D6.3 Application port feedback

Work package	WP6 Middleware Validation and Systems Software Demonstrators	
Author(s)	Marc Pérache Xavier Dussieux Jayesh Badwaik M Salem El Sayed Maxime Martinasso Domokos Sármány	CEA CEA Juelich Juelich ETH Zurich/CSCS ECMWF
Reviewer #1	Utz-Uwe Haus	HPE
Reviewer #2	Dirk Pleiter	Juelich
Dissemination Level	public	
Nature		

Date	Author	Comments	Version	Status
05.03.2021	Pérache Marc	First version	V0.1	Draft
25.04.2021	Pérache Marc	Final version	v1.0	Final

ESTRATION IN HIGH PERFORMANCE COMPUTING
has received funding from the European Union's Horizon 2020 research and innovation
rough grant agreement 801101.

Executive Summary

Contents

Executive Summary 2
Introduction 3
1. IFS numerical weather prediction system 3
1.1 Introduction 3
1.2 Maestro features 4
1.3 Current Status and feedback from porting efforts 5
2. Computational Fluid Dynamics plus in-situ analysis 10
2.1 Introduction 10
2.2 Maestro features 11
2.2.1 Attribute-based memory choice 11
2.2.2 Data-aware task migration 12
2.2.3 Guided I/O 12
2.2.4 CDO data wrapping 13
2.3 Middleware requirements and current status 13
3. Global Earth Modelling system TerrSysMP 17
3.1 Introduction 17
3.2 Maestro features 17
3.2.1 Transparent Transformation of 2D Fields 17
3.2.2 Load Balancing and Redistribution 18
3.3 Covered Middleware Requirement and Current Status 18
4. Electronic structure calculation library SIRIUS 18
4.1 Introduction 18
4.2 Maestro features 18
4.2.1 Move data from CPU to GPU memory 19
4.2.2 Management of memory resources 19
4.3 Middleware Requirement and Current Status 19
5. Concluding Remarks 21

Introduction

This document describes the current status for each of the use-cases detailed in deliverable D6.2 and D2.4, how core features of Maestro will be demonstrated, and the current status of porting these use-cases on top of Maestro. 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 TerrSysMP
- ETHZ: Electronic structure calculation library SIRIUS

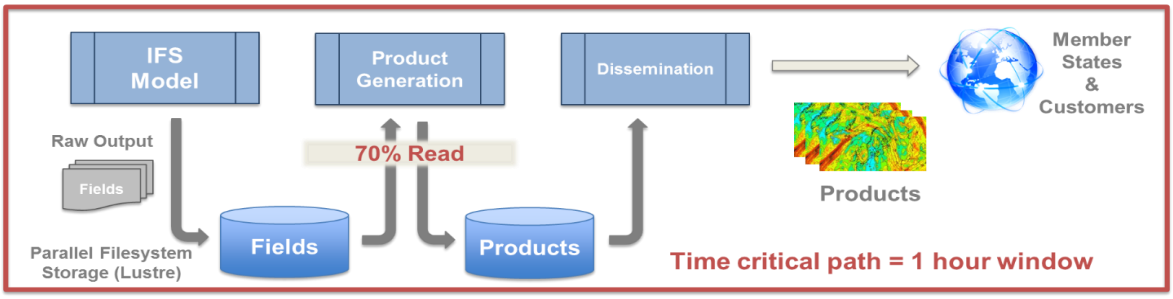
Expectations in terms of performance, usability, or any other metric that is relevant for the use-case will help to evaluate the benefits provided by Maestro. For each scenario, the main description is given as well as a description of a scenario which is the actual demonstrator. Another section explains how Maestro is currently 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). In this deliverable, we also have references to deliverable 2.3. For each use case, we provided a status update for all requirements detailed in deliverable 2.3.
The goal of the four use-cases presented in this document is to validate Maestro's core features. In that sense, a link is made in the last section with a previous deliverable (D6.2).

