ExaQUte

Exascale **Q**uantification of **U**ncertainties for **Te**chnology and Science Simulation

D4.5 Framework development and release

Document information table

Contract number:	800898
Project acronym:	ExaQUte
Project Coordinator:	CIMNE
Document Responsible Partner:	BSC
Deliverable Type:	Other
Dissemination Level:	Public
Related WP & Task:	WP 4, Task 4.6
Status:	Draft

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No **800898**

Authoring

Prepared by:				
Authors	Partner	Modified Page/Sections	Version	Comments
Jorge Ejarque	BSC	Sections 1, 2, 3, 4 and A	V0.2, V0.3	-
Riccardo Tosi	CIMNE	Sections 2 and 3	V0.2	
Marc Núñez	CIMNE	Sections 2 and 3	V0.2	
Stanislav Böhm	IT4I	Sections 2 and 3	V0.3	
Rosa M. Badia	BSC	ToC and Internal Review	V0.1,V0.4	

Change Log

Versions	Modified Page/Sections	Comments
V0.1	First document version with ToC	
V0.2	Sections 1, 2, 3.1, 3.2.1, 3.3 and 3.4	First contributions for the different sections
V0.3	Section 2.6.2, 3.2.2, 3.5, 4 and A	Complete contributions
V0.4	Minor changes in all sections	Internal review
V1.0	Final Version	

Approval

Approved by:				
	Name	Partner	Date	OK
Task leader	Jorge Ejarque	BSC	19/11/2021	OK
WP leader	Rosa M. Badia	BSC	22/11/2021	OK
Coordinator	Riccardo Rossi	CIMNE	23/11/2021	OK

Executive summary

This deliverable presents the final release of the ExaQUte framework as result of task 4.6 of the project focused on the framework development and optimization. The first part of the document presents an overview of the different parts of the ExaQUte framework providing the links to the repositories where the code of the different components can be found as well as the installation and usage guidelines. These repositories will include the final version of the ExaQUte API and its implementation for the runtimes provided in the project (PyCOMPSs/COMPSs and Quake).

The second part of the document presents a performance analysis of the framework by performing strong and weak scaling experiments. In this case, we have focused on the analysis of the new features introduced during the last part of the project to support and optimize the execution of MPI solvers inside the framework. The support for OpenMP was already reported in Deliverable D4.3 [21]. The results of the experiments demonstrate that the proposed framework allow to reach very good scalability for the analysed Monte Carlo problems.

Table of contents

1 Introduction **7**

2 ExaQUte framework **7**
2.1 Solvers . . . 7
2.2 Kratos Multipysics . . . 7
2.3 ParMmg . . . 8
2.4 XMC . . . 8
2.5 ExaQUte API . . . 9
2.6 Supported Runtimes . . . 9
2.6.1 PyCOMPSs/COMPSs . . . 9
2.6.2 Quake . . . 10

3 Evaluation **10**
3.1 MPI Benchmarks . . . 10
3.1.1 High rise building benchmark . . . 12
3.1.2 Rectangle obstacle benchmark . . . 13
3.2 Testbeds . . . 13
3.2.1 MareNostrum Supercomputer . . . 13
3.2.2 Barbora Supercomputer . . . 14
3.3 Experiments description . . . 15
3.4 COMPSs Results . . . 15
3.5 Quake Results . . . 17

4 Conclusions **18**

A Appendix: Final specification of ExaQUte API **20**
A.1 Decorators . . . 20
A.1.1 Task decorator . . . 20
A.1.2 Constraint decorator . . . 21
A.1.3 MPI tasks decorator . . . 21
A.2 API calls . . . 23

List of Figures

1 ExaQUte Framework overview. 8

2 CAARC problem domain. $H = 180\,\text{m}$, $W = 45\,\text{m}$ and $L = 30\,\text{m}$. 12

3 Snapshot of the velocity field. 12

4 Scheme of the computational domain used for the rectangle problem, where $D = 1\,\text{m}$, $B = 5D$, $L_x = 55B$, $L_y = 30B$ and $\Lambda_x = 15B$. Thus, the dimensions of the outer domain are $275 \times 150\,\text{m}$, and the inner rectangle has size $5 \times 1\,\text{m}$. 13

5 Velocity field snapshot. 14

6 Strong scaling results for COMPSs runtime in MareNostrum supercomputer. The baseline for calculating the scalability is the execution time obtained for 16 nodes. 16

7 Comparison of two execution traces of the strong scaling experiments running the MPI Benchmark with the COMPSs runtime in MareNostrum supercomputer with 32 and 64 nodes. The two traces are at the same time scale, showing almost perfect scaling. 16

8 Weak scaling results for COMPSs runtime in MareNostrum supercomputer. The baseline for calculating the efficiency is the execution time obtained for 16 nodes. 17

9 Comparison of two execution traces of the weak scaling experiments running the MPI Benchmark with the COMPSs runtime in MareNostrum supercomputer with 32 and 64 nodes. 17

10 Two-dimensional flow benchmark, executed on Barbora with Quake backend; strong scalability 18

11 Two-dimensional flow benchmark, executed on Barbora with Quake backend; weak scalability 18

12 Task basic usage. Parameters a and b are assumed as input objects and c is an object modified by the task 20

13 Tasks Constraint usage 21

14 Application example with collections and MPI tasks 22

15 Example application execution graph. 23

Nomenclature / Acronym list

API	Application Programming Interface
CAARC	Commonwealth Advisory Aeronautical Council
DAG	Directed Acyclic Graph
ExaQUte	EXAscale Quantification of Uncertainties for Technology and Science Simulation
GPFS	General Parallel File System
HPC	High Performance Computing
IN	Parameter of a function that is not modified
INOUT	Parameter of a function that is modified during the call
Kratos	Kratos Multiphysics
MC	Monte Carlo
MLMC	Multi-Level Monte Carlo
MPI	Message Passing Interface
OpenMP	Open Multi Processing
PBS	Portable Batch System
PyCOMPSs	Python binding for COMPS Superscalar
QoI	Quantity of Interest
SLURM	Simple Linux Utility for Resource Management
SSD	Solid-State Drive
UQ	Uncertainty Quantification

1 Introduction

Task 4.6 in ExaQUte aims at performing continuous improvements to the scheduling mechanisms and the development of optimizations specific to UQ and to the different MC algorithms. During the first phase of the project, we analysed and optimized the execution of MLMC problems using solvers whose internal parallelism was provided within a single node with the OpenMP programming model and runtimes which was reported in D4.3 [21]. In the second phase of the project, we have focused on supporting and optimizing larger problems which require the execution of solvers implemented on top of MPI. This deliverable presents the final version of the ExaQUte framework and the evaluation using as benchmark a UQ problem implemented with a MC algorithm and an MPI enabled solver.

The deliverable is structured as follows: Section 2 provides the overview of the ExaQUte framework and the links to the code repository, installation and usage instructions; Section 3.1 present the evaluation of the framework; and Section 4 draws the conclusions of the deliverable.

2 ExaQUte framework

Figure 1 depicts the different components of the ExaQUte software framework. On the top, we show the Solvers layer and the xMC library which allow developers to easily implement different Monte Carlo approaches for solving uncertainty quantification problems. This library is implemented on top of the ExaQUte API which allow the uniformed usage of different task-level parallel programming models and runtimes.

The following paragraph provide more details about these components as well as the links to repositories and installation and usage guidelines.

2.1 Solvers

The solvers layer mainly includes two software components: Kratos and ParMmg. These components were released in D1.4 [3]. This deliverable include a brief description and links to the download and documentation the latest versions. More details and usage examples can be found in D1.4.

