



PROGRAMME
DE RECHERCHE
NUMÉRIQUE
POUR L'EXASCALE



Deisa

Dask-enabled **in situ** analytics

Analyze your MPI Simulation Outputs with Dask



Maison
de la
Simulation



Benoît Martin

Julien Bigot

Amal Gueroudji

Bruno Raffin

HPC Numerical simulations

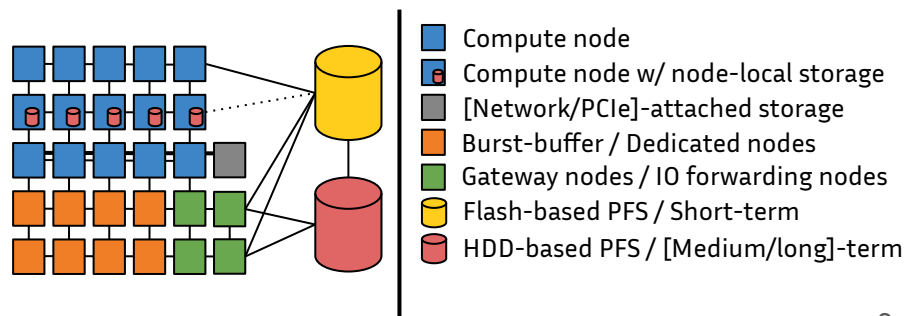
- Typically numerical simulation
 - No data analysis
 - Number crunching
- Written in Fortran/C++
 - MPI for parallelization over multiple nodes
 - OpenMP for shared memory parallelism
 - sometimes GPU
- Iterate over time
- Manipulate very structured data
 - multi-dimensional arrays
 - compute next state from time-step to time-step



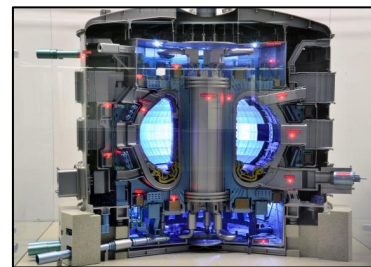
Data at exascale: a challenge in hardware

- Increasing **gap between compute and I/O** performance on large-scale systems
 - Ratio of I/O to computing power divided by ~ 10 over the last 10 years on the top 3 supercomputers
- ... and data deluge!
 - At NERSC, **data volume x41** in 10 years
- New storage tiers and advanced architectures to try to mitigate this increasing bottleneck
 - More complex on-node memory layout
 - Emerging complex applications and workflows have to adapt

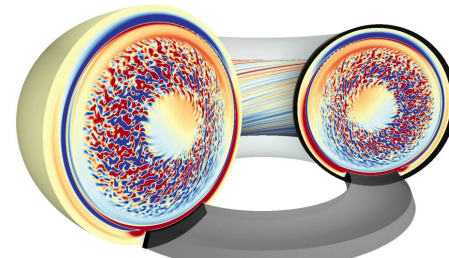
Ratio of I/O bandwidth (GBps) / TFlops of the top 3 of the Top500



The example of GYSELA

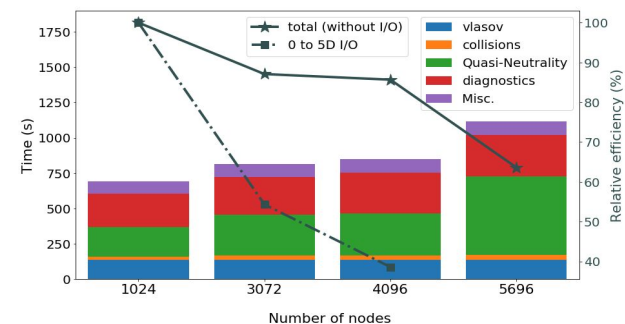


ITER project



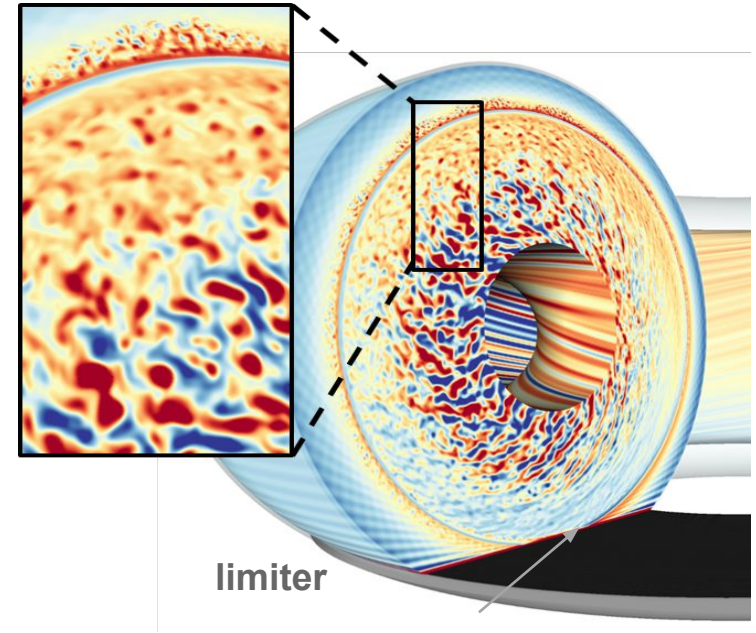
GYSELA simulation

- A gyrokinetic code for plasma simulation
 - Developed @ CEA/DRF/IRFM (lead developer: Virginie Grandgirard)
 - Non-linear 5D simulations (3D in space + 2D in velocity) + multiscale problem in space and time
 - **requires state-of-the-art HPC** techniques (~10k to ~100k CPUs)
- **Fortran 90** code with **hybrid MPI/OpenMP** parallelisation
 - Optimized up to 1,460,000 threads
 - Relative efficiency of **85% on more than 1M threads**
 - **63% on 1.46M threads**
 - on CEA-HF (AMD EPYC 7763)
- **Intensive use of petascale resources:**
 - **~ 150M hour.core / year**
 - GENCI + PRACE + HPC Fusion resources



Diagnostics in GYSELA

- In the code (in Fortran)
 - Reduce data from 5D to 3D, 2D, 1D, 0D...
 - To a single node each
- Write the result to files
 - HDF5
- Analyze the files post hoc
 - In python
 - Interactively
 - FFTs, more reductions, combining data
 - generating graphs, images, videos, ...