1.IFS numerical weather prediction system

1.1 Introduction

ECMWF runs operational weather forecasts four times a day and each of those runs executes the full 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. The forecast system is responsible for creating an ensemble of weather forecasts that predict the weather up to 15 days (or more) ahead. It also contains dedicated I/O-servers that carry out the necessary transformations to be able to produce global, two-dimensional fields, each representing the full horizontal global domain but only one level of the atmosphere. Each field is passed to a domain-specific database of meteorological data, the fields database (FDB). The product-generation workflow is triggered when all the necessary fields for a given product have been made available. It applies user-defined requirements to build pipelines of transformations, such as conversion between spherical-harmonic and grid-point representations, interpolation, rotation, and cropping. The products then are re-encoded and written back to FDB. Finally, the dissemination system reads the products from FDB again and distributes them to ECMWF's

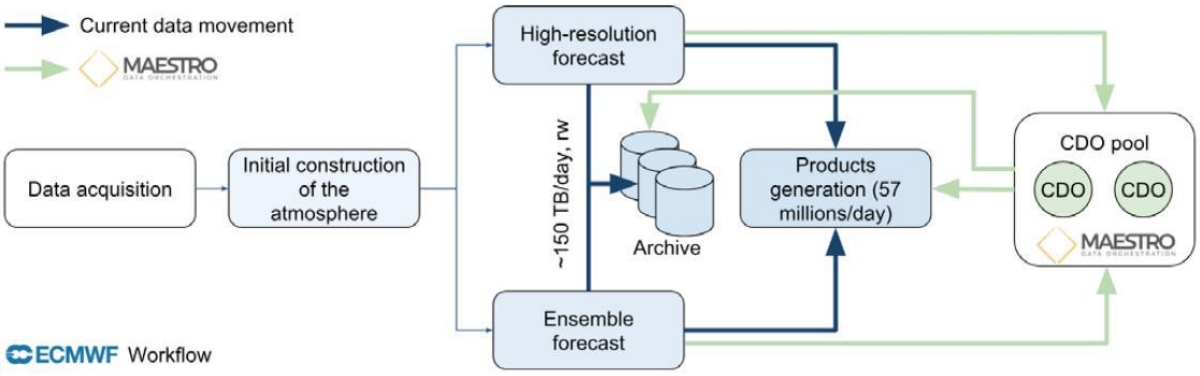
member states and commercial customers. The figure below shows a sketch of the workflow.



Congestion in data movement typically occurs between the forecast workflow and product generation because of the large number of 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`. This is described in detail in D2.3. In short, 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.

1.2 Maestro features

The Maestro middleware will partially or fully replace the component where the congestion occurs: between the forecast and product generation, where data is currently persisted onto a globally-visible parallel file system. The figure below shows how the Maestro middleware could be used to direct the time-critical part of the forecast output to the product-generation task by bypassing the global parallel file system, while archiving another portion of the data in non-time-critical fashion.



The requirements that enable such use of the Maestro middleware in weather forecasting has been listed in detail in D2.3. The demonstrators being built focus on two broad categories of features.

- *Metadata manipulations.* Applications in the workflow need to access semantically related datasets. Data objects are referenced via their scientific meaning, rather than, for example, by their unique ids or names. The Maestro middleware supports these operations through its support for CDO attributes and through an event-detection mechanism, whereby users can subscribe to certain events based on attributes defined by an application-defined attribute schema.
- *Stress conditions.* Operational weather forecasting is time-critical and one of the main motivations for the Maestro middleware is to be able to produce and consume meteorological objects at a high rate. That requires a management solution that supports multiple ways to transfer data. The proposed middleware achieves this by creating a dedicated pool manager and decoupling the data transfer from the pool operations.

Apart from these two major categories, there are requirements to cover basic functionalities, lifetime management, failure resilience, etc. As a result, the IFS-based demonstrators aim to be able to validate if the following high-level features have been correctly implemented.

- *Intelligent data movement.* The general concept to consider information about where the data is likely to be needed in the decisions about where to store it and how to transfer it.
- *Data wrapping.* The concept that data can be encapsulated into a Maestro-specific abstraction and annotated with scientifically meaningful attributes.
- *Avoiding the globally-visible parallel file system.* The concept that it is possible to pass data from the forecast system to the product generation system without having to persist it.
- *Guided I/O.* The concept of using attributes to provide hints about how urgently a request needs to be dealt with. For example, data consumed by product generation needs to be processed more urgently than data scheduled for archiving only.

