



QUANTUM ESPRESSO on heterogeneous architectures

**EUROHPC
USER DAY
2023** Brussels
11.12.23

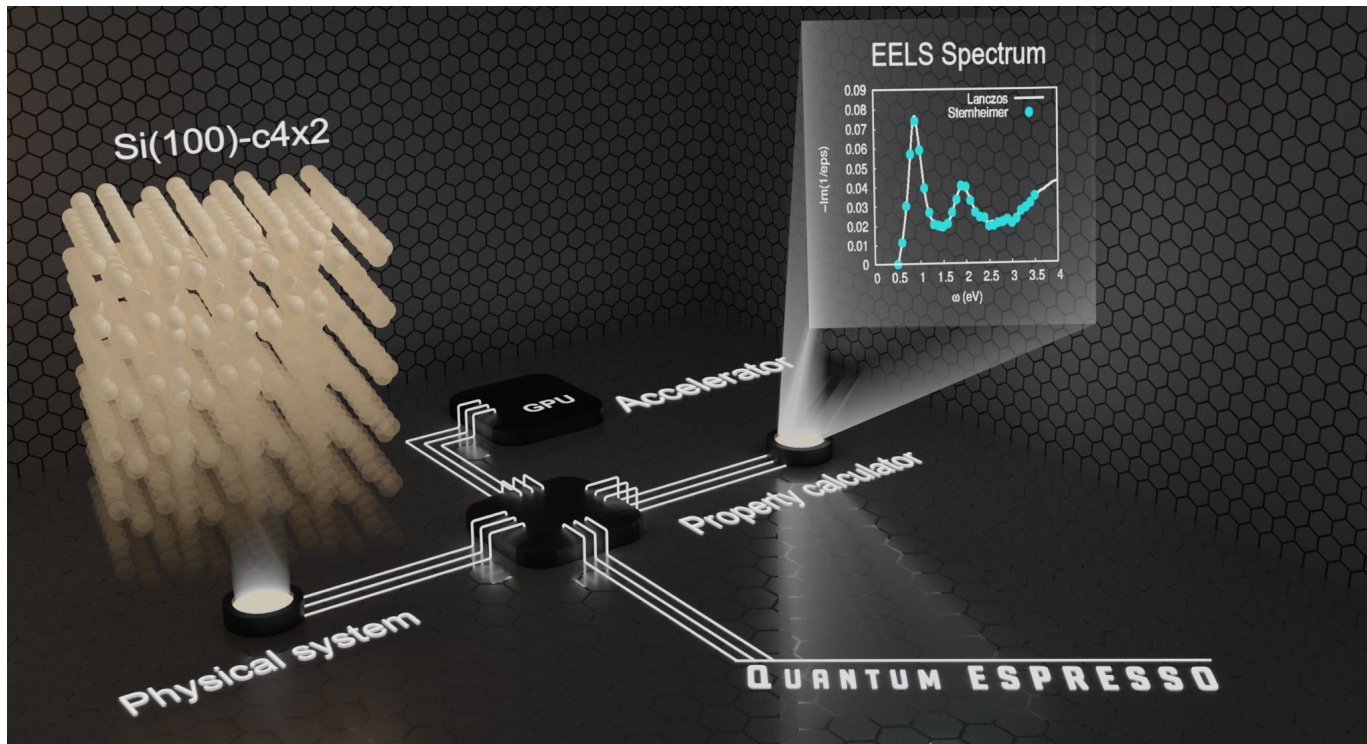


Project: “*EHPC-DEV-2023D06-013*”

EuroHPC used: LUMI, Leonardo

Speaker: Ivan CARNIMEO (SISSA)

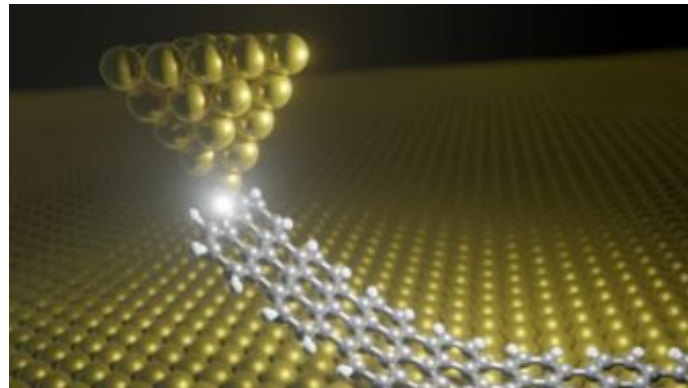
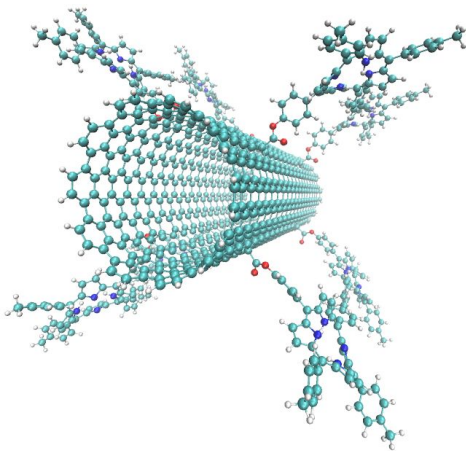
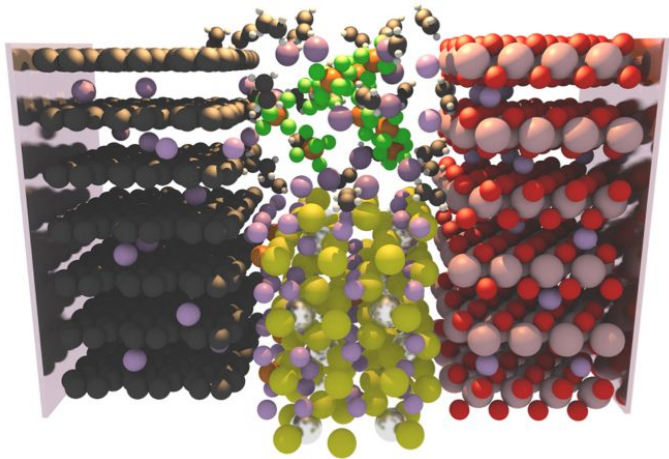
The QUANTUM ESPRESSO project



QUANTUM ESPRESSO is an integrated suite of **Open-Source** computer codes for electronic-structure calculations and materials modeling at the nanoscale. It is based on density-functional theory, plane waves, and pseudopotentials.

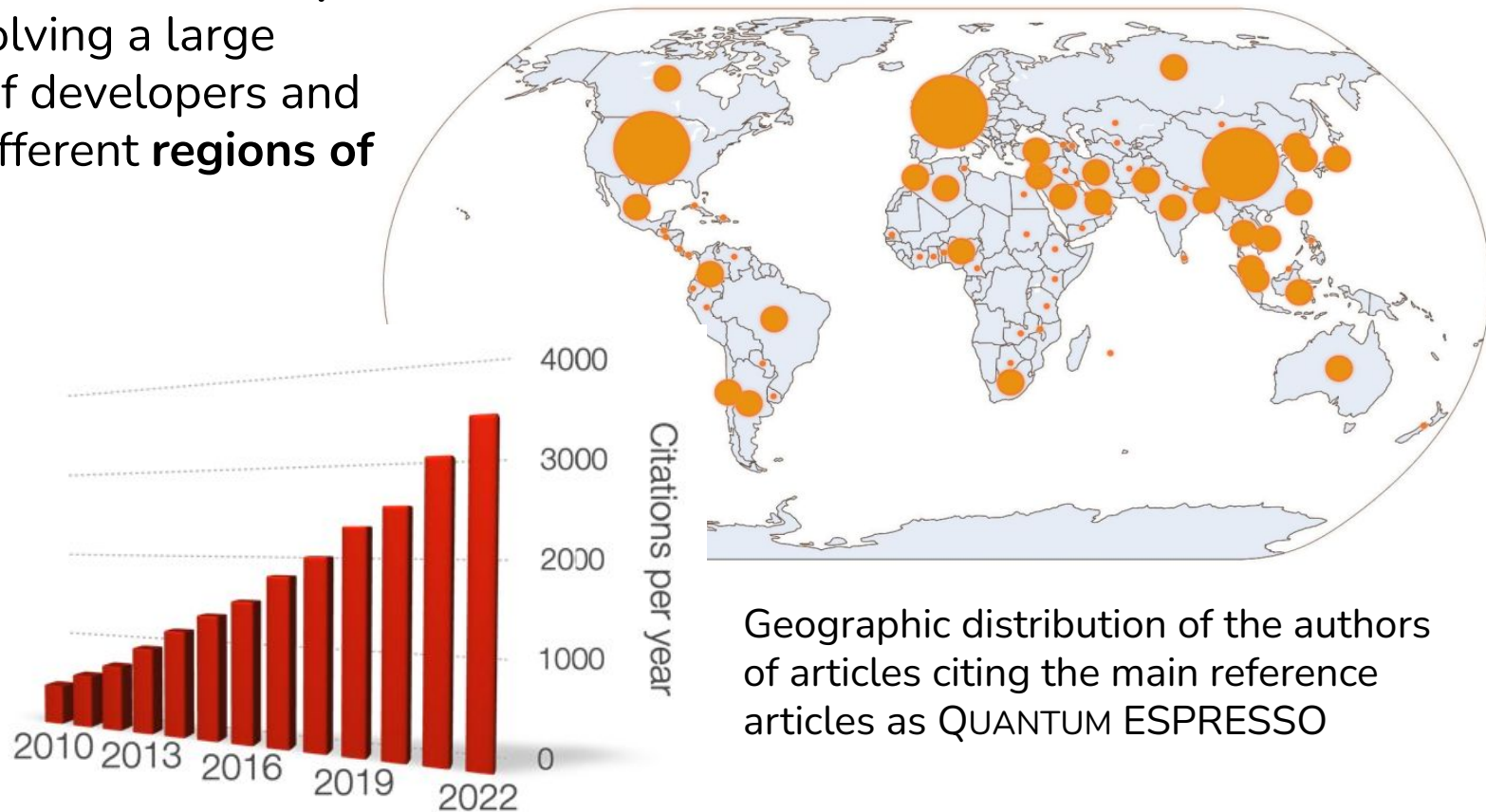
The QUANTUM ESPRESSO project

QUANTUM ESPRESSO is routinely used to simulate **large and complex molecular systems**. Clusters hosted at HPC centers play a crucial role to enhance accuracy and use predictive methods.



The QUANTUM ESPRESSO project

QUANTUM ESPRESSO is an **open initiative** involving a large **community** of developers and users from different **regions of the world**



Data provided by courtesy of the **QUANTUM ESPRESSO** foundation

Geographic distribution of the authors of articles citing the main reference articles as QUANTUM ESPRESSO

The QUANTUM ESPRESSO project

Geographic distribution of downloads from the QE website since the beginning of 2022

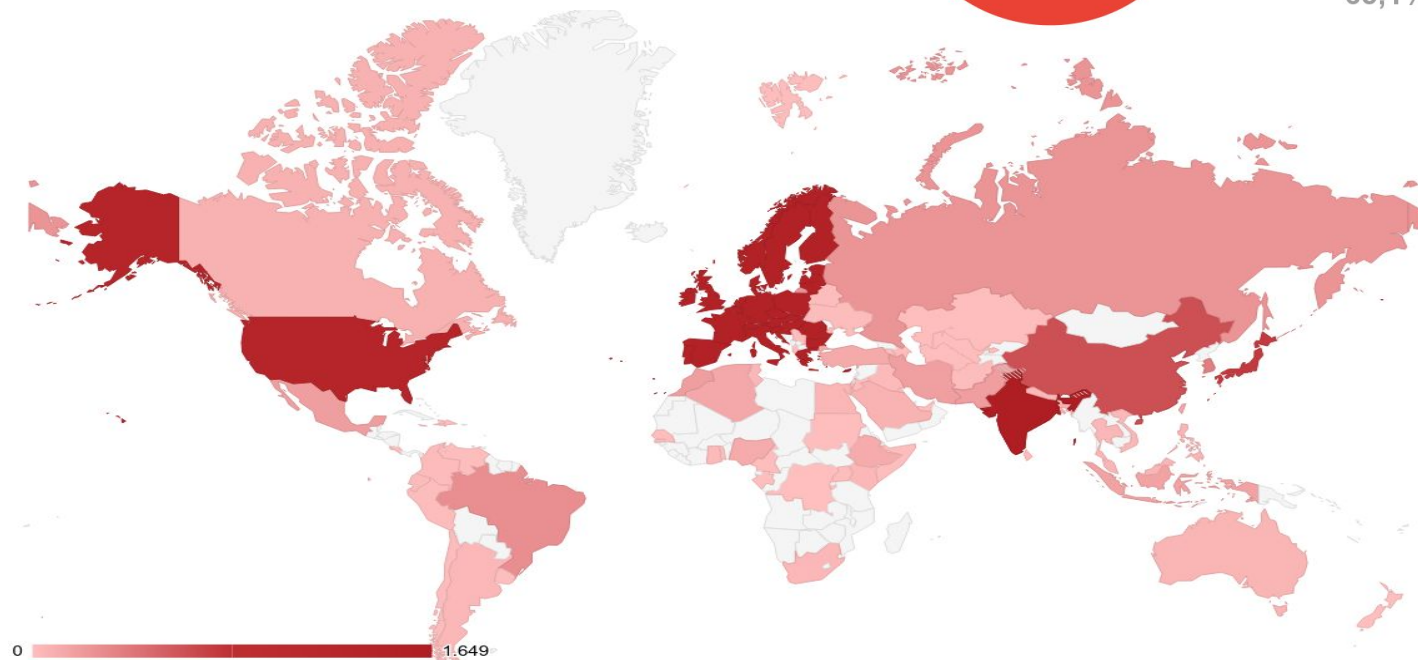
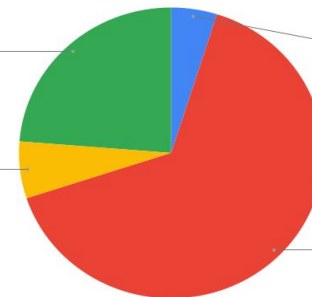
35000+ downloads
20000+ citations
50000+ authors

OTHER
23,7%

NATIONAL LAB
4,9%

INDUSTRY
6,3%

ACADEMIA
65,1%

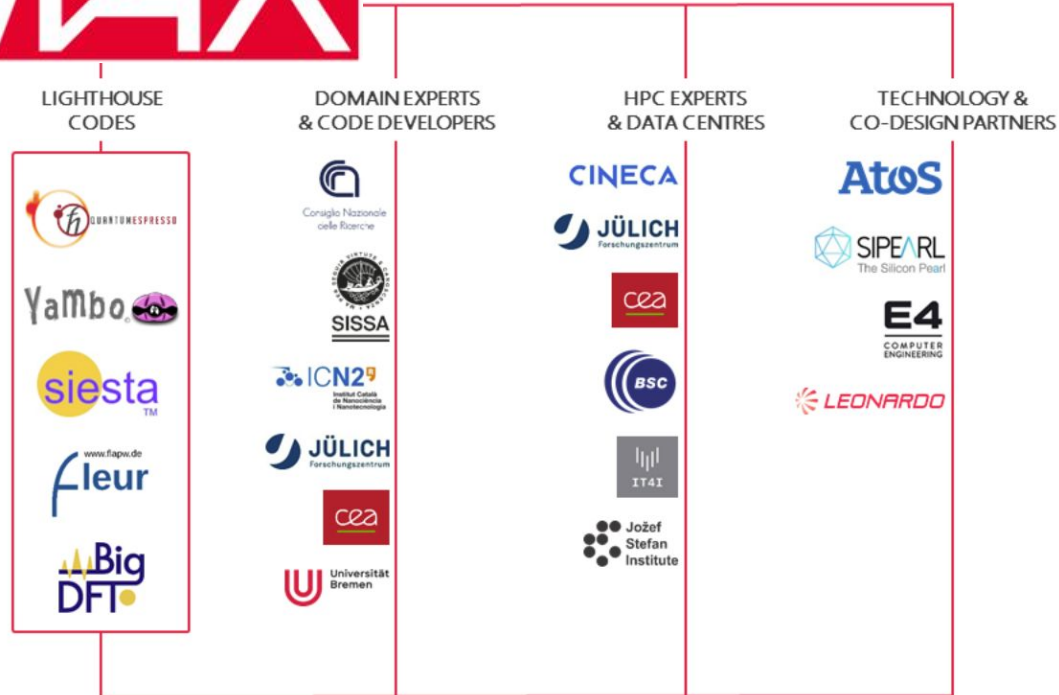


Data provided by courtesy of the **QUANTUM ESPRESSO** foundation

Materials design at the Exascale



CoE for HPC applications in materials science



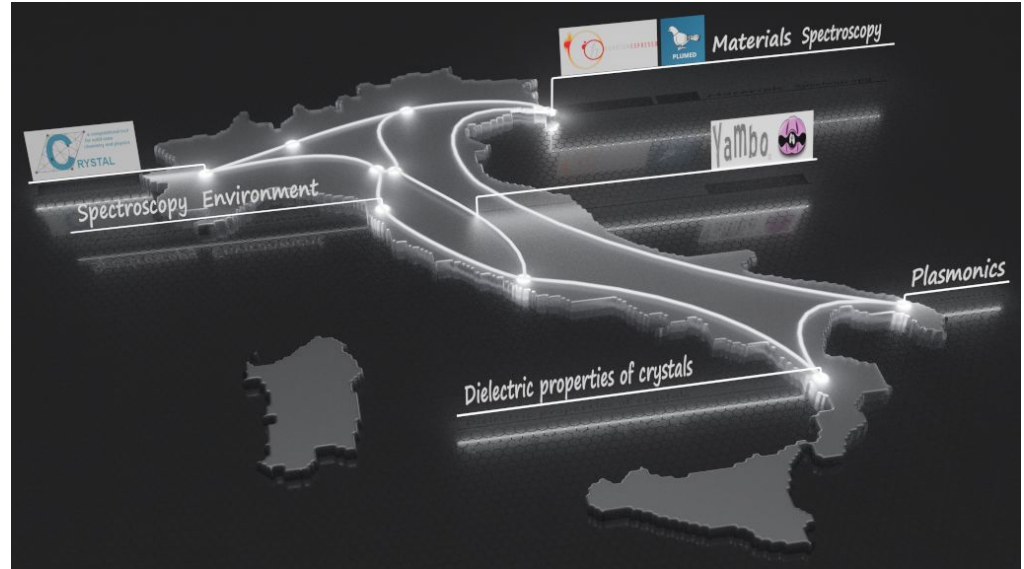
exploit **frontier HPC**
for material science research
in strong link with **scientific
communities**

CODE PORTING

CO-DESIGN

HTC ECOSYSTEM

Spoke 7 – Flagship codes



Porting to heterogeneous architectures



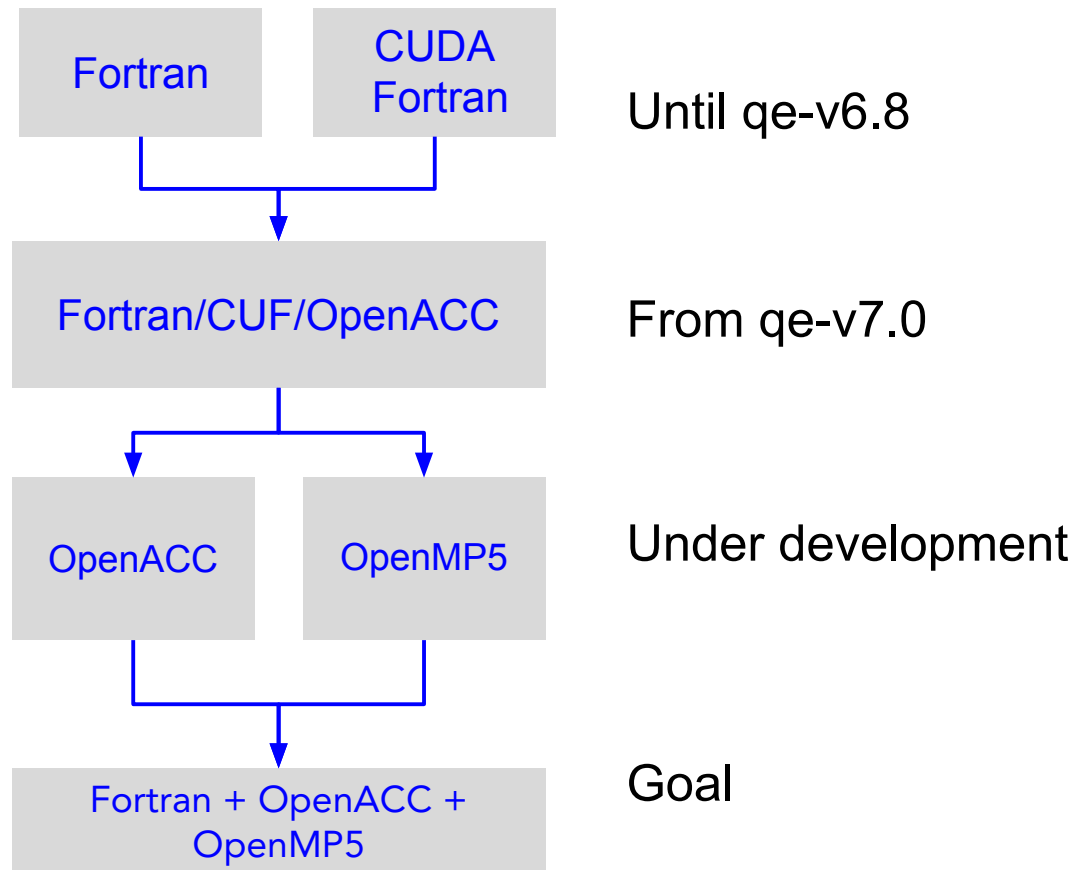
The current strategy for performance portability is to specialize the code to different hardware configurations by using directive based approaches:

