Optimizing the Destination Earth Workflow with *in situ* HPC Task Orchestration

Pablo Goitia, Manuel Giménez and Miguel Castrillo
Barcelona Supercomputing Center (BSC), Spain

Funded by the European Union
Destination Earth
implemented by ECMWF esa EUMETSAT

EGU26 GENERAL ASSEMBLY

1 Motivation

- Supercomputing resources are limited and platforms are frequently congested.
- There is an active concern within the community about the long queue times that simulations have to face because of this.
- A research made in the top of the results of the CMIP6 project, estimated **the queue time overhead between 10% and 20%** of the total execution time of the simulations[1].

```
$ sacct --format=JobName,submit,start                                (simplified view)

JobName                                 Submit              Start
--------------------------------------- ------------------- -------------------
a3b0_20200101_fc2_22_20_APP_HYDROLAND   2026-03-09T16:34:17 2026-03-09T16:34:40
a3b0_20200101_fc3_22_28_APP_HYDROLAND   2026-03-09T16:34:17 2026-03-09T16:34:40
a3b0_20200101_fc1_22_25_APP_HYDROLAND   2026-03-09T16:40:18 2026-03-09T19:39:26
a3b0_20200101_fc2_22_21_APP_HYDROLAND   2026-03-09T16:40:18 2026-03-09T19:39:26
a3b0_20200101_fc3_22_29_APP_HYDROLAND   2026-03-09T16:40:18 2026-03-09T19:39:26
a3b0_20200101_fc1_22_26_APP_HYDROLAND   2026-03-09T19:46:07 2026-03-09T19:55:09
a3b0_20200101_fc2_22_22_APP_HYDROLAND   2026-03-09T19:47:19 2026-03-09T19:55:09
```

[1] Acosta, M. C., Palomas, S., Paronuzzi Ticco, S. V., Utrera, G., Biercamp, J., Bretonniere, P.-A., Budich, R., Castrillo, M., Caubel, A., Doblas-Reyes, F., Epicoco, I., Fladrich, U., Joussaume, S., Kumar Gupta, A., Lawrence, B., Le Sager, P., Lister, G., Moine, M.-P., Rioual, J.-C., Valcke, S., Zadeh, N., and Balaji, V.: *The computational and energy cost of simulation and storage for climate science: lessons from CMIP6*, Geosci. Model Dev., 17, 3081–3098, https://doi.org/10.5194/gmd-17-3081-2024, 2024.

1 Motivation

- As a response to this concern, Giménez de Castro et al.[1] explored the **task aggregation** technique to reduce the total amount of time spent in the queues by the simulations.

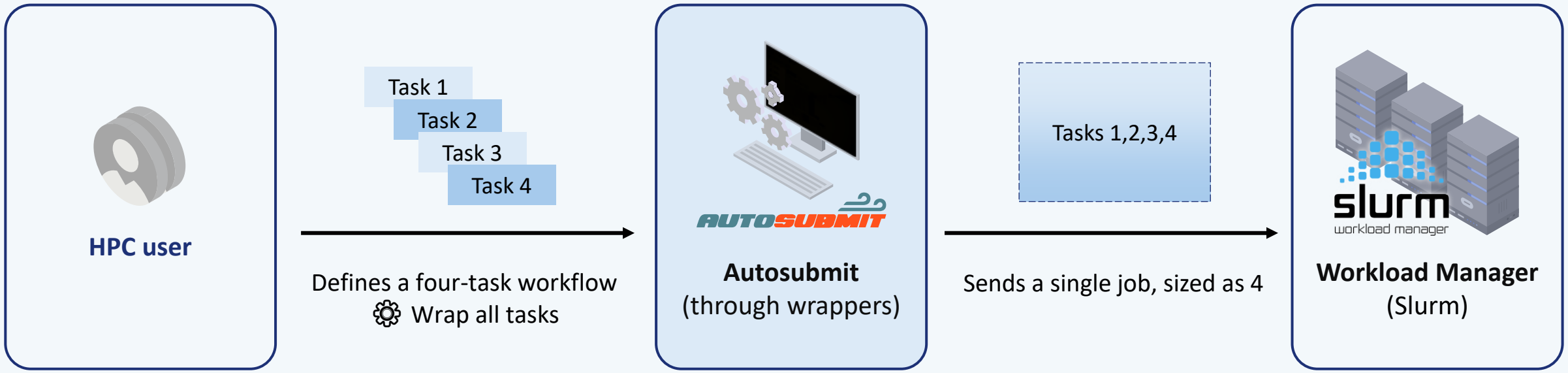
Task aggregation consists of sending multiple tasks grouped together as if they were a single job to the remote scheduler.

