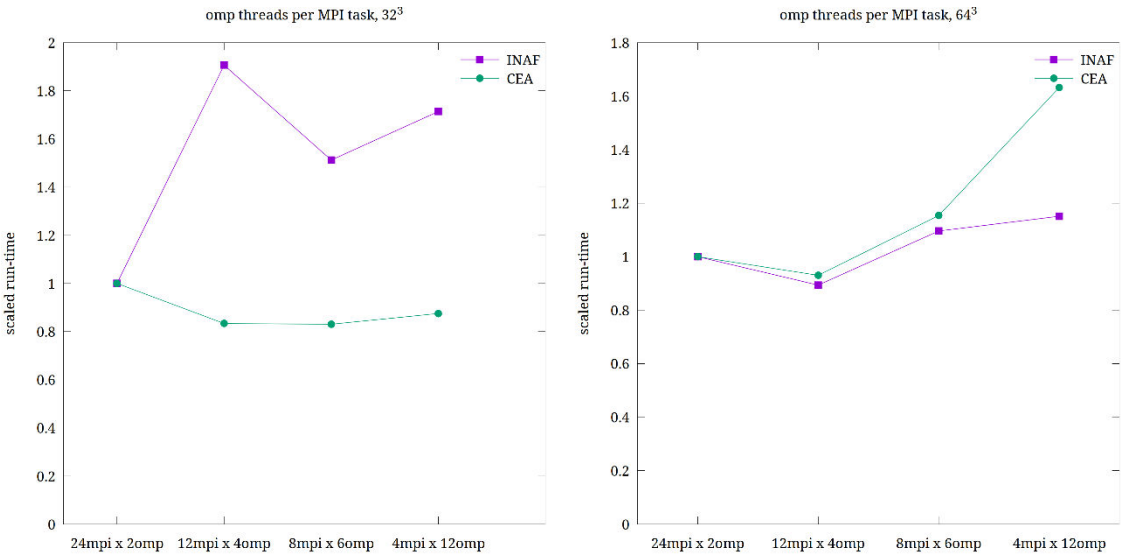


FIGURE 46: IMPACT OF DIFFERENT MPI/THREADS RATIO ON THE RUN-TIME. BOXES A (LEFT PANEL) AND B (RIGHT PANEL) ARE RUN ON A SINGLE NODE.

The OpenMP optimization of the code concerns loops over the particles (as depicted in previous Figure 23) that are both compute intensive and memory intensive and most of the time they are related to neighbours search around the position of a target particle.
The C structures that host particle data are way too large to be optimal for memory traversal, and in fact the code copies the relevant data for the current loop on separated temporary structures that are communicated back to the MPI task that originated the query. Moreover, the neighbour search in more than 1 dimension obviously involves an out-of-order memory traversal that is random with respect to any possible order in memory.
These facts help in the interpretation of the results plotted in Figure 30. The Left Panel, that refers to a smaller problem where the memory boundness is accentuated, shows that the A64FX memory system is more able to cope with the memory access and serves more effectively a large number of threads that share the memory bandwidth of the memory banks connected to they run on.

Impact of the network on the run-time

In this experiment, we study the simulation run-time when distributing the same total number of cores over different nodes. All the runs have been performed with 24 MPI Tasks and 2 OpenMP threads per task. Timings are normalized to the timing on one node.

We noticed that **the difference comes primarily from the gravity and density functions, which are intensive in tree walking, and therefore memory bound.** Also, the network effect, measured by communication time, is significant in relative terms (it increases from 30 to 50%) but negligible in absolute terms being less than 10%.

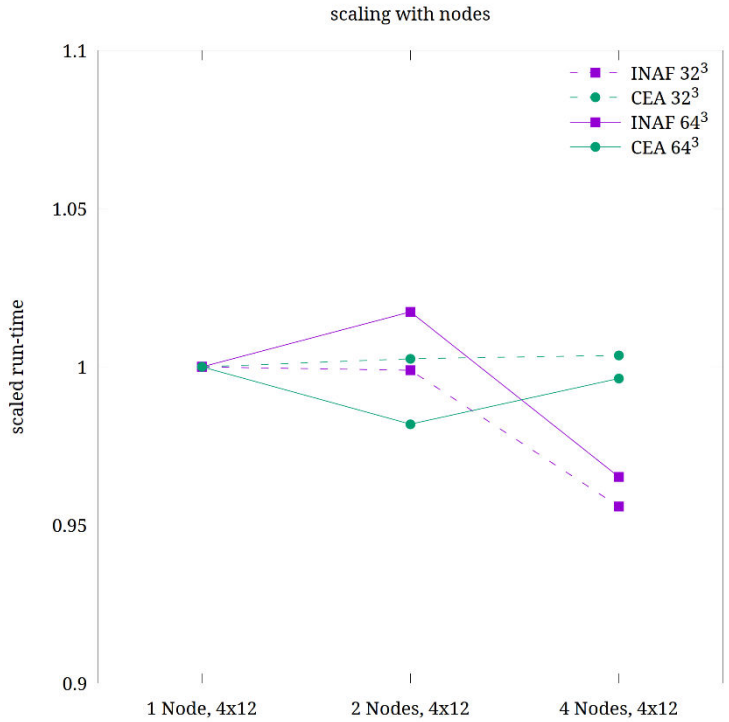


FIGURE 47: IMPACT ON RUN-TIME OF DISTRIBUTING THE SAME TOTAL AMOUNT OF CORES OVER A DIFFERENT NUMBER OF NODES, FROM 1 TO 4. VIOLET LINES ARE FOR OUR BASELINE REFERENCE (THE X86 CLUSTER) WHEREAS GREEN LINES REFER TO THE A64FX TARGET PLATFORM.

Phase decomposition for different compute schemes.

Motivated by the previous experiment, we deemed it necessary to understand the ratio at which each phase contributes to the overall simulation since every phase has unique performance characteristics. In this experiment, we keep the problem of the simulation constant «box64» and investigate various compute configuration schemes «1N, 24T», «2N, 12T», «2N, 24T», «4N, 6T», «4N, 24T».

For all practical purposes, the phase contributions remain constant across configurations. However, a notable exception is Friends-of-Friends (FOF), whose ratio appears to increase as the calculation becomes more parallel and distributed. This trend appears in both INAF and CEA clusters, but the rate of change is great in the INAF cluster.

By combining the knowledge that FoF is a memory-bound operation and that CEA nodes are equipped with High Bandwidth Memory (HBM), whereas INAF nodes have modest DDRs, we can infer the importance of HBM on clustering algorithms.

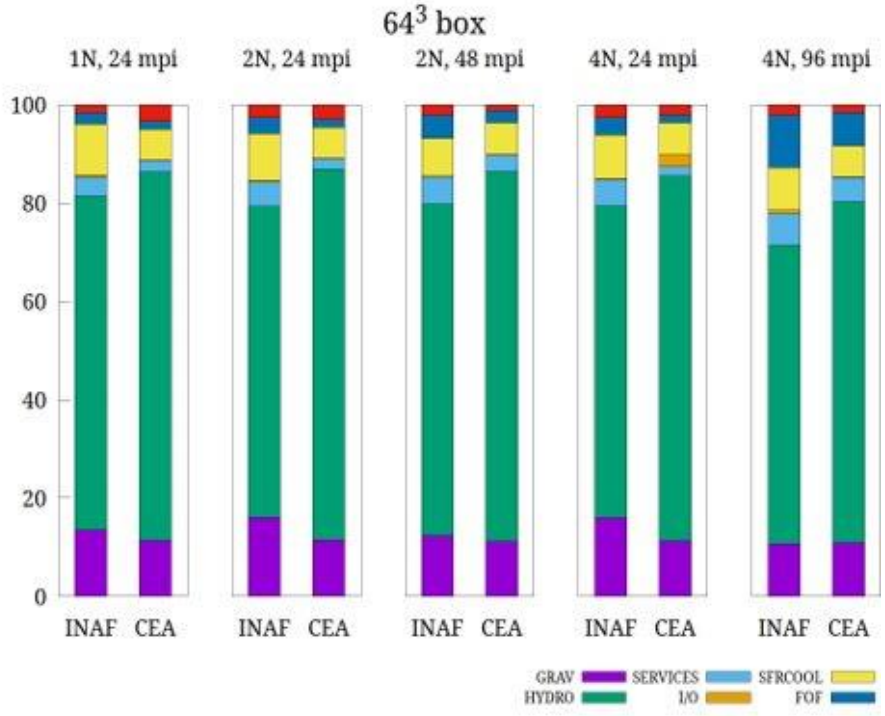


FIGURE 48: ANALYSIS OF PHASE DECOMPOSITION FOR DIFFERENT COMPUTE DISTRIBUTION SCHEMES NOTABLY, THE RATIO OF MEMORY-BOUND FRIENDS-OF-FRIENDS PHASE, APPEAR TO BE INCREASING AS THE SIMULATION BECOMES MORE DISTRIBUTED (IN TERMS OF INCREASING MPI TASKS AND NODES)

Phase decomposition for different problem sizes.

Understanding which phases contribute more (or less) when the simulation scales up should be critical when considering which functions should be profiled in greater detail and optimized more aggressively. To do so, we keep the compute configuration constant «1N, 24x2» and increase the problem size of the simulation «box32, box64, box128».

The following image presents such a simulation breakdown, where the percentage of CPU utilization is examined per phase and subphases.

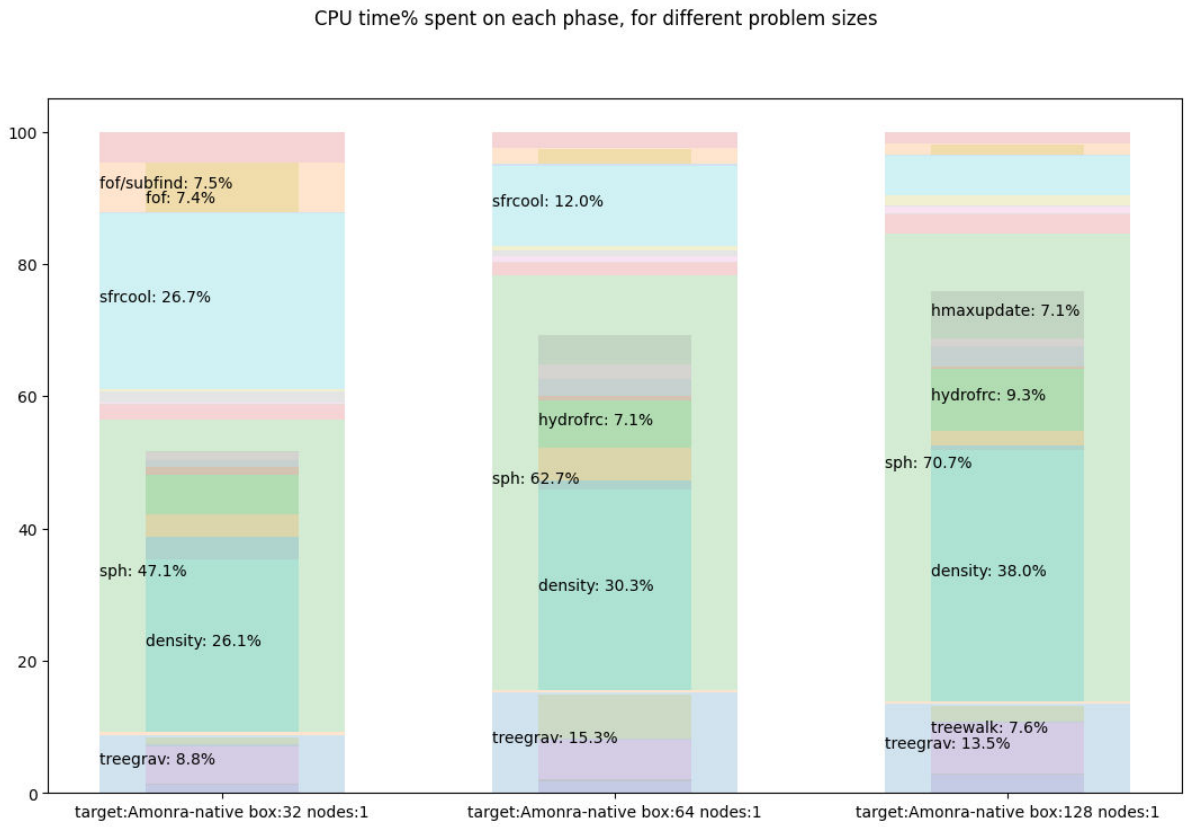


FIGURE 49: ANALYSIS OF PHASE DECOMPOSITION FOR DIFFERENT PROBLEM SIZES. THE CPU UTILIZATION PROFILE (RATIO PER PHASE) CHANGES 50RAPIDLY AS THE PROBLEM SCALES.

Temporal variations within the same run

To better understand the temporal effects of simulation phases, we study the execution time for each time step of a Gadget simulation for two indicative phases and the total real time of the time step, measured by the CPU occupancy of the time step.

In this particular configuration, with a rolling mean of 100-timesteps window, it can be shown that throughout the execution, the simulation performance changes drastically. Besides the initial timesteps, there are large (in terms of timesteps) regions with different average execution times, producing a step-like pattern. On the finer scale, a pulse-like pattern appears. This observation shows that the average execution time for a phase can be a misleading metric, and that profiling/optimization campaigns should focus on the timesteps that are more intensive.

Having such a detailed view of the temporal variations of the performance both in total and per phase allows focussing on the most intensive regions of the simulation both in spatial (regions of code) and temporal (timesteps) domains.

