D6.2 Application Demonstrators Description

Description document

Work package:	**WP6 Middleware Validation and Systems Software Demonstrators**	
Author(s):	François Tessier, Maxime Martinasso	ETHZ
Reviewer #1	Utz-Uwe Haus	Cray
Reviewer #2	Manuel Arenez	Appentra
Dissemination Level	Public	
Nature	Report	

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.

Date	Author	Comments	Version	Status
11/09/2019	François Tessier	Initial version	0.1	Draft
10/10/2019	François Tessier	Updated partner requirements	0.2	Draft
28/11/2019	François Tessier, Maxime Martinasso	Proofreading	0.3	Draft
03/12/2019	François Tessier	For internal review	1.0	Draft
19/12/2019	François Tessier	Updates from reviews	1.1	Final

Contents

1. Introduction

This document will describe, for each of the use-cases detailed in WP2, how core features of Maestro will be demonstrated. A particular scenario is described with and without using the Maestro middleware. The use-cases presented here are maintained by four Maestro partners:

- ECMWF: IFS numerical weather prediction system
- CEA: Computational fluid dynamics plus in-situ analysis
- JUELICH: Earth modelling system TerraSysMP
- ETHZ: Electronic structure calculation library SIRIUS

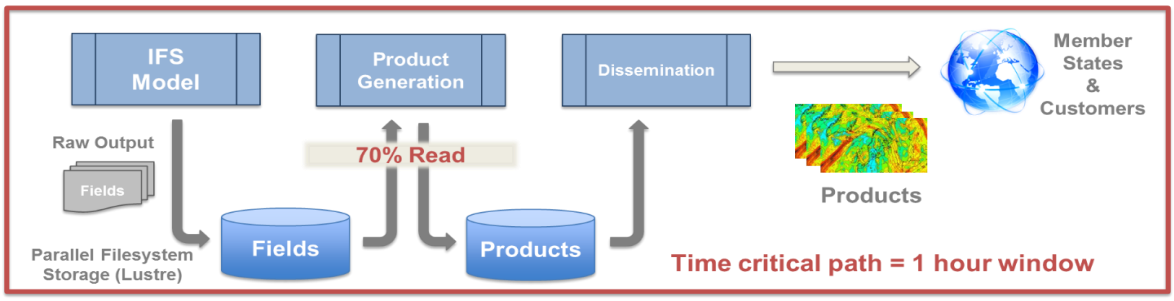
Expectations in terms of performance, usability or any other metric that is relevant for the use-case will help evaluating the benefits provided by Maestro.

For each scenario, a main description is given as well as a description of a scenario which is the actual demonstrator. Another section explains how Maestro will be incorporated in this use-case for improved data movements. A list of expected outcomes is also detailed. Those potential improvements can be expressed in terms of performance gain or usability (automation, code). The goal of the four use-cases presented in this document is to validate Maestro core features. In that sense, a link is made in the last section with a previous deliverable (D2.1) and particularly with Maestro core requirements expressed in this document. The future implementation of those demonstrators will have to cover this list of prerequisites. In particular, the ones tagged as “must have” will be proven.

2. IFS numerical weather prediction system (ECMWF)

2.1. Introduction

ECMWF runs operational weather forecasts four times a day and each of those runs executes the full operational workflow. The figure below shows a sketch of the current operational workflow: it consists of three distinct parts: forecast, product generation and dissemination. Crucially, the entire workflow needs to complete within a time-critical window of one hour.



As part of the forecast system, ECMWF's Integrated Forecast System (IFS) creates an ensemble of weather forecasts (52 different forecasts) that predict the weather for up to 15 days (or more). During this simulation phase, each forecast evolves in three-dimensional meteorological fields, which are distributed geographically on a given number of 'compute' processes. A single process is responsible for a three-dimensional partial domain that extends to the full height of the atmosphere but covers only a fragment of the global horizontal domain. On the other hand, data are archived as units of spherical slices of global two-dimensional fields, each representing the full horizontal global domain but only one level of the atmosphere. The transformation between the three-dimensional partial fields and two-dimensional spherical slices of fields is executed by dedicated I/O servers. Once the transformation is complete, these two-dimensional slices can be treated as independent and require no further synchronisation; the simulation may continue unhindered. Finally, a data multiplexer decides, based on a runtime configuration, which I/O-stack components to use as data sinks; it is currently version 5 of the Fields Database (FDB). The FDB supports different storage technologies as backends but currently it persists data to a globally-visible Lustre filesystem.

The product-generation workflow is triggered when all the necessary fields for a given product have been made available. There is one parallel job per output step across all ensemble members and it needs to read across otherwise independent writing streams. It then applies user-defined requirements to build pipelines of transformations, such as conversion between spherical-harmonic and grid-point representations, interpolation, rotation and cropping. The pipelines are cached and combined to form a directed acyclic graph (DAG) that can be executed efficiently. The products are re-encoded and written back to FDB. The dissemination system reads the products from FDB again and distributes them to ECMWF's member states and commercial customers. The semantics of FDB and movement of data is controlled and driven according to the WMO[1]'s GRIB[2] standard and the language of ECMWF's Meteorological Archival and Retrieval System (MARS).