In 2020, a new diagnostic?

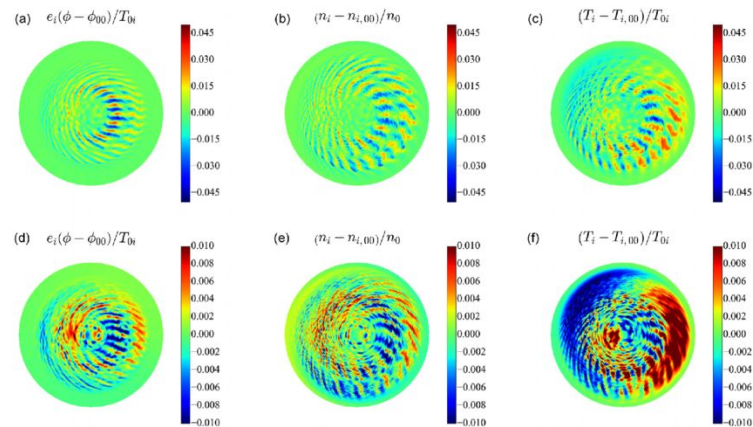
Principal Component Analysis computation on 5D distribution function

- Yuuichi Asahi et al.
- Done on GT5D code

Asahi, Yuuichi & Fujii, Keisuke & Heim, Dennis & Maeyama, Shinya & Garbet, Xavier & Grandgirard, Virginie & Sarazin, Yanick & Dif-Pradalier, G. & Idomura, Yasuhiro & Yagi, Masatoshi. (2021). Compressing the time series of five dimensional distribution function data from gyrokinetic simulation using principal component analysis. Physics of Plasmas. 28. 012304. 10.1063/5.0023166.

Hard to implement in C++/Fortran + MPI + OpenMP

- Parallel PCA already available in Scikit-learn
- \Rightarrow Let's reuse it!



Post hoc data analytics with python

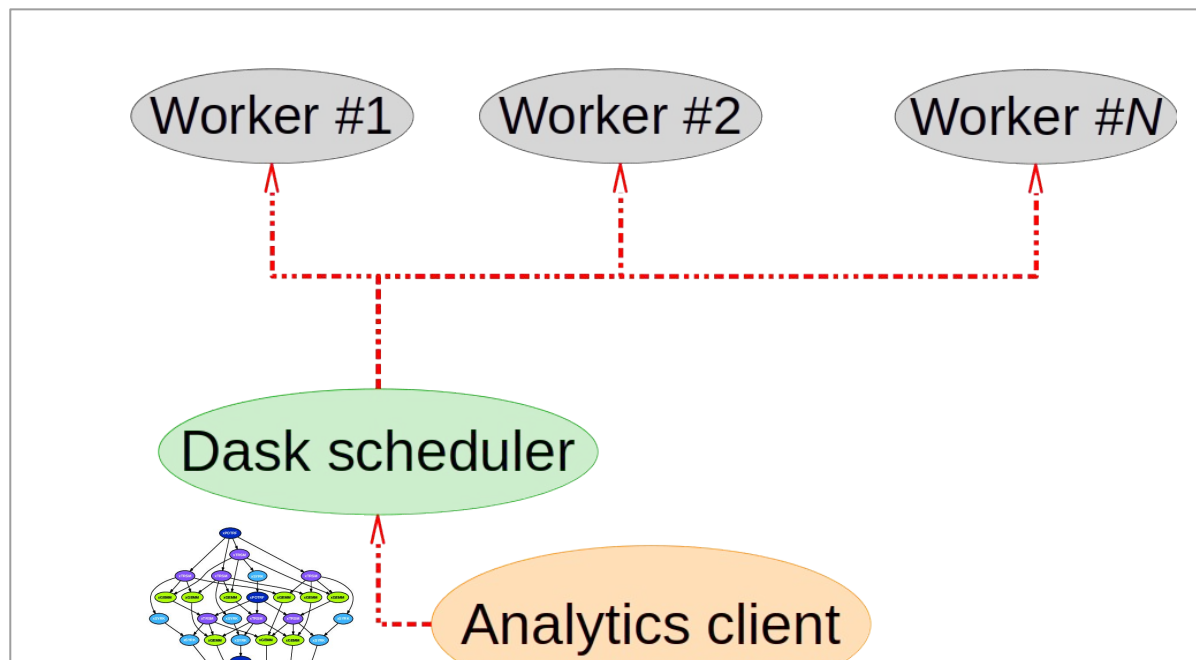
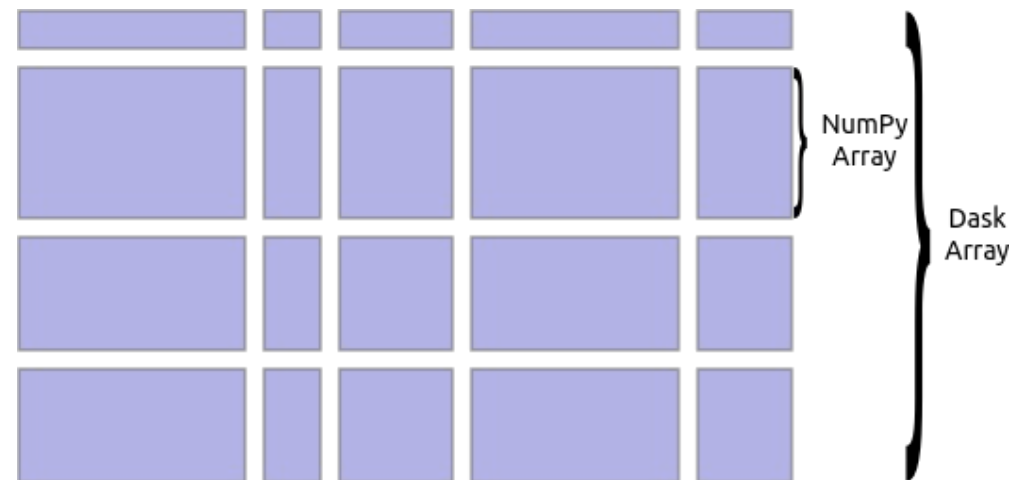
```
1 from sklearn.decomposition import IncrementalPCA
2 import yaml, json
3 import h5py
4 # load the simulation configuration
5 simu = yaml.load(open('simulation.yml'))
6 # Load data from HDF5
7 gtemp = h5py.File('data.hdf5', mode='r')['gtemp']
8 # process each time-step independently
9 for step in range(0, simu['timesteps']):
10     pca = IncrementalPCA(n_components=2, copy_solver='randomized')
11     pca.fit(gtemp[step, :, :])
12     print(pca.explained_variance_)
```



