Master in High Performance Computing

HPC implementations in the SHYFEM hydrodynamic model using the PETSc library and application to modern architectures

Supervisors:
Simone Bnà,
Eric Pascolo,
Alberto Sartori,
Georg Umgiesser

Candidate:
Célia Laurent

6th edition
2019–2020

Introduction

Research activities in environmental fluid dynamics can be used, among hundreds of possible applications, to anticipate tides flooding, analyse the transport of organisms or pollutants, give an opportunity to study the future evolution of the environmental conditions coming with climate changes. Such studies require an increasing need for accuracy and computing productivity, solving problems becoming always bigger and obtaining faster solutions. High Performance Computing makes it possible by the constant evolution of the technology, but the CPU speed increasing since the early 90's following Moore's law might be reaching a limit and in the last 15 years the development of the new technologies evolved towards massively parallel systems and new accelerated partition architectures. The modern source codes must be re-designed to follow the changing technology while taking special care of the bandwidth memory issues caused by the relatively slower evolution of the network technology. Besides the need for increased accuracy and productivity, the development of the modern softwares must face another great challenge by increasing their performance to reduce computational costs in terms of energy and environmental impact. It is in this context that the present work takes place, it consists in the insertion in the SHYFEM model of the Portable, Extensible Toolkit for Scientific Computation Toolkit for Advanced Optimization - PETSc [01] library, a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. PETSc is an open source library under constant development employed in a multitude of scientific softwares; it supports CPU and GPUs parallelism through MPI, CUDA or OpenCL, as well as hybrid MPI-GPU parallelism. The present work is part of the developments of a parallel version of the SHYFEM [02] hydrodynamic finite element model based on the Message Passing Interface (MPI) [03] communication protocol ; it contributes to the improvement of the performance of the model and allows the porting of the system solver on the new accelerated partitions.

The SHYFEM Shallow water HYdrodynamic Finite Element Model is a finite element program solving the shallow water hydrodynamic equations using a semi-implicit time resolution scheme on unstructured meshes. SHYFEM is particularly well suited to solve the hydrodynamics of domains characterized by complex geometries and bathymetries such as lagoons, coastal seas, estuaries and lakes, where the finite elements method allows local refinements of the resolution to have a finer accuracy in the regions where the physics is more complex while saving computational resources elsewhere. SHYFEM can run with either a two or three-dimensional formulation and uses Dirichlet boundary conditions by determining the fluxes velocities along the open-boundaries and imposing no-slip boundary conditions along the coastal lines. It can simulate shallow water flats in a mass conserving way when tidal variations of the water level cover and uncover the elements of the grid, making them pass from "wet" to "dry" states and vice-versa. SHYFEM accounts for both barotropic and baroclinic pressure gradients, wind drag forcing, bottom friction dissipation, Coriolis forcing, and wind wave forcing. Moreover, it is coupled to several models that can be activated depending on the applications: a sediment transport model, an ecological biogeochemistry model for transport and reactions of nutrients and mercury, the BFM ecosystem model, the WWMII spectral wind wave model, and the GOTM turbulence model for sub-grid vertical turbulence effects.

SHYFEM was originally developed by ISMAR/CNR and became since 2020 a community model supported by a consortium of Italian and international Institutes : CNR, OGS, CMCC, DIFA-UNIBO,

CINECA, ARPAE, KU and IU. The task of implementing the PETSc solver was assigned to the National Institute of Oceanography - OGS in the context of the Work Plan 2020-2022 of the SHYFEM consortium, after a parallelization had just been implemented by CNR using the MPI communication protocol, based on a memory distributed node partitioning but was still lacking a distributed solver to solve the implicit system of equations.

SHYFEM is an open source program released under GPL license, it is written in Fortran and it is able to run on different architectures from workstation using OpenMP shared memory parallelization to HPC clusters using Message Passing Interface. The repository hosting the consortium source code is https://github.com/shyfem-cm/shyfem. Before merging into the official repository, the developments described in this document are available in the branches of a forked repository stored at https://github.com/CeliaLaurent/shyfem.

The details of the developments done to insert PETSc into the source code of SHYFEM will be presented, together with Profilings and Benchmark analysis of the software before and after the implementations, an analysis of the MPI partitioning and GPU tests results. The implementations made during the GPU Hackathon of Helmholtz, 7 & 14-16th of september 2020 are as well presented; they allow to use the NVIDIA AmgX solver [04] on the GPU, employing PETSc as a frontend through the AmgXWrapper [05]. Aside from the optimization of the calls to the PETSc solver some performance issues were identified and solutions were proposed. Finally, a modern compilation setup based on the Meson build system [06] was implemented as an alternative to the original Makefiles and provides a set of integration tests on both the CPU and GPU.

The works presented in this document were performed as a training thesis of the Master in High Performance Computing - MHPC co-founded by SISSA and ICTP, and were supported by OGS and CINECA under the HPC-TRES program award number 2020-06.

1. Solving the hydrodynamic equations of the SHYFEM model

We will start by a brief overview of the system of equations solved by SHYFEM, introduce the discretization that is employed and present the actual Sparsekit solver and the PETSc library. The details on the system of equations and discretization are taken from the user manual of SHYFEM [1] .

1.1. System of equations

The SHYFEM model solves the vertically integrated shallow water equations for the water level ζ and the vertically-integrated velocities U and V in the x and y direction :

$$\frac{\partial U}{\partial t} + gH\frac{\partial \zeta}{\partial x} + RU + X = 0 \tag{1}$$

$$\frac{\partial V}{\partial t} + gH\frac{\partial \zeta}{\partial y} + RV + Y = 0 \tag{2}$$

$$\frac{\partial \zeta}{\partial t} + \frac{\partial U}{\partial x} + \frac{\partial V}{\partial y} = 0 \tag{3}$$

where h is the undisturbed water depth and H the total water depth. g is the gravitational acceleration; t is the time; R the bottom friction coefficient; and f is the Coriolis parameter. X and Y contain all the other terms such as advective non-linear terms, wind stress, and horizontal turbulent diffusion.

1.2. Discretization method

SHYFEM usually uses a semi-implicit time formulation to accomplish the time integration.

It is achieved by using the following representation of the time-discretization of the momentum equations (1) and (2) :

$$\frac{U^{n+1} - U^n}{\Delta t} + gH\ \frac{1}{2}\ \frac{\partial(\zeta^{n+1} + \zeta^n)}{\partial x} + RU^{n+1} + X = 0$$

$$\frac{V^{n+1} - V^n}{\Delta t} + gH\ \frac{1}{2}\ \frac{\partial(\zeta^{n+1} + \zeta^n)}{\partial y} + RV^{n+1} + Y = 0$$

Leading to an explicit formulation for the unknowns U^{n+1} and V^{n+1} :

$$U^{n+1} = \frac{1}{1 + \Delta t R}\left(U^n - \Delta t\ gH\ \frac{1}{2}\frac{\partial(\zeta^{n+1} + \zeta^n)}{\partial x} - \Delta t X\right)$$

$$V^{n+1} = \frac{1}{1 + \Delta t R}\left(V^n - \Delta t\ gH\ \frac{1}{2}\frac{\partial(\zeta^{n+1} + \zeta^n)}{\partial y} - \Delta t Y\right)$$

While the following time-discretization of the continuity equation (3) is adopted :

$$\frac{\zeta^{n+1}-\zeta^{n}}{\Delta t}+\frac{1}{2}\,\frac{\partial(U^{n+1}+U^{n})}{\partial x}+\frac{1}{2}\,\frac{\partial(V^{n+1}+V^{n})}{\partial y}=0$$

replacing U^{n+1} and V^{n+1} by their explicit formulations in the discretized continuity equation leads to an implicit linear system for the unknown ζ^{n+1} :

$$\begin{aligned}\zeta^{n+1} \quad - \quad & \left(\frac{\Delta t}{2}\right)^{2}\frac{g}{1+\Delta tR}\left(\frac{\partial}{\partial x}(H\frac{\partial\zeta^{n+1}}{\partial x})+\frac{\partial}{\partial y}(H\frac{\partial\zeta^{n+1}}{\partial y})\right)\\ = \quad & \zeta^{n}+\left(\frac{\Delta t}{2}\right)^{2}\frac{g}{1+\Delta tR}\left(\frac{\partial}{\partial x}(H\frac{\partial\zeta^{n}}{\partial x})+\frac{\partial}{\partial y}(H\frac{\partial\zeta^{n}}{\partial y})\right)\\ & - \quad \frac{\Delta t}{2}\left(\frac{2+\Delta tR}{1+\Delta tR}\right)\left(\frac{\partial U^{n}}{\partial x}+\frac{\partial V^{n}}{\partial y}\right)\\ & + \quad \frac{\Delta t^{2}}{2(1+\Delta tR)}\left(\frac{\partial X}{\partial x}+\frac{\partial Y}{\partial y}\right)\end{aligned}$$

Obtaining and solving a fully explicit formulation is as well made possible within SHYFEM, by changing the weights of the new time level of the transport terms and pressure term, respectively in the continuity and momentum equations . However, the use of the semi-implicit formulation is prefered as is offers the following advantages :

- The implicit formulation of the continuity equation for the unknown ζ^{n+1} makes the scheme unconditionally stable for any time step Δt, allowing to get rid of the Courant–Friedrichs–Lewy (CFL) condition. This leads to a saving of computational time as using a larger time-step lowers the number of time-iterations.
- Respect to a fully implicit formulation the explicit formulation of the two momentum equations allows a cheap direct solving without matrix inversion for the unknowns U^{n+1} and V^{n+1}, reducing the dimensions of the implicit system to one third and leading to a reduction of the solving time by a factor of 30.

The space discretization is made using the finite element approach on triangular elements in the horizontal direction, while a finite volume discretization can be employed in the vertical direction with either z coordinates, sigma or hybrid sigma on top of z.

1.3. Solving the implicit system

The implicit system to be solved for the unknown ζ^{n+1} is a sparse linear systems of equations, solving it can be done in different ways:

- using sparse direct solvers performing per e.g. LU factorization resulting in some kind of Gauss elimination.
- using iterative methods like the preconditioned Krylov (sub)space solver; per e.g. the simplest iterative method uses the Jacobi iteration.

a) Original SparseKit solver

At the time in which this thesis started, the MPI domain partitioning of the source code based on the grid nodes decomposition had just been implemented by CNR and at this intermediary step of the

development of the MPI version, the solver of the implicit linear system of equations was not parallelized and the whole system was solved by every MPI rank. An explicit time formulation of the system could be solved in parallel but was not performant. The solver of the implicit time system implemented in SHYFEM was based on the serial sparsekit [07] FORTRAN library developed by the University of Minnesota for solving sparse linear systems of equations. The following steps were employed to solve the implicit system with sparsekit :

1. Every rank owns a local right hand side and local solution vector, as well as its own version of the full matrix.
2. Each rank owns as well its own copy of the full matrix and full vectors and at every iteration each rank populates the elements of the matrix corresponding to the owned nodes
3. Then every rank calls an all-reduce MPI communication of the full matrix so that every rank receives the full-summed matrix.
4. The local solution and right hand side vectors are subject to an all-gather MPI communication to make the full system available for every single MPI process.
5. Every rank performs a matrix conversion of the matrix from coo to the csr format required by PETSc
6. Every process then solves the whole system
7. Finally every process saves back the owned part of the full solution vector into the local solution vector.

The steps 2, 3, 4, 5 and 6 are all critically expensive tasks that can be avoided by using a parallel distributed solver. The performance of the solver being one of the main aspects of the performance of the software, it was a priority to allow it to be solved in parallel in an optimized way.

b) The PETSc solver

There are many parallel libraries implementing already optimized linear solvers; among them, PETSc is a well-known library under constant development with a large community of users. These guarantees together with individual experiences and opportunities of collaboration and support by CINECA and nVIDIA brought the consortium to choose the PETSc library.

The Portable, Extensible Toolkit for Scientific Computation (PETSc) [1] is a widely used suite of open source software libraries for the parallel solution of linear and nonlinear algebraic equations for large-scale scientific applications modeled by partial differential equations. PETSc provides parallel distributed arrays and supports MPI, and GPUs through CUDA or OpenCL, as well as hybrid MPI-GPU parallelism. PETSc also contains TAO, the Toolkit for Advance Optimization which provides an interface to create solvers and customize them to a particular application. PETSc includes numerous parallel linear and nonlinear equation solvers and provides interfaces for C, C++, Fortran and Python codes.

With respect to the steps described in section 1.3.a), with a parallel solver like PETSc the memory usage should be reduced, as the single processes work with a distributed memory, storing only the portion of the matrix and system vectors corresponding to the portion of domain they have been attributed to. Moreover the MPI communication time would be reduced, as the exchanges of the whole computational domain won't be needed anymore and every process will exchange only the matrix and vector values corresponding to the ghost nodes and only with the neighbour processes

sharing those nodes. Doing so the matrix ghost-cell values will be exchanged only once at every iteration, just before solving the system and in the same way the ghost vector values will be exchanged after solving the system. The matrix conversion from one format to another is another time-expensive operation that will be avoided by storing directly the data in the required format (e.g by calling the PETSc API MatSetValues for each local element-wise matrix).

2. Benchmarking and profiling the original state of the model

By the time in which this thesis was done, the MPI domain partitioning of the source code based on the grid nodes decomposition had just been implemented and were functional only for two-dimensional simulations, for this reason the benchmarks and profilings were all done for two dimensional-simulations only.

2.1. Test case description

The test-case used for benchmarking uses a grid and associated bathymetry covering the whole Mediterranean Sea. This grid was created and shared by ISMAR, it is non-structured, made of 398388 triangular elements and 216764 nodes. The simulations were run in 2D and include wind forcing.

The figure 1 represents the computational domain ; it uses intense colors ranging from blue to brown to represent the sea bottom depth from deep to shallow waters. On the right hand side one can see the extents of the computational grid covering the whole Mediterranean sea up to the Gibraltar strait; while on the left hand side a zoomed view of the northern Adriatic Sea illustrates the mesh coarsening happening in the coastal areas.

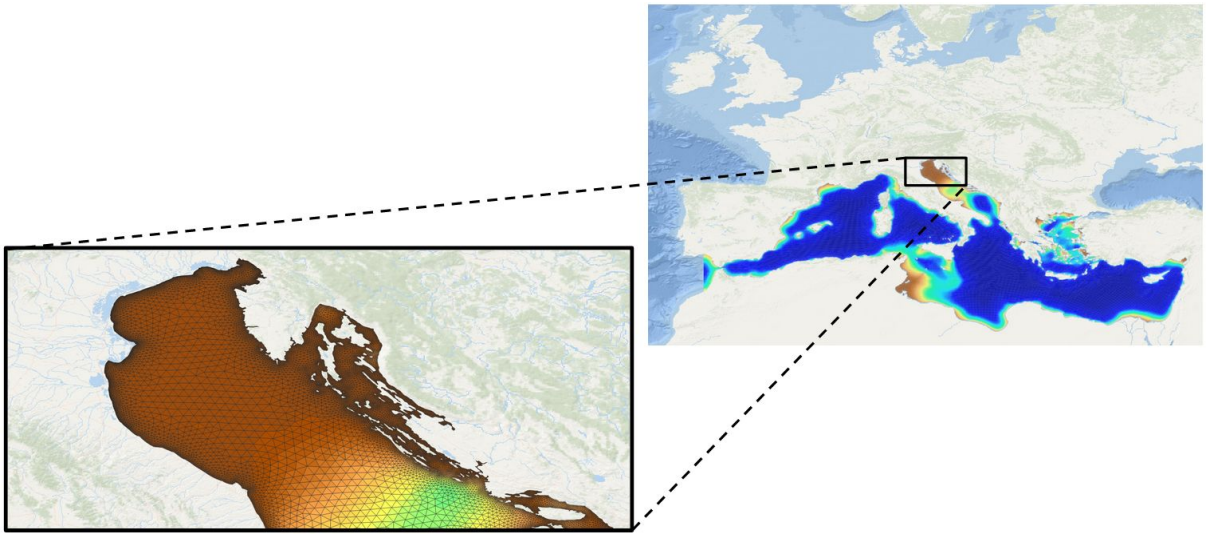


Fig. 1 Domain global view and zoom on the Adriatic Sea region

2.2. CINECA clusters

The choice was made to test the implementations and run the benchmarks and profilings on two different clusters of the CINECA supercomputing center, namely Galileo and Marconi-100, using two different compiler and profiling tool family : Intel and GNU, in order to insure that the stability and performance of the new implementations would be portable to other systems.

a) Galileo

The GALILEO supercomputer [08] is one of the national Tier-1 systems for scientific and industrial research. The full characteristics of GALILEO are the followings:

Model: IBM NeXtScale cluster
Nodes characteristics:

The system includes a total of 1084 compute nodes, plus 8 login nodes and 8 service nodes for I/O and cluster management. All the nodes have the same common characteristics:

- 2 x 18-cores Intel Xeon E5-2697 v4 (Broadwell) at 2.30 GHz
- 128 GB of RAM memory
- interconnected through an Intel OmniPath with OPA v10.6 capable of a maximum bandwidth of 100Gbit/s between each pair of nodes.

Among the 1084 nodes compute nodes, 1022 of them have only CPU while the others are equipped with GPUs:

- 60 compute nodes are equipped with 2 nVidia K80 GPU each
- 2 compute nodes are equipped with 2 nVidia V100 GPU each

b) Marconi-100

MARCONI 100 [09] is the new accelerated cluster acquired by Cineca within PPI4HPC European initiative. This system opens the way to the pre-exascale Leonardo Supercomputer expected to be installed in 2021. The characteristics of Marconi 100 make it suitable both for running simulations but also for artificial intelligence applications. It is available from April 2020 to the Italian public and industrial researchers. Its Peak performance is about 32 PFlops.
Model: IBM Power9 AC922
Nodes characteristics:

The system includes a total of 980 compute nodes, plus 8 Login nodes. All the nodes have the same common characteristics:

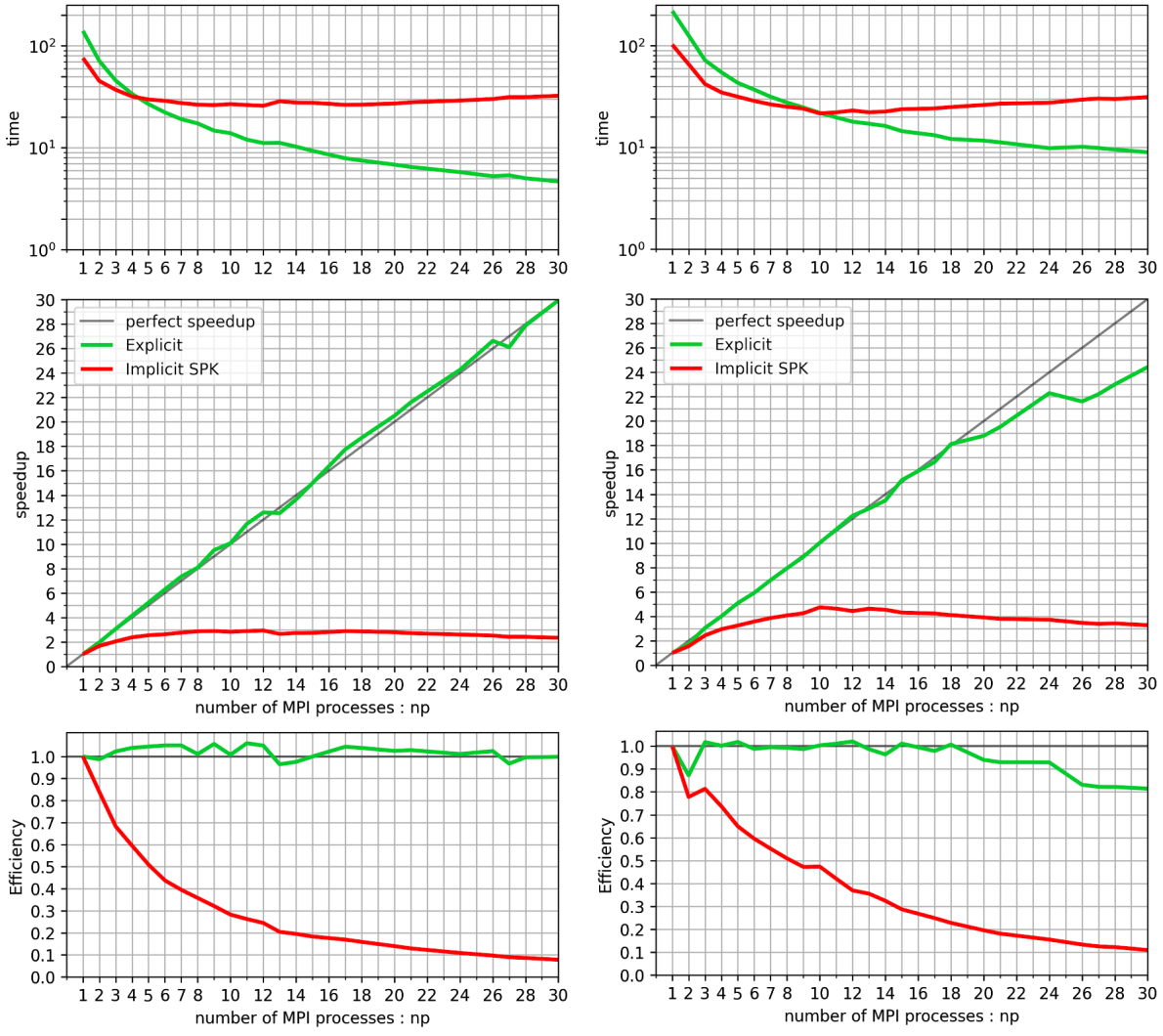
- 2 x 16-cores IBM POWER9 AC922 at 3.1 GHz with hyperthreading x4
- 4 x NVIDIA Volta V100 GPUs, Nvlink 2.0, 16GB
- 256 GB of RAM memory
- internal network made by Mellanox Infiniband EDR DragonFly+
- Memory Bandwidth (nominal/peak freq.) 220/300 GB/s

2.3. Benchmarking the explicit and semi-implicit runs

Depending on the cluster, the original source code was compiled using different characteristics and the following optimization flags were used :

- using intel fortran compiler with flags -O3 -march=native -funroll-loops -xhost -xcore-avx2 -axCORE-AVX2,AVX on Galileo
- using gcc compiler with flags -mcpu=native -ffast-math -ftree-vectorize -ftree-vectorizer-verbose=0 -funroll-loops on Marconi 100

and then run on both clusters for a simulation time of 600 seconds using a 5 seconds timestep, producing the graphics of the figure 2 computing the MPI wall time that the process 0 needs to compute the loop on the time iterations.



a) Galileo cluster *b) Marconi 100*

Fig 2. Time of execution, speedup and efficiency of the SHYFEM program using either the original sparsekit solver to solve the implicit system for the unknown ζ^{n+1} or an explicit system; for the clusters : a). Galileo and b). Marconi 100

Using the wall time the speedup was computed by speedup(n)=wall_time(np=1)/wall_time(np=n) and the efficiency by efficiency(n)=speedup(n)/n. To run better statistics, a whole compute node was

reserved and every run was repeated 6 times for every number of processes and every solver, keeping the smallest run time among the 6 identical runs.