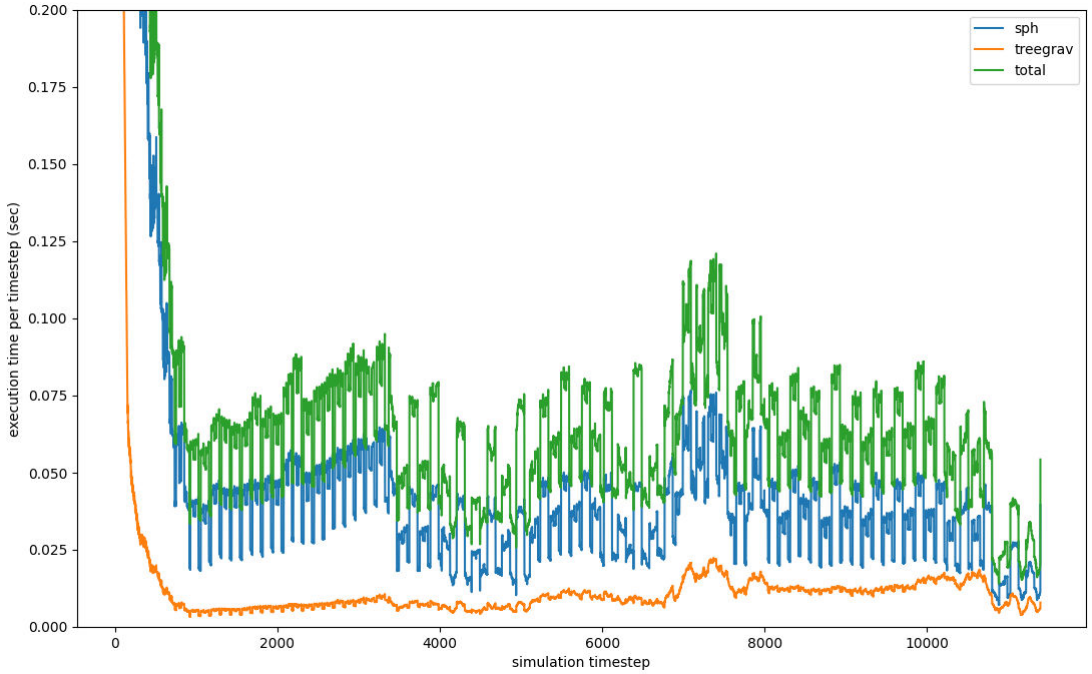


FIGURE 51: TIME ANALYSIS OF THE MOST CPU-INTENSIVE PHASES. WE SEE THAT FOR THE MOST PART, CPU UTILIZATION IS NOT STATIC, BUT IS RATHER DOMINATED BY PEAKS AND VALLEYS.

Containerization with Singularity incurs negligible overhead.

Additionally, we compare the results of running the OpenGadget3 simulator as a native binary and as a container. As it is shown in the following figure, the overhead of Singularity runtime is negligible compared to the native execution. The findings are consistent with more elaborate studies conducted with microbenchmarks (Hu, Zhang e Chen 2019).

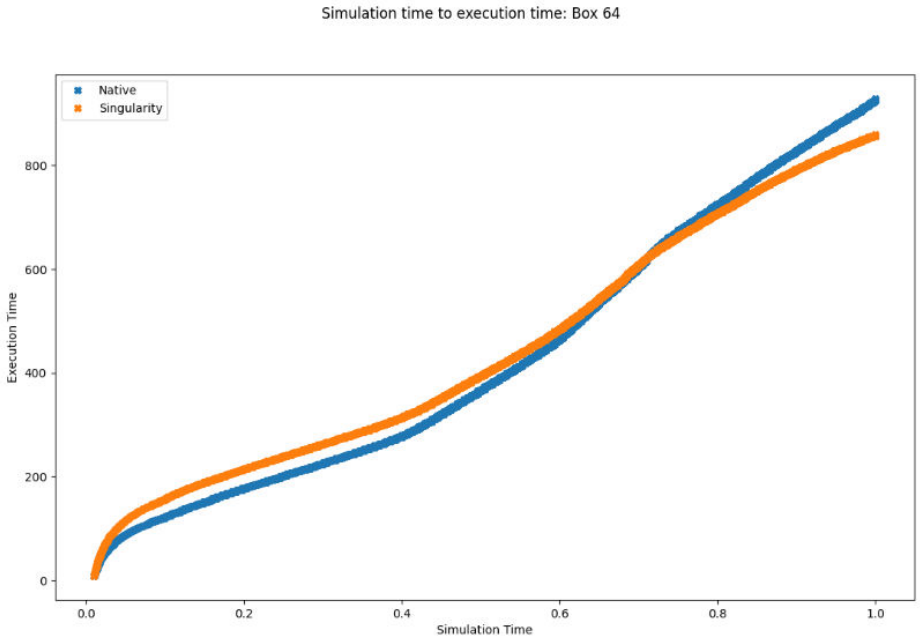


FIGURE 52: TIME ANALYSIS BETWEEN NATIVE AND CONTAINERIZED EXECUTION

Conclusion of the Analysis

We evaluate the OpenGadget3 simulator on the A64FX with 32 GB HBM nodes on CEA cluster, and on x86_64 with 512 GB RAM nodes on INAF cluster. From the evaluation we draw several insights for further optimization. Firstly, nodes with HBM memory can improve the simulation scalability, both at the OpenMP threads and MPI Tasks levels. However, the deep hierarchies of A64fx require the placement of Threads and Tasks to be done carefully with respect to NUMA effects, otherwise the performance penalties are much greater than performance gains of HMB.

Additionally, we were able to isolate and measure the performance characteristics of the simulated processes. We identified not only the performance profiles of different processes vary significantly, (cpu-bound, memory-bound, etc), but also that the performance profile of a single process changes as the simulation progresses. The interplay of these factors points to a collocation strategy that accounts the temporal and localized behaviour of processes, in order to optimize the overall execution time.

Finally, the fact that containers induce negligible overheads, enables us not only to containerize the workflow stages before and after the simulation (e.g., data preparation, data analysis), but also the simulator itself. In turn, this means that we can build entire workflows for the compilation, execution, analysis, and archival of the simulator, and make the workflow portable both across HPC infrastructures and to the Cloud.

Mini Application Analysis

Bolt65

Statistics show that global video IP traffic is approximately 82 percent of all consumer internet traffic in 2022, up from 75 percent in 2017. Annual global IP traffic reaches 4.8 Zettabytes (1 ZB = 1000 Exabytes) in 2022, meaning that approximately 3.9 ZB of video content crosses the network in 2022 (V. N. Cisco 2017). Handling this enormous amount of data is a very challenging task for video content providers in years to come. Another factor that highlights this problem is the fact that the number of different devices that are able to play video content is constantly growing. Globally, devices and connections are growing faster (10 percent CAGR) than both the population (1.0 percent CAGR) and the Internet users (6 percent CAGR). Each year, various new devices in different form factors with increased capabilities and intelligence are introduced and adopted in the market (U. Cisco 2020). These are some of the facts that demonstrate the massive size of video data that is generated and transferred through the Internet daily.

FIGURE 53: INTERNET VIDEO TRAFFIC SHARE PER YEAR, RIGHT - GLOBAL DEVICE AND CONNECTION GROWTH

The performance growth of the technology throughout the years not only enabled the viewing of multimedia content on a variety of devices but also allowed the introduction of High Definition (HD) and Ultra-high Definition (UHD) videos.

The performance growth, video quality enhancements, and huge video traffic are also driving forces behind the development of novel video encoding standards which present an efficient way to compress video data. Storing or transferring uncompressed video data is an unfeasible process. Just one second of uncompressed High-definition video with a spatial resolution of 1920x1080 and temporal resolution of 60 frames per second stored in a 4:4:4 chroma subsampling scheme would take 373.248 MB. Taking this fact into account, it is easy to conclude that efficient video compression is mandatory. Video compression was a fertile ground for research and development from the late ’80s when the first video coding standards that were useful in practical terms were published. The evolution of video compression algorithms leads to one of the latest novel standards called H.265/HEVC (Sullivan, et al. 2012) which is considered one of the most efficient and commonly used video coding standards.

The Bolt65 is one of the applications developed by UNIZG that tries to resolve some of the issues presented above by improving the performance of the video encoding, decoding, and transcoding processes based on the HEVC standard.

Application Description

Bolt65 is a performance-optimized HEVC hardware/software suite for Just-in-Time video processing developed as a part of the research activities of the HPC Architecture and Application Research Group at the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia (Piljić, et al. 2019). Bolt65 is a “clean-room” suite that consists of an encoder, decoder, and transcoder based on HEVC standard. The special focus in the development of the Bolt65 is set on the performance efficiency achieved by low-level optimizations and hardware-software co-design adapted for the efficient exploitation of heterogeneous accelerator-based architectures. Another important focus of Bolt65 is the just-in-time processing requirement which sets constraints on processing time making Bolt65 suitable for encoding/transcoding on demand.

Bolt65 suite is written from scratch in C++, without any external library dependencies. It has successfully been tested and ported to different platforms, from CPU-only architectures to heterogeneous architectures that combine different processing cores (CPU, GPU, customized FPGA-base hardware accelerators), with different compilers (MSVC, g++, armclang++, FCC, Intel C++, …), and on different operating systems (Windows, Linux).

The rest of this chapter will give a brief overview of the HEVC standard and the high-level schemes of HEVC-based encoder, decoder, and transcoder, before describing Bolt65 and its features in more detail.

HEVC overview

High Efficiency Video Coding (HEVC or H.265) is a video compression standard developed by the Joint Collaborative Team on Video Coding (JCT-VC), a collaboration between the Video Coding Experts Group (VCEG) and ITU Telecommunication Standardization Sector (ITU-T) (Sullivan, et al. 2012). The HEVC standard enables major advances in compression relative to its predecessors, such as Advanced Video Coding (AVC) or MPEG-2, doubling the compression rate of encoded bitstream compared to AVC without sacrificing quality. This compression efficiency of the HEVC standard is not accomplished with a single novel compression technique but is a result of multiple contributions in all stages of the encoding process. However, increased compression efficiency comes at the cost of increased computational complexity and resource requirements by up to 10 times (Bossen, et al. 2012).

HEVC encoder

In the process of video encoding, a raw uncompressed video signal is encoded to the final bitstream using a set of syntax and semantic rules specified by a certain standard, in this case, H.265/HEVC. The main goal of the encoder is to compress a huge amount of data that is contained in an uncompressed signal. Figure 54 shows a high-level scheme of an encoder, where FullHD (1080p@60fps) video sequence is encoded from a raw signal that contains 178MB in one second of the video to HEVC encoded bitstream that can contain about 2MB of data per second, which depends on the implementation and configuration of the encoder. Video compression is usually lossy, meaning that some information is lost during the process, resulting in lower video quality.

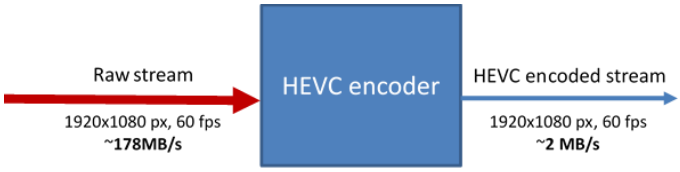


FIGURE 54: HEVC ENCODER

HEVC decoder

The decoder is a module that transforms encoded bitstream to a raw video stream that is later displayed or re-encoded to a different format. In order to display HEVC-encoded video, devices have to be capable to decode HEVC standard. Although decoders are computationally much less expensive, their efficient implementation is also the subject of many research activities. Figure 55 shows a high-level scheme of the HEVC decoder that decodes HEVC bitstream to a raw video signal.

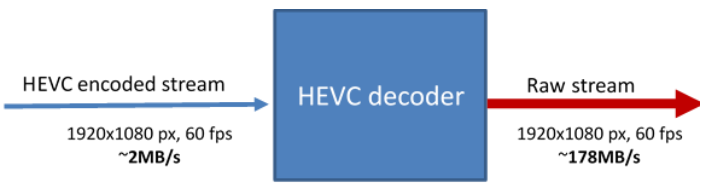


FIGURE 55: HEVC DECODER

HEVC transcoder

Video transcoding is the operation of converting a video from one format to another. A format is defined by such characteristics as the bit rate, frame rate, spatial resolution, coding syntax, and content. The transcoder is achieved as a cascade of a decoder, followed by an encoder. Input for the transcoder is an encoded video bitstream, which is then decoded and re-encoded to obtain transcoded bitstream. During the transcoding process, some of the encoder parameters can be changed, depending on the requirements and the desired end video stream properties (i.e., increasing quantization parameter, reducing the spatial or temporal resolution to achieve bitrate reduction). Figure 56 shows the high-level scheme of the transcoder module.

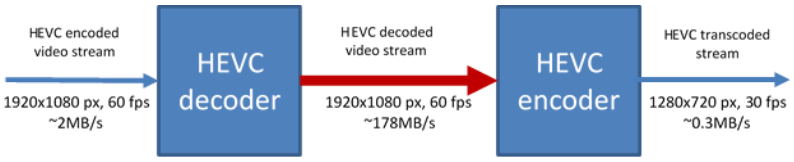


FIGURE 56: HEVC TRANSCODER

Bolt65 overview

The scheme of the Bolt65 video transcoder, which concatenates the Bolt65 decoder and encoder, is shown in Figure 57. The input to the transcoder is HEVC-compliant bitstream which is decoded in the first phase of the process. The decoded raw video data is forwarded to the Advanced transcoding module, which reuses some of the data obtained from the original bitstream to enhance and facilitate the process of the re-encoding that is being performed in the second phase of the transcoding process. The algorithms used in the transcoding module depend on the application's developer and can differ depending on the configuration of the transcoder.