Requires a single node
computer with ~100TB RAM

Dask

- Task-based model
- A scheduler/workers (+client) model
- Dask array: distributed Numpy array
- Many tools ported to Dask
 - Numpy / SciPy
 - Scikit-learn
 - Pandas
 - ...

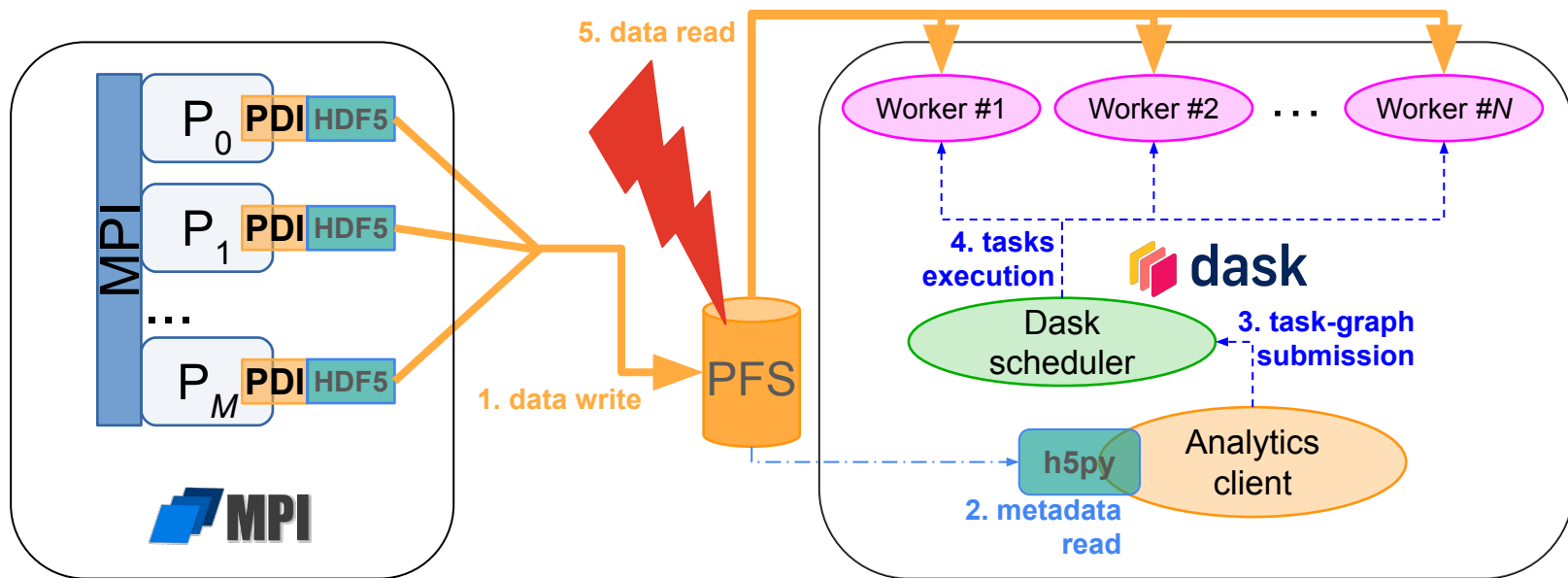


Post hoc data analytics with Dask

```
1 import dask.array as da
2 from dask_ml.decomposition import IncrementalPCA
3 import yaml, json
4 import h5py
5 # Connect to Dask
6 sched = json.load(open('sched.json'))
7 client = dask.distributed.Client(sched["address"])
8 # load the simulation configuration
9 simu = yaml.load(open('simulation.yml'))
10 # Build a lazy array descriptor from HDF5
11 gtemp = h5py.File('data.hdf5', mode='r')['gtemp']
12 gtemp = da.from_array(gtemp, chunks=(1, 4096, 4096))
13 for step in range(0, simu['timesteps']):
14     pca = IncrementalPCA(n_components=2, copy=False,
15                          svd_solver='randomized')
16     pca.fit(gtemp[step, :, :])
17     print(pca.explained_variance_)
```