As anticipated, on both clusters for a single MPI process (np=1) the semi-implicit run solving the **implicit** system for the unknown ζ^{n+1} is **about twice faster** than the run solving the fully explicit scheme :

- on Galileo, 76 seconds with the implicit system instead of 140 seconds for the explicit one
- on Marconi-100, 102 seconds for the implicit against 220 seconds in explicit

On Galileo the **code scales perfectly with the fully-explicit formulation** (on Marconi-100 as well, up to 18 processes) with a speedup approaching the ideal one, while as expected it is absolutely not the case when the implicit system is being solved. The solver cannot scale as it solves the whole system of equations for every mpi process ; this is the reason why the sparsekit solver must be substituted by another one able to scale with the number of MPI processes.

A consequent difference in the computational time can be observed between the two clusters, the runs are slower on Marconi 100 with respect to Galileo, which shouldn't be the case given that Marconi 100 employs more recent hardwares with higher clock rate frequency. The origin of this difference will be identified in the profilings presented in the next section.

For the test-case considered in this study the initialization time of the SHYFEM program ranges from 1.5 to 6 seconds on Galileo and from 1.5 to 3.5 seconds on Marconi 100. It is a relatively short time that we chose to consider of minor interest given that a program like SHYFEM is usually used to run simulations much longer than one day (ranging from weeks to hundreds of years of simulation). Moreover, as the initialization time is almost independent with respect to the chosen solver and to the length of the simulation, only the time spent in the time loop was benchmarked and presented figure 2 in order to save computational resources. In fact, another strategy would have been to run longer simulations to make the initialization time completely negligible, for example running a 1 day long (86400 seconds) simulation, but this would have brought to identical results with a much higher computational cost.

2.4. Profilings with Intel tool suite on Galileo

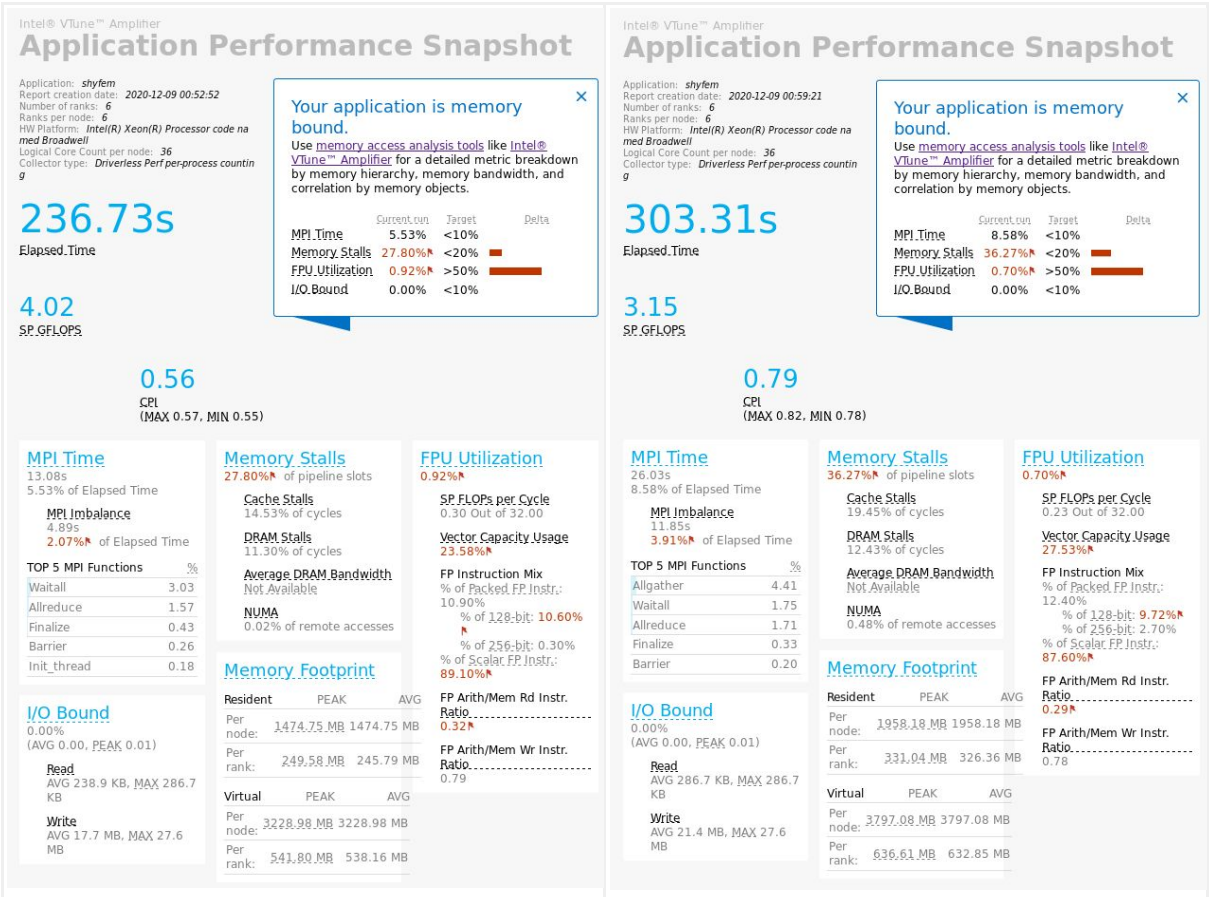
The Galileo cluster is based on Intel technologies, therefore the Intel compiler was used to compile the libraries installed on the cluster and among them the PETSc library. Consequently, we used the Intel libraries and tools to compile, analyze and optimize the applications running on Galileo.

The profilings were all made on a simulation of 6000 seconds using a 5 seconds timestep, so that the time spent initializing the program would be less than 3% of the total wall time for both the implicit and explicit system. The code was compiled with both optimization and debugging flags and run with 6 MPI processes ; all the Intel profiling tools were run on the same node of Galileo reserved in exclusivity.

To extract the informations needed by the intel tools SHYFEM was compiled adding the `-g -traceback` flags summed to the optimization flags given precedently.

2.4.1. Intel Application Performance Snapshot (APS)

The Intel Application Performance Snapshot software [10] allows to have a quick overview of the application's performance and highlights the most critical capacities. The APS outputs of the fully-explicit and semi-implicit simulations are presented in figure 3.



a) explicit system *b) implicit system solved with sparsekit*

Fig 3. outputs of Intel Application Performance Snapshot

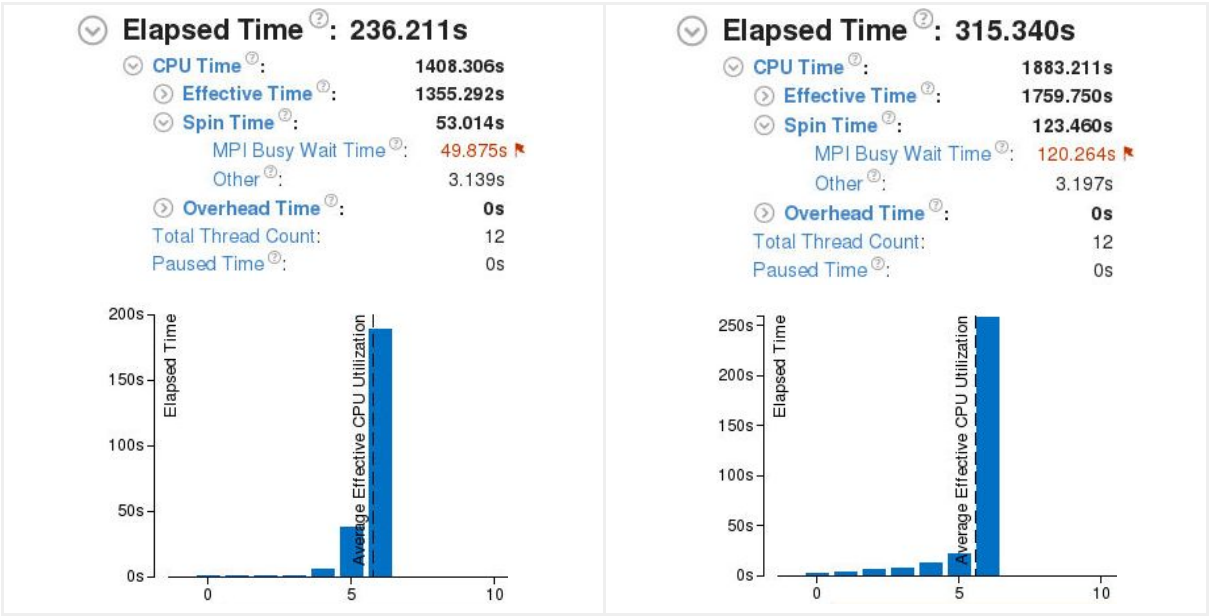
The outputs of Intel APS are clear, the memory stalls are elevated making the code completely memory bounded which is the reason of the low use of the floating point unit FPU (0.5% to 0.68%) and of the limited vectorisation. The MPI calls take a reasonable amount of time, about 3% of the total time; the imbalance between the processes is relatively high: between 8% and 14% for the implicit formulation. Input/Output operations are extremely low, which could be expected as the initialisation time is low respect to the total time that the program has been running, and given the fact that no outputs were produced.

The memory footprint of the semi-implicit run in figure 3.b) is about 25% higher than the one of the fully-explicit run, which is justified by the allocation by every process of arrays for the full matrix and vectors of the implicit system.

Comparing the elapsed time indicated by APS with the one of figure 2 for np=6, we can estimate that the overhead caused by the APS collection of data is reasonably small to not alter the results of the main cost voices.

2.4.2. Intel VTune Profiler

The Intel VTune Profiler [11] is able to locate performance bottlenecks by analyzing the code, isolating issues, and delivering insights for optimizing performance on modern processors. We used VTune to obtain graphical representations of the CPU usage and to analyse in detail the hotspots of the programs; the outputs of VTUNE are presented figure 4 and 5



a) explicit system *b) implicit system solved with sparsekit*

Fig 4. summary outputs of VTune Amplifier

We can notice from the summary output of VTUNE presented figure 4 that the average CPU utilization of the fully explicit run figure 4a is slightly higher than the semi-implicit run of figure 4b. It makes sense given the more elevated number of MPI communication caused by the Sparsekit configuration in which every system must send the full local matrices and vectors to all the other processes.

Comparing the elapsed time indicated by VTUNE in figure 4 with the one of figures 2 and 3 we can estimate that the overhead caused by the VUNE collection of data is reasonably small to not alter the results of the main cost voices.

Function / Call Stack	CPU Time	Spin Time		Ove...
	Effective Time by Utilization	MPI B...	Other	Other
hydro_intern	348.825s	0s	0s	0s
hydro_zeta	173.928s	0s	0s	0s
hydro_transports_final	83.902s	0s	0s	0s
setnar	63.225s	0s	0s	0s
pmpi_waitall_	0.080s	62.361s	0.520s	0s
uvtopr	49.470s	0s	0s	0s
tria_multi	49.177s	0s	0s	0s
pmpi_allreduce_	36.282s	1.620s	3.200s	0s
system_assemble	40.684s	0s	0s	0s
setdepth	39.501s	0s	0s	0s
momentum_advective_stability	37.384s	0s	0s	0s
pmpi_barrier_	30.690s	1.710s	3.269s	0s
setweg	34.350s	0s	0s	0s
uvtop0	31.030s	0s	0s	0s
ttov	27.721s	0s	0s	0s
copy_uvz	27.632s	0s	0s	0s
inext	27.096s	0s	0s	0s
setnod	25.411s	0s	0s	0s
bottom_friction	21.086s	0s	0s	0s
locssp	19.704s	0s	0s	0s
write	19.238s	0s	0s	0s
energ3d	17.879s	0s	0s	0s
nknel	15.624s	0s	0s	0s
setwnk	12.021s	0s	0s	0s
hydro_internal_stability	10.522s	0s	0s	0s
__intel_avx_rep_memset	9.665s	0s	0s	0s
setzev	8.811s	0s	0s	0s
get_elems_around	7.553s	0s	0s	0s
convert_distributed	7.352s	0s	0s	0s
hydro_transports	7.288s	0s	0s	0s
__intel_avx_rep_memcpy	6.956s	0s	0s	0s
pmpi_allgather_	0.010s	5.752s	0.050s	0s

a) explicit system

Function / Call Stack	CPU Time	Spin Time		Ove...
	Effective Time by Utilization	MPI B...	Other	Other
ilut	287.269s	0s	0s	0s
amux	279.917s	0s	0s	0s
lusol	194.679s	0s	0s	0s
hydro_intern	138.113s	0s	0s	0s
coocsr	108.957s	0s	0s	0s
hydro_zeta	89.780s	0s	0s	0s
__intel_avx_rep_memset	86.155s	0s	0s	0s
pmpi_allgather_	6.773s	72.728s	0.270s	0s
system_solve_global	48.395s	0s	0s	0s
system_assemble	46.099s	0s	0s	0s
pmpi_waitall_	0.030s	45.266s	0.369s	0s
hydro_transports_final	41.445s	0s	0s	0s
ddot	35.656s	0s	0s	0s
csort	31.956s	0s	0s	0s
bcgstab	29.353s	0s	0s	0s
uvtopr	26.168s	0s	0s	0s
setnar	25.084s	0s	0s	0s
__intel_avx_rep_memcpy	24.566s	0s	0s	0s
setdepth	23.173s	0s	0s	0s
pmpi_allreduce_	19.861s	0.925s	1.681s	0s
tria_multi	19.572s	0s	0s	0s
_int_free	14.678s	0s	0s	0s
momentum_advective_stability	12.823s	0s	0s	0s
setnod	12.141s	0s	0s	0s
uvtop0	10.788s	0s	0s	0s
inext	10.761s	0s	0s	0s
setweg	10.330s	0s	0s	0s
write	10.230s	0s	0s	0s
energ3d	9.721s	0s	0s	0s
copy_uvz	9.204s	0s	0s	0s
qsplit	8.057s	0s	0s	0s
ttov	7.502s	0s	0s	0s

b) implicit system solved with sparsekit

Fig 5. call-graph outputs of VTune Amplifier

We can see that on figure 5 where the most expensive calls in terms of time are evidentiated on top of the list that solving the explicit system (figure 5.a)) the most expensive calls are the hydro_intern and hydro_zeta calls performing the system matrix assemblies, followed by hydro_transports_final which contains as well a loop on the elements of the domain Solving instead the implicit system with Sparsekit (figure 5.b)) shows that the code spends more time in MPI calls and in the solver functions ilut, amuxm, lusol and coocsr, which was expected given that every process solves the whole system.

2.5. Profiling with HPCToolkit on Marconi 100

On the Marconi-100 cluster, the pre-installed libraries are all compiled using the GNU compilers. It was chosen to install and use HPCtoolkit [12], an integrated suite of tools for measurement and analysis of the performance of serial codes, threaded codes (e.g. pthreads, OpenMP), MPI, and hybrid (MPI+threads) parallel codes. The Spack [13] package manager tool was used for installation on the cluster, enabling mpi and gpu support, with specifically chosen versions of gcc and CUDA (spack install hpctoolkit%gcc@8.4.0+mpi+cuda ^cuda@10.1.105%gcc@8.4.0).

The hpctoolkit profilings presented during this thesis were done one after the other on the same Marconi 100 node reserved exclusively so that no other external process would interfere with the

profilings. The runs that were performed are the same 6000s long simulations with a 5 seconds time-step that were done with the Intel tool suite.

2.5.1. Profiling time and cache loads & misses of the explicit run

The profiling presented in figure 6 uses the fully explicit formulation to get rid of the solver routines of the implicit system which, in the original state of this work, do not scale with the number of mpi processes. The call list was ordered according to the time of the calls in terms of CPU time, the first, second and fifth more time-expensive calls are free and malloc, totalizing 50% of the total CPU TIME. Those calls are memory issues, located in the subroutine hydro_intern that will be further analyzed and are responsible for the more elevated computational time observed on Marconi-100 respect to Galileo (figure 2). The first other expensive call are those already identified on Galileo : the hydro_intern and the hydro_zeta routine running loops on the elements filling in the system matrix.