2.2 Kratos Multipysics

The Kratos software[24] is used as the basic multi-physics solver for dealing with a multitude of different physical problems, such as computational fluid dynamics, convection diffusion, structural applications or fluid structure interaction. The Kratos software can be downloaded from the Kratos release page:

`https://github.com/KratosMultiphysics/Kratos/releases`

Installation and usage instructions can be found in the Kratos documentation page:

`https://github.com/KratosMultiphysics/Kratos/wiki`

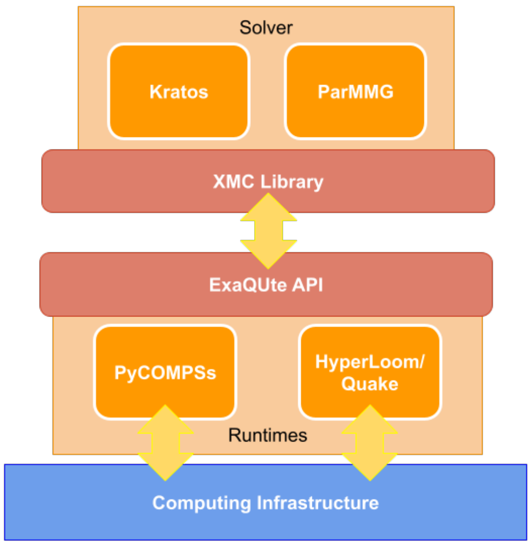


Figure 1: ExaQUte Framework overview.

2.3 ParMmg

ParMmg is an open source software for parallel version of bidimensional and tridimensional surface and volume mesh adaptation [8][13].

The ParMmg software can be downloaded from the following github repository page:
`https://github.com/MmgTools/ParMmg/releases/`
Installation instructions can be found in the following URL:
`https://github.com/MmgTools/ParMmg#readme`
Usage documentation can be found in the following URL:
`https://github.com/MmgTools/ParMmg/wiki#user-guide`

2.4 XMC

The ExaQUte XMC library [4] offers the user the ability to carry out uncertainty quantification simulations using a multitude of Monte Carlo methods. It allows users to configure a wide variety of hierarchical Monte Carlo algorithms, either single- or multi-level, with fixed or adaptive calibration strategies, to estimate various statistics with controlled accuracy. The XMC library was introduced in D5.2 [1]; usage details can be found in this deliverable. It has been interfaced with Kratos Multiphysics and – via the ExaQUte API – with both PyCOMPSs and Hyperloom/Quake.

The XMC code as well as the installation and usage documentation are available in the following public repository:
`https://gitlab.com/RiccardoRossi/exaqute-xmc/`

2.5 ExaQUte API

The ExaQUte API aims at proviving a common and uniformed access to distributed computing resources. It provides a programming interface that allow developers to create application for distributed computing infrastructures following the task-based parallel paradigm. The ExaQUte API is designed to allow the interaction with different runtime systems which will distribute the task execution between the different computing resources. To validate it, two implementations of this API have been developed: one for interacting with the COMPSs runtime developed by BSC and another for the Hyperloom/Quake framework developed by IT4I. The ExaQUte API provides a set of Python decorators to enable the definition of different types of task such as sequential python methods, OpenMP kernels and MPI applications, including hints to enable an efficient execution (such as resource constraints, data directionality and affinity). It also provides a synchronization API to allow users to synchronize remotely generated data and barriers to synchronize tasks' execution. More details about the ExaQUte API are provided in Appendix A

The ExaQUte API implementations and installation guides can be found in the following Github repository:

`https://github.com/ExaQUte-project/exaqute-api`

2.6 Supported Runtimes

As introduced above the ExaQUte API is currently supported by two runtime systems which are able to schedule tasks in distributed computing infrastructures. The following paragraphs provide an overview of these systems

2.6.1 PyCOMPSs/COMPSs

PyCOMPSs [28] is the Python binding of the COMPSs framework [7, 22] that facilitates the development of parallel computational workflows for distributed infrastructures. It offers a programming model based on sequential development – the application is a plain sequential Python script – where the user annotates the functions to be run as asynchronous parallel tasks. This decorator also contains a description of the function parameters, such as type and direction, which are vital for building the dependency graph. In this graph, tasks are represented as nodes and data dependencies between tasks as edges. At execution time, asynchronous tasks are created for each decorated function and forwarded to the COMPSs Runtime which handles data dependency analysis, task scheduling and data transfers. The task creation is performed in an asynchronous way, and once the runtime has added a given task to the dependency graph, the execution of the main Python code continues, possibly generating new tasks. With this aim, PyCOMPSs manages future objects: a representant object is immediately returned to the main program when a task is invoked. A future object returned by a task can be involved in subsequent asynchronous task calls and PyCOMPSs will automatically find the corresponding data dependencies without requiring to wait for the actual result of the task.

PyCOMPSs applications are deployed as master-worker applications, where the master executes the main code and invokes the runtime, and the workers execute the tasks.

During the ExaQUte project, the PyCOMPSs programming model has been extended to support OpenMP an MPI applications as tasks, as well as data layout to support the distribution of data through the different MPI processes to reduce the data management overhead.

These features have been included in version 2.9. This version an higher can be downloaded from the following github repository:
`https://github.com/bsc-wdc/compss`
Further information and documentation about how to install and use PyCOMPSs/-COMPSs can be found in the following link:
`https://compss-doc.readthedocs.io/en/stable/`

2.6.2 Quake

Quake is a tool for scheduling multi-node allocations developed within ExaQUte. It based on the similar ideas as the HyperLoom. HyperLoom was heavily optimized for very different scenario (many subnode tasks); we have decided to create the tool from scratch now fully oriented on multi-node allocations.

The source code as well as the installation and usage instructions can be found in the following public repository.
`https://code.it4i.cz/boh126/quake`

3 Evaluation

To validate the performance of the final ExaQUte framework we have prepared a set of experiments consisting on executing a benchmark application with different configurations and system to evaluate its scalability. In deliverable D4.3 [21], we performed an initial evaluation with an application based on simulations performed with OpenMP. During the last part of the project we have introduced the management of MPI simulations as tasks in ExaQUte framework. For this reason, the evaluation reported in this deliverable will use a benchmark application with the MPI version of the solvers. The next parts of this section will present the benchmark applications, the testbed infrastructure, the description of the experiments, and the obtained results.

3.1 MPI Benchmarks

We solve two different MPI benchmarks. The former is the wind flow past the CAARC building, a standard benchmark in the computational wind engineering field [10, 18, 19, 26], and results are shown in section 3.4. The second benchmark is the wind flow past a rectangle obstacle [11] and results are shown in section 3.5. The second benchmark can be understood as a two-dimensional simplification of the first system.

In both cases the wind flow is modeled with the incompressible Navier-Stokes equations, which read

$$\begin{aligned} \partial_t \boldsymbol{u} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} - \nu\Delta\boldsymbol{u} + \nabla p = \boldsymbol{f} \qquad & \text{in } [0,T]\times D \\ \nabla\cdot\boldsymbol{u} = 0 \qquad & \text{in } [0,T]\times D, \end{aligned} \tag{1}$$

where $\boldsymbol{f}$ is the force vector, ν the kinematic viscosity and, as usual, we denote vectors and tensors using bold characters. In both benchmarks, properties or air are considered.

Equation 1 must be complemented with appropriate boundary conditions and initial conditions. The problem is discretized using linear triangular elements for both pressure and velocity fields. Algebraic subgrid scale stabilization is used to stabilize the problem [14, 15]. A second order fractional step method is used for time stepping that treats both pressure and velocity implicitly.

These benchmarks have been implemented using Kratos as finite element software to solve the computational fluid dynamics problem and XMC as hierarchical MC library. This library uses the ExaQUte API to uniformly interact with the PyCOMPSs/COMPSs and HyperLoom/Quake backends for distributed computing. These examples are available online [25].