- The theoretical study states that **vertical task aggregation is effective,** reducing the time-to-solution up to 7%.
- Since ESMs are typically arranged in vertical structures, Goitia et al.[2] validated its effectiveness in real life climate simulations by running ESM workflows with Autosubmit in multiple supercomputers.
- This practical assessment concluded that **queue times were reduced by up to 12 times**, increasing the workflow performance (ASYPD) by up to 23% in the case of the most congested platform.

[1] Marciani, M. G., Castrillo, M., Utrera, G., Acosta, M. C., Kinoshita, B. P., and Doblas-Reyes, F.: *Evaluating the impact of task aggregation in workflows with shared resource environments: use case for the MONARCH application*, Geosci. Model Dev., 18, 9709–9721, https://doi.org/10.5194/gmd-18-9709-2025, 2025.
[2] Goitia, P., Marciani, M. G. and Bosque, J.L.: *Autosubmit wrappers to speed up scientific production*, Universidad de Cantabria, https://repositorio.unican.es/xmlui/handle/10902/33831, 2024.

1 Motivation

How task aggregation works in Autosubmit (visual example)

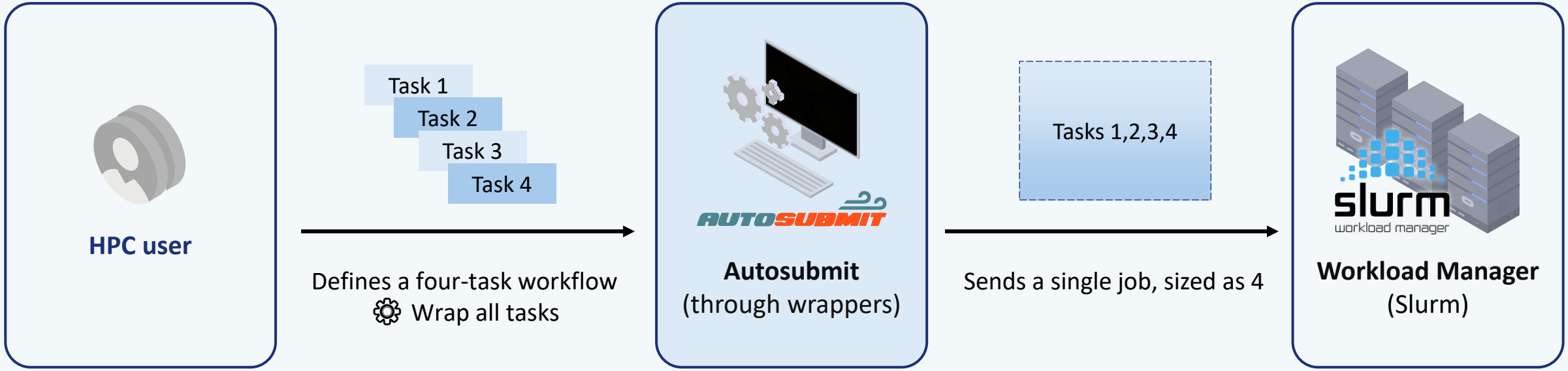


Wrappers in Autosubmit are built in runtime, depending on the **wrapping policy**, the **wrapping method**, and the **wrapper type**.

1 Motivation

The problem with the current implementation

- In the Autosubmit workflow manager, the *wrapper* implementation is **complex** and **limited**.
  - Different running strategies per wrapper type, complex resource management, maintenance challenges.
- "Externalizing" the remote wrapper management would abstract the developers from the complexity of the implementation.



2 Objective

To integrate an HPC tool that can serve as an *in situ* wrapper manager into Autosubmit
to overcome the limitations that default methods for aggregating tasks present by offering better maintainability, scalability, portability, and straightforward resource management.