Top-down view | Bottom-up view | Flat view

Scope	CPUTIME (sec)	L1-DCACHE-LOADS:	L1-DCACHE-LOAD-MISSES	LLC-LOADS:Sum	LLC-LOAD-MISSES
Σ Experiment Aggregate Metrics	2.45e+03 100 %	7.44e+14 100 %	2.85e+13 100 %	1.37e+13 100 %	1.18e+12 100 %
__free	5.18e+02 21.1%	1.73e+14 23.3%	2.24e+10 0.1%	3.64e+09 0.0%	2.08e+07 0.0%
__malloc	4.80e+02 19.6%	8.71e+13 11.7%	7.36e+11 2.6%	2.59e+11 1.9%	3.19e+08 0.0%
_int_malloc	2.28e+02 9.3%	7.23e+13 9.7%	1.59e+11 0.6%	7.48e+10 0.5%	3.39e+08 0.0%
hydro_intern_	1.46e+02 5.9%	5.03e+13 6.8%	1.96e+12 6.9%	6.73e+11 4.9%	2.04e+10 1.7%
hydro_zeta_	1.42e+02 5.8%	2.73e+13 3.7%	1.06e+13 37.3%	4.75e+12 34.7%	6.97e+11 59.0%
hydro_transports_final_	7.93e+01 3.2%	1.64e+13 2.2%	3.99e+12 14.0%	1.84e+12 13.5%	3.36e+11 28.4%
pthread_spin_lock	6.07e+01 2.5%	2.28e+13 3.1%	1.14e+09 0.0%	5.50e+08 0.0%	5.47e+07 0.0%
PAMI_Context_trylock_advancev	5.77e+01 2.4%	3.09e+13 4.2%	7.16e+09 0.0%	5.74e+08 0.0%	2.65e+08 0.0%
memset@plt	5.28e+01 2.2%	3.94e+12 0.5%	2.75e+11 1.0%	1.39e+09 0.0%	1.94e+07 0.0%
tria_multi_	4.97e+01 2.0%	2.18e+13 2.9%	3.13e+10 0.1%	9.26e+09 0.1%	
setnar_	4.27e+01 1.7%	1.43e+13 1.9%	8.22e+11 2.9%	4.63e+11 3.4%	1.39e+10 1.2%
setdepth_	3.52e+01 1.4%	1.20e+13 1.6%	2.06e+11 0.7%	1.21e+11 0.9%	1.33e+09 0.1%
copy_uvz_	3.25e+01 1.3%	6.81e+12 0.9%	2.56e+09 0.0%	8.68e+09 0.1%	3.82e+08 0.0%
setweg_	3.07e+01 1.3%	2.55e+13 3.4%	2.08e+11 0.7%	4.58e+10 0.3%	1.41e+09 0.1%
uvtopr_	2.93e+01 1.2%	1.15e+13 1.5%	5.73e+11 2.0%	2.07e+11 1.5%	2.40e+09 0.2%
__memcpy_power7	2.82e+01 1.1%	4.73e+12 0.6%	3.10e+10 0.1%	2.44e+10 0.2%	7.16e+08 0.1%
inext_	2.46e+01 1.0%	7.39e+12 1.0%	1.22e+12 4.3%	5.18e+11 3.8%	4.67e+09 0.4%
momentum_advective_stability_	2.28e+01 0.9%	7.31e+12 1.0%	1.50e+10 0.1%	3.12e+09 0.0%	9.17e+08 0.1%
system_assemble_	2.27e+01 0.9%	1.12e+13 1.5%	3.33e+12 11.7%	2.50e+12 18.3%	2.59e+10 2.2%
setnod_	2.18e+01 0.9%	8.58e+12 1.2%	1.62e+12 5.7%	1.49e+12 10.9%	3.86e+10 3.3%
__memset_power8	2.16e+01 0.9%	1.89e+12 0.3%	2.42e+11 0.9%	4.80e+09 0.0%	6.60e+07 0.0%
uvtop0_	2.12e+01 0.9%	9.17e+12 1.2%	5.77e+11 2.0%	1.81e+11 1.3%	5.38e+09 0.5%
PAMI_Context_lock	1.96e+01 0.8%	7.80e+12 1.0%	6.21e+08 0.0%	1.87e+08 0.0%	5.37e+07 0.0%
opal_progress	1.91e+01 0.8%	1.24e+13 1.7%	5.53e+08 0.0%	1.76e+08 0.0%	3.67e+07 0.0%
bottom_friction_	1.73e+01 0.7%	6.95e+12 0.9%	1.57e+09 0.0%	8.78e+07 0.0%	1.00e+07 0.0%
nknel_	1.69e+01 0.7%	3.44e+12 0.5%	4.24e+11 1.5%	2.58e+10 0.2%	1.31e+08 0.0%
energ3d_	1.69e+01 0.7%	7.47e+12 1.0%	1.98e+11 0.7%	1.08e+11 0.8%	6.01e+09 0.5%
CCMI::Executor::ShmemBarrier::advance_impl	1.38e+01 0.6%	5.97e+12 0.8%	4.87e+08 0.0%	2.33e+07 0.0%	9.83e+07 0.0%
convert_distributed_	1.16e+01 0.5%	1.95e+12 0.3%	1.55e+11 0.5%	1.33e+11 1.0%	1.11e+10 0.9%
start_libcoll_blocking_collective	1.14e+01 0.5%	4.64e+12 0.6%	5.44e+08 0.0%	8.31e+08 0.0%	3.97e+07 0.0%
hydro_internal_stability_	1.09e+01 0.4%	1.86e+12 0.2%		8.43e+07 0.0%	3.03e+07 0.0%

Fig 6. HPC toolkit profiling when solving the fully explicit system

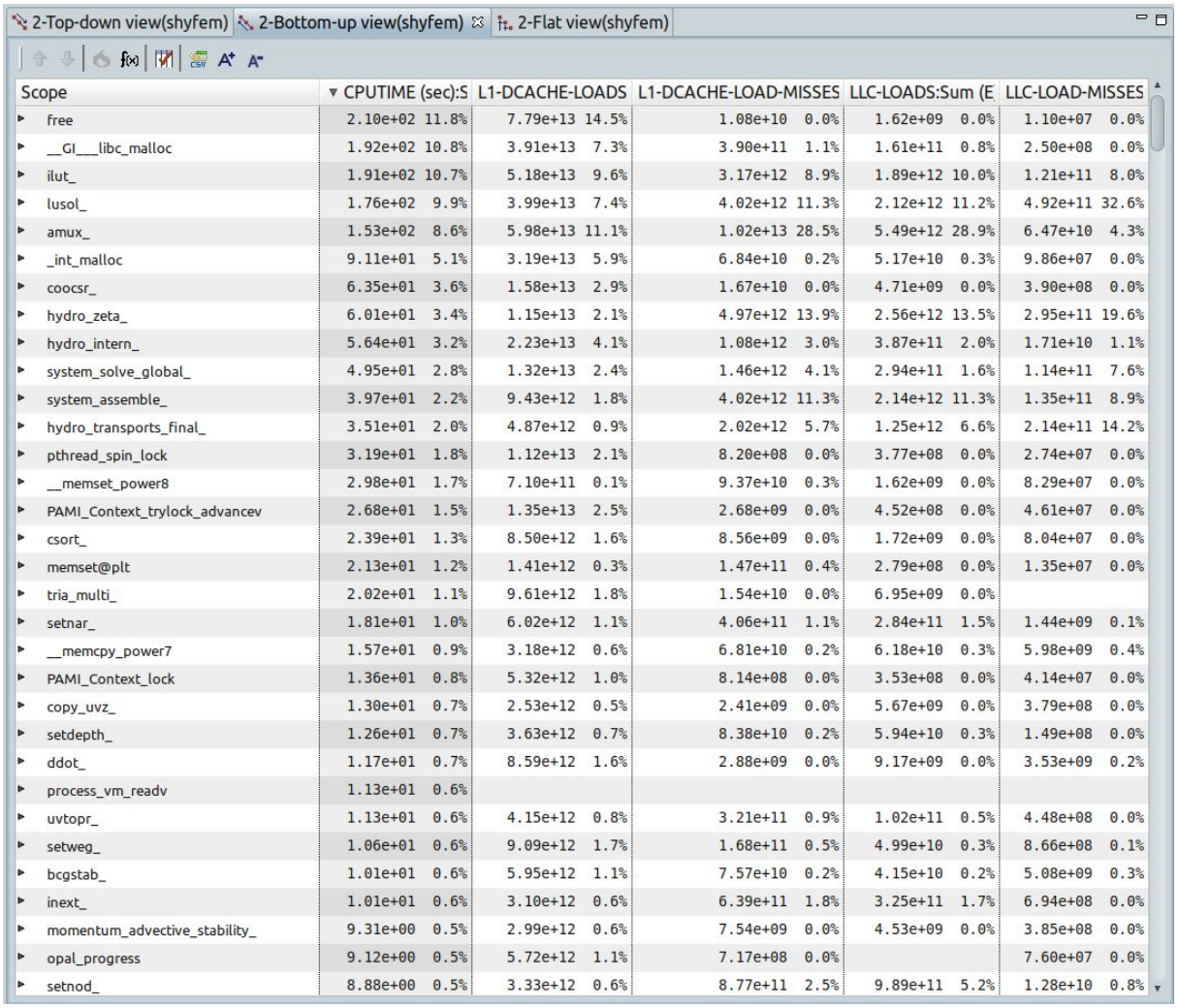
A data collection of the loads & misses of the first and Last Levels of Cache (L1 and LLC) were requested to HPCtoolkit, and show that the call to hydro_zeta is responsible for 37% of the L1 and 59% of the LLC cache misses, which shows that the hydro_zeta routine is the major cause of the memory bound issue identified by Intel APS in the figure 3. The hydro_transports_final is as well in cause, being responsible of 14% and 28% of the L1 and LLC cache misses. Finally, although hydro_intern is a major voice in the time-consumption, it causes less cache misses (7% and 1.6% of L1 and LLC misses) with respect to the hydro_zeta and hydro_intern, indicating that other factors might be acting there.

The run presented figure 6 lasted about 414 seconds, among which 1.5 seconds of initialization and 412.5 seconds for the time-iterative loop. This is coherent with the time to solution estimated in the figure 2.b) for np=6, so that we can estimate that the overhead caused by the HPCToolkit data collection is reasonably small to not alter the results of the main cost voices.

2.5.2. Profiling time and cache loads & misses of the semi-implicit run

The SHYFEM profiling presented in figure 7 was done with the implicit formulation of the ζ system to visualize the cost of the solver routines at the original state of this work when the whole implicit system was still solved without distribution of the work among the processes. The run lasted about 301.5 seconds, among which 1.5 seconds of initialization and 300 seconds for the time-iterative loop.

As expected, in the present view presented in figure 7 evidentiating on top of the list the most expensive calls in terms of CPU time, the first calls are all routines of the sparsekit solver : ilut, amux, lusol. Then come the free and malloc calls as well as the hydro_zeta and hydro_intern calls. We can observe that amux, lusol present elevated cache loads and misses due to the large size of the system that they solve.



2-Top-down view(shyfem) | 2-Bottom-up view(shyfem) | 2-Flat view(shyfem)

Scope	CPUTIME (sec):S	L1-DCACHE-LOADS	L1-DCACHE-LOAD-MISSES	LLC-LOADS:Sum (E	LLC-LOAD-MISSES
free	2.10e+02 11.8%	7.79e+13 14.5%	1.08e+10 0.0%	1.62e+09 0.0%	1.10e+07 0.0%
__GI___libc_malloc	1.92e+02 10.8%	3.91e+13 7.3%	3.90e+11 1.1%	1.61e+11 0.8%	2.50e+08 0.0%
ilut_	1.91e+02 10.7%	5.18e+13 9.6%	3.17e+12 8.9%	1.89e+12 10.0%	1.21e+11 8.0%
lusol_	1.76e+02 9.9%	3.99e+13 7.4%	4.02e+12 11.3%	2.12e+12 11.2%	4.92e+11 32.6%
amux_	1.53e+02 8.6%	5.98e+13 11.1%	1.02e+13 28.5%	5.49e+12 28.9%	6.47e+10 4.3%
_int_malloc	9.11e+01 5.1%	3.19e+13 5.9%	6.84e+10 0.2%	5.17e+10 0.3%	9.86e+07 0.0%
coocsr_	6.35e+01 3.6%	1.58e+13 2.9%	1.67e+10 0.0%	4.71e+09 0.0%	3.90e+08 0.0%
hydro_zeta_	6.01e+01 3.4%	1.15e+13 2.1%	4.97e+12 13.9%	2.56e+12 13.5%	2.95e+11 19.6%
hydro_intern_	5.64e+01 3.2%	2.23e+13 4.1%	1.08e+12 3.0%	3.87e+11 2.0%	1.71e+10 1.1%
system_solve_global_	4.95e+01 2.8%	1.32e+13 2.4%	1.46e+12 4.1%	2.94e+11 1.6%	1.14e+11 7.6%
system_assemble_	3.97e+01 2.2%	9.43e+12 1.8%	4.02e+12 11.3%	2.14e+12 11.3%	1.35e+11 8.9%
hydro_transports_final_	3.51e+01 2.0%	4.87e+12 0.9%	2.02e+12 5.7%	1.25e+12 6.6%	2.14e+11 14.2%
pthread_spin_lock	3.19e+01 1.8%	1.12e+13 2.1%	8.20e+08 0.0%	3.77e+08 0.0%	2.74e+07 0.0%
__memset_power8	2.98e+01 1.7%	7.10e+11 0.1%	9.37e+10 0.3%	1.62e+09 0.0%	8.29e+07 0.0%
PAMI_Context_trylock_advancev	2.68e+01 1.5%	1.35e+13 2.5%	2.68e+09 0.0%	4.52e+08 0.0%	4.61e+07 0.0%
csort_	2.39e+01 1.3%	8.50e+12 1.6%	8.56e+09 0.0%	1.72e+09 0.0%	8.04e+07 0.0%
memset@plt	2.13e+01 1.2%	1.41e+12 0.3%	1.47e+11 0.4%	2.79e+08 0.0%	1.35e+07 0.0%
tria_multi_	2.02e+01 1.1%	9.61e+12 1.8%	1.54e+10 0.0%	6.95e+09 0.0%	
setnar_	1.81e+01 1.0%	6.02e+12 1.1%	4.06e+11 1.1%	2.84e+11 1.5%	1.44e+09 0.1%
__memcpy_power7	1.57e+01 0.9%	3.18e+12 0.6%	6.81e+10 0.2%	6.18e+10 0.3%	5.98e+09 0.4%
PAMI_Context_lock	1.36e+01 0.8%	5.32e+12 1.0%	8.14e+08 0.0%	3.53e+08 0.0%	4.14e+07 0.0%
copy_uvz_	1.30e+01 0.7%	2.53e+12 0.5%	2.41e+09 0.0%	5.67e+09 0.0%	3.79e+08 0.0%
setdepth_	1.26e+01 0.7%	3.63e+12 0.7%	8.38e+10 0.2%	5.94e+10 0.3%	1.49e+08 0.0%
ddot_	1.17e+01 0.7%	8.59e+12 1.6%	2.88e+09 0.0%	9.17e+09 0.0%	3.53e+09 0.2%
process_vm_readv	1.13e+01 0.6%				
uvtopr_	1.13e+01 0.6%	4.15e+12 0.8%	3.21e+11 0.9%	1.02e+11 0.5%	4.48e+08 0.0%
setweg_	1.06e+01 0.6%	9.09e+12 1.7%	1.68e+11 0.5%	4.99e+10 0.3%	8.66e+08 0.1%
bcgstab_	1.01e+01 0.6%	5.95e+12 1.1%	7.57e+10 0.2%	4.15e+10 0.2%	5.08e+09 0.3%
inext_	1.01e+01 0.6%	3.10e+12 0.6%	6.39e+11 1.8%	3.25e+11 1.7%	6.94e+08 0.0%
momentum_advective_stability_	9.31e+00 0.5%	2.99e+12 0.6%	7.54e+09 0.0%	4.53e+09 0.0%	3.85e+08 0.0%
opal_progress	9.12e+00 0.5%	5.72e+12 1.1%	7.17e+08 0.0%		7.60e+07 0.0%
setnod_	8.88e+00 0.5%	3.33e+12 0.6%	8.77e+11 2.5%	9.89e+11 5.2%	1.28e+10 0.8%

Fig 7. HPC toolkit profiling when solving the semi-implicit system

It is worth noticing that in both cases, HPC toolkit does not show the time of the MPI calls, this is much probably due to a wrong setup of the analysis that doesn't collect them.

2.6. Analysis of the METIS partitioning

To understand better the MPI unbalance a short analysis of the partitioning computed using the METIS [14] library obtained by the calls of the SHYFEM-tools was performed. The number of ghosts and maximal unbalance are represented in figure 8; is worth noticing that the number of ghost elements is very high for some configurations, particularly for 2 and 6 MPI processes. The unbalance is limited to 3% because this is the maximal unbalance requested in the entry parameters of the METIS calls. The amount of MPI communications between one MPI processor and it's neighbour is directly proportional to the number of ghosts, which is why the origin of such an elevated number of ghosts must be identified.

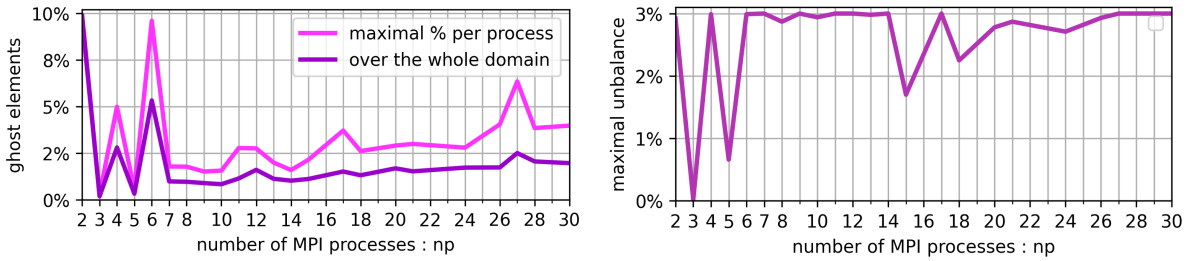


Fig 8. ghosts and maximal unbalance

To understand the origin of this elevated number of ghosts the same METIS configuration as the one used to plot the figure 8 was employed to represent the partitioning obtained for 2 and 4 MPI processes in figure 9 where the different colors represent different domains of the partitioning and the ghost are represented in black.

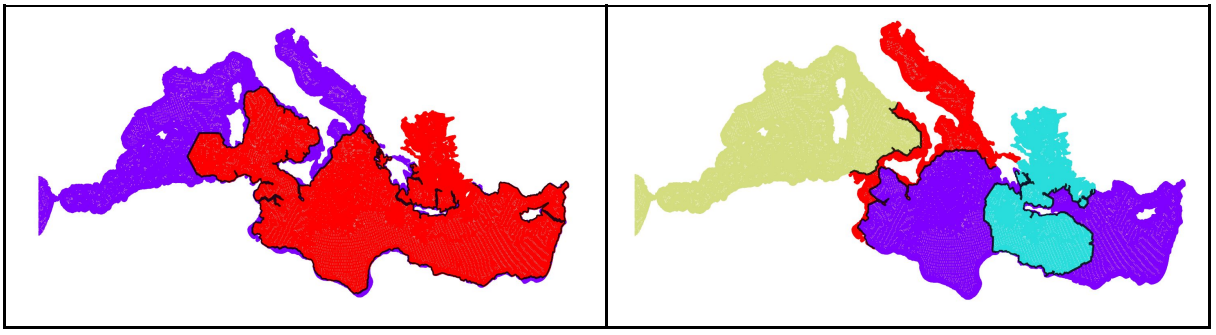


Fig 9. Original partitioning for 2 and 4 MPI processes

It can be seen in figure 9 that in both cases the partitioning creates very elongated boundaries between the domains, this is particularly visible for the 2-domains partitioning where the ghosts line englobes the Tyrrhenian, Ionian, Libyan and Levantine Seas, reaching the coast line only in the Aegean Sea, between Greece and Turkey. It is much probable that the complexity of the domain and the grid refinement happening along the coasts, together with the constraint applied by the maximal unbalance value prevents METIS from doing a domain partitioning that reduces the number of ghosts.

In order to confirm this hypothesis the configuration requested to METIS was modified, increasing step by step the maximal allowed unbalance up to the value that caused a sudden re-organization of the partitioning with a number of ghosts falling suddenly below 0.7% and leading to the repartition shown figure 10. Unfortunately the so-obtained unbalance is about 5% for 2 MPI processes and reaches 10% for 4 MPI processes . On Galileo using either the Sparsekit or PETSc solver the computational time gain was only of the order of 1.5% for 2 MPI procs and a performance loss was even observed with 4 MPI processes due to the too elevated unbalance.

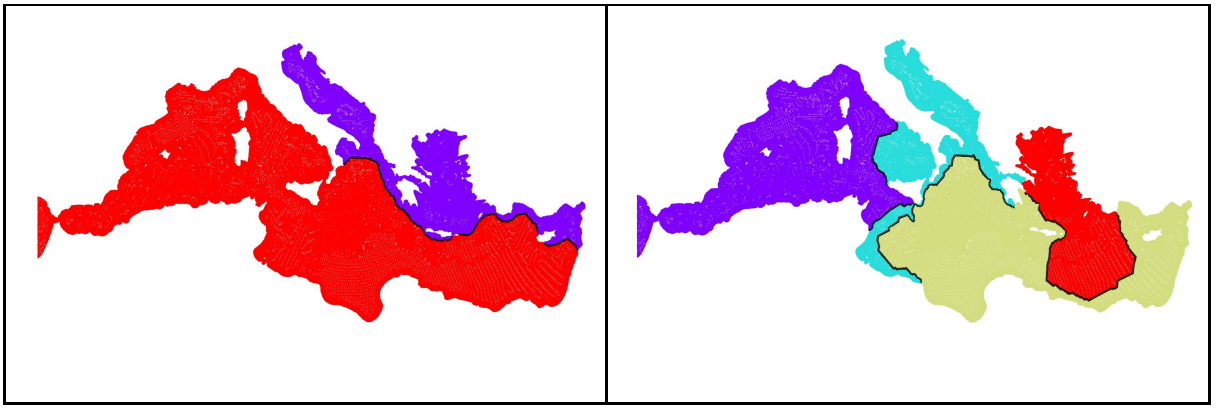


Fig 10. METIS partitioning increasing the maximal allowed unbalance for 2 and 4 MPI processes

Looking for a better setup the constraint of having a single continuous domain was removed, leading to the partitioning presented figure 11 where less than 0.5% of ghosts are present and the unbalance is less than 0.3%. This is for sure the best configuration, and a performance gain of about 7% is observed running SHYFEM with this configuration. In the actual state of development of SHYFEM, the simulation managed to run using the non-contiguous domain but there are some internal issues that must be solved before such partitioning can be used without risks of changing the results of the simulations. However, this short analysis evidentiates that fixing these issues to allow SHYFEM to run on a non-contiguous domain would be worth it.