Both benchmarks are stochastic, in the sense that boundary conditions are unknown. Details about each boundary condition are given in the two following sections. We aim at solving these stochastic problems using hierarchical MC methods, and we refer to [16, 17, 27, 29] for details about hierarchical MC methods and their asynchronous counterparts. Due to the difficulties of satisfying standard MLMC hypotheses for chaotic systems (see deliverables [5, 6]), the problem is solved using the asynchronous MC method [29].

We are interested in estimating the expected value of of an output QoI. Therefore, we define the MC expected value estimator as

$$\mathrm{E}^{MC}[\mathrm{Q}_H] = \frac{\sum_{n=1}^{N} \mathrm{Q}_H(w^{(n)})}{N}, \tag{2}$$

where Q is the QoI we are interested in, Q_H its discretization on a mesh of characteristic length H, N is the total number of MC realizations and w refers to the random event. The convergence of the asynchronous MC method is evaluated using a failure probability convergence criterion, which reads

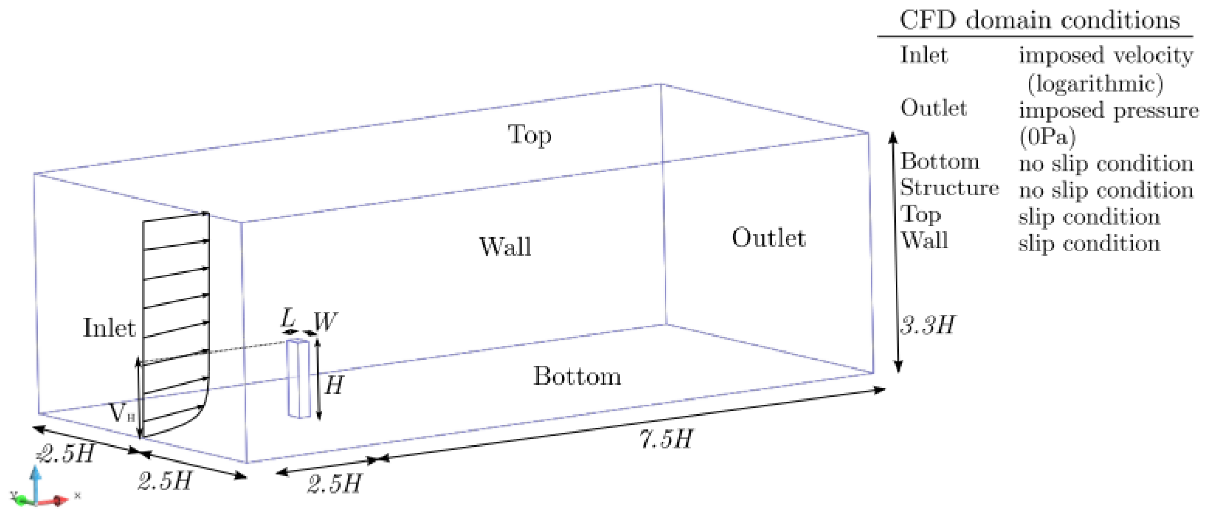
$$\mathbb{P}\left(\left|\mathrm{E}^{MC}[\mathrm{Q}_H] - \mathbb{E}[\mathrm{Q}]\right| \geq \varepsilon\right) \leq \phi, \tag{3}$$

where $\varepsilon > 0$ is the absolute tolerance of the difference between the sampled estimator $\mathrm{E}^{MC}[\mathrm{Q}_H]$ and the true estimator $\mathbb{E}[\mathrm{Q}]$ and $1-\phi \in (0,1)$ is the confidence on the the final statistical estimator. We refer for example to [27, 29] for details about how to compute such a convergence criterion.

A total of six observables are computed at runtime, and are listed next.

- Drag force F_d,
- Base moment M_b,
- Pressure field p on the building surface,
- Time-averaged drag force $\langle F_d \rangle_{T_0,T}$,
- Time-averaged base moment $\langle M_b \rangle_{T_0,T}$,
- Time-averaged pressure field $\langle p \rangle_{T_0,T}$ on the building surface.

We comment that $[0, T_0]$ is the burn-in phase, and is discarded to remove the bias of initial conditions, while $[T_0, T]$ is the effective time. Convergence is assessed only for the time-averaged drag force, even though more observables are computed and their associated statistics estimated. International units are used to measure quantities.



CFD domain conditions	
Inlet	imposed velocity (logarithmic)
Outlet	imposed pressure (0Pa)
Bottom	no slip condition
Structure	no slip condition
Top	slip condition
Wall	slip condition

Figure 2: CAARC problem domain. $H = 180\,\text{m}$, $W = 45\,\text{m}$ and $L = 30\,\text{m}$.

3.1.1 High rise building benchmark

This benchmark solves the wind flow past the CAARC building and is also solved in other ExaQUte reports [2, 9, 30, 31]. The domain is presented in figure 2. The wind velocity mean profile is logarithmic and modeled as in [23], and we refer to our deliverable [9] for details about the wind generation. The wind reference velocity is $40\,\text{m}\,\text{s}^{-1}$ at the reference height of 180 m. The roughness height is a uniformly distributed random variable $\mathcal{U}[0.1, 0.7]$, where 0.1 and 0.7 are the minimum and maximum values, respectively. Such a roughness height is typical of sparsely built-up urban areas [20]. In other words, the wind mean profile is constant in time and stochastic, and on top of it there are fluctuations. The Reynolds number is of the order of 10^8, computed with a characteristic length of 45 m and density and viscosity of air. A velocity field snapshot can be observed in figure 3.

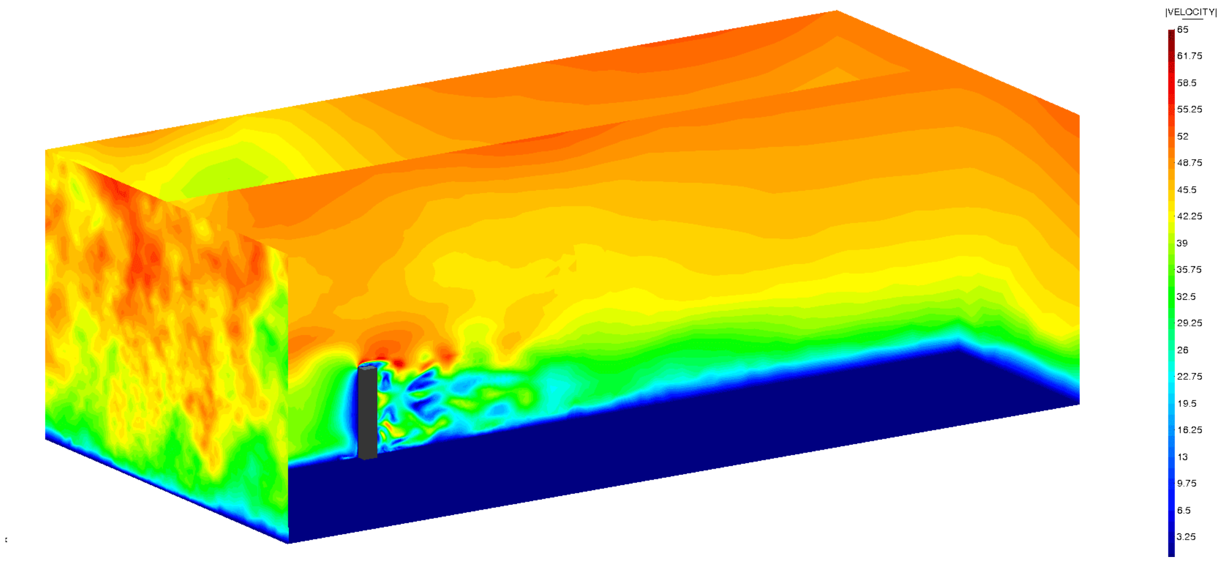


Figure 3: Snapshot of the velocity field.