An *in situ* workflow manager is a tool that manages the execution of a set of tasks inside a resource allocation in a cluster.

3 The *in situ* workflow engine

- Different HPC tools have been evaluated, including WMSs (mainly distributed), pilot systems, tools or libraries for task parallelism, and workload managers.
- The analysis revealed that maintaining a compromise among all initially defined requirements is complex.
- From all the options, Flux emerged as the preferred option.

Why Flux?

It represents an opportunity to integrate state-of-the-art technology into Autosubmit.

Developed at LLNL.

Clear delimitation of resources for aggregations within the same allocation is critical.

Straightforward error handling mechanisms.

Flux is already present in outstanding supercomputers.

Its hierarchical architecture, allows to execute instances even within allocations of other schedulers.

4 Feature design

High-level conceptualization

Outcome: Flux to be fully integrated into Autosubmit as an *in situ* wrapper engine (by now, just for Slurm systems).

Autosubmit side

Autosubmit's main loop does:

1. Checks if some wrapper can be built in the current iteration.
2. Calculates wrapper bounds based on the type of wrapper and the policy.
3. Builds a sorted job list by execution order.
4. Those job scripts are translated into Flux jobspec files.

Then, the Flux wrapper builder:

Builds the script that coordinates the execution of the tasks in the wrapper job list, based on the type of aggregation.

The main loop sends the wrapper script to the HPC, and keeps monitoring the inner jobs.

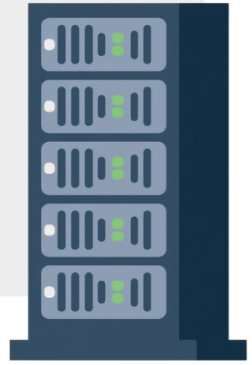
HPC allocation side

The wrapper script:

1. Initiates the Flux instance.
2. Submits the jobs orderly.
3. Waits for each job to finish and checks its status {COMPLETED, FAILED}.
4. Terminates wrapper execution and Flux instance.

Meanwhile, Flux in the background:

Schedules, executes and keeps track of the jobs, which are characterized by the data in the jobspec.



4 Feature design

High-level conceptualization

Outcome: Flux to be fully integrated into Autosubmit as an *in situ* wrapper engine (by now, just for Slurm systems).

Autosubmit side

Autosubmit's main loop does:

1. Checks if some wrapper can be built in the current iteration.
2. Calculates wrapper bounds based on the type of wrapper and the policy.
3. Builds a sorted job list by execution order.
4. Those job scripts are translated into Flux jobspec files.

Then, the Flux wrapper builder:

Builds the script that coordinates the execution of the tasks in the wrapper job list, based on the type of aggregation.

The main loop sends the wrapper script to the HPC, and keeps monitoring the inner jobs.

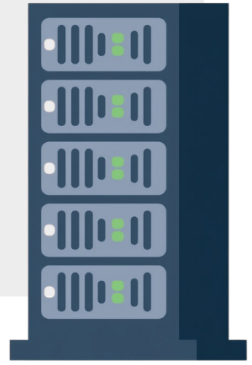
HPC allocation side

The wrapper script:

1. Initiates the Flux instance.
2. Submits the jobs orderly.
3. Waits for each job to finish and checks its status {COMPLETED, FAILED}.
4. Terminates wrapper execution and Flux instance.

Meanwhile, Flux in the background:

Schedules, executes and keeps track of the jobs, which are characterized by the data in the jobspec.



4 Feature design

High-level conceptualization

Outcome: Flux to be fully integrated into Autosubmit as an *in situ* wrapper engine (by now, just for Slurm systems).

Autosubmit side

Autosubmit's main loop does:

1. Checks if some wrapper can be built in the current iteration.
2. Calculates wrapper bounds based on the type of wrapper and the policy.
3. Builds a sorted job list by execution order.
4. Those job scripts are translated into Flux jobspec files.

Then, the Flux wrapper builder:

Builds the script that coordinates the execution of the tasks in the wrapper job list, based on the type of aggregation.

The main loop sends the wrapper script to the HPC, and keeps monitoring the inner jobs.