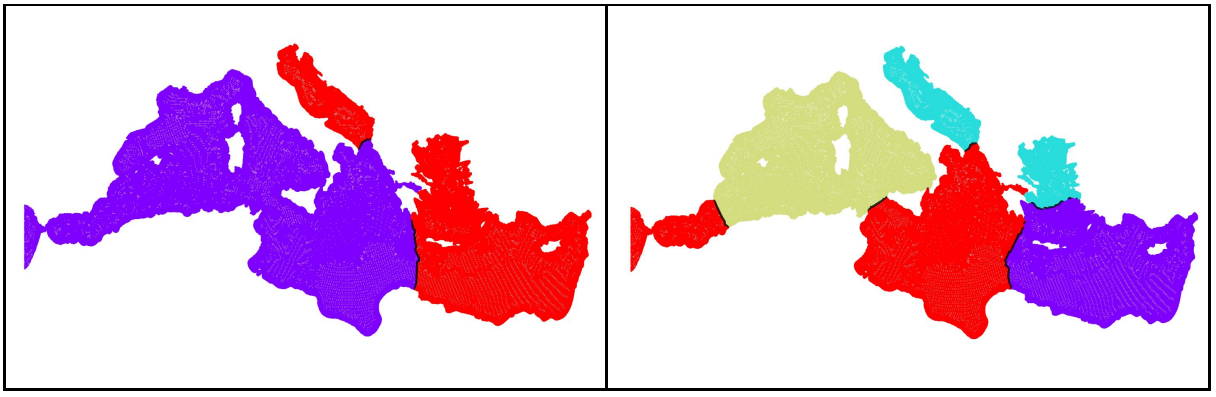


Fig 11. METIS partitioning removing the contiguous domain constraint 2 and 4 MPI processes

3. Implementations

During the implementation of the coupling with PETSc and the continuous porting of the source code between Galileo, Marconi 100 and Taurus [15] which is another x86 cluster used during the GPU hackathon, numerous issues were encountered related to the original source code, such as out-of-memory accesses, uninitialised memory, or floating point comparisons made using == or \= operators. They were identified using different debugging approaches, like the use of compilation debugging flags, running the program using the GDB [16] debugger or performing Intel Inspector [17] analysis. Every time it was possible, a fix was proposed, otherwise the error was highlighted in the source code, and the changes were pushed to a separate branch of the repository for further analysis. Two git branches were dedicated to these works, FIXME and FIXED.

Best programming practices of object-oriented programming have been used. In all the implementations made within SHYFEM a care was given to set the implemented modules contents to be private to protect them from external interferences and make public only the necessary objects or procedures. Pragma directives were employed to reduce the if-conditions executed at run time for the insertion of debugging/verbose outputs, or to activate the calls of the routines specific to one solver or another. When if-conditions couldn't be avoided they were written trying to optimize branch prediction. Numerous checks were done to try to anticipate eventual incompatibilities of setup. All the changes were systematically tested on different architectures: POWER on M100 and x86 on Galileo. The tests were performed both in debug and optimized mode using the flags implemented in shyfem/meson.build.

3.1. Calling the PETSc solver

3.1.1. Structure of the implementation

Object-oriented programming was employed to plug-in the PETSc toolkit in SHYFEM for the solution of the sparse linear system; it gives the possibility, for any future need, to use the same module to solve any other linear system in the SHYFEM model and to be able, at run-time, to select and parameterize the options of each PETSc solver by choosing among all the possible solvers and preconditioners offered by the PETSc library.

Shyfem already provides an interface to several other solvers based on sparsekit, pardiso, and paralution ; each one has its own source files containing the same interface-routines and the user can choose at compile time which source files to include in the compilation in order to use a determined solver. To keep this intercompatibility a source file containing the interface routines was created and named mod_petsc.f ; it is made of the module mod_petsc that contains only a few routines and allows to keep almost unchanged the calls that were setup for the previous solvers:

mod_petsc contains :

- subroutine system_initialize
- subroutine system_init
- subroutine system_solve(n,z)
- subroutine system_get(n,z)
- subroutine system_finalize

the last one, system_finalize, was not present in the source files of the other solvers and had to be added to finalize PETSc and free the memory of the newly created modules.

The mod_petsc module stores also the pointer to the object zeta_system of type petsc_system defined in mod_petsc_system, as well as the names of the PETSc (and AmgX, if required) configuration files to be given at run-time by the user.

The other two modules composing this PETSc implementation are the core of the coupling : mod_petsc_global and mod_petsc_system; they are saved in files of identical names and contain all the PETSc library calls.

The first module, mod_petsc_global, initializes and finalizes the PETSc library, but not only, it contains a serie of routines that intend to analyze the SHYFEM-grid configuration and domain partitioning, building all the informations required by PETSc to initialize the Matrix and Vector objects. To do so the routines of this module identify and store the ghost nodes and all the non-zeros of the sparse matrix, those on the block-diagonal and those outside the block-diagonal, as required by PESTc. The procedures contained in this module are not specific to a given system of equations, they must be called only once, in the sense that they are completely independent of the systems of equations that SHYFEM could solve.

The second module, mod_petsc_system, defines the type petsc_system which contains all the procedures and objects required by any system that must be solved by PETSc ; it uses the sparsity pattern and MPI distribution defined in mod_petsc_global.

3.1.2. Call tree

The following simplified call-tree helps to understand better the implementations highlighting the main components of the system resolution. The routines of the SHYFEM model are represented in black, the routines of the mod_petsc module are in blue, those of mod_petsc_global are in orange and those of mod_petsc_system are in red. The calls to the PETSc library are represented in green and some comments are added in grey.

shyfem
- ➔ **system_initialize**
  - **petsc_global_initialize**
    - **petsc_initialize**
  - **petsc_global_create_setup**
    - **petsc_create_indexes**
    - create arrays storing indexes attributed to the nodes, reordering them so that so that every rank owns a contiguous block of the matrix and vectors.
    - **pestc_identify_non_zeros_and_ghosts**
    - loops optimized to save time and memory, create sparsity pattern and ghost arrays needed to setup A,B and X
  - **allocate(zeta_system)**
  - **zeta_system=petsc_system(PETSc_zeta_config,AmgX_zeta_config)**
  - read configuration files, choose PETSc or AmgX
  - **system_zeta%create_objects**
  - create matrix A and vectors B and X, setting up the ghost nodes and the matrix/vector MPI distribution
    - **MatCreate, MatSet, ...**
    - **VecCreate,VecSet, VecMPISetGhost ...**
  - **petsc_global_close_setup**
  - free sparsity pattern and ghost arrays

- ➔ **loop on time**
  - **hydro**
    - **system_init**
      - **reset_zero_entries**
      - need to set to zeros because mat and vec values are then added
    - **hydro_zeta**
      - **loop on elements**
      - optimisation: element matrix and vectors are set directly in the loop to avoid copying them for every element
        - **zeta_system%mat3x3(:,:) = …**
        - **zeta_system%vecx3(:) = …**
        - **zeta_system%add_matvec_values(element_number)**
          - **MatSetValues** optimisation: adding whole vector
          - **VecSetValues** optimisation: adding whole vector
      - **zeta_system%add_full_rhs(length,values(:))**
        - **VecSetValue** adding only values of the inner nodes
    - **system_solve**
      - **zeta_system%matvec_assemble**
        - **Mat/VecAssemblyBegin/End** barrier
      - **IF (ITER ==1) zeta_system%init_solver** the solver must be initialized after the first assembly
        - **zeta_system%init_solver_PETSc**
          - **KSPCreate, KSPSet, KSPGetPC, …**
          - **PCSetType, PCFactorSetMatSolverType, …**
      - **zeta_system%solve**
        - **KSPSetOperators**
        - **KSPSolve**
    - **system_get**
      - **zeta_system%get_solution**
        - **VecGhostUpdateBegin/End**
        - **VecGetArrayF90**
        - **VecRestoreArrayReadF90**
- ➔ **system_finalize**
  - **zeta_system%destroy_solver**
    - **zeta_system%destroy_solver_PETSc**
      - **KSPDestroy**
  - **zeta_system%destroy_matvecobjects**
    - **VecDestroy**
    - **MatDestroy**
  - **deallocate(zeta_system)**
  - **petsc_global_finalize**
    - **PetscFinalize**

These implementations can be found in the PETSc branch of the https://github.com/CeliaLaurent/shyfem repository.

3.1.3. Optimisation steps

a) The first implementation that was made suffered from very elevated memory costs due to very large arrays used in petsc_create_indexes and petsc_identify_non_zeros_and_ghosts. Optimizing them in terms of memory first brought much higher initialization times, about 35 seconds for our test-case. Further optimization allowed to find an implementation that reduces both time and memory consumption, making petsc_create_indexes and petsc_identify_non_zeros_and_ghosts very affordables. In fact they don't appear anymore in the profilings and the time to initialize the whole SHYFEM model is almost unchanged respect to the previous solver, for our test-case between 2 and 4 seconds depending on the MPI configuration.

b) In the very first implementation of PETSc in SHYFEM, the time spent by the program in the hydro_zeta loop on the elements setting the matrix and vector values was about twice the

time of the actual solving of the system. To optimize this section of the code a particular attention was given to the 3x3 matrix and length-3 vector used to insert the element contributions to the system matrix and vector. In fact the first implementation was copying the arrays passing them by value through the routine add_matvec_values, allocating then in mod_petsc_system at every timestep and for every element, and they were then given to PETSc through the MatSetValue and VecSetValue calls. This approach demonstrated very elevated time costs and was optimized in two ways: by giving the values to PETSc using the vectorized calls MatSetValues and VecSetValues , and by using in the hydro_zeta loop pointers to pre-allocated 3x3 matrix and length-3 vector, allocated as permanent members of the mod_petsc_system module.

c) An additional optimization was performed in the loop on the elements run inside the hydro_zeta routine, where the elements were originally accessed in a disordered way with respect to the storage in memory of the content of the arrays. The access in the order of the memory storage was restored. Moreover, an ulterior modification was brought by limiting the assembly to the "unique" elements of every process, instead of including all the inner elements and all the ghosts elements. The unique elements are a portion of the ghost elements assigned to a given domain if it owns at least 2 of the 3 nodes composing the triangular element. The ghosts elements shared between 3 domains were assigned to the unique elements of only one of these domains.

3.1.4. Practical usage and customization

To use PETSc it must be activated at compile time, in the same way as it was done for the other solvers by setting the SOLVER variable to PETSc in the Rules.make file and indicating the path to $PETSC_HOME. The shyfem/fem3d/Makefile will then handle the choice of the right source files to be compiled and define the use_PETSc pragma directive that commands the compilation of the PETSc-specific instructions in the main source code, such as the inclusion of mod_petsc, the call to system_finalize or the declarations of the pointers to the zeta_system 3x3matrix and length-3-vector. Adding the Verbose pragma directive makes the PETSc implementation very verbose.

To use the meson build system instead of the Makefile to compile SHYFEM with PETSc, it is enough to set the command line option -D use_PETSc=true and eventually add the path to the $PETSC_HOME directories.

➢ meson . build_dir -D use_PETSc -D PETSc_dir=$PETSC_HOME

To parameterize the solver, preconditioner or matrix/vectors at run-time, the flag -petsc_zeta_config must be given in argument to the SHYFEM executable, followed by the name of the file, per example :

➢ mpirun -np 2 shyfem -mpi -petsc_zeta_config <petsc_file_name> input_file.str

This file name will then be passed to the constructor of the object zeta_system of type petsc_system, and the setups requested in the file will automatically replace the default ones. The default configuration that was implemented is the default one of PETSc [18], it is based on the GMRES [19] solver using the point block-Jacobi iterative preconditioner, it can be used "as it is" to run SHYFEM on the CPUs without having to give to SHYFEM any PETSc configuration file.

Examples of configuration files for sequential and parallel configurations, on the CPU or GPU are given in the shyfem/meson_tests folder.

3.2. Interfacing PETSc with NVIDIA's AmgX solver using AmgXWrapper

3.2.1. GPU hackathon

The implementation and testing of an interface to **AmgXWrapper** [05] coupling our PETSc implementation with the **NVIDIA's AmgX multi-GPU linear-algebra package** [04] was performed during the **GPU Hackathon** managed by the OpenACC organization, in Partnership with NVIDIA, Oak Ridge Leadership Computing Facility and Argonne National Laboratory at the **Helmholtz-Zentrum Dresden-Rossendorf** on **7 & 14-16th of september 2020** : **Interfacing PETSc with NVIDIA's AmgX multi-GPU linear-algebra package using AmgXWrapper**.

The members of the team :

- *Célia Laurent, OGS Italian National Institute of Oceanography and Geophysics & MHPC Master in High Performance Computing, ICTP/SISSA*
- *Georg Umgiesser, ISMAR Italian Institute of Marine Sciences - CNR*
- *Simone Bnà, CINECA Italian supercomputing centre*
- *Eric Pascolo, CINECA Italian supercomputing centre*
- *Tobias Huste, Helmholtz-Zentrum Dresden-Rossendorf*
- *Jeffrey Kelling, Helmholtz-Zentrum Dresden-Rossendorf*

During this hackathon the following works were carried on :

- a C-interface was implemented to call the c++ AmgXWrapper from the fortran source code, it is available at the following address : https://github.com/tobiashuste/amgx-c-wrapper
- Three Fortran subroutines themselves calling the routines of the C-interface were set- up to substitute the PETSc equivalent calls :
    - A function that initiates the AmgX solver (zeta_system%init_solver_AmgX)
    - A function setting and solving the system, to be called at every timestep (zeta_system%solve_AmgX)
    - The finalization of the AmgX solver (in zeta_system%destroy_solver_AmgX)

All the others components of the PETSc module are instead used directly, such as the PETSc initialization and finalization, the routines computing non-zero indexes and ghosts identification, the creation of the PETSc matrix and vector objects, the resetting to zero of the objects, the setting of the matrix and vector values, as well as the routine getting back the solution vector from PETSc.

The implementation of the AmgX init_solver and solve functions were integrated in the PETSc branch of the https://github.com/CeliaLaurent/shyfem repository.

3.2.2. Practical usage and customization

To use AmgX the c-interface developed during the hackathon must be downloaded from https://github.com/tobiashuste/amgx-c-wrapper, directly into the shyfem/amgx-c-wrapper folder. The AmgX SOLVER must be chosen in the Rules.make file and the installation path of AmgX, AmgXWrapper, amgx-c-wrapper and CUDA must be provided. The shyfem/fem3d/Makefile will then define the use_PETSc and use_AmgX pragma directives. In this way the source code is only one for both solvers, but depending on whether the use_AmgX pragma directive is or not defined, the linking of the AmgX and CUDA libraries will or won't be required at compile time, which allows to run on clusters without GPUs where AmgX and CUDA won't be installed.

When using meson build system it is enough to set the command line option -D use_AmgX=true and if needed add the links to the AmgX, AmgXWrapper and CUDA [20] HOME directories :

➢ meson . build_dir -D use_PETSc=true -D use_AmgX=true -D amgxwrapper_dir=$AMGXWRAPPER_HOME -D amgx_dir=$AMGX_HOME -D cuda_dir=$CUDA_HOME .

Once compiled, the use of AmgX can be enabled at run time by setting -shyfem_solver amgx in the .petscrc file stored in the directory where SHYFEM will be executed. With the same compiled code, it is still possible to use PETSc by setting -shyfem_solver petsc .

Running AmgX requires a configuration file to be present in the running directory, by default "AmgX.info" is used, but any other file can be used instead by adding the flag -amgx_zeta_config followed by the name of the file, at run-time:

➢ mpirun -np 2 shyfem -mpi -petsc_zeta_config <petsc_file_name> -amgx_zeta_config <amgx_file_name> input_file.str

Examples of such a configuration file can be found in the shyfem/tests_meson directory containing the files used for the integration tests.

3.3. Meson - ninja build system with integration tests

A new build system based on Meson [06] was implemented to offer a modern and automatized alternative to the original hand-written code-heavy Makefile build system of SHYFEM. Meson is implemented in Python and requires the version 3.5 or newer. It can be installed very easily using the Conda or PyPI package managers (conda install meson, sudo pip3 install meson, or pip3 install --user meson without root privileges), or downloading the package from the github repository [21] and executing the python installer,

Meson sets up the precompilation and creates an optimized compilation setup that is then used by the ninja build system [22] to actually perform the compilation.

Meson gives the possibility to work on multiple build directories that can be setup with different options, each of them containing their own object files and executables .

3.3.1. Description of the build system configuration files

A configuration setup for the meson build system was implemented to compile the SHYFEM program, together with the main executable tools included in the SHYFEM tool suite.

The developments can be found in the meson branch of the forked repository https://github.com/CeliaLaurent/shyfem.

The main setup file is shyfem/meson.build , it calls other setup files stored in the subdirectories

- shyfem/femgotm/meson.build to compile the gotm library
- shyfem/tests/meson.build to define the integration tests

Additionally, the files shyfem/meson_options.txt and shyfem/pragma_directives.h.meson_input define the customized command line options allowing the user to customize the compiling rules without having to modify the meson.build file :

- use_PETSc enables or disables the petsc solver (default : false)
- use_AmgX enables or disables the AmgX solver (default : false)
- use_SPK enables or disables the Sparsekit solver (default : false)
- test_zeta makes SHYFEM write the solution vector at every timestep to check validity of the results during integration tests (default : false)
- Verbose enables or disables the Verbose pragma directive ruling the verbose outputs of SHYFEM (default : false)
- num_GPU number of GPUs available for integration tests (tests are run with one GPU per MPI proc) (min:0,max:2, default : 0)
- petsc_dir sets the path to PETSc installation folder (default: 'none')
- amgx_dir sets the path to the installation folder of the AmgX solver (default: 'none')
- amgxwrapper_dir sets the path to the installation folder of AmgXWrapper, the c++ wrapper between PETSc and the AmgX solver (default: 'none')
- cuda_dir sets the path to CUDA installation folder (default: 'none')
- metis_dir sets the path to METIS installation folder (default: 'none')

3.3.2. Command-line setup at pre-compilation step

The different options of compilations can be enabled or disabled at pre-compilation, per eg:

➢ meson . build_dir --buildtype release -Duse_PETSc=true

The --buildtype option is provided by meson, it allows to change the compilation flags between different profiles , per eg: debug, debugoptimized, release

For PETSc, AmgXSolver and AmgXWrapper, both the library and include headers are required during the compilation of SHYFEM. If meson can find the .pc file of the software in the environmental variable PKG_CONFIG_PATH, then there is no need to provide manually the installation folders and everything runs in automatic mode. Otherwise the paths must be provided manually using per example:

➢ meson . build_tmp --buildtype release -D use_PETSc=true -D use_AmgX=true -D amgxwrapper_dir=$AMGXWRAPPER_HOME -D amgx_dir=$AMGX_HOME -D use_GPU=2

For both PETSc and AmgX, the MPI library is required and linked automatically with no need to specify it at pre-compile time, the same is not true for Sparsekit whose compilation does not require the MPI library. To compile it with MPI and allow parallel runs the flag -D use_MPI=true must be added:

➢ meson . build_dir -D use_SPK=true -D use_MPI=true

For METIS, given that only the library is required, and no headers must be included during compilation, if meson can not find the metis.pc file of software in the environmental variable PKG_CONFIG_PATH, then the library will be searched in the environmental variable LIBRARY_PATH. Otherwise, the library can anyway be provided manually through the command line option metis_dir.