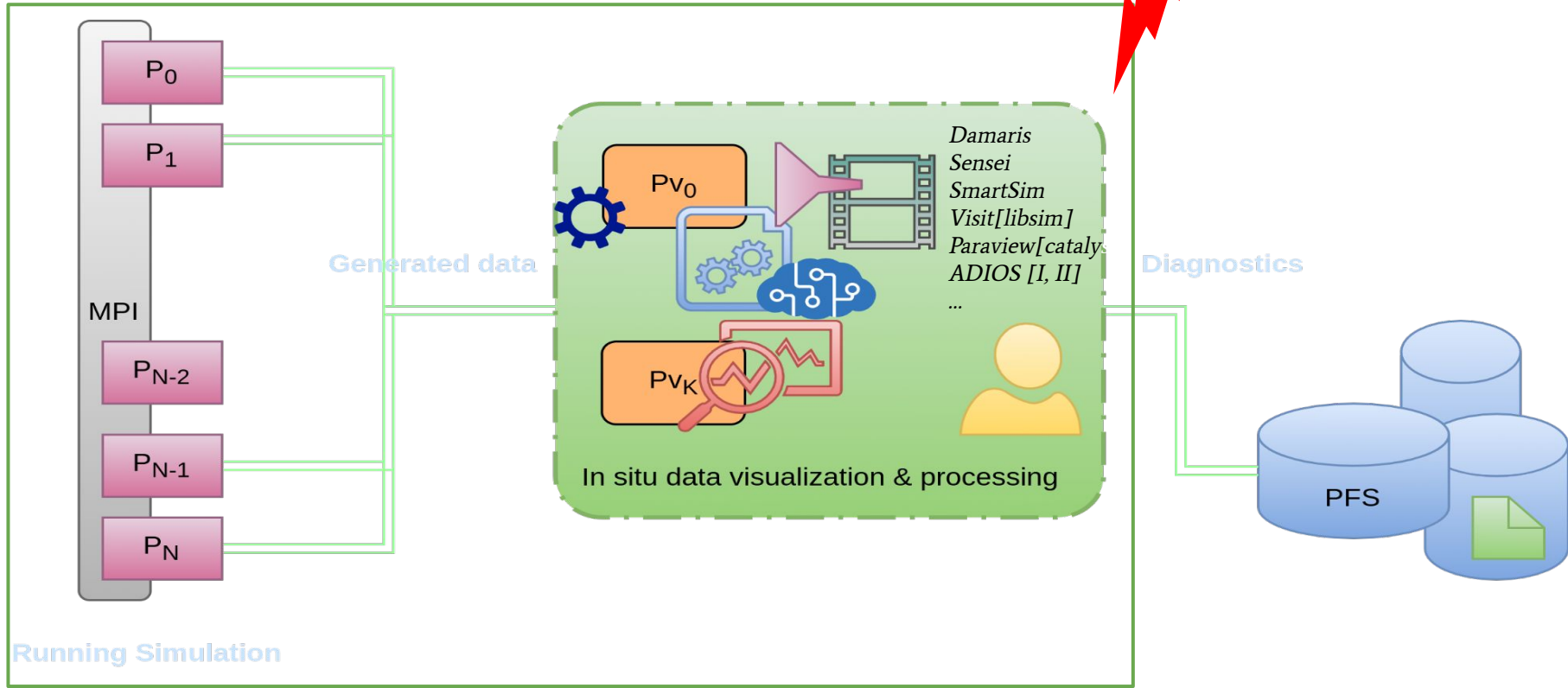
Dask for post hoc analytics



- File-system requirements are huge
 - Let's run simulation & analysis at the same time
 - Erase files as soon as they are not required anymore
- File-system IO performance is still an issue

Can we do better? In situ analytics

Usually MPI-based
Complex to setup



Can we do **even** better? Deisa!

General context

- Python analytics are nice and **many tools are available** :)
 - Dask offers a **great parallel task-based programming model** :)
 - But file-system **performance is a bottleneck** :/
- In situ analytics **solve performance issues** :)
 - Typically close to the application (MPI) programming mode
 - MPI is **not well suited for writing data analytics** :/

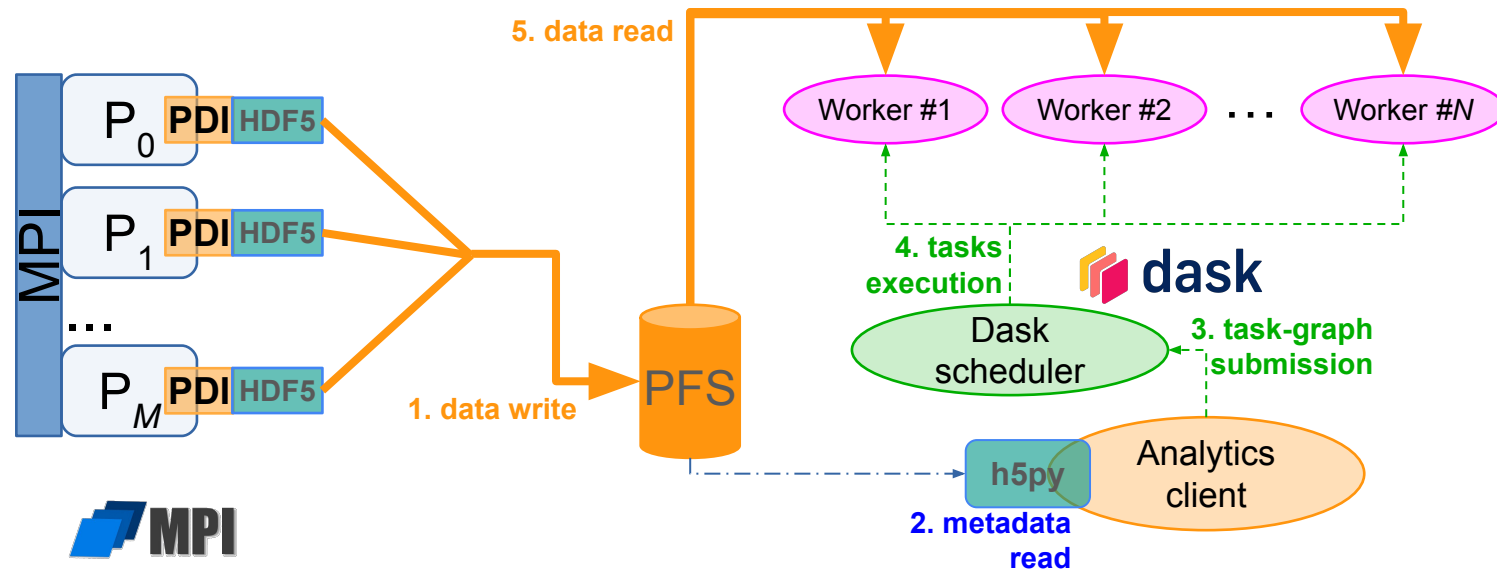
Let's combine these!

