D2.4

Final Model Workflows

Work package	2 Middleware Requirements from Workflows	
Author(s)	Francois Tessier Julien Capul Domokos Sarmany Salem El Sayed Jayesh Badwaik Dirk Pleiter	CSCS CEA ECMWF Juelich Juelich Juelich
Reviewer #1	Javier Novo	APPENTRA
Reviewer #2	Ganesan Umanesan	Seagate
Dissemination Level	Public	
Nature	Other - Software Deliverable	

Date	Author	Comments	Version	Status
22.07.2020	Domokos Sármány		v0.1	Draft
26.07.2020	Domokos Sármány		v0.2	Draft
06.08.2020	Jayesh Badwaik	All application input complete	v0.3	Draft
11.08.2020	Jayesh Badwaik	Moved from Older Template	V0.4	Draft
13.08.2020	Salem El Sayed	Ready for internal review	v0.5	Draft
21.08.2020	Domokos Sármány		v0.6	Draft
29.08.2020	Salem El Sayed	Final Checks	v1.0	Draft

DATA ORCHESTRATION IN HIGH PERFORMANCE COMPUTING
This project has received funding from the European Union's Horizon 2020 research and innovation program through grant agreement 801101.

Executive Summary

The Maestro project is oriented around a co-design process between application authors along with system and middleware designers. One of the challenges of a co-design process is the inherent complexity of real-world applications and the effort required to port them to different systems or to use newly created functionality. These applications also exhibit large variation of input data and configuration parameters.

As a result, the project partners have built simplified workflows that demonstrate the relevant features of larger processes, along with the configuration and input data to run them in a self-contained manner.

This document is a summary of the final release of the model workflows, each corresponding to one or more of the usage scenarios described in deliverable [D2.1]. The introduced workflows represent the most important characteristics of the use cases and together they form the basis for the Maestro middleware evaluation from the applications' perspective. They have been updated with workloads that represent current challenges in HPC applications and this document will also include a summary of performance metrics to be used in the Maestro middleware evaluation.

Contents

Glossary

TerrSysMP or TSMP	Terrestrial System Modeling Platform
CLM	Community Land Model
ParFlow	PARallel Flow
PDAF-D	Parallel Data Assimilation Framework
FDB	Fields Database
MultIO	Multiplexing I/O library
Pgen	Product-generation engine
CFD	Computational Fluid Dynamics

1. Introduction

Fully featured, operational, HPC workflows are large and unwieldy. The challenges they face at scale motivate the efforts of the project to develop the Maestro middleware. However, the effort required to move and restructure these workflows to use a novel middleware, and to run on different systems can be very large.

Within the Maestro project, the application partners (ECMWF, JUELICH, CEA and ETHZ) have built stripped-down model workflows. These correspond to the usage scenarios and use cases as described in deliverable [D2.1]. These workflows can be more straightforwardly modified to make use of the middleware and to demonstrate the impact of the developments in the project.

This deliverable is an accompanying report presenting the final state of the software for the model workflows before porting them to use the Maestro middleware. The term workload includes not only the software, but the parameters, configurations and data required to run a body of work on the system. Each of the supplied model workflows is thus a self-contained collection of components, parameters and input data. Further they contain documentation of prerequisites and instructions for building, installing and running the workloads on different machines.

The focus of these workflows is on features and functionality relevant to the middleware developments in the Maestro project. A model workflow needs to be sufficient to form a demonstrator for the Maestro middleware for given usage scenarios and use cases. These workflows will be used as a proof-of-concept for porting applications to the Maestro middleware. However, these workflows are not intended to be full operational workflows containing all components. Those which are too complex, proprietary or which detract from the feature demonstration are omitted. Any additional components that have not been supplied will be described in the documentation.

The released model workflows represent the most important characteristics of the usage scenarios and together they form the basis for the Maestro middleware evaluation from the applications' perspective. They have been updated with workloads that represent current challenges in HPC applications and this document will also include a summary of performance metrics to be used in the Maestro middleware evaluation.

1.1 Relation to Other Work Packages

The main task of WP2 is to define application requirements for the core middleware (WP3), the high-level middleware (WP4) and, usually indirectly, to the low-level middleware (WP5). These requirements have been derived from actual, real-life workflows and are the result of active co-design work between the application partners as well as between work packages 2, 3, 4 and 5. The requirements and the co-designed middleware API have been documented in deliverables D2.1 and D2.3, respectively.

In addition, usage scenarios from actual workflows were used to provide a list of use cases for each application. A use case captures certain aspects of a usage scenario and it focuses on specific characteristics. Rather than running actual workflows, the application partners have been tasked to build model workflows that can implement one or more of the use cases. The model workflows are designed to capture the most important requirements and will be used as application software demonstrators in WP6. They play an important role in the middleware evaluation, which is also primarily part of WP6.

This deliverable D2.4 is the final version of the model workflows and is based on the initial version of the same model workflows released as part of D2.2.

2. Target applications for the Maestro middleware

The overall objective of the Maestro project is to build a middleware framework that addresses problems of data movement in complex memory hierarchies at multiple levels of the HPC software stack. So, superficially at least, any application running in an HPC environment and requiring intensive data movement and/or data transformations may see a benefit from using the Maestro middleware. The benefits may manifest themselves in terms of increased performance or flexibility.

More specifically, the HPC applications that are likely to see the largest benefits are the ones with the following usage characteristics.

- Creation and movement of large volumes of data through different workflow components, in particular in a time-critical manner. An increasing number of HPC applications experience that data movement represents more of a bottleneck than floating-point computations. This is especially true when large amounts of data are exchanged between different components of a workflow. This may be the case for visualisation and other forms of data processing.
- Management of data transformations across different memory architectures. Many HPC applications run on heterogeneous memory architectures and could benefit from a middleware that abstracts away the transformation from one architecture to the other.
- Management of layout transformations between different software components. This typically occurs where different technologies are used in different components of the workflow – such as Fortran and C/C++ – or when domain-decomposition techniques are applied to distribute workload.
- Intensive use of the global parallel filesystem for sharing data between software components. HPC applications have grown accustomed to storing temporary data on parallel file systems. This is simple but highly inefficient and it prohibits many applications to grow their workflows into exascale computing. They could greatly benefit from a middleware layer that offers both a more flexible and a more performant means to pass data during workflow execution.

Based on these general criteria, the Maestro project has chosen four applications to demonstrate these usage characteristics and to evaluate the viability of the Maestro

middleware. It is anticipated, however, that the range of applications that can potentially benefit from the Maestro technologies is much wider.

3. Software-release content structure

Each workflow prototype is provided in a repository containing the following parts.

Section	Directory and files	Description
Source code	src/<Version> src/README.md	Includes all necessary components to compile and run the given workloads. Ex. src/original src/maestro Note: During development it could be easier to employ tools such as Git to track different code versions. In such a case original, maestro … etc. could refer to for example the available Git branches.
Workload	workload/<Size/Name> workload/<Size/Name>/README.md	Required input data and configuration parameters to run the workload. Ex. workload/large workload/medium workload/small
Documentation	doc/	Documentation should explain all steps required to compile, deploy and run the given workloads.

Any deviation from this form will be described if needed.

As the MAESTRO system is developed the testing requirements are likely to become more developed. To fulfil these testing needs, we anticipate that further workloads will be added over the lifetime of the project.

4. Application Model Workflows

The following lists some notes and location of the released model workflows. The description is listed in this section by application partners.

4.1 Description Template

For each model workflow in this deliverable a description of the following items are included in this report.

Usage Scenarios	Number and Name as given by [D2.1]
Covered use cases	Number and Name as given by [D2.1] (Note that any number or combination of usage scenarios and use cases can be covered by a single workflow prototype)
Workflow owner	The project partner maintaining and porting the released workflow
Version	The software version that is snapshotted in the workflow prototype released as part of this deliverable (Note that this might be several entries for each software element)
Workload sizes	Listing the available workloads that are part of the workflow prototype released in this deliverable (e.g., Small, Medium, Large)[1]
Release location	Path to finding the released workflow prototype (e.g., URL)
Repository type	Type of code repository used for workflow prototype release (e.g., Git, SVN or Tarball)

Each model workflow can be accompanied by further descriptions and details. For example, a brief description can be added for missing components and how their functionality is achieved for the covered use cases.

The main items will be tabulated using the following template:

Usage Scenario	• <Number> <Name> • …

[1] It is difficult to clearly standardise workload size definitions across use cases as they highly depend on data, nodes, compute sizes, etc. As a result we view the workload sizes mentioned here as mere guidelines, where for example a small workload is expected to finish computation on a single node or desktop machine in a reasonable time frame and can therefore be used for testing and development. Meanwhile large workloads are rather intended for multi-node systems and are therefore more suitable for longer running performance tests. Generally, the workload documentation will further describe the needed compute size for a given time frame.

Covered use cases	• <Number> <Name> • …
Workflow owner	
Version	
Workload sizes	
Release location	
Repository type	Git/SVN/Tarball/…

4.2 Numerical Weather Prediction

4.2.1 Description

ECMWF have built a model workflow that represents the 'operational weather forecast' usage scenario from the deliverable document on application requirements [maestro_d21]. It captures the essence of the I/O-related challenges being tackled within the Maestro project, whilst it avoids running the large number of tasks required to coordinate a full forecast. It simplifies the workflow to its three essential components.

- *Data generation* In an actual operational workflow, forecast data is routed through a number of dedicated I/O nodes, each running a predefined number of I/O producer tasks. The I/O tasks output data via the domain-specific I/O-library, called `multio`. In the model workflow, we create a tool on top of the `multio` library to generate data.
- *Metadata manipulation* A key element of the actual workflow is that each consumer needs to read data produced by every I/O node. They have to read 'across' the producer data. This is achieved solely by metadata manipulation, so it is important that the generated data have the same metadata as the actual data, even though data itself is scientifically meaningless.
- *Data transfer* The transfer between I/O-node producers and the product-generation consumers is the bottleneck in the current operation workflow, so it is naturally essential that they are well represented in the model workflow. This is achieved by using the same consumer task in the model workflow as in the actual workflow.

Figure 1 shows both the operational workflow and the model workflow, as well as the steps taken to derive the model workflow form the operational workflow. The model workflow is executed by kronos, a workload manager that is also used for benchmarking at ECMWF.

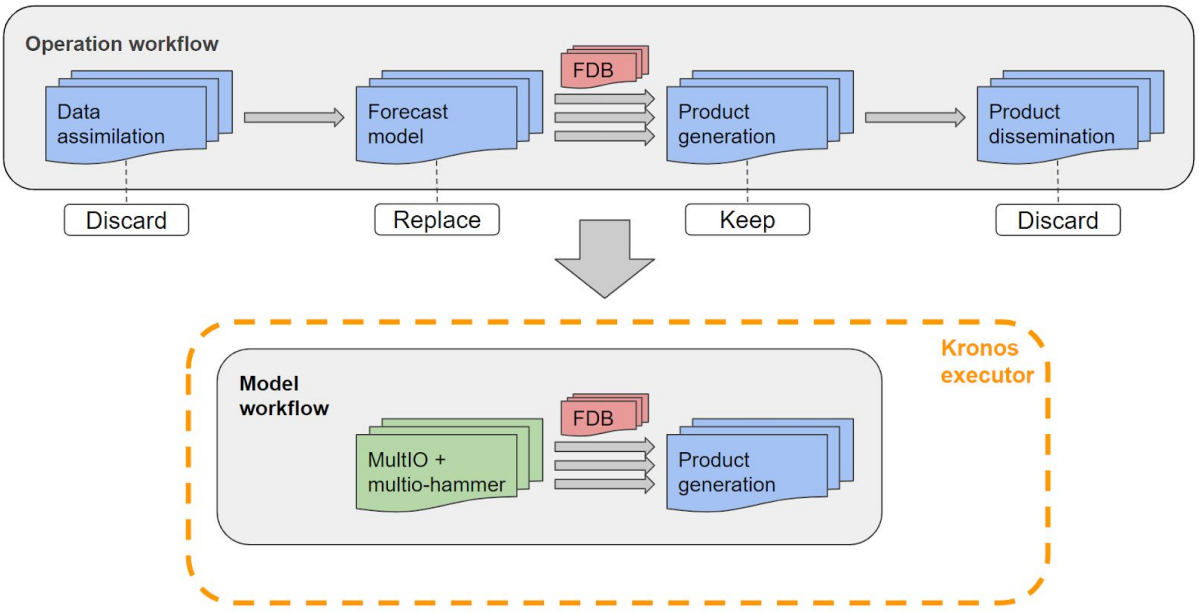

Figure 1: High-level overview of both the operational and model workflow of forecast run at ECMWF

The following are the main software components in the model workflow.

- `kronos` Amongst other things, `kronos` [kronos] is an event-driven workload manager. Given a (configurable) schedule, it acts as a meta-scheduler, submitting tasks to the system scheduler. These tasks can notify `kronos` of events (startup, completion, error and the completion of intermediate steps) that will in turn trigger further submission of jobs. This component acts as a substitute for the Centre's large-scale forecast workflow manager, ecflow.
- `fdb` The Fields DataBase (FDB) [fdb] is the library and data service used to handle, transfer, store and index meteorological objects in the forecast pipeline [fdb_17, fdb_19]. It provides the I/O layer used by other components. This is the component that we intend to bypass (at least for the time-critical path) by making use of the Maestro middleware.
- `multio-hammer` The real forecast model is large, uses significant computational resources, is complex to configure and needs to be carefully tuned to specific HPC systems. For the purposes of the Maestro project the primary area of interest is the output from the model via its I/O servers. `multio-hammer` -- a tool built on the `multio` library [cit_multio] used in operations -- simulates this output process. It takes sample data as an input and permutes it through the range of metadata required for a forecast run, thus outputting dummy data with various sets of metadata. This data is output through the full I/O stack used in normal operations. In this way it simulates the forecast model from an I/O perspective.
- `pgen` The product-generation engine at ECMWF is used to transform forecast output data into the form requested by customers according to requirement definitions. One `pgen` task is launched per completed forecast step, and it reads output data from across all ensemble members. The pgen tasks included in this workflow perform tasks very close to the real post-processing, although on synthetic data output from `multio-hammer`.

Details on how to configure and run this workflow are included in the top-level `README` in the release repository. The schedule generator is configured according to the number of forecast steps and forecast ensemble members to be emulated. This allows the workflow to be scaled from very small to (at least) the full size of the operational workload.

The following table summarises which usage scenario and use cases this model workflow represents and what software packages are released as part of the model workflow.

Usage Scenario	US1.1 Operational weather forecast
Covered use cases	• UC1.1 Access semantically-related datasets • UC1.2 High-velocity production of meteorological objects • UC1.6 Sustained high-volume production of meteorological objects • UC1.7 Consumer applications request sets of objects
Workflow owner	ECMWF
Version	• kronos: 0.7.0 • eccodes: 2.18.0 • eckit: 1.12.2 • metkit: 1.5.8 • multio: 1.2.0 • fdb: 5.6.4 • atlas: 0.21.0 • mir: 1.5.2 • pgen: 1.7.1
Workload sizes	Variable and configurable. Small to large.
Release location	https://gitlab.version.fz-juelich.de/maestro/ecmwf
Repository type	Git

4.2.2 Workflow Execution and Workload Configuration

Figure 1 above also depicts the simplifications in the model workflow compared with the actual workflow. The execution of the model workflow consists of the following broad steps.

- `multio-hammer` generates mock data and passes the generated data to `fdb`, a domain-specific database for meteorological data.
- `multio-hammer` also sends a notification to `kronos` once it has produced all data for a given forecast step. It thus fulfils its role as substitute for an I/O-server task of one ensemble member in a forecast run.
- `pgen` can start post-processing as soon as data for a given forecast step are available from all multio-hammer instances. There is one pgen task per forecast step and it is the responsibility of `kronos` to start a pgen task once it has received all notifications for a given time step. When a `pgen` task is completed, the generated product is written to disk and the dissemination step is omitted.

There are additional domain-specific libraries developed at ECMWF that are required for the successful execution of the model workflow. These can be viewed as dependencies for `multio-hammer`, `pgen` and/or `fdb`. A full list is given in the section on licensing information.

Specifying the workload

At the highest level, the workload depends on three factors: the number of forecast steps, the number of ensemble members and a requirements description for dissemination. The number of forecast steps and ensemble members are configurable at the point of kronos generating the schedule. For example, the line

```
python generate_kschedule.py --nohres --nocf --steps=12 6
```

will generate a schedule of six ensemble members, each running for twelve steps. These two variables are thus easily configurable and can be used to increase the workload.