Note that using AmgX requires to activate both use_PETSc and use_AmgX, and that both PETSc and Sparsekit (use_PETSc and use_SPK) cannot be activated simultaneously.

3.3.3. Compilation step

Then, the user must go into the newly created build_dir directory, where the actual compilation can be launched by entering the command:

➢ ninja (or ninja -v to run a verbose compilation)

The compilation is automatically run in parallel by the ninja build system.

3.3.4. Running integration tests

The integration tests (together with an automatic recompilation of any modified source file) can be run by command :

➢ ninja test

Those integration tests are run in parallel in an optimized way on a very small and short test-case; their setup is provided in the shyfem/tests_meson directory. According to the precompilation options that were turned on, the maximal number of tests will be automatically effectuated. Different setups of integration tests are provided to run the solvers on both the explicit and implicit systems for 1 and 2 MPI processes dependent on the MPI dependency required. The integration tests will use either Sparsekit or PETSc-CPU depending on the solver that was compiled. Moreover, if the precompilation options included 1 or 2 GPUs (use_GPU=2) an additional PETSc-GPU test will be run ; if the compilation included as well AmgX (use_AmgX) then two additional tests for 1 and 2 MPI processes will be run automatically.

- building in parallel for 1 and 2 mpi procs the binary file (.bas) containing the (eventually partitioned) mesh required by SHYFEM, this step is commanded by the bash script shyfem/tests/mpi_basin.sh running the SHYFEM tools (shyparts, shybas, shypre) with the necessary checks of successful completion
- synchronizing the threads (implementing a sort of barrier using task priorities) so that the simulations do not start until the meshes are ready.
- running in parallel two CPU simulations of the SHYFEM program solving the fully explicit system of equations, a serial one and another one with 2 mpi processes.
- running in parallel two CPU simulations of the SHYFEM program using PETSc, a serial one and another one with 2 mpi processes.
- running one simulation using PETSc on the GPU with 1 MPI proc and 1 GPU

- running in parallel two simulations of the SHYFEM program using the PETSc and the AmgX GPU solver, the first simulation with one mpi proc and 1 GPU while the other simulations uses 2 mpi processes and 2 GPUs.

In every case the test scripts are doing the necessary checks of successful completion, following the instructions of the bash script shyfem/tests/run_and_diff_floating_points.sh

If the test_zeta flag has been set to true during the pre-compilation, the success of the SHYFEM runs rely on a file test_zeta.* (written by SHYFEM when the integration tests are enabled : pragma directive run_tests) which stores the values of the solution vector zeta at every time step. The values contained in this file are compared with those of the reference files stored in shyfem/tests/ by using an awk script written in shyfem/tests/run_and_diff_floating_points.sh which compares the floating point values of both files, computing the L2 and L_infinity norm of the difference between the solution arrays and checking at every timestep that they remain lower than a given threshold (set to 1E-08) given in argument to the bash file. The test is considered to have failed if the norm of the difference between the reference array and the computed array exceeds the threshold at any timestep.

3.4. hydro_intern free and malloc optimization

As shown in section 2.5.1. a large overhead was observed on the Marconi 100 cluster, the first, second and fifth most expensive calls in terms of CPU time were free and malloc calls, totalizing 36.5% of the total CPU TIME and were attributed to the subroutine hydro_intern.

Those calls are memory issues, using the HPTtoolkit profilings their origin was identified in the allocation of a few arrays whose dimension is proportional to the number of vertical layers nlvdi, even in our 2d case these memory allocation and de-allocation have a large impact because hydro_intern is called inside a loop on the elements.

```
real hact(0:nlvdi+1)
real rhact(0:nlvdi+1)
real alev(0:nlvdi)
double precision rmat(10*nlvdi)
double precision smat(-2:2,2*nlvdi)
double precision s2dmat(-1:1,2)
double precision rvec(6*nlvdi)
double precision rvecp(6*nlvdi)
double precision solv(6*nlvdi)
```

This issue was solved by simply removing these allocations from hydro_intern, and moving them into the module mod_hydro where they were allocated only once, at the beginning of the simulation, and deallocated at the end of the run. The implementation of this optimization is stored in a specific branch of the https://github.com/CeliaLaurent/shyfem repository called optim_hydromalloc.

4. Results

To give an overview of the improvements brought by the new implementations, the same configuration of the profiling tools and benchmarked that were run section 1 will be presented in section 4 using the PETSc solver. The effects of the optimization presented in section 3.4 will be assessed on Marconi 100 by analyzing first the profiling of the simulation solving the explicit system, to compare it directly with the one presented in the second chapter ; then the profiling of the simulation of the implicit system solved by PETSc. The benchmarks of the new implementations will be presented to evaluate the efficiency of the PETSc implementation and the efficiency of the additional optimization, both on the CPU and on the GPU. The benchmarks of the PETSc-GPU solver and PETSc-AmgX solver will be as well presented. Finally, the performances of the new building method based on Meson will be described and we will show examples of the outputs illustrating the tasks that it can perform.

The Intel APS snapshots of figure 3 and figure 12 were obtained in the exact same conditions reserving an Galileo node in exclusive and used the same simulation length which makes them completely confrontables. In the same way for the VTUNE profilings of figures 4, 5, 13 and 14 that can be compared among themselves, and the same thing applies to the hpctoolkit profilings of figures 6, 7, 15 and 16 that were obtained in the exact same conditions on a Marconi 100 node reserved in exclusivity.

4.1. Final Profilings of time and cache loads/misses

4.1.1. CPU profilings on Galileo

The Intel APS snapshot figure 12 shows very good performances with respect to the original solver of the explicit system presented in fig 3.a, and even better performances with respect to the Sparsekit simulation using the implicit solver in figure 3.b. The time to solution is reduced by 60% and still allows a reduction of the time spent in the memory stalls (22.6% instead of 27.8%) ; the percentage of time spent in MPI communication remains almost unchanged; the percentage of FPU utilization respect to the original solver of the explicit system decreased very slightly: 0.87% instead of 0.92%, A consequent reduction of the percentage of time spent in Memory stalls is visible, which is even more true given the reduction of the total compute time. As expected even the memory footprint is lower than the one of the original code.

The VTUNE summary presented that the Average Effective CPU utilization is slightly better than the ones of the original explicit-system and sparsekit-implicit-system presented figure 4. As it can be seen of figure 13 and 14, the MatSetValuesMPIAIJ function of the PETSc library is the most expensive call, responsible of the addition figure 13 shows of the 3x3matrix and length-3 vector into the full distributed matrix and vector, followed the hydro_intern and hydro_zeta routines setting the 3x3matrix and length-3 vector values and finally comes the call to hydro_transports_final.

The figure 13 shows the time spent in the different modules/libraries, among them 288 seconds are spent in the PETSc library which corresponds to 35% of the total time.

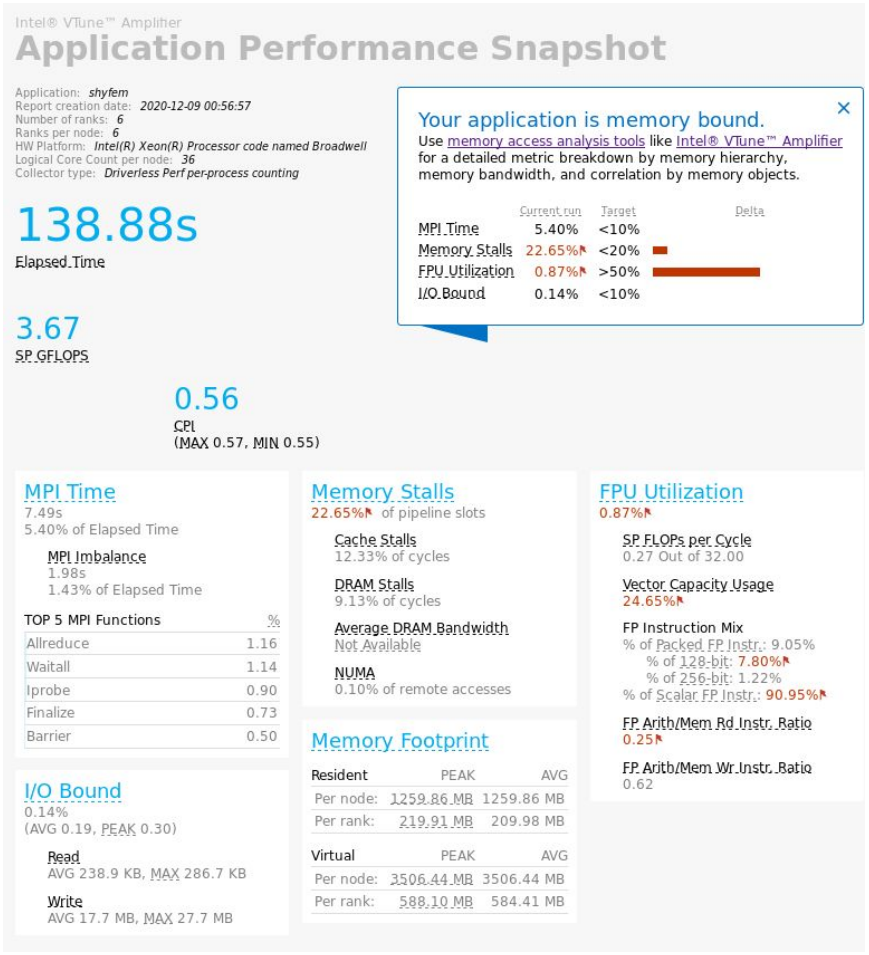


Fig 12. Intel APS snapshot of the new PETSc implementation

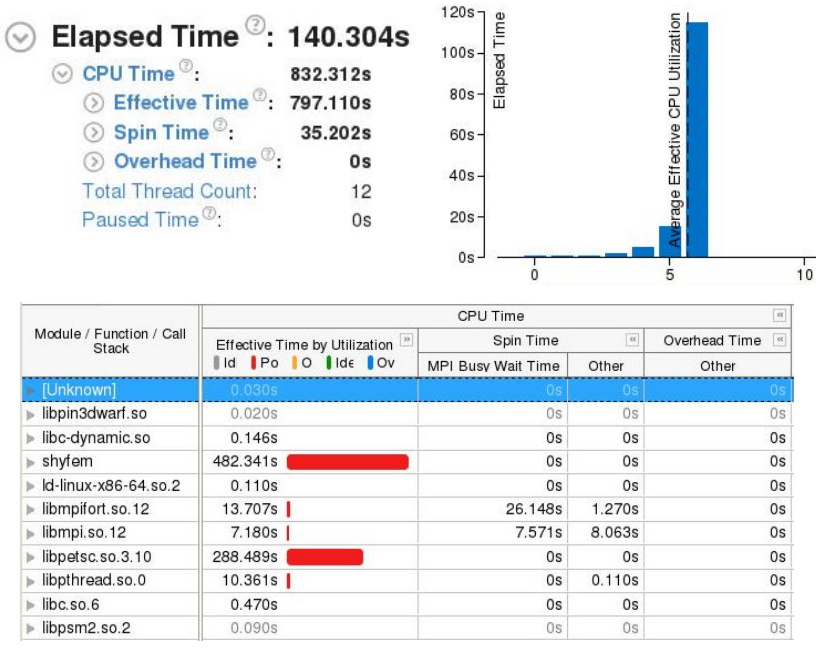


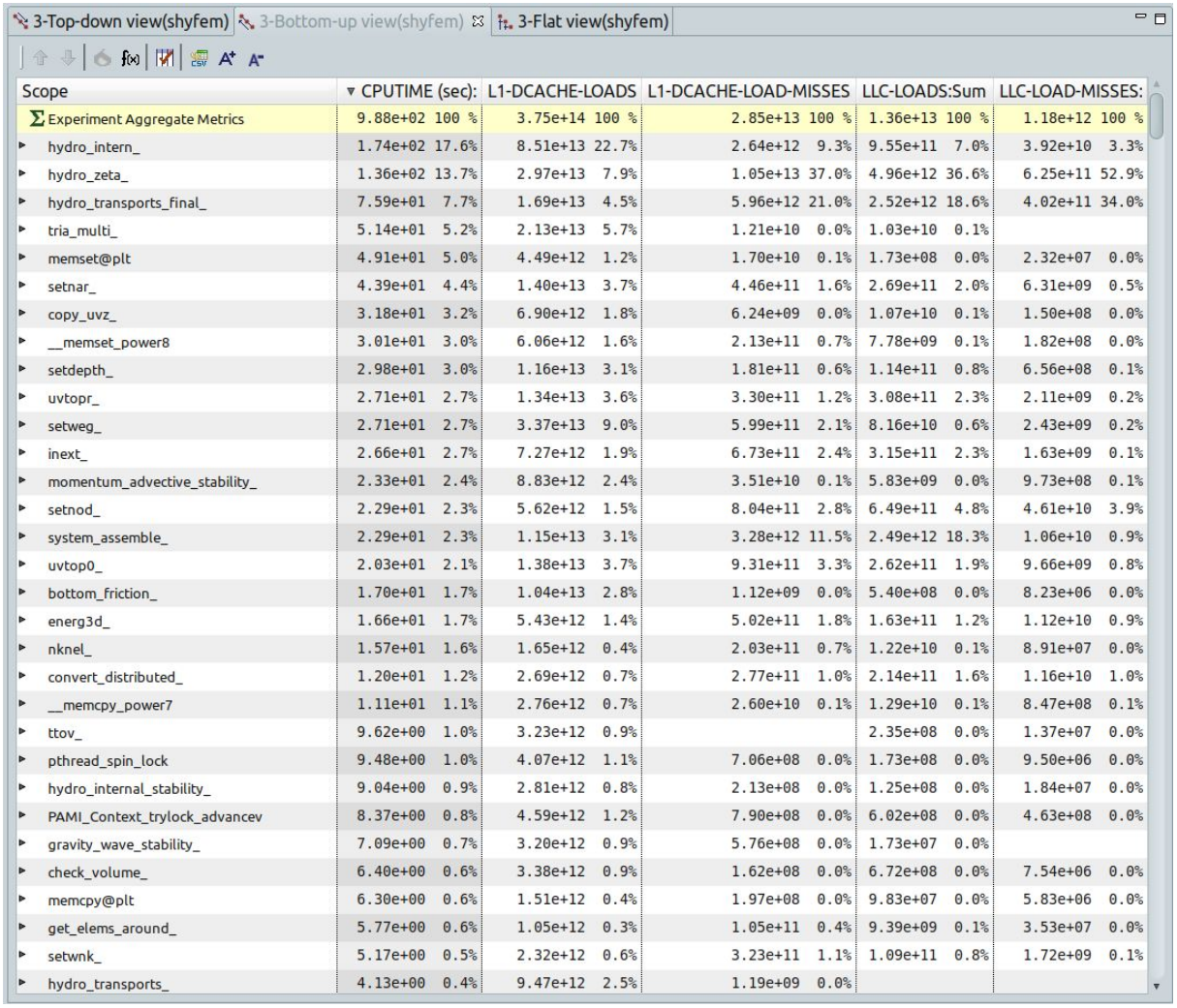
Fig 13. VTUNE summary with the new PETSc implementation

Function / Call Stack	CPU Time			Module
	Effective Time by Utilization	Spin Time	Ove...	
		MPI B...	Other	Other	
MatSetValues_MPIAIJ	155.936s	0s	0s	0s	libpetsc.so.3...
hydro_intern	143.191s	0s	0s	0s	shyfem
hydro_zeta	52.415s	0s	0s	0s	shyfem
hydro_transports_final	41.398s	0s	0s	0s	shyfem
MatLUFactorNumeric_SeqAIJ	40.652s	0s	0s	0s	libpetsc.so.3...
MatSolve_SeqAIJ_NaturalOrdering	26.556s	0s	0s	0s	libpetsc.so.3...
pmpi_waitall_	0.060s	25.000s	0.160s	0s	libmpifort.so...
uvtopr	24.609s	0s	0s	0s	shyfem
setnar	23.625s	0s	0s	0s	shyfem
setdepth	23.398s	0s	0s	0s	shyfem
tria_multi	19.052s	0s	0s	0s	shyfem
MatMult_SeqAIJ	18.409s	0s	0s	0s	libpetsc.so.3...
VecSetValues_MPI	17.877s	0s	0s	0s	libpetsc.so.3...
momentum_advective_stability	12.381s	0s	0s	0s	shyfem
PMPI_Iprobe	1.939s	5.040s	3.923s	0s	libmpi.so.12
energ3d	10.899s	0s	0s	0s	shyfem
uvtop0	10.706s	0s	0s	0s	shyfem
setweg	10.278s	0s	0s	0s	shyfem
write	10.237s	0s	0s	0s	libpthread.so.0
inext	10.113s	0s	0s	0s	shyfem
pmpi_allreduce_	8.663s	0.390s	0.820s	0s	libmpifort.so...
setnod	9.592s	0s	0s	0s	shyfem
copy_uvz	8.384s	0s	0s	0s	shyfem
ttov	7.711s	0s	0s	0s	shyfem
locssp	7.618s	0s	0s	0s	shyfem
__intel_avx_rep_memset	7.541s	0s	0s	0s	shyfem
nknel	6.531s	0s	0s	0s	shyfem
MatAssemblyEnd_SeqAIJ	5.312s	0s	0s	0s	libpetsc.so.3...
convert_distributed	5.029s	0s	0s	0s	shyfem
PMPI_Allreduce	4.391s	0.270s	0.340s	0s	libmpi.so.12
bottom_friction	4.610s	0s	0s	0s	shyfem
hydro_internal_stability	4.471s	0s	0s	0s	shyfem
pmpi_barrier_	3.910s	0.210s	0.280s	0s	libmpifort.so...

Fig 14. VTUNE detailed analysis of the new PETSc implementation

4.1.2. CPU profilings on Marconi-100

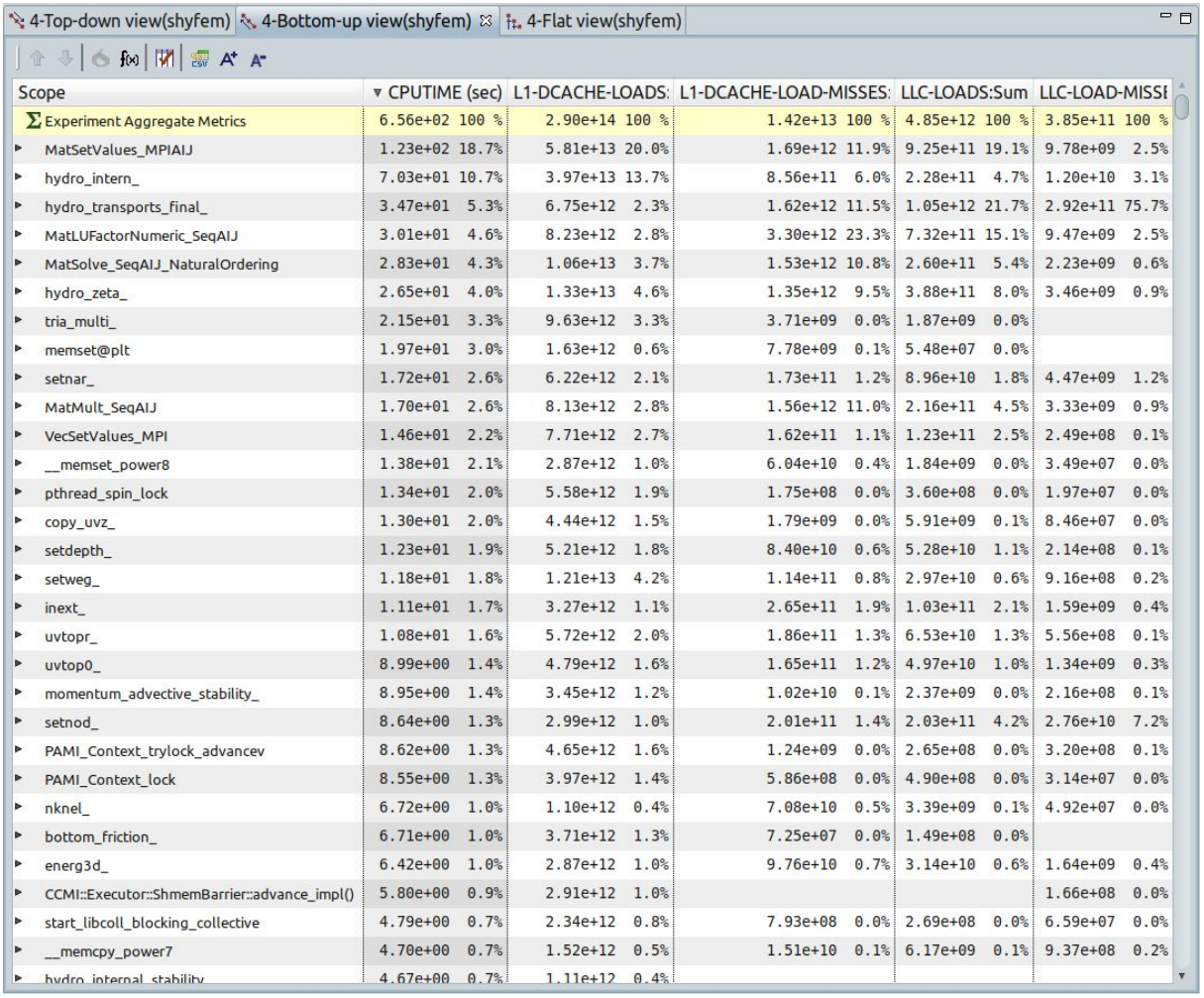
On Marconi 100, a first step was to apply the memory optimization proposed in section 3.4 on the simulation solving the fully explicit system of equations. This profiling is presented in figure 15 and, as expected, the free and malloc functions do not appear anymore, which validates the proposed optimization. The run presented figure 15 lasted about 168 seconds, among which 1.5 seconds of initialization and 166.5 seconds for the time-iterative loop. The simulation time was divided by more than a factor 2.