1.3 Current Status and feedback from porting efforts

For the evaluation of the Maestro middleware we implement a set of demonstrators. These will be used to demonstrate the extent to which the requirements outlined in D2.3 are satisfied. They will also be used to compare the Maestro-enabled version of the model workflow to the original version using certain performance baselines and evaluation metrics. The deliverable document D2.4 describes in detail the IFS model workflow together with the associated evaluation metrics and performance baselines.

Each of the set of demonstrators for the IFS application consists of one or more of the following components, depending on level of complexity.

- *One or more pool-manager applications.* It is part of the investigation of the Maestro middleware to establish the optimal number of pool managers for the full model workflow. In smaller-to-medium-sized demonstrators, one pool manager suffices.
- *One or more* `multio-hammer` *instances, acting as producers.* It is often enough to run a small number of these to validate parts of the Maestro middleware's instances. However, a large number of these will be used to demonstrate some functionality at scale.
- *One or more consumer applications.* Similarly, a small number of these are enough to validate many of the requirements. But a larger number of these are needed to demonstrate the Maestro middleware at scale.
- *The* `kronos` *workload manager.* This is unnecessary, and thus omitted, for smaller demonstrators that consists of only a few number of producers and consumers. It will be used, however, to demonstrate the Maestro-enabled version of the full model workflow.

This is an interim report and the evaluation efforts have so far focused on the semantic requirements, many of which are prerequisites to building a full Maestro-enabled model workflow and measuring it against metrics and baselines. Nevertheless, we have managed to run a setup that aims to verify parts of the Maestro management API that interfaces with the data producers and uses the subscription mechanism to verify that communication between the producer and consumer based on metadata can be established via the pool manager. It also measures the overhead of metadata operations and the interaction with the Maestro middleware API, in particular with the Maestro pool manager. The preliminary results have shown that the metadata communications represent an overhead of a few percent on top of the generation of mock meteorological data, which in turn is much faster than the generation of actual data.

We list below the demonstrators that are being developed to validate one or more requirements from D2.3. The current status of verification is provided for each as feedback. At this stage of the project, we focus on validating functionality using relatively small demonstrators, consisting of a small number of producers and/or consumers, and a single pool manager.

Reference in D2.3	**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.

Status/feedback	*Validated*. There exists a mechanism to create a user-defined attribute schema for domain-specific metadata. Its use has been successfully demonstrated.

Reference in D2.3	**R1.2, R1.27**
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 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.
Status/feedback	*Validated.* The Maestro middleware's support for named attributes is adequate and its use has been successfully demonstrated.

Reference in D2.3	**R1.3a-d**
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.
Status/feedback	*Partially validated.* The existing Maestro interface for handling metadata queries supports lists of values and those queries are successful. However, we identified a small number of cases where the behaviour is not as expected. These include polling based on values of arbitrary metadata and initiating data transfer after polling even if the metadata query is successful. We have raised tickets for each of these which are now being investigated.

Reference in D2.3	**R1.7a-c**
Description	Once data objects are handed over to Maestro, managing these 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.
Status/feedback	*Partially validated.* A small demonstrator has successfully validated that transport and access are no longer the application's concern, and that data objects remain uniquely identifiable. There is support for locating and accessing objects through their metadata, but behaviour is not yet fully to the specifications.

Reference in D2.3	**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.
Status/feedback	*Still to be validated.* There is support for this in the design.

Reference in D2.3	**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.
Status/feedback	*Partially validated.* There exists an extensive error-handling mechanism, most of which is useful. Recommendations on improved error-handling are continuously fed back into the development of the Maestro middleware.

Reference in D2.3	**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.
Status/feedback	*Still to be validated.* Demonstration of the telemetry capabilities is scheduled at a later stage.

Reference in D2.3	**R1.15a-d**
Description	The Maestro middleware must 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.
Status/feedback	*Partially validated.* Synthetic meteorological data has been generated and passed to Maestro to validate its metadata-handling capabilities, which has been successful. Full validation that includes data transfers is yet to be conducted.