Figure 2 shows how `kronos` manages data dependencies between producers and consumers. There is one consumer job for every forecast step. `kronos` can only trigger the consumer once the data for that forecast step has been made available by all producers. In this example, there are six producers running for twelve forecast steps but `kronos` is also used to execute workflows up to the full sizes of operational runs. With the model workflow created for this project, it would be possible to go beyond those sizes if there is a need to stress the Maestro middleware.

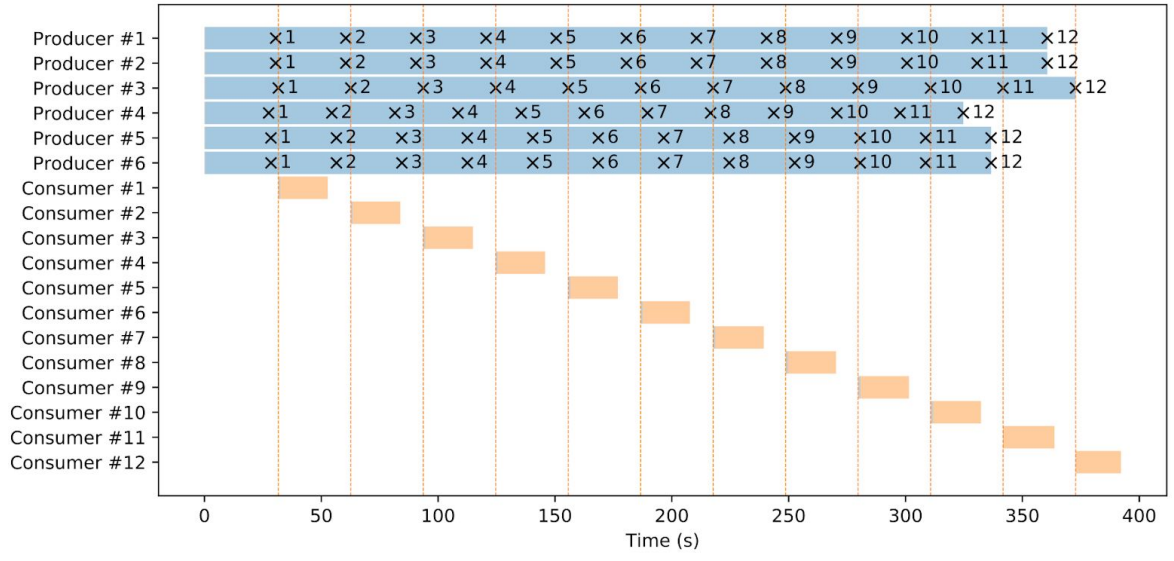

Figure 2: Example to show how Kronos manages data dependencies between producer and consumers. Each consumer is only triggered once all its data dependencies are met.

The overall workload also to a large extent depends on the requirements description. It is essentially a list of meteorological fields for which to carry out post-processing. This final release of the model workflows includes three dissemination files with

increasing number of fields: `small.diss`, `medium.diss` and `large.diss`. When run with 50 `multio-hammer` producers, which is the number of ensemble members in the operational workflow, the largest dissemination file will require about 48 GiB of the date to be produced and read per step. It will also create about 15GiB of the post-process output that it will also write to disk. Given that the number of steps can be increased indefinitely in the model workflow, this setup is clearly suitable to demonstrate exascale challenges in an HPC I/O stack.

Various configurations have been tested on workstations as well as two different HPC architectures available at ECMWF, a Cray XC40 machine with PBS submission system and 34-node Dell Linux cluster with the Slurm submission system. The Cray XC40 architecture is the same as used for the operational runs [ec-xc40]. The workflow has been also tested on the Sage prototype at the Jülich Supercomputing Centre during the internal review of the deliverable.

The top-level `README` file in the release repository includes detailed instructions on how to build, configure and execute the model workflow. In general, there are a few necessary and a few optional steps to be able to execute the workflow. The necessary steps are:

- A basic configuration is required to specify various paths, such as the output and read_cache folders, as well as setting the scheduling system and the number of processing elements per node. The configuration file could look something like this:

```
{
  "coordinator_binary": "",
  "job_class": "slurm",
  "job_dir": "<working-dir>/kronos-run/output",
  "job_dir_shared": "<working-dir>/kronos-run/shared",
  "procs_per_node": 4,
  "read_cache": "<working-dir>/kronos-run/read_cache"
}
```

- A bash script for the data producers needs to be configured with the correct path information to executables and folders.
- A similar python template for the data consumers needs to be configured with the correct path information to executables and folders.
- Python templates may be used to define scheduling parameters for the scripts.

Furthermore, it is possible to manipulate the command-line options to the producer and consumer executables: `multio-hammer` and `pgen-run`.

4.2.3 Maestro-enabled version and evaluation metrics

During the existing workflow, the `multio` library is responsible for routing meteorological data to multiple datasinks. In operational use, it puts data into `fdb`, which in turn persists it on the global parallel file system. The product-generation task `pgen` is triggered when all its data-dependencies are met. Then a `pgen` tool requests the dependent data from `fdb`.

The Maestro core middleware would constitute an alternative to parts of the functionality currently carried out by `fdb`. Making the model workflow Maestro-enabled thus requires three major developments.

- The `multio` library is being re-designed to accept a configurable, extendable and composable list of actions. One action will be the output (or 'sink') action that is responsible for passing meteorological data to a storage system, database or, indeed, a data-management system. A backend to `multio` to support output to the Maestro core middleware is being developed. The `multio` library will only directly use (i.e. link against) the Maestro core middleware (e.g. `libmaestro.so`). What technologies to use from the low-level middleware may be configurable at compile/run time, but the low-level APIs will not be exposed to `multio`.
- A software component that is able to request data from the Maestro middleware based on scientifically meaningful metadata needs to be developed. This component will then need to pass the data onto `pgen` for post-processing so it will in practice act as a backend to `pgen`. This component will make use of the Maestro core middleware's subscribe and select functionality, which form part of their events-handling architecture [maestro_d33].
- Support for the Maestro middleware's data-management component requires additional developments for both `multio` and `pgen`. In particular, the concept of the pool manager needs to be incorporated into the design of how `multio` and `pgen` conduct their work. This either requires introducing a new software component for the pool manager or deciding which instance of `multio` or `pgen` will act as the pool manager. Elements of the workflow have been made Maestro-enabled already by using `multio-hammer` with a Maestro backend and a simple pool manager used in the testing framework of the Maestro core middleware.

The purpose of the model workflows is to make the effort required for Maestro modifications achievable within the project's timeframe. It is important to emphasise that these modifications are relatively self-contained both thanks to the effort to derive the model workflows and thanks to the effort to incorporate application requirements into the co-design of the Maestro core middleware API.

Evaluation metrics and performance baselines

There are various different performance metrics that can be of interest for any given workflow. In order to evaluate the impact of the Maestro middleware on operational forecast runs, we have selected three metrics to watch closely:

- amount of data written to the globally visible parallel filesystem that the product-generation step needs to read to be able to carry out its task;
- the model workflow's overall execution time for a given configuration and for a given system;
- sensitivity of the workflow execution time to 'misbehaving' components, such as producers lagging behind or speeding ahead throwing large volumes of data at the middleware.

As the baseline for the amount of data written to and read from the global parallel filesystem, we take the workload requirements `large.diss`, which is the largest we provide. Running this with 50 producers, mimicking an ensemble run of 50 members, will produce approximately 48GiB of data each step that needs to be read by the `pgen` job responsible for that step. In all the following examples, each `pgen` job executes on a separate node, running 16 tasks: one 'broker', two 'writers' and 13 'workers'. Among other things, the workers between them read all the data required by the `pgen` job. The beginning of the log for a summary of the workers reads, for example, as

```
000 2020-07-22 17:30:02 (I) ====================================
000 2020-07-22 17:30:02 (I) SUMMARY: 13 workers
000 2020-07-22 17:30:02 (I) ====================================
000 2020-07-22 17:30:02 (I) Requirement batches : 61,550
000 2020-07-22 17:30:02 (I) Total requirements : 512,200
000 2020-07-22 17:30:02 (I) No. of target files created : 390
000 2020-07-22 17:30:02 (I) No. of target files opened : 390
000 2020-07-22 17:30:02 (I) Total fields read : 61,550
000 2020-07-22 17:30:02 (I) Size fields read : 50,614,349,950 (47.1383 Gbytes)
000 2020-07-22 17:30:02 (I) Fields interpolated : 512,200
000 2020-07-22 17:30:02 (I) Total fields written : 512,200
000 2020-07-22 17:30:02 (I) Size fields written : 17,108,011,100 (15.9331 Gbytes)
```

This is a simple statistics about the amount of the data to be consumed and produced by the `pgen` job. The Maestro-enabled version is, naturally, expected to produce the same amount. However, it will consume all the data from the Maestro middleware and it is expected that the Maestro middleware will only have to persist only a fraction of that to the globally-visible parallel filesystem.

The gain from needing to read much less from the parallel filesystem should translate into faster execution time for the model workflow for a given configuration and system setup. We have run the model workflow with a large number of different configurations. The table below shows the execution times for a few of those runs. Again, we only show examples for the large.diss requirements workload.

System	Producers	Consumers	Nodes	Elapsed time (s)
Dell	6	12	16	1113.08
Dell	50	40	24	1737.07
XC40	50	1	18	1230.47
XC40	50	6	18	1543.56
XC40	50	10	18	1939.88

As for the metrics for robustness, one can take certain ‘ideal situations’ as baselines and measure deviations from it. For example, a producer speeding ahead will not cause a delay in the overall workflow execution in an ideal situation. Similarly, a producer lagging behind should only delay the workflow’s overall execution by the

amount of time it itself was delayed. Measuring the deviation from these baselines would give a basic indication of the middleware's robustness.

4.2.4 Licensing Information

Component	License
kronos	Apache-2.0, available at ECMWF GitHub
atlas	Apache-2.0, available at ECMWF GitHub
ecbuild	Apache-2.0, available at ECMWF GitHub
eckit	Apache-2.0, available at ECMWF GitHub
fdb	Apache-2.0, available at ECMWF GitHub
metkit	Apache-2.0, available at ECMWF GitHub
mir	Proprietary
multio	Apache-2.0, available at ECMWF GitHub
pgen	Proprietary

All but the two proprietary packages are freely available at ECMWF's GitHub account [ec-gh].

4.3 TerrSysMP

The Terrestrial System Modeling Platform (TSMP or TerrSysMP) is a scale-consistent, highly modular, massively parallel, fully integrated soil-vegetation-atmosphere modeling system. TSMP implements a physically-based representation of transport processes of water, energy and momentum across scales down to sub-km resolution including explicit feedback between groundwater, the land surface and atmosphere with a focus on the terrestrial hydrological and energy cycles. TSMP currently comprises three component models: The Consortium for Small-scale Modeling (COSMO) atmospheric model, the Community Land Model (CLM), and the hydrologic model ParFlow coupled with OASIS3-MCT coupler and data assimilation capabilities through the Parallel Data Assimilation Framework (PDAF-D) (See Figure 3).

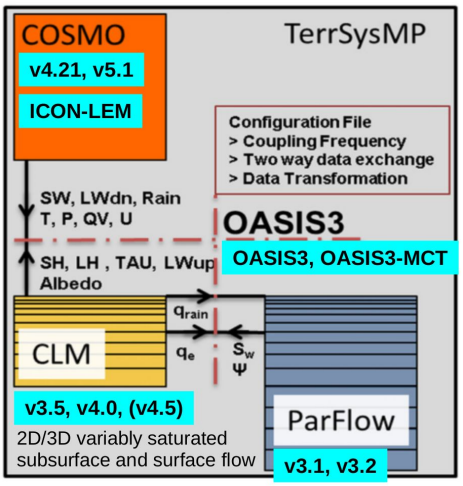

Figure 3: Interaction of Model Components using the OASIS3-MCT coupler in TerrSysMP

TSMP is extensively used for idealized and real data process and sensitivity studies in water cycle research, for climate change simulations, data assimilation studies including reanalyses, as well as experimental real time forecasting and monitoring simulations, ranging from individual catchments to continental model domains.

4.3.1 Description

As described above, TerrySysMP [tsmp] couples three model components COSMO, CLM and ParFlow. The three components interact with each other through a coupler named OASIS3-MCT. Both COSMO and ParFlow couple to CLM, but not to each other. As a result COSMO (which has a more restrictive license) can be omitted while maintaining the characteristic behaviour relevant to the project. Within the Maestro project, the relevant changes will be implemented as part of the coupling between ParFlow and CLM. It is anticipated that the improvements to the two-way coupling can be applied to the coupling of the three models.

In this workflow prototype we have additionally provided a data assimilation tool called PDAF-D (Parallel Data Assimilation Framework), which is used to ingest the small workload provided.

Usage Scenario	• US4.1 Global earth modelling
Covered use cases	• UC4.1 Exchange 2D fields between climate models • UC4.2 Dynamic load balancing of coupled models
Workflow owner	Juelich
Version	• TerrSysMP: 1.0 • ParFlow: 3.1 • Oasis3-MCT: oasis3-mct_121022 • CLM: 3.5 • PDAF-D: 1.10

Workload sizes	• Small; can run on a single node or a desktop system. • Larger workloads, which can scale to 100 of nodes are available and will be added to the repository as part of D2.4
Release location	https://gitlab.version.fz-juelich.de/maestro/terrsysmp
Repository type	Git

4.3.2 Description

- The two main components of the simplified workflow are the CLM component and the Parflow component. CLM and Parflow are the simulation components that produce and consume data while the data is transferred between these two components through the Oasis3-MCT coupler. The data is produced and consumed by CLM and Parflow in a closed loop model wherein the output data produced by CLM is used as input by Parflow to produce the next set of input data for CLM. Due to direct dependencies, the transfers sequence is blocking and serial.
- The Oasis3-MCT coupler and the CLM component are written in Fortran while Parflow is written in C. The data transferred between CLM and Parflow is in the form of two-dimensional arrays. Fortran has a column major representation for matrices while C has a row major representation for matrices. This means that a transpose operation is needed while transferring the data between the two components.
- The configuration of the components to use is done at build time using the `./build_tsmp.ksh` configure script where one can specify the models to be used during the different runs. For example, the following command will generate a debug build for a CLM-Parflow model which can then be setup using a similar `./setup_tsmp.ksh` command.

```
./build_tsmp.ksh -c clm-pfl -o “O0”
```

Specifying the workload

For our experiments with TerrSysMP, we will consider a down-scaled version of the test case described in Kurtz et al. (2016) [kurz2016] which deals with a relatively simple land surface-subsurface problem. With this forward model we will perform experiments for the assimilation of soil moisture data including parameter estimation. The model domain for our test case has a size of 1000 x 1000 x 12 meters with a horizontal discretization of 20 meters and a variable vertical discretization resulting in 50 x 50 x 12 grid cells for the subsurface domain. The land surface is covered by three land use types (forest, crop and grassland), which are shown in Figure 4.

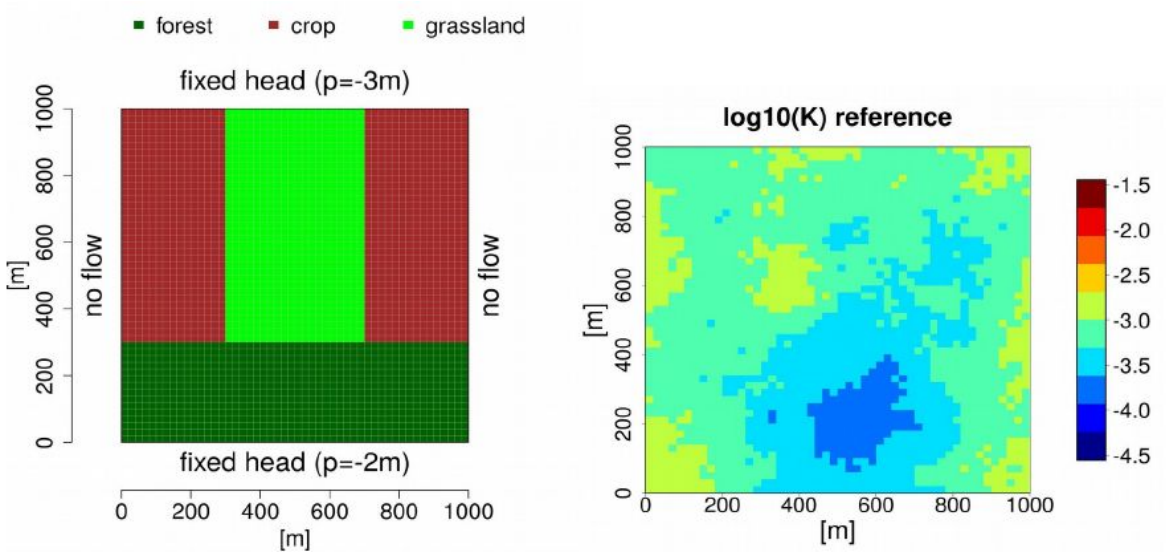

Figure 4: Boundary conditions and land used types of the model domain (left) and reference log10(K) field (right).

4.3.3 Maestro-Enabled Version

- A major task while porting TerrSysMP to Maestro would be to replace the Oasis3-MCT coupler component with Maestro. We expect the rest of the program to remain mainly untouched. This would involve replacing Oasis3-MCT coupler send/receive calls with Maestro's offer and demand operations.
- We expect the following steps are needed to port the TerrSysMP workflow to Maestro:
    - Since the dependency is only on the immediately generated data, Oasis3-MCT coupler currently does not uniquely label the generated data. This is not compatible with Maestro where each CDO should have a unique name. So, the first task would be to ensure that the information to uniquely identify the transferred data is available at the call site.
    - Call to Oasis3-MCT will have to be replaced by a function which can take care of generating a CDO and offering it to the Maestro pool manager, following which, it has to wait for the CDO it will receive from the other process.
    - A point to note here is that we must already know the name of the CDO we are waiting for, and this requires us to come up with a naming scheme for the CDOs or an equivalent mechanism within Maestro.
- The workflow mainly uses the Maestro core component to manage the CDOs and could use high-level Maestro middleware components such as the workflow translators and workflow models to ensure correctness and improved workflow execution.

4.3.4 Evaluation Metrics

In this use case, Maestro is supposed to replace the Oasis3-MCT Coupler. The main workload of the coupler is to transfer the data between two processes. Furthermore,

the modifications in the original code are tailored for the use with the coupler. As such, the workload is primarily bottle-necked by the hardware and not the software. Therefore, over the course of replacing the coupler with Maestro, we do not expect performance gains. Instead, we want to evaluate the viability of using a general purpose solution like Maestro to replace application specific couplers which will then allow users to do many more tasks with the data instead of just transferring it to another application and use more applications.

In summary, the evaluation criteria for the Maestro-enabled version of TerrSysMP are as follows:

- No performance regression. The table below provides baseline values to compare with.

System	Sage Prototype
#Nodes	1
Execution time (s)	**2435**

- Replacement of a tailor-made coupler with a general purpose Maestro core.
- Decoupling of code of Parflow and CLM.

4.3.5 Licensing Information

Component	**License**
Parflow	GNU Lesser General Public License
CLM	GPL
Oasis3-MCT	GNU Lesser General Public License
PDAF-D	GNU Lesser General Public License
TerrSysMP	MIT

4.4 Computational Fluid Dynamics and In situ Analysis (CEA)

4.4.1 Description

The CEA workflow that will be eventually provided for MAESTRO is representative of a typical chaining scenario of two simulation codes as depicted in Figure 5.

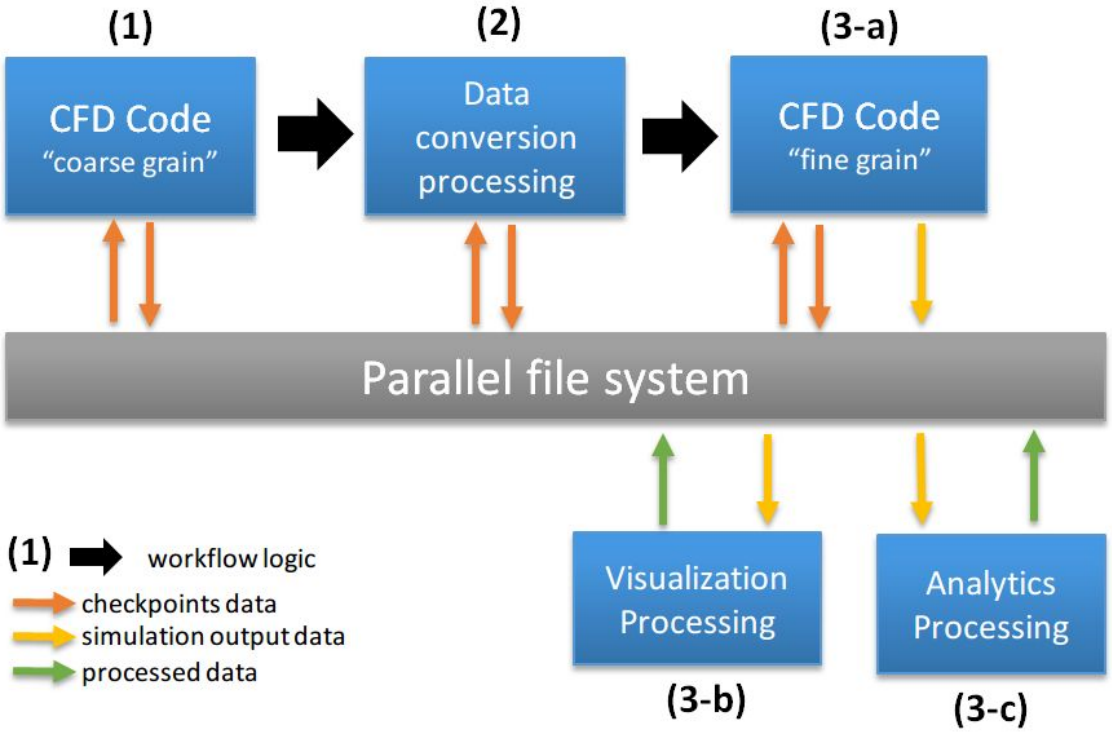

Figure 5: Typical chaining scenario of two simulation codes

Chaining of simulation codes is often used to model different physics and/or scales, with the output of the first code being used as the input of the second with an intermediate data transformation/preparation step and a final post-processing step.

Due to the nature of CEA's activities, our actual simulation codes cannot be publicly disseminated. As a substitute we will use the open source proxy application Hydro, a 2D CFD simulation code, to model our two simulation codes.

Since some development is required to chain Hydro with itself (based on a restart with a different domain decomposition), the workflow prototype provided as part of the present document is a pipeline consisting of one run of Hydro, followed by a post-processing step based on ParaView, a high performance visualization application. This is shown in Figure 6.

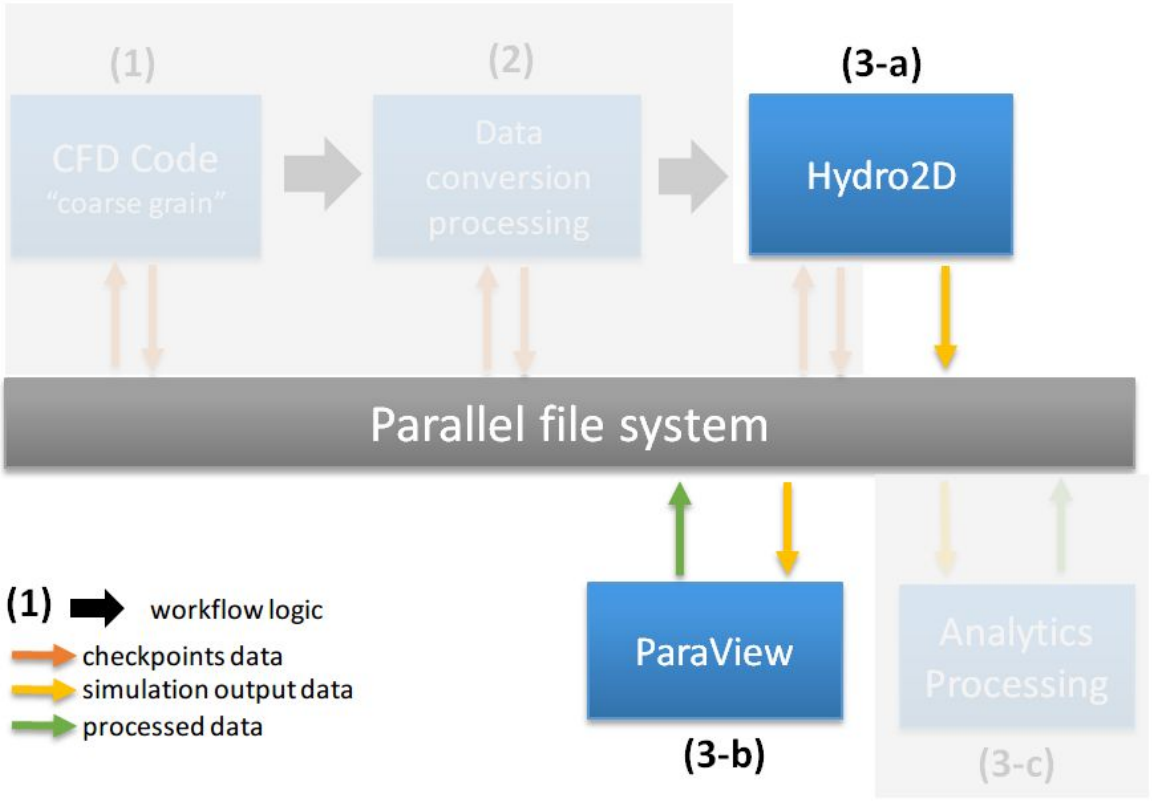

Figure 6: The Hydro workflow

A final step of the pipeline not represented on the above diagram consists in running FFmpeg, a popular open source software for creating and converting audio video files, on ParaView image outputs to produce a movie of the simulation.

The final workflow prototype will be provided as part of deliverable D2.4 Final Workflow Models.

Usage Scenario	• US2.1 Multi-physics simulation pipeline (partly)
Covered use cases	• UC2.1 Applications write/read data collections to/from MAESTRO • UC2.2 "Streaming" data records between producers and consumers • UC2.4 Simulation checkpoints spooling
Workflow owner	CEA

Version	• Hydro (develop branch) • ParaView 5.6.0 • FFmpeg • Hercule 2.4.0_rc7
Workload sizes	• Small; can run on a single node or a desktop system. • Normal; runs on 2 nodes on a supercomputer using SLURM scheduler • Large; can scale to 100 nodes. It also uses a SLURM scheduler
Release location	https://gitlab.version.fz-juelich.de/maestro/hydro
Repository type	Git

4.4.2 Workflow Execution and Workload Description

4.4.2.1 Workflow Description

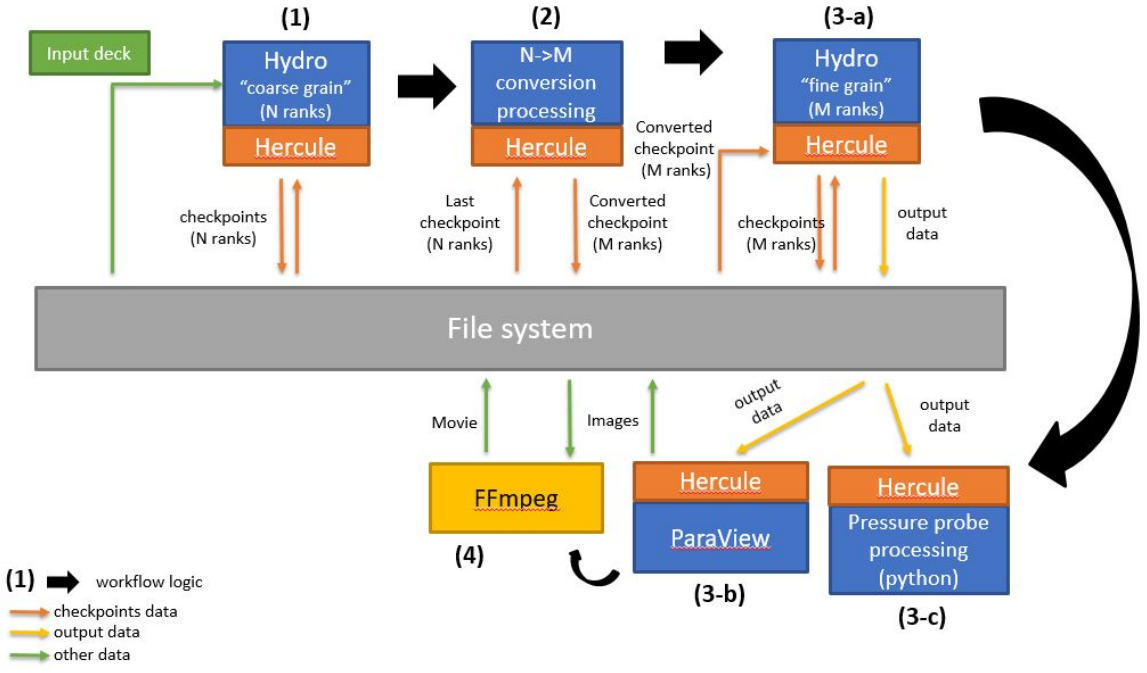

Figure 7: Hercule version of the workflow

Figure 7 depicts the model workflow. More details about the different components of are given hereafter:

- **Hercule**: is a CEA in-house IO library. Filesystem based, Hercule can be used in very large workflows to store and visualize data over many nodes. In

our workflow, Hercule is used to store simulation output and checkpoint data. During the execution of the workflow, output data is post processed by 2 scripts (as described further below).

- **Hydro**: is a full hydrodynamic code used for benchmarking purposes implementing a 2D Eulerian scheme using a Godunov method. Multiple versions have been developed and for the purpose of this workflow model we use the MPI version developed in C.

  The code is launched through SLURM scheduler like this (Hydro only):

```
$ srun -n <nprocs> hydro -i inputs.nml
```

  where `<nprocs>` is the number of MPI tasks and `inputs.nml` is the Hydro configuration file.

  Every step, the data of the simulation is stored through the Hercule library which will be post processed later on. At the same time, checkpoints are stored at every <n_checkpoint> step (where <n_checkpoint> can be modified in the workflow configuration file).

- **ParaView**: is an open source high performance visualization and graphics processing application. In our workflow, we use the headless batch processing capabilities of ParaView combined with a Hercule Reader to read Hercule databases and produce a set of JPEG images (one per timestep) using predefined camera settings with the final goal of producing a movie of the simulation.

  This processing uses a specialized python distribution provided as part of ParaView installation (`pvpython`) that includes ParaView python library for dynamically defining camera settings, output parameters display, color palette, etc. A `pvpython` script (`paraview_run_hercule.py`) has been written to process the Hercule database (and produce one image per step). This script is used by another python script (`process_all_hercule.py`) which spawns a pool of worker processes where each worker runs the `paraview_run_hercule.py` script. Each worker is focusing on some steps and ignores other steps that are being processed by the other workers.

  This ParaView step is launched through the SLURM scheduler with a command line similar to this one:

```
$ srun -N 1 -n 1 -c <ncores> process_all_hercule.py \
paraview_run_hercule.py <execution_path>/depouillement/hercule <nWorkers>
```

  where `<ncores>` is the number cores assigned to the single task launched (only one instance of `process_all_hercule.py` is run), `<nWorkers>` is the number of workers to spawn for processing the Hercule database and

<execution_path> is the path to the directory where the simulation is running (which can be edited in the workflow configuration file).

- **HerculeManip**: is a python package for Hercule database exploration. In this workload, it is used to extract the pressure on one point for each time step. When it is done, we use the `matplotlib` library to generate a graph of the pressure over the time.

  The pressure extracting step is launch through a command line similar to this one :

```
$ start_simple_compute.sh simple_compute.py
<execution_path>/depouillement/hercule
```

- **FFmpeg**: is open source software for creating and converting audio and video files. It is run as the final stage of our workflow reading all JPEG files produced by ParaView and building a MP4 video file of the simulation.

  The command line used to run FFmpeg is similar to this one:

```
$ ffmpeg -i image_%04d.jpg <ffmpeg-options> hydro_movie.mp4
```

4.4.2.2 Workflow execution and configuration

There are 3 different workloads that can be executed :

- **Small workload** : This little workload can run on a single node. The Hydro simulation is computed once and produces a set of VTK files. Then the ParaView script is called to process the VTK files and generates one image per step of the simulation. Finally, ffmpeg is called to process the images and produce a movie. To execute the workflow:

```
$ ./run.sh
```

- **Normal workload** : This workload is running over 2 nodes on a supercomputer using the SLURM scheduler. The Hydro simulation and the ParaView script are run simultaneously. The post-processing is using the VTK files as soon as they are produced by the simulation. Then, ffmpeg is called to process the images and produce a movie. To execute the workflow in **batch mode**:

```
$ sbatch ./job.sh
```

To execute the workflow in **interactive mode**:

```
$ ./run_salloc.sh
```

- **Large workload** : This workload can scale to 100 nodes. The Hydro simulation, the ParaView script and the pressure extraction are run simultaneously. The 2 post-processing steps are exploring the Hercule database to fetch the simulation data. The slurm script `job.sh` can be edited to adjust configuration properties such as the number of nodes, the number of cores per node or the execution path. To execute the workflow in **batch mode**:

```
$ sbatch ./job.sh
```

To execute the workflow in **interactive mode**:

```
$ ./run_salloc.sh
```

4.4.3 Maestro-Enabled Version and Evaluation Metrics

Using MAESTRO with this workflow will consist in replacing the file system « bus » used to exchange data between workflow applications (central grey box on the workflow Figure 7) by the MAESTRO middleware as shown in Figure 8.