1 World Meteorological Organization
2 A general purpose, bit-oriented data exchange format (GRIdded Binary)

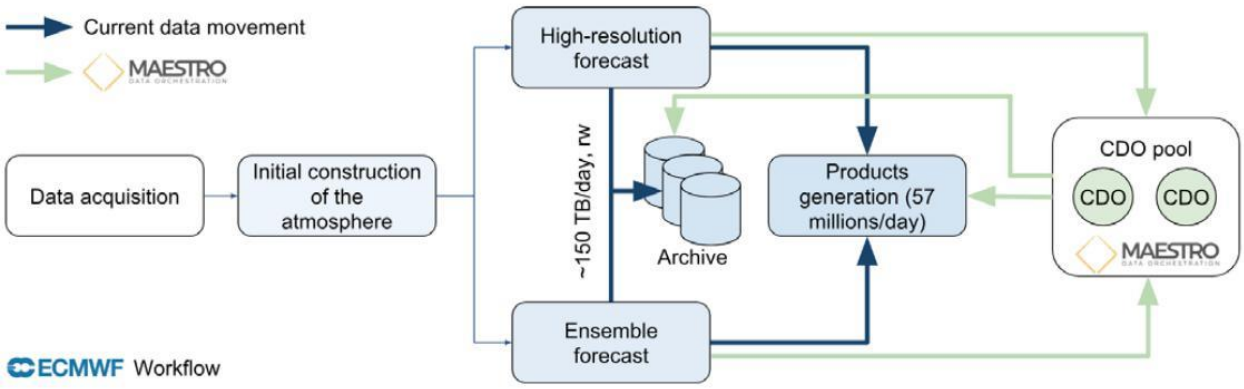
An in-house workflow manager called `ecFlow` is used to co-ordinate the various tasks in the entire ECMWF operational workflow.

2.2. Current Status

Congestion in data movement typically occurs between the forecast workflow and product generation because of the large number or simultaneous read and write operations. To capture this without running the entire operational pipeline, we have created a model workflow that contains most of the actual workflow components relevant for the Maestro project, but where the forecast workflow is replaced by a simple tool called `multio-hammer`. It mocks the output production from the I/O servers and it is configurable to reproduce output patterns of current operational (and research) workflows as well as to match expected patterns from future forecasts (with higher resolution or more ensemble members, for example). It can sink large number of fields of arbitrary size to FDB (or, indeed, any other object store) and send notifications to the workflow manager to trigger the product-generation workflow.

2.3. Maestro-enabled Version

Potential benefits from the Maestro project are twofold. First, the maestro-core software could act as an alternative coupler between data output from the forecast and data input to product generation, effectively providing an alternative to FDB. Second, Maestro would facilitate running improved (Maestro-enabled) workflows in existing workflow managers, such as `ecFlow`, that better orchestrate data movement between components. The figure below provides an example of what role Maestro could play in future operational workflows: it may act as a hot cache between forecast and product generation.



2.4. Expected Outcomes

We would expect Maestro-enabled versions of our workflow to meet all the requirements that were described as ‘must have’ and the majority of the requirements that were described as ‘should have’ in the document D2.1 Workload Characterisation and Middleware Requirements. We would also expect to be able to carry out both a qualitative and a quantitative comparison between the

current version and the Maestro-enabled version. The demonstrators should be able to highlight differences in performance, workflow structure and data movement.

2.5. Covered Middleware Requirements

The demonstrators are software applications designed to illustrate one or both of the usage scenarios (US) from the requirements document 'D2.1 Workload Characterisation and Middleware Requirements'. Each demonstrator is either a single software application or a simplified workflow of a set of software applications, with a given configuration. Each implements one or more use cases (UC) to demonstrate some or all the requirements imposed by that use case. We will use the following list of ECMWF demonstrators:

- A demonstrator that illustrates operational weather forecast (US1.1) and implements accessing semantically-related datasets (UC1.1).
- A demonstrator that illustrates both the operational weather forecast (US1.1) and the research experiment (US1.2) scenarios, and implements requesting sets of objects (UC1.7). This will also imply accessing semantically-related datasets (UC1.1).
- A demonstrator that illustrates both the operational weather forecast (US1.1) and the research experiment (US1.2) scenarios, and implements monitoring system characteristics (UC1.4).
- A demonstrator that illustrates operational forecast (US1.1) and implements high-velocity production of meteorological objects (UC1.2).
- A demonstrator that illustrates both the operational weather forecast (US1.1) and the research experiment (US1.2) scenarios, and implements sustained high-volume production of meteorological objects (UC1.6).

The above demonstrators will also partially implement monitoring workflows UC1.5) and potentially implement restarting a forecast from previously computed forecast data (UC1.3).

These demonstrators will validate the following requirements.

Reference in D2.1	**R1.1**
Description	Objects handled within Maestro can be described and handled according to user-defined domain-specific metadata.
Validation	The demonstrator will describe data according (or similar) to the MARS language before handing it over to Maestro.

Reference in D2.1	**R1.2**
Description	User-defined metadata should take the form of a dictionary of key-value pairs, or equivalently a set of arbitrarily named attributes. Calls to the Maestro core API are expected to behave according to the specifications.

Validation	The demonstrator will define metadata as a dictionary of key-value pairs or a set of arbitrarily named attributes. Calls to the Maestro core API are expected to behave according to the Maestro core specifications.