Reference in D2.3	**R1.16a-b**
Description	The Maestro middleware must be able to support multiple non time-critical workflows, creating at least a combined 150TiB data in an hour.

Validation	The demonstrator will create a large volume (many hundreds of TiB) of mock data per hour to demonstrate Maestro's 'limits'.
Status/feedback	*Still to be validated.* Demonstration with multiple workflows is scheduled at a later stage.

Reference in D2.3	**R1.17**
Description	Object visibility within Maestro is handled transactionally. No partial state must be accessible.
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.
Status/feedback	*Partially validated.* During the demonstrator's development, some requests have resulted in errors that suggest Maestro ensures partial states cannot be accessed.

Reference in D2.3	**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.
Status/feedback	*Partially validated.* No data corruption or signs of undefined behaviour have been observed so far.

Reference in D2.3	**R1.21**
Description	A single consumer must 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.
Status/feedback	*Partially validated.* There is support for this in the Maestro interface, but it has not yet been demonstrated for a large group of objects.

Reference in D2.3	**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.

Status/feedback	*Partially validated.* This is strongly supported in the Maestro middleware design and there is no evidence to the contrary, but systematic validation is still to be conducted.

Reference in D2.3	**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.
Status/feedback	*Partially validated.* The Maestro middleware's interface supports requesting data objects via its metadata only, without specifying its name, unique identifier, or size. The behaviour, however, is sometimes not as expected.

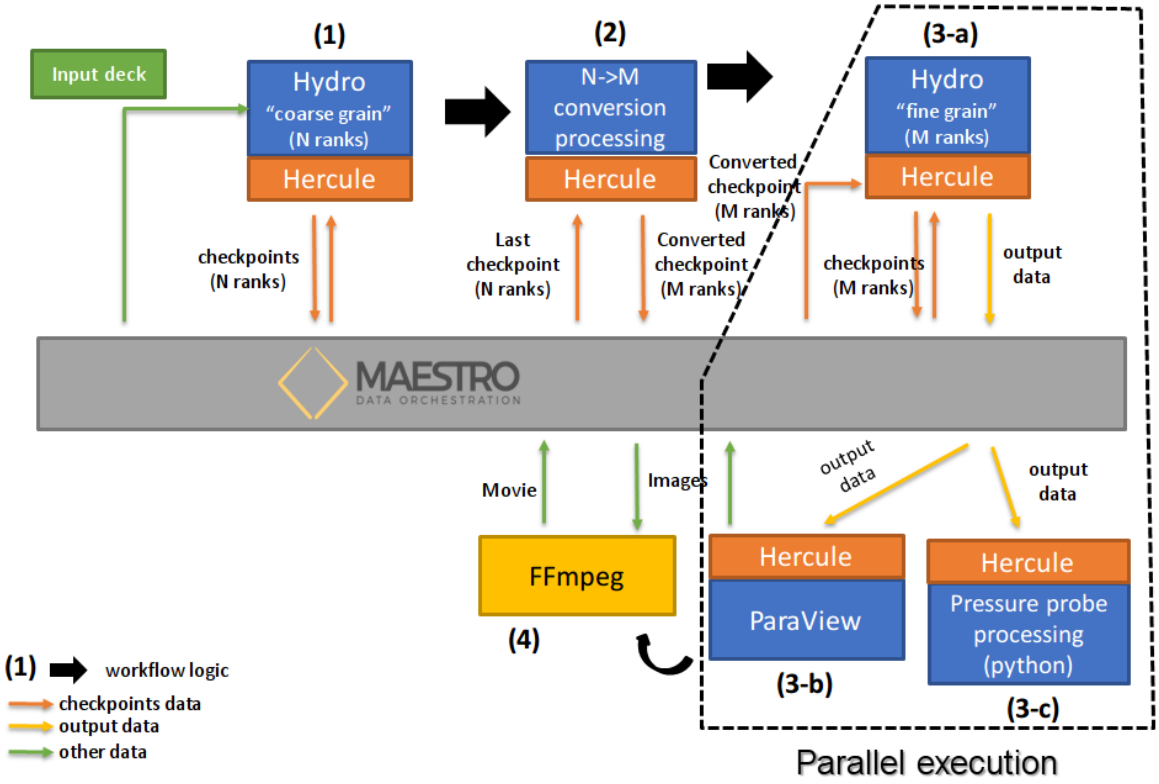
2. Computational Fluid Dynamics plus in-situ analysis

2.1 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 checkpointing data serving as data exchange between the "two" codes. Two post-processings 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.