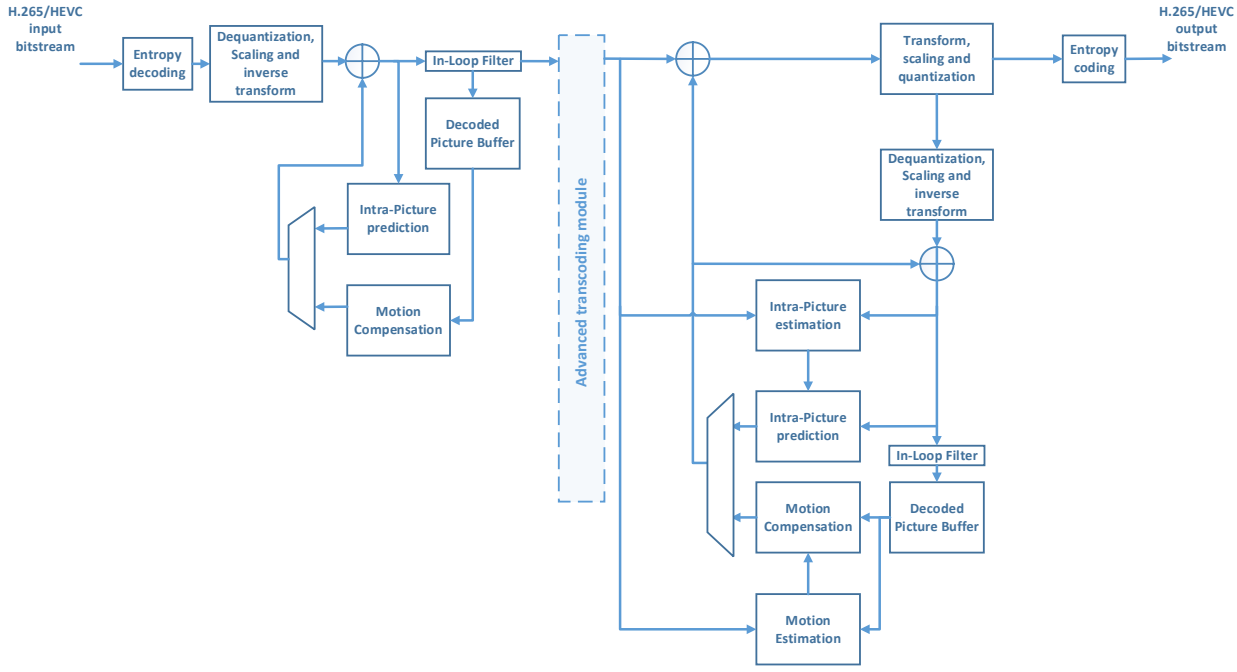


FIGURE 57: BOLT65 TRANSCODER SCHEME

Inputs and outputs

Input to the Bolt65 encoder is a raw video stream that is being encoded to the final HEVC bitstream. Two types of files that represent raw video streams are supported by Bolt65: YUV and Y4M. Both of these types contain raw pixel data, with the difference that the Y4M file also contains a header with additional information about the video. In the Bolt65 decoder, input and output are reversed compared

with the encoder, while the transcoder, both receives and outputs HEVC bitstreams, using raw pixel data only internally.

Raw video data with 8 bits per sample and the representation with luma brightness signal and two chroma channels that have half the luma resolution both horizontally and vertically (color space 4:2:0) is supported by Bolt65.

Block structures

The first step after fetching the frame from the input video sequence is to divide the frame into smaller square-shaped blocks called Coding Tree Units (CTU) (Kim, et al. 2012). A CTU represents a basic processing unit in HEVC and all future operations in the encoding process are based on CTU. CTU can be split into smaller Coding Units (CU) of variable sizes, with a minimum CU size of 8x8 and a maximum of 64x64. Each coding unit consists of precisely three Coding Blocks (CB), one luma block, and two corresponding chroma blocks. Dividing CTU into multiple smaller CUs follows the quadtree structure as shown in Figure 58.

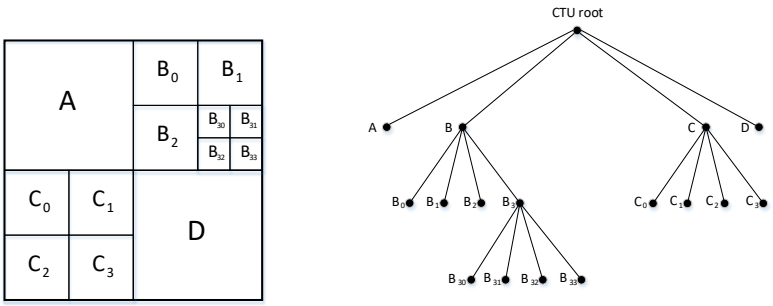


FIGURE 58: EXAMPLE OF PARTITIONING CTU TO SMALLER CUS

Each of the CUs can be furtherly split into Prediction Units (PU) used in the prediction process, and/or into Transform Units (TU) used in the transformation phase.

Bolt65 supports CTU sizes of NxN, where N∈{16,32,64}, resulting in luma CTB size of NxN and chroma CTB sizes of (N/2xN/2) due to 4:2:0 color subsampling. CTUs can be divided into four smaller units CUs following a quadtree structure. CU supported by Bolt65 are NxN, with $N_{min} \leq N \leq N_{max}$ and N∈{8,16,32}. N_{min} and N_{max} can be defined in the configuration. Leaf CUs can also be split into up to four PUs that can be used for more precise motion estimation. HEVC standard supports 8 partitioning modes for splitting CU to PU: MxM, Mx(M/2), (M/2)xM, (M/2)x(M/2), Mx(M/4), Mx(3M/4), (M/4)xM and (3M/4)xM, all of which are also supported in Bolt65 (Figure 59). The coding unit with minimal CU size defined in the configuration can also be split into four TUs.

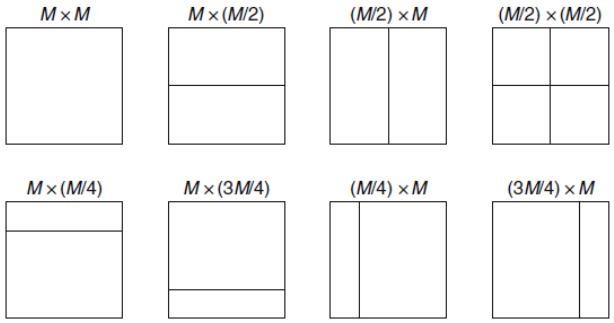


FIGURE 59: SPLITTING CU INTO PUS

Prediction

Each of the leaf CUs can be partitioned into one or more prediction units (PU) that are predicted by taking advantage of spatial (Intra-prediction) (Lainema, et al. 2012) or temporal (Inter-prediction) (Ugur, et al. 2013) dependency of video frames. Prediction output represents a block of predicted samples

which is then subtracted from the original block to form a residual (i.e., prediction errors) used as an input in the following steps of the encoder.

Whether the frame is intra or inter-encoded depends on the structure of the Group of Pictures (GOP). GOP is a collection of successive pictures within a coded video stream and it specifies the order in which intra and inter-frames are arranged.

Intra-prediction uses neighbouring pixels from adjacent coding blocks to calculate residual signal. There are 35 different modes used to intra-predict block; 33 angular predictions and two non-directional predictions (DC and Planar). To get the best possible residual signal, all 35 modes are appraised and the one with the smallest prediction error is chosen. Several algorithms for determining prediction errors are available in Bolt65: Sum of absolute differences (SAD), Sum of absolute transform differences (SATD), and Mean square error (MSE) and it is up to the user to choose which of these will be used.

Predicting the current frame based on the previously decoded frames is known as Inter prediction and the process of determining motion between the current and referent block is called motion estimation. The result of motion estimation is a motion vector that represents the direction of movement for each block in quarter-pixel accuracy for the luma component and eighth-pixel accuracy for chroma components. If the motion vector has half or quarter-pixel accuracy, interpolation must be performed for pixels at fractional positions. The process of motion estimation is depicted in Figure 60.

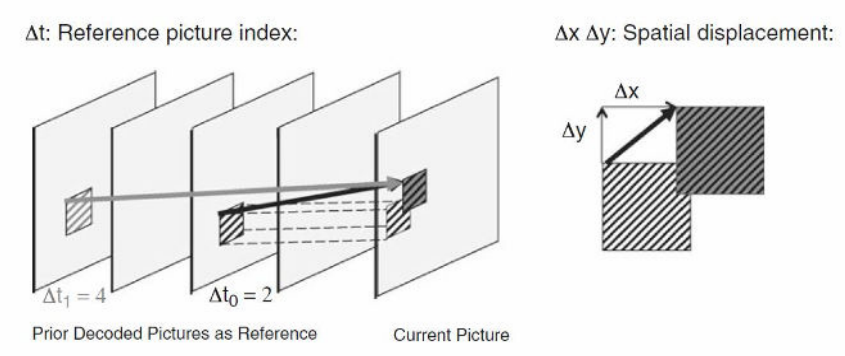


FIGURE 60: INTER PICTURE PREDICTION

The process of motion estimation is one of the most complex parts of the HEVC algorithm and depending on the implementation of the algorithm used, motion estimation can consume a significant amount of overall time in the application lifecycle (Bossen, et al. 2012). The user can control the motion estimation process in the Bolt65 encoder part by defining several parameters in the configuration: search algorithm (Full search, Three step search….), search area, and block matching criteria (SAD, SATD or MSE).

Transform and quantization

The transform is applied to the residual signal resulting from the prediction. Each TU residual block is input to a two-dimensional NxN forward transform, which is a separable operation that can be also performed as two one-dimensional transforms for each row and column. The resulting transform coefficients are then quantized (i.e., divided with the quantization step - Qstep) to obtain quantized transform coefficients that are used as input to the entropy encoder. To retrieve the reconstructed frame that is stored in Decoded Picture Buffer (DBP) for future inter predictions, each block has to be de-quantized and inverse transformed. This process in the encoder and decoder is shown in Figure 61.

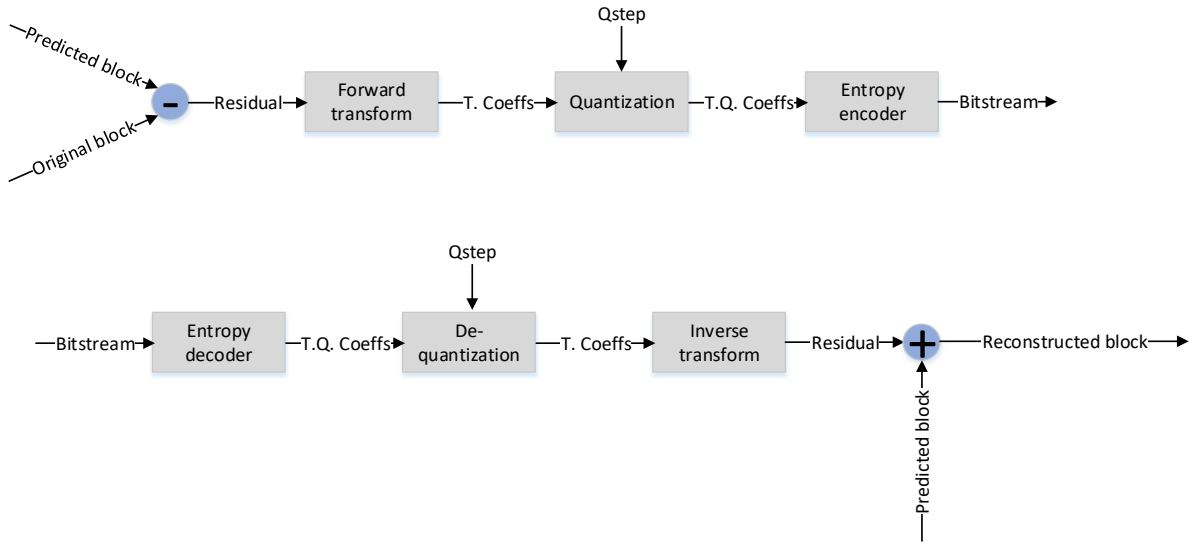


FIGURE 61: TRANSFORM AND QUANTIZATION PROCESS IN BOLT65 ENCODER AND DECODER

For transformation purposes, Discrete Cosine Transform (DCT) is used for most of the blocks in the HEVC standard. In the original form, the DCT uses floating point operation which increases computational complexity and errors between the forward and inverse transform. Therefore, HEVC specifies a two-dimensional finite precision integer approximation of the DCT transform, referred to as the core transform. The core transform specifies the kernel matrices for each block size, designed to enable efficient implementation in both, software and hardware. To achieve more optimal de-correlation of the residual input block, HEVC also specifies alternate transform based on Discrete Sine Transform (DST), which is used exclusively for 4x4 luma blocks.

A quantization process is performed based on the quantization parameter (QP) set as an input to the encoder. Depending on the QP, which can be in the range from 0 to 51 inclusive, for 8-bit pixel samples, Qstep is calculated.

Entropy coding

The quantized transform coefficients obtained after transform and quantization are entropy-coded using context-adaptive binary arithmetic coding (CABAC) (Auyeung, et al. 2011). CABAC is a lossless compression scheme that uses statistical properties to compress data. Entropy coding is the only part of the algorithm that cannot be parallelized.

In-loop filters

There are two types of In-loop filters in the HEVC standard: Deblocking filter (Norkin, et al. 2012) and the Sample adaptive offset (SAO) (Fu, et al. 2011). Both of them are applied in the encoding and decoding loops, first deblocking filter if enabled and then SAO, before storing the frame in DPB. The goal of In-loop filters is to increase the subjective quality of reconstructed pictures which also leads to higher compression efficiency.

In Bolt65, only deblocking filter is implemented. The purpose of the deblocking filter is to smooth possible artifacts at the boundaries of adjacent coding blocks. An example of using the deblocking filter is shown in Figure 62., where pixels A0 – A3 belong to one line in one coding block, while pixels B0 – B1 belong to a corresponding line of another. The dotted line shows pixels after deblocking filter is applied. Deblocking filter is applied on the vertical and horizontal boundaries of all coding blocks in the frame.

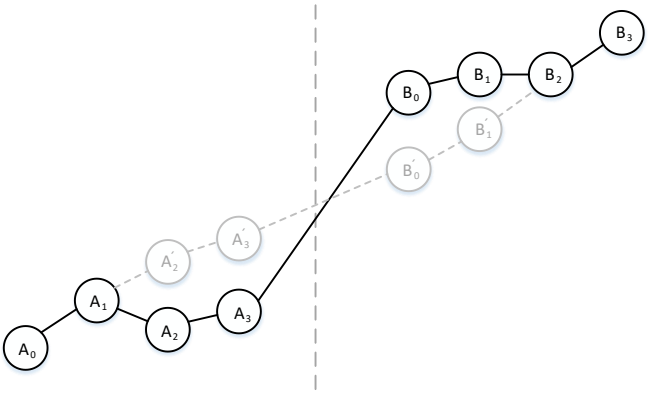


FIGURE 62: USING DEBLOCKING FILTER AT THE BLOCK BOUNDARY

Using deblocking filter increases the quality of the encoded video and coding efficiency, but at the same time introduces additional operation that has to be performed. Although the deblocking filter is not as complex as some other operations, it can affect satisfying strict timing requirements that can be set in some real-time systems.