Reference in D2.1	**R1.3**
Description	Object retrieval should support metadata queries that describe lists of values.
Validation	The demonstrator will make queries using lists of values. These queries are expected to succeed.

Reference in D2.1	**R1.4**
Description	Object retrieval should support metadata queries that describe range of values, for example by automatically expanding it to a list.
Validation	The demonstrator will make queries for ranges of values. These queries are expected to succeed and behave according to the specifications.

Reference in D2.1	**R1.7**
Description	Once data objects handed over to Maestro, transport, indexing and access must no longer be the application's concern.
Validation	The demonstrator will make the Maestro API calls that are required by the ECMWF pipelines. It will not carry out any of the above operations once objects are handed over to Maestro.

Reference in D2.1	**R1.8**

Description	Objects may have a minimum lifetime (possibly zero).
Validation	The demonstrator will have consumers that require access to data some time after the data was handed over to Maestro. The access is expected to be granted during a user-defined period.

Reference in D2.1	**R1.10, R1.11, R.12**
Description	Requirements on Maestro's error-handling functionality.
Validation	During the demonstrators' development, configuration and execution, errors are expected to occur and thus these requirements will at least partially be validated.

Reference in D2.1	**R1.14**
Description	Maestro to record and communicate usage statistics – type of storage used and their load, write (production) rate, read (consumption) rate, etc. – to human operators or system monitoring/logging tools.
Validation	The demonstrator will make queries to Maestro about its usage statistics. These queries are expected to behave according to the specifications.

Reference in D2.1	**R1.15**
Description	Maestro to be able to cope with at least 20,000 object creations per second per workflow. This is sustained for most of the workflow duration.

Validation	The demonstrator will be able to be configured to create an arbitrarily large amount of mock data, which is scientifically meaningless but otherwise has the same characteristics as actual data. These will be fed into the Maestro-enabled pipeline.

Reference in D2.1	**R1.16**
Description	Maestro to be able to cope with the creation of at least 150TiB data in an hour per workflow.
Validation	The demonstrator will create a large volume (many hundreds of TiB) of mock data per hour to demonstrate Maestro's 'limits'.

Reference in D2.1	**R1.17**
Description	Objects within Maestro are handled transactionally. No partial state must exist.
Validation	The demonstrator will carry out sanity checks that objects returned from Maestro are not in a partial state. Objects in partial state will trigger a hard failure. These sanity checks, however, will not be able to prove that partial states cannot exist.

Reference in D2.1	**R1.19**
Description	Objects within Maestro are handled consistently. No risk of undefined behaviour or data corruption.
Validation	The demonstrator will carry out sanity checks for consistency.

Reference in D2.1	**R1.21**

Description	A single consumer should be able to iterate over a set of objects, even when this is too large to fit in memory at once.
Validation	The demonstrator will make attempts at iterating over large datasets that typically do not fit into memory. The operations are expected to succeed.

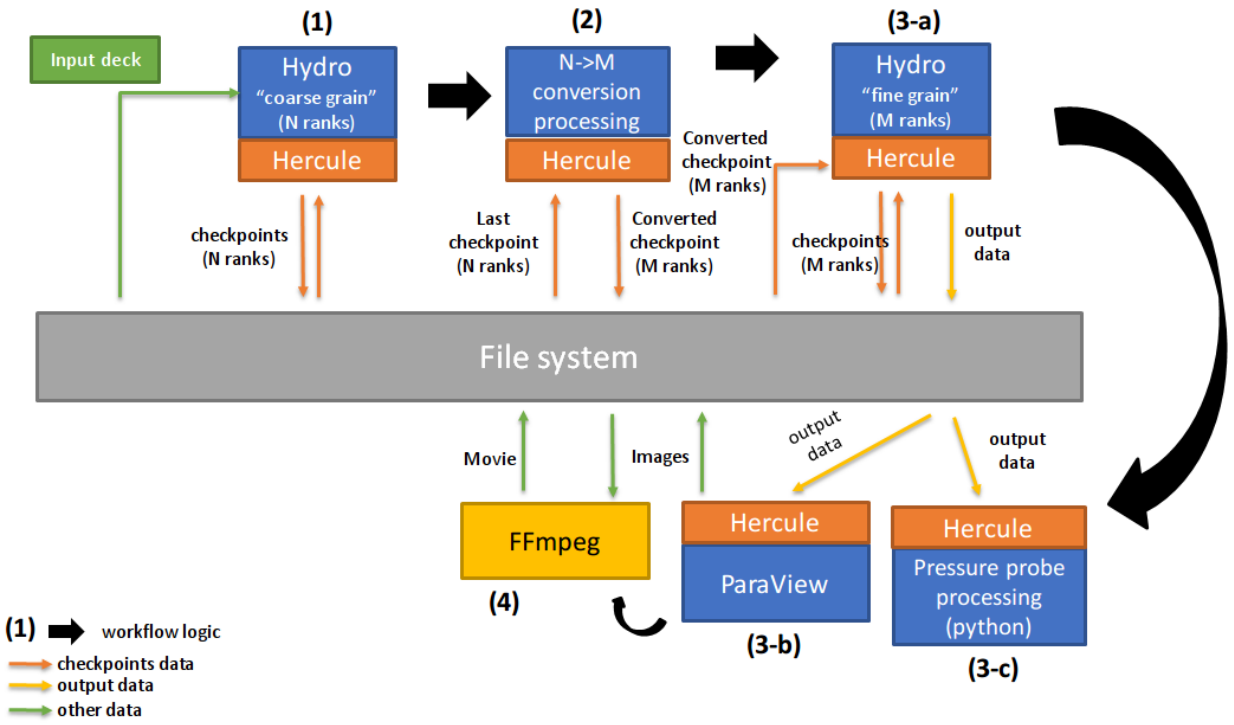
Reference in D2.1	**R1.24**
Description	Once data objects are handed over to Maestro, they must be immutable.
Validation	The demonstrator will attempt to create data with semantic meaning (metadata) identical to data that already exists within Maestro. That attempt is expected not to mutate the existing data. It is expected either to fail or to create new data.