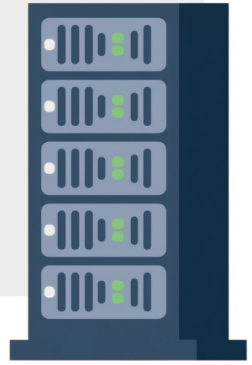
HPC allocation side

The wrapper script:

1. Initiates the Flux instance.
2. Submits the jobs orderly.
3. Waits for each job to finish and checks its status {COMPLETED, FAILED}.
4. Terminates wrapper execution and Flux instance.

Meanwhile, Flux in the background:

Schedules, executes and keeps track of the jobs, which are characterized by the data in the jobspec.



4 Feature design

High-level conceptualization

Outcome: Flux to be fully integrated into Autosubmit as an *in situ* wrapper engine (by now, just for Slurm systems).

Autosubmit side

Autosubmit's main loop does:

1. Checks if some wrapper can be built in the current iteration.
2. Calculates wrapper bounds based on the type of wrapper and the policy.
3. Builds a sorted job list by execution order.
4. Those job scripts are translated into Flux jobspec files.

Then, the Flux wrapper builder:

Builds the script that coordinates the execution of the tasks in the wrapper job list, based on the type of aggregation.

The main loop sends the wrapper script to the HPC, and keeps monitoring the inner jobs.

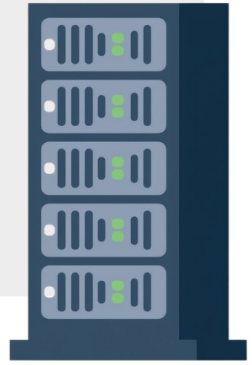
HPC allocation side

The wrapper script:

1. Initiates the Flux instance.
2. Submits the jobs orderly.
3. Waits for each job to finish and checks its status {COMPLETED, FAILED}.
4. Terminates wrapper execution and Flux instance.

Meanwhile, Flux in the background:

Schedules, executes and keeps track of the jobs, which are characterized by the data in the jobspec.



4 Feature design

High-level conceptualization

Outcome: Flux to be fully integrated into Autosubmit as an *in situ* wrapper engine (by now, just for Slurm systems).

Autosubmit side

Autosubmit's main loop does:

1. Checks if some wrapper can be built in the current iteration.
2. Calculates wrapper bounds based on the type of wrapper and the policy.
3. Builds a sorted job list by execution order.
4. Those job scripts are translated into Flux jobspec files.

Then, the Flux wrapper builder:

Builds the script that coordinates the execution of the tasks in the wrapper job list, based on the type of aggregation.

The main loop sends the wrapper script to the HPC, and keeps monitoring the inner jobs.

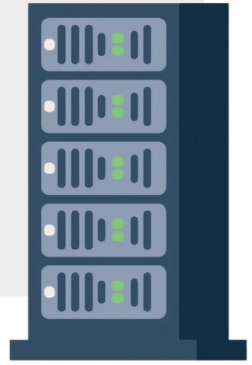
HPC allocation side

The wrapper script:

1. Initiates the Flux instance.
2. Submits the jobs orderly.
3. Waits for each job to finish and checks its status {COMPLETED, FAILED}.
4. Terminates wrapper execution and Flux instance.

Meanwhile, Flux in the background:

Schedules, executes and keeps track of the jobs, which are characterized by the data in the jobspec.



4 Feature design

High-level conceptualization

Outcome: Flux to be fully integrated into Autosubmit as an *in situ* wrapper engine (by now, just for Slurm systems).

Autosubmit side

Autosubmit's main loop does:

1. Checks if some wrapper can be built in the current iteration.
2. Calculates wrapper bounds based on the type of wrapper and the policy.
3. Builds a sorted job list by execution order.
4. Those job scripts are translated into Flux jobspec files.

Then, the Flux wrapper builder:

Builds the script that coordinates the execution of the tasks in the wrapper job list, based on the type of aggregation.

The main loop sends the wrapper script to the HPC, and keeps monitoring the inner jobs.