Scope	CPUTIME (sec):	L1-DCACHE-LOADS	L1-DCACHE-LOAD-MISSES	LLC-LOADS:Sum	LLC-LOAD-MISSES:
Σ Experiment Aggregate Metrics	9.88e+02 100 %	3.75e+14 100 %	2.85e+13 100 %	1.36e+13 100 %	1.18e+12 100 %
hydro_intern_	1.74e+02 17.6%	8.51e+13 22.7%	2.64e+12 9.3%	9.55e+11 7.0%	3.92e+10 3.3%
hydro_zeta_	1.36e+02 13.7%	2.97e+13 7.9%	1.05e+13 37.0%	4.96e+12 36.6%	6.25e+11 52.9%
hydro_transports_final_	7.59e+01 7.7%	1.69e+13 4.5%	5.96e+12 21.0%	2.52e+12 18.6%	4.02e+11 34.0%
tria_multi_	5.14e+01 5.2%	2.13e+13 5.7%	1.21e+10 0.0%	1.03e+10 0.1%	
memset@plt	4.91e+01 5.0%	4.49e+12 1.2%	1.70e+10 0.1%	1.73e+08 0.0%	2.32e+07 0.0%
setnar_	4.39e+01 4.4%	1.40e+13 3.7%	4.46e+11 1.6%	2.69e+11 2.0%	6.31e+09 0.5%
copy_uvz_	3.18e+01 3.2%	6.90e+12 1.8%	6.24e+09 0.0%	1.07e+10 0.1%	1.50e+08 0.0%
__memset_power8	3.01e+01 3.0%	6.06e+12 1.6%	2.13e+11 0.7%	7.78e+09 0.1%	1.82e+08 0.0%
setdepth_	2.98e+01 3.0%	1.16e+13 3.1%	1.81e+11 0.6%	1.14e+11 0.8%	6.56e+08 0.1%
uvtopr_	2.71e+01 2.7%	1.34e+13 3.6%	3.30e+11 1.2%	3.08e+11 2.3%	2.11e+09 0.2%
setweg_	2.71e+01 2.7%	3.37e+13 9.0%	5.99e+11 2.1%	8.16e+10 0.6%	2.43e+09 0.2%
inext_	2.66e+01 2.7%	7.27e+12 1.9%	6.73e+11 2.4%	3.15e+11 2.3%	1.63e+09 0.1%
momentum_advective_stability_	2.33e+01 2.4%	8.83e+12 2.4%	3.51e+10 0.1%	5.83e+09 0.0%	9.73e+08 0.1%
setnod_	2.29e+01 2.3%	5.62e+12 1.5%	8.04e+11 2.8%	6.49e+11 4.8%	4.61e+10 3.9%
system_assemble_	2.29e+01 2.3%	1.15e+13 3.1%	3.28e+12 11.5%	2.49e+12 18.3%	1.06e+10 0.9%
uvtop0_	2.03e+01 2.1%	1.38e+13 3.7%	9.31e+11 3.3%	2.62e+11 1.9%	9.66e+09 0.8%
bottom_friction_	1.70e+01 1.7%	1.04e+13 2.8%	1.12e+09 0.0%	5.40e+08 0.0%	8.23e+06 0.0%
energ3d_	1.66e+01 1.7%	5.43e+12 1.4%	5.02e+11 1.8%	1.63e+11 1.2%	1.12e+10 0.9%
nknel_	1.57e+01 1.6%	1.65e+12 0.4%	2.03e+11 0.7%	1.22e+10 0.1%	8.91e+07 0.0%
convert_distributed_	1.20e+01 1.2%	2.69e+12 0.7%	2.77e+11 1.0%	2.14e+11 1.6%	1.16e+10 1.0%
__memcpy_power7	1.11e+01 1.1%	2.76e+12 0.7%	2.60e+10 0.1%	1.29e+10 0.1%	8.47e+08 0.1%
ttov_	9.62e+00 1.0%	3.23e+12 0.9%		2.35e+08 0.0%	1.37e+07 0.0%
pthread_spin_lock	9.48e+00 1.0%	4.07e+12 1.1%	7.06e+08 0.0%	1.73e+08 0.0%	9.50e+06 0.0%
hydro_internal_stability_	9.04e+00 0.9%	2.81e+12 0.8%	2.13e+08 0.0%	1.25e+08 0.0%	1.84e+07 0.0%
PAMI_Context_trylock_advancev	8.37e+00 0.8%	4.59e+12 1.2%	7.90e+08 0.0%	6.02e+08 0.0%	4.63e+08 0.0%
gravity_wave_stability_	7.09e+00 0.7%	3.20e+12 0.9%	5.76e+08 0.0%	1.73e+07 0.0%	
check_volume_	6.40e+00 0.6%	3.38e+12 0.9%	1.62e+08 0.0%	6.72e+08 0.0%	7.54e+06 0.0%
memcpy@plt	6.30e+00 0.6%	1.51e+12 0.4%	1.97e+08 0.0%	9.83e+07 0.0%	5.83e+06 0.0%
get_elems_around_	5.77e+00 0.6%	1.05e+12 0.3%	1.05e+11 0.4%	9.39e+09 0.1%	3.53e+07 0.0%
setwnk_	5.17e+00 0.5%	2.32e+12 0.6%	3.23e+11 1.1%	1.09e+11 0.8%	1.72e+09 0.1%
hydro_transports_	4.13e+00 0.4%	9.47e+12 2.5%	1.19e+09 0.0%		

Fig 15. HPC Toolkit profiling for the fully explicit system with the hydro_intern memory optimization

The same profiling was performed using the PETSc solver in the implicit system and is presented figure 16, it is in good agreement and consistent with the profiling done on Galileo (with 6000 seconds instead of 600 seconds) presented previously in figure 14 and we can see that here again the free and malloc calls do not appear anymore.



4-Top-down view(shyfem) | 4-Bottom-up view(shyfem) | 4-Flat view(shyfem)

Scope	CPUTIME (sec)	L1-DCACHE-LOADS:	L1-DCACHE-LOAD-MISSES:	LLC-LOADS:Sum	LLC-LOAD-MISSI
Σ Experiment Aggregate Metrics	6.56e+02 100 %	2.90e+14 100 %	1.42e+13 100 %	4.85e+12 100 %	3.85e+11 100 %
MatSetValues_MPIAIJ	1.23e+02 18.7%	5.81e+13 20.0%	1.69e+12 11.9%	9.25e+11 19.1%	9.78e+09 2.5%
hydro_intern_	7.03e+01 10.7%	3.97e+13 13.7%	8.56e+11 6.0%	2.28e+11 4.7%	1.20e+10 3.1%
hydro_transports_final_	3.47e+01 5.3%	6.75e+12 2.3%	1.62e+12 11.5%	1.05e+12 21.7%	2.92e+11 75.7%
MatLUFactorNumeric_SeqAIJ	3.01e+01 4.6%	8.23e+12 2.8%	3.30e+12 23.3%	7.32e+11 15.1%	9.47e+09 2.5%
MatSolve_SeqAIJ_NaturalOrdering	2.83e+01 4.3%	1.06e+13 3.7%	1.53e+12 10.8%	2.60e+11 5.4%	2.23e+09 0.6%
hydro_zeta_	2.65e+01 4.0%	1.33e+13 4.6%	1.35e+12 9.5%	3.88e+11 8.0%	3.46e+09 0.9%
tria_multi_	2.15e+01 3.3%	9.63e+12 3.3%	3.71e+09 0.0%	1.87e+09 0.0%	
memset@plt	1.97e+01 3.0%	1.63e+12 0.6%	7.78e+09 0.1%	5.48e+07 0.0%	
setnar_	1.72e+01 2.6%	6.22e+12 2.1%	1.73e+11 1.2%	8.96e+10 1.8%	4.47e+09 1.2%
MatMult_SeqAIJ	1.70e+01 2.6%	8.13e+12 2.8%	1.56e+12 11.0%	2.16e+11 4.5%	3.33e+09 0.9%
VecSetValues_MPI	1.46e+01 2.2%	7.71e+12 2.7%	1.62e+11 1.1%	1.23e+11 2.5%	2.49e+08 0.1%
__memset_power8	1.38e+01 2.1%	2.87e+12 1.0%	6.04e+10 0.4%	1.84e+09 0.0%	3.49e+07 0.0%
pthread_spin_lock	1.34e+01 2.0%	5.58e+12 1.9%	1.75e+08 0.0%	3.60e+08 0.0%	1.97e+07 0.0%
copy_uvz_	1.30e+01 2.0%	4.44e+12 1.5%	1.79e+09 0.0%	5.91e+09 0.1%	8.46e+07 0.0%
setdepth_	1.23e+01 1.9%	5.21e+12 1.8%	8.40e+10 0.6%	5.28e+10 1.1%	2.14e+08 0.1%
setweg_	1.18e+01 1.8%	1.21e+13 4.2%	1.14e+11 0.8%	2.97e+10 0.6%	9.16e+08 0.2%
inext_	1.11e+01 1.7%	3.27e+12 1.1%	2.65e+11 1.9%	1.03e+11 2.1%	1.59e+09 0.4%
uvtopr_	1.08e+01 1.6%	5.72e+12 2.0%	1.86e+11 1.3%	6.53e+10 1.3%	5.56e+08 0.1%
uvtop0_	8.99e+00 1.4%	4.79e+12 1.6%	1.65e+11 1.2%	4.97e+10 1.0%	1.34e+09 0.3%
momentum_advective_stability_	8.95e+00 1.4%	3.45e+12 1.2%	1.02e+10 0.1%	2.37e+09 0.0%	2.16e+08 0.1%
setnod_	8.64e+00 1.3%	2.99e+12 1.0%	2.01e+11 1.4%	2.03e+11 4.2%	2.76e+10 7.2%
PAMI_Context_trylock_advancev	8.62e+00 1.3%	4.65e+12 1.6%	1.24e+09 0.0%	2.65e+08 0.0%	3.20e+08 0.1%
PAMI_Context_lock	8.55e+00 1.3%	3.97e+12 1.4%	5.86e+08 0.0%	4.90e+08 0.0%	3.14e+07 0.0%
nknel_	6.72e+00 1.0%	1.10e+12 0.4%	7.08e+10 0.5%	3.39e+09 0.1%	4.92e+07 0.0%
bottom_friction_	6.71e+00 1.0%	3.71e+12 1.3%	7.25e+07 0.0%	1.49e+08 0.0%	
energ3d_	6.42e+00 1.0%	2.87e+12 1.0%	9.76e+10 0.7%	3.14e+10 0.6%	1.64e+09 0.4%
CCMI::Executor::ShmemBarrier::advance_impl()	5.80e+00 0.9%	2.91e+12 1.0%			1.66e+08 0.0%
start_libcoll_blocking_collective	4.79e+00 0.7%	2.34e+12 0.8%	7.93e+08 0.0%	2.69e+08 0.0%	6.59e+07 0.0%
__memcpy_power7	4.70e+00 0.7%	1.52e+12 0.5%	1.51e+10 0.1%	6.17e+09 0.1%	9.37e+08 0.2%
hydro_internal_stability	4.67e+00 0.7%	1.11e+12 0.4%			

Fig 16. HPC Toolkit profiling for the semi-explicit system using PETSc with the hydro_intern memory optimization

It can be seen as well in figure 16 that the time spent in the hydro_zeta loop is now less than half the time spent in the hydro_transports_final loop, and the L1 and LLC cache misses are as well much lower, respect to the initial profilings shown figure 6 and 7. This is due to the last level of optimization presented section 3.1.3.c) and justifies the reduction of memory stalls observed in the figure 12.

The run presented figure 16 lasted about 110.5 seconds, among which 1.5 seconds of initialization and 109 seconds for the time-iterative loop. Among the most time-expensive calls listed figure 16, the sum of the time spent in the PETSc routines is 32.4%, while 20% of the time is still spent the three most-time expensive SHYFEM calls : hydro_intern, hydro_transport_final and hydro_zeta, which would take enormous benefits from a better gestion of the memory accesses. The most critical one being now hydro_transport_final that totalizes 75% of the last level cache misses, which are the most expensive ones as they imply the access to the memory stored in RAM.

4.1.3. GPU profilings on Marconi-100

The figure 17 represents the profilings of 3 runs performed on the GPUs, the first one uses the single process-single device PETSc-GPU base on the GMRS solver with Jacobi preconditionner, will the second and third profiles correspond to the PETSc-AmgX runs performed with respectively 1 and 4 MPI processes, using one GPU device per every MPI process. While the first two cases show the same 3 first hotspot calls : MATSetValues_SeqAIJ, hydro_intern and hydro_transports_final, the PETSc-AmgX 4MPIprocess-4devices shows instead a prevalence of the time spent in some functions of the cuda library.

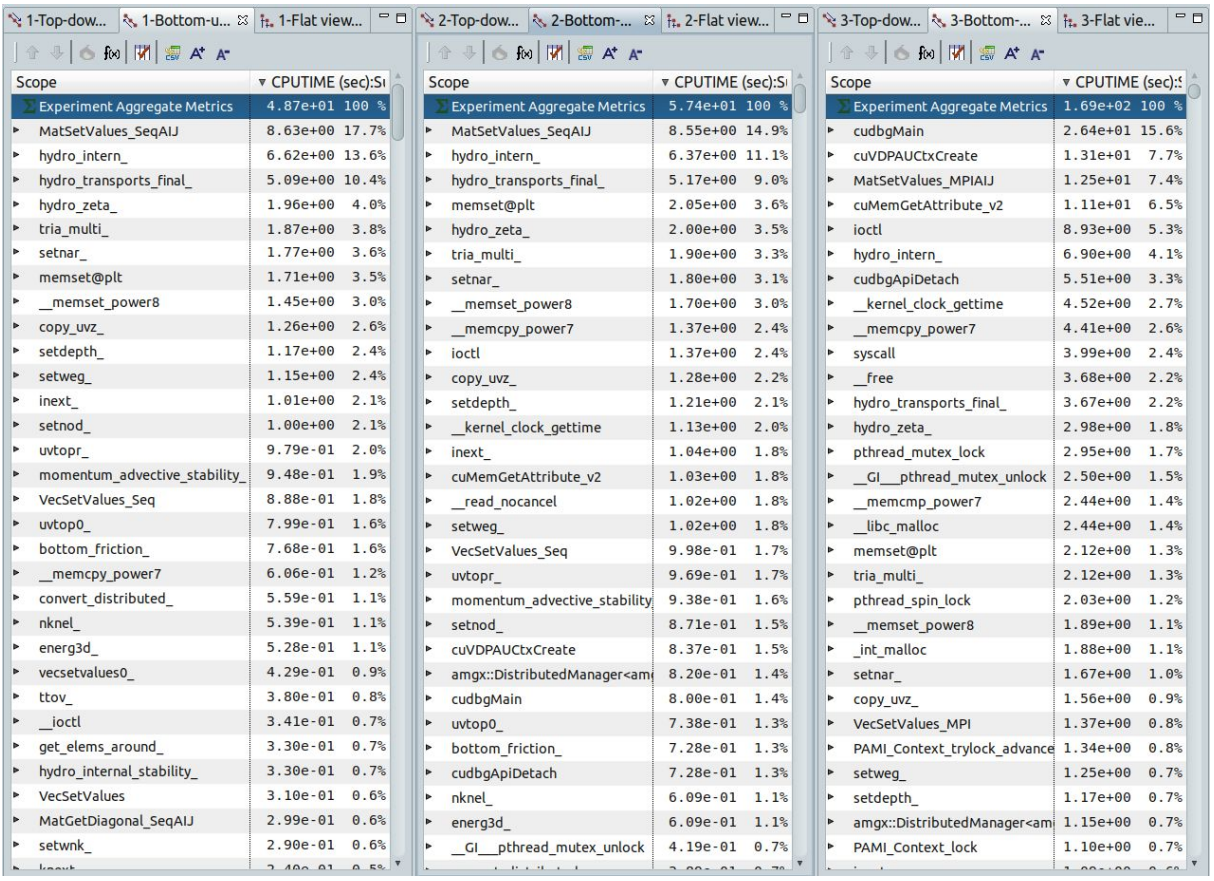


Fig 17. HPC Toolkit profiling for a) PETSc-GPU, b) PETSc-AmgX (1 core 1 device) and c) PETSc-AmgX (4 cores 4 device)

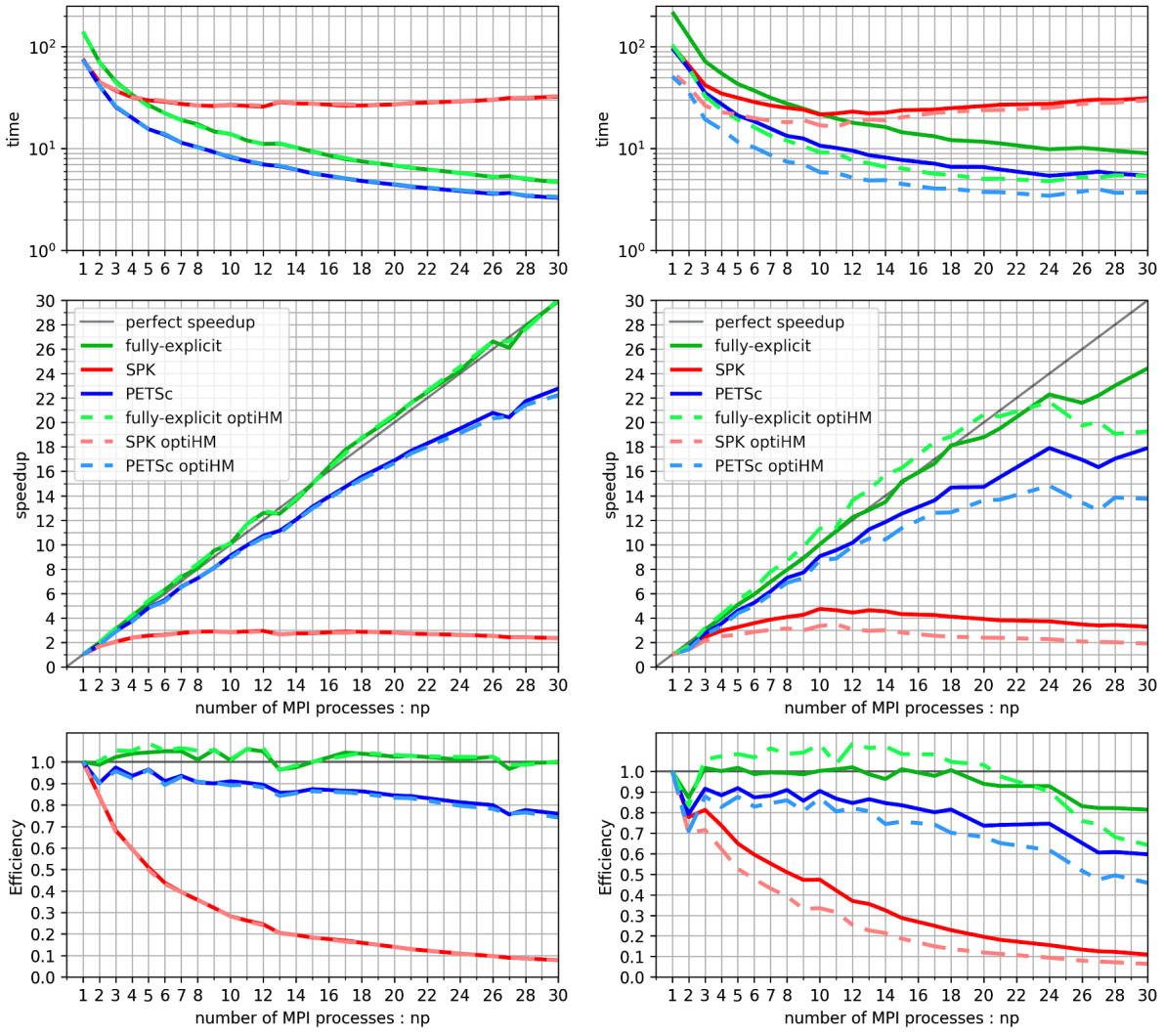
4.2. Benchmarking

The new implementations were benchmarked using the same configurations as presented in section 2.3: on both clusters SHYFEM simulations of 600 seconds were performed using a 5 seconds time-step, extracting the MPI wall time that the process 0 needs to compute the loop on the time iterations. The runs were made reserving a whole compute node and every run was repeated 6 times for every number of processes and every solver, keeping the smallest run time among the 6 identical runs. Figure 18 and 19 show once again the benchmarking results of the original explicit and implicit

sparsekit serial solvers already presented section 2, together with the benchmarks of the new implementations.