The Maestro-enabled version will consist of 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 of executing in parallel 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:

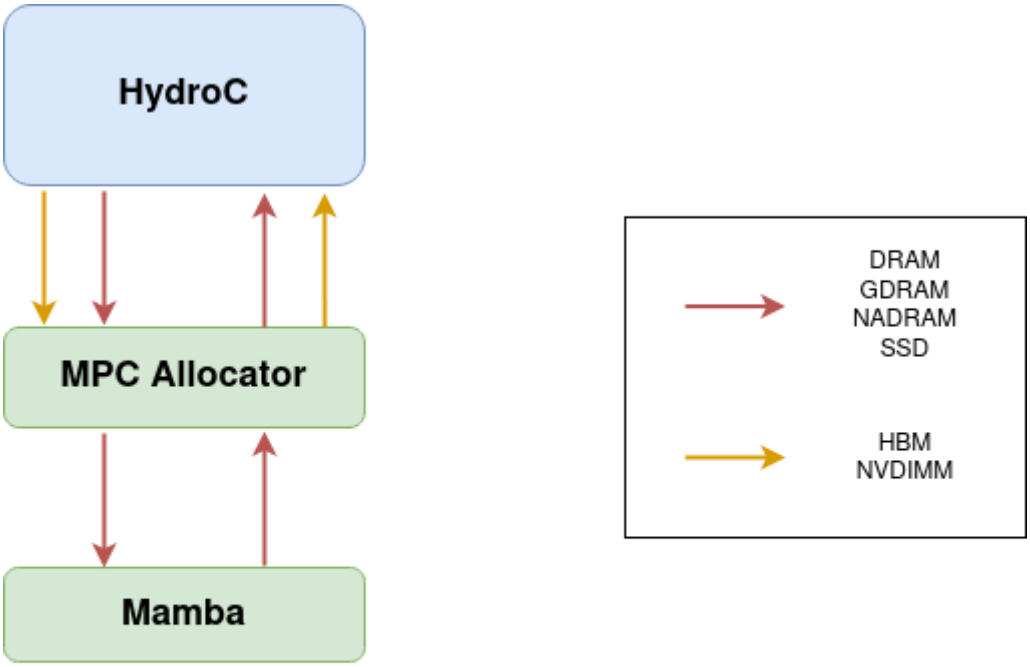


2.2 Maestro features

2.2.1 Attribute-based memory choice

The Mamba library is providing an abstract memory model which allows the programmer to transparently allocate, access, and transport data across heterogeneous memory tiers. In the case of the Data-Aware Runtime Demonstrator, the hydrodynamic application HydroC is parallelized with MPC-Halos. MPC-Halo is implemented on the top of the Maestro-core so that the application processes are communicating with Maestro CDOs.

The MPC component responsible for the allocation of the simulation data is called MPC Allocator. The idea is to make it benefit from the Mamba library as an extent of the handled memory layers: HBM and NVDIMM allocations will be performed by the MPC Allocator and DRAM, GDRAM, NADRAM, and SSD allocations by the mamba library (as shown in the figure below).