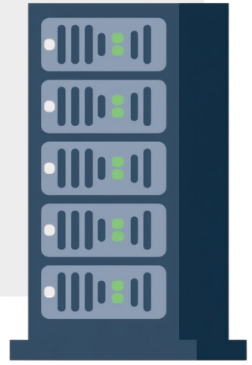
HPC allocation side

The wrapper script:

1. Initiates the Flux instance.
2. Submits the jobs orderly.
3. Waits for each job to finish and checks its status {COMPLETED, FAILED}.
4. Terminates wrapper execution and Flux instance.

Meanwhile, Flux in the background:

Schedules, executes and keeps track of the jobs, which are characterized by the data in the jobspec.



4 Feature design

High-level conceptualization

Outcome: Flux to be fully integrated into Autosubmit as an *in situ* wrapper engine (by now, just for Slurm systems).

Autosubmit side

Autosubmit's main loop does:

1. Checks if some wrapper can be built in the current iteration.
2. Calculates wrapper bounds based on the type of wrapper and the policy.
3. Builds a sorted job list by execution order.
4. Those job scripts are translated into Flux jobspec files.

Then, the Flux wrapper builder:

Builds the script that coordinates the execution of the tasks in the wrapper job list, based on the type of aggregation.

The main loop sends the wrapper script to the HPC, and keeps monitoring the inner jobs.

HPC allocation side

The wrapper script:

1. Initiates the Flux instance.
2. Submits the jobs orderly.
3. Waits for each job to finish and checks its status {COMPLETED, FAILED}.
4. Terminates wrapper execution and Flux instance.

Meanwhile, Flux in the background:

Schedules, executes and keeps track of the jobs, which are characterized by the data in the jobspec.

4 Feature design

High-level conceptualization

Outcome: Flux to be fully integrated into Autosubmit as an *in situ* wrapper engine (by now, just for Slurm systems).

Autosubmit side

Autosubmit's main loop does:

1. Checks if some wrapper can be built in the current iteration.
2. Calculates wrapper bounds based on the type of wrapper and the policy.
3. Builds a sorted job list by execution order.
4. Those job scripts are translated into Flux jobspec files.

Then, the Flux wrapper builder:

Builds the script that coordinates the execution of the tasks in the wrapper job list, based on the type of aggregation.

The main loop sends the wrapper script to the HPC, and keeps monitoring the inner jobs.

HPC allocation side

The wrapper script:

1. Initiates the Flux instance.
2. Submits the jobs orderly.
3. Waits for each job to finish and checks its status {COMPLETED, FAILED}.
4. Terminates wrapper execution and Flux instance.

Meanwhile, Flux in the background:

Schedules, executes and keeps track of the jobs, which are characterized by the data in the jobspec.

5 Testing the new implementation

- It is necessary to **validate** the implementation. This was done by running the Climate DT workflow (the **digital twin for the climate change adaptation**) with **Autosubmit**.
- Three workflows were executed concurrently: the standard (non-wrapped), Autosubmit's default wrappers, and Flux wrappers.
- This extensive workflow consists of all the tasks required to initialize and execute the model, and to post-process its outputs.
- In this case, only the post-processing was executed, with multiple applications enabled.
- There is a time gap between the simulation and the post-process in which **the queue time is partially responsible**.
- Running Climate DT with this new wrapping method aims both to validate the implementation and to provide a basis for future optimizations of the workflow.

6 Conclusions

Results of the validation

- Wrapped workflows achieved a **2.2× of speed-up**.
- The new Flux method does **not introduce workflow management overhead**.
- The performance difference between the new Flux method and the default is **negligible**, of only 0.78% in favor of the default.

Distribution of wrapper sizes (exploratory analysis)

- The efficiency of the scheduler in dispatching wrappers depends on factors such as the temporal size of the wraper, the workload of the machine, and the topology of the workflow.
- **The selected wrapping configuration was effective**: approximately a 90% of the wrappers were dispatched quickly.

Optimizing the Destination Earth Workflow with *in situ* HPC Task Orchestration

Pablo Goitia, Manuel Giménez and Miguel Castrillo
Barcelona Supercomputing Center (BSC), Spain

Thanks for attending. Questions?

pablo.goitia@bsc.es

Funded by the European Union **Destination Earth**

implemented by ECMWF esa EUMETSAT

EGU26 GENERAL ASSEMBLY

Background image from https://destination-earth.eu/