4.2.1. CPU benchmarks of PETSc solver and optimizations

The memory optimization presented in section 3.4 was applied on both clusters and to both the sparsekit and PETSc runs, they are labelled opti_HM in the legend and they are plotted by dashed orange and light blue lines for the SParseKit and PETSc simultations. This optimization did not impact the results on the Galileo cluster, but they brought a very important improvement on Marconi 100 making it much faster than the equivalent run on Galileo.



a) Galileo cluster *b) Marconi 100*

Fig 18. Time of execution, speedup and efficiency of the SHYFEM program using different solvers and optimizations on the different clusters. SPK holds for the sparsekit solver, by opposition to the PETSc solver, and the label optiMH stands for the Memory allocation optimization made in the Hydro module

On both clusters for a single MPI process (np=1) the **implicit PETSc solver** shows to be **slightly faster than the sparsekit solver** :

- on Galileo, 75 seconds using PETSc instead of 76 seconds using sparsekit

- on Marconi-100 (without the optiHM optimization), 97 seconds using PETSc against 102 seconds using sparsekit

The figure 18 shows that **using PETSc the code scales almost perfectly on both clusters up to 12 MPI processes, with an efficiency above 85%**. For a higher number of MPI processes the efficiency decreases a little, although it remains very acceptable up to 18 MPI processes and only then starts to really deteriorate. This result was expected, given that the PETSc library is optimized for very large scale applications while the **Mediterranean test case** used for this benchmark remains relatively **"light" respect to the PETSc capacities** and the use of a larger test case would probably show much better performances for more processes.

For 1 MPI process the computing time on Marconi 100 is now 50 seconds using PETSc against 55 seconds using Sparsekit, which makes the computation about 40% faster on Marconi with respect to Galileo, and can be justified by the more elevated clock speed.

It is worth noticing the decrease of efficiency appearing in correspondence of the pics of ghosts observed previously in figure 8 for 2, 4 and 27 MPI processes. For the other MPI configurations the influence of the peaks of ghosts seems to have less negative consequences. A reason for that might be that for a small number of processes the amount of nodes per process is much larger and so is the amount of work that every process must perform, which gives more available time for overlapping MPI communications.

For 3 and 5 processes the unbalance of nodes among processes represented at figure 8 is particularly low and so is the number of ghosts, compared to the other configurations. The higher optimization of the partitioning has direct effects on the efficiency which is in fact slightly better with respect to the other surrounding configurations.

After applying the optiHM optimization the fully-explicit run gains in efficiency up to arriving at values larger than 1. This is much probably an effect of the fact that SHYFEM is memory bounded, and with the increasing of the number of MPI processes the dimension of the local domains decreases, reducing by the same way the memory stalls. The fully-explicit case being almost free of MPI communications there's not enough overhead induced by the increase of the number of processes to compensate the gain in time due to the memory bound, inducing the super linear speedup that was observed.

4.2.2. PETSc and PETSc-AmgX solver on the GPU

The figure 19 shows the PETSc-CPU and GPU benchmarks obtained with the following configurations:

- the "PETSc GPU" configuration uses the PETSc GMRES solver with Jacobi preconditioner ported on the GPU, as suggested however at this time such solver allows to use only sequential-sequential matrix and vector type as it does not handle ghost cells. For this reason the benchmarked is presented only for np=1 process and 1 GPU device. The choice was made to test both the ILU and Jacobi preconditioner following the recommendations of PETSc developers [23]. The Jacobi preconditioner showed slightly better performances on our test-case.

- the “PETSc AmgX” configuration uses the implementation described section 3.2, based on AmgXWrapper, setting up the AmgX configuration with the same parameters used by [24]

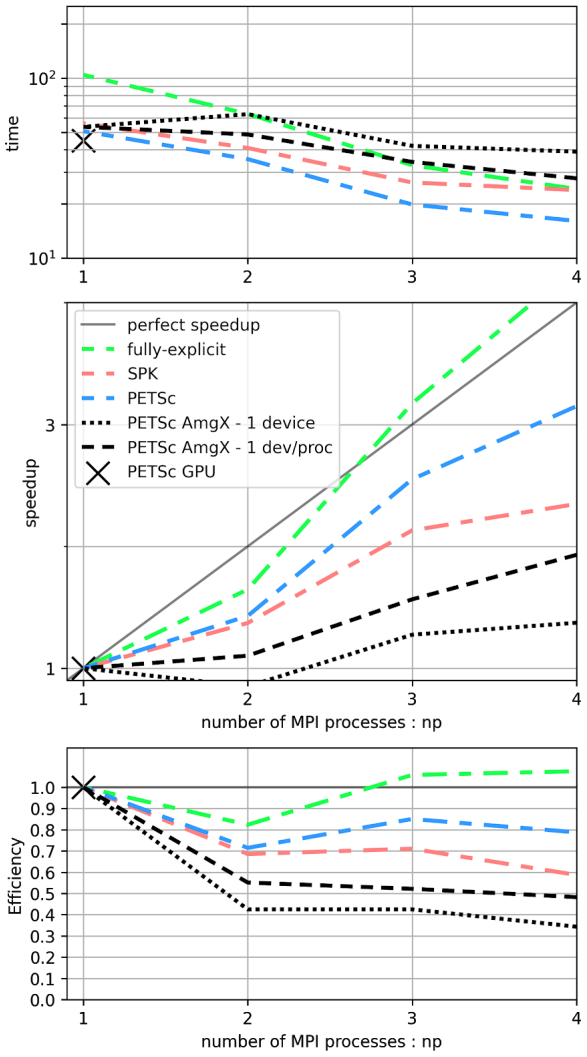


Fig 19. Time of execution, speedup and efficiency of the SHYFEM program using different solvers and optimizations on Marconi 100, all obtained after application of the optiHM optimization.

The PETSc-GPU solver is faster than all the other ones and runs in 45 seconds. For 1 MPI process (and 1 GPU) the PETSc-AmgX solver showed to be slightly slower, but still more efficient than the old Sparsekit solver, and only very slightly less efficient than PETSc-CPU.

The scaling of the PETSc-AmgX solver with the number of MPI processes is instead very bad, both using a single device, or using one device per process The main reason for this is that our test-case is really small, even for a single GPU, and the overhead caused by the communication with the GPU are too large to be worth it given the extremely small time required to solve the system, in facts the outputs of the AmgX solver indicate :

- for 1 to 4 MPI process - 1 GPU device : a setup time of about 0.04 seconds and a solve time of about 0.01 second, with an average convergence rate of 0.06 and a convergence residual passing from 3e+06 to 3e-05 in 9 iterations.

- for 4 MPI processes - 4 GPU devices : a setup time of about 0.09 seconds and a solve time of about 0.08 second, with an average convergence rate of 0.06 and a convergence residual passing from 3e+06 to 2e-05 in 9 iterations.

The reason why running multi-cores - 1 device worsens the results is, most probably because in such case more CPUs must send data to the same device and an overhead appears due to the fact that, in the actual state of implementation of the PETSc-AmgX coupling, the GPU device actually communicates with a single CPU that must gather and scatter the data with the other CPUs.

These outputs show that for the test-case considered in this study the GPUs are really under-utilized, and larger systems would take better advantage of them. Some future developments will for sure allow to improve the performance of the PETSc-AmgX coupled librairies with multi-CPU - 1 device and will allow a better scaling for a test case of the dimension of the one that we employed.

4.3. Semi-implicit runs using a larger timestep

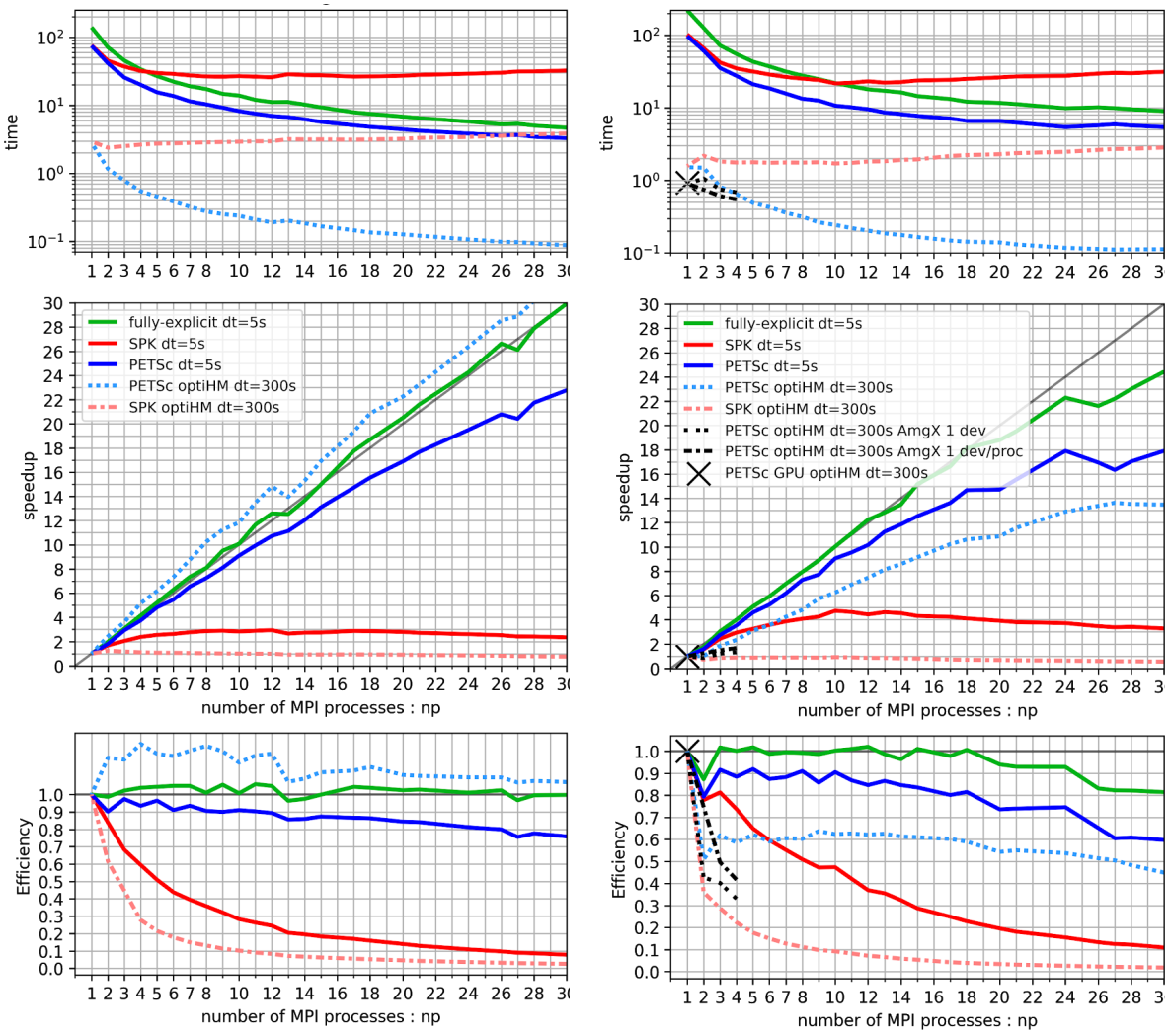
To avoid inserting bias in the comparison of the results, in all the studies presented above, the simulations that were presented used a 5 seconds timestep, which was the default value coming with the shyfem example-configuration file. Time-steps of this order of dimensions are imposed by the Courant–Friedrichs–Lewy condition, which is necessary for the convergence of explicit time integration schemes. It states that the distance that any information travels during a time-step of the simulation must not exceed the size of the mesh elements, that is in 1D: (U Dt /Dx) < 1.

Considering that the average current speed is only of the order of 1m/s while the gravity wave speed reaches about sqrt(g*depth)=sqrt(10*4000)=200m/s in the Mediterranean Sea ; it is the largest one of them that determines the time-step Dt required to satisfy the CFL condition. For the test case considered in this study it is the speed of the gravity wave that determines the CFL condition; in order to respect it given the dimension of the mesh cells it was estimated that a time-step below 2.82 seconds must be employed for the fully-explicit run.

Using the semi-implicit model we can get free of the CFL condition and we can then use a much larger timestep, that becomes limited only by the physics we want to solve. Per example, it should remain below 1hour to simulate tidal variations. So, in practice the semi-implicit runs do not need to respect the CFL condition, and thanks to that we can use timesteps hundreds of times larger than the fully-explicit runs, allowing us to perform simulations that are hundreds of times faster. A timestep of 300 seconds is usually employed by the users of the SHYFEM model to solve the hydrodynamics equations. When such a larger time-step can be obtained by getting free of the CFL constraint, the fully-explicit configuration is generally not an option anymore. We analyzed it in the previous sections in order to evaluate the scaling of the run solving the fully explicit system to ensure that the non-solver portions of the code were scaling well, which was confirmed as we could reach perfect speedup on galileo and even super linear speedup on Marconi 100.

To evaluate the effective gain allowed by the new implementations, another benchmark was performed solving the semi-implicit system with either PETSc or Sparsekit, with a timestep of 300 seconds on the Mediterranean domain for a 1 day long simulation. To make them comparable with the 600s long results of the runs performed in the previous figures, the time to solution was divided

by 86400/600=144. In this way, the time to solution presented figure 20.a) all correspond to a 600s day long simulation. Passing from a 5 seconds timestep to a 300 seconds timestep the number of solver-iterations performed by the PETSc to converge to a solution passed from 8 to 30, resulting in an increase of the time required to perform the single time-iteration.



a) Galileo cluster *b) Marconi 100 cluster*

Fig. 20. Benchmarking the semi-implicit runs using a 300s timestep

The figure 20 highlights the great improvement of performance brought to the SHYFEM model through the coupling with the PETSc library. In fact the dashed blue curve representing the PETSc run with a 300s timestep demonstrates time to solution at least 50 times smaller than the ones of the explicit run, and a perfect strong scaling with the number of MPI processes on Galileo with super linear speedup. On Marconi 100 improvements of the same order can be observed with respect to the fully-explicit run, even though the scaling is worse with respect to Galileo with an efficiency limited to 60% from 3 to 18 MPI processes. To help interpret the overhead observed on Marconi 100 a profiling of the PETSc - opti_HM run was performed on both Galileo and Marconi 100, the results are presented in figure 21. a) and b) respectively.

Function / Call Stack	CPU Time ▼				Module
	Effective Time by Utilization	Spin Time		Over…	
	Id Po O Ide Ov	MPI B…	Other	Other	
MatSolve_SeqAIJ_NaturalOrdering	90.907s	0s	0s	0s	libpetsc.so.3…
MatMult_SeqAIJ	71.900s	0s	0s	0s	libpetsc.so.3…
MatSetValues_MPIAIJ	37.313s	0s	0s	0s	libpetsc.so.3…
hydro_intern	33.355s	0s	0s	0s	shyfem
VecMAXPY_Seq	31.858s	0s	0s	0s	libpetsc.so.3…
VecMDot_Seq	28.164s	0s	0s	0s	libpetsc.so.3…
PMPI_Allreduce	13.468s	0.620s	1.590s	0s	libmpi.so.12
hydro_zeta	11.987s	0s	0s	0s	shyfem
write	10.628s	0s	0s	0s	libpthread.so.0
hydro_transports_final	10.013s	0s	0s	0s	shyfem
MatLUFactorNumeric_SeqAIJ	9.934s	0s	0s	0s	libpetsc.so.3…

a) Galileo cluster

Top-down view | Bottom-up view | Flat view

Scope	▼ CPUTIME (sec):Su	L1-DCACHE-LOADS:	L1-DCACHE-LOAD-MISSES:	LLC-LOADS:Sum (E)	LLC-LOAD-MISSES
MatSolve_SeqAIJ_NaturalOrdering	1.98e+02 16.6%	3.54e+13 10.8%	6.32e+12 24.7%	2.02e+12 22.4%	1.46e+11 27.5%
MatMult_SeqAIJ	1.57e+02 13.1%	3.19e+13 9.7%	7.08e+12 27.7%	2.83e+12 31.3%	9.56e+10 18.0%
VecMDot_Seq	8.31e+01 7.0%	2.57e+13 7.8%	2.32e+12 9.1%	9.76e+11 10.8%	7.49e+10 14.1%
VecMAXPY_Seq	7.96e+01 6.7%	1.35e+13 4.1%	1.53e+12 6.0%	1.04e+12 11.6%	5.43e+10 10.2%
MatSetValues_MPIAIJ	7.40e+01 6.2%	1.37e+13 4.2%	1.16e+12 4.5%	2.84e+11 3.1%	3.93e+10 7.4%
pthread_spin_lock	5.89e+01 4.9%	2.38e+13 7.2%	6.63e+09 0.0%	8.92e+08 0.0%	8.54e+07 0.0%
hydro_intern_	5.47e+01 4.6%	1.09e+13 3.3%	1.88e+12 7.3%	4.00e+11 4.4%	1.10e+09 0.2%

b) Marconi 100 cluster

Fig. 21. Profiling the semi-implicit runs using a 300s timestep

At first sight no macroscopical difference appears, but looking at the HPCToolkit analysis it appears that the LLC cache misses are very high in the top time-demanding functions of the PETSc solver. Additional informations on the compute nodes hardware were extracted and presented in table 1.

Galileo $ lscpu

Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
CPU(s):	36
On-line CPU(s) list:	0-35
Thread(s) per core:	1
Core(s) per socket:	18
Socket(s):	2
NUMA node(s):	2
Vendor ID:	GenuineIntel
CPU family:	6
Model:	79
Model name:	Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30GHz
Stepping:	1
CPU MHz:	1258.441
CPU max MHz:	3600,0000
CPU min MHz:	1200,0000
BogoMIPS:	4599.76
Virtualization:	VT-x
L1d cache:	32K
L1i cache:	32K
L2 cache:	256K
L3 cache:	**46080K**
NUMA node0 CPU(s):	0-17
NUMA node1 CPU(s):	18-35

a) Galileo cluster

M100 $ lscpu

Architecture:	ppc64le
Byte Order:	Little Endian
CPU(s):	128
On-line CPU(s) list:	0-127
Thread(s) per core:	4
Core(s) per socket:	16
Socket(s):	2
NUMA node(s):	6
Model:	2.1 (pvr 004e 1201)
Model name:	POWER9, altivec supported
CPU max MHz:	3800,0000
CPU min MHz:	2300,0000
L1d cache:	32K
L1i cache:	32K
L2 cache:	512K
L3 cache:	**10240K**
NUMA node0 CPU(s):	0-63
NUMA node8 CPU(s):	64-127

b) Marconi 100 cluster

Table 1. Hardware characteristics obtained with "lscpu"

It appears that Marconi 100 has 4 times less L3 memory than Galileo, which deteriorates the performances due to an increase of DRAM accesses when L3 cache misses occur. To test this hypothesis the L3 cache misses and memory bandwidth on both clusters must be compared.