OpenACC and **OpenMP**

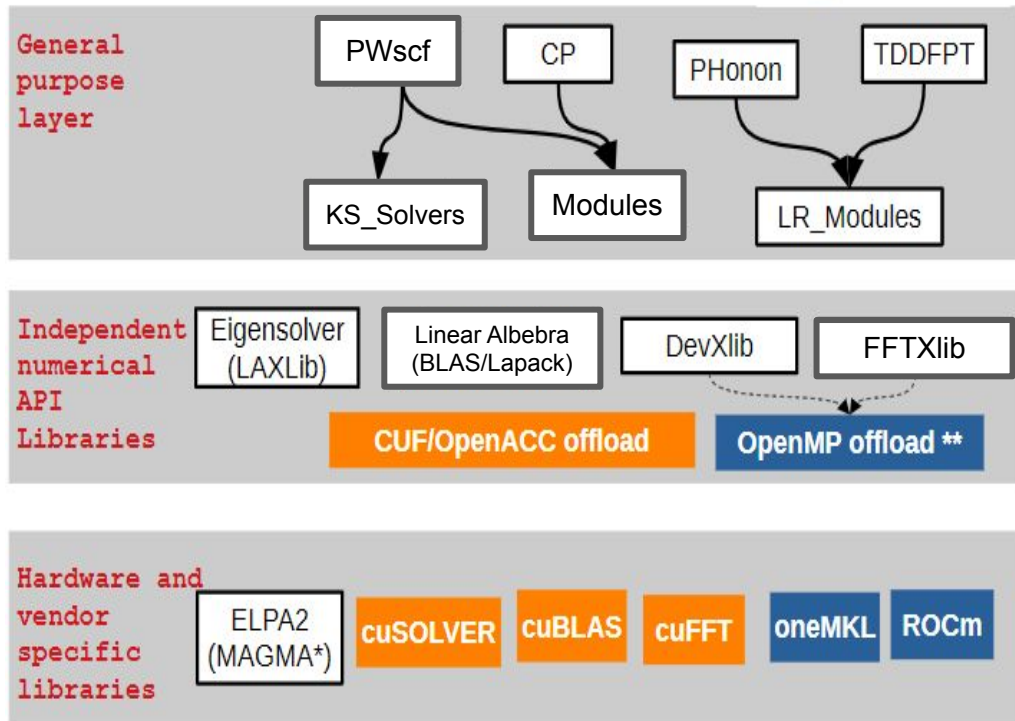
Porting to heterogeneous architectures

J. Chem. Phys. **152**,
154105 (2020)

J. Chem. Theory
Comput. **19**, 6992
(2023)



Porting to heterogeneous architectures



directives

MAINTAINABILITY

PORTABILITY

multiple back-ends

FLEXIBILITY

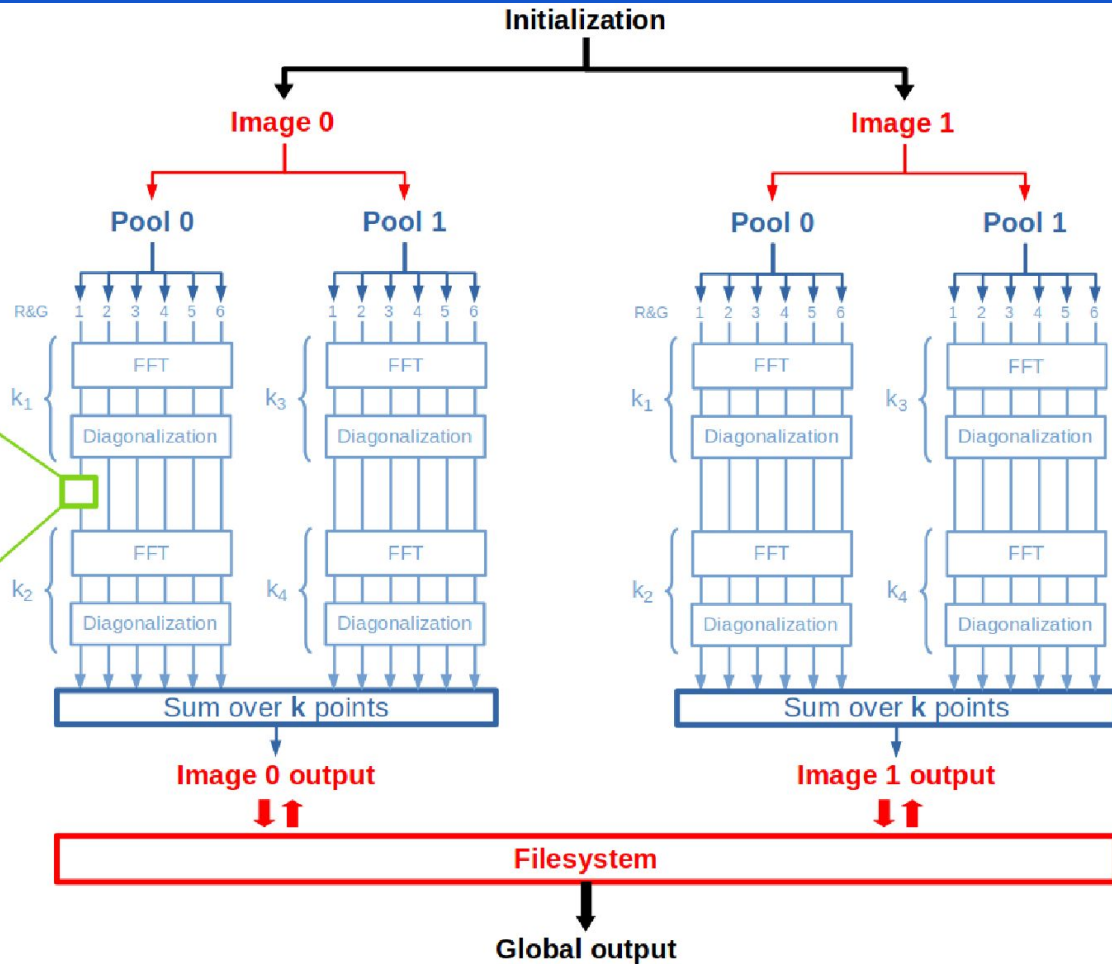
PERFORMANCE

Modularity enables interoperability and new programming models

Porting to heterogeneous architectures

Several parallelization schemes are implemented and integrated with GPU offload

```
#ifdef(_OPENACC)
!$acc parallel loop
#else
!$omp parallel do
#endif
DO i = 1, Npw
  [...]
END DO
```



CUF/OpenACC offload

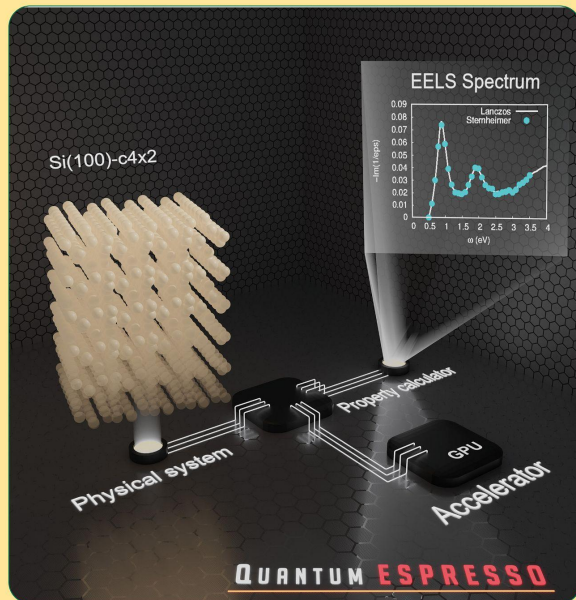
	6.4	6.5a1	6.5a2	6.7	6.8	7.0	7.1	7.2
Davidson	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange
Energy	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange
Magnetism	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange
USPP	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange
PAW	Orange	Orange	Orange	Orange	Orange	Orange	Orange	Orange
Forces	Grey	Orange	Orange	Orange	Orange	Orange	Orange	Orange
Stress	Grey	Grey	Orange	Orange	Orange	Orange	Orange	Orange
KS_Solvers	Grey	Orange	Grey	Orange	Orange	Orange	Orange	Orange
DFT+U	Grey	Orange	Grey	Orange	Orange	Orange	Orange	Orange
XC	Grey	Orange	Grey	Grey	Orange	Orange	Orange	Orange
VdW(D3)	Grey	Orange	Grey	Grey	Orange	Orange	Orange	Orange
CP	Grey	Orange	Grey	Grey	Grey	Orange	Orange	Orange
Phonon	Grey	Orange	Grey	Grey	Grey	Grey	Grey	Orange
turbo_eels	Grey	Orange	Grey	Grey	Grey	Grey	Grey	Orange
turbo_lanczos	Grey	Orange	Grey	Grey	Grey	Grey	Grey	Orange
hp.x	Grey	Orange	Grey	Grey	Grey	Grey	Grey	Orange

The QUANTUM ESPRESSO suite has been accelerated using a mixed **CUDA Fortran/OpenACC** scheme. A version based on **OpenMP** offloading is under heavy development, in order to enhance portability to hardware from different vendors.

JCTC

Journal of Chemical Theory and Computation

XXXXX XX, XXXX Volume XX Number XX pubs.acs.org/JCTC



JCTC

Journal of Chemical Theory and Computation

pubs.acs.org/JCTC

Open Access

This article is licensed under [CC-BY 4.0](#)

Article

QUANTUM ESPRESSO: One Further Step toward the Exascale

Ivan Carnimeo,* Fabio Affinito, Stefano Baroni, Oscar Baseggio, Laura Bellentani, Riccardo Bertossa, Pietro Davide Delugas, Fabrizio Ferrari Ruffino, Sergio Orlandini, Filippo Spiga, and Paolo Giannozzi



Cite This: *J. Chem. Theory Comput.* 2023, 19, 6992–7006



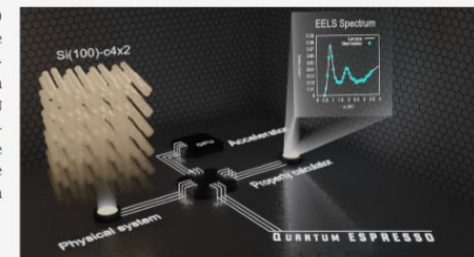
Read Online

ACCESS |

Metrics & More

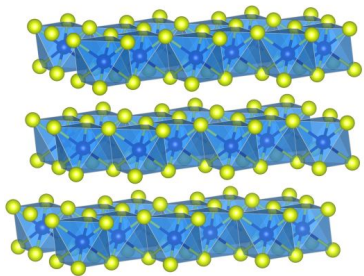
Article Recommendations

ABSTRACT: We review the status of the QUANTUM ESPRESSO software suite for electronic-structure calculations based on plane waves, pseudopotentials, and density-functional theory. We highlight the recent developments in the porting to GPUs of the main codes, using an approach based on OpenACC and CUDA FORTRAN offloading. We describe, in particular, the results achieved on linear-response codes, which are one of the distinctive features of the QUANTUM ESPRESSO suite. We also present extensive performance benchmarks on different GPU-accelerated architectures for the main codes of the suite.

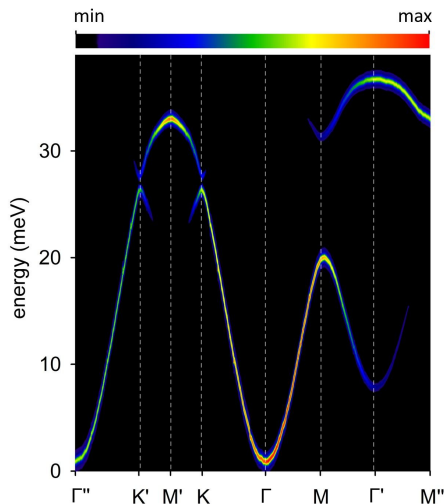


CUF/OpenACC offload

Chromium
Iodide, 7776
electrons, 1152
atoms

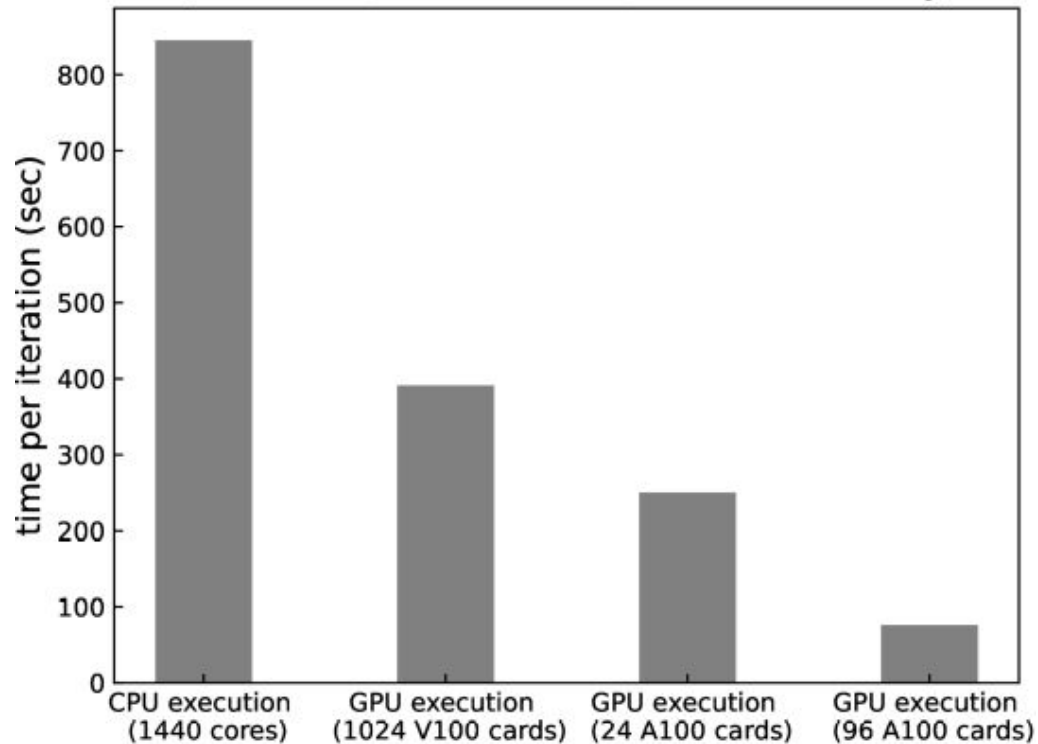


Magnon
dispersions along
the
high-symmetry
directions of the
BZ



Delugas et al., *Phys. Rev. B.*, 107, 214452 (2023)
Gorni et al., *Phys. Rev. B.*, 107, L220410 (2023)

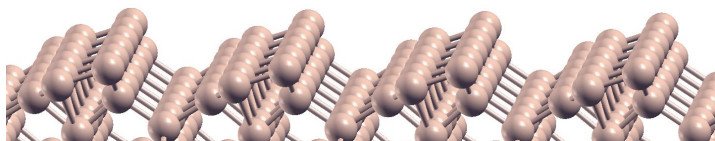
PWSCF - Chromium Iodide orthorhombic supercell
(1152 atoms, 7776 electrons, NCPP+CC:240/60Ry)



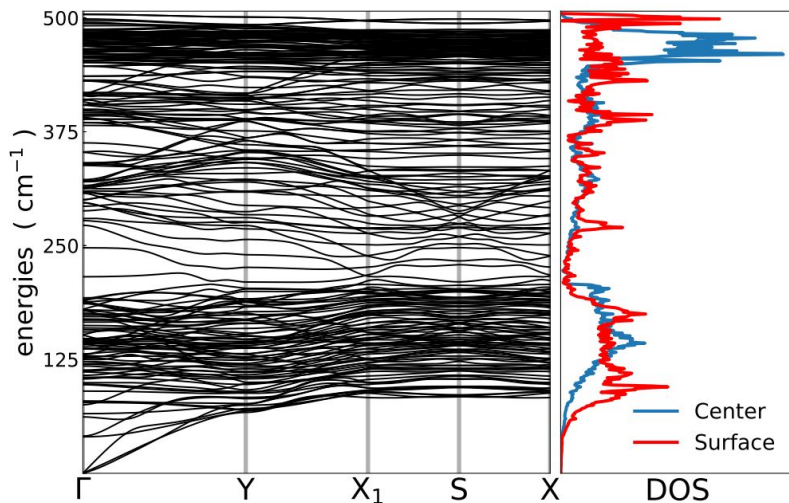
Quantum ESPRESSO: one further step towards the exascale, I.
Carnimeo et al., *JCTC*, 19, 20, 6992-7006 (2023)

CUF/OpenACC offload

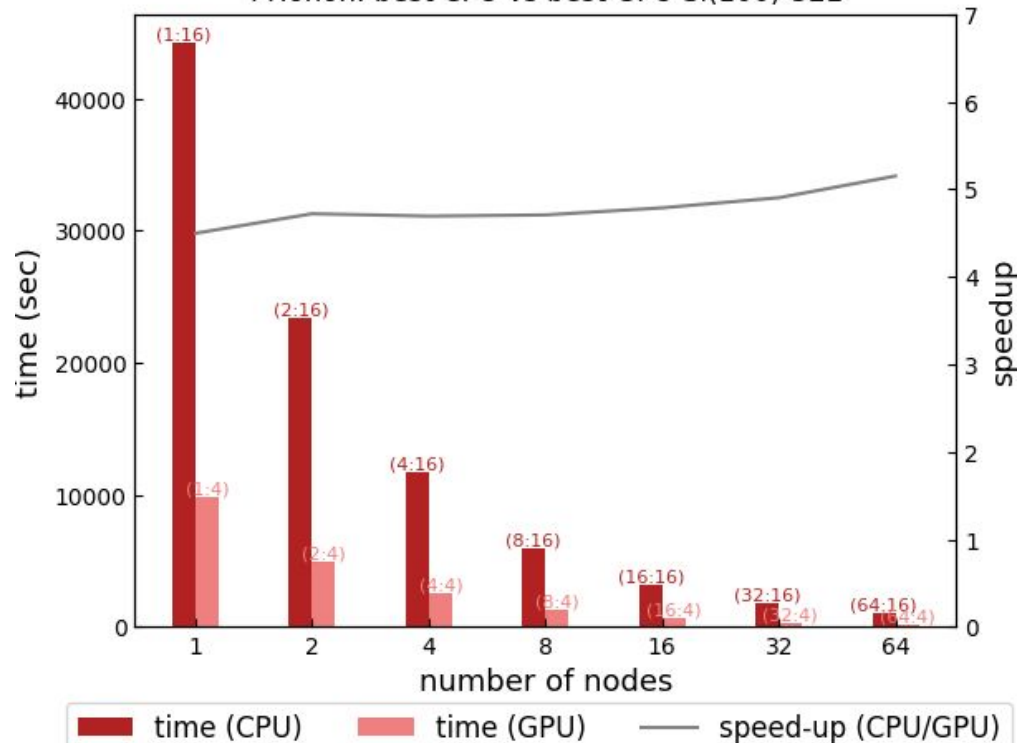
Silicon 100 surface, 512 electrons, 128 atoms



Phonon dispersions along the high-symmetry directions of the BZ

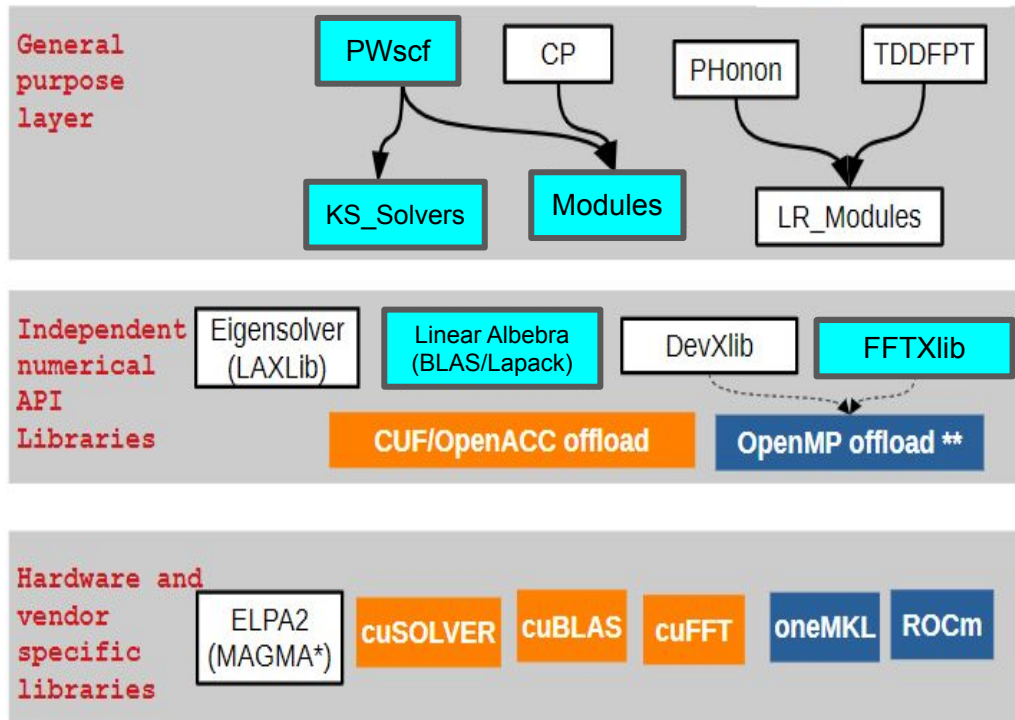


PHonon: best CPU vs best GPU Si(100)-32L



Quantum ESPRESSO: one further step towards the exascale, I.
Carnimeo et al., *JCTC*, **19**, 20, 6992-7006 (2023)

OpenMP5 offload



Basic features:

- **loop** offloading;
- **global variables**; offloading and pinning;
- manage different **backends** (linear algebra and FFTs);
- **streams** and/or **tasks** (for async batched FFTs).

OpenMP5 offload

CUF only

Host to Device

```
if ( use_gpu ) then
  arg_d = arg
endif
```

Routine calls

```
if ( use_gpu ) then
  call abc( arg_d )
else
  call abc( arg )
endif
```

Interfaces

```
interface abc
  subroutine abc_cpu( v )
  subroutine abc_gpu( v_d )
end interface
```

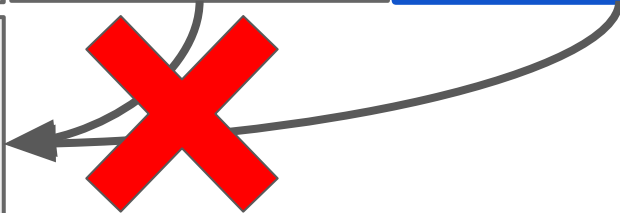
OpenMP5 offload

	CUF only	CUF interfaces OpenACC parent code
Host to Device	<pre>if (use_gpu) then arg_d = arg endif</pre>	<pre>!\$acc update device(arg)</pre>
Routine calls	<pre>if (use_gpu) then call abc(arg_d) else call abc(arg) endif</pre>	<pre>!\$acc host_data use_device(arg) call abc(arg) !\$acc end host_data</pre>
Interfaces	<pre>interface abc subroutine abc_cpu(v) subroutine abc_gpu(v_d) end interface</pre>	

OpenMP5 offload

	CUF only	CUF interfaces OpenACC parent code	OpenACC only
Host to Device	<pre>if (use_gpu) then arg_d = arg endif</pre>	<pre>!\$acc update device(arg)</pre>	
Routine calls	<pre>if (use_gpu) then call abc(arg_d) else call abc(arg) endif</pre>	<pre>!\$acc host_data use_device(arg) call abc(arg) !\$acc end host_data</pre>	<pre>call abc_acc(arg)</pre>
Interfaces	<pre>interface abc subroutine abc_cpu(v) subroutine abc_gpu(v_d) end interface</pre>		<pre>subroutine abc_acc(v)</pre>

OpenMP5 offload

	CUF only	CUF interfaces OpenACC parent code	OpenACC only	OpenACC + OpenMP5	
Host to Device	<pre>if (use_gpu) then arg_d = arg endif</pre>	<pre>!\$acc update device(arg)</pre>		<pre>!\$acc update device(arg) !\$omp target update to (arg)</pre>	
Routine calls	<pre>if (use_gpu) then call abc(arg_d) else call abc(arg) endif</pre>	<pre>!\$acc host_data use_device(arg) call abc(arg) !\$acc end host_data</pre>	<pre>call abc_acc(arg)</pre>	<pre>#if def __OPENACC call abc_acc(arg) #elif def __OPENMP call abc_omp(arg) #endif</pre>	
Interfaces	<pre>interface abc subroutine abc_cpu(v) subroutine abc_gpu(v_d) end interface</pre>				