3.1.2 Rectangle obstacle benchmark

This benchmark solves the wind flow past a rectangle obstacle problem and is also solved in other ExaQUte reports [5, 6, 30, 31]. The problem domain is shown in figure 4. The

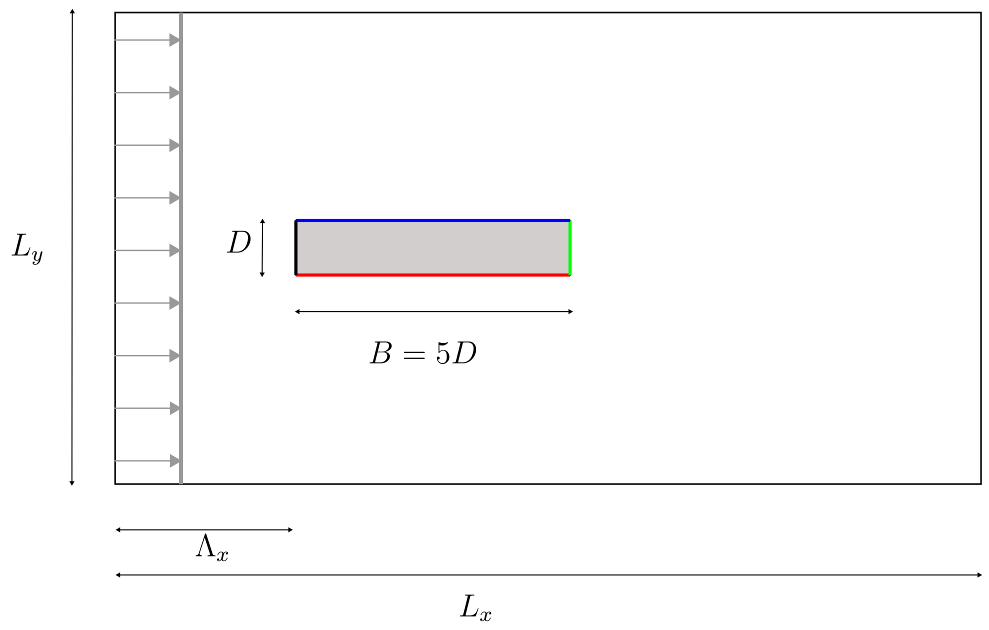


Figure 4: Scheme of the computational domain used for the rectangle problem, where $D = 1\,\mathrm{m}$, $B = 5D$, $L_x = 55B$, $L_y = 30B$ and $\Lambda_x = 15B$. Thus, the dimensions of the outer domain are $275 \times 150\,\mathrm{m}$, and the inner rectangle has size $5 \times 1\,\mathrm{m}$.

inlet velocity is constant in time, uniformly distributed on the y-axis, and has an average value of $2\,\mathrm{m\,s^{-1}}$. We assume the wind inlet velocity magnitude is represented by a normal distribution,

$$\boldsymbol{u}_{inlet} \sim \mathcal{N}(2.0, 0.02). \tag{4}$$

Slip boundary conditions are applied on the external boundaries, and no-slip boundary conditions are enforced on the rectangle body. The Reynolds number of the problem is 132719. We remark that, even though turbulence cannot be identified in two-dimensional systems, the rectangle obstacle problem we define is particularly important, since it provides a cheap yet accurate system for studying features of chaotic flows. A velocity field snapshot can be observed in figure 5.

3.2 Testbeds

3.2.1 MareNostrum Supercomputer

The MareNostrum 4 Supercomputer [12] located at the Barcelona Supercomputing Center (BSC). Its current peak performance is 11.15 Petaflops, ten times more than its previous version, MareNostrum 3. The supercomputer is composed by 3,456 nodes, each of them with two Intel®Xeon Platinum 8160 (24 cores at 2,1 GHz each). Regarding memory, there are two types of nodes: regular memory nodes with 2GB per core and fat memory nodes with 4GB per core. In this evaluation, we have only used regular nodes. All nodes are connected through an 100Gb Intel®Omni-Path Full-Fat Tree Interconnection

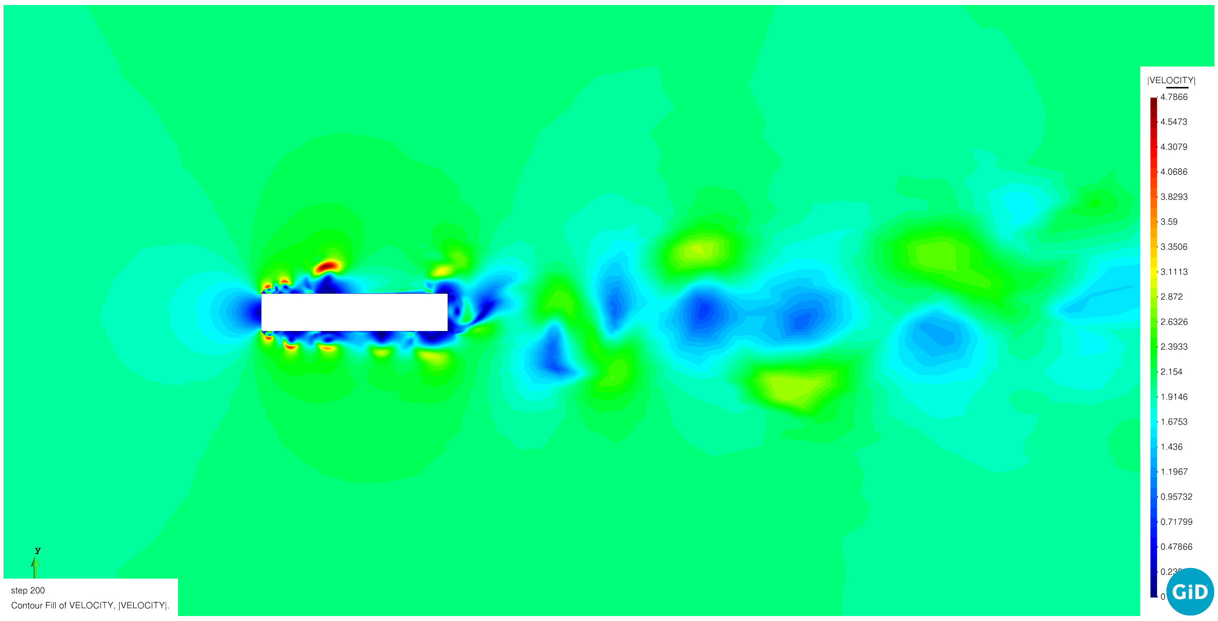


Figure 5: Velocity field snapshot.

network and have access to a shared disk storage managed by the IBM®Global Parallel File System.

3.2.2 Barbora Supercomputer

The Barbora supercomputer was installed in autumn 2019 with a theoretical peak performance of 849 TFlop/s.

The computing power consists of:

- 192 standard computational nodes; each node is equipped with two 18-core Intel processors and 192 GB RAM,
- 8 compute nodes with GPU accelerators; each node is equipped with two 12-core Intel processors, four NVIDIA Tesla V100 GPU accelerators with 16 GB of HBM2 and 192 GB of RAM,
- 1 fat node is equipped with eight 16-core Intel processors and 6 TB RAM.

The supercomputer is built on the Bull Sequana X architecture and for cooling its standard compute nodes the direct liquid cooling technology is used. The computing network is built on the latest Infiniband HDR technology. The SCRATCH computing data storage capacity is 310 TB with 28 GB/s throughput using Burst Buffer acceleration. Another computing data storage is NVMe over Fabric with a total capacity of 22.4 TB dynamically allocated to compute nodes. It is also equipped with the Bull Super Computer Suite cluster operation and management software solution as well as PBS PRO scheduler and resource manager. The computing network is built with the latest Infiniband HDR technology.