Reference in D2.1	**R1.26**
Description	Consumer applications should not be required to know the size of the data prior to requesting it.
Validation	The demonstrator will request data from Maestro without specifying the size of the data.

3. Computational Fluid Dynamics plus in-situ analysis (CEA)

3.1. Introduction

The CEA workflow provided for MAESTRO is representative of a 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 hydrodynamic simulation code, to model our two simulation codes. The chaining mechanism will be represented by a restart of the simulation with a different domain decomposition, the check-pointing data serving as data exchange between the "two" codes. Two post-processing pipelines are then run on Hydro output data: 1/ a custom analytical processing written in python, 2/ a graphical processing with ParaView and FFmpeg to produce a movie of the simulation. This workflow is illustrated in the figure below.



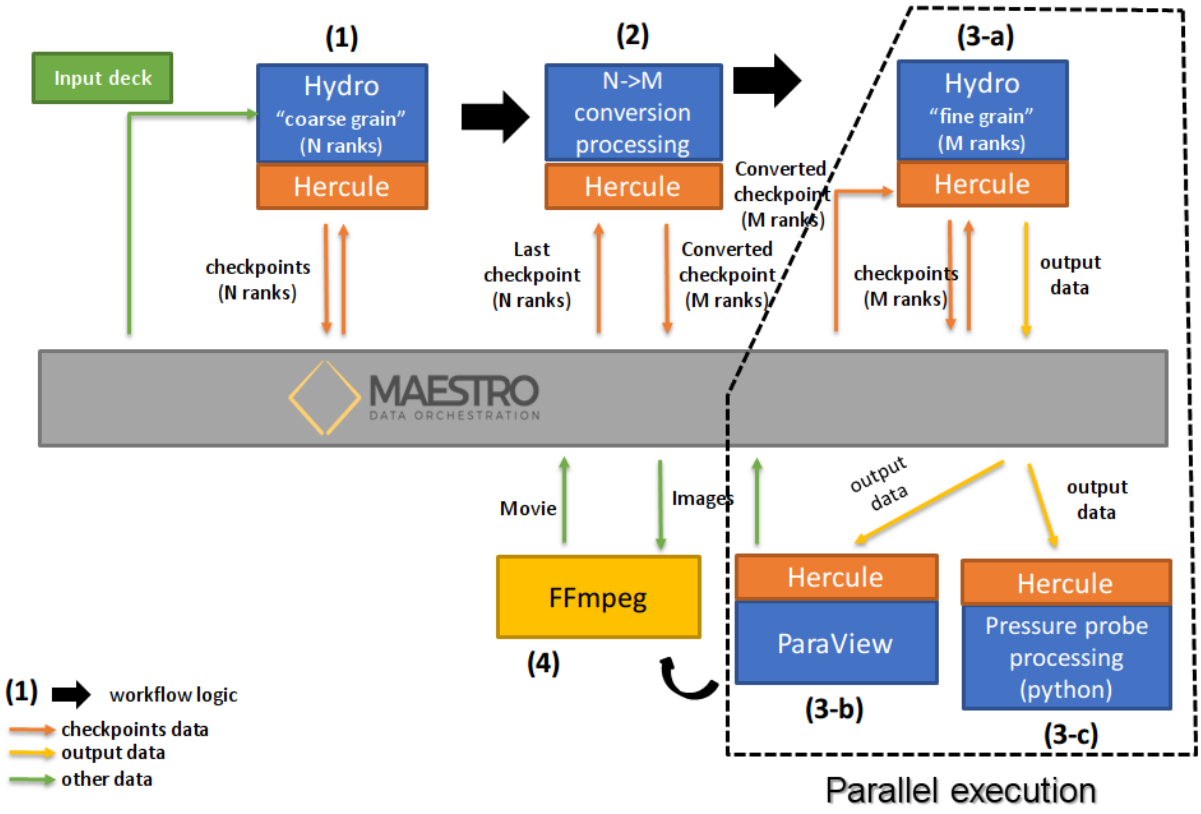
3.2. Current Status

The open source version of Hydro currently uses VTK files for writing output data and does not support checkpoint/restart. Developments are on-going to integrate CEA's high-performance I/O library Hercule into Hydro to allow checkpoint/restart and domain recomposition in order to model the chaining of Hydro with itself (based on a restart with a different domain decomposition).