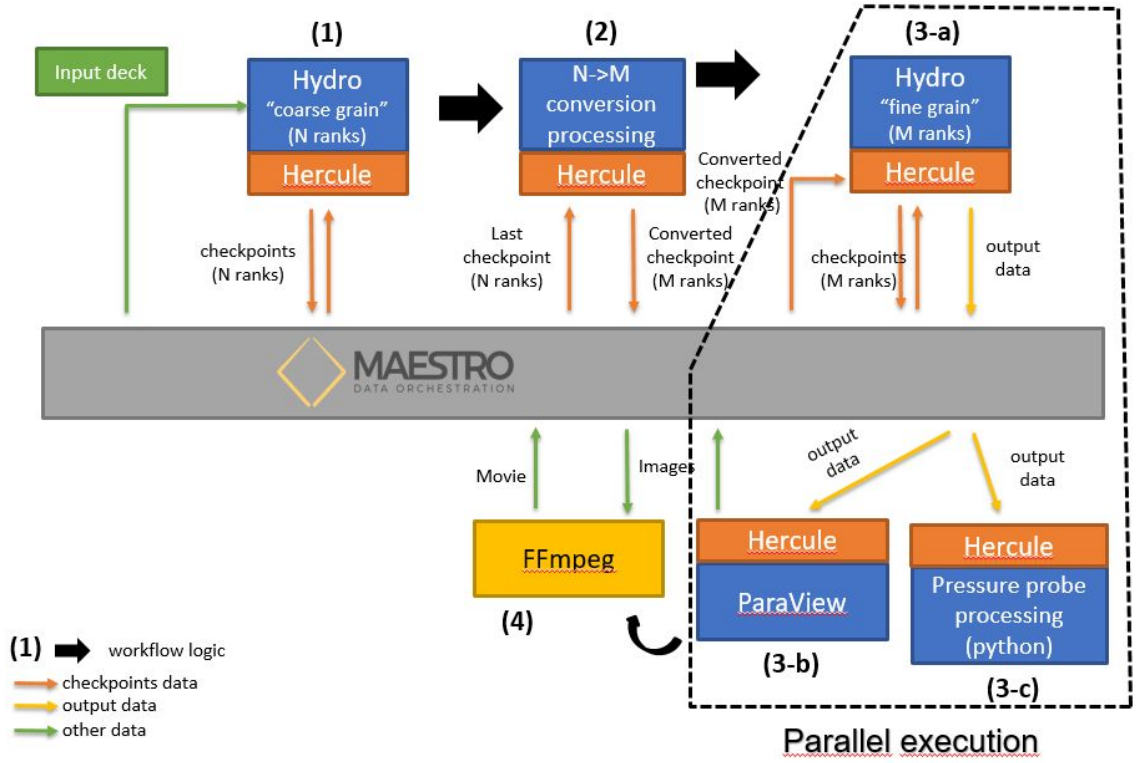

Figure 8: Maestro-enabled version of the workflow

MAESTRO will then be responsible for:

- managing all data transfers (represented by arrows),
- persisting data as specified by the user (at least the final data).

In addition, we expect MAESTRO to handle data transfers appropriately to allow for running the applications of step 3 in parallel (ie, streaming/synchronizing data between producing and consuming applications). In summary, the evaluation criteria for the Maestro-enabled version of CEA are as follows:

- No performance regression. The table below provides baseline values to compare with.

System	Inti	Inti	Inti
Nodes (Haswell)	1	5	10
Total cores	32	180	320
Time (sec)	150.031	59.654	49.11
Cycles (MC/s)	31.41	104.64	159.18

- Streaming data between producing and consuming applications
- Persisting the final data for checkpoint/restart

4.4.4 Licensing Information

Component	License
Hydro	CeCILL v2.0 Licence (GPL compatible)
ParaView	BSD
FFmpeg	GNU Lesser General Public License
Hercule	CeCILL v2.0 Licence (GPL compatible)
Hercule Reader (ParaView)	CeCILL v2.0 Licence (GPL compatible)

4.5 Sirius

SIRIUS is a domain specific library for electronic structure codes developed at ETHZ/CSCS. It is a collection of classes that abstracts away the different building blocks of pseudopotential plane-wave and full-potential linearized augmented plane-wave codes. SIRIUS can extend the legacy Fortran codes with the API calls to

a domain-specific library which runs on GPU and other novel architectures. The code is written in C++11 with MPI, OpenMP and CUDA programming models.

4.5.1 Description

The DFT Loop mini-app is part of the SIRIUS package. This mini-app is used to benchmark all components of the DFT self consistency cycle – diagonalization of the Kohn-Sham Hamiltonian, charge density construction, mixing and generation of the effective potential – using a pseudopotential or full-potential method. DFT Loop is an interesting use-case for the Maestro middleware as significant data movements are involved and manually managed to properly use the GPU memory. When the data size cannot fit into the GPU memory, allocation/deallocation is manually done by blocks, meaning that the memory management is fully controlled by the developer.

In this context, the benefit Maestro could provide is two-fold: first, Maestro could optimize the movement of data between host memory (CPU) and HBM of the GPU, based on the communication requirements (targeted memory, timings, transport layer). Secondly, our middleware could insure the reusability of the allocated memory space. All of this would take advantage of the memory abstraction offered by Maestro for a better portability and extendibility to emerging memory and storage technologies.

4.5.1.1 Building and Running the DFT Mini-App

Requirements

- GCC 7.x.x (incompatibility with GCC 8+ so far)
- CUDA 10.x.x
- CMake 3.13.x and later
- OpenMPI
- OpenBLAS

Cray-specific prerequisites

On Cray systems, CMake 3.10.2 (and probably below) cannot find BLAS and LAPACK that should be made available by the *cray-libsci* module. A more recent version of CMake is necessary. CMake 3.13.4, 3.14.4 and 3.14.5 have been successfully tested. In addition, *CC* and *CXX* environment variables must be set to the corresponding Cray wrappers.

```
module load craype-accel-nvidia60
module load CMake/3.14.5
export CC=cc
export CXX=CC
```

Build SIRIUS and all the dependencies

First of all, it is necessary to take a look at the beginning of the *src/original/build.sh* file to properly set the GPU model for the target

architecture. The *nvidia-smi* command can help you for that. After editing the *GPU_MODEL* variable, the building process can be launched:

```
cd src/original
# Edit build.sh to get the GPU model
sh build.sh
source ./set_envvars.sh
```

This script creates two directories, *build* and *install*. Then, it downloads and compiles all the dependencies required by SIRIUS into the *build* folder. Header files, binaries and libraries are then moved into the *install* directory. The same process is finally applied to SIRIUS. A bash script *set_envvars.sh* is also generated during the build process. This script has to be "sourced" to properly set the *$LD_LIBRARY_PATH* environment variable.

Run SIRIUS example

By default, only one MPI process per node can be launched.

```
cd workload/<use-case>/
srun  -N<#nodes>  -n<#nprocs>  ../../src/original/install/bin/sirius.scf
--input=sirius.json
```

The problem size can be changed while varying *gk_cutoff* and *pw_cutoff* in the *sirius.json* file. The second value has to be at least twice the value of the first parameter.

4.5.2 Workflow Execution and Workload Configuration

The mini-app is a parallel and distributed code. Figure 9 below, shows the offloading step(s) on one node from one or more processes. Chunks of data are sequentially moved to the GPU memory for computation. Data is sent back to the host memory then the next step can start. The *mdarray* class is the data abstraction used by SIRIUS. Within this class, allocation and deallocation are made as well as data movement to/from the GPU memory. This memory management process also goes through the *memory_pool* class. As a large number of allocation and deallocation can impact the overall performance, this C++ class handles a group of memory pools that are allocated once and reused multiple times to mitigate the memory allocation bottleneck.

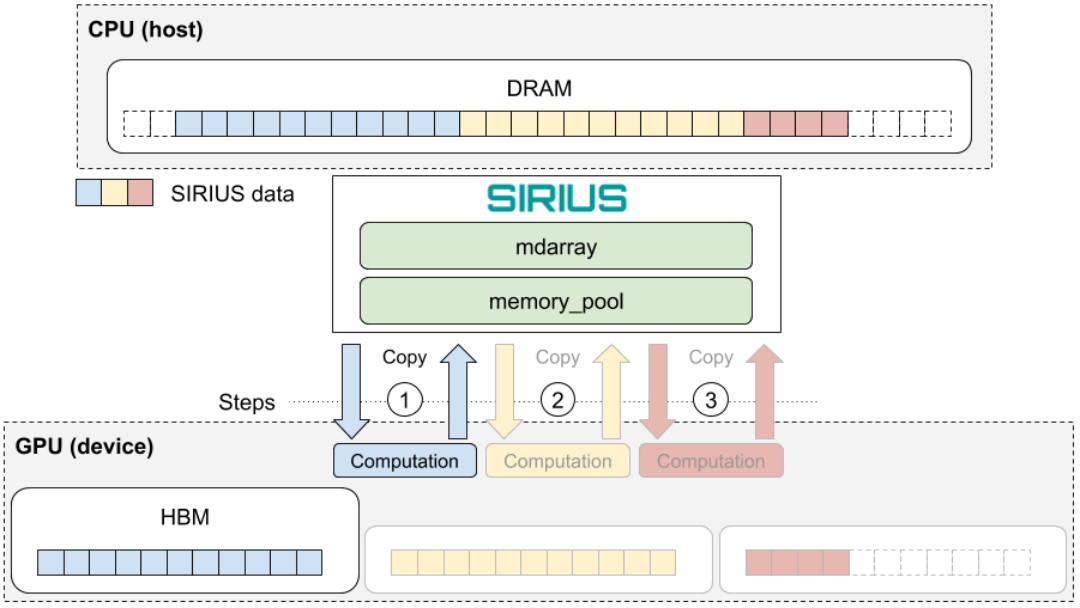

Figure 9: The offloading step(s) on one node from one or more processes.

4.5.3 Maestro-Enabled Version and Evaluation Metrics

The Maestro-enabled version of our workflow will impact the low-level layer of the SIRIUS library. Maestro will be a substitute of the memory management and the memory_pool classes. The use of CDOs (Core Data Object) as described in WP3 deliverables to encapsulate data is not mandatory. Likewise, as data movement happens within a single application, there is no need for Maestro to take and give ownership of the data. However, the memory abstraction layer developed in the Maestro middleware will be used to express data movement. Figure 10 below depicts this Maestro-enabled version of SIRIUS.

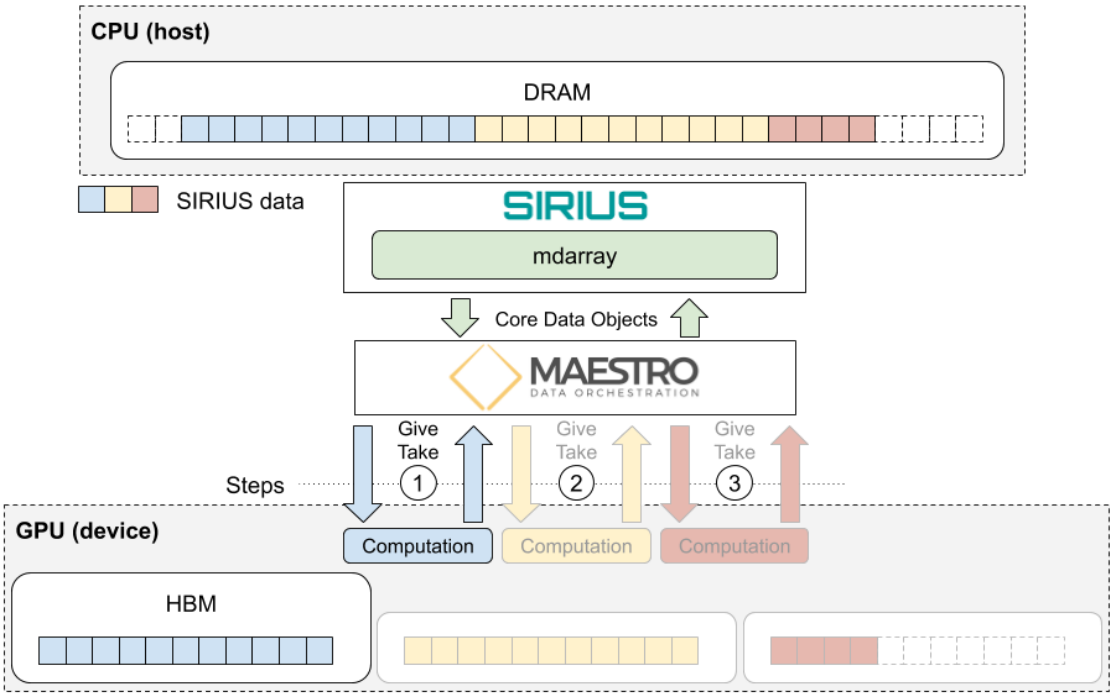

Figure 10: Maestro enabled version of SIRIUS.

SIRIUS is a very active project and one of its priorities is to provide the community with highly optimized code. Thus, performance gains will not be a criterion for evaluating the success of the porting with Maestro. However, the absence of regression is of paramount importance. To date, CPU <-> GPU data movement in SIRIUS relies on both manual data slicing and a dedicated class for managing memory pools. In both cases, Maestro will simplify the code and make it more flexible. In summary, the evaluation criteria for the Maestro-enabled version of SIRIUS are as follows:

- No performance regression. The table below provides baseline values to compare with.

Use-case	B_14-K_18-Li_6-Nd_6-O_42	C_18-F_2-Ga_8-N_6-O_42-P_10
System	Piz Daint	Piz Daint
#Nodes	64	64
num_dft_iter	10	10
gk_cutoff	10	8.366
pw_cutoff	25	23.664
Execution time (s)	**513.28**	**259.09**

- Semi-automatic data partitioning
- Automatic memory pool management (decrease in lines of code)

4.5.4 Licensing Information

SIRIUS has been distributed under a BSD 2-Clause simplified license. The code is freely available here: https://github.com/electronic-structure/SIRIUS

5. Concluding Remarks

This document is a summary of the final model workflows released corresponding to the usage scenarios described in deliverable [D2.1]. These are used by the project partners to investigate data use patterns, inform the development of new middleware functionality and provide a baseline against which the new developments can be tested. The model workflows included here have been extended and updated from the prototype versions in D2.2 Workflow Model Prototypes. They include larger workloads geared towards further testing of the Maestro middleware and they also act as the performance baseline for the Maestro evaluation. A summary of major components of each model workflow and expected modifications for porting them to use the Maestro middleware have been outlined in this document. The actual porting and demonstration of the Maestro-enabled applications will be delivered in D6.5 Applications Demonstration.

The four applications focus on different parts of the Maestro design and therefore should be seen as a proof of concept for a larger set of applications that could

benefit from the Maestro middleware. The final application porting results will be demonstrated towards the end of the Maestro project.

References

[D2.1]	Maestro D2.1 Workload characterization and Middleware Requirements
[tsmp]	https://datapub.fz-juelich.de/slts/index.html
[cecill]	https://cecill.info/index.en.html
[ffmpeg]	https://ffmpeg.org/
[hydro]	https://github.com/HydroBench/Hydro
[paraview]	https://www.paraview.org/
[kronos]	https://github.com/ecmwf/kronos
[fdb]	https://github.com/ecmwf/fdb
[multio]	https://github.com/ecmwf/multio
[ec-xc40]	https://www.top500.org/system/178431
[ec-gh]	https://github.com/ecmwf
[fdb_17]	https://dl.acm.org/doi/10.1145/3093172.3093238
[hercule]	O. Bressand, L. Colombet, A. Fontaine, G. Harel and J-B; Lekien, 2012, *Hercule: a library of scientific data management for numerical simulation*, CHOCS, 41, 29-37
[kurz2016]	TerrSysMP–PDAF (version 1.0): a modular high-performance data assimilation framework for an integrated land surface–subsurface model