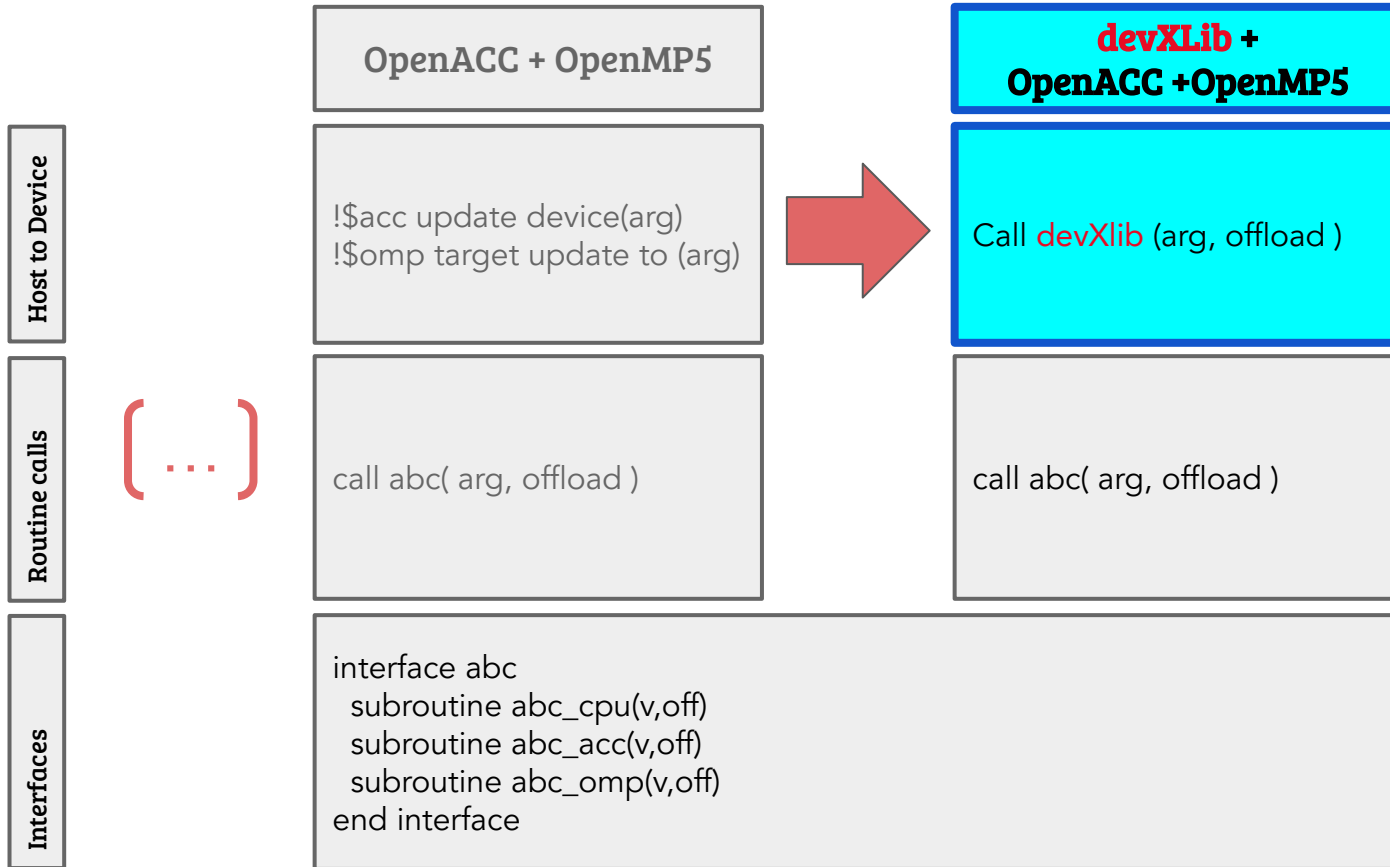
OpenMP5 offload

	CUF only	CUF interfaces OpenACC parent code	OpenACC only	OpenACC + OpenMP5
Host to Device	<pre>if (use_gpu) then arg_d = arg endif</pre>	<pre>!\$acc update device(arg)</pre>		<pre>!\$acc update device(arg) !\$omp target update to (arg)</pre>
Routine calls	<pre>if (use_gpu) then call abc(arg_d) else call abc(arg) endif</pre>	<pre>!\$acc host_data use_device(arg) call abc(arg) !\$acc end host_data</pre>	<pre>call abc_acc(arg)</pre>	<pre>#if def __OPENACC call abc_acc(arg) #elif def __OPENMP call abc_omp(arg) #endif</pre>
Interfaces	<pre>interface abc subroutine abc_cpu(v) subroutine abc_gpu(v_d) end interface</pre>	<pre>subroutine abc_acc(v)</pre>		<pre>subroutine abc_acc(v) subroutine abc_omp(v)</pre>

OpenMP5 offload

	CUF only	CUF interfaces OpenACC parent code	OpenACC only	OpenACC + OpenMP5
Host to Device	<pre>if (use_gpu) then arg_d = arg endif</pre>	<pre>!\$acc update device(arg)</pre>		<pre>!\$acc update device(arg) !\$omp target update to (arg)</pre>
Routine calls	<pre>if (use_gpu) then call abc(arg_d) else call abc(arg) endif</pre>	<pre>!\$acc host_data use_device(arg) call abc(arg) !\$acc end host_data</pre>	<pre>call abc_acc(arg)</pre>	<pre>call abc(arg, offload)</pre>
Interfaces	<pre>interface abc subroutine abc_cpu(v) subroutine abc_gpu(v_d) end interface</pre>		<pre>subroutine abc_acc(v)</pre>	<pre>interface abc subroutine abc_cpu(v,off) subroutine abc_acc(v,off) subroutine abc_omp(v,off) end interface</pre>

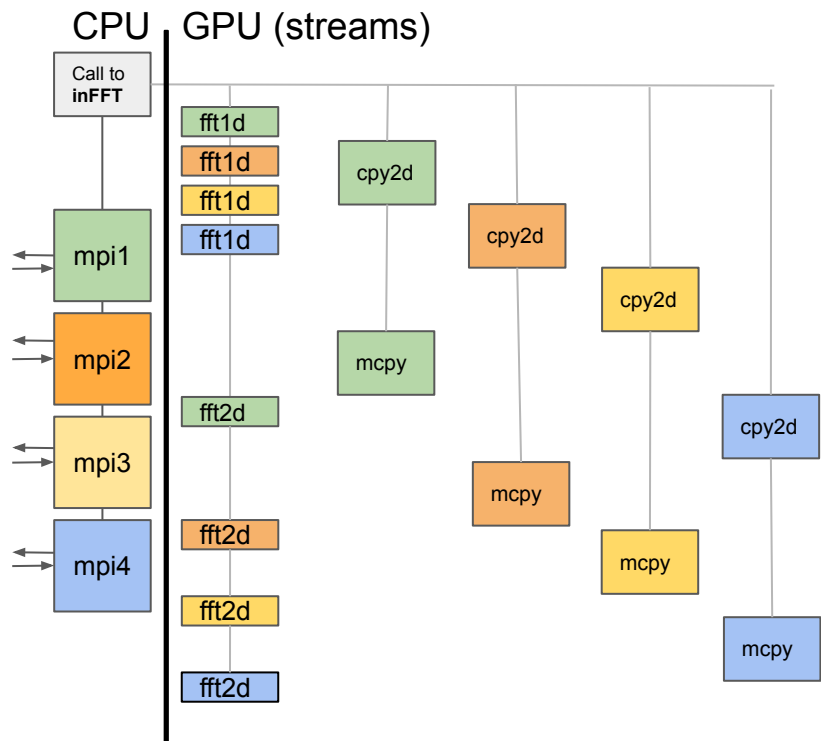
OpenMP5 offload



The Yambo group in Modena is developing a portable library (**devXLib**) to manage porting to multiplatform heterogeneous architectures

Main developers:
N. Spallanzani (CNR-NANO)
G. Rossi (Intel)
A. Ferretti (CNR-NANO)

Batched FFTs - CUF, HIP

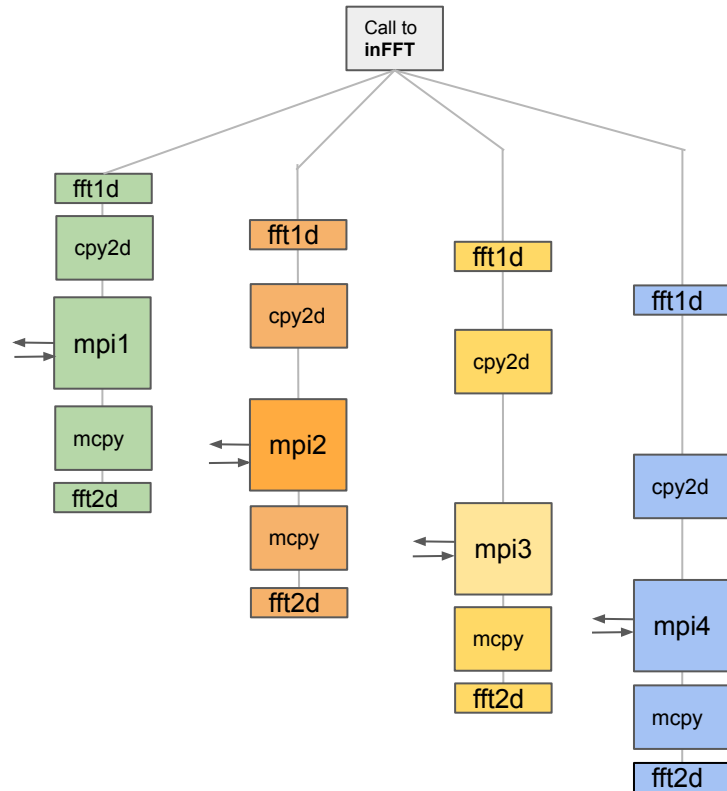


- **Batched 3d-FFT of the wave-function;**
- the input array divided in **4 batches** (on bands);
- 1 stream for **FFTs**, 4 streams for **data movements**;
- 4 **async mpi** communications (ISEND, IRECV).

Notes:

- **non-asynchronous memcpy**;
- memcpy operations **D2H/H2D** much more time consuming than FFT calls;
- memcpy operations **D2D** same order of FFT calls.

Batched FFTs - oneMKL

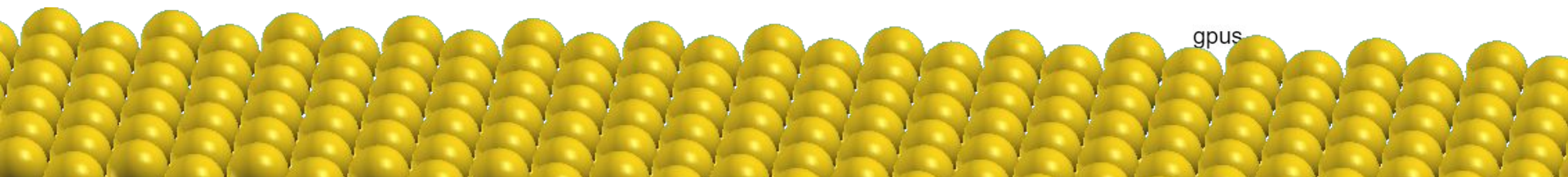
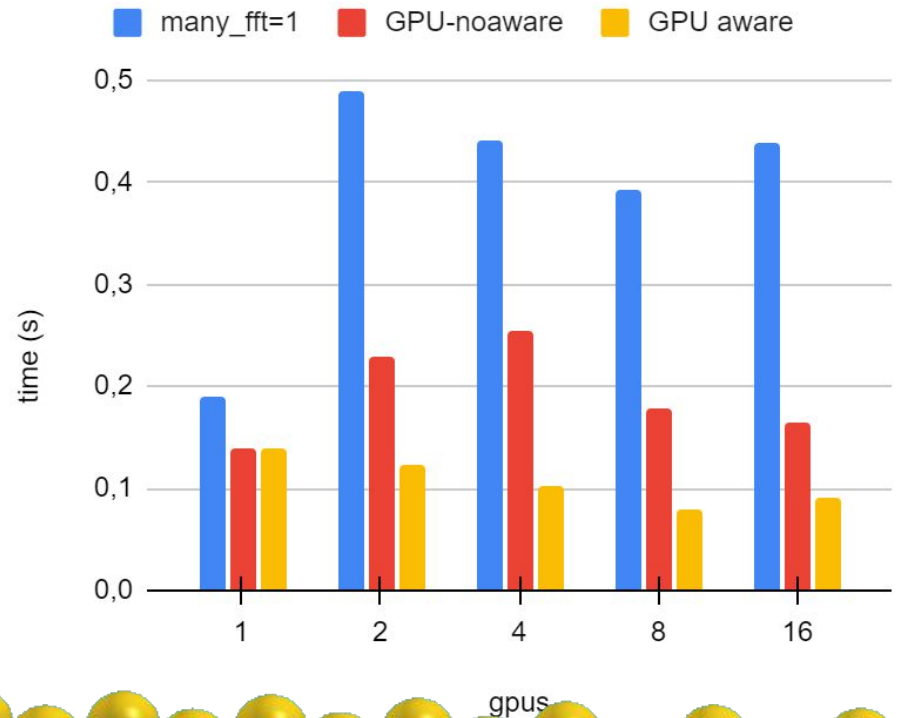


- ntasks associated to nbatches
- work in progress...

Execution on LUMI

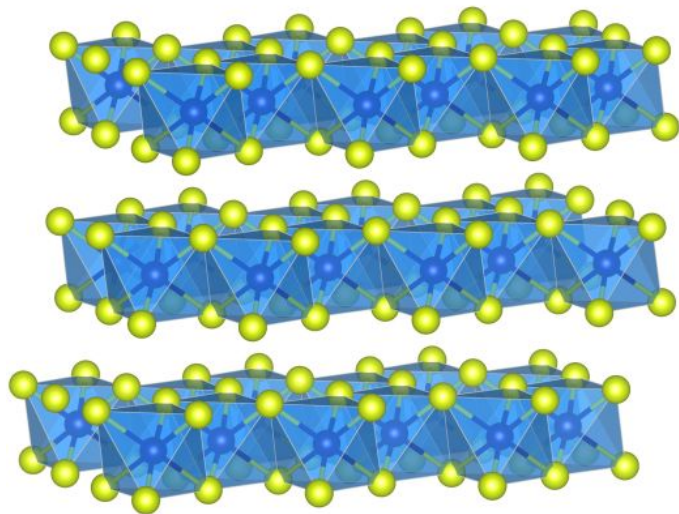
Au surface
~1600 electrons
112 atoms

vloc_psi/call

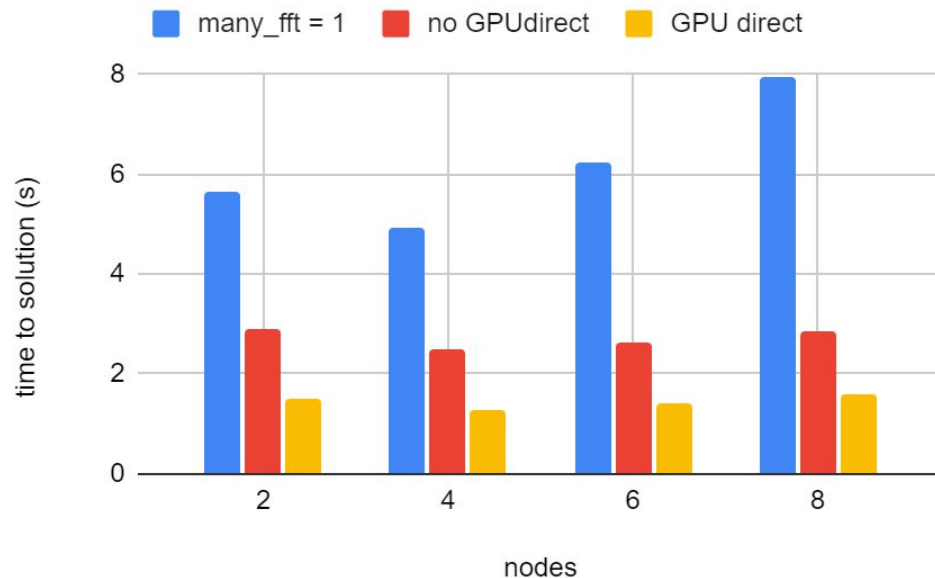


Execution on LUMI

Chromium Iodide
~2700 electrons
480 atoms



cri3-small, vloc_psi / call



What's next

- **Complete the OpenMP porting** of PWscf minor routines;
 - FFT optimization with OpenMP
- **Medium/large-size benchmarks**
- Port QE codes other than PWscf (**PHonon**, **CP**, **EELS**, ...);
 - incorporate **DevXlib**.

Acknowledgments

QUANTUM ESPRESSO developers

- Pietro Delugas, SISSA
- Ivan Carnimeo, SISSA
- Fabrizio Ferrari Ruffino, CNR-IOM
- Oscar Baseggio, SISSA
- Riccardo Bertossa, SISSA
- Aurora Ponzi, CNR-IOM
- Stefano Baroni, SISSA, CNR-IOM
- Paolo Giannozzi, UniUD, CNR-IOM

CINECA

- Laura Bellentani
- Sergio Orlandini
- Fabio Affinito

QUANTUM ESPRESSO Foundation

- Francesca Garofalo

Other collaborators and vendor technical support (chronological order)

- Ye Luo (Argonne)
- Filippo Spiga (NVIDIA)
- Louis Stuber (NVIDIA)

- Giacomo Rossi (Intel)
- Ossian O'Reilly (AMD)
- Jakub Kurzak (AMD)



Exploring the Ultimate Regime of Turbulent Rayleigh-Bénard Convection Through Unprecedented Spectral-Element Simulations

**EUROHPC
USER DAY
2023** Brussels
11.12.23



Project: Extreme-scale high-fidelity turbulence simulations of convection and boundary layers using accelerators (EHPC-EXT-2022E01-059)

EuroHPC used: LUMI and Leonardo

Speaker: Niclas JANSSON (KTH)

2023 ACM Gordon Bell Prize Finalist



Martin Karp



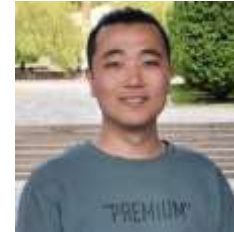
Adalberto Perez



Timofey Mukha



Yi Ju



Jiahui Liu



Szilárd Páll



Erwin Laure



Tino Weinkauff



Jörg Schumacher



Philipp Schlatter



Stefano Markidis

Turbulent thermal convection

- Applications in nature and technology
 - From chip cooling, heat exchanges in power plants, to heat convection in the Earth's mantle and the sun.
- **Rayleigh-Bénard convection:** Canonical turbulent convection with fundamental open question: **Is there an ultimate regime**, i.e. anomalous scaling of Nusselt number (heat transfer) and Rayleigh number (buoyancy)?
 - Long-standing open issue in turbulence (Kraichnan 1962)
 - Difficult to conduct controlled experiments at high Rayleigh numbers $Ra > 10^{15}$
- Challenges with direct numerical simulations
 - **Large computational cost** due to resolution needs: $(H/\eta)^3 \sim Ra^{9/8}$
 - Numerical method with **minimal dissipative and dispersive errors** to capture and track small scales in time
 - Produces **unmanageable volumes of data**
 - **Long integration** times for steady state statistics
 - **Efficient implementation** on modern hardware

Illustration of the canonical problem at $Ra = 10^{13}$, iso-surfaces of temperature



Introduction

- Exascale will require either **unreasonably large problem** sizes or **significantly improved efficiency** of current methods
 - Finite-Volume LES of a full car on the entire K computer (京) required **more than 100 billion grid points** to run efficiently
 - What problem size is needed to fill the 379 PFlop/s LUMI...
- High-order methods
 - Attractive numerical properties, **small dispersion** errors and more "accuracy" per degree of freedom
 - Better suited to take advantage of **modern hardware** (accelerators)

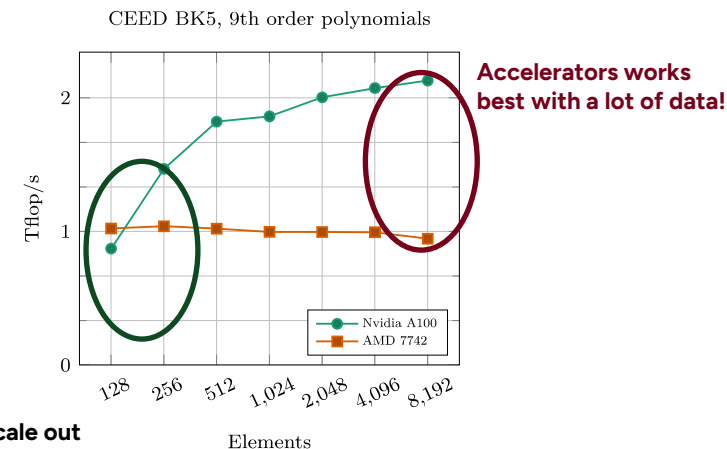


...but we rather scale out our problems...

京: 82944 nodes, 663552 Cores, 10 PFlop/s

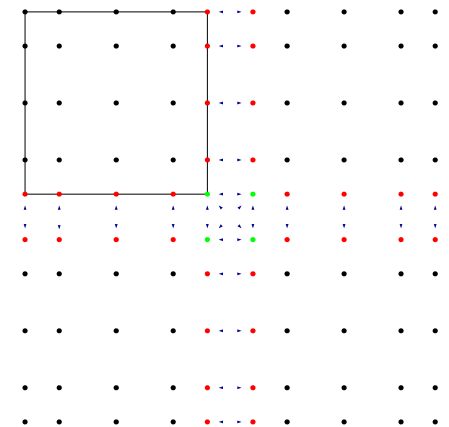
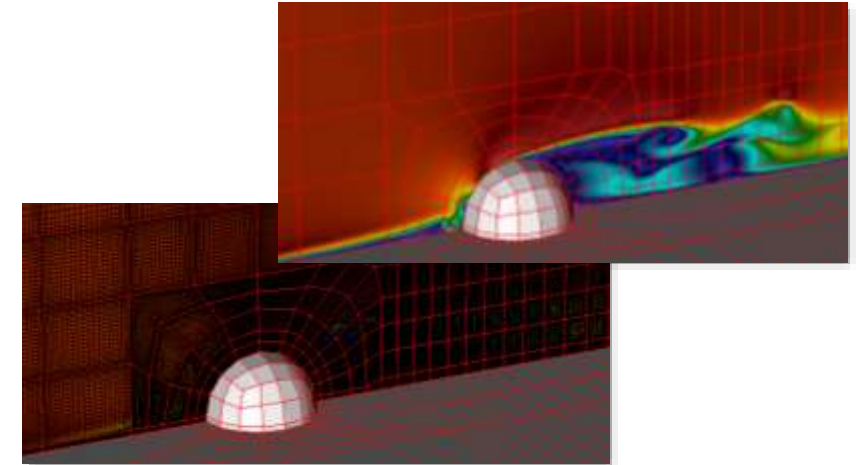


Dardel: 56 nodes, 448 MI250X GCDs, ≈10 PFlop/s



Spectral Elements

- Finite Elements with high-order basis functions
 - N -th order Legendre-Lagrange polynomials $l_i(\xi)$
 - Gauss-Lobatto-Legendre quadrature points ξ_i
 - Fast tensor product formulation
 - $u^e(\xi, \eta, \gamma) = \sum_{i,j,k}^N u_{i,j,k}^e l_i(\xi) l_j(\eta) l_k(\gamma)$
 - High-order at low cost! (**Level 3 BLAS!**)
- Too expensive to assemble matrices
 - Element stiffness matrices $A_{i,j}^k$ with $\mathcal{O}(N^6)$ **non-zeros**
- Matrix free formulation, key to achieve good performance in SEM
 - Unassembled matrix $A_L = \text{diag}\{A^1, A^2, \dots, A^E\}$ and functions $u_L = \{u^e\}_{e=1}^E$
 - Operation count is **only $\mathcal{O}(N^4)$ not $\mathcal{O}(N^6)$**
 - Boolean gather/scatter matrix Q^T and Q
 - Ensure continuity of functions on the element level $u = Q^T u_L$ and $u_L = Qu$
- Q and Q^T formed, only the action QQ^T is used
 - Matrix-vector product $w = Au \Rightarrow w_L = QQ^T A_L u_L$