The workflow prototype currently provided in WP2 is a simple pipeline consisting of one run of Hydro writing VTK files, followed by the ParaView+FFmpeg processing.

3.3. Maestro-enabled Version

The Maestro-enabled version will consist in modifying the Hercule I/O library to allow using Maestro CDOs to produce and/or consume simulation data with the primary objective of executing the processing steps, no longer *post-*, but *in situ* in a loosely coupled fashion (also called *in transit*). In other words, it consists in executing in parallel the steps 3-a), 3-b) and 3-c), with simulation output data "streamed" from the simulation to the two processing. This is shown in the figure below:



3.4. Expected Outcomes

Our expectations are that Maestro will allow to set up and run in a user-friendly manner such workflow with a portion run in situ/in transit efficiently. The maestro-enabled version will allow CEA to demonstrate the use of memory hierarchy we plan to have on future production systems. This demonstration will help in using maestro in other CEA applications and in applications from other communities which use CEA computing centres.

3.5. Covered Middleware Requirements

The software demonstrator is designed to illustrate both usage scenarios (US) 2.1 and 2.2 from the requirements document 'D2.1 Workload Characterisation and Middleware Requirements'. The entire workflow modelled by the demonstrator will illustrate US2.1, while a portion of it will be used to represent US2.2. The following uses cases (UC) will be covered by this demonstrator:

- UC2.1: Applications write/read Data Collections to/from Maestro
- UC2.2: "Streaming" data records between producers and consumers
- UC2.3: Ownership and persistence of a data
- UC2.4: Simulation checkpoints spooling

Finally, we list the requirements per deliverable D2.1 covered by the demonstrator.

Reference in D2.1	**R1.1**
Description	Objects handled within Maestro can be described and handled according to user-defined domain-specific metadata.
Validation	The demonstrator will use our I/O library Hercule to passed simulation data to/from Maestro. This library attaches a series of attributes to data arrays including, dimensionality, domain-specific semantic information and relationships between arrays.

Reference in D2.1	**R1.2**
Description	User-defined metadata should take the form of a dictionary of key-value pairs, or equivalently as a set of arbitrarily named attributes.
Validation	The demonstrator will use Maestro user-defined named attributes through the Hercule I/O library to record data arrays attributes, see previous requirement.

Reference in D2.1	**R1.6**
Description	Support describing a set of fields as 'complete'.
Validation	The second Hydro code (fine grain) will start once the conversion processing (NxM domain decomposition) is finished / « completed ».

Reference in D2.1	**R1.7**
Description	Once data objects are handed over to Maestro, transport and access must no longer be the application's concern. Maestro should take care of maintaining unique references to the data objects, and facilitate locating objects according to their metadata.

Validation	The demonstrator will give data to Maestro that will get ownership on it. This data is expected to be transported and accessed without the application's concern.

Reference in D2.1	**R1.22**
Description	Multiple consumers must be able to iterate over a set of requested objects simultaneously, either all of them consuming all the objects or distributed amongst the consumers as they are able to consume them.
Validation	Hydro output data will be used simultaneously by both a custom analytical processing and graphical processing.

Reference in D2.1	**R1.25**
Description	Maestro may support iteration granularity larger than a single data object.
Validation	Hercule I/O library will use Maestro to create groups of data objects to represent a record of related objects (e.g. all data produced by the simulation for a given time step).

Reference in D2.1	**R2.1**
Description	Maestro must allow a conceptual data organization based on the following data container concepts: - array: a multi-dimensional typed array - record: a grouping of related datasets published at the same time (e.g. simulation output fields at a specific time step) - collection: a grouping of related records (e.g. checkpoint data collection, post-processing data collection)
Validation	Hercule I/O library will use grouping of data objects in Maestro to model this data organization.

Reference in D2.1	**R2.2**
Description	Maestro must provide the ability to add (and query) named attributes (metadata) to data arrays, records and collections.
Validation	Hercule library will attach and query named attributes to data objects and to groups of data objects.

Reference in D2.1	**R2.4**
Description	A data collection must accommodate records produced by multiple

	independent producers. Hence, labelling of data records is at least a 2-tuple made of a sequence integer (the sequence number of a record with the list of records) and a producer ID integer. Maestro must provide an indexing or attribute system to be able to support this.
Validation	Hercule I/O library groups together data objects produced by a simulation rank at a given time step, hence this group is tagged with a time step number and a producer id.

Reference in D2.1	**R2.6**
Description	Maestro should provide a configuration system to describe properties of the workflow/pipelines for which it will manage the data exchange, where these properties are known in advance.
Validation	Analytic and graphic processing applications will not need all the data provided by the hydro code to Maestro.

Reference in D2.1	**R2.7**
Description	In “streaming” mode, a consumer should have the opportunity to declare in advance (in configuration) the data arrays it will consume.
Validation	Analytic and graphic processing applications will consume specific data generated by the hydro code.

Reference in D2.1	**R2.8**
Description	Maestro must provide the ability to persist data beyond the lifetime of the workflow that generated it, and make it available to appropriate consumers.
Validation	Data checkpoint made persistent will be used to resume the workflow in a state different from its start.

Reference in D2.1	**R2.9**
Description	When used for persisting data produced by an application, Maestro should provide the ability to configure which data collection is persisted.
Validation	User will instruct Maestro that checkpoint collections need to be persisted.