3.3 Experiments description

To evaluate the performance of the ExaQUte framework, we are going to perform a strong and weak scaling analysis using the two different runtimes in the different available supercomputers.

For the strong scalability analysis, we have executed the MPI Benchmark applications with a fixed configuration and a variable number of nodes. In the case of weak scalability analysis, we are changing the configuration of each execution in order to scale the load of the execution in the same portion as the increment of resources. Regarding the MPI Benchmark applications, the parallel load of the application is given by the number of parallel simulations which can run at the same time. This is controlled by the number of simulations per batch and number of initial batches. So, for the strong scaling analysis, we keep all the parameters fixed (the simulation time window, number of MPI processes per simulation, number of simulations per batch, number of batches and other input parameters such as the considered mesh, etc.). In the case of the weak scaling, we just scale the number of initial batches with the same proportion as number of cores.

The following sections show the results obtained from the execution of the strong and weak experiments with the different benchmarks, runtimes and clusters.

3.4 COMPSs Results

This section presents the results obtained for running the the High rise building benchmark, described in Section 3.1 with the COMPSs backend in the the MareNostrum supercomputer. Table 1 and Table 2 show the different application and computing insfrastructure configurations selected for running the strong and weak scaling experiments described in Section 3.3.

Application Setup	Sim. End time	5			
	MPI Processes/sim.	16			
	Sims./batch	386			
	Initial batches	2			
	Iterations	2			
Cluster Setup	Nodes	16	32	64	128
	Cores	768	1536	3072	6144

Table 1: Setup for the Strong Scaling Experiments with COMPSs in the MareNostrum supercomputer.

Figure 6 shows the results for the strong scaling experiment. In this figure, we can see we have obtained a good strong scalability which is close to the ideal. The deviation from the ideal scaling is due to the sequential initialization, where only a task is active, and the simulations variability which is accumulated at the end of the execution.

In figure 7 we can see the comparison of two execution traces of the strong scaling experiments corresponding to the same application execution with 32 and 64 nodes. We can observe that the initialization task (blue line at the left of the traces) and the variability parts (tail of white executions) remains constant while the main computation is reduced when increasing the number of nodes. For a larger number of nodes, this part is becoming more relevant and limits strong scalability.

Application Setup	Sim. End time	5			
	MPI Processes/sim.	16			
	Sims./batch	386			
	Initial batches	2	4	8	16
	Iterations	2	4	8	16
Cluster Setup	Nodes	16	32	64	128
	Total Cores	768	1536	3072	6144

Table 2: Setup for the Weak Scaling experiments with COMPSs in the MareNostrum supercomputer.

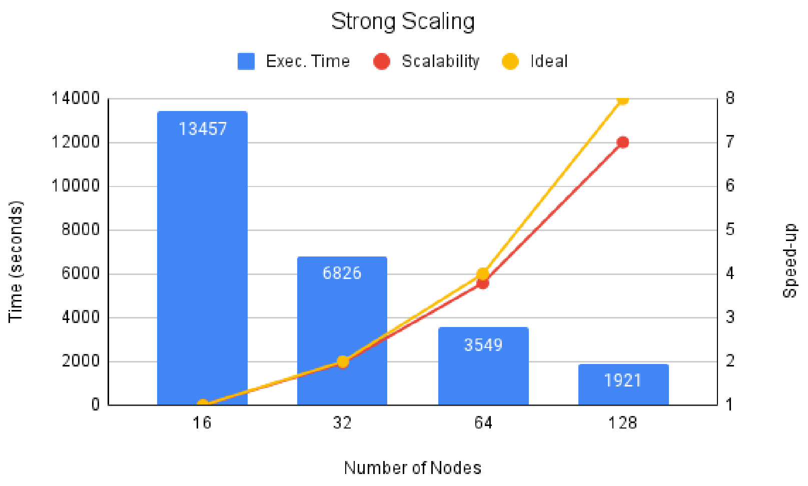


Figure 6: Strong scaling results for COMPSs runtime in MareNostrum supercomputer. The baseline for calculating the scalability is the execution time obtained for 16 nodes.

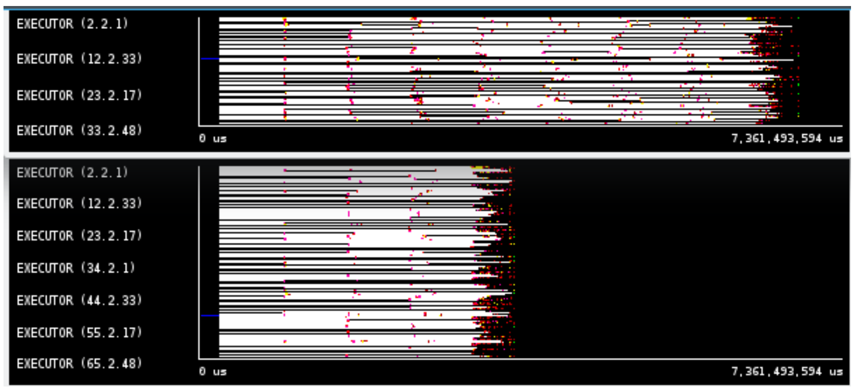


Figure 7: Comparison of two execution traces of the strong scaling experiments running the MPI Benchmark with the COMPSs runtime in MareNostrum supercomputer with 32 and 64 nodes. The two traces are at the same time scale, showing almost perfect scaling.

Figure 8 shows the results for the weak scaling experiment. In this case, we are scaling the computation load the same way as the resources, and the time remains almost

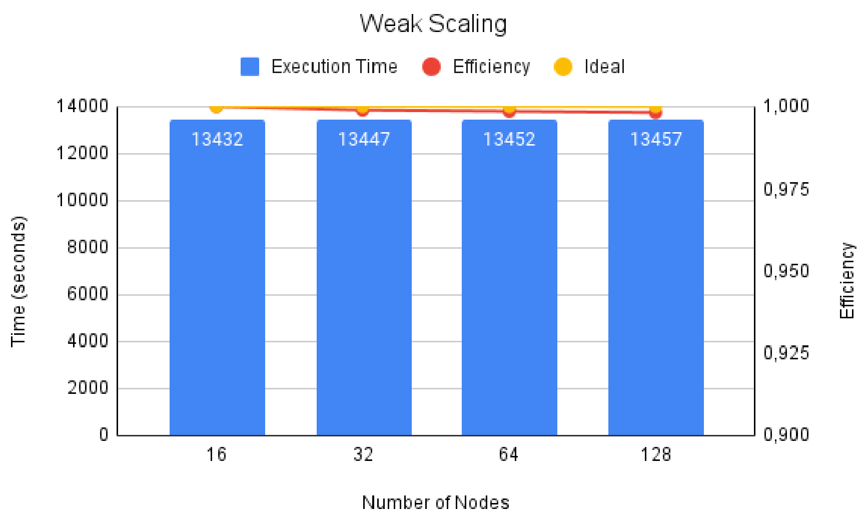


Figure 8: Weak scaling results for COMPSs runtime in MareNostrum supercomputer. The baseline for calculating the efficiency is the execution time obtained for 16 nodes.

constant, achieving a very good weak scalability.

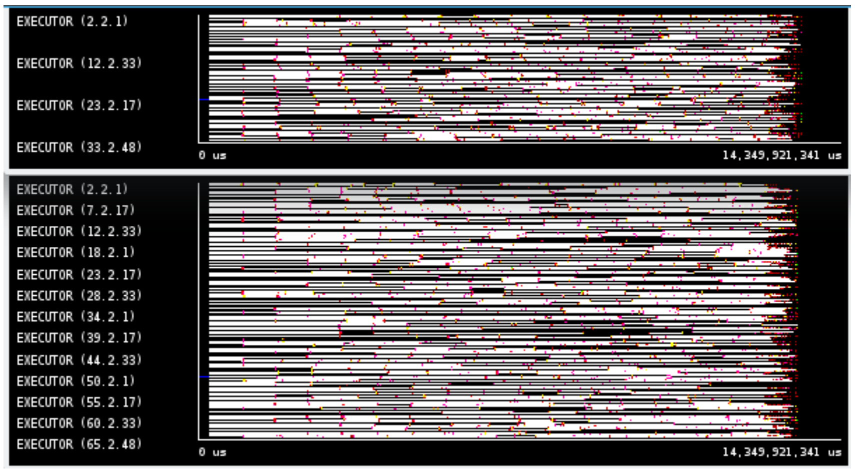


Figure 9: Comparison of two execution traces of the weak scaling experiments running the MPI Benchmark with the COMPSs runtime in MareNostrum supercomputer with 32 and 64 nodes.

As we can observe in Figure 9 the initialization task and variability remains constant for all the executions and proportionally small in all cases, not impacting the weak scaling.

3.5 Quake Results

Quake backend was tested on MPI version of Rectangle obstacle benchmark. Quake was designed from the scratch as a multi-node scheduler and does not address individual CPUs; it assigns a whole node for one MPI processes. Kratos in the current version does not support MPI+OpenMP; hence it would lead to utilization of a single core per node if it is used in combination of Quake. Therefore, we have decided to benchmark it on a smaller instance.

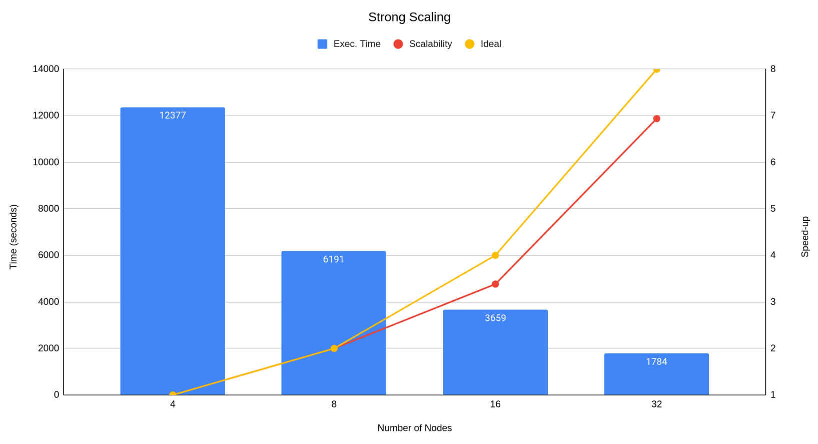


Figure 10: Two-dimensional flow benchmark, executed on Barbora with Quake backend; strong scalability

The strong scalability results are shown in Figure 10; as a baseline, the result on 4 nodes was taken.

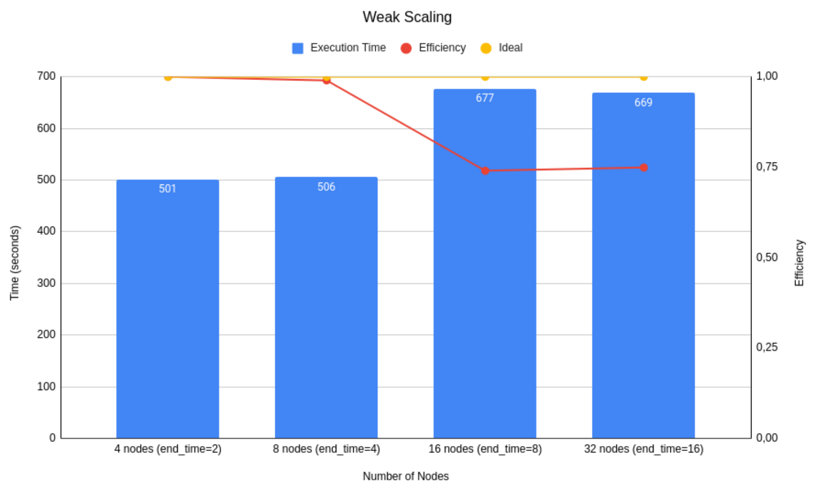


Figure 11: Two-dimensional flow benchmark, executed on Barbora with Quake backend; weak scalability

The weak scalability results are shown in Figure 10. The instance was scale up by modifying `end_time` parameter.

4 Conclusions

This deliverable provides the release of the final version of the ExaQUte framework. The first part of the document briefly describes the different components of the framework and links to the code and installation and usage guidelines.

The second part of the document reports the evaluation of the latest developments of the framework. The results of the evaluation showed that the framework achieves good strong and weak scalability for the evaluated benchmarks.

A Appendix: Final specification of ExaQUte API

A.1 Decorators

A.1.1 Task decorator

First of all, a decorator has been defined to identify the tasks. Its basic syntax is shown in Figure 12. All the functions marked with this decorator will be executed remotely.

```
from exaqute import task
from exaqute import INOUT

@constraint (computing_units="4")
@task (c = INOUT)
def func(a, b, c):
    c += a*b
    ...
```

Figure 12: Task basic usage. Parameters *a* and *b* are assumed as input objects and *c* is an object modified by the task

The following lists shows the possible properties of a task decorator:

- `returns=<number|string with environment variable>` allows the user to indicate how many return values have the task. By default, if the this property is not defined, the scheduler assumes that the task does not generates any return value.
- `keep=<true|false>` indicates if the user wants to hold the resulting value(s) of the task. If the flag is set to *true*, the the task keep the results and the user is responsible for an explicit clean-up. Otherwise, the task result is considered temporary and it can be removed when the runtime considers appropriate.
- `<variable_name>=<parameter description>` allow the users to provide hints to the scheduler about task parameters to optimize the application execution as much as possible.

Parameters are mainly described by its type and direction. The supported types in the ExaQUte tasks are Basic parameter(integer, boolean, strings, ...), serializable object, FILE and COLLECTION. The type parameter type can be inferred from the code. However, there are ambiguous cases, such as files or collections, that the automatic inference can lead to an unexpected behaviour. For instance a filename can be treated as an string or a file. If it is treated as object the path must be available from all the nodes, otherwise the remote execution of a task will fail. In the case, it is defined as file, transfer management and the path transformation can be managed by the scheduler. In the cases of lists, we could treat the whole list as a single object or a collection of objects. A wrong definition can make that the scheduler do not detect task dependencies properly. A parameter description can be provided done in two ways as a *key:value* fashion or as an abbreviation, as indicated below.

- `<variable_name>={key1:val1, key2:val2>}`
- `<variable_name>=Abbreviation`

The supported keys used in the ExaQUte API are the following:

- `Type:<FILE|COLLECTION>` indicates the parameter type when no automatic inference.
- `Direction:<IN|OUT|INOUT>` indicates the parameter direction when no automatic inference.
- `Depth:<number>` when nested collections, indicates the depth level used to detect dependencies.

The supported abbreviations in the ExaQUte API are the following:

- `IN|OUT|INOUT` indicates the parameter is an object whose direction is IN, OUT or INOUT.
- `FILE_<IN|OUT|INOUT>` indicates the parameter is a file whose direction is IN OUT or INOUT.
- `COLLECTION_<IN|OUT|INOUT>` indicates the parameter is a list of parameters whose direction is IN OUT or INOUT.