Dask-Enabled In Situ Analytics

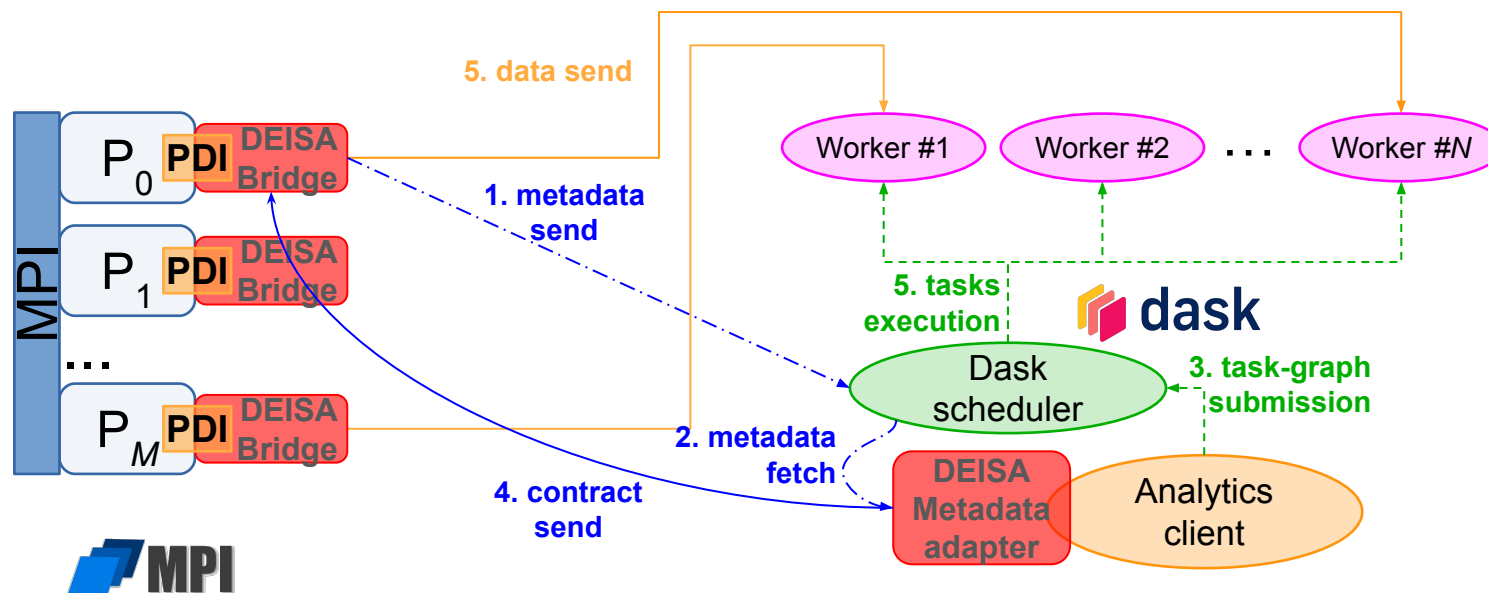
- PhD work by **Amal Gueroudji**, advised by J. Bigot & B. Raffin

Amal Gueroudji. Distributed Task-Based In Situ Data Analytics for High-Performance Simulations. Université Grenoble Alpes [2020-..], 2023. English.

Dask for post hoc analytics



Deisa



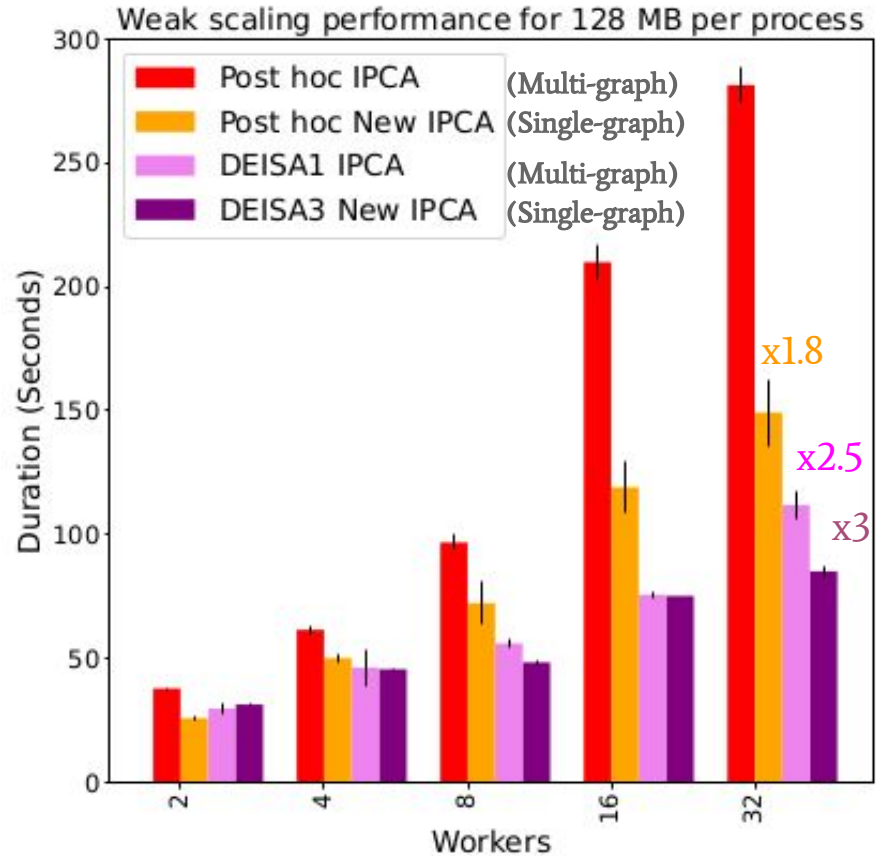
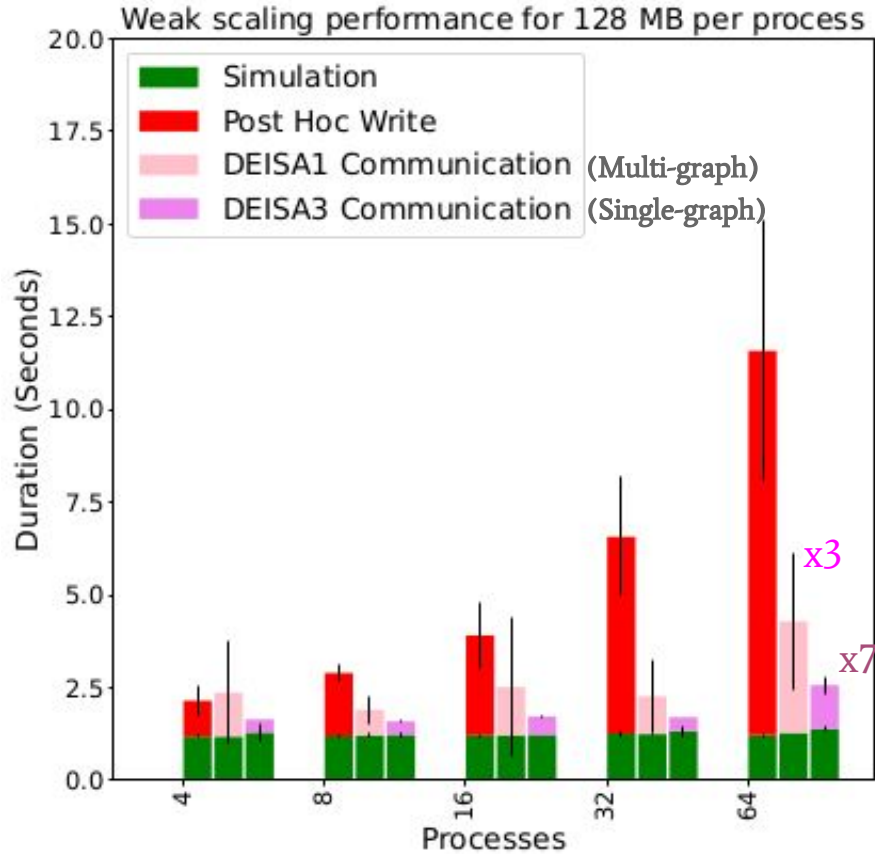
Performance evaluation