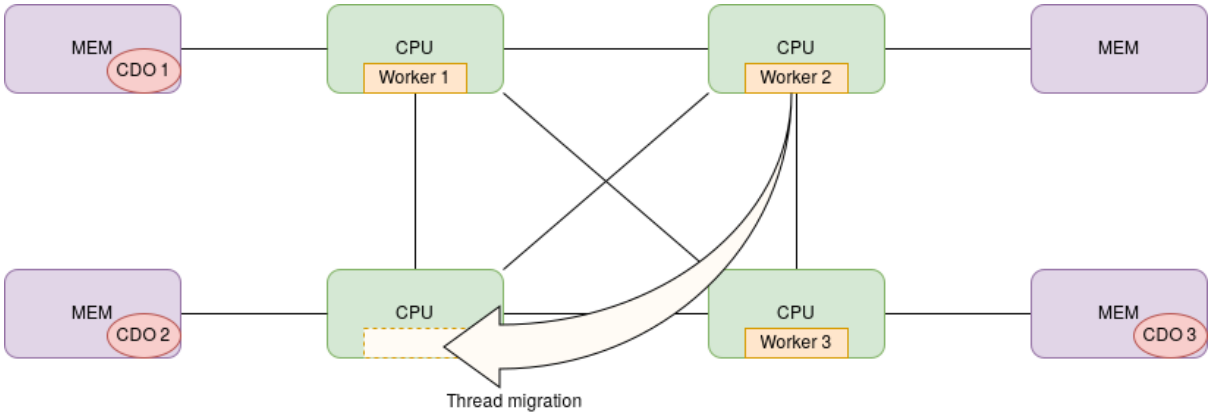
Memory allocation process

2.2.2 Data-aware task migration

This feature aims to migrate threads closer to the memory. To do so, 2 steps are performed:

1. Get the NUMA domain for each CDO
2. Thread worker should be pinned to cores within the same NUMA domain as the CDO

The figure below shows an example of data-aware task migration.



2.2.3 Guided I/O

There are 2 kinds of IO in the demonstrator: the checkpoint IO and output IO. On the one hand, the priority is very low for checkpoint IO since it is only used

to restart the demonstrator after a system crash. Yet, on the other hand, output IO is used at every simulation step for the data post-processing.

Guided I/O allows us to prioritize outputs over checkpoints and optimize the global performances of the whole demonstrator.

2.2.4 CDO data wrapping

The demonstrator will use our I/O library, Hercule, to pass 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.

User-defined attributes are used to generate custom CDOs that contain the metadata needed by Hercule.

2.3 Middleware requirements and current status

The table below shows the baseline performances. No major changes that affect performances have been performed since the D2.4 deliverable.

System	**Inti**			**Sage**		
Test case	2000x2000 ; time (end) = 5					
Nodes	1	5	10	1	3	5
Total cores	32	180	320	8	24	40
Time (sec)	150.031	59.654	49.11	163.24	83.17	73.14
Cycles (MC/s)	31.41	104.64	159.18	6.07	16.10	20.36

This is the list of the requirements provided in the deliverable "D2.3 Middleware Requirements and API Design".

Reference in D2.3	**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	Most simulation codes and tools at CEA use the in-house Hercule

	library to perform IO-related tasks. To minimize the effort to use Maestro, we intend to port Hercule on top of Maestro (using Maestro as a backend store). This is a fundamental organization of data within Hercule.
Status/feedback	*Partially validated.* Records and collections are effectively supported in the Maestro middleware with the use of groups. However, the ability to handle multi-dimensional arrays is still to be validated.

Reference in D2.3	**R2.2**
Description	Maestro must provide the ability to add (and query) named attributes (metadata) to data arrays, records, and collections, where these terms are defined in R2.1.
Validation	As the Hercule I/O library (which will use Maestro) manipulates data objects but also groups of data objects, the ability to use named attributes on data objects and groups of data objects will facilitate it.
Status/feedback	*Still to be validated.*

Reference in D2.3	**R2.3**
Description	Maestro should provide a way to hierarchically group related data arrays within a record (and query this group structure), similar to HDF5 groups.
Validation	This will assist in porting the Hercule I/O library on top of Maestro (though this hierarchical organization could be implemented in the data naming, having a native system for hierarchical grouping would be more efficient).
Status/feedback	*Still to be validated.*

Reference in D2.3	**R2.4**
Description	A data collection must accommodate records produced by multiple independent producers (see R2.1 for the definition of ‘collection’ and ‘records’). Hence, the 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	When one of the simulations writes data with the Hercule I/O library, each rank produces a record that is self-describing and contains all data and metadata produced by that rank. Unlike in other I/O libraries, these records are not combined to produce a global view of the domain (or a different domain decomposition). They are written as-is (keeping the same domain decomposition). The recombination into a different domain decomposition is deferred until reading if needed. This is the default behaviour and is part of optimisation for writing. This principle implies that records are indexed by their issuing

	sequence (here the time step number) and by the producer id (here the MPI rank). Therefore, to allow porting Hercule to leverage Maestro, it must provide an indexing or attribute system compatible with this approach.
Status/feedback	*Partially validated.* Records seem to be quite well accommodated within groups. Yet, validation is still to be performed.