A.1.2 Constraint decorator

It is possible to define constraints for each task. To this end, the decorator @*constraint* followed by the desired constraints needs to be placed over the @task decorator.This decorator enables the user to set the particular constraints for each task, such as the number of Cores required explicitly. Alternatively, it is also possible to indicate that the value of a constraint is specified in an environment variable. Figure 13 shows how to express these constraints. The example of the figure can can be used to indicate the task is internally parallelized by threads (such as pthreads or OpenMP) and it will require 4 cores (*computing_units*) and a certain memory(*memory_size*) to be efficiently executed.

```
from exaqute import task
from exaqute import constraint
from exaqute import INOUT

@constraint (computing_units="4", app_software="numpy",
    memory_size="$MIN_MEM_REQ")
@task (returns=1)
def func(a, b, c):
   return a*b

```

Figure 13: Tasks Constraint usage

A.1.3 MPI tasks decorator

A new Python decorator is used to indicate that a task is implemented with MPI. In this decorator, a developer can specify the following properties:

- ***runner*** This specifies the command to spawn the MPI processes. By default an MPI application is executed with the *mpirun* command, but there are other runners to spawn MPI processes such as the *srun* command in systems managed by Slurm
- ***processes*** Indicates the number of MPI processes used by this task

- ***processes_per_node*** Indicates the number of MPI processes co-allocated in the same node to limit the distribution of the mpi processes. (This flag is ignored in the case of Quake backend)
- ***data layout*** Indicates the mapping between the elements of a collection task parameters and the MPI processes. It allows the scheduler to optimize data transfers and object serialization/deserialization in order to reduce the MPI execution overhead. The syntax used to define the data layout is the same as the used by MPI to define a strided vector (*MPI_type_vector*), where the developer has to indicate: the number of blocks (*block_count*), the number of contiguous elements per block (*block_length*) and the number of elements between start of each block (*stride*). If no layout is defined for a parameter, it is assumed that all the MPI processes will require all the elements of the parameter.

```python
nprocs = 4

@mpi(runner="mpirun", processes=nprocs)
@task(returns=nprocs)
def init(seed):
    from mpi4py import MPI
    rank = MPI.COMM_WORLD.rank
    return rank+seed

@mpi(runner="mpirun", processes=nprocs)
@task(input_data=COLLECTION_IN, returns=nprocs)
def scale(input_data, i):
    from mpi4py import MPI
    rank = MPI.COMM_WORLD.rank
    a = input_data[rank]*i
    return a

@mpi(runner="mpirun", processes=nprocs,
     input_data_layout={block_count:nprocs, block_length:1, stride:1}))
@task(input_data=COLLECTION_IN, returns=nprocs)
def increment(input_data):
    from mpi4py import MPI
    rank = MPI.COMM_WORLD.rank
    a = input_data + 1
    return a

@mpi(runner="mpirun", processes=nprocs,
     idata_layout={block_count:nprocs, block_length:nprocs, stride:nprocs})
@task(idata={Type:COLLECTION_IN, Depth:2}, returns=nprocs)
def merge(idata):
    from mpi4py import MPI
    rank = MPI.COMM_WORLD.rank
    a=0
    for data in idata:
        a=a+data
    return a

if __name__ == '__main__':
    input_data = init(0)
    partial_res=[]
    for i in [1,10,20,30]:
        p_data = scale(input_data, i)
        for j in range(2):
            p_data = increment(p_data)
        partial_res.append(p_data)
    results= merge(partial_res)
```

Figure 14: Application example with collections and MPI tasks

Figure 14 shows an application example o where the *@mpi* decorator and *COLLECTION* data type are combined. This sample application is performing several simple operations (*scale*, *increment*) over a set of data generated in the *init* task. All these operations are parallelized with MPI an defined as MPI tasks. In the code snippet, you can see that the init task implements an MPI task which returns a list of four values, each one generated by a different MPI process. Then the *scale* and *increment* tasks get a collection of four values and generate a new collection, where each value of the collection is computed by a MPI process. To illustrate the data layout fuctionality, we have defined a layout for the *increment* task, but not for the *scale* task. The difference is that for the *scale* task all the elements of the *input_data* collection will be transferred and deserialized in all the processes, while only one element per process will be transferred and deserialized in the*increment* task. Finally, the *merge* task is getting a collection of 16 values (four lists of four elements) and we are indicating with the layout property, that indicates data is treated in four disjoint blocks of four elements.

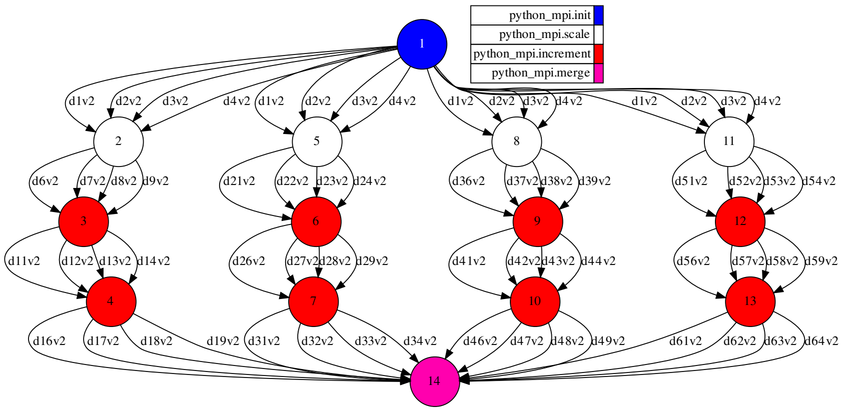


Figure 15: Example application execution graph.

When the code of the figure is executed, the runtime system is able to detect the dependencies between tasks generating the DAG depicted in Figure 15. Next section provides the details about how the runtime manages the execution of this type of graphs composed by MPI tasks.

A.2 API calls

This section provides the definition of the ExaQUte API calls. The following paragraphs provide list of functions and its expected behavior:

- `init()`
  Explicit scheduler environment initialization.
- `barrier()`
  The purpose of this call is to wait until all the tasks have been executed.
- `get_value_from_remote(obj)`
  This API call brings back to the master/client the value of a remote computed object.

- `delete_object(obj)`
  This API call removes all the copies allocated in the workers of the given object.
- `delete_file(path_to_file)`
  This API call removes all the copies allocated in the workers of the given file.

References

[1] R. Amela, Q. Ayoul-Guilmard, S. Ganesh, R. Tosi, R. M. Badia, F. Nobile, R. Rossi, and C. Soriano. D5.2 Release of ExaQUte MLMC Python engine. Technical report, Open Access Repository of the ExaQUte project: Deliverables, 2019. URL `https://doi.org/10.23967/exaqute.2021.2.024`.

[2] A. Apostolatos, R. Rossi, and C. Soriano. D7.2 Finalization of "deterministic" verifcation and validation tests. Technical report, Open Access Repository of the ExaQUte project: Deliverables, 2020. URL `https://doi.org/10.23967/exaqute.2021.2.006`.

[3] Q. Ayoul-Guilmard, S. Ganesh, F. Nobile, R. M. Badia, J. Ejarque, B. Keith, A. Kodakkal, M. Núñez, C. Roig, R. Rossi, R. Tosi, and C. Soriano. D1.4 Final public Release of the solver. Technical report, Open Access Repository of the ExaQUte project: Deliverables, 2020. URL `https://doi.org/10.23967/exaqute.2021.2.009`.

[4] Q. Ayoul-Guilmard, S. Ganesh, F. Nobile, R. Rossi, R. Tosi, R. M. Badia, and R. Amela. XMC: Python library for hierarchical Monte Carlo algorithms. software, ExaQUte, 2020. URL `https://doi.org/10.5281/zenodo.3235832`.

[5] Q. Ayoul-Guilmard, S. Ganesh, M. Núñez, R. Tosi, F. Nobile, R. Rossi, and C. Soriano. D5.3 Report on theoretical work to allow the use of MLMC with adaptive mesh refinement. Technical report, Open Access Repository of the ExaQUte project: Deliverables, 2020. URL `https://doi.org/10.23967/exaqute.2021.2.002`.