Reference in D2.1	**R2.10**
Description	When used for persisting data produced by an application, Maestro

	should provide the ability to specify a subsampling rate of persistence or filter.
Validation	Hydro application will generate data that will be made persistent every N iterations / time period.

Reference in D2.1	**R2.11**
Description	When used for persisting checkpoint data, Maestro must provide the ability to specify the number of recent checkpoint records that will eventually be persisted.
Validation	Checkpoints spooling will be performed to keep only the last N checkpoints.

Reference in D2.1	**R3.1**
Description	Grant ownership of data to Maestro
Validation	Data produced will be provided to Maestro.

Reference in D2.1	**R3.3**
Description	Require data owned by Maestro
Validation	Consumer applications will query data owned by Maestro.

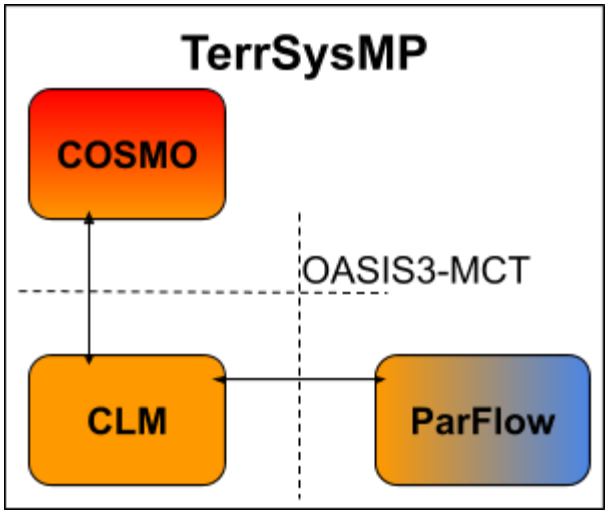
4. Global Earth Modelling system TerrSysMP (JUELICH)

4.1. Introduction

As described in D2.1, the Terrestrial Systems Modelling Platform (TerrSysMP or TSMP) targets the simulation of interactions between lateral flow processes in river basins and lower atmospheric layers. This is achieved by combining three model components COSMO, CLM and Parflow. The coupling is done using OASIS3-MCT which is an external coupler. The COSMO-model is a non-hydrostatic limited area atmospheric model developed by the COnsortium for Small-scale MOdelling. The Community Land Model (CLM) is used to study the effect of terrestrial ecosystems on climate. Finally, Paraflow (PARallel FLOW) is an integrated hydrology model used to simulate surface and subsurface flow. To couple all three models OASIS3-MCT is used, which is a fully parallel implementation for coupling field re-gridding and exchange.

4.2. Current Status

In the current version of TerrSysMP, OASIS3-MCT is used to couple 2D fields between all three models. To achieve this, a configuration file provides the coupling frequency, defines the two-way data exchange and the required data transformations.

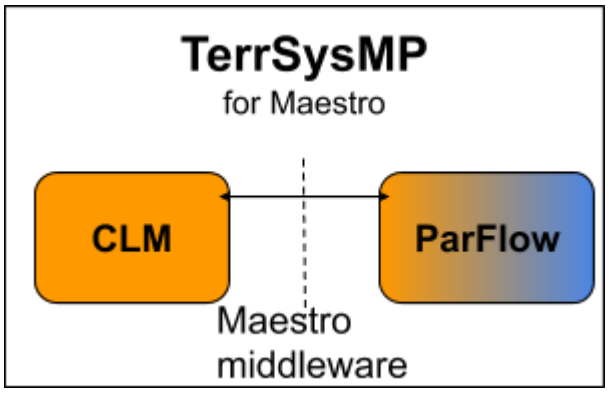


The distributed fields are exchanged via MPI across processes and nodes. The transformations include frequent row/column major order transformation, along with frequent data redistribution or remapping. Additionally, frequent data interpolation is done based on SCRIP.

4.3. Maestro-enabled Version

As part of the effort done in WP2, we have identified two possible opportunities for leveraging Maestro in TerrSysMP. First, we consider the opportunity of exchanging 2D fields

between climate models. For demonstration purposes we will focus only on the interaction of two models, specifically CLM and ParFlow. The target is to replace OASIS3-MCT as a coupler in the process of exchanging the 2D fields. The fields need to be transferred, remapped and interpolated by the Maestro middleware.



A secondary identified opportunity is the ability to create a better load balancing across models. In its current setup, load balancing is done a-priori and heuristically, based on intuition and user experience. Using Maestro for coupling could allow for finding a better distribution of resources across models by leveraging a profile-driven workflow. Repeatedly running several coupling cycles and measuring performance could determine a better load balance for a given system and experiment.

4.4. Expected Outcomes

It is our ambition to demonstrate that Maestro, based on a more general concept, can reach at least similar performance as a highly specialised coupler like OASIS3. If implemented the dynamic load balancing of TerrSysMP coupled models, would allow for better resource utilisation.

4.5. Covered Middleware Requirements

The here described opportunities for leveraging Maestro in TerrSysMP directly reflect the Use Cases (UC) identified in D2.1, namely UC4.1 “Exchange of 2D fields between climate models” and UC4.2 “Dynamic load balancing of coupled models”. The following tables list the requirements imposed by the two TerrSysMP use cases.