Reference in D2.3	**R2.5**
Description	In ‘streaming’ mode, a data collection should accommodate multiple consumers organized into groups, similar to Kafka groups (or Redis Stream consumer groups). Records are broadcasted to all consumer groups. Consumers within a consumer group retrieve records from a queue.
Validation	This allows mixing two types of pattern: ● all records: consumers registering in different groups receive (and process) all the records, e.g. statistical processing that integrates data over time needs all records ● fair-share: consumers registering in the same group split the work by concurrently consuming records from a queue (hence a record will be processed by only one consumer within the group), e.g. stateless processing such as an image rendering can work independently on a record and does not need the whole sequence
Status/feedback	*Partially validated.* Records are broadcast to all consumers. A queue is set when multiple consumers are retrieving the same record simultaneously. However, groups of consumers are not yet validated.

Reference in D2.3	**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	An example property that could be leveraged by Maestro at configuration time would be the identifiers of data that are known to be consumed by a consumer application. Let’s say a simulation code dumps many physical quantities, but for a particular workflow, the consumer application only requires a few of them. This is a piece of information that Maestro could leverage to avoid transporting data from the producer application that will not be consumed at all.
Status/feedback	*Validated.* The Maestro middleware’s support for workflows is appropriate. For each workflow, a pool manager may be started to handle records separately.

Reference in D2.3	**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	In CEA's simulation pipelines, upstream simulation codes produce many arrays that are not necessarily consumed by downstream consuming applications. The arrays consumed is information that is often available ahead of time (e.g. the user knows that its consuming application only reads pressure fields). Maestro could leverage this knowledge to optimise data flow (filtering) between producer and consumer applications.
Status/feedback	*Validated.* Consumers have the opportunity to require records and post a demand request once the data is effectively needed.

Reference in D2.3	**R2.8**
Description	Maestro must provide the ability to persist indefinitely a data object provided that one of the storage tiers managed by Maestro is non-volatile.
Validation	Required if Maestro is to replace an existing I/O stack.
Status/feedback	*Still to be validated.* Persistency is not yet supported in the current Maestro development version.

Reference in D2.3	**R2.9**
Description	Relates to R2.8 When used for persisting data produced by an application, Maestro should provide the ability to configure which group of data objects is persisted.
Validation	An application may produce different data collections serving different purposes. The user should be able to instruct Maestro which collection needs to be persisted.
Status/feedback	*Still to be validated.* Relates to R2.8

Reference in D2.3	**R2.10**
Description	Relates to R2.8. When used for persisting data produced by an application, Maestro should provide the ability to specify a subsampling rate of persistence or filter.
Validation	This allows different branches of a pipeline to work on different data flow rates. In particular, the main simulation branch executed 'in transit' can work on a higher data flow rate to have refined data processing.
Status/feedback	*Still to be validated.* Relates to R2.8

3. Global Earth Modelling system TerrSysMP

3.1 Introduction

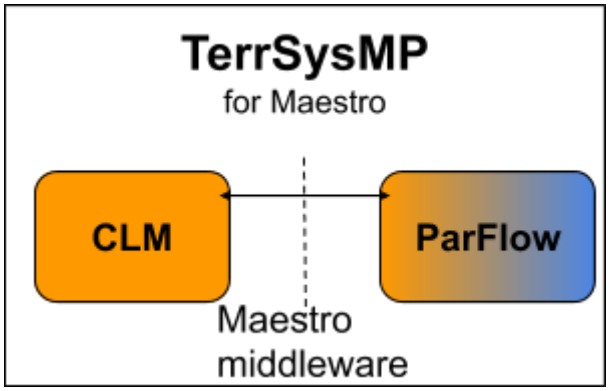
As described in D2.1, the Terrestrial Systems Modelling Platform (TerrSysMP or TSMP) targets the simulation of interactions between lateral flow processes in river basins and lower atmospheric layers. This is achieved by combining three model components COSMO, CLM, and Parflow. The coupling is done using 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.