- IRENE supercomputer @ TGCC
- Nodes:
 - 2x24-cores Intel Skylake@2.7GHz
 - 180GB RAM
- InfiniBand network (100Gb/s)
- Scratch disks: 300GB/s transfer rate
- Mini App 2D heat solver

Parameter	Value
Number of runs	3
Number of iterations IPCA	10
Number of iteration Derivative	12
MPI nodes / Dask worker node	2
MPI process / MPI node	2
Dask worker / Dask worker node	2
Thread / Dask worker	24
MPI process / Dask worker	2

Configuration	XP1:128 MiB	XP1:256 MiB	XP1:512 MiB	XP1:1 GiB
MPI block size	128	256	512	1
Dask chunk size	128	256	512	1
MPI Nodes	[4, 8, 16, 32, 64, 128, 256]			
Dask Nodes	[2, 4, 8, 16, 32, 64, 128]			

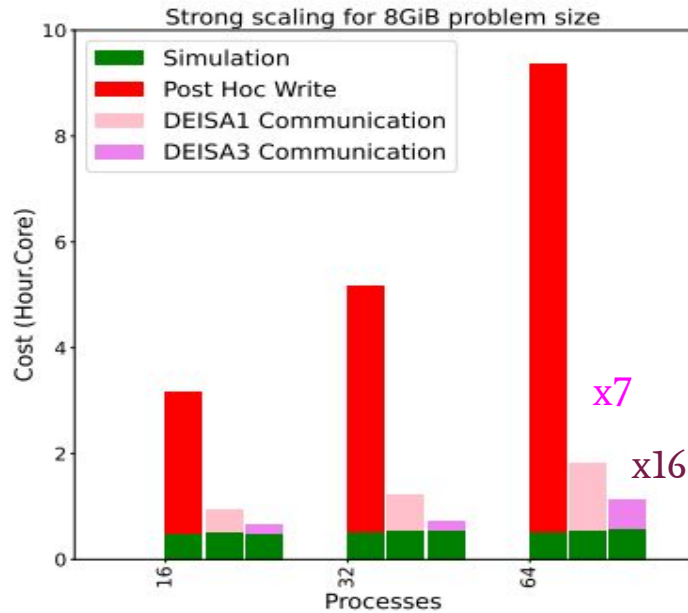
DEISA vs Post hoc Weak Scalability



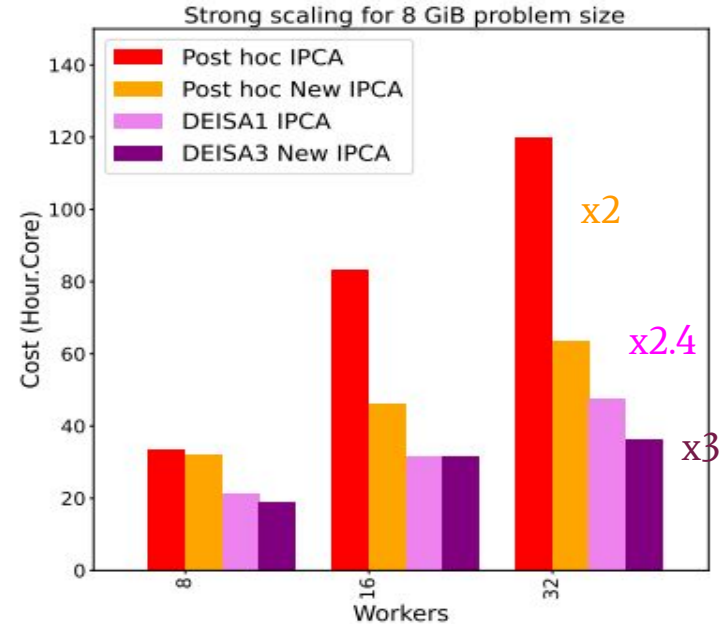
(reading data
+ Analytics)

(waiting data
+ Analytics)

DEISA vs Post hoc efficiency in hour.core



(c) Strong scaling results represented in hour-core for an 8 GiB problem size

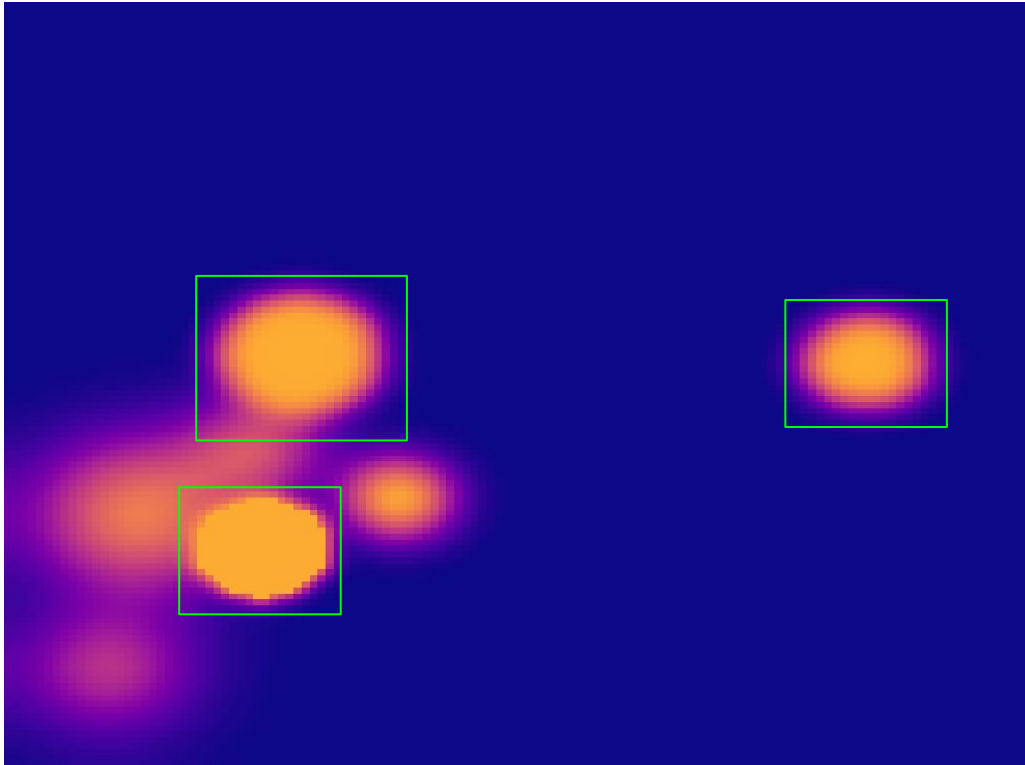


(c) Strong scaling results represented in hour-core for a 8 GiB problem size

Tutorial: 2D heat simulation + in-situ image generation

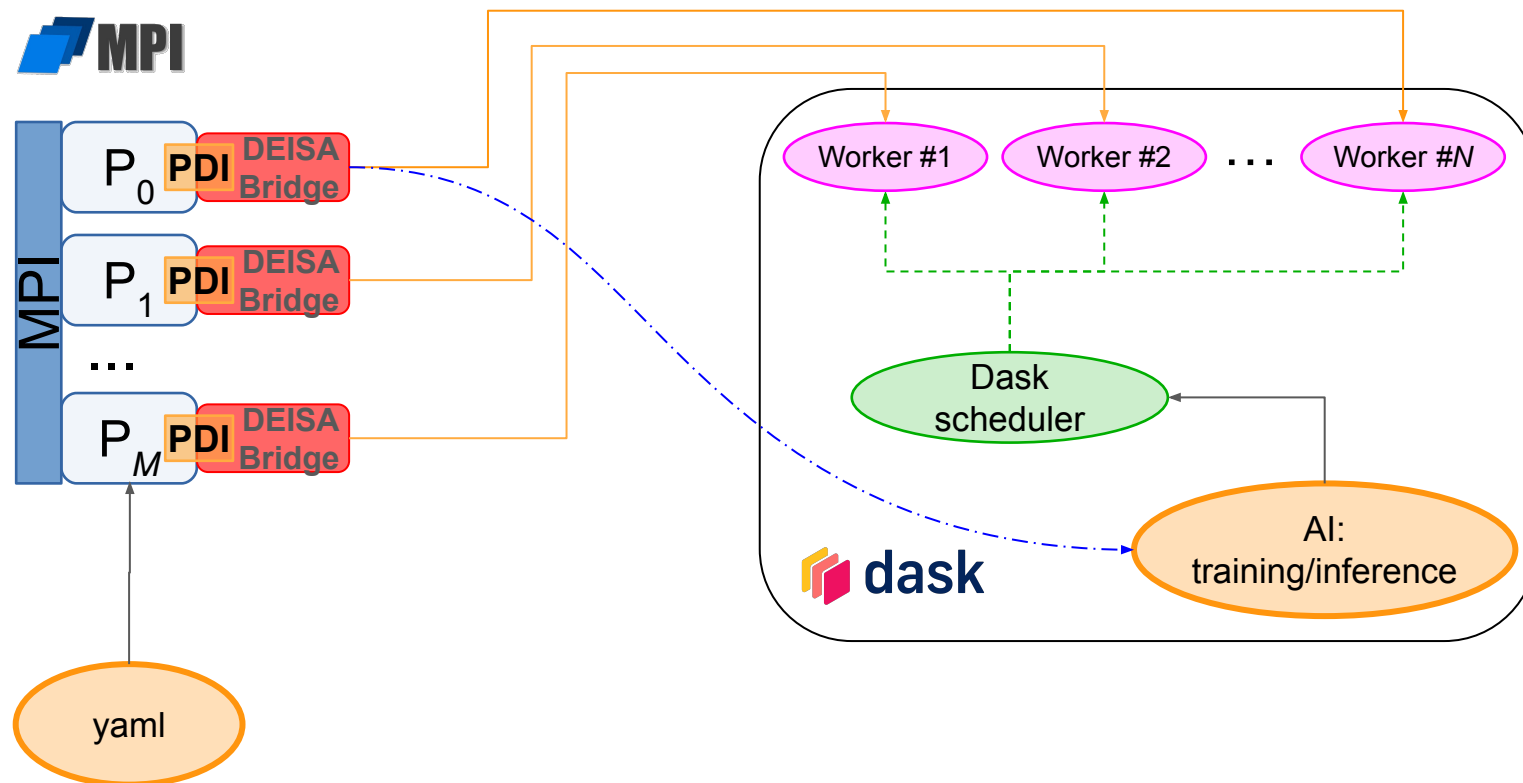


Motivation: automatic detection using AI

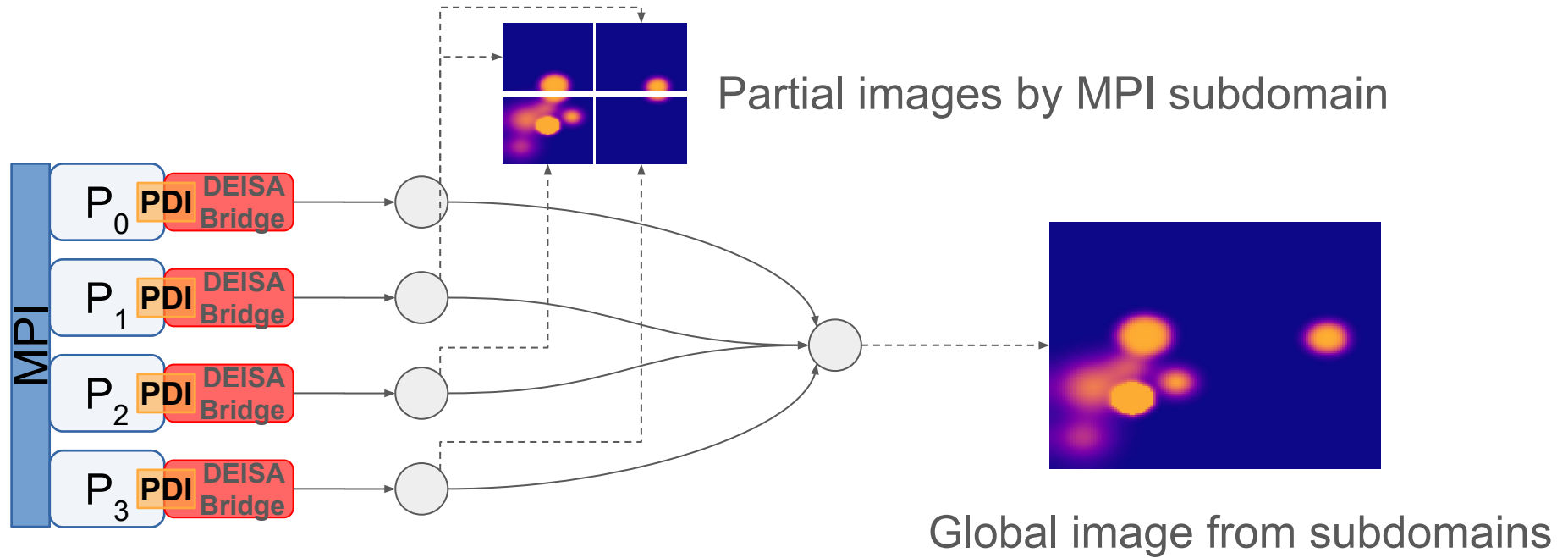


- ↘ stress on file system: only write interest zone
- automatic notification
- feedback to simulation
 - parameters: i/o frequency, resolution, ...
 - replay from previous snapshot

Deisa architecture



Hands-on: in-situ visualization



Hands-on: environment

- ssh ruche
- `./gpfs/workdir/shared/pdi-deisa/setup-env.sh` `# setup environment`

- git clone <https://github.com/pdidev/tutorial.git> && cd tutorial/ex_deisa
- `mkdir build && cd build` `# create build directory`
- `cmake .. && make ex0` `# build`
- `mpirun -np 4 ./ex0` `# run code`

Hands-on: PDI yaml configuration

```
pdi:
  metadata:
    ii: int
    [...]
  data:
    main_field: {type: array, subtype: double, size: ['$dsize[0]', '$dsize[1]']}

plugins:
  [...]

  deisa:
    scheduler_info: scheduler.json      # Dask generated config file
    init_on: initialization              # PDI event called after sharing all metadata
    time_step: $ii                      # Timestep variable
    deisa_arrays:                       # Deisa virtual arrays equivalent to Dask arrays
      global_t:                         # Name associated to the virtual array
        type: array
        subtype: double
        size: [1001, '$dsize[0]*$psize[0]', '$dsize[1]*$psize[1]']
        subsize: [1, '$dsize[0]', '$dsize[1]']      # Size of each chunk
        start: [$ii, '$dsize[0]*$pcood[0]', '$dsize[1]*$pcood[1]'] # Start of each chunk
        +timedim: 0                                # Index of the time dimension
    map_in: # Map local data to the Deisa array
      main_field: global_t
```

What's next in Deisa? NumPEX !

- Make Deisa **production-grade** (in progress)
 - Improve scalability & performance
 - Upstream Dask modifications
 - Improve packaging
- Validate PDI + Deisa + AI for **event detection**
- Integrate in GYSELA rewrite
 - New analytics based on PDI/Deisa, support post hoc / in situ transparently
- **New features**
 - Triggers & feedback from analytics to simulation
 - Prevent data copy
 - Standardized AI pipeline (training + inference)