Parallelization

HEVC standard introduces two novel parallelization concepts, Tile and Wavefront parallel processing, along with the slices, which were also available in previous AVC standard. Tiles, performance-wise, outperform other parallelization concepts in HEVC (Misra, et al. 2013), so in the Bolt65 the focus was set on efficient implementation and usage of this mechanism.

The frame can be divided into rectangular-shaped groups of CTUs separated by vertical and horizontal boundaries. Those groups are called tiles and they can be processed independently, without need for the communication between processing units that process different tiles. This fact makes tiles an ideal tool for multicore processing. Dividing frame to multiple tiles can be done uniformly, where each row and column of tiles has approximately the same number of CTUs, or non-uniformly, where the number of CTUs in each row or column is specifically defined. An example of a frame that is split into uniform 3x3 tiles is shown in Figure 63.

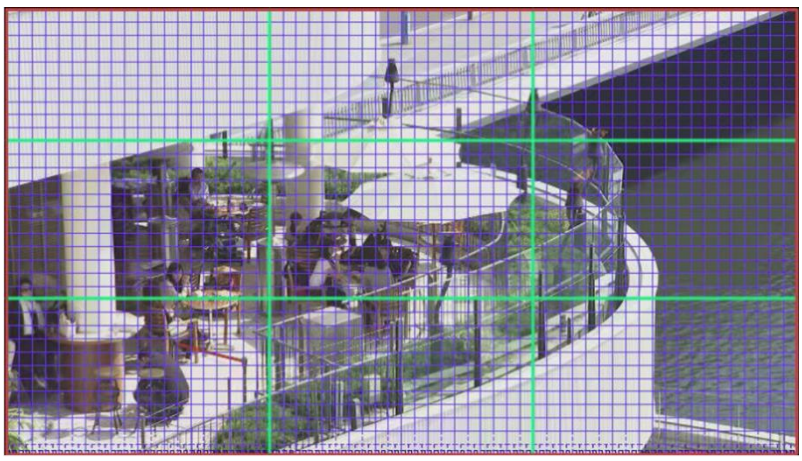


FIGURE 63: FRAME SPLIT TO 3X3 TILES UNIFORMLY

Configuration

Configuration for Bolt65 encoder, decoder, and transcoder can be defined in two ways before running the application: through the configuration file which contains all necessary data, or through the console, where each parameter can be set individually. The two options can be combined so that some parameters are set in the configuration file, while others are set as console parameters. If the same option is

present in both, the configuration file and as a console parameter, the option set as a console parameter will override the option set in the configuration file.

Application Use Cases

Two main use cases will be used to evaluate the EUPEX platform: Bolt65 encoder and transcoder. The Bolt65 decoder is omitted from experiments since it is computationally much less expensive than the encoder and transcoder, and is also a basic element of the transcoder.

A Uniform set of test conditions and benchmarks are defined to conduct experiments in a well-defined environment and to ease evaluation of the application performance per experiment. We define two different application-specific benchmarks which are based on real-world use cases of the video processing applications:

- ***B***$_{b65\text{-}h}$ *– high computational complexity benchmark*
- ***B***$_{b65\text{-}l}$ *– low computational complexity benchmark*

B$_{b65\text{-}h}$ is a high-complexity benchmark that corresponds to real-life use cases in which high-quality videos are pre-encoded and stored on the server. This use case excels in compression efficiency and video quality but on the other hand, the processing time is prolonged due to increased computational complexity. This increase is mainly caused by an in-depth mode decision algorithm that tests a vast number of candidates before it finds the optimal one. This means that for each block, the same kernels are executed over and over for each of the probable candidates. The complexity increase is also caused by a lower quantization parameter which leads to a bigger number of non-zero matrices that must be processed.

B$_{b65\text{-}l}$ is a low-complexity benchmark that represents a real-life use case in which videos are being encoded just-in-time before streaming to the user. This encoding approach is mostly used in video conferencing where the video captured from the camera is encoded on the fly and streamed to the other users participating in a video call. One of the other use cases is just-in-time video transcoding in which a video stream is decoded and then re-encoded to occasion-specific properties that can be constrained with network resources or decoding capabilities of the device accessing the media. ***B***$_{b65\text{-}l}$ has lower processing complexity because of the simpler mode decision algorithm which tests a smaller number of candidates.

The following subchapters define video sequences selected for the experiments, the encoding configuration used in ***B***$_{b65\text{-}h}$ and ***B***$_{b65\text{-}l}$ benchmarks, and the parallelization options for the Bolt65.

Video sequences

For each of the benchmarks, the same test set of video sequences is used. Video sequences are ranging in spatial resolutions and frame rates, as shown in Table 20. The set covers spatial resolutions, from small video sequences with resolutions as low as 176x144 to high-definition (HD) video sequences with a resolution of 1920x1080. A wide spectre of different video sequences enables experiments that can test a final platform in different manners. Encoding or transcoding the HD video sequence puts more stress on the memory system, while for lower-resolution videos, most of the runtime is devoted to computational complexity. Video sequences also differ in the amount of motion in the consecutive frames. The amount of motion is correlated with the processing complexity since more motion makes the search for good reference candidates more complex. All of the test video sequences are widely used for video processing benchmarks and are available for download (Aalborg University s.d.).

TABLE 20: SELECTED TEST VIDEO SEQUENCES

#	Video name	Resolution	Frames	Frame rate
1	Akiyo	176x144	300	30
2	Mobile	352x288	300	30
3	Soccer	704x576	600	60
4	City	704x576	600	60
5	Johnny	1280x720	600	50

6	ParkRun	1280x720	500	60
7	Shields	1280x720	500	60
8	BasketballDrive	1920x1080	500	50
9	BlueSky	1920x1080	217	25

Configurations

The computational complexity of benchmarks $B_{b65\text{-}h}$ and $B_{b65\text{-}l}$ are regulated by parameters controlled by the user when running the application. Table 21 shows defined benchmarks and basic configuration for each benchmark, highlighting the differences in encoding parameters.

TABLE 21: BENCHMARK CONFIGURATION

Benchmark	Configuration
$B_{b65\text{-}h}$	QP: 22 GOP structure: Random Access Prediction: Both INTRA and INTER prediction is tested in P frames Symmetric motion partitioning: enabled Asymmetric motion partitioning: enabled Deblocking filter: enabled
$B_{b65\text{-}l}$	QP: 32 GOP structure: Random Access Prediction: Only INTER prediction is tested in P frames Symmetric motion partitioning: disabled Asymmetric motion partitioning: disabled Deblocking filter: disabled

Parallelization

The basic benchmarking will be performed on a single processing core to measure the pure performance benefits without the effects of the overhead of communication between different cores. The multi-core benchmarking will take advantage of HEVC tiles (as described in Figure 63: Frame split to 3x3 tiles uniformly, parallelization subsection), where each tile will be processed on a separate processing unit.

Results and Analysis

Bolt65 encoder is used for the initial benchmarking, profiling, and analysis of the application, since it is computationally much more expensive than the decoder, while the transcoder is the cascade of the encoder, followed by the encoder. Therefore, all the conclusions obtained by profiling the encoder will be valid for the transcoder.

The application is first profiled on a single node and a single thread. Due to the characteristics of parallelization techniques in the HEVC standard (see Parallelization section**Errore. L'origine riferimento non è stata trovata.**), the main conclusions can be mapped to the multi-threaded implementation, since each core is performing the same processes, but on different parts of the frame. To measure the initial thread scalability of the Bolt65 encoder, we employ the HEVC tiles mechanism by splitting the frame into a different number of tiles, depending on the number of threads used.

Execution time

To measure the pure execution time, we ran a test on all video sequences (Table 23), with both configurations (Table 21**Errore. L'origine riferimento non è stata trovata.**). To measure the influence of different compilers, we also tested two available compilers: ARM's armclang++ and Fujitsu's FCC. Auto-vectorization options were disabled for both compilers, to avoid the influence of vectorization capabilities in the initial experiments. The results are shown in Table 22**Errore. L'origine riferimento non è stata trovata.**.

TABLE 22: EXECUTION TIMES OF BOLT65 ENCODER

Video	armclang++		FCC		Speed-ups	
	Low-com-plex	High-com-plex	Low-com-plex	High-com-plex	Low-com-plex	High-com-plex
Akiyo	4.386	14.399	3.4942	11.325	x1.26	x1.27
Mobile	27.912	75.9	23.696	62.033	x1.18	x1.22
Soccer	156.45	489.24	123.55	380.7	x1.27	x1.29
City	160.31	521.33	127.47	409.39	x1.26	x1.27
Johnny	314.9	1037.5	239.49	794.14	x1.31	x1.31
Park run	353	1139.1	288.77	911.9	x1.22	x1.25
Shields	297.3	1034.5	234.73	822.66	x1.27	x1.26
Basketball drive	520.82	1659.8	409.76	1344.3	x1.27	x1.23
Blue sky	296.63	938.63	236.27	747.26	x1.26	x1.26
	2131.71	**6910.40**	**1687.23**	**5483.71**	**x1.25**	**x1.26**

The results show that the FCC compiler outperforms armclang++ by approximately x1.25 times in both encoding configurations. The speed-ups are calculated as a ratio between armclang++ implementation and FCC implementation.

Hotspots

To identify key kernels and bottlenecks of the application, we used two different profiling tools on two different architectures. On the IRENE cluster, with A64FX processors, we used perf profiler, while on Intel core i5, we used Intel Vtune Amplifier. The main idea of this type of profiling is to pinpoint methods and algorithms that consume the most time in the overall application lifecycle so that we can optimize the application more efficiently in future steps. The example output of the two profilers is shown in Figure 64.

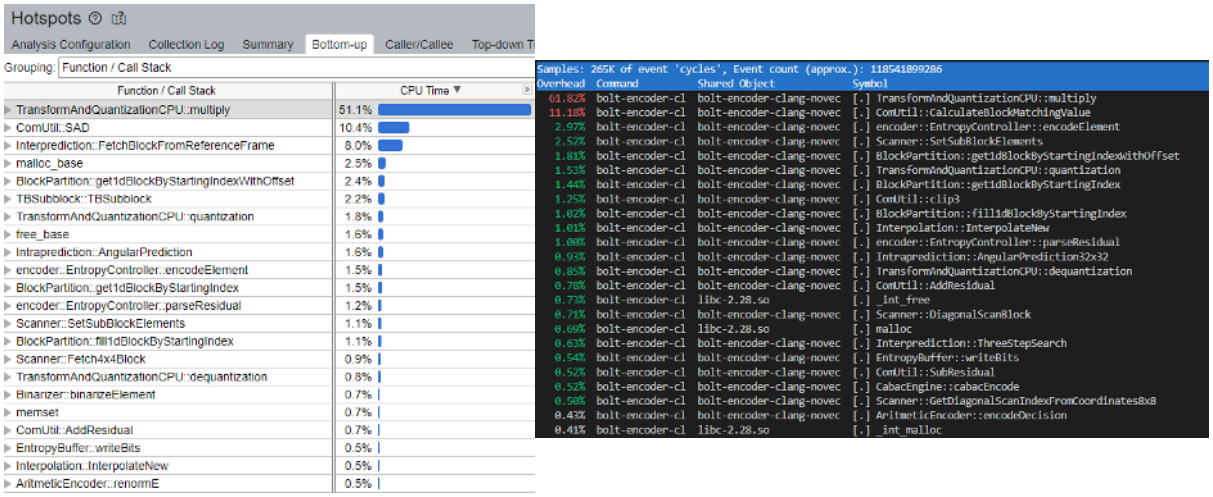


FIGURE 64: EXAMPLE OUTPUT OF TWO PROFILERS: INTEL VTUNE AMPLIFIER (LEFT), PERF (RIGHT)

We profiled the Bolt65 encoder on all video sequences, with two configurations ($B_{b65\text{-}h}$ and $B_{b65\text{-}l}$). The average percentage of the CPU time for each method is calculated for each configuration separately. The results are shown in Table 23, where a subset of methods that are the most time-consuming are included. In Figure 65**Errore. L'origine riferimento non è stata trovata.**, the same results are presented in a form of pie charts to better visualize the influence of each method on the overall application execution.

TABLE 23: BOLT65 ENCODER PROFILING RESULTS

Method name	**Vtune** $B_{b65\text{-}l}$	**Vtune** $B_{b65\text{-}h}$	**Perf** $B_{b65\text{-}l}$	**Perf** $B_{b65\text{-}h}$
TransformAndQuantizationCPU::multiply	49.50%	9.20%	62.30%	17.76%
ComUtil::SAD	10.50%	33.90%	10.10%	48.86%
Interprediction::FetchBlockFromReference-Frame	8.20%	23.70%	0.80%	4.12%
EntropyController::encodeElement	2.20%	2.60%	2.97%	3.50%
BlockPartition::get1dBlockByStartingIndex	5.00%	2.60%	2.95%	2.59%
TransformAndQuantizationCPU::quantization	1.80%	0.40%	1.53%	0.63%
BlockPartition::fill1dBlockByStartingIndex	1.10%	0.30%	0.77%	0.30%
EntropyController::parseResidual	1.20%	0.50%	1.03%	1.10%
Intraprediction::AngularPrediction	1.60%	10.70%	0.58%	8.32%
TransformAndQuantizationCPU::dequantiza-tion	0.90%	0.20%	0.91%	0.30%
Other	18.00%	15.90%	16.06%	12.52%

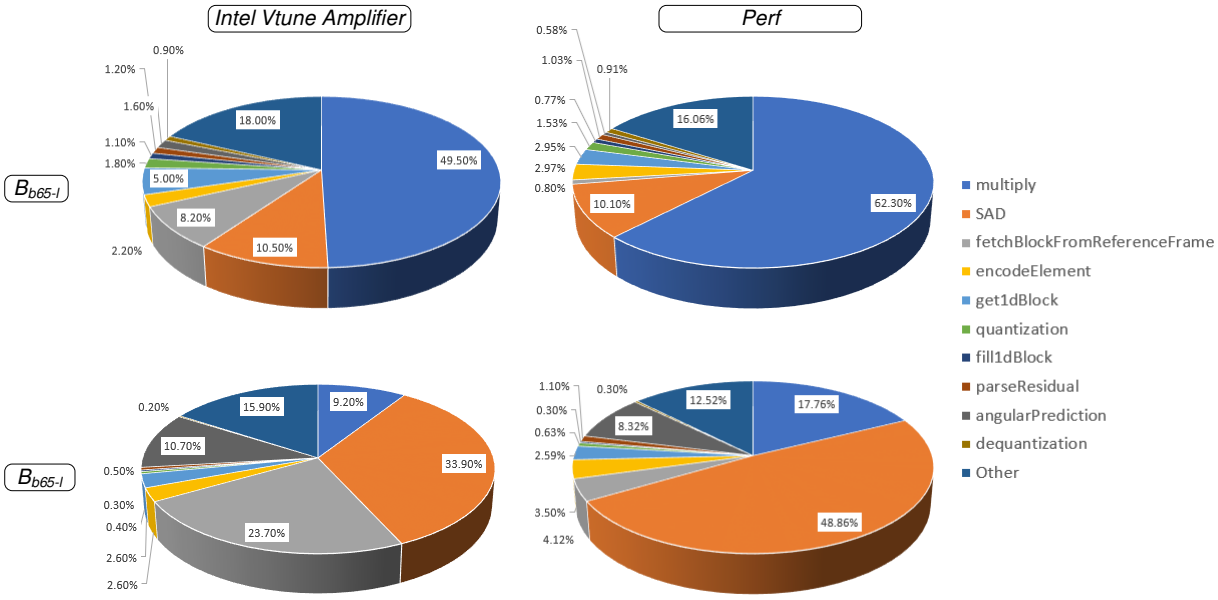


FIGURE 65: BOLT65 ENCODER PROFILING RESULTS

Although there is a lot of difference between the two different configurations, and two different architectures (Intel CPU measured with Vtune Amplifier, and A64fx measured with perf), a couple of methods always take the top spot in the share of the overall application lifecycle. These are the "*multiply*" function, which is integer matrix multiplication that is exclusively being called from DCT and IDCT methods, and "*SAD*" or Sum of absolute differences, which is being used for comparing two blocks in the process of prediction. Some other algorithms can be used instead of SAD, such as SATD (Sum of absolute transform differences) or MSE (mean square error), and that can affect these results by increasing the ratio of this kernel. Nevertheless, whichever kernel is used to calculate similarities between two blocks, it is one of the most time-consuming in every scenario. The biggest difference between the two architectures can be seen in the memory-related methods, such as "*fetchBlockFromReferenceFrame*", "*get1dBlock*", and "*fill1dBlock*", where the ratio of such methods is significantly smaller for A64fx that has HBM memory with much higher bandwidth than a general desktop Intel CPU.

From the results obtained by profiling and analysing the application, we can extract several methods that are the best candidates for optimization, specifically by implementing them using SVE vector extensions. Those are DCT, IDCT, and SAD. However, kernels such as quantization and dequantization are also good candidates due to their interconnectedness with DCT and IDCT, even though they are not as computationally expensive.

Hardware counters

Using the perf profiler on ARM A64FX and Intel Vtume Amplifier on Intel i5, we extracted several hardware events and counters that can be used for better insights into the Bol65 performance. **Errore. L'origine riferimento non è stata trovata.**Table 24 shows the metrics obtained and the tool used. The results are calculated as an average of encoding all video sequences for each configuration.

TABLE 24: HW EVENTS AND COUNTERS

Name	Tool	B_{b65-l}	B_{b65-h}
FLOPS	/	N/A	N/A
Instructions per cycle (IPS)	Perf	1.38	1.50

Cache misses	Perf	0.193%	0.089%
L1 instruction cache load misses	Perf	0.01%	0.01%
L1 data cache load misses	Perf	0.19%	0.09%
CPI rate (Cycles per Instruction Retired)	Vtune Amplifier	0.355	0.346
Memory operations	Vtune Amplifier	15.9%	16.2%

Floating point operations per second (FLOPS) is not available for Bolt65, since all the operations in the HEVC standard are integer based.

Thread scalability

The main parallelization currently supported in Bolt65 relies on implementing tiles as one of the main parallelization strategies used in the HEVC standard. The parallelization in Bolt65 is implemented with a concurrency support standard library, supported since C++11. After the frame is split into a certain number of tiles, each tile is processed on a separate thread. Therefore, most of the workload in the encoder can be distributed between processing cores. Only at the end, encoded streams of each tile have to be joined together to form a final output file. To measure the thread scalability, we encoded videos with different tile arrangements, where each tile was processed on a separate thread of the same node, which means that the number of tiles always corresponds to the number of threads. The experiments were done on the IRENE cluster ARM A64FX with 48 cores per node. An example of a frame that is split into 3x3 tiles can be seen in Figure 63, and the results of encoding BasketballDrive video with the FullHD resolution (1920x1080) are shown in Table 25**Errore. L'origine riferimento non è stata trovata.**. For better visualization of the results, we show the ratio of the number of threads used compared with the performance speedup achieved in Figure 66.

TABLE 25: MULTITHREAD PERFORMANCE SPEEDUP

Threads	Tiles	Execution time	Speedup
1	1x1	1661.82	x1.00
2	1x2	842.02	x1.97
3	1x3	573.55	x2.90
4	2x2	456.78	x3.64
6	2x3	313.88	x5.29
8	2x4	250.6	x6.63
9	3x3	218.44	x7.61
10	2x5	195.56	x8.50
12	3x4	177.4	x9.37
16	4x4	141.63	x11.73
20	4x5	112.47	x14.78
25	5x5	96.87	x17.16
30	5x6	84.3	x19.71
36	6x6	69.05	x24.07
42	6x7	66.98	x24.81

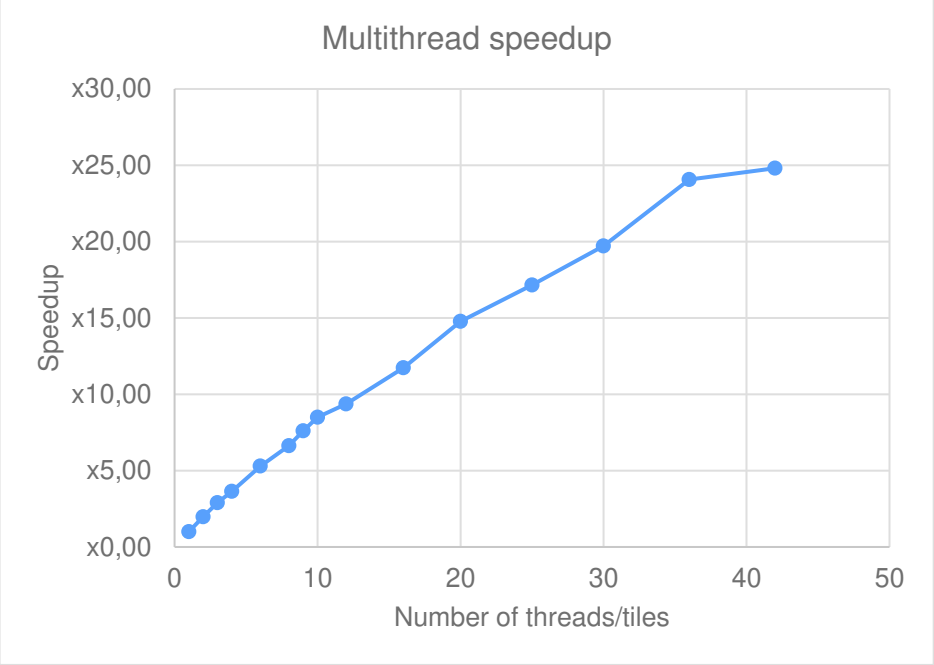


FIGURE 66: MULTITHREAD PERFORMANCE SPEEDUP

The results show that for a lower number of threads/tiles the speedup closely follows the increase of the used threads (e.g., x2.9 speedup for 3 threads or x5.29 for 6 threads). However, the higher the number of threads, the speedup became less linear. The main reason behind this behaviour is the fact that the workload that is being offloaded to each core is not the same. Since the frame is uniformly divided into the tiles solely depending on the number of tiles in each row and column, the computation needed to encode each tile differs. An example of this workload disbalance between tiles is shown in Figure 67 on a frame composed of 6x6 tiles.

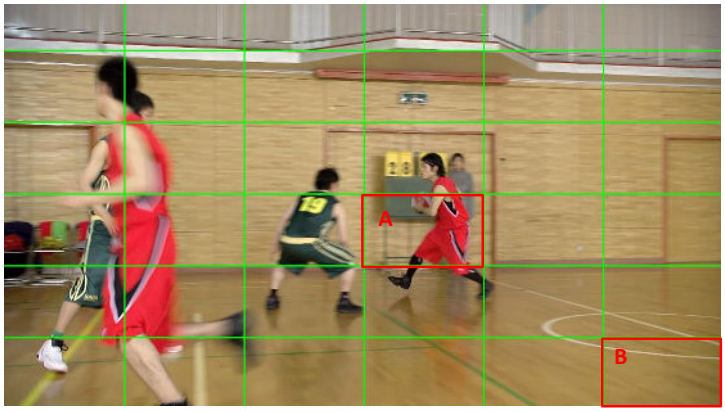


FIGURE 67: WORKLOAD DISBALANCE BETWEEN TILES

The tile marked as B contains only the background and does not have a lot of motion, meaning that it is computationally much less expensive than tile A in the figure, where there is an object in motion. However, since each tile is assigned to one core, the overall processing time depends on the processing time of the most complex tile. To avoid this, more advanced load-balancing algorithms that divide frames non-uniformly are necessary and will be a part of future research in the EUPEX project.

Conclusion of the Analysis

With extensive profiling and analysis of the application with two different profilers and on two different architectures, we identified several key kernels that consume a significant portion of the execution time.

These kernels will be the basis for future optimization, specifically for vectorization on the ARM SVE vector extension that will be available on the final platform.

An additional aspect that has to be taken into consideration is the compiler. The initial results show that Fujitsu's FCC compiler outperforms ARM's armclang++ by approximately 25% on average. However, the results are measured only on the scalar version of the application. The compiler's capabilities for efficient manual and automatic vectorization using SVE will also have to be investigated during the subsequent experiments.

With the current parallelization implementation of Bolt65 that uses HEVC tiles, we can achieve linear scalability up to 7 or 8 cores, but with a higher number of cores, the efficiency is decreasing. In order to improve thread scalability on more cores, we have to investigate different load-balancing algorithms that will arrange workloads between tiles more efficiently. Also, with a larger number of cores, we can investigate the combination of different parallelization techniques, where we can combine HEVC tiles, with wavefronts or OMP parallelization.

Dyablo-Whole Sun

Application description

dyablo is a new simulation code developed at CEA in Saclay for the modelling of magneto-hydrodynamics on Adaptive Mesh Refinement (AMR) grids. To reach exascale performances, the code relies on the portability performance library Kokkos (Trott, et al. 2021) to do shared parallelism while distributed parallelism is done using MPI.

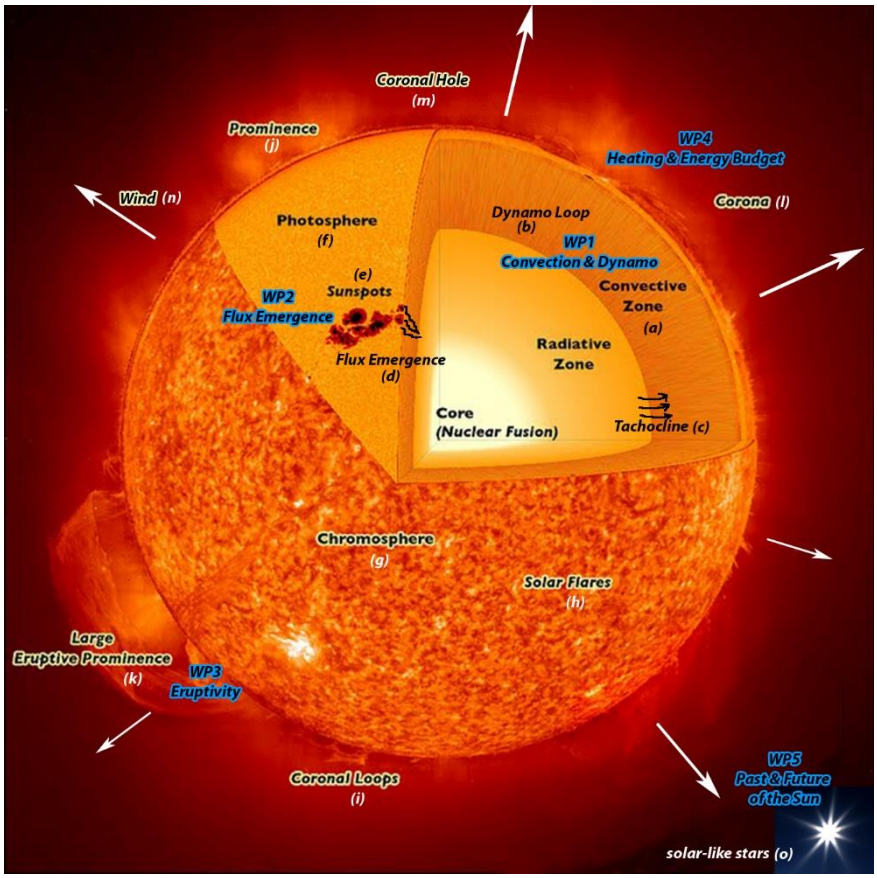


FIGURE 68: THE SUN IS STRUCTURED IN ZONES WHICH HAVE VERY DIFFERENT CHARACTERISTIC TIME AND SPATIAL SCALES AND THEIR OWN PHYSICS REGIMES. THE AIM OF DYABLO AND OF THE WHOLE SUN PROJECT IS TO PRODUCE THE FIRST SIMULATIONS FROM INSIDE THE RADIATIVE ZONE TO THE CORONA. THESE SIMULATIONS WILL THEN BE EXTENDED TO OTHER STELLAR TYPES

As part of the ERC project Whole Sun, *dyablo* will be used to produce the very first global simulations of the Sun, from inside the radiative layer up to the corona (see Figure 68). Despite the close proximity of our star and decades of research, there is still a lot to understand about the Sun and Sun-like stars (Brun e Browning, Magnetism, dynamo action and the solar-stellar connection 2017). Understanding solar activity is not a major scientific goal for stellar physics but also for the study of Earth's direct spatial neighbourhood and its climate. To be able to answer questions such as the solar magnetic field emergence, sunspots formation, the origin of the 11-year activity cycle or the extreme temperature of the corona, it is absolutely necessary to consider the Sun as a whole. Modelling the Sun requires to be able to simulate magneto-hydrodynamics, radiative transfer and gravitational on domains reaching several millions of kilometres in size, while reaching resolutions of a few tens of kilometres vertically and a few hundreds of kilometres horizontally at the surface of the Sun. These resolutions are necessary if we are to resolve the turbulent solar plasma energy spectrum. The same considerations apply to temporal scales: characteristic timescales for surface convection are about a few minutes while magnetic polarity inversion cycles of the Sun last on average 22 years.

Modelling such scales requires very high Reynolds' numbers associated with a very strong turbulence. These in turn require high performance and high scalability on future exascale machines. This is the reason for the choice of the Kokkos library allowing us to port dyablo on a large variety of hardware while conserving high performance and high scalability.

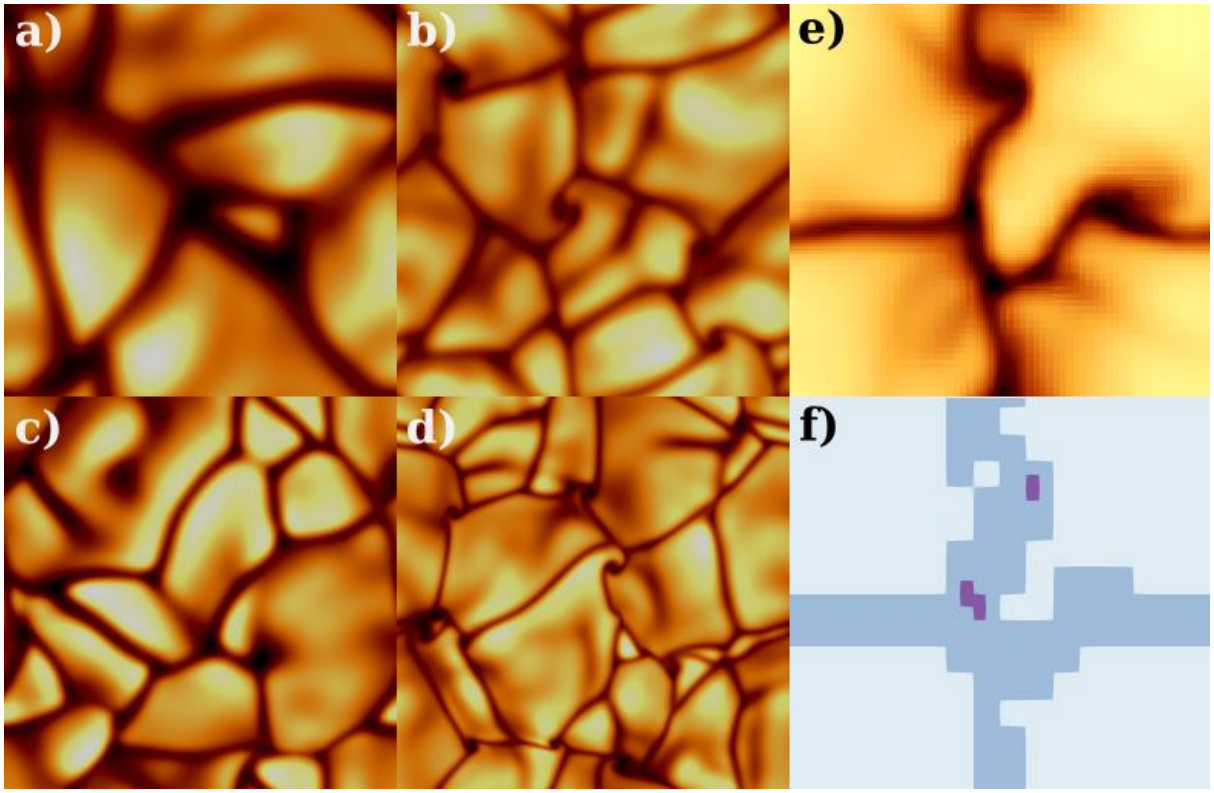


FIGURE 69: HORIZONTAL TEMPERATURE SLICES TAKEN FROM DIFFERENT CONVECTION RUNS USING DYABLO. IMAGES A TO D REPRESENT DIFFERENT RUNS ON A FIXED GRID MODEL WITH DIFFERENT PARAMETERS LEADING TO INCREASING TURBULENCE. IMAGES E-F SHOW A RUN WITH ADAPTIVE MESH REFINEMENT

dyablo is still in its prototyping phase but the code is already ready to perform some runs. We have a code base that relies on Godunov's method simulating compressible Navier-Stokes equations, static gravity as well as thermal conduction on an AMR grid. We are able to reproduce well-known hydrodynamical tests (SOD shock tube, Rayleigh-Taylor, Kelvin-Hemlholtz and Rayleigh-Bénard instabilities). As part of the Whole-Sun project, Dyablo is at the centre of a compressible hydrodynamics convection benchmark in three dimensions and is already reproducing well-known results (Cattaneo, et al. 1991) in solar convection, see for instance Figure 69 and Figure 70.

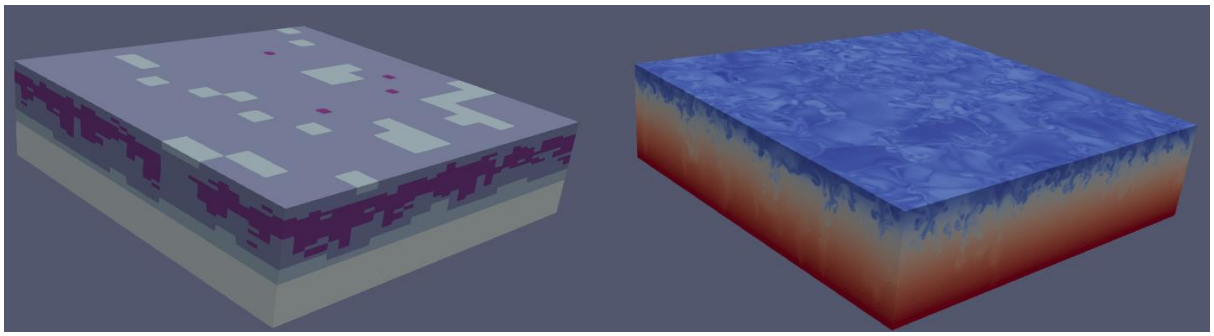


FIGURE 70: TURBULENT CONVECTION RUN WITH AMR IN 3D. LEFT PANEL: THE REFINEMENT LEVEL OF THE GRID WHERE LIGHT COLOURS ARE LOW-LEVELS OF REFINEMENT AND DARK COLOURS ARE HIGH-LEVELS OF REFINEMENT. RIGHT PANEL: TEMPERATURE OF THE FLUID, BLUE IS COLD, RED IS HOT.

At its core, dyablo applies computational kernels to cells in an AMR tree. The tree is stored in memory along a Morton Z-curve, and dyablo is in charge of domain decomposition and load balancing this structure over multiple MPI-processes. To be compatible with multiple architectures, and in particular with accelerator architectures where part of the memory has to be offloaded from the main CPU memory to the accelerator, dyablo exposes two different backends. One is relying the PABLO library and is dedicated to CPU computations, while the other is our own in-house implementation of an AMR backend residing on the device. This second backend allow us to benefit as much as possible from “direct to GPU" solutions (such as CUDA aware communications). Moreover, to improve parallelism efficiency, decrease the complexity of the AMR tree, and expose more vectorization power in the application, dyablo implements block-based AMR where all the leaves of the refinement tree are in fact regular cartesian grids. These regular cartesian grid allow the code to minimize the number of level-jumps in refinement and allow for a better streamlining of the calculations.

Application use-case

Convection is a central process in the dynamics of the Sun and solar-like stars. Although Solar convection has been studied and simulated for decades from the seminal works of (Gilman e Glatzmaier 1981), (Nordlund 1982) and (Hurlburt, Toomre e Massaguer 1984) until the more recent advances of (Brun e Toomre, Turbulent convection under the influence of rotation: sustaining a strong differential rotation 2002), (Miesch, et al. 2008) and (Hotta e Kusano 2021), this process and its consequences on the solar dynamics is still under active scrutiny. One classical problem with the simulation of convection is that no code is currently capable of generating rising and emerging sunspots from first principles. Sunspots in simulations are often introduced by hand in the setups, while global dynamo simulations do not have realistic enough atmospheres allowing for flux emergence.

One way to make progress on these topics is to take a holistic approach, linking the global interior of the Sun with the atmosphere and the corona (See for instance arguments developed in (Nelson e Miesch 2014), (Warnecke, et al. 2016) and (Perri, et al. 2021)). However, building a code able to approach the problem globally is extremely difficult. The variety of physical regimes, as well as the sheer number of points required to resolve convection to a sufficient level is huge. For instance, an ideal resolution at the surface of the Sun would require meshes 10 km high and 30 km wide on a sphere of 700 Mifsm radius with the need to adapt along the radius both the horizontal and vertical resolutions. A gross estimate indicates that more than 100 billion cells are needed, so this requires extreme computing resources. Thus, developing a new solar global code requires high-performance computing expertise and modern hardware.

The use case we present in the application is the simulation of a convective slab similar to a part of the convective zone of the Sun. The convective slab is parametrized by an aspect ratio, a vertical stratification and a turbulence parameter. These three parameters define a grid of models that have already been simulated using dyablo on GPU and traditional CPU architectures. The objective of this use-case is to analyze, adapt and optimize the computation of this runs on ARM architectures.

Results and Analysis

The workflow for running and analysing the test-case runs on the target architecture is decomposed in four parts:

1. Compilation and fine-tuning of the build step for the target architecture using Kokkos.
2. Preparation of the setup on the target architecture.
3. Running the setup until thermal relaxation where the solution is sufficiently relaxed that the AMR grids and the workload are representative of a full scientific production run.
4. Parameter study on the stratification and the turbulence of the domains as well as their performance

At the present stage, the analysis of performance has already been established and the workflow defined for CPU and GPU runs on the Jean-Zay machine in France. Performance results are shown on Figure 71.

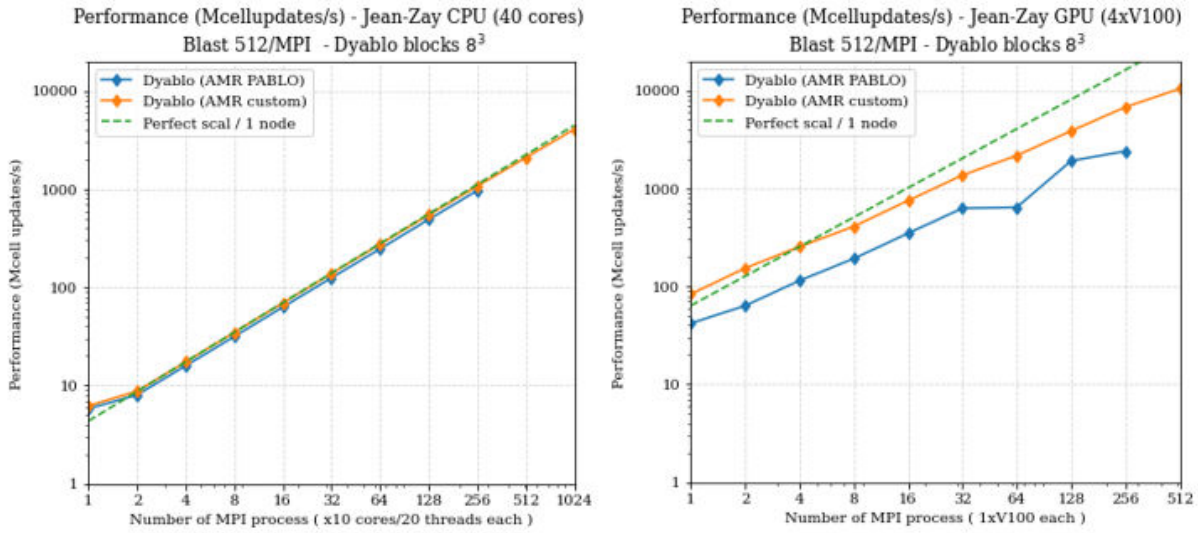


FIGURE 71: PERFORMANCE STUDY OF DYABLO ON THE JEAN-ZAY SUPERCOMPUTER. X-AXIS: NUMBER OF MPI-PROCESSES, Y-AXIS: PERFORMANCE IN MILLION-CELLS UPDATES PER SECOND. LEFT: CPU PARTITION; RIGHT: GPU PARTITION. BLUE: AMR AN EXTERNAL LIBRARY; ORANGE: CUSTOM-AMR SOLUTION.

Conclusion of the Analysis

In the context of Eupex, the Dyablo Whole-Sun project has 9+3PM to spend on deploying the use-case and analysing the performances of the code on ARM architectures. The work has not started as of today and CEA and ATOS will work together in 2023 to compile, execute, analyse and optimize the code to the A64FX partition of IRENE. Due to the usage of the Kokkos framework for performance portability, we are confident that dyablo will be running out of the shelf on the A64FX architecture. Moreover, the usage of block-based AMR where the leaves of the refinement tree are regular cartesian grid is in theory well adapted for highly vectorial processing units such as the A64FX. However, the high level of hierarchical parallelism we use in dyablo might impede the exploitation of vectorization by Kokkos on the machine. The performance study will show if Kokkos is able to correctly explicit the vectorial parts of the code and if not, an adaptation of the code will be made for these architectures. Determining if Kokkos is a viable option for ARM architectures will be crucial in the shaping of future codes for exascale.

Codesign

Codesign EUPEX platform

Even if most of the design choices of EUPEX platform was defined at the proposal time of the project, the WP3 provides a set of codesign outputs based on the use cases of the EUPEX applications and on the design expertise of the WP3 partners. Indeed, application leaders from WP3 have surveyed the software stack and hardware characteristics of the systems on which the selected uses cases of the EUPEX benchmark suite usually are executed. We use this material to drive the codesign choices and we provide as input for WP4 and WP5 work.

In Table 26, we report a summary for the codesign inputs for the CPU node of the EUPEX platform:

TABLE 26: CODESIGN INPUTS FOR THE CPU NODE

CPU Node	Details	Codesign
CPU	Number of cores, cache size, HBM size, etc.	Fixed by Rhea spec
DRAM	4 channels of DDR5	Codesign
Number of sockets	1 or 2 due to Rhea limitation	Codesign
Network bandwidth per node	Limited by the number of NICs	Codesign
Number of NICs per node	1 to 4 due to blade limitation	Codesign
Local disk	Size and technology	Codesign

Furthermore, in Table 27 we report a summary for the codesign inputs for the GPU node of the EUPEX platform:

TABLE 27: CODESIGN INPUTS FOR THE GPU NODE CODESIGN

GPU Node	Details	Codesign
CPU	Number of cores, cache size, HBM size, etc. (Rhea specs)	Fixed by Rhea spec
DRAM	4 channels of DDR5	Codesign
Number of sockets	1 or 2 due to Rhea limitation	Codesign
Network bandwidth per node	Limited by the number of NICs	Codesign
Number of NICs per node	1 to 4 due to blade limitation	Codesign
Local disk	Size and technology	Codesign
Number of GPUs	1 to 4 due to blade limitation	Codesign
GPU technology	See Milestone 3	Codesign

In Table 28, we report other codesign input at system level:

TABLE 28: CODESIGN INPUTS FOR THE SYSTEM

CPU Node	Details	Codesign
Fabric technology	BXI v2 interconnect	Fixed by project design
Fabric topology	BXI v2 supports only Fat Tree or Dragonfly+ topology	Codesign
Bandwidth per NIC	BXI v2 supports up to 100Gbit/s per NIC	Fixed by project design
Cabinet technology	BullSequana XH3000 – 32 blades	Fixed by project design
Number of cabinet (or nodes)	1 cabinet GPU (32 nodes) and 1 cabinet CPU (96 nodes)	Fixed by project design
Number of nodes per cabinet	96	Fixed by spec of BullSequana XH3000

Number of nodes per blade	32	Fixed by spec of BullSequana XH3000
System global storage	Provided by CEA through a gateway	Fixed by project design

Based on the codesign inputs that WP3 received, we report in the following the codesign outputs collected:

For the CPU node, we defined the following codesign parameters:

- **DRAM**: from the survey of the application requirements that we collected during the WP3 work, we discovered that the EUPEX applications have similar memory requirements. In Table 29, we report the memory utilization of the EUPEX applications when run the use cases identified in the project. We notice that on average the memory utilization of the applications range between 2GB/core and 4GB/core. If we consider that Rhea will be equipped with 48 cores, we need at least 256GB of DDR5 per socket (512GB of DDR5 per bi-socket node).

TABLE 29: SURVEY OF MEMORY REQUIREMENTS OF THE EUPEX APPLICATIONS FOR THE USE CASES IDENTIFIED IN THE PROJECT

Application	GB/core
Integrated Forecasting System	2
Forestering Precision Agriculture	4
Machine Learning for Earth Observation	4
SPECFEM3D	4
ESPRESO FEM	4
LiGen	4
BigDFT	4
OpenGadget3	2

- **Number of sockets**: From the survey, we also found that all applications of the EUPEX benchmark suite have been optimized to run on double-socket servers due most of the Top500 HPC systems use this configuration. Indeed, Rhea has been equipped with link-to-link socket capabilities to support double-socket configuration. Moreover, the blades of the BullSequana XH3000 are optimized to host double-socket server, so designing dual-socket nodes would improve the overall efficiency and performance. For this reason, we aim to have dual-socket server for the CPU node of EUPEX platform.
- **Network bandwidth per node (or number of NICs per node)**: the network bandwidth is limited by the number of NICs with which a node can be equipped, 4 in the case of the BullSequana XH3000 blade. The design of the network is always limited by a cost-effective solution due a higher number of network ports requires a higher number of switches leading to a higher network cost. From the point of view of the applications, the higher the bandwidth, the lower the time spent in data movement. The EUPEX benchmark suite is composed of representative HPC applications that require high data movement. For this reason, the minimum number of NICs would be no fewer than 2 (200 Gbit/s in total). Two NICs per node would also provide some degree of resiliency, with dual-NIC nodes connected to two different switches, avoiding disconnection in case of a faulty switch.
- **Local disk**: applications that compose the EUPEX benchmark suite have no need of local storage, so a small hard disk would be helpful only in the management of the node. In this case, we suggest a small SSD NVMe disk technology to minimize the power consumption of the node.

TABLE 30: CODESIGN OUTPUTS FOR THE CPU NODE

CPU Node	Codesign outputs
DRAM	512GB (4x128GB DDR5 DIMM)
Number of sockets	2
Network bandwidth per node	200 Gbit/s
Number of NICs per node	2 (1 NIC per socket)

Local disk	Not required for the applications

Instead, for the GPU node, we defined the following codesign parameters:

- **Number of GPUs**: In Table 31, we report the number of GPU that the EUPEX applications usually use when run the use cases identified for the project. We can see that in average the EUPEX applications use 4 GPU per node, this is also the average number of GPU equipped in most of the Top500 system. The higher the number of GPUs per node, the higher the shared GPU memory that allows larger job to run on a single node, this is also very important for AI applications that require large GPU memory to fit ML models. Moreover, all the GPU technologies that we are considering for the EUPEX platform, have enough GPU-to-GPU links (e.g., NVlink, Xe link, Infinity fabric link) to connect P2P four GPUs without requiring a high-cost link switch. This allows efficient near-neighbour communication, improving time-to-solution for all the applications of the EUPEX benchmark suite. For these reasons, we aim for 4 GPUs per node.

TABLE 31: SURVEY OF GPU REQUIREMENTS OF THE EUPEX APPLICATIONS FOR THE USE CASES IDENTIFIED IN THE PROJECT

Application	GPU per node
Integrated Forecasting System	N/A (4 for testing)
Forestering Precision Agriculture	N/A
Machine Learning for Earth Observation	4
SPECFEM3D	4
ESPRESO FEM	8
LiGen	4
BigDFT	4
OpenGadget3	4

- **DRAM**: the DRAM memory size of a GPU node is driven by the memory size of the GPUs due the data contained in the GPU memory should be also contained in the host memory. Nvidia H100 and Intel Ponte Vecchio respectively have 80GB and 128GB of embedded HBM memory. For this reason, we target to have at least 512 GB of DDR5 for nodes equipped with 4 GPUs.
- **Network bandwidth per node (or number of NICs per node)**: each GPU should have a dedicated NIC to allow efficient GPU-to-GPU communication (in particular, AI applications like AI4EO are very sensible on GPU-to-GPU RDMA communication in term of performance). For this reason, 4 NICs are mandatory for a node with 4 GPUs.
- **Number of sockets**: In order to install 4 GPUs and 4 NICs per node, double-socket servers are mandatory due the number of PCIe lanes available to a single Rhea processor is insufficient to host 4 GPUs and 4 NICs.

TABLE 32: CODESIGN OUTPUTS FOR THE GPU NODE

GPU Node	Codesign outputs
Number of GPUs	4
DDR Memory	512GB (4x128GB DDR5 DIMM)
Number of sockets	2
Network bandwidth per node	400 Gbit/s
Number of NICs per node	4
Local disk	No required for the applications

To conclude, we report in the following the codesign outputs for the system:

- **Fabric topology**: Due the relatively low number of nodes in the system (128), we aim to design a Fat Tree topology due Dragonfly+ topology has a cost-advantage design only for HPC systems with a high number of nodes (>500). In order to achieve maximum system performance,

we aim to have non-blocking communication among the nodes with a Full Fat Tree topology. But some level of oversubscription is accepted in case of cost reduction strategies imposed by the project management.

TABLE 33: CODESIGN OUTPUTS FOR THE EUPEX SYSTEM

System	Codesign outputs
Fabric topology	Full Fat Tree (some level of oversubscription in case of cost-reduction policies)

Selection of the GPU technology

A key strategic action for the WP3 is the selection of the GPU technology for the accelerated partition of the EUPEX platform. In collaboration with the WP5, we have to select the GPU technology that the EUPEX platform will be equipped. For this reason, WP3 established a collaboration contact with the main GPU vendors on the HPC market to discuss the support of Rhea with different GPU technologies, in the following we report the outcome of all the discussions:

- **AMD**: AMD initially proposed the AMD Mi250x to support the EUPEX platform, but unfortunately, we were not able to accept this proposition due AMD (at that time) didn't have a roadmap to support Rhea processor and, at large, the ARM ecosystem. For this reason, we excluded a priori this technology for the EUPEX platform.
- **Nvidia**: The Nvidia technology and its ecosystem is supported by project design to the EUPEX applications due they natively support Nvidia CUDA ecosystem. During this collaboration, Nvidia also signed a strategic agreement with SiPearl to support Rhea processor on the Nvidia software ecosystem. Nvidia proposed the Hopper technology (H100) to equip the EUPEX platform and we invited them for a detailed presentation of the H100 for all the project WPs.
- **Intel**: Before to start the discussion with the GPU technology providers, Intel had signed a strategic agreement with SiPearl to support Rhea processors on Intel OneAPI ecosystem. Intel was the first technology provider to have a strategic agreement with SiPearl and they suddenly propose their GPU technology to the EUPEX project. In particular, Intel proposed Ponte Vecchio GPU and the next Rialto GPU technology that would be both available in the time frame of the project. Intel was particular committed to the project, they also provided a dedicated team that worked in couple with all the application proponent partners of the WP3 to analyse the compatibly of the EUPEX applications with the Intel OneAPI GPU stack. Moreover, Intel proposed support on the code porting and optimization on Intel GPU technology for the project. After the presentation of the Intel team, we also received a letter of intent from Jeff McVeigh (VP, GM Super Compute Group of Intel Corporation) to remark the Intel commitment on the project (Annex 1), which is also linked to the strategic investment that Intel has been doing in Europe[22].

We report in Table 34 a comparison between Intel Ponte Vecchio and Nvidia H100 GPU.

TABLE 34: COMPARISON OF INTEL PONTE VECCHIO AND NVIDIA H100 GPU

Product	Ponte Vecchio	H100 80GB
Release Date	2022	2022
Architecture	Xe-HPC	Hopper
Transistors	100 B	80 B
Tiles (inc HBM)	47	6 + 1 spare
Manufacturing	Intel 7 (7nm)	TSMC N4 (4nm)
Compute Units	128	132
Matrix Cores	1024	528
Form Factors	OAM (600W)	SXM5 (700W)
TFLOPS FP32	52	67
TFLOPS (F32) / Peak W (theoretical)	0.086	0.096

[22] https://ec.europa.eu/commission/presscorner/detail/en/STATEMENT_22_1764

TFLOPS FP64	52	34
TFLOPS (F64) / Peak W (theoretical)	0.086	0.046
L2 / L3	2 x 204MB	50MB
VRAM Capacity	128 GB	80 GB
VRAM Type	HBM2e	HBM3
VRAM Width	8192-bit	5120-bit
VRAM Bandwidth	3.2 TB/s	3.35 TB/s
Chip-to-Chip Total BW	64 x 11.25 GB/s 720 GB/s	18 x 50 GB/s 900 GB/s
Chip-to-Host	PCIe Gen5	PCIe Gen5
CPU Coherency	Yes	With NVLink 4
GPU Direct	IB / BXI	IB / BXI

After the internal compatibility analysis of EUPEX applications with Intel GPU technology, the WP3 partners unanimously agree with the following statement:

Intel GPU can be a technology suitable for the EUPEX benchmark suite but could require some extra effort on the code porting.

The WP3 reports the above statement to the project opening the possibility to use both GPU technologies. The decision on the final GPU technology will be taken in collaboration with the WP5 and the project management team considering also external factors of the WP3.

Early Access Program

EUPEX in collaboration with the EPI-SGA2 project, has organised a kick-off meeting to start the discussion on the organization of the early access program (EAP) of EUPEX platform due it will become the first platform with EPI processor available to the scientific communities, centre of excellence, and research projects. In EUPEX, we proposed that a team composed of benchmark experts involved in WP3, in conjunction with WP4, WP5 & WP6, will provide support to the scientific communities selected from a EUPEX committee to get access to the system. Instead, EPI-SGA2 will support the EAP for microarchitecture details analysis of Rhea. In the Figure 72, we report an initial organization of the EAP.

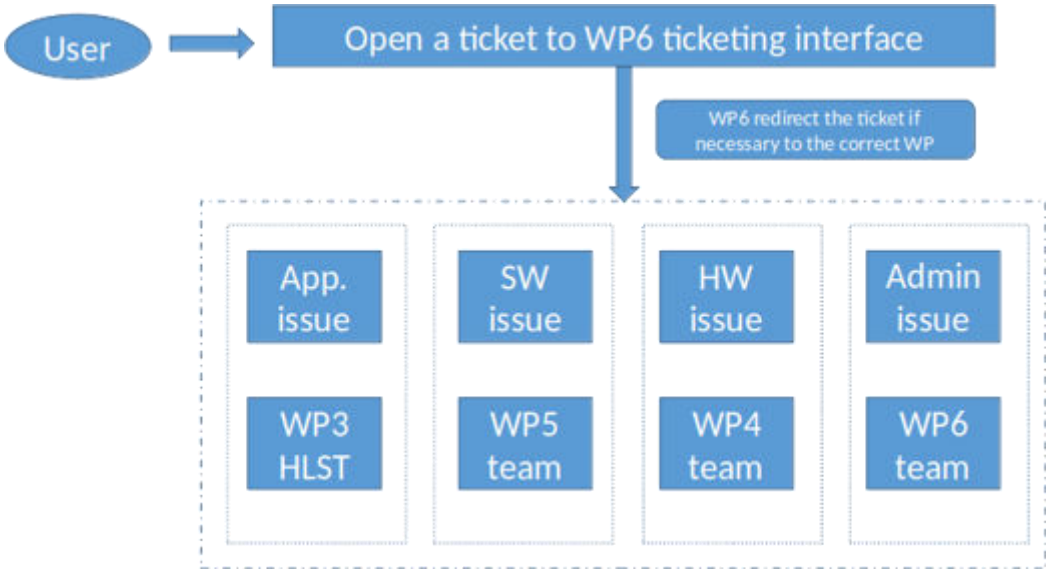


FIGURE 72: OVERVIEW OF THE EARLY ACCESS PROGRAM ORGANIZATION OF EUPEX PLATFORM

EAP is still an activity in progress due we aspect the platform will be available in the last year of the project. In the meantime, we are currently working on the access to some SDV platforms with the software stack developed in the WP5 available for testing.

Conclusion

In this document, we presented the work done in the WP3 in the first year of the project (from M0 to M12).

We reported the application descriptions and the use cases identified to benchmark the EUPEX platform. We ported the codes on ARM ecosystem using the target SDVs (Fujitsu A64FX and Nvidia A100/V100). We analysed the main application workloads providing a global overview of the characteristic of the applications of EUPEX benchmark suite. In particular, our work was focused on the workflow of the applications and of the selected use case to reproduce real scientific results. We characterized the application workload in order to codesign the modular supercomputing architecture of the EUPEX platform.

We conclude the document with a set of codesign outcomes for the EUPEX platform and for the GPU technology selection based on the application analysis of the EUPEX benchmark suite that, it will use from the WP4 to design the system architecture of the EUPEX platform. Moreover, we provided some updates on the early access program developed in collaboration with the EPI-SGA2 project.

References

Aalborg University. *Video Trace Library, YUV Video Sequence.* s.d. http://trace.eas.asu.edu/yuv.

Aschbacher, Josef. «ESA's earth observation strategy and Copernicus.» In *Satellite earth observations and their impact on society and policy*, 81–86. Springer, Singapore, 2017.

Auyeung, C., et al. «A combined proposal from JCTVC-G366 JCTVC-G657 and JCTVC-G768 on context reduction of significance map coding with CABAC.» *Joint Collaborative Team on Video Coding (JCT-VC), Document JCTVC-G1015, Geneva*, 2011.

Bossen, Frank, Benjamin Bross, Karsten Suhring, e David Flynn. «HEVC complexity and implementation analysis.» *IEEE Transactions on circuits and Systems for Video Technology* (IEEE) 22 (2012): 1685–1696.

Brun, Allan Sacha, e Juri Toomre. «Turbulent convection under the influence of rotation: sustaining a strong differential rotation.» *The Astrophysical Journal* (IOP Publishing) 570 (2002): 865.

Brun, Allan Sacha, e Matthew K. Browning. «Magnetism, dynamo action and the solar-stellar connection.» *Living Reviews in Solar Physics* (Springer) 14 (2017): 1–133.

Bruzzone, Lorenzo, e Begüm Demir. «A review of modern approaches to classification of remote sensing data.» *Land Use and Land Cover Mapping in Europe* (Springer), 2014: 127–143.

Canty, Morton J. *Image analysis, classification and change detection in remote sensing: with algorithms for ENVI/IDL and Python.* Crc Press, 2014.

Cattaneo, Fausto, Nicholas H. Brummell, Juri Toomre, Andrea Malagoli, e Neal E. Hurlburt. «Turbulent compressible convection.» *The Astrophysical Journal* 370 (1991): 282–294.

Chang, Bill and Kurian, Rajiv and Williams, Doug and Quinnell, Eric. «DOJO: Super-Compute System Scaling for ML Training.» *2022 IEEE Hot Chips 34 Symposium (HCS)*, 2022: 1-45.

Cisco, U. «Cisco annual internet report (2018–2023) white paper.» *Cisco: San Jose, CA, USA*, 2020.

Cisco, V. N. I. «Cisco visual networking index: Forecast and methodology, 2016–2021.» *CISCO White paper*, 2017.

Congalton, Russell G., Jianyu Gu, Kamini Yadav, Prasad Thenkabail, e Mutlu Ozdogan. «Global land cover mapping: A review and uncertainty analysis.» *Remote Sensing* (MDPI) 6 (2014): 12070–12093.

Ding, Nan, e Samuel Williams. *An instruction roofline model for gpus.* IEEE, 2019.

Dostal Zdenek, Horak David, Kucera Radek. «Total FETI-an easier implementable variant of the FETI method for numerical solution of elliptic PDE.» *Communications in Numerical Methods in Engineering*, 2006.

Ester, Martin, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, e others. «A density-based algorithm for discovering clusters in large spatial databases with noise.» *kdd.* 1996. 226–231.

Evangelos Maliaroudakis, Alexandros Kanterakis Manolis Marazakis, e Angelos Bilas. «Interactive, Cloud-Native Workflows on HPC Using KNoC.» *Proceedings of the 17th Workshop on Virtualization in High-Performance Cloud Computing (VHPC 2022). In conjunction with ISC-HPC*, 2022.

Farhat C., Mandel J., Roux F.X. «Optimal convergence properties of the FETI domain decomposition.» *Computer Methods in Applied Mechanics and Engineering*, 1994.

Fu, Chih-Ming, Ching-Yeh Chen, Yu-Wen Huang, e Shawmin Lei. «Sample adaptive offset for HEVC.» *2011 IEEE 13th International Workshop on Multimedia Signal Processing.* 2011. 1–5.

George Zervas, Yannis Sfakianakis Christos Kozanitis, e Angelos Bilas. «Virtual Clusters: Isolated, Containerized HPC Environments in Kubernetes.» *Proceedings of the 17th Workshop on Virtualization in High-Performance Cloud Computing (VHPC 2022). In conjunction with ISC-HPC*, 2022.

Gilman, Peter A., e Gary A. Glatzmaier. «Compressible convection in a rotating spherical shell. I-Anelastic equations. II-A linear anelastic model. III-Analytic model for compressible vorticity waves.» *The Astrophysical Journal Supplement Series* 45 (1981): 335–388.

Gosselet, Pierre, Rey, Christian. «Non-overlapping domain decomposition methods in structural mechanics.» *Archives of Computational Methods in Engineering*, 2006.

Götz, Markus, Christian Bodenstein, e Morris Riedel. «HPDBSCAN: highly parallel DBSCAN.» *Proceedings of the workshop on machine learning in high-performance computing environments.* 2015. 1–10.

Goyal, Priya, et al. «Accurate, large minibatch sgd: Training imagenet in 1 hour.» *arXiv preprint arXiv:1706.02677*, 2017.

Graves, R. W., E. Seyhan, e J. P. Stewart. «Broadband Ground Motion Simulations for a Kinematic Variation of the Mw 7.8 ShakeOut Rupture.» *AGU Fall Meeting Abstracts.* 2011. S52B–06.

Hansen, Matthew C., Ruth S. DeFries, John R. G. Townshend, e Rob Sohlberg. «Global land cover classification at 1 km spatial resolution using a classification tree approach.» *International journal of remote sensing* (Taylor & Francis) 21 (2010): 1331–1364.

Hotta, H., e K. Kusano. «Solar differential rotation reproduced with high-resolution simulation.» *Nature Astronomy* (Nature Publishing Group) 5 (2021): 1100–1102.

Hu, Guangchao, Yang Zhang, e Wenbo Chen. «Exploring the performance of singularity for high performance computing scenarios.» *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS).* 2019. 2587–2593.

Hurlburt, Neal E., J. Toomre, e Joseph M. Massaguer. «Two-dimensional compressible convection extending over multiple scale heights.» *The Astrophysical Journal* 282 (1984): 557–573.

Kim, Il-Koo, Junghye Min, Tammy Lee, Woo-Jin Han, e JeongHoon Park. «Block partitioning structure in the HEVC standard.» *IEEE transactions on circuits and systems for video technology* (IEEE) 22 (2012): 1697–1706.

Lainema, Jani, Frank Bossen, Woo-Jin Han, Junghye Min, e Kemal Ugur. «Intra coding of the HEVC standard.» *IEEE transactions on circuits and systems for video technology* (IEEE) 22 (2012): 1792–1801.

Lee, Shiann-Jong, Yu-Chang Chan, Dimitri Komatitsch, Bor-Shouh Huang, e Jeroen Tromp. «Effects of realistic surface topography on seismic ground motion in the Yangminshan region of Taiwan based upon the spectral-element method and LiDAR DTM.» *Bulletin of the Seismological Society of America* (Seismological Society of America) 99 (2009): 681–693.

Li, Shen, et al. «Pytorch distributed: Experiences on accelerating data parallel training.» *arXiv preprint arXiv:2006.15704*, 2020.

Liaw, Richard, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E. Gonzalez, e Ion Stoica. «Tune: A research platform for distributed model selection and training.» *arXiv preprint arXiv:1807.05118*, 2018.

Lloyd, S. «Least squares quantization in PCM.» *IEEE Transactions on Information Theory* 28 (1982): 129-137.

Magnoni, Federica, et al. «Spectral-element simulations of seismic waves generated by the 2009 l'Aquila earthquake.» *Bulletin of the Seismological Society of America* (Seismological Society of America) 104 (2014): 73–94.

Mazzieri, Ilario, Marco Stupazzini, Roberto Guidotti, e Chiara Smerzini. «SPEED: SPectral Elements in Elastodynamics with Discontinuous Galerkin: A non-conforming approach for 3D multi-scale problems.» *International Journal for Numerical Methods in Engineering* (Wiley Online Library) 95 (2013): 991–1010.

Meca Ondřej, Říha Lubomír, Jansík Branislav, Brzobohatý Tomáš. «Toward highly parallel loading of unstructured meshes.» *Advances in Engineering Software*, 2022.

«Mediate: Molecular docking at home.» *Mediate: Molecular docking at home.* s.d.

Miesch, Mark S., Allan Sacha Brun, Marc L. DeRosa, e Juri Toomre. «Structure and evolution of giant cells in global models of solar convection.» *The Astrophysical Journal* (IOP Publishing) 673 (2008): 557.

Misra, Kiran, Andrew Segall, Michael Horowitz, Shilin Xu, Arild Fuldseth, e Minhua Zhou. «An overview of tiles in HEVC.» *IEEE journal of selected topics in signal processing* (IEEE) 7 (2013): 969–977.

Nelson, Nicholas J., e Mark S. Miesch. «Generating buoyant magnetic flux ropes in solar-like convective dynamos.» *Plasma Physics and Controlled Fusion* (IOP Publishing) 56 (2014): 064004.

Nordlund, A. «Numerical simulations of the solar granulation. I-Basic equations and methods.» *Astronomy and Astrophysics* 107 (1982): 1–10.

Norkin, Andrey, et al. «HEVC deblocking filter.» *IEEE Transactions on Circuits and Systems for Video Technology* (IEEE) 22 (2012): 1746–1754.

numbox.it4i.cz. 2022.

«NVIDIA Nsight Compute Kernel Profiling Guide.» *NVIDIA Nsight Compute Kernel Profiling Guide.* s.d.

Paolucci, Roberto, Ilario Mazzieri, e Chiara Smerzini. «Anatomy of strong ground motion: near-source records and three-dimensional physics-based numerical simulations of the M w 6.0 2012 May 29 Po Plain earthquake, Italy.» *Geophysical Supplements to the Monthly Notices of the Royal Astronomical Society* (The Royal Astronomical Society) 203 (2015): 2001–2020.

Paris, Claudia, Lorenzo Bruzzone, e Diego Fernández-Prieto. «A novel approach to the unsupervised update of land-cover maps by classification of time series of multispectral images.» *IEEE Transactions on Geoscience and Remote Sensing* (IEEE) 57 (2019): 4259–4277.

Perri, Barbara, Allan Sacha Brun, Antoine Strugarek, e Victor Réville. «Dynamical coupling of a mean-field dynamo and its wind: Feedback loop over a stellar activity cycle.» *The Astrophysical Journal* (IOP Publishing) 910 (2021): 50.

Peter, Daniel, et al. «Forward and adjoint simulations of seismic wave propagation on fully unstructured hexahedral meshes.» *Geophysical Journal International* (Blackwell Publishing Ltd Oxford, UK) 186 (2011): 721–739.

Piljić, Igor, Leon Dragić, Alen Duspara, Mate Čobrnić, Hrvoje Mlinarić, e Mario Kovač. «Bolt65–performance-optimized HEVC HW/SW suite for Just-in-Time video processing.» *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO).* 2019. 966–970.

Sagan, H. *Space-Filling Curves.* 1994.

Springel, Volker and White, Simon DM and Tormen, Giuseppe and Kauffmann, Guinevere. «Populating a cluster of galaxies--I. Results at z= 0.» *Monthly Notices of the Royal Astronomical Society*, 2011: 726--750.

Sullivan, Gary J., Jens-Rainer Ohm, Woo-Jin Han, e Thomas Wiegand. «Overview of the high efficiency video coding (HEVC) standard.» *IEEE Transactions on circuits and systems for video technology* (IEEE) 22 (2012): 1649–1668.

Trott, Christian R., et al. «Kokkos 3: Programming model extensions for the exascale era.» *IEEE Transactions on Parallel and Distributed Systems* (IEEE) 33 (2021): 805–817.

Ugur, Kemal, et al. «Motion compensated prediction and interpolation filter design in H. 265/HEVC.» *IEEE Journal of Selected Topics in Signal Processing* (IEEE) 7 (2013): 946–956.

Volkov, Vasily. «Better performance at lower occupancy.» *Proceedings of the GPU technology conference, GTC.* 2010. 16.

Warnecke, Jörn, Petri J. Käpylä, Maarit J. Käpylä, e Axel Brandenburg. «Influence of a coronal envelope as a free boundary to global convective dynamo simulations.» *Astronomy & Astrophysics* (EDP Sciences) 596 (2016): A115.

Yang, Qiang, e Xindong Wu. «10 challenging problems in data mining research.» *International Journal of Information Technology & Decision Making* (World Scientific) 5 (2006): 597–604.

Yifang, Ban, Peng Gong, e Chandra Gini. «Global land cover mapping using Earth observation satellite data: Recent progresses and challenges.» *ISPRS journal of photogrammetry and remote sensing (Print)* (Elsevier) 103 (2015): 1–6.

You, Yang, Igor Gitman, e Boris Ginsburg. «Large batch training of convolutional networks.» *arXiv preprint arXiv:1708.03888*, 2017.

Zhang, Xiao, Liangyun Liu, Xidong Chen, Yuan Gao, Shuai Xie, e Jun Mi. «GLC_FCS30: Global land-cover product with fine classification system at 30 m using time-series Landsat imagery.» *Earth System Science Data* (Copernicus GmbH) 13 (2021): 2753–2776.