[6] Q. Ayoul-Guilmard, S. Ganesh, M. Núñez, R. Tosi, F. Nobile, R. Rossi, and C. Soriano. D5.4 Report on MLMC for time dependent problems. Technical report, Open Access Repository of the ExaQUte project: Deliverables, 2020. URL `https://doi.org/10.23967/exaqute.2021.2.005`.

[7] R. M. Badia, J. Conejero, C. Diaz, J. Ejarque, D. Lezzi, F. Lordan, C. Ramon-Cortes, and R. Sirvent. {COMP} Superscalar, an interoperable programming framework. *SoftwareX*, 3–4, 2015. doi:10.1016/j.softx.2015.10.004.

[8] P. Benard, G. Balarac, V. Moureau, C. Dobrzynski, G. Lartigue, and Y. d'Angelo. Mesh adaptation for large-eddy simulations in complex geometries. *International journal for numerical methods in fluids*, 81(12):719–740, 2016.

[9] S. Bidier, U. Khristenko, R. Tosi, R. Wuchner, A. Michalski, R. Rossi, and C. Soriano. D7.3 Report on UQ results and overall user experience. Technical report, Open Access Repository of the ExaQUte project: Deliverables, 2021. URL `https://doi.org/10.23967/exaqute.2021.9.002`.

[10] A. L. Braun and A. M. Awruch. Aerodynamic and aeroelastic analyses on the CAARC standard tall building model using numerical simulation. *Computers and Structures*, 87(9-10):564–581, may 2009. ISSN 00457949. doi:10.1016/j.compstruc.2009.02.002. URL `http://dx.doi.org/10.1016/j.compstruc.2009.02.002`.

[11] L. Bruno, M. V. Salvetti, and F. Ricciardelli. Benchmark on the aerodynamics of a rectangular 5:1 cylinder: An overview after the first four years of activity. *Journal of Wind Engineering and Industrial Aerodynamics*, 126:87–106, mar 2014. ISSN 01676105. doi:10.1016/j.jweia.2014.01.005.

[12] B. S. C. (BSC). MareNostrum IV Technical Information. `https://www.bsc.es/marenostrum/marenostrum/technical-information`, 2021.

[13] L. Cirrottola and A. Froehly. Parallel unstructured mesh adaptation based on iterative remershing and repartitioning. In *WCCM-Eccomas 2020-14th World Congress on Computational Mechanic*, 2021.

[14] R. Codina. Comparison of some finite element methods for solving the diffusion-convection-reaction equation. *Computer Methods in Applied Mechanics and Engineering*, 156(1-4):185–210, apr 1998. ISSN 00457825. doi:10.1016/S0045-7825(97)00206-5.

[15] R. Codina. Pressure Stability in Fractional Step Finite Element Methods for Incompressible Flows. *Journal of Computational Physics*, 170(1):112–140, 2001. ISSN 00219991. doi:10.1006/jcph.2001.6725.

[16] M. B. Giles. Multilevel Monte Carlo path simulation. *Operations Research*, 56(3): 607–617, 2008.

[17] M. B. Giles. Multilevel Monte Carlo methods. *Acta Numerica*, 24:259–328, 2015.

[18] J. D. Holmes and T. K. Tse. International high-frequency base balance benchmark study. *Wind and Structures, An International Journal*, 18(4):457–471, 2014. ISSN 12266116. doi:10.12989/was.2014.18.4.457.

[19] S. Huang, Q. S. Li, and S. Xu. Numerical evaluation of wind effects on a tall steel building by CFD. *Journal of Constructional Steel Research*, 2007. ISSN 0143974X. doi:10.1016/j.jcsr.2006.06.033.

[20] J. JCSS. Probabilistic model code. *Joint Committee on Structural Safety*, 2001.

[21] T. Karasek, S. Böhm, B. Keith, R. Amela, R. M. Badia, R. Rossi, and C. Soriano. D4.3 Benchmarking report as tested on the available infrastructure. Technical report, Open Access Repository of the ExaQUte project: Deliverables, 2020.

[22] F. Lordan, E. Tejedor, J. Ejarque, R. Rafanell, J. Álvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia, and R. M. Badia. ServiceSs: An Interoperable Programming Framework for the Cloud. *Journal of Grid Computing*, 12(1):67–91, 2014. ISSN 15707873. doi:10.1007/s10723-013-9272-5.

[23] J. Mann. Wind field simulation. *Probabilistic Engineering Mechanics*, 13(4):269–282, 1998. ISSN 02668920. doi:10.1016/s0266-8920(97)00036-2. URL `https://ac-els-cdn-com.recursos.biblioteca.upc.edu/S0266892097000362/1-s2.0-S0266892097000362-main.pdf?_tid=35c100fe-45d6-41b7-8534-b299ed7855a3{&}acdnat=1531413687_51eb28e70aa14ade51f9d2c0ef662d38`.

[24] V. Mataix Ferrándiz, P. Bucher, R. Rossi, R. Zorrilla, J. Cotela Dalmau, J. Maria, M. A. Celigueta, and G. Casas. KratosMultiphysics/Kratos: KratosMultiphysics 9.0, 2021. URL `https://doi.org/10.5281/zenodo.3234644`.

[25] V. Mataix Ferrándiz, R. Tosi, I. de Pouplana, R. Zorrilla, L. Gracia, S. Warnakulasuriya, M. Núñez, B. Chandra, A. Ghantasala, P. Bucher, J. Cotela, A. Franci, F. Arrufat, K. B. Sautter, S. Latorre, J. González-Usúa, A. Geiser, P. Dadvand, M. Maso, D. Baumgaertner, M. A. Celigueta, R. Kikkeri Nagaraja, S. Wenczowski, B. Saridar, C. Roig, M. Zidan, and G. Casas. KratosMultiphysics/Examples: Kratos Examples 9.0, 2021. URL `https://doi.org/10.5281/zenodo.4293799`.

[26] E. D. Obasaju. Measurement of forces and base overturning moments on the CAARC tall building model in a simulated atmospheric boundary layer. *Journal of Wind Engineering and Industrial Aerodynamics*, 1992. ISSN 01676105. doi:10.1016/0167-6105(92)90361-D.

[27] M. Pisaroni, F. Nobile, and P. Leyland. A Continuation Multi Level Monte Carlo (C-MLMC) method for uncertainty quantification in compressible inviscid aerodynamics. *Computer Methods in Applied Mechanics and Engineering*, 326:20–50, 2017. ISSN 00457825. doi:10.1016/j.cma.2017.07.030.

[28] E. Tejedor, Y. Becerra, G. Alomar, A. Queralt, R. M. Badia, J. Torres, T. Cortes, and J. Labarta. PyCOMPSs: Parallel computational workflows in Python. *International Journal of High Performance Computing Applications*, 31(1):66–82, 2017. ISSN 17412846. doi:10.1177/1094342015594678.

[29] R. Tosi, R. Amela, R. Badia, and R. Rossi. A parallel dynamic asynchronous framework for Uncertainty Quantification by hierarchical Monte Carlo algorithms. *Journal of Scientific Computing*, 89(28):25, 2021. doi:10.1007/s10915-021-01598-6.

[30] R. Tosi, M. Núñez, J. Pons-Prats, J. Principe, and R. Rossi. D3.3 Report of ensemble based parallelism for turbulent flows and release of solvers. Technical report, Open Access Repository of the ExaQUte project: Deliverables, 2021.

[31] R. Tosi, M. Núñez, J. Pons-Prats, J. Principe, and R. Rossi. D3.4 Report on the calibration of parallel methods for transient problems in wind engineering. Technical report, Open Access Repository of the ExaQUte project: Deliverables, 2021.