Reference in D2.1	**R4.1, R4.2 and R4.3**
Description	**(R4.1)** Redistribution of parallel 2D fields **(R4.2)** Row and column-major layout transformation of distributed arrays **(R4.3)** Interpolation of distributed arrays
Validation	The exchanged fields between coupled models should be transferred,

	remapped and interpolated by the Maestro middleware.

Reference in D2.1	**R4.4**
Description	Start and end multiple MPMD runs within the same job allocation
Validation	The redistribution of coupled models over different processes is required to repeatedly measure performance to determine a better load balance for a given system and experiment.

5. Electronic structure calculation library SIRIUS (ETHZ)

5.1. Introduction

The DFT Loop mini-app is part of the SIRIUS package, a domain specific library for electronic structure codes developed at ETHZ/CSCS. 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. The diagonalization based methods used in those core algorithms rely heavily on the linear algebra operations. They are usually compute intensive and thus are the perfect candidates for the GPU acceleration. However, the matrices involved in the calculation are usually large and not always fit into the memory of a device. This implies significant data movements between CPU and GPU memories.

5.2. Current Status

The mini-app is a parallel and distributed code. The figure 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.

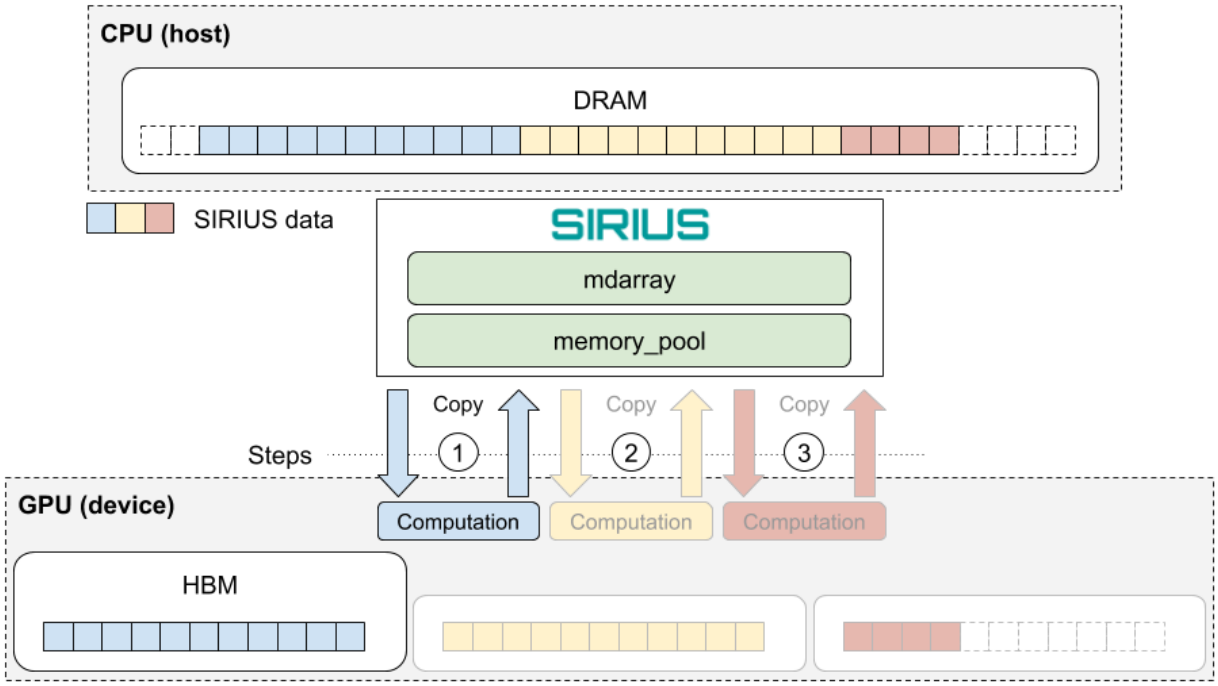


Figure 1: Data flow in SIRIUS between CPU and GPU memories

5.3. Maestro-enabled Version

The Maestro-enabled version of our workflow will impact the low-level layer of the SIRIUS library. Maestro will be a substitute for memory management and the `memory_pool` classes. Chunks of data will be encapsulated into a CDO (Core Data Object) and given to Maestro. The GPU component will ask ownership of the data either using Maestro pool operations or Maestro's transformation API and Mamba, process it and give it back to Maestro and SIRIUS. Figure 2 depicts this Maestro-enabled version of SIRIUS.