3.2 Maestro features

Given the highly optimized and fine-tuned nature of the OASIS-MCT3 coupler, we do not expect any performance benefit from porting the application to Maestro. The objective instead is to show the suitability of Maestro's general approach as a coupler for the applications which currently used a specialized coupler. As such, the objective is to at least achieve comparable performance with the OASIS-MCT3 coupler.

3.2.1 Transparent Transformation of 2D Fields

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.



3.2.2 Load Balancing and Redistribution

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.

3.3 Covered Middleware Requirement and Current Status

Reference in D2.1	**R4.1**
Description	**(R4.1)** Redistribution of parallel 2D fields
Validation	Still to be validated.

Reference in D2.1	**R4.2**
Description	**(R4.2)** Row and column-major layout transformation of distributed arrays
Validation	Still to be validated.

4. Electronic structure calculation library SIRIUS

4.1 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 linear algebra operations. They are usually compute-intensive and thus are the perfect candidates for GPU acceleration. However, the matrices involved in the calculation are usually large and do not always fit into the memory of a device. This implies significant data movements between CPU and GPU memories.

4.2 Maestro features

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.

4.2.1 Move data from CPU to GPU memory

When the data cannot fit into GPU memory, allocation/deallocation is currently manually done by blocks, meaning that the memory management is fully controlled by the developer. This memory management is implemented in a dedicated library within SIRIUS. The data movements are synchronous. A GPU kernel cannot start until all the data is available on the device. It is also important to note that the source of the data is not in use during the transfer. In addition, the host memory always stores the correct and valid data arrays.

This use case goal is to replace the smartness of moving data created by a developer of SIRIUS with Maestro's capabilities to handle CPU-GPU data movements using a generic abstraction. In that way, it simplifies the code of SIRIUS or any other applications using Maestro and enables such applications to delegate the smartness and portability among different GPU hardware of data movements to Maestro.

In terms of workflow, SIRIUS creates the initial dataset in the host memory and gives the dataset ownership to Maestro. Later on, SIRIUS requests such data to Maestro in the GPU memory. Maestro is responsible to manage the transfer of the data from the CPU to the GPU to fulfil the request. Once the data is on the GPU, the ownership is transferred back to SIRIUS.

4.2.2 Management of memory resources

Copying data to the GPU memory for computation implies repeatedly allocating and deallocating GPU memory. The same behaviour is also observed on host memory. As the frequency of allocation and deallocation is correlated to the number of steps required for computation like described in UC3.1, a performance overhead is observed when the number of memory operations performed on both memories is high. A memory pool where allocated memory spaces are re-used is necessary to mitigate this overhead. Maestro should provide a similar capability.

4.3 Middleware Requirement and Current Status

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 ownership on it. That way, Maestro will be able to take data movement decisions from host to GPU memory or the opposite way.
Status/feedback	*Present in Maestro, need to be validated.*
Evaluation metric	*Semi-automatic data partitioning.*

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.
Status/feedback	*Present in Maestro, need to be validated.*
Evaluation metric	*Semi-automatic data partitioning.*

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 afterward.
Status/feedback	*Present in Maestro, need to be validated.*
Evaluation metric	*Semi-automatic data partitioning.*

Reference in D2.1	R3.4
Description	Maestro can 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.
Status/feedback	*Present in Maestro, need to be validated. Provided by the Mamba library.*
Evaluation metric	*Automatic memory pool management*

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.
Status/feedback	*Present in Maestro, need to be validated.*
Evaluation metric	*Semi-automatic data partitioning.*

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.
Status/feedback	*Present in Maestro, need to be validated. Provided by the Mamba library.*
Evaluation metric	*Automatic memory pool management*

5.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 and requirements as detailed in a previous deliverable (D6.2).

The current status for each application demonstrator can be summarized in the following figure: