D1.1
Version 1

Document Information

Contract Number	828947
Project Website	www.enerxico-project.eu
Contractual Deadline	M8
Dissemination Level	Public
Nature	Report
Author	Okba Hamitou & Soline Laforêt
Contributor(s)	Rafael Mayo Garcia, Sebastian Wolf, Anne Reinarz, Michael Bader, Maurizio Hanzich, Claudia Rosas, Jaime Klapp
Reviewer	Albert Farres
Keywords	Efficiency; Optimization; HPC portability

Notice:
The research leading to these results has received funding from the European Union's Horizon 2020 Program under the ENERXICO Project (www.enerxico-project.eu), grant agreement no 828947 and under the Mexican CONACYT-SENER-Hidrocarburos grant agreement B-S-69926.

Change Log

Version	Author	Description of Change
0.1	Okba Hamitou	Initial draft
0.2	Soline Laforet	Executive summary, introduction, conclusion
0.3	Albert Farrès	Review
1	Okba Hamitou	Final

Table of Contents

1. Executive Summary

This deliverable presents performance audits of the codes within the Enerxico project, which collects all the technical information, test cases and results for each Enerxico code (listed below).

Its content gathers all mandatory inputs to identify what optimization/portability needs to be performed within this project, so that the "Exascale Enabling" work package (WP1) may reach the final targets.

From the Performance Audits these are the most relevant highlights per code:

1. ALYA: the main cause of efficiency loss is the load balancing issue. A shared memory parallel version of the function causing the load imbalance may mitigate this problem.
2. BSIT: the roofline analysis shows a low arithmetic intensity, L1 and L2 memory access should be improved to data reuse. GPU utilization shows that efficiency is bounded by the memory bandwidth.
3. WRF: there is an effective good scalability up to 32 processes. To use more processes, it is recommended to increase the test case size. In general, the application presents good results in terms of efficiency, scalability and load balance.
4. SeisSol: in previous audits and analysis, SeisSol shows excellent results in terms of node-level performance (on Intel architectures), scalability and load balance. The focus for performance optimization will therefore be on model extensions and on non-Intel architectures.
5. ExaHyPE: the analysis was performed by Atos due to the lack of support for hybrid MPI+TBB (Intel's Thread Building Blocks) programming paradigms in Extrae. The analysis has shown a good efficiency although some issues have been identified: cache stalls limit the performance of the application and the compiler was not able to vectorize the intensive parts of the code.
6. EM46: the first track of optimization is the improvement of a relatively low IPC (Instruction Per Cycle) for the second configuration (Full Waveform Inversion with all waves and one shot). The second track consist in the improvement of the MPI synchronizations by removing unnecessary barriers.
7. DualSPHysics: The Mexican partners have started working on the application with a 6-month delay. The POP's analysis of DualSPHysics is not yet completed. However, we have decided to share the preliminary information we have available for our use.

2. Introduction

A pre-work has been performed to list all the codes that partners intended to use for the scientific research on the work packages 2, 3 and 4.

This deliverable summarizes the performance evaluation of applications done in collaboration with the Performance Optimization and Productivity (POP[1]) Center of Excellence and using the toolset developed by POP partners and collaborators (Extrae[2] and Score-p[3]).

It is based on formal performance evaluation using POP's protocols and preliminary performance results focusing on the code readiness, and/or on the test cases and tools flexibility (i.e., Intel Threading Building Blocks TBB).

The document is composed of three main parts :

1. The first part lists the applications which will be analyzed and optimized in work package 1;
2. The second part covers for each application the following subtopics :
   - Code description, obtained from the surveys provided by the developers;
   - Known hardware and software dependencies;
   - Test cases to be evaluated during the performance audit;
   - Results of the initial evaluation;
   - Conclusion and next steps.

In this part, a detailed description such as the code version, the computing environment, the test case description, of each application is given. A precise analysis of the results is carried out to identify the main bottlenecks. Finally, for each application, a conclusion is given with the identified optimizations that shall be investigated.

3. The third part is a conclusion which highlights the overall main results of the analysis and summarizes the next steps to reach the project targets for work package 1:
   - **Tasks 1.1 "Intra-node optimization and portability"** targets to run correctly and efficiently codes on a single node, that may involve one or more of the following optimizations: memory utilization, thread parallelism, vectorization, optimizations at the algorithmic level, porting to accelerators and so on.
   - **Tasks 1.2 "Exploiting multi-node parallelism in Exascale architecture"** targets to exploit communication efficiency, parallel I/O capabilities and mitigate load balancing issues.

[1] Performance Optimisation and Productivity (https://pop-coe.eu/)
[2] Extrae Profiling Tool (https://tools.bsc.es/extrae)
[3] Scalable Performance Measurement Infrastructure for Parallel Codes (https://www.vihps.org/projects/score-p/)

- **Tasks 1.3 “Enabling computational and energy efficient codes for the Exascale”** focuses on constant performance evaluation, both at the computational and energetic level to guaranty more efficient applications.

3. Codes

Eight Codes have been identified for the project:

1. ALYA
2. BSIT
3. WRF
4. SeisSol
5. ExaHyPE
6. SEM46
7. DualPhysics

Code information specificity has been collected for each of them. Extract from POP reports have been reported when fitting the WP1 expectations.

4. Code Analysis

4.1 ALYA

4.1.1 Overview

Alya is a high-performance computational mechanics code to solve engineering coupled problems. The different physics solved by Alya are incompressible/compressible flow, solid mechanics, chemistry, particle transport, heat transfer, turbulence modeling, electrical propagation, among others.

Multiphysics coupling is achieved in a multi-code manner. MPI is used to communicate between the different instances of Alya, where each instance solves a particular physics, with the potential of performing asynchronous executions of the different physics. Alya is specially designed for massively parallel supercomputers.

Code description

Source Code Repository

- Available upon request

Version

- Version 2.0

Code Versioning Tool

- Currently migrating from SVN to Git
- Repository only accessible previously from CASE Department at BSC

Sanity Check / Unit Testing Framework

- Unit testing framework available

Documentation

- User and Developer documentation available at http://gitlab.bsc.es/alya/alya/wikis/home

Code Current Performance

- Alya has been widely evaluated and improved regarding scalability and parallel I/O. Potential further evaluations will be related to executing Alya on accelerators, such as GPUs.

Ongoing Research

- Contact
  - Guillaume Houzeaux
  - Mariano Vasquez
- Developer/Maintainers (ENERXICO related)
  - Daniel Mira: daniel.mira@bsc.es
  - Oriol Lehmkul: oriol.lehmkul@bsc.es
- Latest publications
  - Macià, Sandra & Mateo, Sergi & Martínez-Ferrer, Pedro & Beltran, Vicenç & Mira, Daniel & Ayguadé, Eduard. (2018).

Saiph: Towards a DSL for High-Performance Computational Fluid Dynamics. 1-10. 10.1145/3183895.3183896. ○ S. Gövert, D. Mira, M. Zavala-Ake, J.B.W. Kok, M. Vázquez, G. Houzeaux, (2017). Heat loss prediction of a confined premixed jet flame using a conjugate heat transfer approach, International Journal of Heat and Mass Transfer, Volume 107, 2017, Pages 882-894, ISSN 0017-9310, https://doi.org/10.1016/j.ijheatmasstransfer.2016.10.122. ○ Vázquez, Mariano & Houzeaux, Guillaume & Koric, Seid. (2014). Alya: Towards Exascale for Engineering Simulation Codes.

Software Requirements

Compiler and runtime
• Intel, GNU, Plefortran
External or Third-Party Libraries
• Metis for parallelism (mandatory)
Management tools
• Makefile
I/O Libraries
• HDF5
Tools/Libraries for the code workflow
• Available within Alya

Hardware Requirements

Node Level
• Alya has shown excellent performance in almost any architecture • GPUs are targeted
Network
• No special requirement
Storage
• No special requirement

4.1.2 Tests Conditions

Computational Environment Description
• Platform: MareNostrum IV • 2 x Intel Xeon Platinum 8160 24C at 2.1 GHz per node • 97 or 384 GB of memory per node • Storage: 200 GB local SSD • Interconnection network: 100 Gb Intel OmniPath

Test Case Description

- Methane/air premixed Bunsen flame in a confined domain at ambient conditions T= 300K and p=1 bar. The chemical kinetics is based on a GRI3.0 chemical scheme with 53 species and 325 reactions and the problem is solved with a finite rate chemistry model.
- Scale: from 8 to 768 MPI tasks
- Initial set-up: 50 iterations, reduced to 7 iterations.

4.1.3 Results

Application structure

- First analysis with 50 iterations and 48 MPI
  - Phases clearly identified in the tracefile: initialization and iterative computation,
  - Great density of communications, better focus on less iterations.
- The application is structured with a master task that does not perform useful computations; it communicates with all the processes using MPI collectives.
- The Useful Duration view suggests a load balance problem, some processes have more density of useful computation than the rest.

Focus of Analysis

- We identify a large time inside MPI communications (around 50%)
  - Very low useful computation time, in average, half of the time inside MPI,
  - Big difference between useful computation time among processes.
- Difference between minimum (19.31%) and maximum (80.84%) and the value of standard deviation (10.87%) suggest a load balance problem.
- Maximum is not ideal. It suggests that there is space for improvement.
- The low value of useful computation time is due to load imbalance. The tasks with less work remain in MPI calls waiting to others.

Scalability

- Bad scalability.
- Speed up below 5 for 8 node (384 MPI ranks) runs.

Efficiency analysis

- The main factor limiting the scalability identified by the efficiency analysis is load imbalance
  - 75.82% for 8 MPI tasks,
  - 35.75% for 768 MPI tasks.
- When scaling the number of nodes, the transfer inefficiency has a low value
  - 99.88% for 8 MPI tasks,
  - 82.25% for 768 MPI tasks.

- The IPC of the application is reduced when the node is full of processes
  - 100% for 8 MPI tasks,
  - 93.59% for 48 MPI tasks,
  - 92.90% for 768 MPI tasks.
- The number of useful instructions required to solve the same problem increases substantially when the number of processes increases.

Load balance

- Instructions are the leading cause of the load imbalance.
- This means that the work is not getting distributed between the processors evenly.
- The load balance of the application is better than the instruction balance
  - Processors that have more work do not share node resources and thus, have a better IPC.

4.1.4 Conclusion

Summary

- The main loose of efficiency is due to load balance
- Load imbalance can be addressed from an algorithmic point of view. This should be the main target for next code optimizations.
- For further analysis on the communication efficiency and instruction scalability, we recommend repeating the analysis on a bigger input set.

Next steps

- We suggest implementing a shared memory parallel version of the function causing the load imbalance using OmpSs or OpenMP and finally apply Dynamic Load Balancing (DLB).

4.2 BSIT

4.2.1 Overview

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

Geophysical exploration is a field with huge amount of computational resources needs. BSIT platform was developed to cope with such needs, including different type of processing systems running over a wide range or HPC architectures. The main systems included in BSIT are forward modelling, reverse time migration and full waveform inversion.

In addition, the software supports different rheologies including acoustic, acoustic with variable density, elastic, viscoelastic and electromagnetic. Moreover, several levels of anisotropy are supported: VTI/HTI, Orthorhombic, TTI and arbitrary anisotropy (for elastic and viscoelastic rheologies).

Code description

Source Code Repository
• Repository only accessible previously from CASE Department at BSC
Version
• Version 2018.12 property of Repsol S.A.
Code Versioning Tool
• Currently migrating from SVN to Git • Repository only accessible previously from CASE Department at BSC
Sanity Check / Unit Testing Framework
• Unit testing • Integration testing • Regression testing • Testing framework: STEEL (In-house development)
Documentation
• Within BSIT's repository
Code Current Performance
• Measured performances ○ Previous performance analyses have been carried out on General Purpose architectures, such as Intel Skylake, and on accelerators (e.g. Intel Knight Landing and Nvidia K80) • Recognized bottlenecks ○ Memory bottlenecks: FSG (Full Staggered Grid) is memory expensive and will be conditioned by the total memory available in the node. ○ Lack of scalability: We have observed efficiency degradation starting from 4 CPUs, further evaluation with more than one node is pending. ○ Compute/communication ratio in GPUs (pending to evaluate).
Ongoing Research

- Contact
  - Albert Farrés albert.farres@bsc.es
- Developer/Maintainers (ENERXICO related)
  - Mauricio Hanzich
  - Josep de la Puente
  - Natalia Gutierrez
  - Juan Esteban Rodriguez
  - Albert Farrés
  - Claudio Marquez
  - Miguel Ferrer
  - Samuel Rodriguez
  - Jean Corman
  - Raúl de la Cruz
  - Genis Aguilar
- Latest publications
  - Albert Farrés, Claudia Rosas, Mauricio Hanzich, Marc Jordà and Antonio Peña. (2019). "Performance Evaluation of Fully Anisotropic Elastic Wave Propagation on NVIDIA Volta GPUs". 81st EAGE Conference and Exhibition. DOI: 10.3997/2214-4609.201901307
  - Albert Farrés, Claudia Rosas, Mauricio Hanzich, Charles Yount, and Alejandro Duran. (2018). "Performance optimization of fully anisotropic elastic wave propagation on 2nd Generation Intel Xeon Phi processors". The 19th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing. Vancouver, Canada. 21-25 May 2018. ISBN: 978-1-5386-5555-9. DOI: 10.1109/IPDPSW.2018.00158
  - Matheus S. Serpa, Eduardo HM Cruz, Matthias Diener, Arthur M. Krause, Philippe O.A. Navaux, Jairo Panetta, Albert Farrés, Claudia Rosas, and Mauricio Hanzich (2019). Optimization strategies for geophysics models on manycore systems. The International Journal of High-Performance Computing Applications, 33(3), 473–486. https://doi.org/10.1177/1094342018824150

Software Requirements

Compiler and runtime
• Intel, GNU, IBM XL, CUDA version: 10.1, CUDA driver: 418.39
External or Third-Party Libraries
• Mandatory ○ LAPACK ○ FFTW ○ OpenMP (included in the compiler) • Optional ○ MKL ○ PETSc
Management tools

- Makefile

I/O Libraries

- None

Tools/Libraries for the code workflow

- Available within BSIT

Hardware Requirements

Node Level

- Memory may vary depending on the problem size
- GPU: Nvidia V100

Network

- Mellanox EDR

Storage

- > 200 GB

4.2.2 Tests Conditions

Computational Environment Description

CTE-POWER is a cluster based on IBM Power9 processors, with a Linux Operating System and an Infiniband interconnection network.

It has the following configuration:

- 2 login node and 52 compute nodes, each of them:
- 2 x IBM Power9 8335-GTH @ 2.4GHz (3.0GHz on turbo, 20 cores and 4 threads/core, total 160 threads per node)
- 512GB of main memory distributed in 16 DIMMs x 32GB @ 2666MHz
- 2 x SSD 1.9TB as local storage
- 2 x 3.2TB NVME
- 4 x GPU NVIDIA V100 (Volta) with 16GB HBM2.
- Single Port Mellanox EDR
- GPFS via one fiber link 10 GBit

The operating system is Red Hat Enterprise Linux Server 7.5 alternative.

Test Case Description

Tests to be covered are 3D Elastic Fully Staggered Grid:

- **Case A (small): unity test (seconds/minutes)**
  This test describes a homogeneous half space which is excited by a point explosion in a configuration known as Garvin's problem. This test compares the results of a simulation running with a mimetic free-surface with a 6 ppw configuration against an analytical solution obtained with the code EX2DVAEL by Berg and If (1994). The receivers are spaced 100m along the x-direction.
- **Case B (medium/large): production chain (hours)**
  The main goal of this test set is to evaluate the elastic propagation of the modeling PoC+kernel case for isotropic materials. The test runs an

Overthrust campaign of 72 shots. Main Parameters:
- Force source (1.0, 0.0, 0.0) with a peak frequency of 5 Hz.
- Boundary condition used is "sponge" with a 40-point halo width.
- Mimetic free surface used.
- Four seismogram channels produced at 0.001 seconds interval.
- No illumination nor snapshots are produced.
- The FSG propagation kernel is being in use for all the HPC environments.
- Maximum Z value: 3180.0 mts.
- 72 shots campaign - Line 37
- Ricker wavelet from file.
- Source peak frequency at 10 Hz
- Grid: 440x2201x2201 with dz = 7.24, dx = dy = 7.26 m.

4.2.3 Results

Application structure

- From approximately 5 seconds of total execution time with 8 computing threads:
    - It is easy to identify all the processing phases: from initialization to iterative computation and finalization,
    - There is some serialization in the communications phase, better focus on a few iterations, e.g. 8 time steps.
- The implementation of this algorithm relies on all-to-all communications between all the processing threads that end up serialized.

There are no apparent unbalances but is clearly highlighted a synchronization when communication from CPU (host) to GPU.

Focus of Analysis

Our base implementation is the direct result of developing the finite differences method over a Full Staggered Grid (FSG). It creates a loop in time to update velocities based on stresses values in odd iterations and the other way around for even iterations.

To update velocities, 12 different 3D stencils plus another 12 3D material interpolations for each point of the grid must be calculated. Materials are stored in a single vertex of the FSG cell for memory saving issues, trading storage per computation. On the other hand, the computation involves 28 3D-stencils computation plus 84 3D interpolations for the material properties to update stresses. Notice that both velocities and stresses calculations are typically dominated by accesses to global memory to retrieve the data needed to update the corresponding values.

Our baseline version of the code has the two innermost loops in space mapped to a 2D Cuda grid, streaming cells to update to each thread over the slowest dimension (Y)

Efficiency analysis
After observing the arithmetical intensity of the kernel in this architecture, and despite stencil calculations are usually limited by memory bandwidth, we observe similar efficiencies at both L2 and High Bandwidth memory. The roofline model shows that at that it will be necessary to improve that arithmetic intensity to achieve higher FLOPS.

GPU Utilization
Low Global Memory Efficiency: kernels accounting for 40.6% of compute have low efficiency (72.7% in average) For Tesla V100-sMX2-16GB the kernel's compute utilization is significantly lower than its memory utilization. These utilization levels indicate that the performance of the kernel is most likely being limited by the memory system. For this kernel, the limiting factor in the memory system is the bandwidth of the Device memory. Global load efficiency indicates how well the application global loads are using device memory bandwidth. The efficiency is the number of bytes requested divided by the number of bytes that were transferred from device memory to satisfy request. Because device memory transfers bytes in blocks, the alignment and the access pattern of a given load determines how many blocks must be transferred and thus determines the efficiency of that load. Low efficiency indicates that one or more global memory loads have a poor access pattern or alignment**.**

MPI analysis
• On average, approximately a 49.10% of total MPI Calls time is invested in Waiting. • Despite having implemented asynchronous MPI calls, threads communications end up complete serialized.

4.2.4 Conclusion

Summary

- The main loss of efficiency was due to poor scalability
    - Due to synchronous data transfer between host and device
    - This issue has been fixed and scalability is now good
- Roofline analysis shows a low arithmetic intensity (common in stencil codes)
- GPU utilization analysis confirms that performance is bounded by the memory system
    - Limiting factor: Bandwidth in device memory

Next steps

- L1 and L2 memory access should be improved to reuse data and increase the arithmetic intensity.

- If arithmetic intensity is increased there is room for a performance (GFLOPS) improvement
- Efficiency can be increased by fixing data alignments and improving memory access patterns

4.3 WRF

4.3.1 Overview

WRF is a mesoscale numerical weather prediction system designed for both atmospheric research and operational forecasting applications. It features two dynamical cores, a data assimilation system, and a software architecture supporting parallel computation and system extensibility. The model serves a wide range of meteorological applications across scales from tens of meters to thousands of kilometers. The effort to develop WRF began in the latter 1990's and was a collaborative partnership of the National Center for Atmospheric Research (NCAR), the National Oceanic and Atmospheric Administration (represented by the National Centers for Environmental Prediction (NCEP) and the Earth System Research Laboratory), the U.S. Air Force, the Naval Research Laboratory, the University of Oklahoma, and the Federal Aviation Administration (FAA).

For researchers, WRF can produce simulations based on actual atmospheric conditions (i.e., from observations and analyses) or idealized conditions. WRF offers operational forecasting a flexible and computationally efficient platform, while reflecting recent advances in physics, numeric, and data assimilation contributed by developers from the expansive research community. WRF is currently in operational use at NCEP and other national meteorological centers as well as in real-time forecasting configurations at laboratories, universities, and companies.

WRF has a large worldwide community of registered users (a cumulative total of over 48,000 in over 160 countries), and NCAR provides regular workshops and tutorials on it. The WRF system contains two dynamical solvers, referred to as the ARW (Advanced Research WRF) core and the NMM (Nonhydrostatic Mesoscale Model) core. The ARW has been developed in large part and is maintained by NCAR's Mesoscale and Microscale Meteorology Laboratory, and its users' page is: WRF-ARW Users' Page. The NMM core was developed by the National Centers for Environmental Prediction (NCEP) and is currently used in their HWRF (Hurricane WRF) system.

Code description

Source Code Repository
• https://github.com/wrf-model/WRF/releases • https://www2.mmm.ucar.edu/wrf/users/wrfv4.1/updates-4.1.html • https://www2.mmm.ucar.edu/wrf/users/downloads.html
Version

• WRF V4.2
• Tests are focused on the following two modules included in the main WRF module:
  o PBL: YSU
  o SL: Revised MM5 surface layer scheme

Code Versioning Tool
The recommended method is to clone the code from public GitHub repository: • git clone https://github.com/wrf-model/WRF
Numerical scheme(s) used and implementation issues:

- **Short description:**

The weather research and forecast models have a lot of numerical schemes. The schemes are available at the following web page: http://www2.mmm.ucar.edu/wrf/users/docs/user_guide_v4/v4.1/users_guide_chap5.html#Phys

- **Specificity:**

The only implementation issues are the incompatibility between numerical schemes

Sanity Check / Unit Testing Framework

- https://www2.mmm.ucar.edu/wrf/users/wrfv4.0/testing.html

Documentation

- https://github.com/wrf-model/WRF/wiki
- https://www2.mmm.ucar.edu/wrf/users/pub-doc.html

Code Current Performance

There is a POP performance study of WRF that will be used as a baseline for the new performance characterization planned within Enerxico

Ongoing Research

Contacts :

- Jorge Navarro (jorge.navarro@ciemat.es);
- Rafael Mayo-García (rafael.mayo@ciemat.es)

Ongoing research

- Wind energy, Downscaling, Climate analysis

Latest publications

- Multidecadal to centennial surface wintertime wind variability over Northeastern North America via statistical downscaling. E. Lucio-Eceiza et al. CLIMATE DYNAMICS 53 1-2, 41-66 (2019)
- Quality Control of Surface Wind Observations in Northeastern North America. Part I: Data Management Issues. E. Lucio-Eceiza et al. JOURNAL OF ATMOSPHERIC AND OCEANIC TECHNOLOGY 35 1, 163-182 (2018)
- Quality Control of Surface Wind Observations in Northeastern North America. Part II: Measurement Errors. E. Lucio-Eceiza et al. JOURNAL OF ATMOSPHERIC AND OCEANIC TECHNOLOGY 35 1, 183-205 (2018)

Software Requirements

Compiler and runtime

- FORTRAN 90 or 95 and C,
- Perl,
- OpenMP,
- MPI RSL-LITE

External or Third-Party Libraries

- **Mandatory**
    - netCDF-4,

-
  - PHD5,
  - GriB-1,
  - NCL,
  - RIB4,
  - ARWpost
- **Optional**
  - PNETCDF
  - netCDF-4 (such as parallel I/O based on HDF5)
  - JasPer (an implementation of the JPEG2000 standard for "lossy" compression)
  - PNG (compression library for "lossless" compression)
  - zlib (a compression library used by the PNG library)

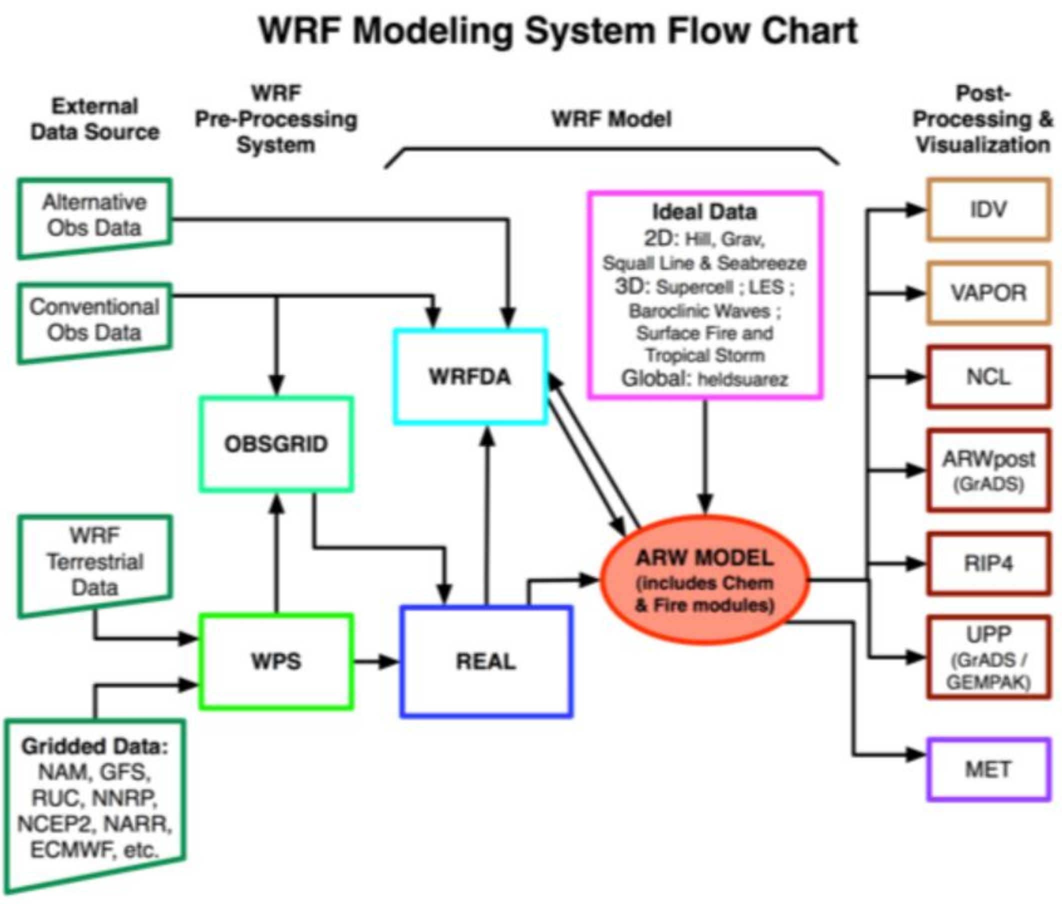
Management tools

- C-shell and Bourne shell, make, M4, sed, awk

I/O Libraries

- HDF5
- netCDF

Tools/Libraries for the code workflow


WRF Modeling System Flow Chart
External Data Source
WRF Pre-Processing System
WRF Model
Post-Processing & Visualization
Alternative Obs Data
Conventional Obs Data
WRF Terrestrial Data
Gridded Data: NAM, GFS, RUC, NNRP, NCEP2, NARR, ECMWF, etc.
OBSGRID
WPS
WRFDA
REAL
Ideal Data 2D: Hill, Grav, Squall Line & Seabreeze 3D: Supercell ; LES ; Baroclinic Waves ; Surface Fire and Tropical Storm Global: heldsuarez
ARW MODEL (includes Chem & Fire modules)
IDV
VAPOR
NCL
ARWpost (GrADS)
RIP4
UPP (GrADS / GEMPAK)
MET

Hardware Requirements

Architectures/Machines Characterization

Initially, WRF will be executed on ACME, a dedicated local cluster for R&D

purposes composed of

- 14 PowerEdge C6420 nodes composed of:
  - 2 Intel Gold 6138 processor (20C/40T) @2,0 GHz
  - 192 GB RDIMM, 2667MT/s
  - IB FDR port
  - 2TB SATA @7.2 krpm and 240GB SSD SATA
- 2 Bullx R424-E4 chassis with 4 computing nodes each
  - 2 Intel Xeon 8C processors E5-2640 V3 @2,6 GHz
  - DDR4 memory of 64 GB @2133 MHz
  - IB FDR port
  - 1 TB SAS2 disc @7.2 krpm and 1 SSD of 240 GB
- 2 Bullx R421-E4 chassis
  - 2 Intel Xeon 8C processors E5-2640 V3 @2,6 GHz
  - DDR4 memory of 64 GB @2133 MHz
  - 1 TB SATA III disc @7.2 krpm and 1 SSD of 240 GB
  - 2 Tesla P100
- 1 storage server composed of 48 disks (168 TB raw storage)
- Rpeak = 59.40 TFlops

Defined according to the current CIEMAT cluster:

- 1 to 44 nodes PowerEdge C6420 with 2 Xeon Gold 6148 2,4Ghz 40 cores/node
- ram: 192 GB/node
- Infiniband EDR 100Gb/s
- Storage: 28 x disks 12TB 7.2K NLSAS 12GB 3,5”

4.3.2 Tests conditions

Computational Environment Description

1 to 44 nodes PowerEdge C6420 with 2 Xeon Gold 6148 2,4Ghz 40 cores/node
ram: 192 GB/node
Infiniband EDR 100Gb/s
Storage: 28 x disks 12TB 7.2K NLSAS 12GB 3,5”

Test Case Description

Strong scalability test:

- from1 up to 675 cores (it continuously crashed for the larger option)

Weak scalability test:

- 1.0·105 and 2.5·105 points per domain and core. Cores running from 20 to 630

4.3.3 Results

Application structure

The spatio-temporal structure of the run shows a pattern that is repeated 6 times in the iterative region.

When we look at one iteration, the first part of the iteration corresponds to a region with an average of more than 24700 calls to MPI_Comm_rank() per process (in the whole iteration the number is around 25000). Despite the overhead to call the library to get the rank would be small, accessing to a local variable that stores the value would eliminate all these library calls.

After discarding the initial phase of the iteration, we focused the analysis on a region that reflects the most common behavior. The profile of both the full iteration and the selected region validated the similarity with respect to the percentage of time on the different MPI calls (except for MPI_Comm_rank). We also validated the main efficiencies report very similar values with differences lower than 1.5%.

The application shows a good granularity, where most of the time the computations last between tens of milliseconds up to 400 milliseconds.

Efficiency analysis

The efficiencies analysis allows to identify the observed problems on the scalability are correlated with serialization, transfer and computational scalability. On the other hand, the efficiency with 16 MPI processes is 80.5% and it is mainly caused by serialization and also a little of global load imbalance. We can consider that metrics above 80% report a good performance, while values under that threshold identify factors that are limiting the execution.

The computation scalability is determined by the number of instructions and the instructions per cycle (IPC) in the computational phases.

The total number of instructions executed by the computations increases a with the scale, suggesting a potential code replication or increase of instructions due to the higher number of boundary cells when increasing the scale. The efficiencies analysis also detects an improvement on the IPC with 128 MPI ranks that compensates the increase on instructions making a small improvement of the computation when comparing 32 and 128 ranks.

<u>Efficiencies for 13, 32, 128 MPI ranks</u>

Global efficiency:	80.5/62.9/53.2
Parallel efficiency:	80.5/76.9/64.6
Load Balance:	92.2/92.4/92.3
Communication eff. :	87.3/83.3/69.9
Serialization	89.5/87.7/81.4
Transfer	97.6/95.0/85.9
Computation scalability	100/81.7/82.4
IPC Scaling efficiency	100/75.4/113.2
Instructions Scaling efficiency	100/96.3/72.5

Scaling analysis
The speed-up plot reports the scalability of the code has a small degradation with 32 processes and a significant reduction with 128. In a perfectly linear strong scaling execution, we expect that each time the number of processes doubles, the total execution time per iteration reduces by half. Going from 16 to 128 we increase the resources by 8 and the execution obtains a 5.28x improvement (66% of the ideal speedup).

MPI analysis
The efficiencies and scalability analysis identified that the communications are the main bottleneck for the instrumented runs. The analysis also reports the main loose of efficiency is due to the serialization and dependencies between processes. The transfer of data that reported a good efficiency with 16 tasks shows relevant reductions when the scale is increased. MPI_Alltoall : imbalance MPI_Alltoallv : transfer MPI_Wait : both (imbalance and transfer) improving the imbalance of the previous region would reduce the MPI_Wait time for these calls

Computing performance
This section analyses the efficiency and scalability of the WRF runs with respect to the execution of the serial computing regions. The efficiencies analysis identified the computation suffer a significant degradation mainly correlated with the total number of instructions executed. It also identified an improvement of the IPC with 128 MPI ranks as well as a reduction with 32 ranks. With 32 processes, the IPC for almost all the regions is degraded while it improves with 128. The improvement with 128 cores may be expected due to the strong scaling approach that can improve cache access with a larger scale. It is more surprising the reduction of the IPC with 32 cores. With the available counters information collected there is not a clear justification of the cause, but it can be deeply analyzed in a performance plan if the user is interested. The IPC achieved by the other runs (16 and 128) are considered reasonable as they are bigger than 1 instruction per cycle for all the regions with a good IPC between 1.8 and 2.1 for the largest computing region. The trace files report that the increase of instructions is spread in most of the computations with an important increase on the largest and unbalanced computing phase.

4.3.4 Conclusion

Summary

The parallel efficiency achieved is good for 16 and 32 processes but not for 128

(64.6% means the code spends close to 36% in the parallel runtime). It is recommended to use less than 128 processes or to increase the input case.

The observed loose of efficiency is due to the serialization and temporal unbalances of the code that increase with the scale. The analysis identified different regions with temporal unbalances as well as how the imbalance of the largest computing region is absorbed by the point to point communications.

The global balance of the code is good, and it does not increase with the scale in the range of MPI ranks analyzed.

The IPC achieved is good with 16 ranks and improves with 128 but it decreases with 32. There is no justification for this decrease that can be deeply studied on a Performance Plan if the user is interested.

The code has around 25000 calls to MPI_Comm_rank() per process on each iteration. Despite the overhead is small, checking a local variable is an easy solution to eliminate the library calls.

Next steps
Reduce the number of calls to MPI_Comm_Rank().

4.4 SeisSol (Earth and Atmospheric Sciences)

1.1.1 Overview

SeisSol is a software package for simulating wave propagation and dynamic rupture based on the arbitrary high-order accurate derivative discontinuous Galerkin method (ADER-DG). Characteristics of the SeisSol simulation software are:

- use of arbitrarily high approximation order in time and space (ADER-DG with Godunov flux formulation)
- use of tetrahedral meshes to approximate complex 3D model geometries (faults & topography) for rapid model generation
- use of elastic, viscoelastic and viscoplastic material to approximate realistic geological subsurface properties
- parallel geo-information input (ASAGI)

Code description

Source Code Repository
• SeisSol is available at https://github.com/SeisSol/SeisSol.
Version
• **Rolling release mode**
Code Versioning Tool
• Git and GitHub
Sanity Check / Unit Testing Framework
• SeisSol is regularly verified against community benchmark cases (e.g. LOH1, TPV12/13) • Cxxtest • Travis-ci SeisSol Autotuning Proxy (performance reproducer)
Documentation
• https://seissol.readthedocs.io/en/latest/
Code Current Performance
• **Measured performances** ○ Currently, SeisSol uses handwritten, embedded flop and time counters. Additionally, Score-P has been recently adapted for profiling our SeisSol proxy application. Furthermore, a report from POP already existed regarding the strong scaling properties of the code. • **Recognized bottlenecks** ○ SeisSol uses a hybrid architecture with one MPI rank per node and OpenMP parallelization within each node. A dedicated communication thread has allowed to improve the scaling properties.

    - SeisSol features cluster-wise local time stepping. In the strong-scaling limit small cluster sizes limit shared-memory scalability.
- **Memory bottlenecks**
    - none
- The POP audit showed good strong scalability up to 128 nodes (audit was performed for comparably small problem sizes). SeisSol weakly scales up to full machine runs on SuperMUC.

Ongoing Research

- **Contact for ENERXICO related inquiries**
    - Sebastian Wolf (wolf.sebastian@in.tum.de)
- **Ongoing research**
    - Advanced elasticity models
    - Elastic – Acoustic coupling,
    - port to CUDA, not in the scope of ENERXICO but within the ChEESE Center of Excellence
- **Latest publications**

Computational

- [1] **A. Breuer, A. Heinecke, and M. Bader**. "Petascale Local Time Stepping for the ADER-DG Finite Element Method". In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS). 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS). May 2016, pp. 854–863.
- [2] **A. Heinecke et al**. "Petascale High Order Dynamic Rupture Earthquake Simulations on Heterogeneous Supercomputers". In: SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Nov. 2014, pp. 3–14.
- [3] **Carsten Uphoff et al.** "Extreme Scale Multi-physics Simulations of the Tsunamigenic 2004 Sumatra Megathrust Earthquake". In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '17. event-place: Denver, Colorado. New York, NY, USA: ACM, 2017, 21:1– 21:16.

Scientific

- [4] **Thomas Ulrich, Alice-Agnes Gabriel, et al.** "Dynamic viability of the 2016 Mw 7.8 Kaikoura earthquake cascade on weak crustal faults". In: Nature Communications 10.1 (Mar. 14, 2019), pp. 1–16.
- [5] **Thomas Ulrich, Stefan Vater, et al.** Coupled, Physics-based Modeling Reveals Earthquake Displacements are Critical to the 2018 Palu, Sulawesi Tsunami. preprint. EarthArXiv, Feb. 16, 2019.
- [6] **Stephanie Wollherr, Alice-Agnes Gabriel, and Carsten Uphoff.** "Off-fault plasticity in three-dimensional dynamic rupture simulations using a modal Discontinuous Galerkin

	method on unstructured meshes: implementation, verification and application". In: Geophysical Journal International 214.3 (Sept. 1, 2018), pp. 1556–1584.

Software Requirements

Compiler and runtime
• Intel, GNU
External or Third-Party Libraries
• Python/NumPy (for code generation) • parmetis • libxsmm • PspaMM (for Knights Landing or Skylake) • MPI • asagi (optional)
Management tools
• Scons • Travis-ci
I/O Libraries
• Netcdf • HDF5
Tools/Libraries for the code workflow
• Paraview (visualization) • gmsh (meshing for simple cases) • simmetrix (parallel meshing for advanced geometries)

Hardware Requirements

Architectures/Machines Characterization
The typical use case are simulations with 10 to 20 million cells. On SuperMUC-NG this requires 50 to 100 nodes. Landmark simulations may use more than 200 million cells (with >100 Billion unknowns) on several 1000 compute nodes. Currently SeisSol runs on CPU based clusters, e.g. SuperMUC-NG at LRZ or MareNostrum 4 at BSC. Nevertheless, for small meshes SeisSol also runs on consumer hardware. The memory requirements depend on the mesh size and on the chosen discretization order. For a mesh with 7 million cells at least 4 nodes on SuperMUC Phase 2 (i.e. 58 GB of RAM per node) are needed. In general, a lack of memory is not a problem. As part of the ChEESE project (https://cheese-coe.eu/) SeisSol is being ported to run on GPU based architectures.

1.1.2 Tests conditions

The POP audit for the following results was performed within the ChEESE Centre of excellence. Extensive performance data is also available from SuperMUC and SuperMUC-NG. In ENERXICO we thus decided to not invest further work into additional performance analysis.

Computational Environment Description

- Platform: MareNostrum IV
- 2 x Intel Xeon Platinum 8160 24C at 2.1 GHz per node
- 97 or 384 GB of memory per node
- Storage: 200 GB local SSD
- Interconnection network: 100 Gb Intel OmniPath
- Intel Compiler and MPI (2017.4)

Test Case Description

- Test were performed on a scenario for the simulation of the 2004 sumatra earthquake.
- mesh with 51,020,237 cells and 8,766,031 vertices:
- Scale:
  - from 192 to 18432 cores (4 to 384 compute nodes)
  - each compute node with a single MPI rank, 47 OpenMP threads & 1 or 4 Pthreads
- Score-P measurements of dedicated commThread build & runs directed by client
  - Pthread instrumentation plus manually-instrumented source regions

1.1.3 Results

Application structure

First analysis with the small input case

- 4 MPI ranks
- each with 47 runtime OMP threads (captured as parent-less orphans) plus ‘commThread’ Pthreads bound to additional remaining core per node
- 4 Pthreads (3 for MPI file I/O, and 1 for MPI communication)
- only 1 Pthread (for MPI comm.) since file writing disabled

Phases clearly identified in the trace file:

- Initialization/setup
- Simulate phase (Simulator::simulate)

Efficiency analysis

Efficiencies are computed for runs with 4, 8, 16, 32, 64, 128, 192, 256, 384 nodes

- Global efficiency initially very good, and degrades relatively slowly with scale
    - Max/Min efficiencies : 0.95/0.72
- Small Load balance efficiency remains very good with scale
    - Max/Min efficiencies : 0.96/0.92
- Very good Communication efficiency (including OpenMP synchronization)
    - Max/Min efficiencies : 1.00/0.80
- Computation efficiency relatively good
    - Max/Min efficiencies : 1.00/0.77
- IPC efficiency relatively good
    - Max/Min efficiencies : 1.00/0.85
- Very good Instructions scaling
    - Max/Min efficiencies : 1.00/0.97
- Useful IPC very good
    - Max/Min : 1.60/1.89

Scaling analysis

Strong scaling

- Medium-sized testcase (as previously)
- Simulate time of test case for 2 time-steps
- Scaling relative to the smallest configuration with 4 compute nodes
- 1 MPI process per compute node
    - Original with 48 OpenMP threads
    - Dedicated commThread version with47 OpenMP threads (bound)
    - Dedicated commThread version w/o file writing
- Original shows performance benefit to 128 nodes, but scales well only up to 16 nodes
- Somewhat faster with dedicated core for commThread and much improved scaling to 128 nodes and beyond
- File writing largely amortized to 128 nodes, but significant beyond that

Strong scaling relative speedup

- 80% scaling efficiency originally sustained to 32 compute nodes
- 80% scaling efficiency extended to 128 compute nodes using dedicated commThread

MPI analysis

- Computation time increases very slowly with scale
- MPI communication and synchronization are relatively small
- OpenMP implicit barrier synchronization at end of parallel for loops grows progressively
- Slow growth with scale of idle threads/cores until 192 nodes, then somewhat larger

I/O analysis

Medium-sized testcase (as previously)

- File writing failed for <64 nodes using commThread

commThread file writing results in roughly 50 seconds slowdown of simulate time

- Time for commThread file writing of around 150 seconds

Dedicated commThread apparently overloaded

1.1.4 Conclusion

Summary

- commThread variant has notably better performance and scalability than original
    - MPI communication and particularly file I/O ‘offload’ to POSIX threads bound to a dedicated core more than compensates for one less OpenMP thread
- Scaling significantly improved with 80% scaling efficiency to 128 compute nodes
- File writing impacts the rest of the execution, reducing performance and scalability, most significantly for more than 128 compute nodes
    - lowers otherwise good Global & Communication efficiencies and deteriorates with scale
    - impacts otherwise excellent Load balance efficiency, with significant phases outside of OpenMP parallel regions when associated cores are idle
- The audit setup was artificially I/O heavy, to over-emphasize I/O bottlenecks.
- Computation efficiency and IPC relatively good

Next steps

- Performance optimization will concentrate on model extensions and on non-Intel architectures.

Disclaimer

- *The use of MPI_THREAD_MULTIPLE is not supported by performance tools*
    - *where measurements are provided, they should be considered unreliable (until proven otherwise)*
- *The use of OpenMP and Pthreads in combination is not supported*
    - *measurement of all threads as Pthreads may be possible, with OpenMP runtime threads captured as parent-less “orphan” threads*
    - *some or all OpenMP thread activity/events may be lost*
    - *manual instrumentation of OpenMP has been done to recover some activity but may be incomplete*
    - *Efficiency metrics are likely to be unreliable particularly with respect to multiple threads bound to a single core*

4.5 ExaHyPE

4.5.1 Overview

The ExaHyPE engine solves systems of hyperbolic PDEs, as stemming from conservation laws. A concrete model for seismic wave propagation problems has been developed within the ExaHyPE project and is being further developed in the ChEESE cluster of excellence. ExaHyPE is based on high-order Discontinuous Galerkin (DG) discretization on tree-structured Cartesian meshes. For non-linear problems it offers an a-priori Finite-Volume limiter. Parallelisation of the ExaHyPE engine relies on the underlying Peano framework for parallel adaptive mesh refinement. MPI is used for distributed-memory parallelism. For shared-memory parallelization, Intel's TBB is used.

Code description

Source Code Repository
• ExaHyPE is available at https://gitlab.lrz.de/exahype/ExaHyPE-Engine
Version
• Rolling release mode • Stable version available
Code Versioning Tool
• Gitlab
Sanity Check / Unit Testing Framework
• Jenkins • Benchmarks for the main two application areas (seismology and astrophysics)
Documentation
• Guidebook: http://www.peano-framework.org/exahype/guidebook.pdf • Doxygen (class documentation): http://www.peano-framework.org/exahype/ • Release paper: https://arxiv.org/abs/1905.07987
Code Current Performance
• **Measured performances**   ○ The performance of the curvilinear mesh benefits from higher order (with a sweet spot at order 7)   ○ The curvilinear method reaches around 100 Mdof/s on a single Intel Skylake node   ○ Speedup and performance are almost independent of the order for the diffuse interface method   ○ Peano's geometrical domain-decomposition has load balancing sweet-spots at 1, 28 and 731 ranks. Thus, best scaling can be obtained via hybrid (MPI+TBB) parallelism: exploit MPI sweet spots and increase number of TBB-Threads per rank

- Weak scaling can be obtained up to 731 Skylake nodes of SuperMUC-ng (approx. 2^15 cores)

- **Recognized bottlenecks**
  - At a high parallelization level, the serial task production cannot be hidden behind the consumption
  - Shared Memory: Riemann solves still sequential
- **I/O limitations (serial)**
  - IO was not tested
- **Memory bottlenecks**
  - None known
- **Lack of scalability**
  - Shared Memory scalability stagnates at around 14 cores
  - Distributed memory scaling only available at certain "sweet spots", i.e. at 1,28, 731… ranks. This can be somewhat mitigated by a hybrid parallelization strategy.
  - Strong scaling only in very small regimes

Ongoing Research

- **Contact**
  - Anne Reinarz, Technical University of Munich (reinarz@in.tum.de)
- **Ongoing research**
  - Code-generation and vectorization: Jean-Matthieu Gallard (TUM, ChEESE-COE)
  - Dynamic Rupture: Leonhard Rannabauer (TUM, ChEESE-COE)
- **Latest publications**
  Computational:
  - [1] **J.-M. Gallard, L. Krenz, L. Rannabauer, A. Reinarz & M. Bader.** Role-Oriented Code Generation in an Engine for Solving Hyperbolic PDE Systems. SC19 SE-HER
  - [2] **A. Reinarz, D. Charrier, M. Bader, L. Bovard, M. Dumbser, K. Duru, F. Fambri, A.-A. Gabriel, J.-M. Gallard, S. K. Koeppel, L. Krenz, L. Rannabauer, L. Rezzolla, P. Samfass, M. Tavelli & T. Weinzierl.** ExaHyPE: An Engine for Parallel Dynamically Adaptive Simulations of Wave Problems. Computer Physics Communications. Accepted, 2019.
  - [3] **D. E. Charrier, B. Hazelwood, E. Tutlyaeva, M. Bader, M. Dumbser, A. Kudryavtsev, A. Moskovsky, and T. Weinzierl.** Studies on the energy and deep memory behaviour of a cache-oblivious, task-based hyperbolic PDE solver. International Journal of High Performance Computing Applications. Accepted, 2019.

  Scientific:
  - [1] **Kenneth Duru, Alice-Agnes Gabriel, and Gunilla Kreiss.** On energy stable discontinuous Galerkin spectral element approximations of the perfectly matched layer for the wave equation. Computer Methods in Applied Mechanics and Engineering, 350:898–937, 2019.

- [2] **Kenneth Duru, Leonhard Rannabauer, Alice-Agnes Gabriel, and Gunilla Kreiss.** A stable discontinuous Galerkin method for the perfectly matched layer for elastodynamics in first order form. https://arxiv.org/abs/1910.06477, 2019.
- [3] **M. Dumbser, F. Fambri, E. Gaburro, and A. Reinarz.** On GLM curl cleaning for a first order reduction of the CCZ4 formulation of the Einstein field equations. Journal of Computational Physics, 2019.

Software Requirements

Compiler and runtime
• Intel, GNU
External or Third-Party Libraries
• Python3 • Intel's TBB 2017 • MPI, versions newer than 1.3 • to use ExaHyPE's optimized compute kernels, Intel's libxsmm and the Python module Jinja2 are required.
Management tools
• GNU Make
I/O Libraries
• Limited HDF5 support available • Peano file format
Tools/Libraries for the code workflow
• Paraview/TecPlot

Hardware Requirements

Architectures/Machines Characterization
The ExaHyPE project relies on code generation to tailor its compute kernels toward a given architecture. A focus is given to Intel Skylake architecture as it highlights the need for advanced data layout to exploit every vectorization opportunity and cache-aware kernels as cache misses become a significant performance bottleneck. ExaHyPE also been extensively tested on the Intel Haswell nodes of SuperMUC-Phase 2.

4.5.2 Tests Conditions

Computational Environment Description
Processor • **Intel Xeon Gold 6148 @ 2.4 GHz codename Skylake** **Interconnect** • Infiniband EDR 100Gb/s **Filesystem** • Scratch Lustre DDN 7K

Job management system

- SLURM 17.11.5

Compiler

- Intel Compiler 2018 build 5

MPI

- Intel MPI 2018 build 5

Intra-node parallelism

- Intel TBB 2018 build 5

Flags

- COMPILER_CFLAGS = -DTrackGridStatistics -g -03 -ip -fma -Wall -restrict -std=c++11 -pedantic -qopenmp-simd -xCORE-AVX512 -qopt-zmm-usage=high -DSharedTBB -DParallel -DnoPackedRecords -DnoPersistentRegularSubtrees -DIprobeEveryKIterations=0 -DUsePeanosRLEBoundaryExchanger -DUsePeanosRLEBoundaryExchangerForMetaData

- COMPILER_LFLAGS = -xCORE-AVX512 -qopt-zmm-usage=high FCOMPILER_CFLAGS = -g -O2 -r8 -cpp -auto -qopenmp-simd -xCORE-AVX512 -qopt-zmm-usage=high

- PROJECT_CFLAGS = -DDim3 -DALIGNMENT=64

Job resources

- 7 nodes
- 28 MPI processes
- 4 MPI processes per node
- 10 TBB-threads per MPI process

Test Case Description

The ExaSeis application on the ExaHyPE elastic solver. The latter is based on the underlying Peano framework. Peano linearizes the domain with a space filling curve. Distributed memory parallelization (MPI) is performed by assigning single sub-tree of the space-tree to MPI ranks. Within these sub-trees, shared memory parallelization is built on a task-based paradigm (Intel TBB).
The test case consists in a high order discontinuous Galerkin solver for elastic wave equation called Curvilinear available in the ExaSeis repository under Applications.

This test setup models complex topographies by applying a curvilinear transformation to the elements of an adaptive Cartesian mesh. A mesh of surface quadrature nodes is generated depending on the topography, creating a 2D curvilinear interpolation of those quadrature nodes on domain boundaries with topography curves and domain edges as constraints.

With the curvilinear model we solve for the LOH.1 benchmark on a domain of size [47,47,47]. The mesh contained 27 elements in each direction, as well as one layer of adaptive mesh refinement near the jump in material parameters. The model was run for 100 time steps in each configuration with MPI and TBB parallelization enabled.

The following optimization settings were used:

```
"fuse_algorithmic_steps" : "all"
"fuse_algorithmic_steps_rerun_factor" : 0.99
"fuse_algorithmic_steps_diffusion_factor" : 0.99
"spawn_predictor_as_background_thread" : true
"spawn_amr_background_threads" : true
"disable_vertex_exchange_in_time_steps" : true
"time_step_batch_factor" : 1.0
"disable_metadata_exchange_in_batched_time_steps" : true
"double_compression" : 0.0
"spawn_double_compression_as_background_thread" : false
```

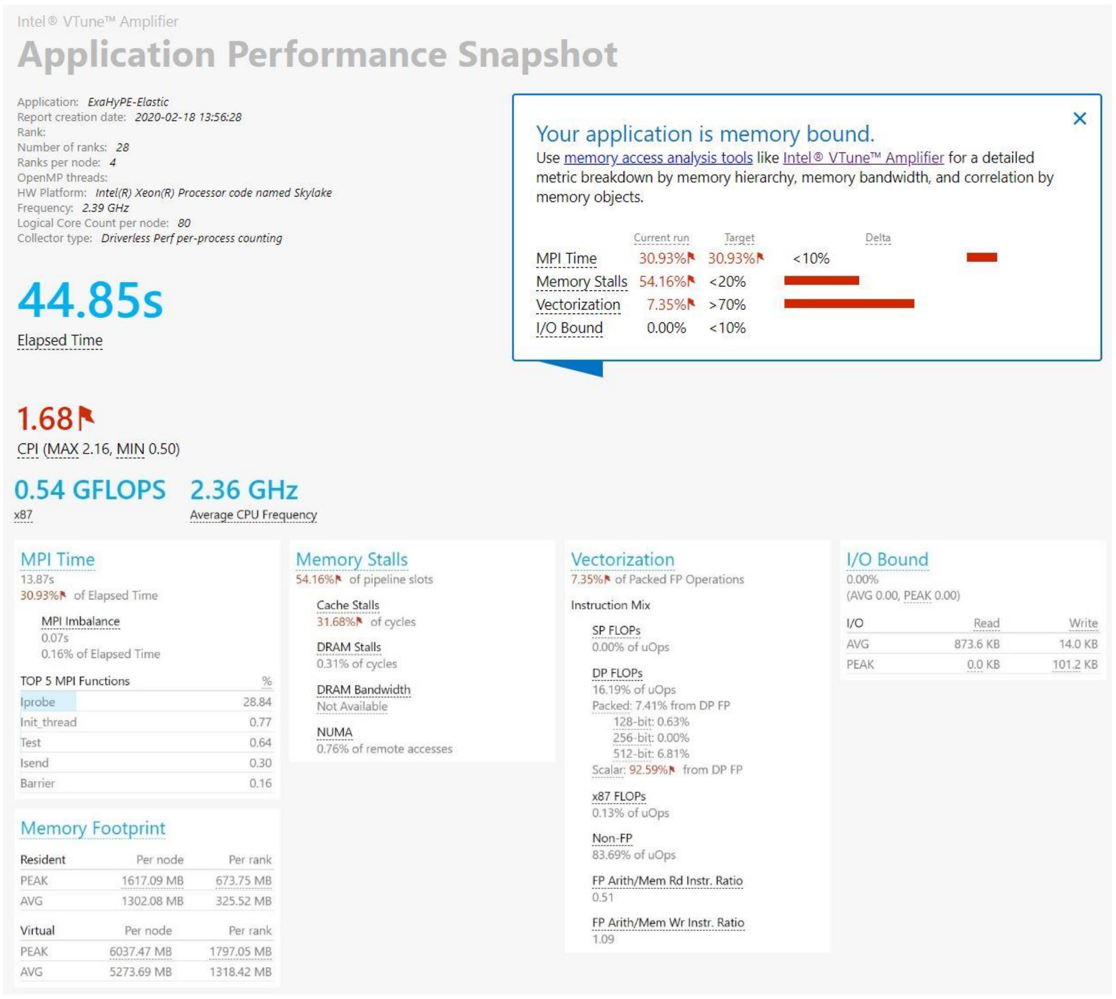
However, generic kernels were used. As such no architecture-specific optimizations were enabled in the following tests.

In the experiments, 7 cluster nodes with Intel Skylake processors, each with 80 logical CPUS were used to launch the application. The ExaHyPE-Elastic application scales well with 1, 28 et 731 MPI processes. As a result, the 28 MPI processes configuration is used. 40 cores are loaded on each node with MPI processes or (TBB) threads. Increasing the number of TBB-threads, the best scaling is obtained with 10 threads.

4.5.3 Results

Performance overview

The first step in analyzing the hybrid MPI/TBB application ExaHyPE is getting an overview of the performances. The profiler Intel Application Performance Snapshot is used. This includes information about MPI and multi-threading time and load balance, memory and disk usage and most used MPI operations. The tests were run with ExaHyPE's "generic kernels", which do not rely on optimized small matrix multiplication kernels (LIBXSMM, e.g.) – this explains the bad vectorization figures observed.



Intel® VTune™ Amplifier

Application Performance Snapshot

Application: *ExaHyPE-Elastic*
Report creation date: *2020-02-18 13:56:28*
Rank:
Number of ranks: *28*
Ranks per node: *4*
OpenMP threads:
HW Platform: *Intel(R) Xeon(R) Processor code named Skylake*
Frequency: *2.39 GHz*
Logical Core Count per node: *80*
Collector type: *Driverless Perf per-process counting*

Your application is memory bound.
Use memory access analysis tools like Intel® VTune™ Amplifier for a detailed metric breakdown by memory hierarchy, memory bandwidth, and correlation by memory objects.

	Current run	Target	Delta
MPI Time	30.93%	30.93%	<10%
Memory Stalls	54.16%	<20%	
Vectorization	7.35%	>70%	
I/O Bound	0.00%	<10%	

44.85s
Elapsed Time

1.68
CPI (MAX 2.16, MIN 0.50)

0.54 GFLOPS
x87

2.36 GHz
Average CPU Frequency

MPI Time
13.87s
30.93% of Elapsed Time

MPI Imbalance
0.07s
0.16% of Elapsed Time

TOP 5 MPI Functions	%
Iprobe	28.84
Init_thread	0.77
Test	0.64
Isend	0.30
Barrier	0.16

Memory Footprint

Resident	Per node	Per rank
PEAK	1617.09 MB	673.75 MB
AVG	1302.08 MB	325.52 MB

Virtual	Per node	Per rank
PEAK	6037.47 MB	1797.05 MB
AVG	5273.69 MB	1318.42 MB

Memory Stalls
54.16% of pipeline slots

Cache Stalls
31.68% of cycles

DRAM Stalls
0.31% of cycles

DRAM Bandwidth
Not Available

NUMA
0.76% of remote accesses

Vectorization
7.35% of Packed FP Operations

Instruction Mix

SP FLOPs
0.00% of uOps

DP FLOPs
16.19% of uOps
Packed: 7.41% from DP FP
128-bit: 0.63%
256-bit: 0.00%
512-bit: 6.81%
Scalar: 92.59% from DP FP

x87 FLOPs
0.13% of uOps

Non-FP
83.69% of uOps

FP Arith/Mem Rd Instr. Ratio
0.51

FP Arith/Mem Wr Instr. Ratio
1.09

I/O Bound
0.00%
(AVG 0.00, PEAK 0.00)

I/O	Read	Write
AVG	873.6 KB	14.0 KB
PEAK	0.0 KB	101.2 KB

- **I/Os**

In the Curvilinear test case, I/Os have been disabled in purpose. Therefore, it will not be investigated in this study.

- **Memory footprint**

The performance overview shows results on the per-process and per-node memory usage of both virtual and resident memory. For this specific test case, the application does not require a large amount of memory. The total memory available is 192 GB per node, while the maximum memory used is 6037 MB per node.

- **MPI time**

The MPI time is high and represents almost 31% of the elapsed time. it is highly used by the MPI function “Iprobe()” which is a non-blocking test for messages. Such problem may be caused by a non-optimal communication schema. On the other hand, it is shown that there is no MPI imbalance issue. Here, a dedicated rank oversees the load balance. Therefore, a big portion of time is spent in synchronizations.

- **Cache stalls**

A significant proportion of cycles (~32 %) are spent on data fetches from cache. The strategies adopted in the applications (data alignments …) do not seem to work optimally. Such issues are related to intra-node parallelization and therefore, the data sharing should be investigated in depth.

- **Vectorization**

A significant fraction of floating-point arithmetic instructions (~93 %) are scalar and do not benefit from the advanced vectorization capabilities of the processor. Only a small portion of the code is vectorized (~7%) using AVX512 instruction set. This is due to the generic kernels employed in this test. In the optimized kernels a very high vectorization level is reached.

Frequency dependency analysis

The frequency dependency of the application is analyzed here.

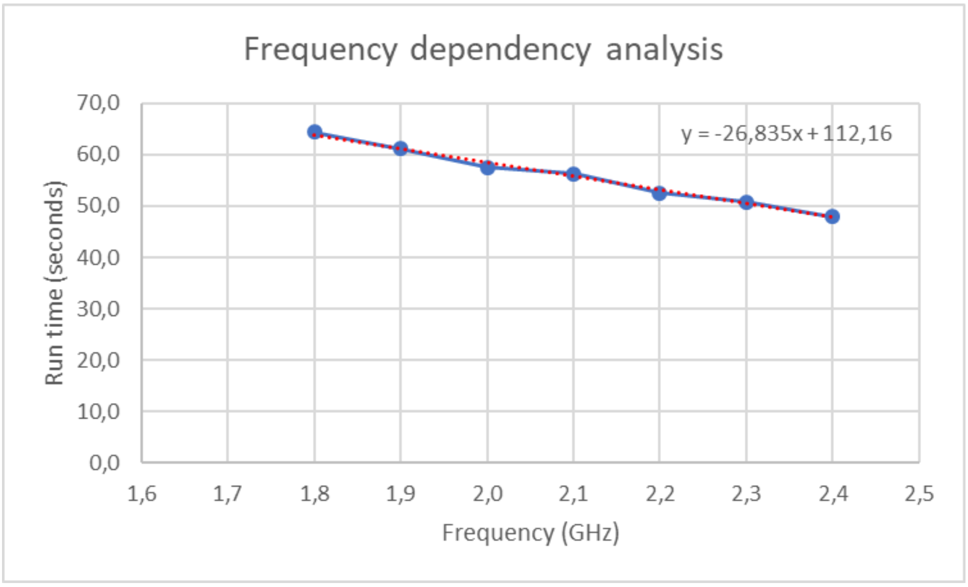


Figure 1: Frequency dependency analysis

There is clearly a linear frequency dependency despite the MPI synchronizations. The application is 80 to 90% frequency dependent.

MPI analysis

There are several reasons for an application to be MPI-bound and some major reasons are :

- High times inside the MPI library which occurs when a process waits for the data from other processes. Such problem was pointed out in the performance overview section earlier with the MPI Iprobe function.
- Active communications

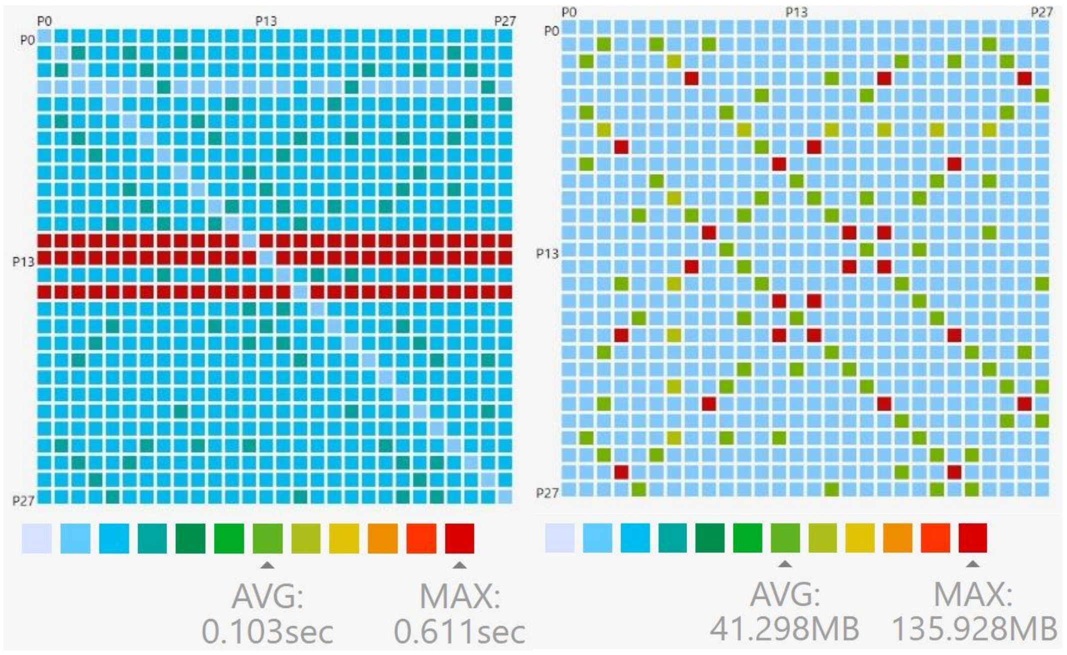
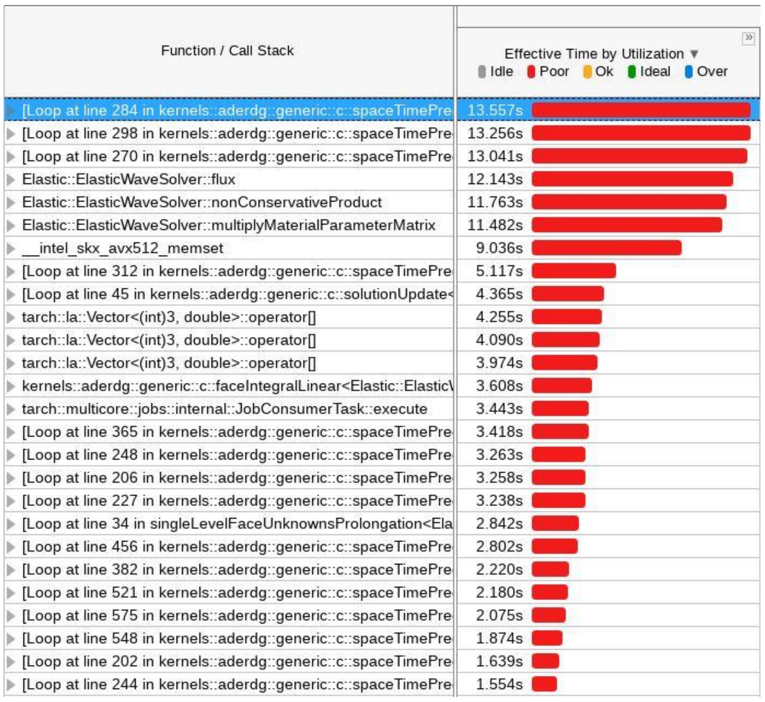


Figure 2: Rank-to-rank communication matrix: with respect to time (left) and data transfers (right).

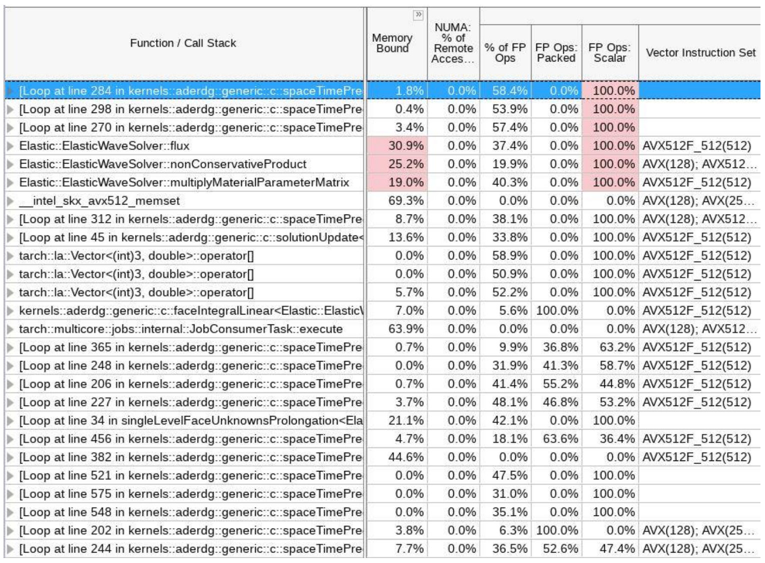
The message profile here above indicates the intensity of point-to-point communications for each sender-receiver pair. The vertical processes axis represents the senders and he horizontal processes axis represents the receivers.
Such a communication pattern is unusual. Few MPI processes transfer large amount of data to other few MPI processes. MPI ranks 12, 13 and 15 utilize a large amount of time (more than x6) communicating as well.

Computations analysis
The most time-consuming loops/functions with floating-points operations are identified as : - Loop at line 284 in kernels::aderdg spaceTimePredictorLinear - Loop at line 298 in kernels::aderdg spaceTimePredictorLinear - Loop at line 270 in kernels::aderdg spaceTimePredictorLinear - Loop at line 312 in kernels::aderdg spaceTimePredictorLinear - Loop at line 45 in kernels::aderdg solutionUpdate - Function Elastic:ElasticWaveSolver::flux - Function Elastic:ElasticWaveSolver::nonConservativeProduct - Function Elastic:ElasticWaveSolver::multiplyMaterialParameterMatrix



Function / Call Stack	Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)
[Loop at line 284 in kernels::aderdg::generic::c::spaceTimePre	13.557s 🟥
[Loop at line 298 in kernels::aderdg::generic::c::spaceTimePre	13.256s 🟥
[Loop at line 270 in kernels::aderdg::generic::c::spaceTimePre	13.041s 🟥
Elastic::ElasticWaveSolver::flux	12.143s 🟥
Elastic::ElasticWaveSolver::nonConservativeProduct	11.763s 🟥
Elastic::ElasticWaveSolver::multiplyMaterialParameterMatrix	11.482s 🟥
__intel_skx_avx512_memset	9.036s 🟥
[Loop at line 312 in kernels::aderdg::generic::c::spaceTimePre	5.117s 🟥
[Loop at line 45 in kernels::aderdg::generic::c::solutionUpdate<	4.365s 🟥
tarch::la::Vector<(int)3, double>::operator[]	4.255s 🟥
tarch::la::Vector<(int)3, double>::operator[]	4.090s 🟥
tarch::la::Vector<(int)3, double>::operator[]	3.974s 🟥
kernels::aderdg::generic::c::faceIntegralLinear<Elastic::Elastic\	3.608s 🟥
tarch::multicore::jobs::internal::JobConsumerTask::execute	3.443s 🟥
[Loop at line 365 in kernels::aderdg::generic::c::spaceTimePre	3.418s 🟥
[Loop at line 248 in kernels::aderdg::generic::c::spaceTimePre	3.263s 🟥
[Loop at line 206 in kernels::aderdg::generic::c::spaceTimePre	3.258s 🟥
[Loop at line 227 in kernels::aderdg::generic::c::spaceTimePre	3.238s 🟥
[Loop at line 34 in singleLevelFaceUnknownsProlongation<Ela	2.842s 🟥
[Loop at line 456 in kernels::aderdg::generic::c::spaceTimePre	2.802s 🟥
[Loop at line 382 in kernels::aderdg::generic::c::spaceTimePre	2.220s 🟥
[Loop at line 521 in kernels::aderdg::generic::c::spaceTimePre	2.180s 🟥
[Loop at line 575 in kernels::aderdg::generic::c::spaceTimePre	2.075s 🟥
[Loop at line 548 in kernels::aderdg::generic::c::spaceTimePre	1.874s 🟥
[Loop at line 202 in kernels::aderdg::generic::c::spaceTimePre	1.639s 🟥
[Loop at line 244 in kernels::aderdg::generic::c::spaceTimePre	1.554s 🟥

Figure 3: Code regions with effective time utilization



Function / Call Stack	Memory Bound	NUMA: % of Remote Acces…	% of FP Ops	FP Ops: Packed	FP Ops: Scalar	Vector Instruction Set
[Loop at line 284 in kernels::aderdg::generic::c::spaceTimePre	1.8%	0.0%	58.4%	0.0%	100.0%	
[Loop at line 298 in kernels::aderdg::generic::c::spaceTimePre	0.4%	0.0%	53.9%	0.0%	100.0%	
[Loop at line 270 in kernels::aderdg::generic::c::spaceTimePre	3.4%	0.0%	57.4%	0.0%	100.0%	
Elastic::ElasticWaveSolver::flux	30.9%	0.0%	37.4%	0.0%	100.0%	AVX512F_512(512)
Elastic::ElasticWaveSolver::nonConservativeProduct	25.2%	0.0%	19.9%	0.0%	100.0%	AVX(128); AVX512…
Elastic::ElasticWaveSolver::multiplyMaterialParameterMatrix	19.0%	0.0%	40.3%	0.0%	100.0%	AVX512F_512(512)
__intel_skx_avx512_memset	69.3%	0.0%	0.0%	0.0%	0.0%	AVX(128); AVX(25…
[Loop at line 312 in kernels::aderdg::generic::c::spaceTimePre	8.7%	0.0%	38.1%	0.0%	100.0%	AVX(128); AVX512…
[Loop at line 45 in kernels::aderdg::generic::c::solutionUpdate<	13.6%	0.0%	33.8%	0.0%	100.0%	AVX512F_512(512)
tarch::la::Vector<(int)3, double>::operator[]	0.0%	0.0%	58.9%	0.0%	100.0%	AVX512F_512(512)
tarch::la::Vector<(int)3, double>::operator[]	0.0%	0.0%	50.9%	0.0%	100.0%	AVX512F_512(512)
tarch::la::Vector<(int)3, double>::operator[]	5.7%	0.0%	52.2%	0.0%	100.0%	AVX512F_512(512)
kernels::aderdg::generic::c::faceIntegralLinear<Elastic::Elastic\	7.0%	0.0%	5.6%	100.0%	0.0%	AVX512F_512(512)
tarch::multicore::jobs::internal::JobConsumerTask::execute	63.9%	0.0%	0.0%	0.0%	0.0%	AVX(128); AVX512…
[Loop at line 365 in kernels::aderdg::generic::c::spaceTimePre	0.7%	0.0%	9.9%	36.8%	63.2%	AVX512F_512(512)
[Loop at line 248 in kernels::aderdg::generic::c::spaceTimePre	0.0%	0.0%	31.9%	41.3%	58.7%	AVX512F_512(512)
[Loop at line 206 in kernels::aderdg::generic::c::spaceTimePre	0.7%	0.0%	41.4%	55.2%	44.8%	AVX512F_512(512)
[Loop at line 227 in kernels::aderdg::generic::c::spaceTimePre	3.7%	0.0%	48.1%	46.8%	53.2%	AVX512F_512(512)
[Loop at line 34 in singleLevelFaceUnknownsProlongation<Ela	21.1%	0.0%	42.1%	0.0%	100.0%	
[Loop at line 456 in kernels::aderdg::generic::c::spaceTimePre	4.7%	0.0%	18.1%	63.6%	36.4%	AVX512F_512(512)
[Loop at line 382 in kernels::aderdg::generic::c::spaceTimePre	44.6%	0.0%	0.0%	0.0%	0.0%	AVX512F_512(512)
[Loop at line 521 in kernels::aderdg::generic::c::spaceTimePre	0.0%	0.0%	47.5%	0.0%	100.0%	
[Loop at line 575 in kernels::aderdg::generic::c::spaceTimePre	0.0%	0.0%	31.0%	0.0%	100.0%	
[Loop at line 548 in kernels::aderdg::generic::c::spaceTimePre	0.0%	0.0%	35.1%	0.0%	100.0%	
[Loop at line 202 in kernels::aderdg::generic::c::spaceTimePre	3.8%	0.0%	6.3%	100.0%	0.0%	AVX(128); AVX(25…
[Loop at line 244 in kernels::aderdg::generic::c::spaceTimePre	7.7%	0.0%	36.5%	52.6%	47.4%	AVX(128); AVX(25…

Figure 4: Code regions with memory bound problems and floating-point operations types

Unfortunately, most of the intensive loops perform 100% scalar operations and/or suffer from memory bound issues due to data access.

4.5.4 Conclusion

Summary

The ExaHyPE/ExaSeis application was tested on the Curvilinear case study. The analysis has shown a good efficiency although some issues have identified:

- MPI issues due to large time with the Iprobe() function, a dedicated MPI

rank oversees the load balancing and spends a lot of time in synchronizations,
- Cache stalls limit the performance of the application,
- Despite the vectorization direction inserted in the code and forced by the compiler, the intensive parts of the code could not be vectorized.

The final two issues have been resolved in the optimized kernels of ExaHyPE in the ChEESE-COE and a high-level of vectorization and reduction of the cache-stalls has been achieved. Future tests should work with directly with these optimized variants of the code.

Next steps

As for the next steps, the application presents some performance issues which can be investigated. These are some points which require a deeper analysis and first attempts for optimization:
- Effect of the communication pattern,
- Data sharing to improve cache utilization and avoid stalls,

4.6 SEM46

4.6.1 Overview

SEM46 is a 3D seismic modeling and inversion code, developed mainly in the frame of the SEISCOPE project (https://seiscope2.osug.fr/) for tackling modeling and full waveform inversion topics from the near surface to the deep crustal scale. The modeling kernel is based on spectral elements designed on Cartesian-based hexahedral meshes. The code implements in its current version elastic and visco-elastic equations for both modeling and inversion tasks. The inversion part is coupled with the non-linear Optimization toolbox of SEISCOPE (https://seiscope2.osug.fr/SEISCOPE-OPTIMIZATION-TOOLBOX) in order to implement efficient large-scale non-linear optimization schemes.

Code description

Source Code Repository

- SEM46 is available upon request to Romain Brossier romain.brossier@univ-grenoble-alpes.fr
  SEM46 relies on a BSD-like license but include restrictions for the diffusion, imposed by the funding partners of the SEISCOPE project.

Version

- Version 2.3

Code Versioning Tool

- SubVersioN

Sanity Check / Unit Testing Framework

- Some inner time measures are implemented, allowing to have reference times for each step on each test-case identified.

Documentation

- Up to date Manual available with the source code

Code Current Performance

- **Measured performances**
  - A few test cases are available in our data-base, based on intel Sandybridge, KNL and Skylake.

Ongoing Research

- **Contact**
  - Romain Brossier romain.brossier@univ-grenoble-alpes.fr
- **Ongoing research**
  - implementation of fluid/solid coupled equations to tackle modeling and inversion in marine environments
  - extension to anisotropic media (the compute kernels of modeling and inversion implement anisotropy since the initial development of SEM46, but only isotropic I/Os are currently implemented)
  - Optimal-transport-based misfit function in the inversion

- **Latest publications**
  - **Phuong-Thu Trinh, Romain Brossier, Ludovic Metivier, Laure Tavard, and Jean Virieux.** Efficient time-domain 3d elastic and visco-elastic FWI using a spectral-element method on flexible cartesian-based mesh. Geophysics, 84(1):R75--R97, 2019. http://users.isterre.fr/brossier/Trinh_2019_GEO.pdf
  - **Weiguang He, Romain Brossier, and Ludovic Métivier.** 3d elastic fwi for land seismic data: A graph space ot approach. In *SEG Technical Program Expanded Abstracts 2019*, pages 1320--1324, 2019. http://users.isterre.fr/brossier/He_2019_SEG.pdf
  - **T.M. Irnaka, R. Brossier, L. Métivier, T. Bohlen, and Y. Pan.** Towards 3d 9c elastic full waveform inversion of shallow seismic wavefields - case study ettlingen line. In *Expanded Abstracts, 81th Annual EAGE Conference & Exhibition, London*, page We P01 04. EAGE, 2019. http://users.isterre.fr/brossier/Irnaka_2019_EAGE.pdf
  - **P.T. Trinh, R. Brossier, L. Métivier, J. Virieux, and P. Wellington.** Bessel smoothing filter for spectral-element mesh. Geophysical Journal International, 209(3):1489--1512, 2017. http://users.isterre.fr/brossier/Trinh_2017_GJI.pdf

Software Requirements

Compiler and runtime
• Intel, GNU
External or Third Party Libraries
• MPI • External "in-house" libraries
Management tools
• Makefile
I/O Libraries
• MPI I/O
Tools/Libraries for the code workflow
• Paraview • Seismic Unix

Hardware Requirements

Architectures/Machines Characterization
Hardware requirements vary a lot depending on the test case. The code runs on standard intel-based nodes usually, it is also working on intel KNL

1.1.1 Tests Conditions (for POP audit)

Computational Environment Description

- Platform: MareNostrum IV
- 2 x Intel Xeon Platinum 8160 24C at 2.1 GHz per node
- 97 or 384 GB of memory per node
- Storage: 200 GB local SSD
- Interconnection network: 100 Gb Intel OmniPath

Test Case Description

Input case:

- 2 different inputs from production runs.
- Each input with 2 configurations
  - M1 – forward 1 shot,
  - M2 – FWI all waves 1 shot
- Scale:
  - 48 cores for the small case,
  - 96 and 192 cores for the big case
- Initial set-up: 9000 iterations. Reduced to 180 iterations after check

1.1.2 Results

Application structure

First analysis with the small input case

- 48 MPI,
- 9000 iterations

Phases clearly identified in the tracefile:

- Initialization
- iterative computation

A quick look on this trace file showed there is no time correlation (similar unbalance in all the iterations)

- Can focus the analysis in a smaller number of iterations

Focus of Analysis

- The selected focus of analysis is the 180 iterations as the M2 configuration has two phases

- The first phase of M2 configuration is very similar to M1

- Zooming into a small area of M2 where we can see the iterations in both phases
  - With the small input, the granularity of the main computations in phase 1 for M1 and M2 are around 45 ms while M2 phase 2 goes up to 63 ms.
  - The values grow to 90 ms and 123 ms on the big case
  - The two input cases cannot be used to analyze scaling

- The communications are done using MPI_Sendrecv() and the iterations are synchronized with MPI_Barrier()

Efficiency analysis

- The efficiencies are very good (smaller value 94.48).
- Both input cases report lightly worst efficiency for M2
  - The degradation may be related with phase 2.
  - Seems to be correlated with load balance and serialization.
- The IPC is a little bit low (smaller value 0.88) and should be the main target for optimizations.
- The rest of the report would focus on M2 as M1 is equivalent to M2 phase 1.

Note:

Efficiencies lower than 80% indicate space for improvement. Lower than 60% there is a clear need for improvement. Good IPC in MareNostrum use to be around 1.2 and 1.5

Load balance

- The Focus of the analysis is on small test case - M2 as it represents the execution with lower load balance (minimum value 97.16)
- There is a structured pattern of unbalance that it is correlated with instructions unbalance.
- The same structured unbalance appears in both phases. It is perhaps due to the domain decomposition.
- In the analyzed executions (small and big test cases), there is no need to improve the load balance (lower unbalance is 97.16)

Computations analysis

- The same computational behavior is identified in both small and big runs
- There is an increase in the number of instructions from the small to the big test cases
- IPC for phase 1 seems acceptable while for phase 2, it has a lower value
- Phase 2 should be the first target for optimizations

MPI analysis

- Both phases have a similar communication pattern
- The only difference is that phase 2 has two calls to MPI_Barrier() per iteration, while phase 1 only has one call
- As the synchronization is guaranteed by the MPI_Sendrecv() calls, all the calls to MPI_Barrier() can be eliminated in both phases.
- Nevertheless, the impact would be small as the total MPI time is less than 6% in the worst case

- After eliminating the calls to MPI_Barrier(), the small M1 case was rerun but with a different number of iterations. Very similar results were obtained with differences lower than 1%
- Nevertheless, the elimination of barriers would improve the execution at very large scales

1.1.3 Conclusion

Summary

- The audit of SEM64 shows very good efficiencies for both input cases (small and big) and the two configurations (M1 and M2)

- The analysis identified the weakest factor is a relatively low IPC on the main computations being worst for the second phase of M2. That should be the main target for optimizations of the code.

- A very small unbalance correlated with the work distribution has been detected but the benefits improving it would be very limited.

- The study allowed the user to identify that the calls to MPI_Barrier() were not needed. Moreover, despite the limited improvement they represent in the analyzed scale, it may make a significant difference when running at very large scales.

Next steps

- The first step of investigation is the improvement of the IPC for the second phase of M2
- The second step of investigation is the improvement of the synchronization of MPI ranks by reducing the unnecessary MPI_Barrier() calls. At large scale executions, improvements should be observed.

5. DualSPHysics

5.1.1 Overview

DualSPHysics is a hardware accelerated Smoothed Particle Hydrodynamics code developed to solve free-surface flow problems. The code is developed to study free-surface flow phenomena where Eulerian methods can be difficult to apply, such as waves or impact of dam-breaks on off-shore structures. DualSPHysics is a set of C++, CUDA and Java codes designed to deal with real-life engineering problems. DualSPHysics is an open-source code developed and released under the terms of GNU General Public License (GPLv3). Along with the source code, a complete documentation that makes easy the compilation and execution of the source files is also distributed. The code has been shown to be efficient and reliable. The parallel power computing of Graphics Computing Units (GPUs) is used to accelerate DualSPHysics by up to two orders of magnitude compared to the performance of the serial version.

Code description

Source Code Repository
• **https://github.com/DualSPHysics/DualSPHysics/tree/develop**
Version
• Version 4.4.036
Code Versioning Tool
• Git
Sanity Check / Unit Testing Framework
• None
Documentation
• Manual available at https://github.com/DualSPHysics/DualSPHysics/wiki
Code Current Performance
The performance tests are being conducted by Sandra Mendez (sandra.mendez@bsc.es).
Ongoing Research
• **Contact** Dr. José M. Domínguez (jmdominguez@uvigo.es) Dr. Jaime Klapp (jaime.klapp@inin.gob.mx) Full list of developers at https://dual.sphysics.org/index.php/developers/ • **Latest publications** • Reference paper: A.J.C. Crespo, J.M. Domínguez, B.D. Rogers, M. Gómez-Gesteira, S. Longshaw, R. Canelas, R. Vacondio, A. Barreiro, O. García-Feal. 2015. DualSPHysics: open-source parallel CFD solver on Smoothed

Particle Hydrodynamics (SPH). Computer Physics Communications, 187: 204-216. doi:10.1016/j.cpc.2014.10.004.

- Other references:
  J.M. Domínguez, A.J.C. Crespo, M. Hall, C. Altomare, M. Wu, V. Stratigaki, P. Troch, L. Cappietti, M. Gómez-Gesteira. 2019. SPH simulation of floating structures with moorings. Coastal Engineering, 153, 103560. doi:10.1016/j.coastaleng.2019.103560.

  J.M. Domínguez, A.J.C. Crespo and M. Gómez-Gesteira. 2013. Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method. Computer Physics Communications, 184(3): 617-627. doi:10.1016/j.cpc.2012.10.015.

  J. Klapp, O.S. Areu-Rangel, M. Cruchaga, R. Aránguiz, R. Bonasia, M. Godoy-Seura and R. Silva-Casarín. 2019. Tsunami hydrodynamic force on a building using a SPH real scale numerical simulation. Natural Hazards, Pags. 1-21. doi: 10.1007/s11069-019-03800-3.

  C.E. Alvarado-Rodríguez, L.D.G. Sigalotti and J. and J. Klapp. 2019. Anisotropic dispersion with a consistent smoothed particle hydrodynamics scheme. Advances in Water Resources 131, 103374, doi: https://doi.org/10.1016/j.advwatres.2019.07.004 .

  Full list of references by developers at https://dual.sphysics.org/index.php/references/

Software Requirements

Compiler and runtime
• gcc 7.3.0 • CUDA 9.2
External or Third Party Libraries
• None
Management tools
• Cmake
I/O Libraries
• None
Tools/Libraries for the code workflow
• Own pre-processing and post-processing tools and ParaView

Hardware Requirements

Architectures/Machines Characterization
• NVIDIA GPU Kepler or higher

5.1.2 Tests Conditions

Computational Environment Description
IBM Power 8 and Power 9 machines

Test Case Description

In order to study the efficiency of the devices and the scaling of the DualSPHysics V4.4 code, a pipe flow was simulated using a Hagen-Poiseuille profile with the following characteristics. Internal radius nine centimeters and length two meters. The gravity was not considered. Instead an acceleration of 1 m / s was inserted in the positive direction of the x-axis. Periodic conditions at the inlet and outlet of the tube were considered.

The purpose of the case study is to know the computation time that is carried out for different resolutions and computing equipment, as well as the convergence of the velocity profile inside the tube and its comparison with the analytical solution. One second of real time was simulated with a minimum time step of 0.00001 seconds to at best have only ten thousand time steps.

The case study was executed for 1, 2, 4, 8, 16, 32, 64 and 128 million particles. The last case was only executed for the computer equipment that supported the required memory.

5.1.3 Results

Note that the same chapter structure is kept even if all inputs were not available at the moment.

Application structure
To be completed in the next deliverables

Focus of Analysis
To be completed in the next deliverables

Efficiency analysis

The results of the computation time for different equipment are presented in Table 1, Table 2 and Table 3. The velocity profile inside the tube is presented in Figure 5.
Currently qualitative and quantitative comparison of the numerical results with the analytical solution is performed, as well as the evaluation of the convergence in the results.

Power 8 P100			
Case	Total number of particles	Total execution time	Total number of steps
1	1020067	148.242599	10309

2	2154885	407.568939	13434
3	4120704	899.27887	16515
4	8012568	2161.48853	20665
5	16130214	5647.53418	26307
6	32615684	14604.1367	33205
7	63847981	47323.9609	41623

Table 1: Computation time and time steps executed on the IBM Power 8 machine.

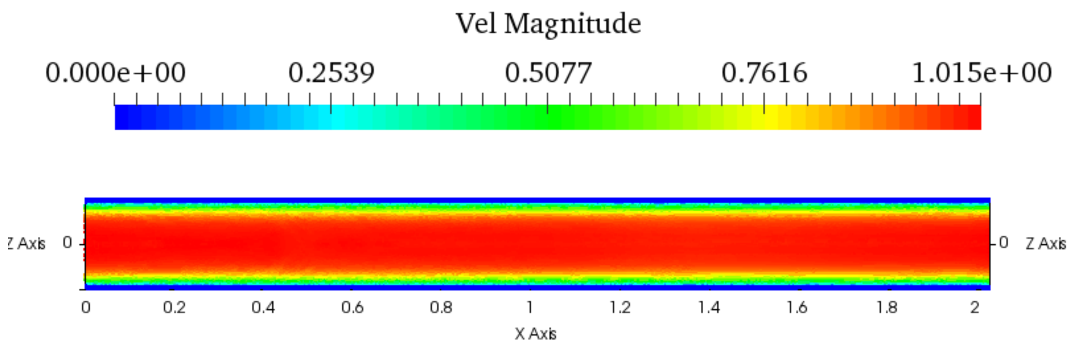


Figure 5: Speed profile in m / s obtained after a second simulation.

Power 9 V100			
Case	Total number of particles	Total execution time	Total number of steps
1	1020067	74.771965	10310
2	2154885	188.661972	13436
3	4120704	405.29187	16524
4	8012568	952.104065	20663
5	16130214	2419.27222	26295
6	32615684	6103.99951	33198
7	63847981	15040.0557	41623
8	127251947	37498.9453	52267

Table 2: Computation time and time steps executed on the IBM Power 9 machine.

NVIDIA V100			
Case	Total number of particles	Total execution time	Total number of steps
1	1020067	68.641159	10312
2	2154885	178.312317	13437
3	4120704	436.019562	16518
4	8012568	897.938599	20665
5	16130214	2291.48071	26302
6	32615684	5776.74072	33209
7	63847981	14194.5986	41619

8	127251947	35253.6016	52255

Table 3: Computation time and time steps executed on the NVIDIA machine.

Figure 6 and Figure 7 show the comparative graphs of the results presented in Table 1, Table 2 and Table 3.

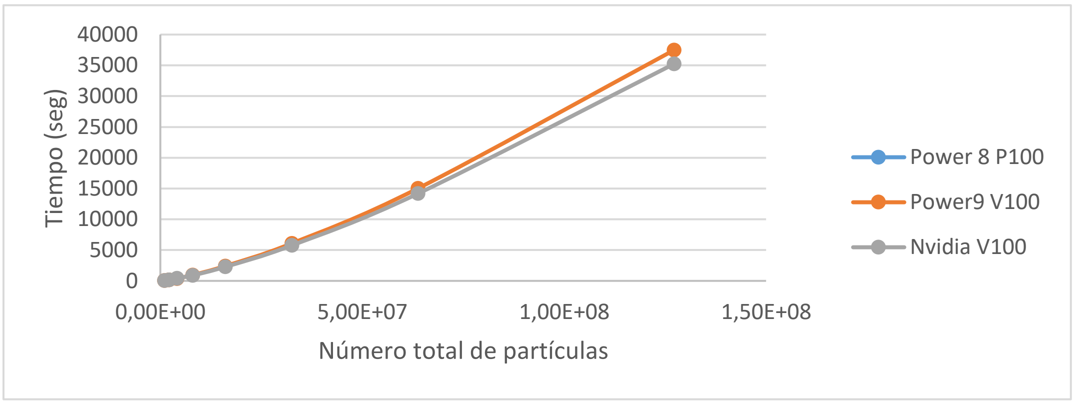


Figure 6: Execution time based on the total number of particles for the different computer equipment used.

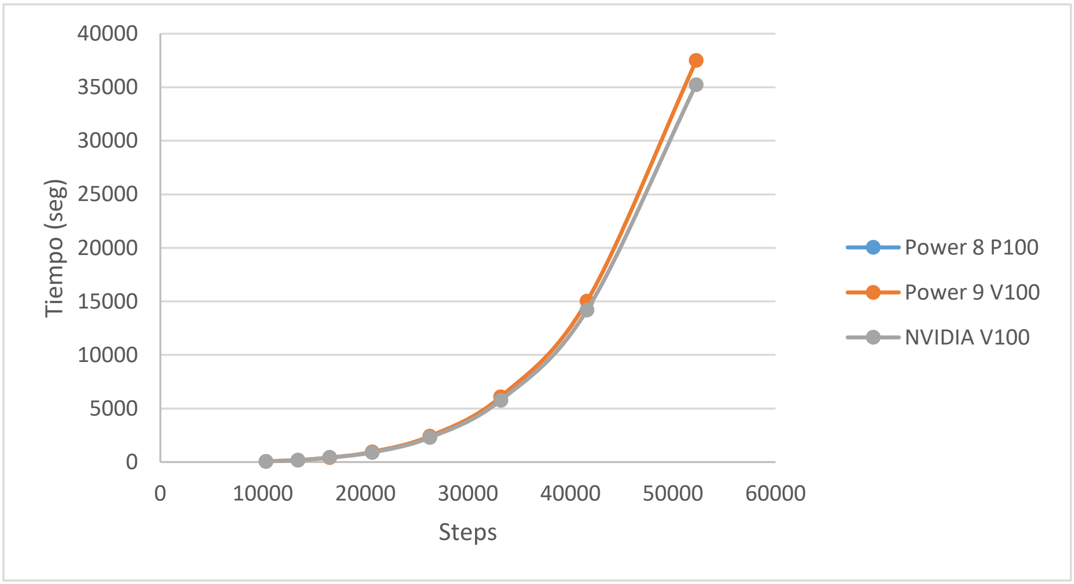


Figure 7: Execution time according to the total number of steps executed for the different computer equipment used.

Load balance

To be completed in the next deliverables

Computations analysis

To be completed in the next deliverables

MPI analysis
To be completed in the next deliverables

5.1.4 Conclusion

Summary

Qualitative and quantitative comparison of the numerical results with the analytical solution is performed, as well as the evaluation of the convergence in the results. With respect to the Nvidia K-40, the code at high resolution runs 15 times faster with P100 and about 40-50s times faster with the V100. This effect shall be investigated in depth.

Next steps

As all the POP results (and used test cases) are not yet available, the next steps will consist in analyzing the optimization priorities linked to the research domain for Enerxico. The additional information will be gathered in the next deliverables allowing to provide an update of this deliverable.

6. Conclusion

This deliverable shows a homogeneous scheme for the seven codes with variations depending on the level of performance of the applications. Some applications have more optimization and performance improvement possibilities because they are in "early stages" of scalability; and some others seem to have already a very good efficiency and scalability. For these ones, it could be more interesting to focus on the optimization of new features, as they will be developed for work packages 2, 3 and 4.

The next steps section, at the end of the description and evaluation of each code, collects the observations and outlines the following actions, according to each case, to be carried out by developers or analysts. It provides the versatility to be as concise as “improvement of the IPC for the second phase of M2” for a given code or as flexible as “investigate the effect of new features in the overall performance” of another code.

At this point all developers have already a detailed list of the most significant performance bottlenecks in their current code version, regarding in particular the parallelization of the code, the effect of communications when scaling, the overall instructions and IPC efficiency, the load balance for the main characteristics.

Regarding the objectives of this deliverable, at least two performance bottlenecks have been identified. Below, the main performance bottlenecks are listed per application:

1. ALYA: the main cause of efficiency loss is the load balancing issue. A shared memory parallel version of the function causing the load imbalance may mitigate this problem.
2. BSIT: the roofline analysis shows a low arithmetic intensity, L1 and L2 memory access should be improved to data reuse. GPU utilization shows that efficiency is bounded by the memory bandwidth.
3. WRF: there is an effective good scalability up to 32 processes. To use more processes, it is recommended to increase the test case size. In general, the application presents good results in terms of efficiency, scalability and load balance.
4. SeisSol: in previous audits and analysis, SeisSol shows excellent results in terms of node-level performance (on Intel architectures), scalability and load balance. The focus for performance optimization will therefore be on model extensions and on non-Intel architectures.

5. ExaHyPE: the analysis has shown a good efficiency although some issues have been identified: cache stalls limit the performance of the application and the compiler was not able to vectorize the intensive parts of the code.
6. SEM46: the first track of optimization is the improvement of a relatively low IPC (Instruction Per Cycle) for the second configuration (Full Waveform Inversion with all waves and one shot)..The second track consist in the improvement of the MPI synchronizations by removing unnecessary barriers.
7. DualSPHysics: The Mexican partners have started working on the application with a 6-month delay. The POP's analysis of DualSPHysics is not yet completed. However, we have decided to share the preliminary information we have available for our use.

Depending on the bottleneck issues, the optimization that will be performed may be considered as part of the task 1.1 “Intra-node optimization” or as part of the Task 1.2 “Multi-node optimization” : for example, if there are issues at the communication level with load balancing, then any optimization performed in that direction will be linked to the Task 1.2, similarly if there is interest to vectorize the code to achieve a higher performance within a node, this will be linked to Task 1.1.

The introduction of each next deliverables (“Report on intra-node and multi-node optimizations for HPC codes” and “Report on enabling computational and energy efficient codes for the Exascale”) will present the choices that have been performed for each task and the reason why for a total transparency.