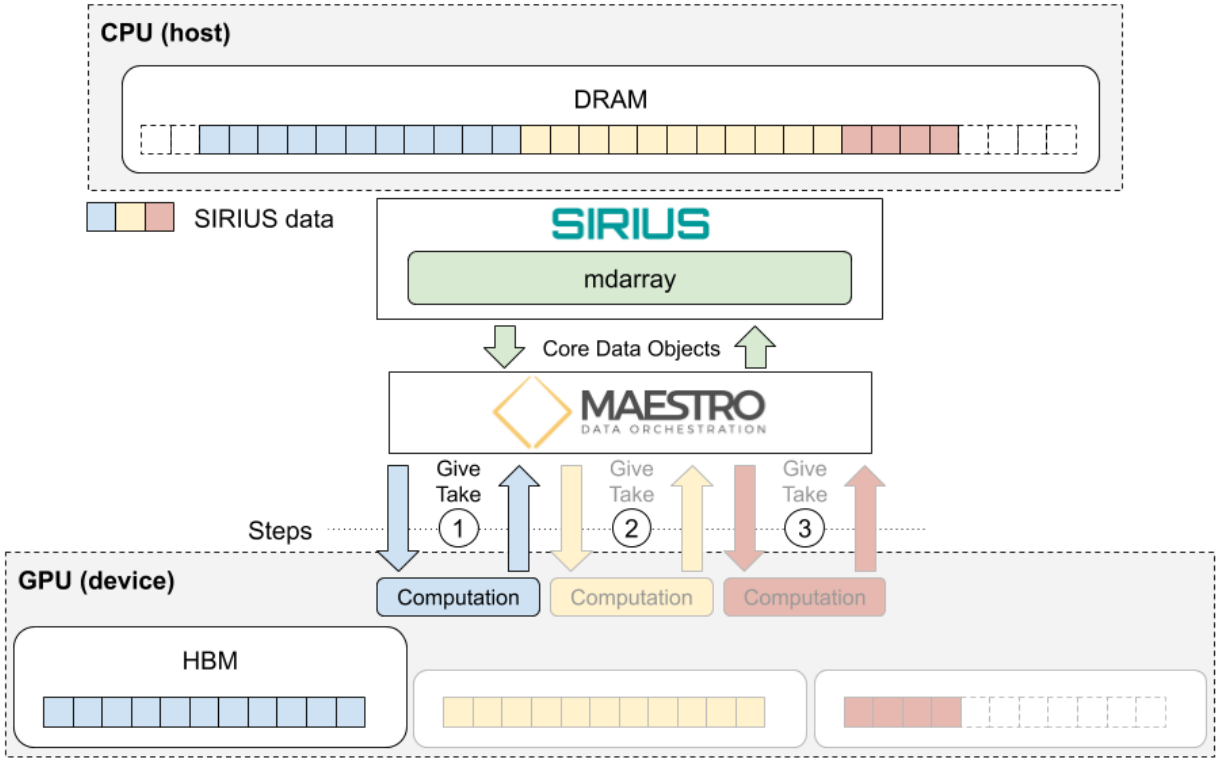


Figure 2: Maestro-enabled data flow in SIRIUS between CPU and GPU memories

5.4. Expected Outcomes

The memory management layer of the SIRIUS library is well optimized. No performance gain is to be expected with a Maestro-enabled version. However, along with performance stability, we expect Maestro to bring a better usability and automation of data movement between host and accelerator's memory.

5.5. Covered Middleware Requirements

Using references to deliverables D2.1 (M9), below is a list of requirements this SIRIUS-based demonstrator will cover. For each requirement, we give its index into the previously submitted deliverable, a short description and how this feature will be validated in our demonstrator. Those requirements were initially extracted from a set of use-cases:

- UC3.1: Move data from CPU to GPU memory
- UC3.2: Management of memory resources

These use-cases are themselves part of a more high-level SIRIUS-based scenario (US3.1: electronic structure calculation).

Reference in D2.1	**R3.1**

Description	Grant ownership of data to Maestro
Validation	Data allocated through the `mdarray` library will be "given" to Maestro that will get the ownership on it. That way, Maestro will be able to take data movement decisions from host to GPU memory or the opposite way.

Reference in D2.1	**R3.2**
Description	Indicate Maestro to move or allocate data to a particular tier of memory.
Validation	While giving ownership of data to Maestro, the calling process (host) implicitly or explicitly requests an allocation on the GPU to move data.

Reference in D2.1	**R3.3**
Description	Require data owned by Maestro.
Validation	Both the CPU and the GPU will require data owned by Maestro. The GPU will ask for ownership to apply linear algebra algorithms. The CPU will retrieve ownership afterwards.

Reference in D2.1	**R3.4**
Description	Maestro is able to manage memory resources (including allocation and deallocation) across multiple layers of the memory hierarchy.
Validation	Allocation and deallocation will be carried out by Maestro on both sides, CPU and GPU memories.

Reference in D2.1	**R3.6**
Description	Maestro can move data from one memory space to another
Validation	Maestro will have to move data from the CPU memory to the GPU high-bandwidth memory for computation.

Reference in D2.1	**R3.7**
Description	Maestro can manage different memory spaces within a node

Validation	In our demonstrator, Maestro will manage two types of memory: the CPU memory and the high-bandwidth memory available on the accelerator.

6. Concluding remarks

This document presents four demonstrators based on the four use-cases identified for the Maestro middleware. These demonstrators generally correspond to a particular scenario as detailed in a previous deliverable (D2.1). For each application, a list of requirements that will be validated is given. The aforementioned D2.1 document can give more details about those requirements. However, it has to be noted that most of them describe a low-level core functionality of Maestro.

This document lays the foundation of D6.5, due at the end of the project, whose goal is to adapt the source code of the characterized demonstrators (original source code provided in D2.2[3]) and show the experimental results of their Maestro-enabled version on HPC systems.

3 https://gitlab.version.fz-juelich.de/maestro