The Drivers installed on the Galileo cluster wouldn't allow the collect and counting of the memory-access using VTUNE, so that we used **perf** to evaluate if the difference in L3 cache size could be the origin of the overhead observed on M100. The results are presented in table 2.

Before analysing the perf profiling, we must precise that the predominant hydro function appearing only in the Galileo profiling is in reality the sum of the hydro_transports_final and hydro_zeta functions called by hydro. The perf tool fails to distinguish and sums them into hydro ; we are sure of that because we experienced that the same "sum" occurs running intel Vtune with a program compiled without the -g debug flag.

450881,838733 task-clock (msec) 5,818 CPUs utilized 1033190915365 cycles 2,291 GHz 1850075850908 instructions 1,79 insn per cycle 65998102592 cache-references 146,37 M/sec 4430919310 cache-misses 6,71 % of cache refs **18507074495 LLC-loads 41,04 M/sec** **909285126 LLC-load-misses 4,9% of LL-cache hits**	805564,927248 task-clock (msec) 5,951 CPUs utilized 2834821973160 cycles 3,519 GHz 3061085517388 instructions 1,08 insn per cycle 822103922932 cache-references 1020,531 M/sec 68871561557 cache-misses 8,37 % of cache refs **24497497913 LLC-loads 30,410 M/sec** **1378384649 LLC-load-misses 5,6% of LL-cache hits**
# Samples: 1M of event 'LLC-loads' # Event count (approx.): **19'325'406'255** 35.34% libpetsc MatMult_SeqAIJ 17.43% libpetsc MatSolve_SeqAIJ_NaturalOrder 8.72% shyfem hydro_ 7.18% shyfem ddot_ 6.02% libpetsc VecMAXPY_Seq 4.13% libpetsc MatSetValues_MPIAIJ 3.42% libpetsc MatLUFactorNumeric_SeqAIJ 3.15% libpetsc VecMDot_Seq 3.12% shyfem hydro_intern_ 1.98% shyfem setnod_	# Samples: 2M of event 'LLC-loads:u' # Event count (approx.): **25'996'876'456** 41.50% libpetsc MatMult_SeqAIJ 30.33% libpetsc MatSolve_SeqAIJ_NaturalOrder 3.53% libpetsc VecMAXPY_Seq 3.52% libpetsc VecMDot_Seq 3.35% shyfem hydro_transports_final_ 2.65% libpetsc MatSetValues_MPIAIJ 2.62% shyfem hydro_intern_ 2.29% libpetsc MatLUFactorNumeric_SeqAIJ 1.36% shyfem hydro_zeta_
# Samples: 693K of event 'LLC-load-misses' # Event count (approx.): 947'507'948 42.52% shyfem hydro_ 13.15% shyfem hydro_intern_ 9.05% shyfem setnod_ 8.09% libpetsc MatMult_SeqAIJ 5.02% libpetsc MatSetValues_MPIAIJ 2.50% libpetsc VecMDot_Seq 2.17% shyfem meteo_force_ 1.98% libpetsc MatLUFactorNumeric_SeqAIJ 1.97% libpetsc MatSolve_SeqAIJ_NaturalOrder 1.51% shyfem daxpy_	# Samples: 2M of event 'LLC-load-misses:u' # Event count (approx.): 1'472'321'472 34.99% shyfem hydro_transports_final_ 22.57% libpetsc MatSolve_SeqAIJ_NaturalOrder 16.25% libpetsc MatMult_SeqAIJ 6.50% libpetsc MatSetValues_MPIAIJ 2.64% libpetsc VecMDot_Seq 2.33% libpetsc VecMAXPY_Seq 1.46% libpetsc MatLUFactorNumeric_SeqAIJ 1.27% shyfem hydro_intern_ 0.84% shyfem hydro_zeta_ 0.84% libpetsc MatMultAdd_SeqAIJ

a) Galileo cluster *b) Marconi 100 cluster*

Table 2. Software performance obtained using the perf tool collecting stat and record.

We can observe looking at the table 2 at the "Event Count" that the run performed on Marconi 100 has 25% more LLC loads and 50% more LLC load-misses than the run performed on Galileo. In particular, the hydro* routines and the PETSc solver routines `MatMult_SeqAIJ` and `MatSolve_SeqAIJ_NaturalOrder` demonstrate the highest LLC loads and load-misses and must have altered performances on Marconi 100 due to the smallest L3 cache size with respect to Galileo. However, it is not possible without deeper insights to know if these more frequent cache misses can or not be compensated by a faster memory bandwidth of Marconi 100, nor what is the weight of time spent in the MPI communications with respect to the time lost in cache misses. This analysis is still incomplete and would deserve further investigations.

4.4. Efficiency of the meson build system

On Marconi 100, the "make fem" command compiles using the SHYFEM original build system based on hand-written Makefiles; it takes about 150 seconds to compile the whole code and in the actual state appears broken when trying to run it in parallel.

Using the meson build system the addition of new source files becomes much more simple because their name should simply be appended to the list of source files and meson will evaluate automatically the tree of dependencies. The addition of libraries is as well made simpler as it is enough to have them referenced in the PKG_CONFIG_PATH environmental variable, or alternatively in the LIBRARY_PATH environmental variable when no headers must be included in the source code. It is still possible to indicate the installation directory path and program meson to search for the library and headers in the subdirectories. Meson then handles automatically the linking and including. Meson offers options to automatically add debugging of optimized flags, but any other flag can be added manually. Meson handles as well automatically the detection of the source code language and multi-languages compilation.

Using our meson setup the precompilation checks shown figure 22 last between 6 to 9 seconds depending on the options that were selected, they assess the correct linking of the needed libraries and prepare the compilation setup ; then the actual compilation illustrated figure figure 23 using command "ninja" takes about 20 to 30 seconds, compiling automatically in parallel. The integration tests represented figure 24 takes from 3 to 8 seconds, depending on the eventual request to test PETSc GPU and PETSc AmgX.An example of the outputs of the 3 command line instructions is given below.

```
shyfem $ time meson . build_R_PETSc_AmgX_ztest --buildtype release -D use_PETSc=true  -D
use_AmgX=true  -D amgxwrapper_dir=$AMGXWRAPPER_HOME -D amgx_dir=$AMGX_HOME -D
cuda_dir=$CUDA_HOME  -D test_zeta=true -D num_GPU=2
The Meson build system
Version: 0.55.3
Source dir: /m100_scratch/userexternal/clauren1/shyfem
Build dir: /m100_scratch/userexternal/clauren1/shyfem/build_R_PETSc_AmgX_ztest
Build type: native build
Using 'PKG_CONFIG_PATH' from environment with value:
':/m100_scratch/userinternal/mvalent3/modules/opt/libraries/petsc-system-blas-hypre-viennacl-ml-info-fortran/3.13.4/spectrum_mpi--10.3.
1--binary/lib/pkgconfig/'
Project name: shyfem
Project version: undefined
Fortran compiler for the host machine: gfortran (gcc 8.4.0 "GNU Fortran (GCC) 8.4.0")
Fortran linker for the host machine: gfortran ld.bfd 2.27-34
C compiler for the host machine: gcc (gcc 8.4.0 "gcc (GCC) 8.4.0")
C linker for the host machine: gcc ld.bfd 2.27-34
C++ compiler for the host machine: g++ (gcc 8.4.0 "g++ (GCC) 8.4.0")
C++ linker for the host machine: g++ ld.bfd 2.27-34
Host machine cpu family: ppc64
Host machine cpu: ppc64le
Message: ------------------- preparing global settings --------------------
../meson.build:94: WARNING: test_zeta option is activated, it creates an output file of the solution vector which slows down shyfem
Configuring pragma_directives.h using configuration
Message: Compiling with mpi dependency
Dependency mpi found: YES 10.03.01 (cached)
Checking if "can include mpi" compiles: YES (cached)
Message: ------------------- preparing shyfem build target ----------------
Message: Selecting the PETSc solver
Dependency PETSc found: YES 3.13.99 (cached)
Checking if "can include PETSc" with dependencies mpifort, PETSc compiles: YES (cached)
Message: Using user-provided amgxsh directory
/m100_scratch/userinternal/mvalent3/modules/opt/libraries/AMGx/v2.1.0/spectrum_mpi--10.3.1--binary
```

```
Library amgxsh found: YES
Checking if "can include amgxsh" with dependencies mpifort, PETSc, -lamgxsh compiles: YES (cached)
Message: Using user-provided AmgXWrapper directory
/m100_scratch/userinternal/mvalent3/modules/opt/libraries/AMGxWrapper-01/v1.5/spectrum_mpi--10.3.1--binary
Library AmgXWrapper found: YES
Checking if "can include AmgXWrapper" with dependencies mpifort, PETSc, -lamgxsh, -lAmgXWrapper compiles: YES (cached)
Library cublas found: YES
Library cudart found: YES
Library cusparse found: YES
Library cusolver found: YES
Message: - - - - - [ preparing dependency libgotm build target ] - - - - -
Message: - - - - - [ meson configuring running tests ] - - - - - - - - - -
Program run_and_check_success.sh found: YES
Program create_basin.sh found: YES
Program synch.sh found: YES
Message: only one thread can be used with PETSc_GPU
Message: ---------- preparing shyparts build target ----------
Found pkg-config: /usr/bin/pkg-config (0.27.1)
Did not find CMake 'cmake'
Found CMake: NO
Run-time dependency metis found: NO (tried pkgconfig and cmake)
Message: metis dependency couldnt be found using pkg-config, now trying with find_library as we only need the library path,  if it does not work please provide installation path in var "_metis_dir" or add metis library path to environmental variable "LIBRARY_PATH"
Message: metis dependency couldnt be found using pkg-config, now trying with find_library as we only need the library path,  if it does not work please provide installation path in var "_metis_dir" or add metis library path to environmental variable "LIBRARY_PATH"
Library metis found: YES
Checking if "can include metis" compiles: YES (cached)
Message: ---------- preparing shypart build target ----------
Message: ---------- preparing shybas build target ----------
Message: ---------- preparing shypre build target ----------
Build targets in project: 6
real    0m7.637s
user    0m5.710s
sys     0m2.154s
```

Fig 22. Pre-compilation meson setup

```
shyfem/build_dir $ time ninja
[598/598] Linking target shyfem
real    0m20.134s
user    5m10.552s
sys     1m9.067s
```

Fig 23. Ninja compilation based on the setup built by meson

```
shyfem/build_dir $ time ninja test
[60/61] Running all tests.
 1/14 create .bas file for 0001 mpi procs                OK          0.07s
 2/14 create .bas file for 0002 mpi procs                OK          0.13s
 3/14 ---------- barrier ----------                      OK          0.01s
 4/14 run shyfem zeta tests for EXPLICIT solver using 0001 mpi procs   OK          0.72s
 5/14 run shyfem zeta tests for EXPLICIT solver using 0002 mpi procs   OK          0.72s
 6/14 ---------- barrier ----------                      OK          0.01s
 7/14 run shyfem zeta tests for PETSc_CPU solver using 0001 mpi procs  OK                0.72s
 8/14 run shyfem zeta tests for PETSc_CPU solver using 0002 mpi procs  OK                0.72s
 9/14 ---------- barrier ----------                      OK          0.01s
10/14 run shyfem zeta tests for PETSc_GPU solver using 0001 mpi procs  OK                1.77s
11/14 ---------- barrier ----------                      OK          0.01s
12/14 run shyfem zeta tests for PETSc_AMGX solver using 0001 mpi procs OK                3.43s
13/14 run shyfem zeta tests for PETSc_AMGX solver using 0002 mpi procs OK                3.53s
14/14 ---------- barrier ----------                      OK          0.02s

Ok:                 14
Expected Fail:      0
Fail:               0
Unexpected Pass:    0
Skipped:            0
```

```
Timeout:                    0
Full log written to /m100_scratch/userexternal/clauren1/shyfem/build_R_PETSc_AmgX_ztest/meson-logs/testlog.txt

real    0m7.769s
user    0m11.253s
sys     0m4.510s
```

Fig 24. Integration tests using meson

Conclusions & Perspectives

The PETSc library is now part of the SHYFEM Shallow water HYdrodynamic Finite Element Model and is used to solve the implicit system of equations for the water level ζ. The fortran modules calling the PETSc API were implemented using object-oriented programming to offer the possibility to use the solvers of PETSc for any sparse linear equation system that must be solved by the model. Best programming practices were employed and some optimizations of the source code were proposed.

The original Makefile-based compilation system was modified to be able to enable the compilation and library linking of the source code calling PETSc and leaves the possibility to use the previous Sparsekit solver.

A setup for the Meson build system was implemented, it allows to compile SHYFEM and the main tools of the SHYFEM suite with Sparsekit, PETSc, AmgX, MPI and CUDA, and it can run automatic CPU and GPU tests to insure the successful ending of the SHYFEM execution.

Benchmarks were performed on the CINECA Galileo and Marconi 100 cluster and showed when using a timestep constrained by the CFL condition (5 seconds) a very good scaling of the model up to 12 MPI processes, with an efficiency above 85% on Galileo and above 80% on Marconi 100. Obtaining such results for a relatively small global problem is already very satisfying, as in such a configuration with 12 processes the dimension of the matrix stored by every process is only 18,000, while the PETSc documentation [25] indicates an absolute minimum of about 10,000 unknowns per process, and recommend 20,000 or more to observe a satisfying speedup.

Solving the semi-implicit system of equations allows us to get free of the constraint on the timestep imposed by the CFL condition, and the choice of the timestep will then only depend on the choice of the physical phenomena that we want to represent. A timestep of 300 seconds is usually employed by SHYFEM users, and such a configuration allows to take the best advantage of the new implementations, with a time to solution reduced at least by a factor 50.

To resume the effects of the implementations for the Mediterranean test case considered in this work (about 400 000 elements and 215 000 nodes) we present in table 3 the time to solution required to run a one year long simulation with the different solvers :

MPI processes	**Solving approach**	**Galileo**		**Marconi 100**	
		dt = 5 s	**dt = 300 s**	**dt = 5 s**	**dt = 300 s**
1	**Fully-explicit**	90 days	not possible	60 days	not possible
	Sparsekit	45 days	>25 days	30 days	>15 days
	PETSc	<45 days	<25 days	<30 days	<15 days
12	**Fully-explicit**	7 days	not possible	4.5 days	not possible
	Sparsekit	11 days	1.2 days	16 days	1.1 days
	PETSc	4.25 days	3 hours	3 days	3 hours

Table 3. Time to solution of a one year long simulation with the different solver configurations

With the new implementations and the PETSc library, using 12 MPI processes and a timestep of 5 seconds the hydrodynamic model is now able to run a one year long simulation in 3 days on Marconi-100 and 4.25 days on Galileo. Instead, using a 300 seconds timestep a 1 year long simulation would now last only 3 hours with the newly implemented PETSc solver.

Using a 5 seconds timestep about one third of the computational time can be assigned to the PETSc APIs, while the other two thirds of the time are spent in the SHYFEM code. Using a 300 seconds timestep, more than 50% of the time is spent in the PETSc library.

SHYFEM was run on Marconi 100 using the GPU ported PETSc solver and showed to be slightly faster than the CPU version when only 1 CPU and 1 GPU were employed.

During the GPU hackathon of Helmoltz, Sept.2020 the AmgXWrapper was used to interface PETSc with the nVIDIA AmgX solver to provide another option for GPU computations. SHYFEM was run on Marconi 100 using PETSc with the AmgX solver running on the GPU and showed very slightly worse performance than the CPU version when only 1 CPU and 1 GPU were employed and a deterioration of the performance for multi-CPUs and GPUs. But this might be solved by future new versions of the libraries that are under constant development.

In the meanwhile, larger systems would take better advantage of the GPUs, especially because the test case that was run was only 2D and passing to the 3D SHYFEM model could change everything.

The goal of this thesis work was not to test the performance of the different solvers and configurations provided by PETSc and AmgX. To ensure a robust and stable configuration the default solvers and parameters suggested by the libraries developers were employed and it is much probable that better performances can be reached looking for the adequate solver and parameterization for the physical problem that must solve SHYFEM.

The profilings performed on both clusters using Intel tools and HPCtoolkit evidentiated that SHYFEM is memory bounded, the origin of this bound was evidentiated in the elevated cache loads & misses happening inside the loops over the elements and in particular hydro_intern, hydro_transport_final and hydro_zeta, which would take enormous benefits from a better management of the memory accesses.

The analysis of the METIS partitioning allowed to evidentiate that the code performance would benefit from the implementation in SHYFEM of the possibility to use non-contiguous hydrodynamic domains. Such a possibility would allow a large reduction of the number of ghosts, and by consequence, of the MPI communications, as well as a reduction of the imbalance that would reduce the time spent in idle state, when a processor waits for the others to finish their work.

Another point of possible optimization regards the ordering of the nodes of the hydrodynamic model. In the actual state of development there is no re-ordering happening to decrease the matrix bandwidth; such an improvement done as a pre-processing would make the solver faster.

The works presented in this thesis contribute to the modernisation of SHYFEM and conclude the MPI parallelization implemented by ISMAR/CNR. They open the door to many possibilities, starting from the possibility to run more accurate simulations by increasing the size of the problem, obtaining faster solutions and finally giving access to the opportunities offered by the new accelerated architectures.

References

[01] PETSc Portable, Extensible Toolkit for Scientific Computation Toolkit for Advanced Optimization : https://www.mcs.anl.gov/petsc/

[02] SHYFEM, Finite Element Model for Coastal Seas, User Manual : https://github.com/shyfem-cm/shyfem/blob/develop/femdoc/final/shyfem.pdf

[03] MPI : the Message Passing Interface library http://www.mpi-forum.org

[04] AmgX solver : Multi-Grid Accelerated Linear Solvers for Industrial Applications, https://github.com/NVIDIA/AMGX

[05] AmgXWrapper: An interface between PETSc and the NVIDIA AmgX library, P-Y Chuang and L. A. Barba, 2017, Journal of Open Source Software 2(16)DOI: 10.21105/joss.00280 https://github.com/barbagroup/AmgXWrapper

[06] The Meson build system https://mesonbuild.com/

[07] Sparsekit https://people.sc.fsu.edu/~jburkardt/f77_src/sparsekit/sparsekit.html)

[08] The Galileo supercomputer- CINECA https://www.hpc.cineca.it/hardware/galileo

[09] The Marconi 100 accelerated system - CINECA https://www.hpc.cineca.it/hardware/marconi100

[10] Intel Application Performance Snapshot https://software.intel.com/content/www/us/en/develop/documentation/application-snapshot-user-guide/top/introducing-application-performance-snapshot.html

[11] Intel VTune Profiler https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/vtune-profiler.html

[12] HPCToolkit: Tools for performance analysis of optimized parallel programs, L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent. Concurrency and Computation: Practice and Experience, 22(6):685–701, 2010. (doi:10.1002/cpe.1553) ; http://hpctoolkit.org/

[13] The Spack package manager tool, https://spack.readthedocs.io/en/latest/

[14] METIS - Serial Graph Partitioning and Fill-reducing Matrix Ordering, http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

[15] The Taurus Cluster - TU Dresden https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/HardwareTaurus

[16] GDB, The GNU Project Debugger, https://www.gnu.org/software/gdb/

[17] Intel Inspector, https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/inspector.html

[18] PETSc - KSP linear solver documentation : https://docs.petsc.org/en/latest/manual/ksp/

[19] GMRES : A generalized minimal residual algorithm for solving nonsymmetric linear systems, S. Yousef and M. H. Schultz, *SIAM J. Sci. Stat. Comput.*, July 1986, Vol. 7, No. 3, pp. 856-869, DOI:10.1137/0907058

[20] CUDA parallel computing platform and programming model, https://developer.nvidia.com/cuda-zone

[21] meson repository https://github.com/mesonbuild/

[22] Ninja, a small build system with a focus on speed. https://ninja-build.org/

[23] Progress with PETSc on Manycore and GPU-based Systems on the Path to Exascale, R. T. Mills et al., PETSc 2019 User Meeting, Atlanta, GA, USA June 6, 2019 https://www.mcs.anl.gov/petsc/meetings/2019/slides/mills-petsc-2019.pdf

[24] AmgXGPU Solver Developments for OpenFOAM, M. Martineau, S. Posey, F. Spiga, 8th OpenFOAM Conference, October 13 - 15, 2020 https://www.esi-group.com/sites/default/files/resource/other/1672/8th_OpenFOAM_Conference_NVIDIA_Posey.pdf

[25] https://www.mcs.anl.gov/petsc/documentation/faq.html#slowerparallel