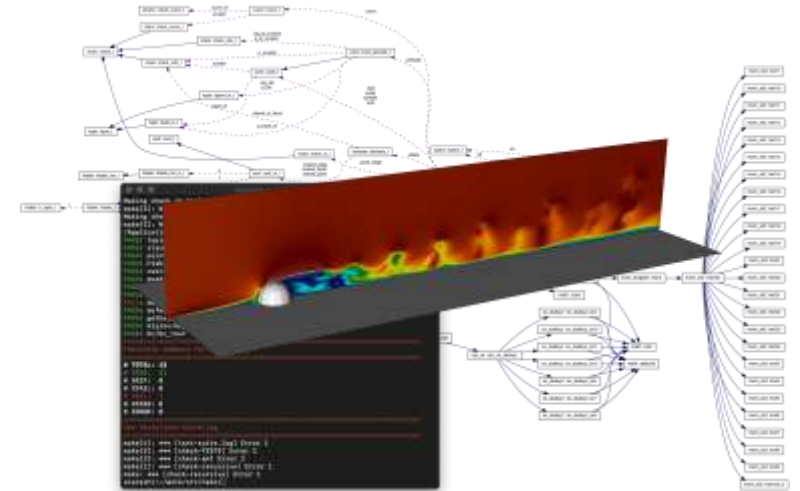
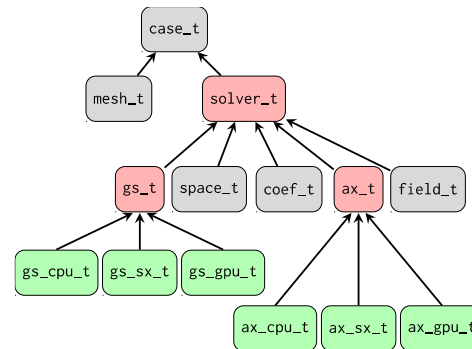


Portable Spectral Element Framework *NEKO*

- High-order spectral element flow solver
 - Incompressible Navier-Stokes equations
 - Matrix-free formulation, **small tensor products**
 - **Gather-scatter** operations between elements
- Modern **object-oriented** approach (Fortran 2008)

```
! Base type for a matrix-vector product providing Ax
type, abstract :: ax_t
contains
  procedure(ax_compute), nopass, deferred :: compute
end type ax_t

! Abstract interface for computing Ax
abstract interface
  subroutine ax_compute(w, u, coef, msh, Xh)
    implicit none
    type(space_t), intent(inout) :: Xh
    type(mesh_t), intent(inout) :: msh
    type(coef_t), intent(inout) :: coef
    real(kind=dp), intent(inout) :: w(:,:,:)
    real(kind=dp), intent(inout) :: u(:,:,:)
  end subroutine ax_compute
end interface
```



- Various hardware-backends
 - CPUs, GPUs down to exotic vector processors and FPGAs
 - **Device abstraction layer** for accelerators (CUDA/HIP/OpenCL)
 - Modern software engineering (pFUnit, ReFrame, **Spack**)



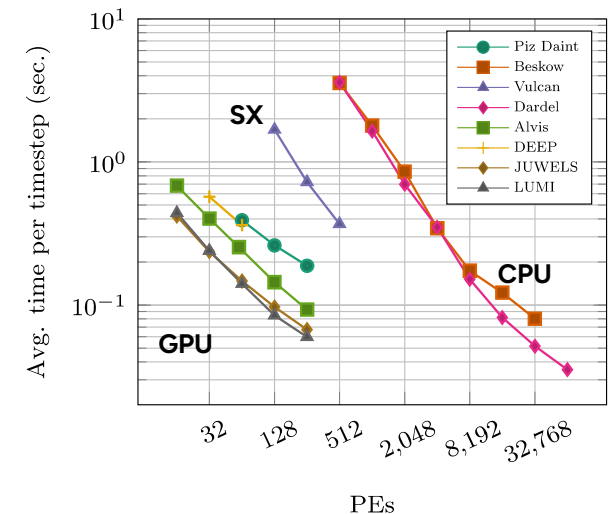
```
> spack install neko+cuda
```



ExtremeFLOW/neko

www.neko.cfd

Neko, Taylor-Green vortex, $Re = 5000$



Device Abstraction Layer

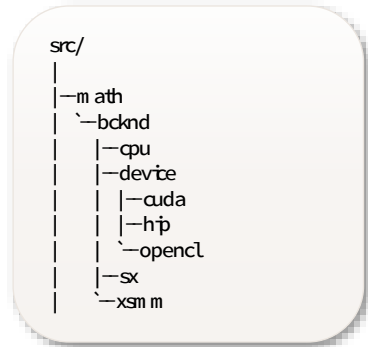
How to interface Fortran with accelerators?

- Native CUDA/HIP/OpenCL implementation via C-interfaces
- Device pointers in each derived type

```

type field_t
  real(kind=rp), allocatable :: x(:, :, :, :) !< Field data
  type(space_t), pointer :: Xh !< Function space
  type(mesh_t), pointer :: msh !< Mesh
  type(dofmap_t), pointer :: dof !< Dofmap
  type(c_ptr) :: x_d = C_NULL_PTR !< Device pointer
end type field_t

```



```

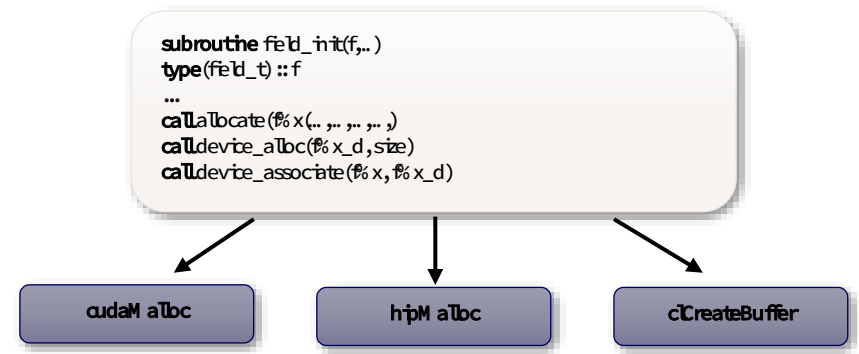
!> Enum @a hipError_t
enum, bind(c)
  enumerator :: hipSuccess = 0
  ...
end enum

!> Enum @a hipMemcpyKind
enum, bind(c)
  enumerator :: hipMemcpyHostToHost = 0
  enumerator :: hipMemcpyHostToDevice = 1
  ...
end enum

interface
  integer (c_int) function hipMalloc(ptr_d, s) &
    bind(c, name='hipMalloc')
  use, intrinsic :: iso_c_binding
  implicit none
  type(c_ptr) :: ptr_d
  integer(c_size_t), value :: s
end function hipMalloc
end interface

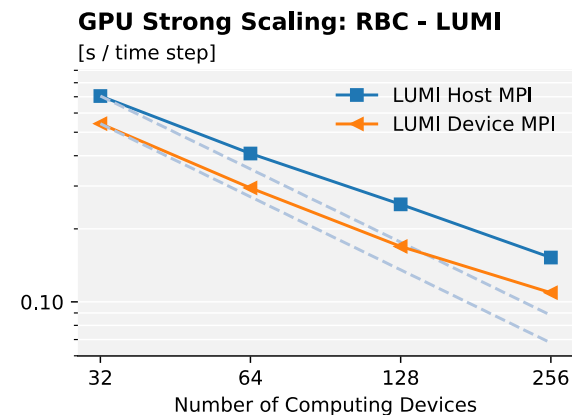
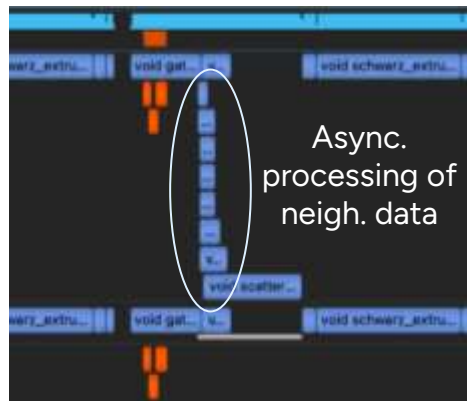
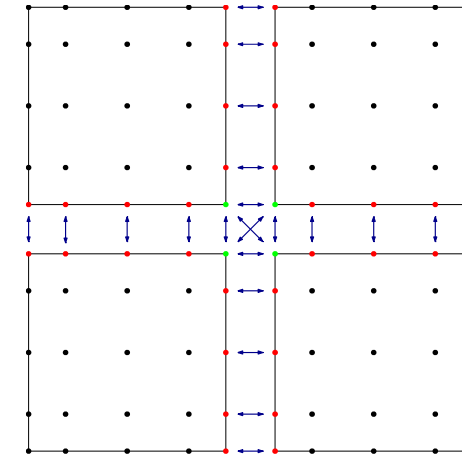
```

- Abstraction layer hiding memory management
- Hash table associating x with x_d
- Kernels invoked from the object hierarchy via C interfaces (Ax, vector ops)
 - **Wrapper functions** for each supported accelerator backend
 - **Templated** (CUDA/HIP) or **pre-processor macros** (OpenCL) for runtime parameters
- **Auto/runtime tuning** based on polynomial order



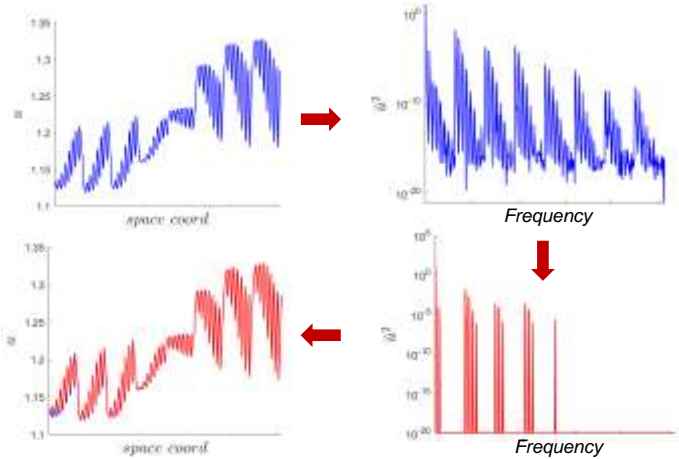
Gather-Scatter

- Uses indirect addressing and are (mostly) non-injective
- Topology aware optimisations
 - Facets (single neighbour), **red** points
 - Injective, **vectorizable** (always operating on **sorted** tuples)
 - Non facets (arbitrary number of neighbours), **green** points
 - **Cannot** be made injective, **not vectorizable** (small amount)
- Multiple levels of overlapping communication and computation
 - Overlapping with **non-blocking MPI** (device aware)
 - **Asynchronous** GPU kernels (neighbours in streams)
 - **Auto/runtime** tuning of all combinations



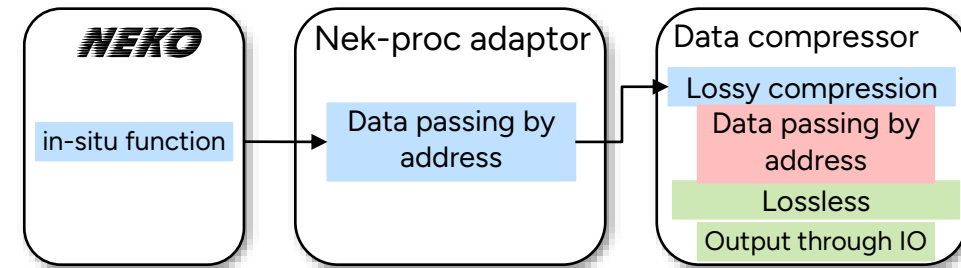
Synchronous and Hybrid Data Compression

- **Lossy compression, physics-based method:**
discard data not associated with the most energetic flow motions¹

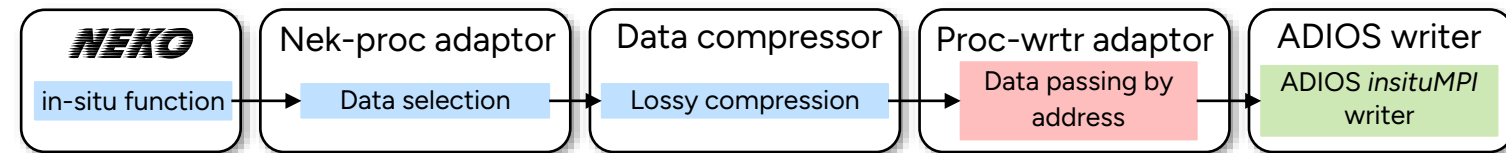


- **Lossless compression:**
ADIOS2 operator with runtime configuration
- **97% data reduction with a relative error of 2.5%**

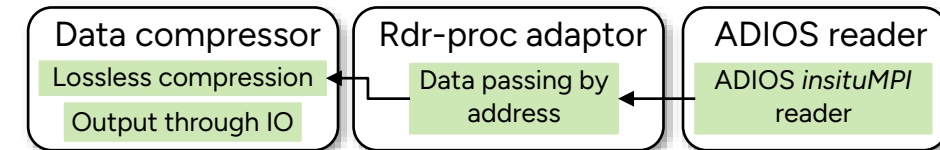
In-situ approach²



Synchronous compression



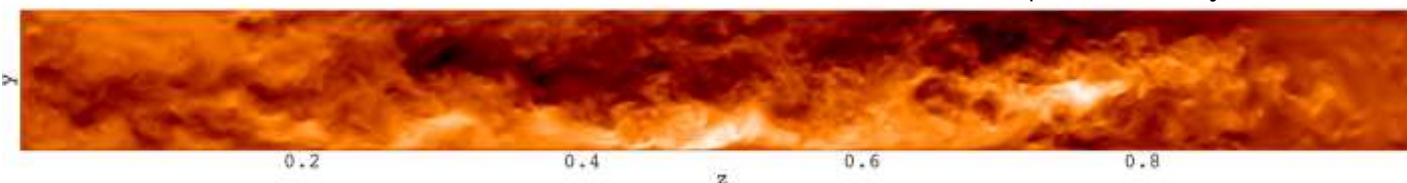
Hybrid compression



Fortran functions

C/C++ functions called in Fortran

C++ functions



Compressed velocity field $Ra = 10^{11}$

1: E. Otero et al., "Lossy data compression effects on wall-bounded turbulence: bounds on data reduction," Flow, Turbulence and Combustion, vol. 101, no. 2, pp. 365–387, 2018.

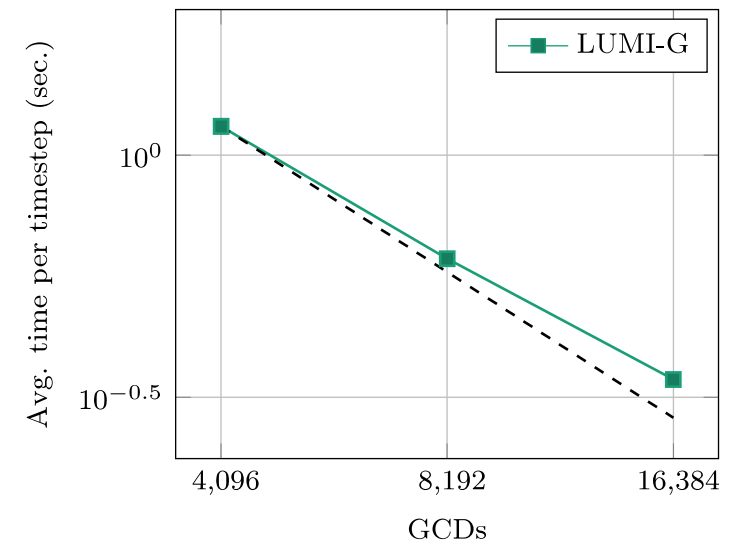
2: Y. Ju et al., "In-Situ Techniques on GPU-Accelerated Data-Intensive Applications," eScience, 2023.

Performance Baseline

- Full machine runs towards the end of the LUMI-G pilot phase
- DNS of flow past a circular cylinder at $Re = 50,000$
 - 113M elements
 - 7th order polynomials (8 GLL points)
- Simulation restarted from prebaked low-order runs
 - Restart checkpoint: 453GB
 - Extrapolated to 7th order polynomials
 - Computed solution (snapshot): 1.5TB
- Preliminary results
 - Achieved close to 80% parallel efficiency
 - Using 20%, 40% and 80% of the entire machine



Cylinder Re 50k, 113M el., 7th order poly.



Numerical Method $P_N - P_N$

- Time integration is performed using an implicit-explicit scheme (BDF k /EXT k)

$$\sum_{j=0}^k \frac{b_j}{dt} u^{n-j} = -\nabla p^n + \frac{1}{Re} \nabla^2 u^n + \sum_{j=1}^k a_j (u^{n-j} \cdot \nabla u^{n-j} + f^n)$$

with b_k and a_k coefficients of the implicit-explicit scheme, solving at time-step n

$$\Delta p^n = \sum_{j=1}^k a_j (u^{n-j} \cdot \nabla u^{n-j} + f^n)$$

$$\frac{1}{Re} \Delta u^n - \frac{b_0}{dt} u^n = \nabla p^n + \sum_{j=1}^k \left(\frac{b_j}{dt} u^{n-j} + a_j (u^{n-j} \cdot \nabla u^{n-j} + f^n) \right)$$

- Three velocity solves using CG with block Jacobi preconditioner (**fast**)
- One Pressure solve using GMRES with an additive overlapping Schwarz preconditioner (**expensive**)

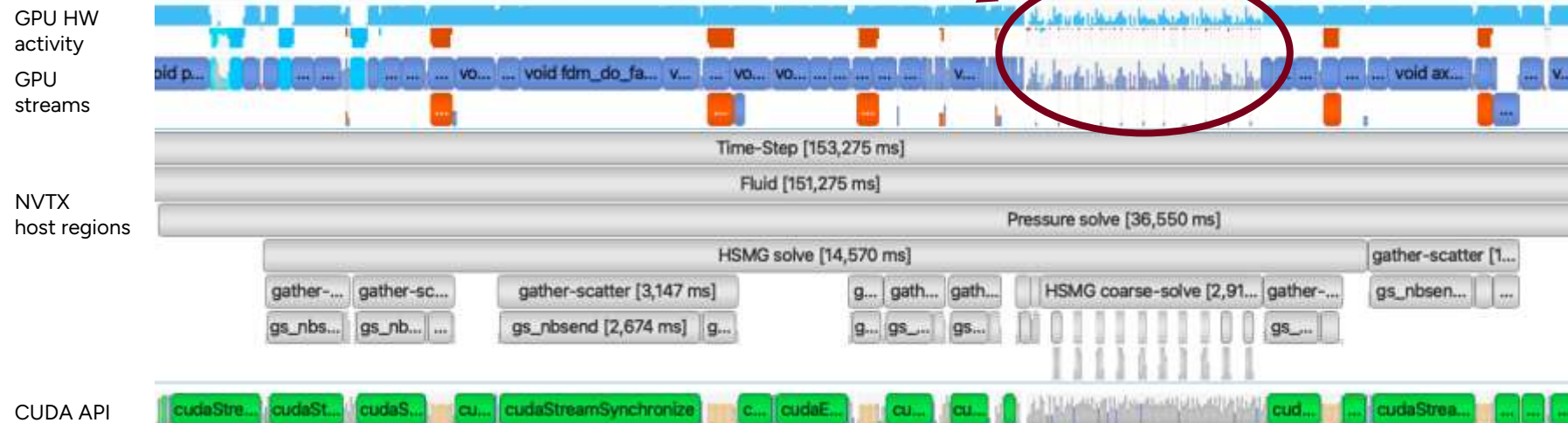
$$M_0^{-1} = \underbrace{R_0^T A_0^{-1} R_0}_{\text{Coarse grid (linear elements)}} + \sum_{k=1}^K R_k^T \tilde{A}_k^{-1} R_k, \text{ key is to have a } \mathbf{scalable\ coarse\ grid\ solver}$$

Coarse grid (linear elements)

Additive Schwarz Preconditioner on GPUs

- Coarse grid solved using an approximate Krylov solver
 - Preconditioned Pipelined Conjugate Gradient with a low, maximum iteration limit
- Low computational efficiency on GPUs
 - A_0 is on linear elements, too little data to keep the GPU busy.
 - Many small kernels, dominated by kernel launch latency

$$M_0^{-1} = R_0^T A_0^{-1} R_0 + \sum_{k=1}^K R_k^T \tilde{A}_k^{-1} R_k$$



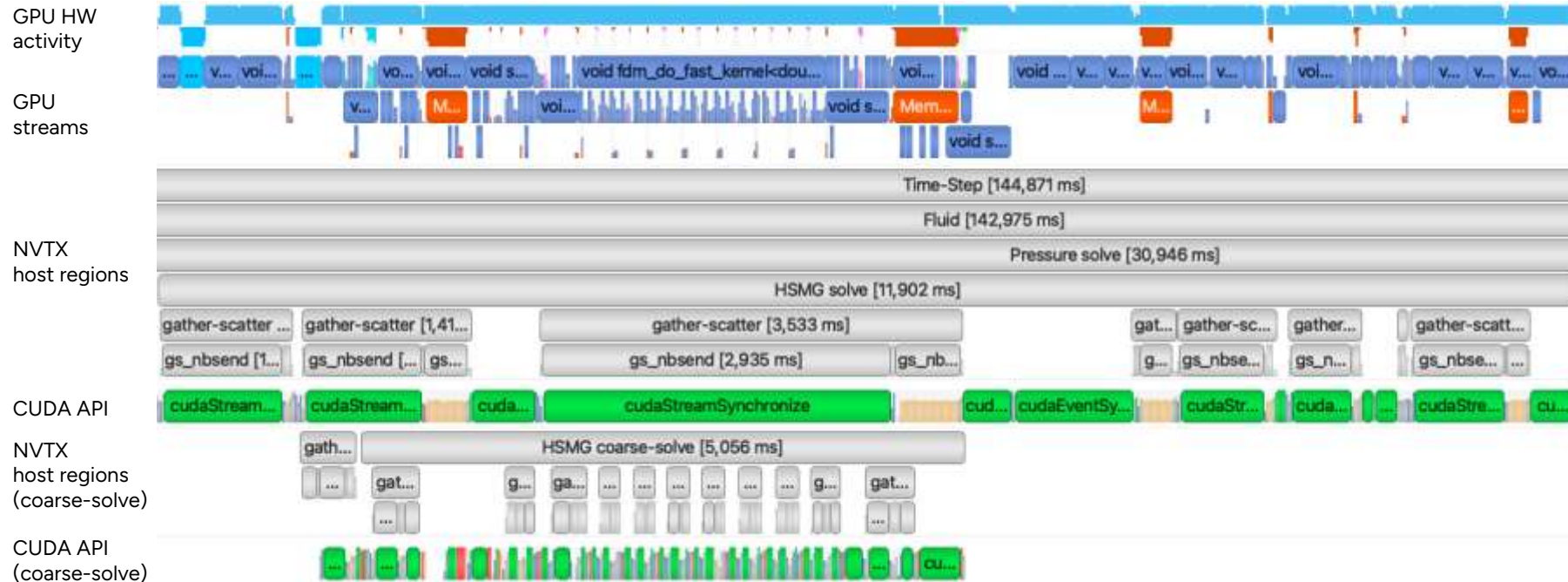
Task-decomposed Overlapped Preconditioner

- Exploit available **task-parallelism**
 - Launch the left and right part of M_0^{-1} in parallel on the device
 - Launch independent work in parallel from **different threads** in an OpenMP region
 - Launch tasks in **separate streams** to allow overlap and increase GPU utilization
 - Maximise kernel overlap using **stream priority** to ensure progress in both stream

$$M_0^{-1} = R_0^T A_0^{-1} R_0 + \sum_{k=1}^K R_k^T \tilde{A}_k^{-1} R_k$$

Thread 0
Thread 1

Stream 1
Stream 2



Performance Results

- Performance measurements on two of the EuroHPC-JU pre-exascale supercomputers **LUMI** and **Leonardo**
- Experiments were performed between
 - March–April 2023 on LUMI
 - April 2023 on Leonardo (pre-production)
- RBC in a cylinder with aspect ratio 1:10
 - $Ra = 10^{15}$
 - 108M elements, 7th order polynomials
 - 37B unique grid points and more than 148B degrees of freedom
- Strong Scalability
 - Average time per timestep (after transient)
- One MPI rank per logical GPU
 - One rank per GCD (AMD)
 - One rank per device (Nvidia)

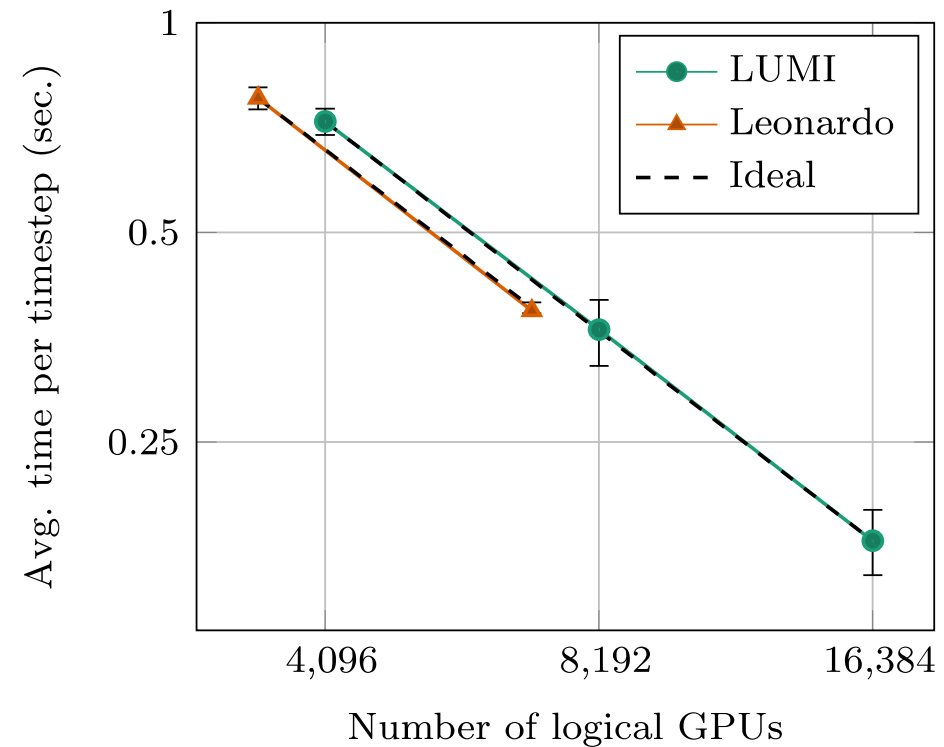


System	LUMI	Leonardo
Computing device	AMD MI250X	Nvidia A100 (custom)
Peak Tflop FP64/s	47.9 (95.7 Matrix)	11.2 (22.4)
Peak BW/s	3300	1640
No. devices	10240	13824
Interconnect	HPE Slingshot 11 200 GbE NICs (4x200 Gb/s)	Nvidia HDR 2x(2x100 Gb/s)
MPI	Cray MPICH 8.1.18	OpenMPI 4.1.4
Compiler	CCE 14.0.2	GCC 8.5.0
GPU Driver	5.16.9.22.20	520.61.05
CUDA/ROCm	ROCm 5.2.3	CUDA 11.8

Performance Results

- Close to perfect parallel efficiency on both LUMI and Leonardo
- Close to perfect parallel efficiency with less than 7000 elements per logical GPU
- Significantly reducing the smallest required problem size for strong scalability limits
- Improvements mainly due to the new overlapped pressure preconditioner

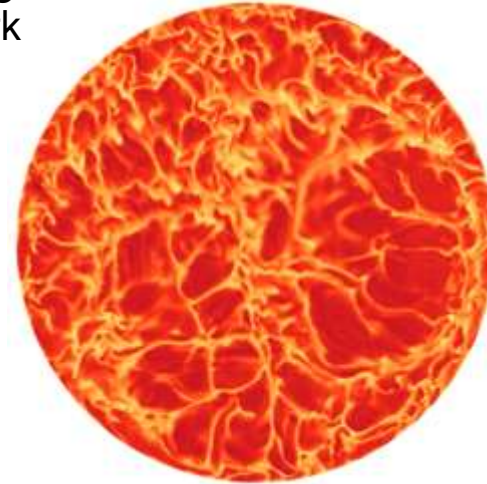
RBC Ra 10^{15} , 108M el., 7th order poly.



99% confidence intervals is illustrated as error bars

Summary

- Insight into Rayleigh-Bénard convection
 - The question about an ultimate regime can only be settled through simulations made possible through the developments in this work
- In-situ data processing
 - Hybrid data compression, streaming data to the CPU for online post-processing while the simulation continues to run on the GPU
 - New ways of analysing and processing data from simulations
- Task-decomposed overlapped pressure preconditioner
 - Expressing more of the available concurrency of the application
 - Key ingredient to achieve good strong scalability on LUMI and Leonardo





Solving Large Systems at Exascale on GPU

Finite Element Solvers

**EUROHPC
USER DAY
2023** Brussels
11.12.23



Project: *“Excalibur SysGenX”*

EuroHPC used: LUMI-G

Speaker: *Chris Richardson (University of*

Abstract

- Extreme scale simulations for science and engineering
- A framework to translate mathematics to a model
- Solving very large linear algebra problems on GPU
- Results from LUMI-G are promising



Motivation

ExCALIBUR
10

Virus characterisation

Stiffness and Deflection Analysis of Complex
Structures

M. J. TURNER,* R. W. CLOUGH,† H. C. MARTIN,‡ AND L. J. TOPP**

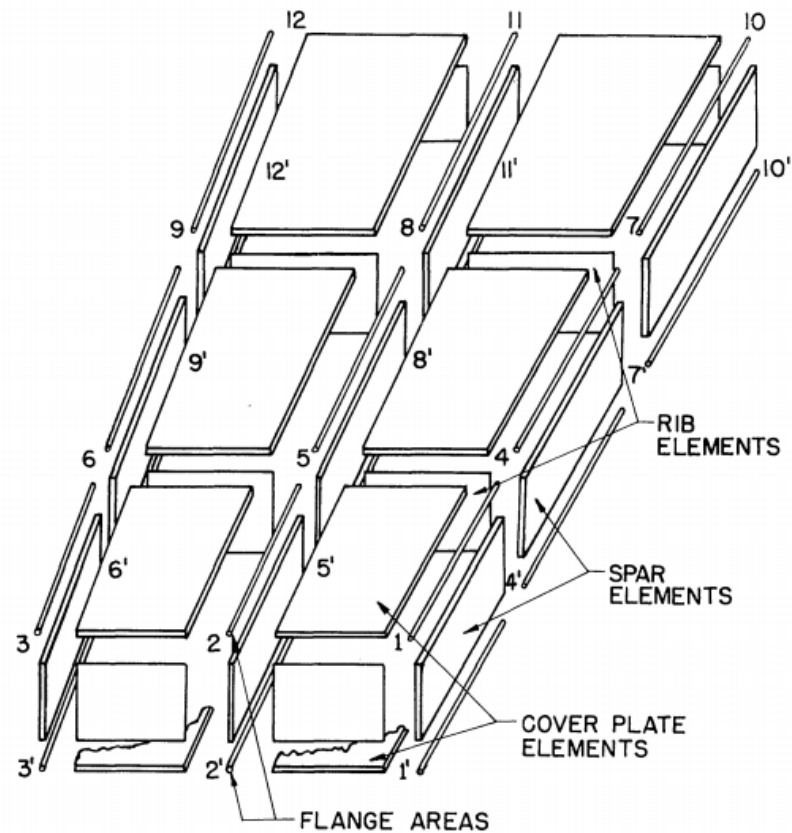
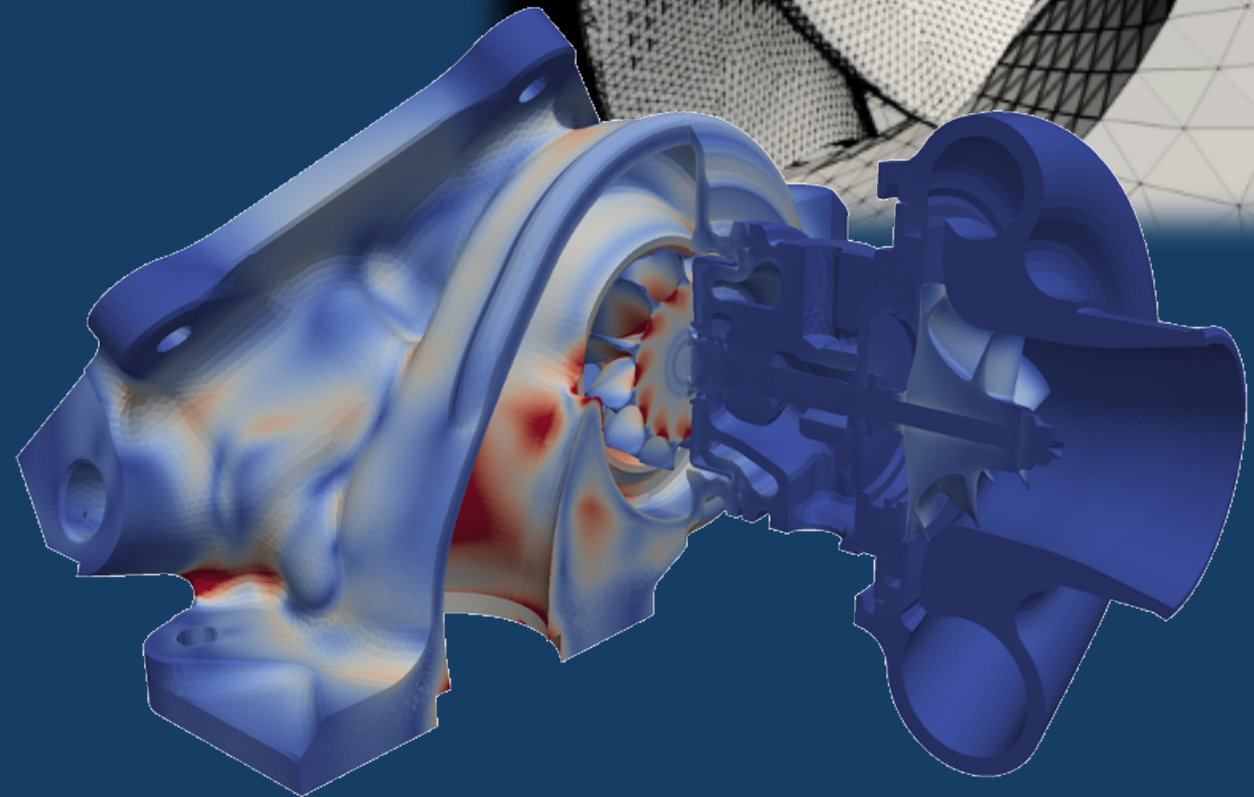


FIG. 3. Wing structure breakdown.

Finite Element Method (FEM)



FEniCS Project



FENICS
PROJECT
fenicsproject.org

- Domain Specific Language (DSL) to describe equations
- Turns symbolic code into machine instructions
- Examples: Poisson, Helmholtz, Maxwell, Stokes, Elasticity, etc.

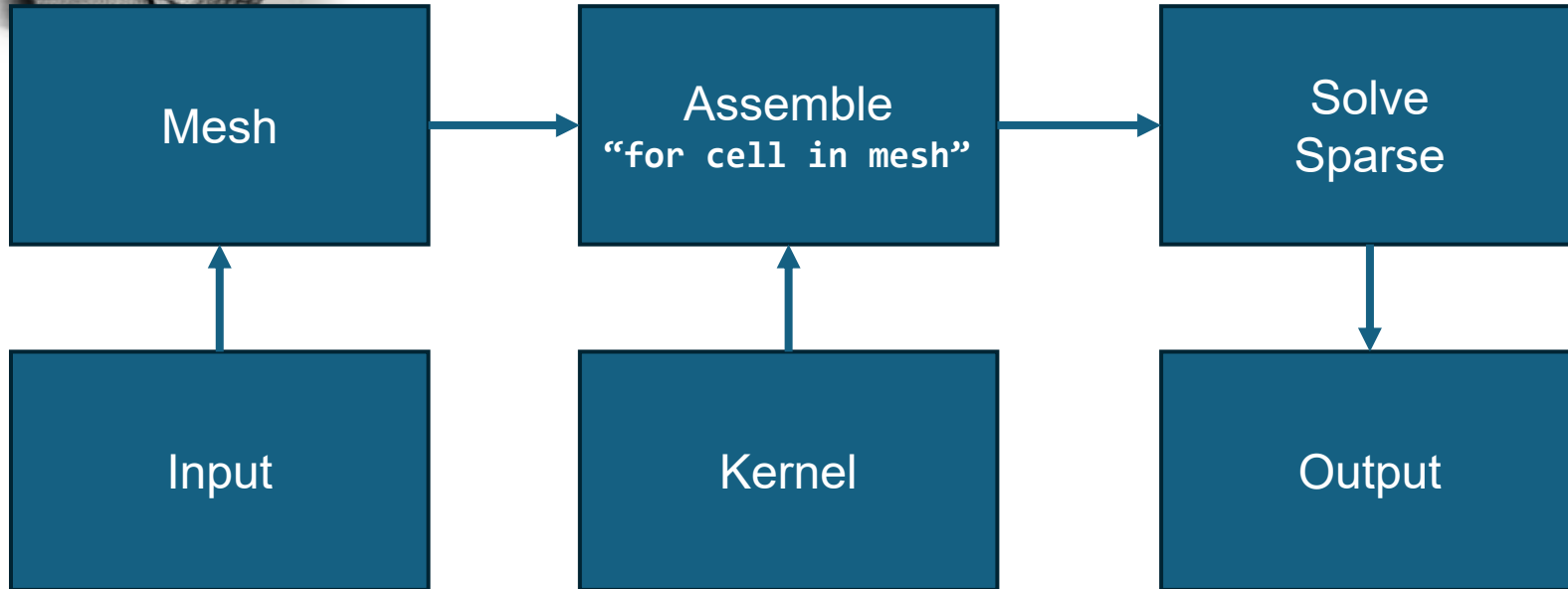
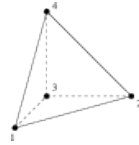
$$\nabla^2 u + k^2 u = f \quad (\text{inner}(\text{grad}(u), \text{grad}(v)) + k^{**2}*u*v - f*v)*dx$$

$$\nabla^2 u = \rho \quad (\text{inner}(\text{grad}(u), \text{grad}(v)) - \rho*v)*dx$$

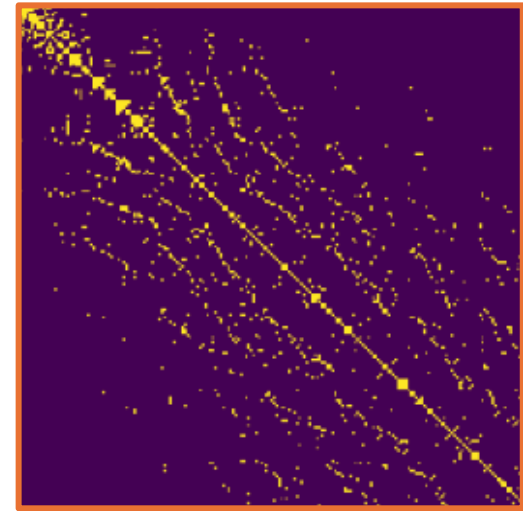
$$\frac{1}{\mu} \nabla \times \nabla \times A + \sigma \frac{\partial A}{\partial t} = 0$$

$$\rho \left(\frac{\partial u}{\partial t} + u \cdot \nabla u \right) - \mu \nabla^2 u + \nabla p = F \quad (\text{inner}(\text{curl}(A), \text{curl}(Av))/\mu + \text{sigma}*(A_t-A)*Av)*dx$$

FEniCS FEM workflow



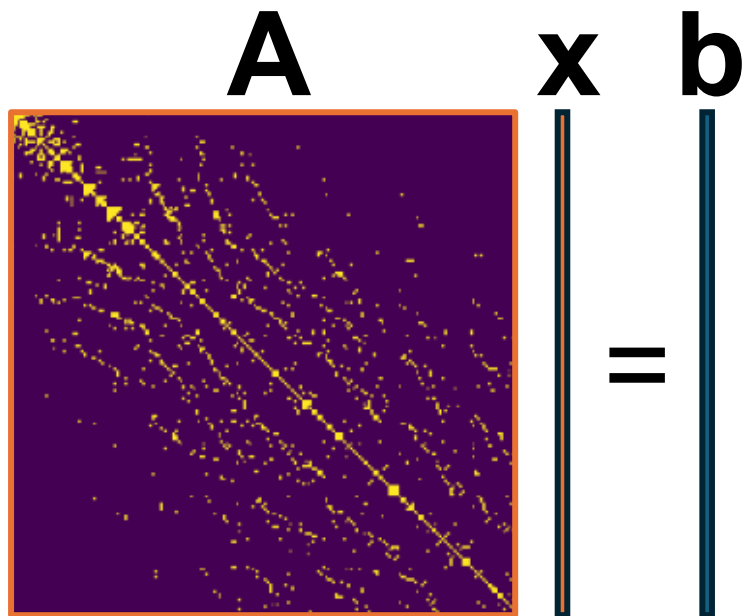
```
inner(grad(u), grad(v))*dx
```



$$Ax = b$$

Solving sparse $Ax=b$

A, x and b are distributed across MPI ranks...



Direct methods:

- Gaussian elimination (with pivoting)
- LU or Cholesky factorisation (multifrontal, parallel)
- Good stability for well-formed matrix
- OK for $N < 10^6$

Time complexity = $O(N^3)$

Iterative methods:

- Krylov subspace methods: CG, GMRES, etc.
- May be difficult to converge

Time complexity $\sim O(N^{3/2})$

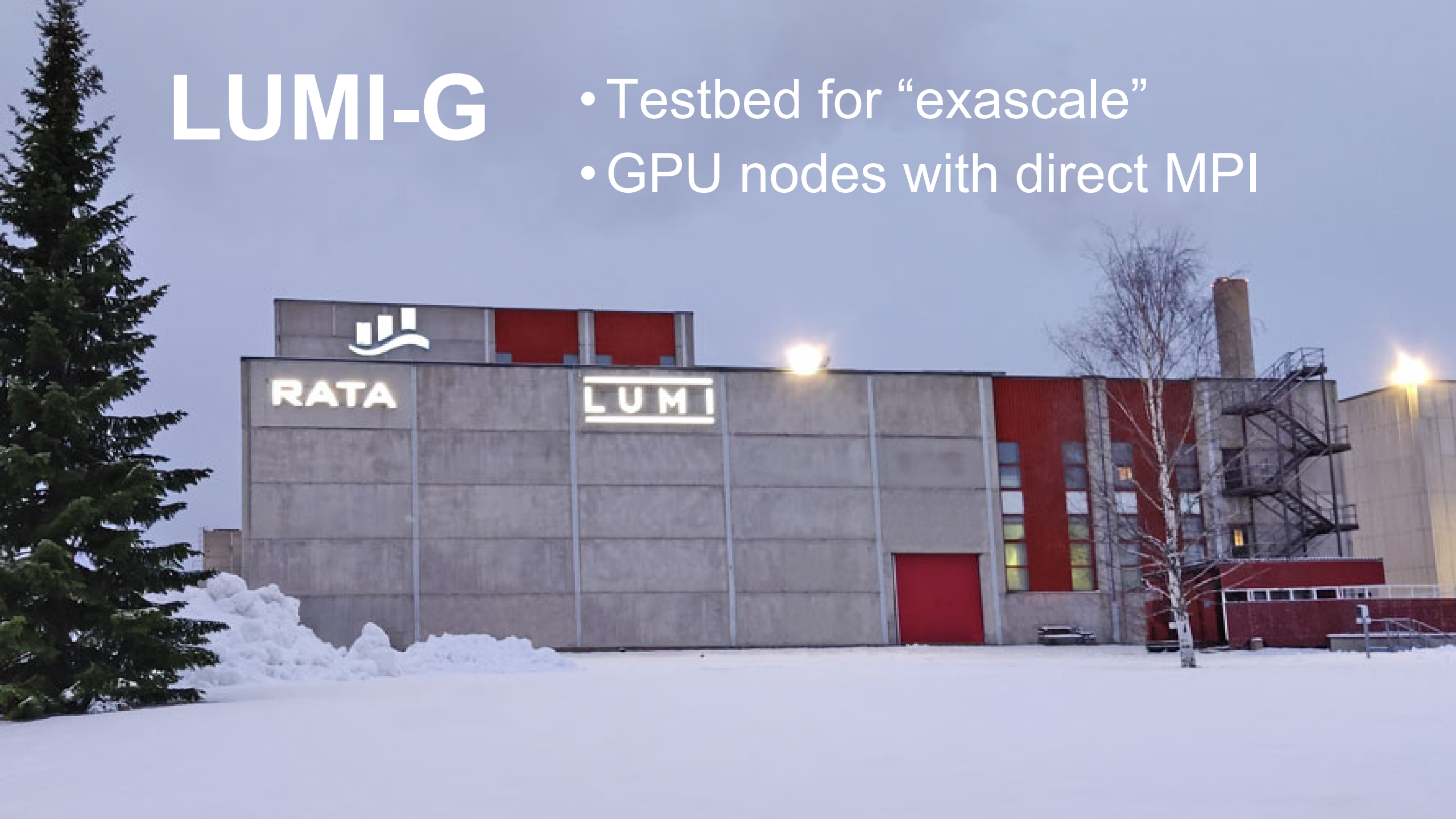
Preconditioned iterative methods:

- e.g. CG+Multigrid

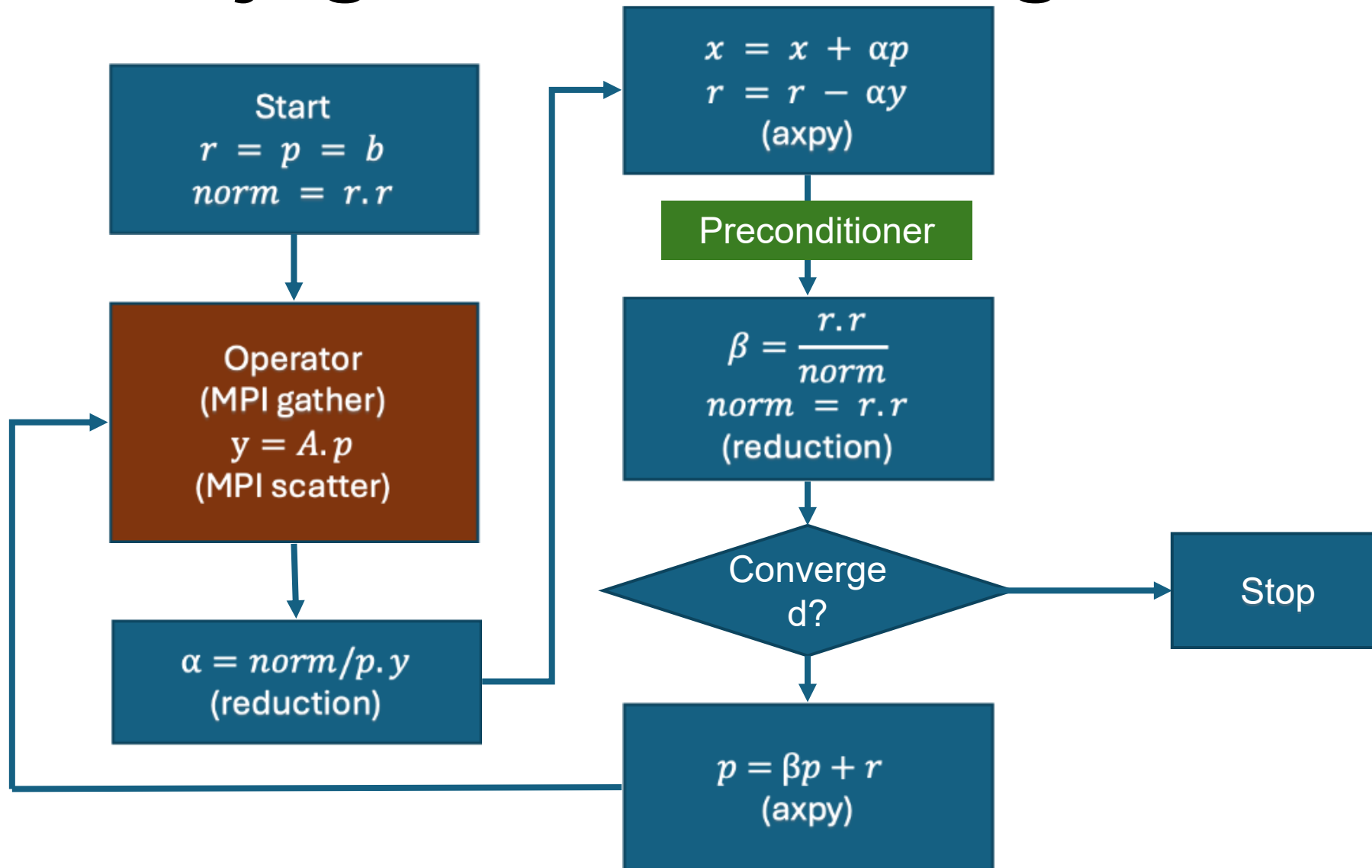
Can be $O(N)$ – i.e. perfect weak scaling!

LUMI-G

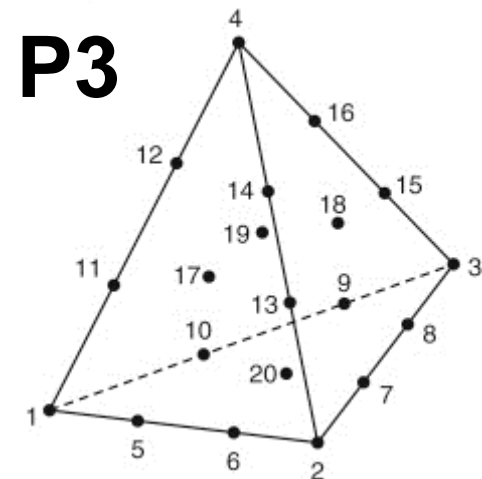
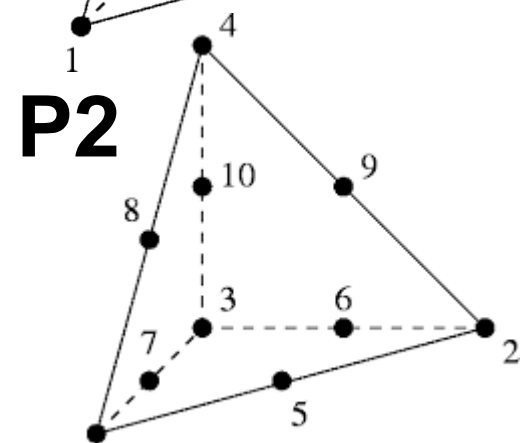
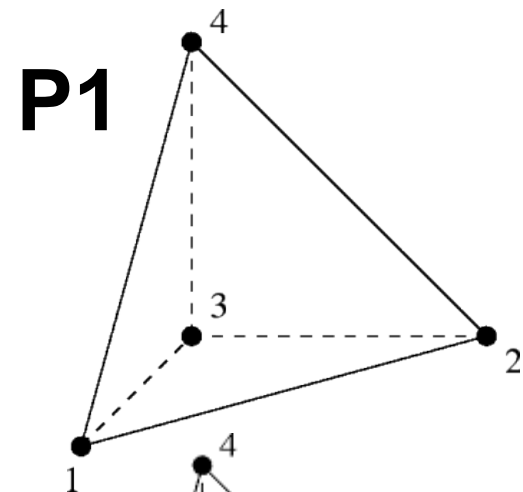
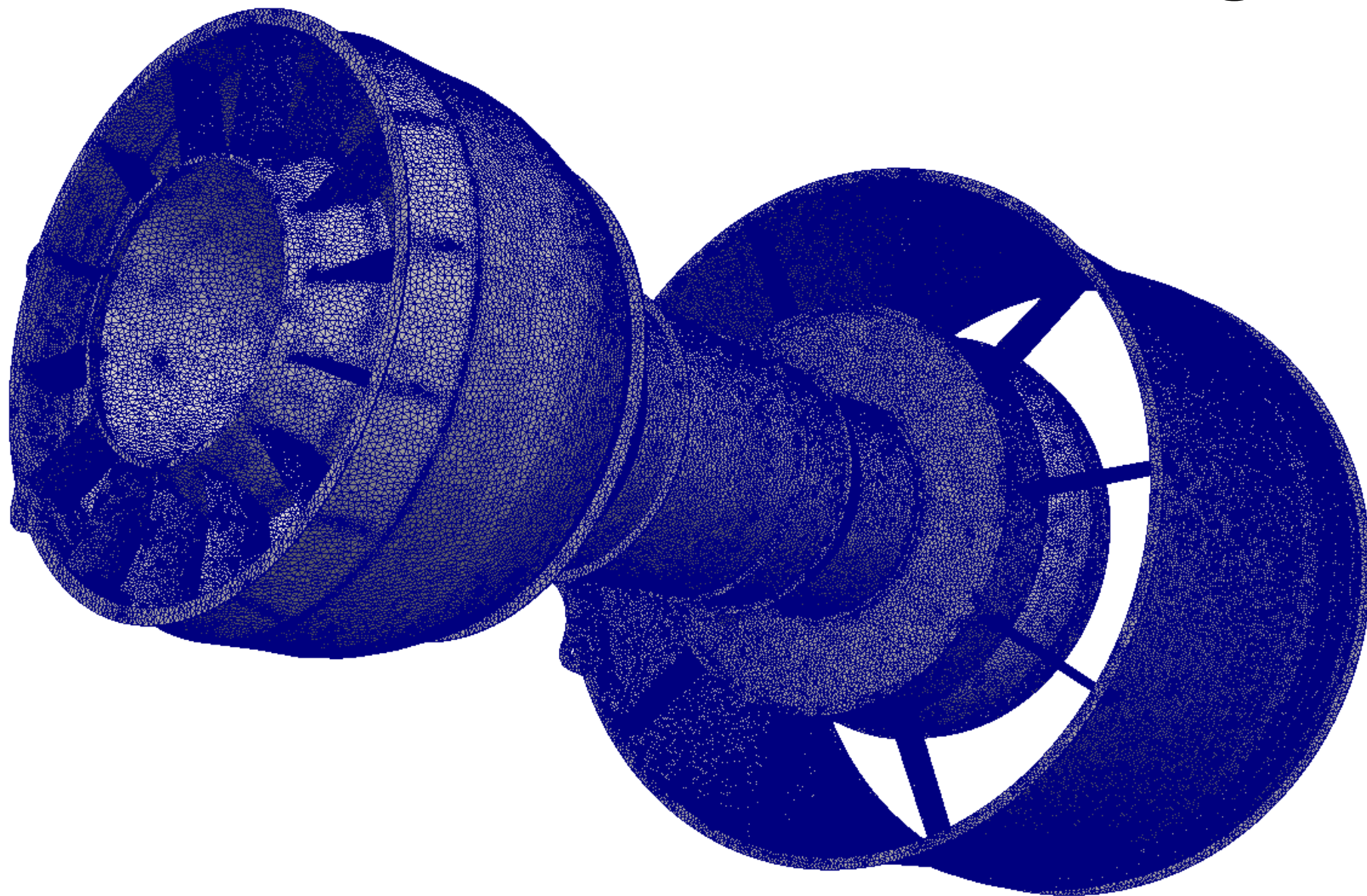
- Testbed for “exascale”
- GPU nodes with direct MPI



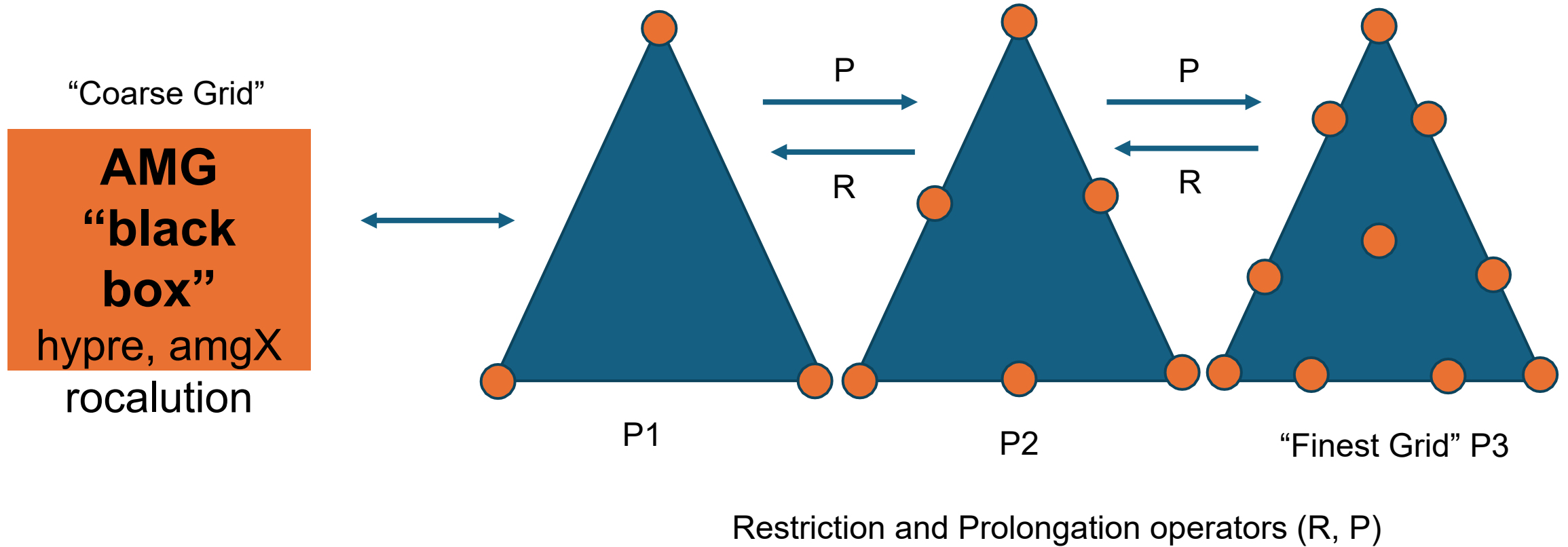
Conjugate Gradient Algorithm



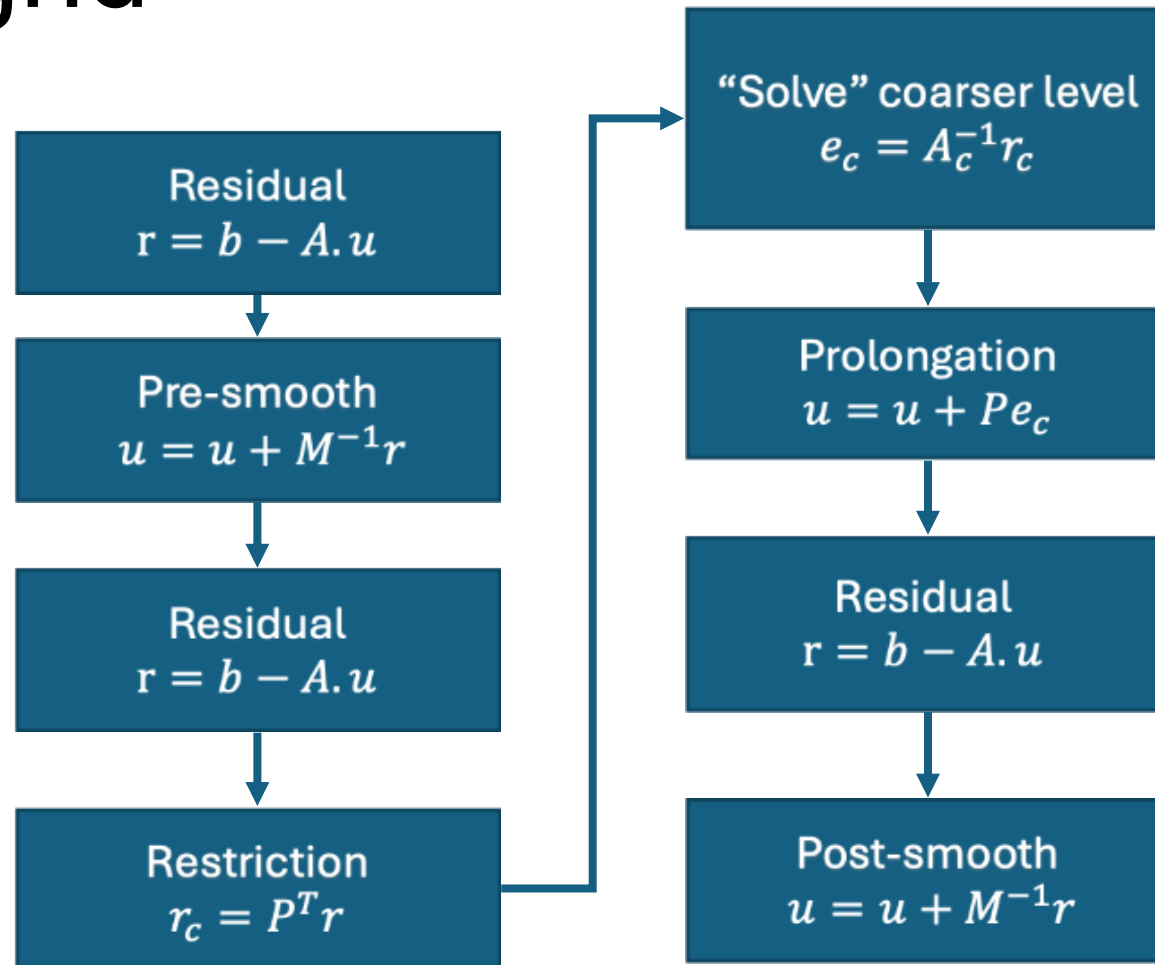
P-refinement multigrid



Preconditioning: p -multigrid



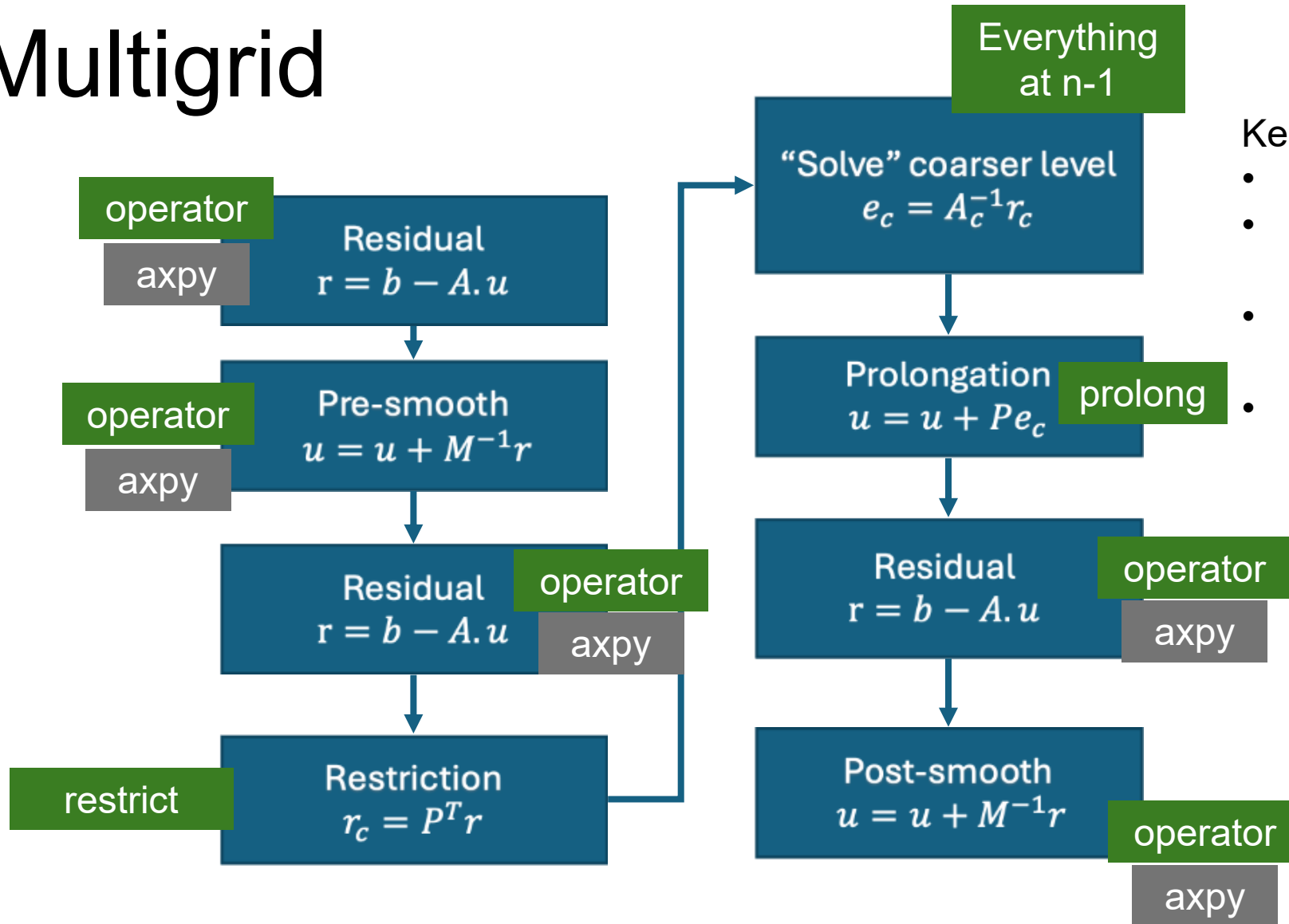
Multigrid



Kernels:

- scatter/gather (MPI)
- MatVec (prolong, restrict)
- Operator "A" (residual, smoother)
- axpy

Multigrid



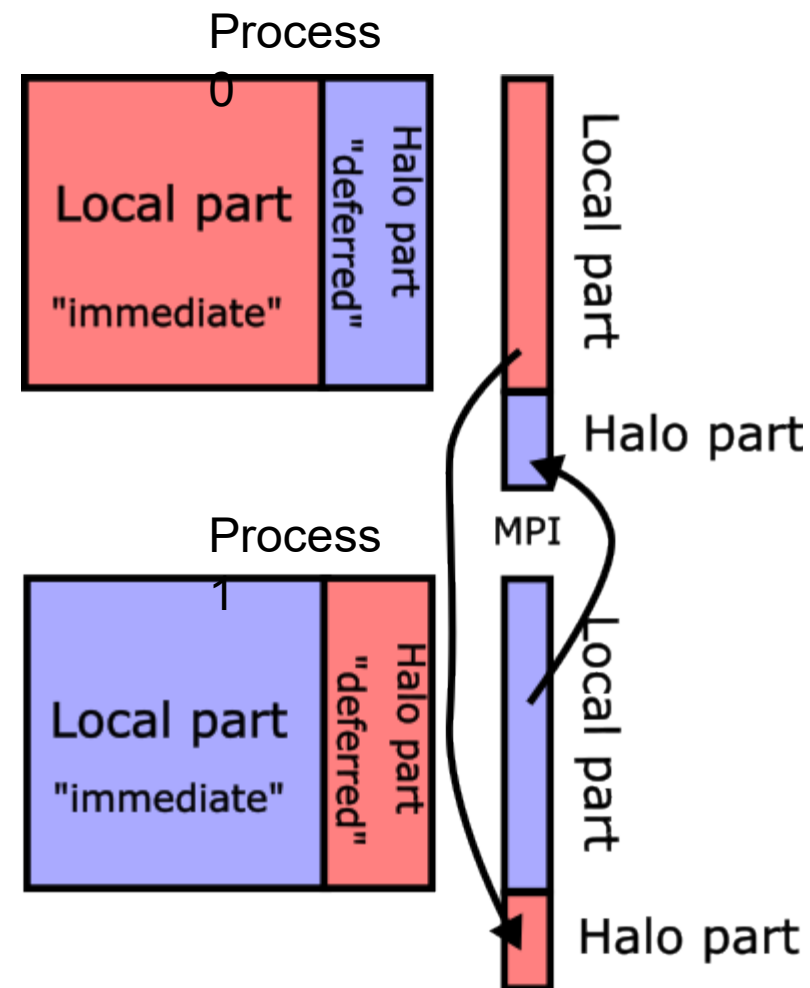
Kernels:

- scatter/gather (MPI)
- MatVec (prolong, restrict)
- Operator "A" (residual, smoother)
- aaxy

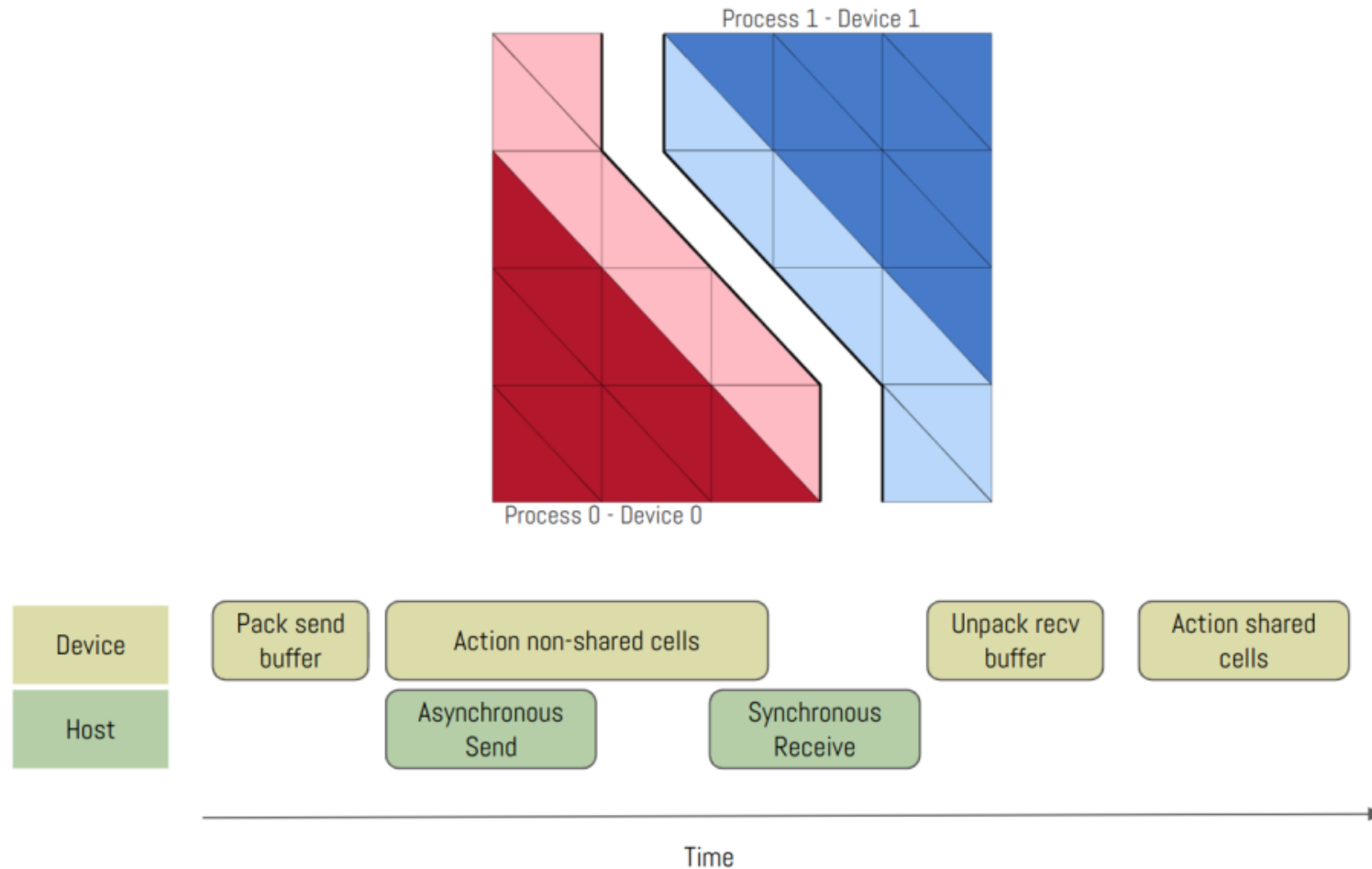
Forward operator

$$y=Ax$$

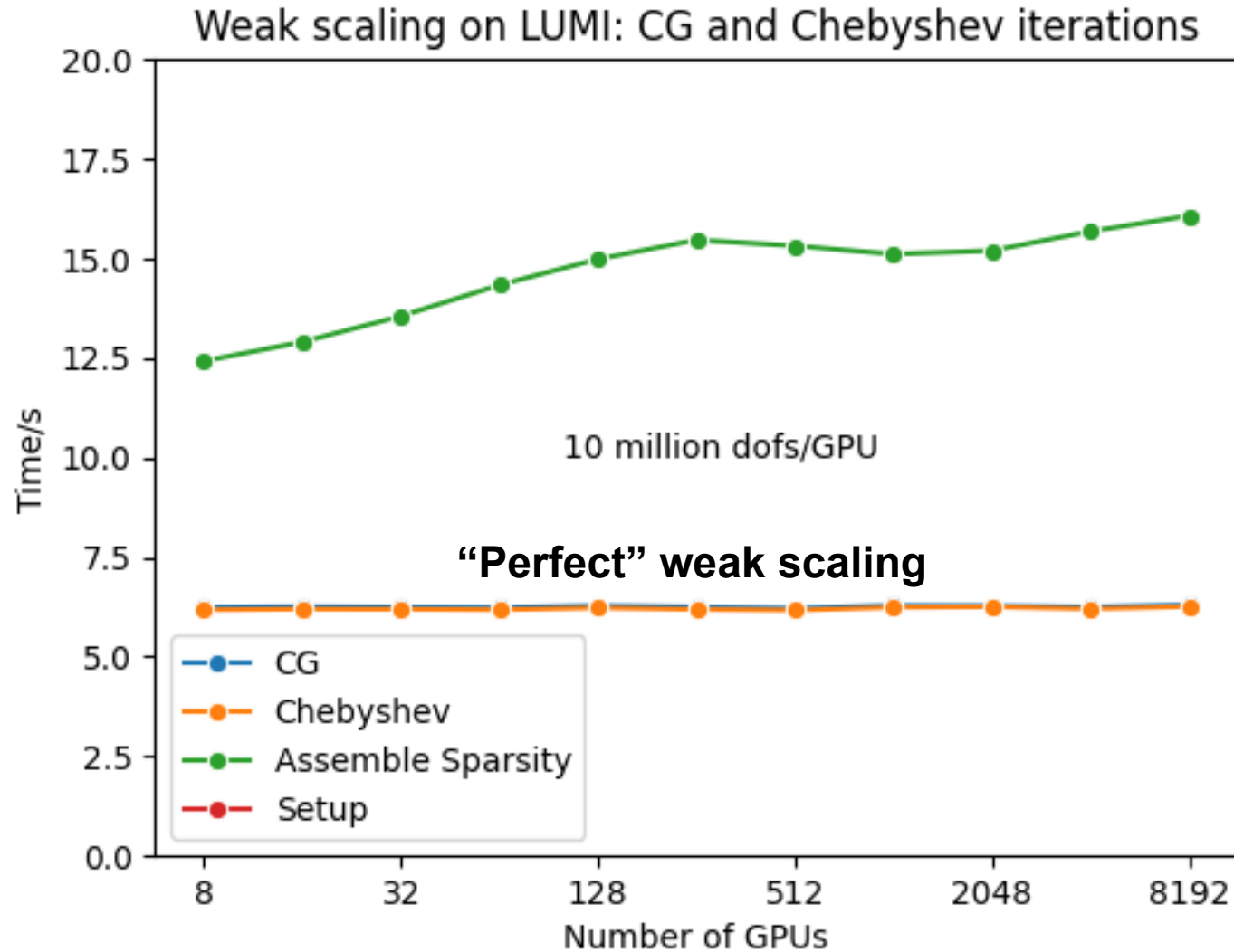
- Forward operator is the most expensive part of the solver
- Maybe don't need to form matrix, just "action"?
- When "matrix" is split across processes, need to update (MPI) RHS (x) before and LHS (y) after each MatVec...
- Off-process part is small, so can overlap computation of the local part of the MatVec with communication of the off-process part



Without forming a matrix



Results from LUMI-G



Summary

- Very large FEM models can be solved with multigrid methods
- Distributed Sparse Matrix-Vector product is the main bottleneck
- We can get good scaling by overlapping communication and computation
- LUMI-G results are promising



UNIVERSITY OF
CAMBRIDGE



A wireframe bear is shown in a futuristic server room. The bear is composed of white lines forming a mesh, and it is positioned in the center-left of the frame. The server room is filled with blue light, with vertical beams of light illuminating the floor and the server racks. The server racks are on the right side of the image, and they are filled with various components and lights. The overall atmosphere is high-tech and digital.

Legio: a Framework for Fault Resilience in MPI

**EUROHPC
USER DAY
2023** Brussels
11.12.23



Project: EHPC-DEV-2023D10-018

EuroHPC used: Karolina

Speaker: *Roberto ROCCO (Politecnico di Milano)*

A (very) brief introduction to MPI



- Message Passing Interface
- De-facto standard for inter-process communication at scale since 1994
- Many communication schemes supported:
 - Point to point
 - Collective routines
 - Remote memory access
 - I/O
 - Process topologies
- Continuously developing with new features and possibilities

How do we use our HPC clusters

ECP Milestone Report

A Survey of MPI Usage in the U. S. Exascale Computing Project
WBS 2.3.1.11 Open MPI for Exascale (OMPI-X) (formerly WBS
1.3.1.13), Milestone STPM13-1/ST-PR-13-1000

David E. Bernholdt^{1,*}, Swen Boehm¹, George Bosilca²,
Manjunath Gorentla Venkata¹, Ryan E. Grant³, Thomas Naughton¹,
Howard P. Pritchard⁴, Martin Schulz^{5,6}, and Geoffroy R. Vallee¹

Question 53: **How do you plan to make your application fault tolerant?**

- Why do we need such an analysis?

Faults in HPC scenario

Silent Data Corruptions at Scale

Cores that don't count

Peter
P
Jeff
Sun

HOME > NEWS > HPC & QUANTUM

Frontier supercomputer suffering 'daily hardware failures' during testing

Being exascale ain't easy, ORNL's Justin Whitt says teething troubles are normal

October 10, 2022 By: Dan Swinhoe [Have your say](#)

The MPI standard

After an error is detected, the state of MPI may become undefined.

State of the Art Solutions

No.	Question and Responses	AD	ST	Overall
53	How do you plan to make your application fault tolerant? <i>(single+text)</i>			
	It is already fault tolerant	4%	0%	2%
	I plan to use checkpoint/restart	61%	32%	46%
	Don't know	18%	25%	21%
	I'm not going to worry about fault tolerance	7%	18%	12%
	<i>Other responses</i>			
	Avoid use of MPI	0%	7%	4%
	Data checksums between memory and storage	0%	4%	2%
	Legion capabilities in addition to checkpoint/restart	4%	0%	2%
	Local-failure/local-recovery	0%	4%	2%
	MPI Reinit	4%	0%	2%
	MPI ULFM	4%	0%	2%
	MPI fault tolerance features in addition to checkpoint/restart	4%	0%	2%
	Selective reliability	0%	4%	2%
	Skeptical programming	0%	4%	2%
	Task-based capabilities in addition to checkpoint/restart	4%	0%	2%
	Task-based rollback/recovery, replication	0%	4%	2%
	Treat as proper distributed system, with group membership	0%	4%	2%

State Resilience
Mitigate damage,
loose priority

Execution Resilience
Continue execution,
but expertise needed,
application dependent

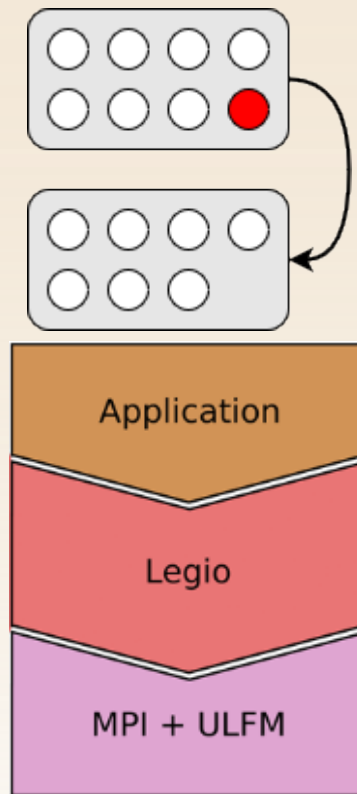
**State Resilience &
Execution Resilience**
keep priority

The Legio Fault Resilience Framework goals

- Simplify the use of the User Level Fault Mitigation extension (**ULFM**)...
 - Aim towards **minimal code intrusiveness**
- ...for MPI applications supporting **graceful degradation**...
 - Despite losing some data, they still can produce meaningful results
- ...preserving their **performance** and **scalability**.
 - Checkpoint based solutions are not scalable
 - **Faster recovery**
 - No operation running alongside the application

The Legio framework overview

- Upon fault (abrupt termination), let survivor processes finish their execution
 - Manage presence of missing process
 - Result will differ, possibly an approximation of the correct one
- Integration through PMPI: catch all the calls to the MPI layer and perform resilience operations
- No changes in behaviour in fault-free executions



Design Principles: Transparency

Using the Legio framework should require the minimum amount of code changes in the application.

- Legio operates through PMPI, no code change needed.
 - Legio provides an API to check the status of the processes but its use is not mandatory
- The user must just link Legio to the application
- For more complex functionalities (like critical process management) the user may leverage some functions present in the Legio API.

Design Principles: Flexibility

The use of Legio should not limit the application in the choice of the MPI functionalities to use.

- Legio supports most functionalities present in the latest version of the standard
 - Point-to-point
 - Collectives
 - RMA
 - I/O
 - Group collective communicator creation
 - Sessions
 - Dynamic process management

Design Principles: Efficiency

The use of Legio should not compromise the scalability and performance of the application.

- Legio operates only when performing MPI calls
 - No background thread running
- The additional code added by Legio scales at worst logarithmically with the size of communicators
- We were able to prove this point with our experiments...

The experimental campaigns

We used CPU nodes of the Karolina cluster

- 2 x AMD Zen 2 EPYC™ 7H12, 2.6 GHz
- 256 GB of RAM
- 128 MPI processes



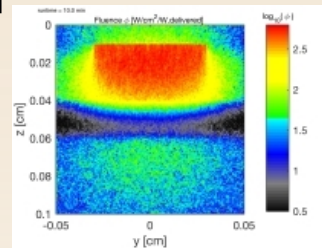
We also used Marconi100 nodes

- 2 x IBM POWER9 AC922 2.6 GHz
- 256 GB of RAM
- 32 MPI processes

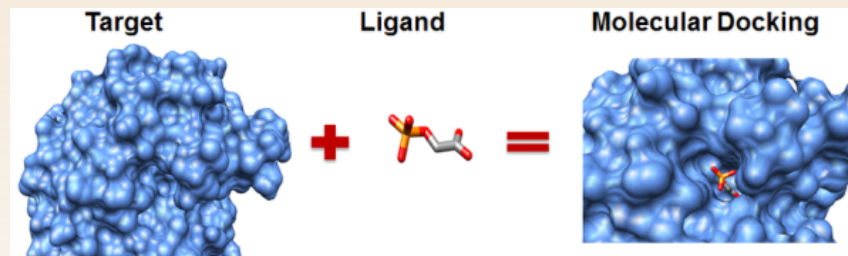


The experimental campaign applications

- Montecarlo photon simulation;



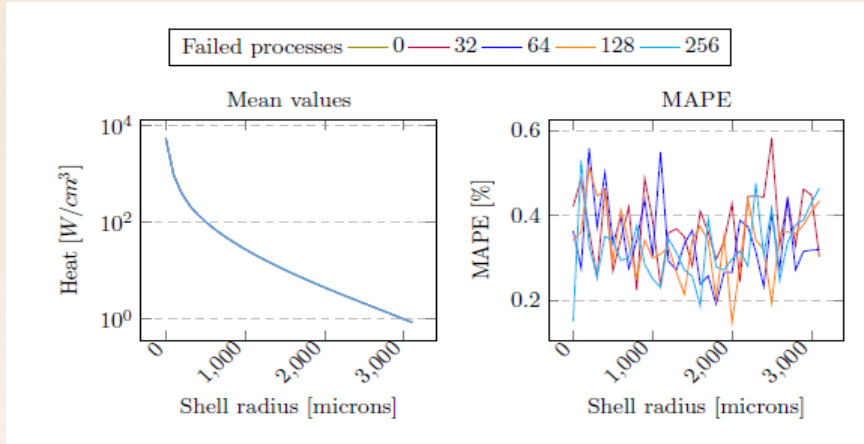
- Molecular docking;



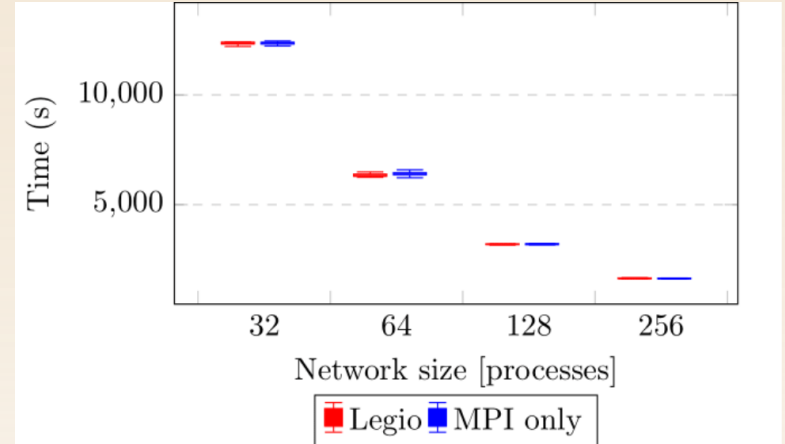
- NAS Parallel benchmarks.



Some campaign results

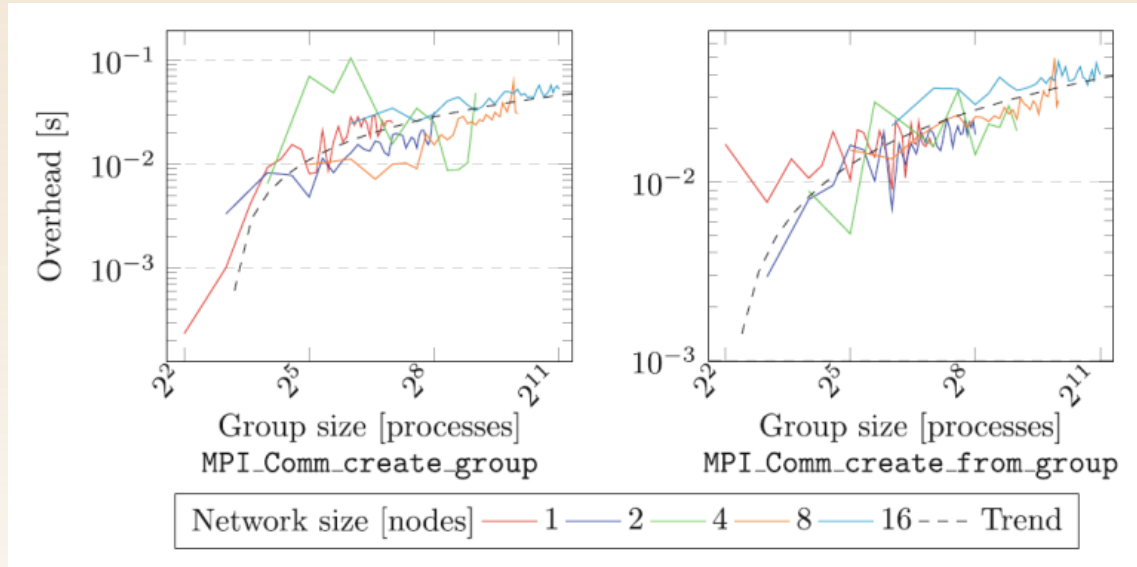


Photon simulation, using 4 Karolina nodes
Measured the accuracy loss due to faults.



Molecular docking application, using 1 to 8
Marconi100 nodes.

Some campaign results



Scalability of two group-collective communicator creation functions, executed on a varying amount of Karolina nodes.

We measure the overhead compared to an execution without fault management features

How can YOU use it?

- Source code available at:
<https://github.com/Robyroc/Legio>
- It requires ULFM
 - Present in the latest versions of OpenMPI
- For any issue, feel free to mail me:
roberto.rocco@polimi.it



Next steps

- Evaluate the use of **MPI Sessions** to handle faults instead of ULFM
 - Upon failure, get rid of the Session and recreate it
- Measure the fault impact on **energy consumptions**
 - Also the impact of countermeasures like Legio
- Extend the range of **MPI functionalities supported**
 - Topologies
 - Persistent communication
 - ...
- And much more!

Thank you for your attention.

We also acknowledge EuroHPC JU for awarding this project access to the Karolina CPU partition.

A wireframe dinosaur, resembling a T-Rex, is positioned in the center-left of the image. The background is a server room with blue lighting and server racks. The dinosaur is composed of a network of white lines forming its shape. The overall scene is a digital or futuristic environment.

Porting of Tinker-HP to AMD GPU based supercomputers

Project: *"Porting code Tinker-HP"*

EuroHPC used: Lumi-G

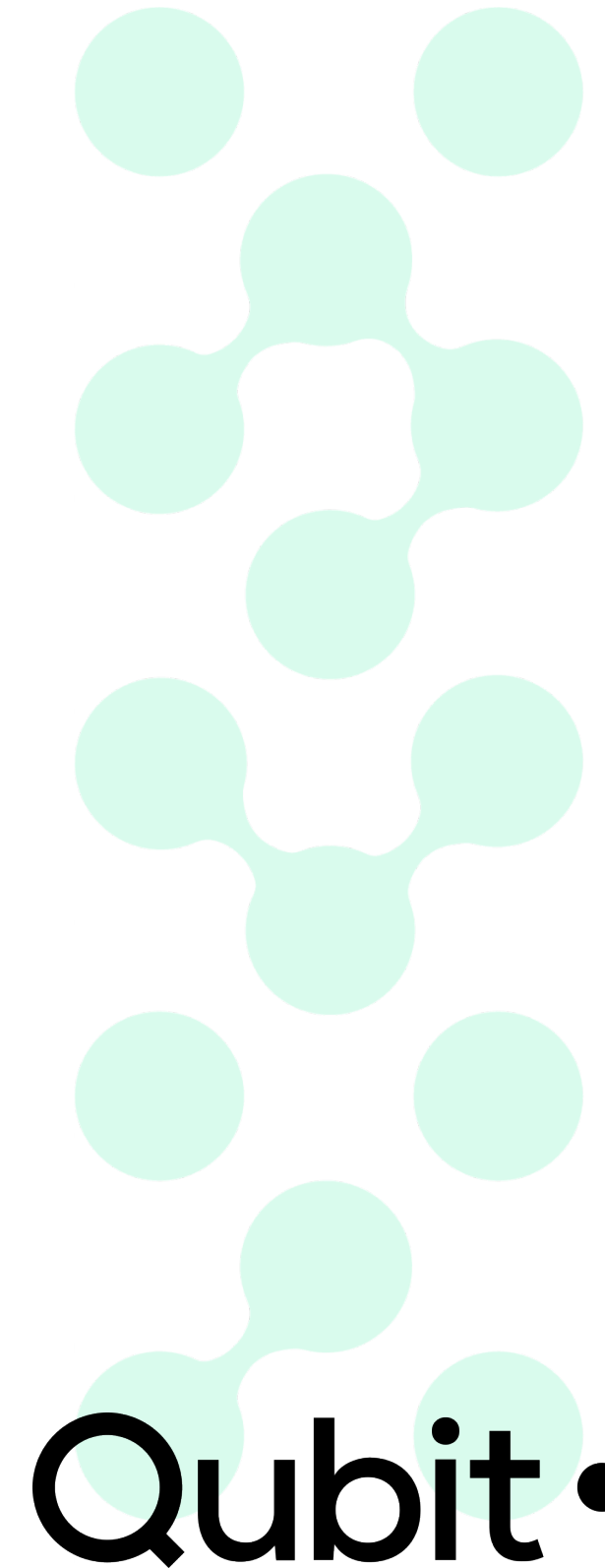
Speaker: Umesh *SETH (Qubit Pharmaceuticals)*

**EUROHPC
USER DAY
2023** Brussels
11.12.23



Contents

- Tinker-HP & DeepHP
- What we do at Qubit Pharmaceuticals ?
- Features to Port
- Work accomplished (until now)
- Some specific portings
 - Port of CUDA Fortran Kernels to HIP C++
 - Port of python part (DeepHP) etc.
- Concluding remarks



Tinker-HP

Tinker-HP is a state of the art software package dedicated to molecular dynamics simulations and to hybrid QM/MM. Massively parallel implementation on CPUs and GPUs

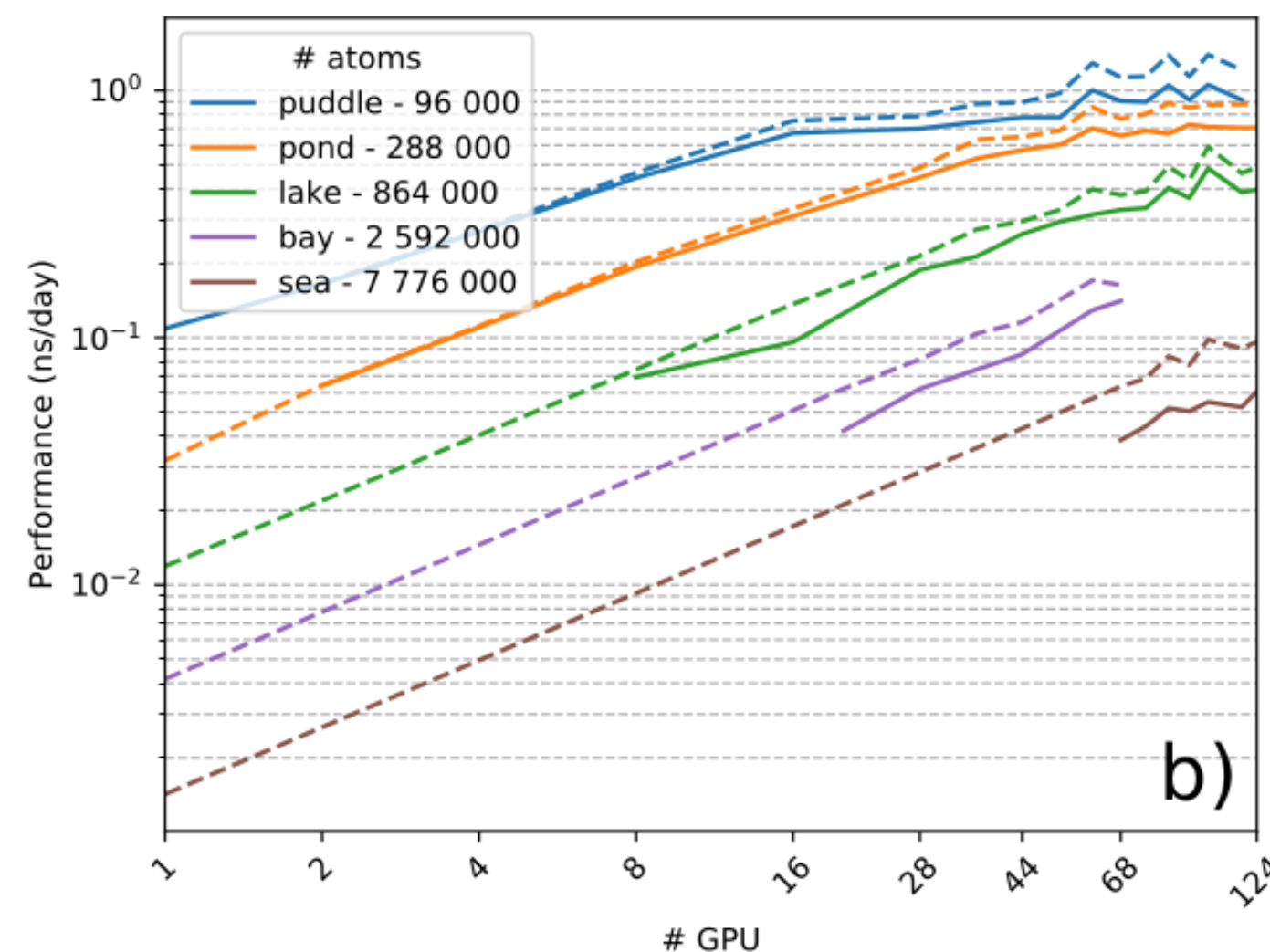
- Advanced electrostatic interaction models (AMOEBA force field etc.)
- Several numerical methods (PCG, PME, Verlet integration etc.)
- Parallel simulations of millions to billions of particles systems

Deep-HP

- Extension of Tinker-HP
- A deep learning platform for polarizable molecular potentials
- Deep learning coupled with Force Fields for biological simulations

[Relevant publications for details :](#)

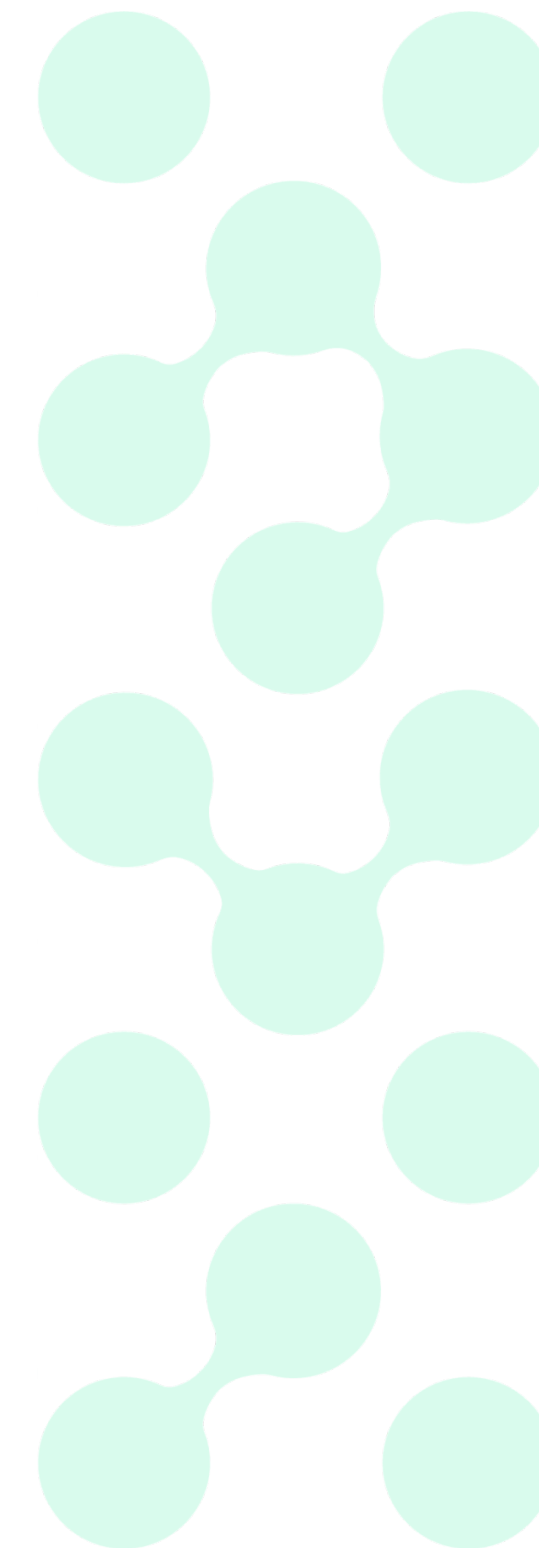
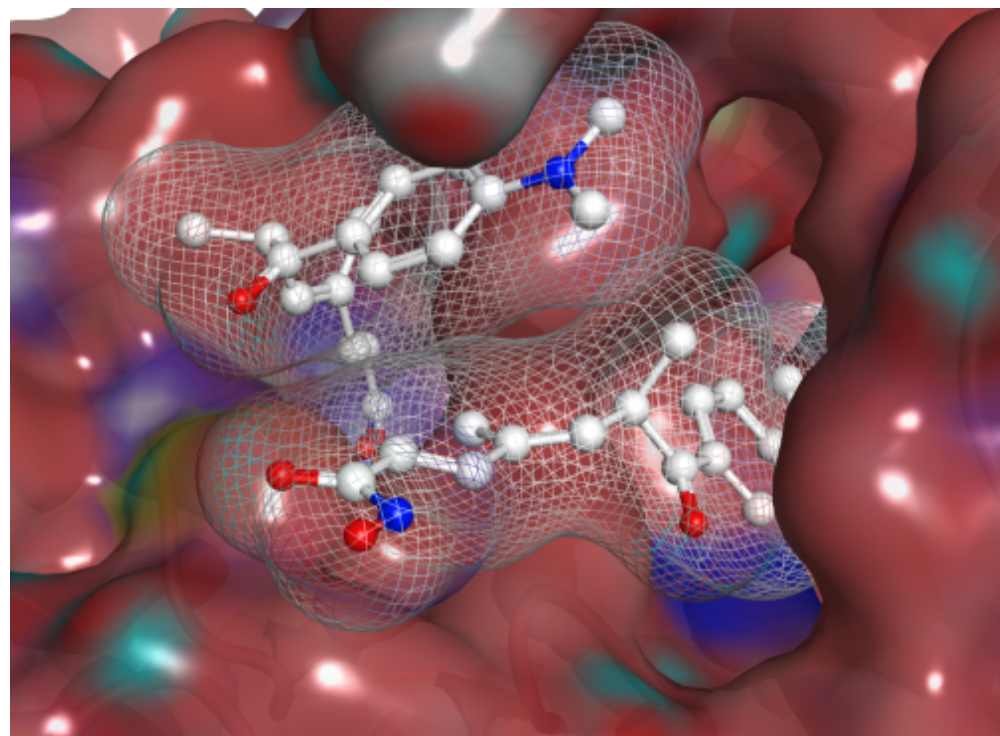
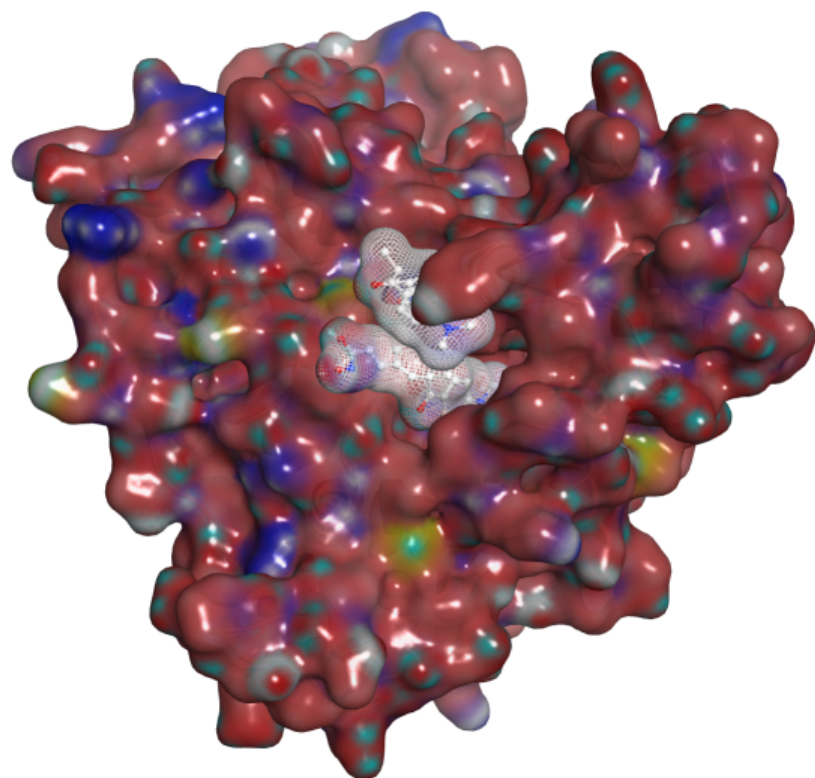
- <https://pubs.acs.org/doi/10.1021/acs.jpcllett.2c00936>
- <https://pubs.acs.org/doi/10.1021/acs.jctc.0c01164>



What we do at Qubit Pharmaceuticals ?

- Drug design platform, **Atlas**,
- To discover, optimize & validate drug candidates
- Complex computations at scale,
- Tinker-HP : MD simulations in this drug discovery pipeline.

10s of thousands of GPU hours of MD simulations every month with Tinker-HP !



Tinker-HP features to port

Nvidia GPUs	AMD GPUs
OpenACC	OpenMP
CUDA Fortran & CUDA C++	HIP C++
Python (PyCuda etc.)	Python (CUPY etc.)
CUDA libs (cufft,curand,Thrust)	ROCm libs (hipfft,hiprand,rocThrust)
NVSHMEM	ROCSHMEM
Nvidia compilers (nvfortran, nvcc,pgi)	Cray (cce) + AMD (hipfc,hipcc)

Some key points of Nvidia version :

- 500+ source files (*.f,*.cu,*.h) → **mostly Fortran source code**
- 200k + codes lines
- 80+ CUDA kernels + device fun.
- 1100+ OpenACC GPU kernels

Why porting was absolutely necessary ?

given that Cray compiler can compile Fortran+OpenACC code !

- **complex mix of OpenACC and CUDA code**



Not a feature of the code itself but compiler environment needs porting

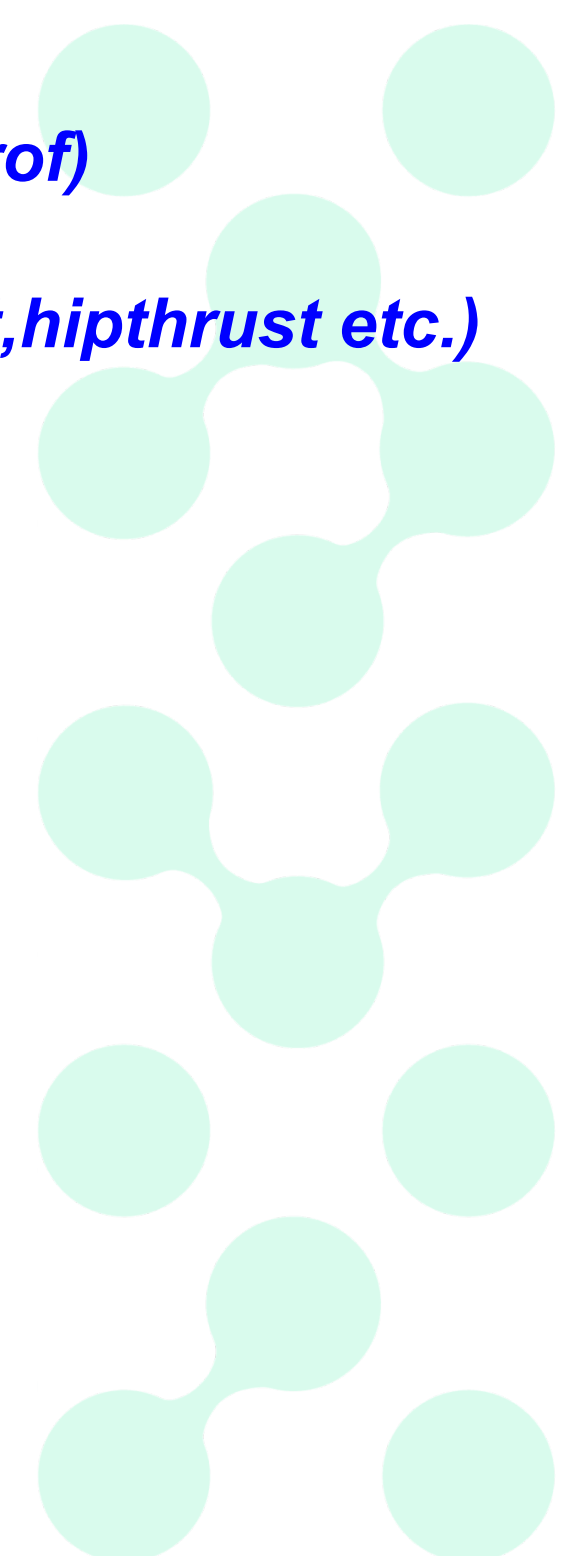


Work accomplished until now

- Potential energy subroutines
- Atom-Atom pair lists generation
 - Fortran subroutines (ACC to OMP)
 - CUDA Fortran Kernels to HIP C++
- Memory management routines (allocation, memset, initialization etc.)
- Python side of DeepHP
 - PyCUDA to CUPY + DLpack

~80 gpu kernels (info rocprof)

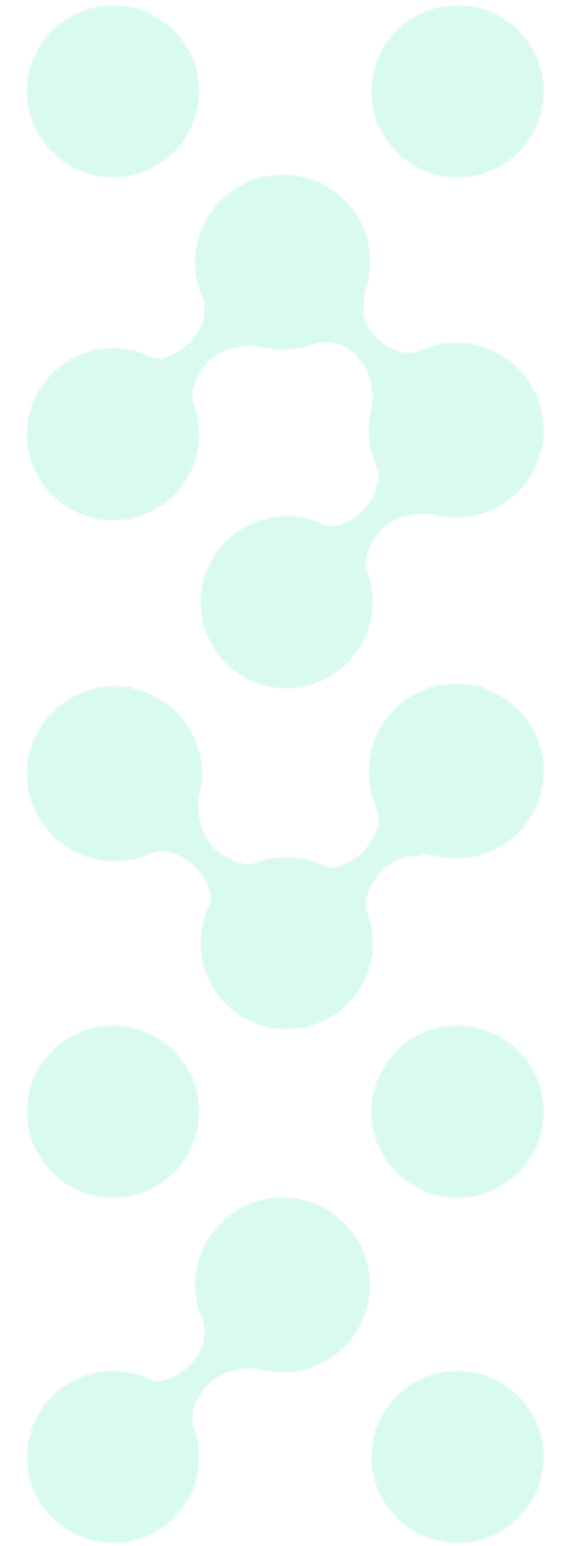
- *OMP kernels*
- *GPU Library calls (hipfft,hipthrust etc.)*
- *2 HIP C++ kernel*



WIP

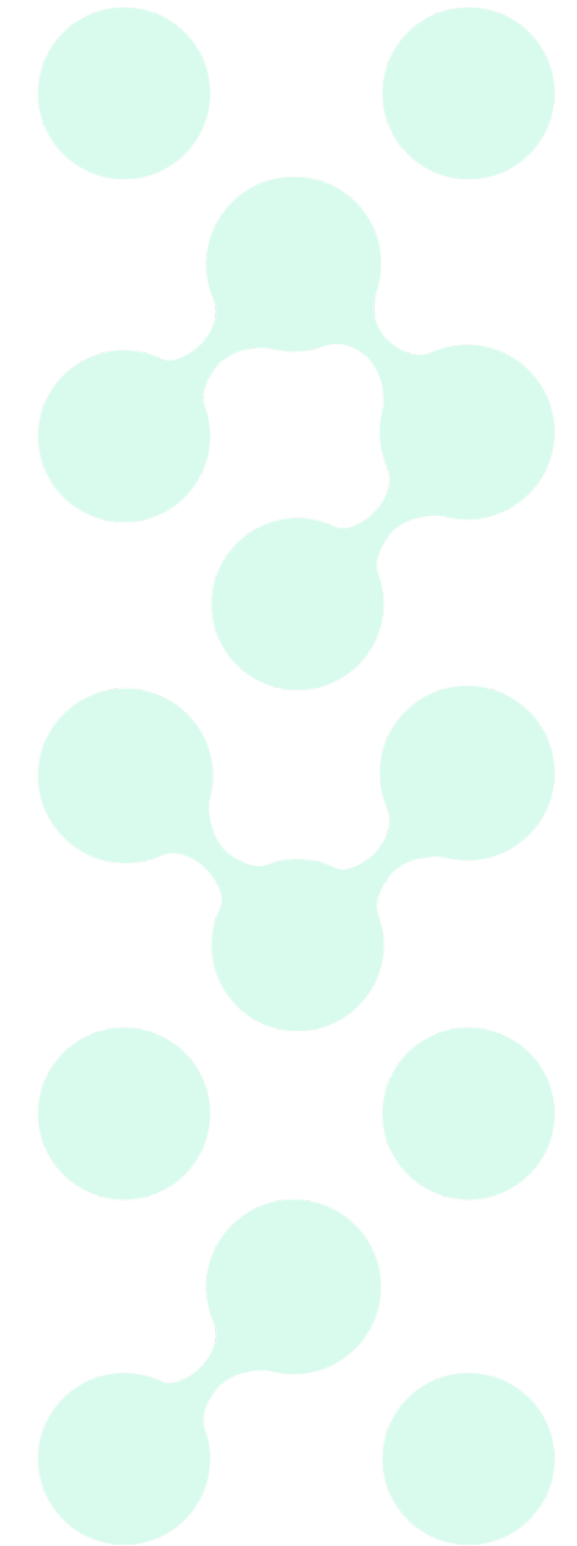
- ~ 240 OMP kernels
- ~ 210 OMP memory update
- ~ 120 OMP enter data
- ~ 60 OMP use_device_addr
- ~ 35 OMP reductions

Discussion on some specific porting challenges we faced !



Porting CUDA Fortran to HIP

- **CUDA Fortran features**
 - An **extension to standard Fortran** : device, pinned, attribute, kernel loop directives, memory allocation
 - Even the host side Fortran code has to be ported to standard Fortran
- **Indexing** in Fortran (1:x) and HIP(0:x-1)
 - Some standard variables : threadIdx, warpID, laneID [index starts @ 1]
 - Several code specific Fortran arrays (used C++ indexing)
- **Tool GPUFORT**
 - Project to port CUDA Fortran to HIP , and OpenACC to OpenMP
 - Project was discontinued
- **HIPFort** module and '**hipfc**' compiler wrapper installation



CUDA Fortran Vs HIP C++ kernel

```

attributes(global)
subroutine filter_pairwise_atom_lst(FILTER_PARAMS)
  implicit none

  -----Usual Variable declaration-----

  ithread = threadIdx%x + (blockIdx%x-1)*blockDim%x
  iwarp   = (ithread-1) / WARP_SIZE
  nwarp   = (blockDim%x*gridDim%x) / WARP_SIZE
  ilane   = iand( threadIdx%x-1,WARP_SIZE-1 ) + 1
  pair_a  = nb_pair*(BLOCK_SIZE**2)

  do ii = iwarp,nb_pair-1,nwarp
    iblock= blst(2*ii+1)
    idx   = (iblock-1)*WARP_SIZE + ilane
    iscan = ii*BLOCK_SIZE**2
    xi    = x[idx]
    yi    = y[idx]
    zi    = z[idx]
    do j = 1, WARP_SIZE
      srclane = j
      kdx_    = kdx - ilane + j
      xk_     = __shfl(xk,srclane)
      yk_     = __shfl(yk,srclane)
      zk_     = __shfl(zk,srclane)
      if ((xpos**2+ypos**2+zpos**2.lt.cutbuff2) then
        lst(      iscan+(j-1)*WARP_SIZE+ilane) = idx-1
        lst(pair_a+iscan+(j-1)*WARP_SIZE+ilane) = kdx_-1
      end if
    end do
  end do
end subroutine

```

```

__global__ void hip_filter_pairwise_atom_lst(PAIRWISE_PARAMS)
{
  const int ithread = threadIdx.x + blockIdx.x*blockDim.x;
  const int iwarp   = ithread / WARP_SIZE;
  const int nwarp   = (blockDim.x * gridDim.x) / WARP_SIZE;
  const int ilane   = threadIdx.x & (WARP_SIZE-1);
  const int pair_a  = nb_pair*(BLOCK_SIZE*BLOCK_SIZE);

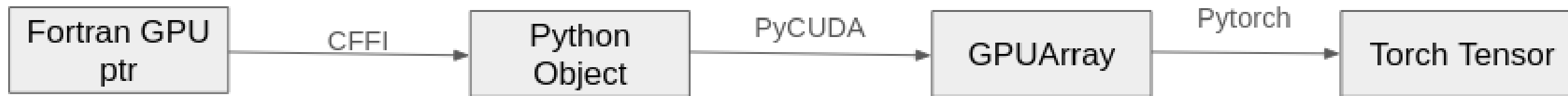
  for(int ii=iwarp; ii<nb_pair; ii+=nwarp) {
    int iblock = blst[2*ii];
    int idx    = (iblock-1)*WARP_SIZE + ilane;
    int iscan  = ii*BLOCK_SIZE*BLOCK_SIZE;
    real xi = x[idx];
    real yi = y[idx];
    real zi = z[idx];
    for(int j=0;j<WARP_SIZE;j++) {
      int srclane = j;
      int kdx_    = kdx - ilane + j;
      real xk_    = __shfl(xk,srclane,WARP_SIZE);
      real yk_    = __shfl(yk,srclane,WARP_SIZE);
      real zk_    = __shfl(zk,srclane,WARP_SIZE);
      if((xpos*xpos+ypos*ypos+zpos*zpos < cutbuff2) {
        lst[iscan+j*WARP_SIZE+ilane] = idx;
        lst[pair_a+iscan+j*WARP_SIZE+ilane] = kdx_;
      }
    }
  }
}

```

Porting of Python part (DeepHP)

- Fortran, C and Python interface (iso_c_binding and CFFI package)
 - CFFI : C Foreign Function Interface for Python

Pointer transfer pipeline for Nvidia

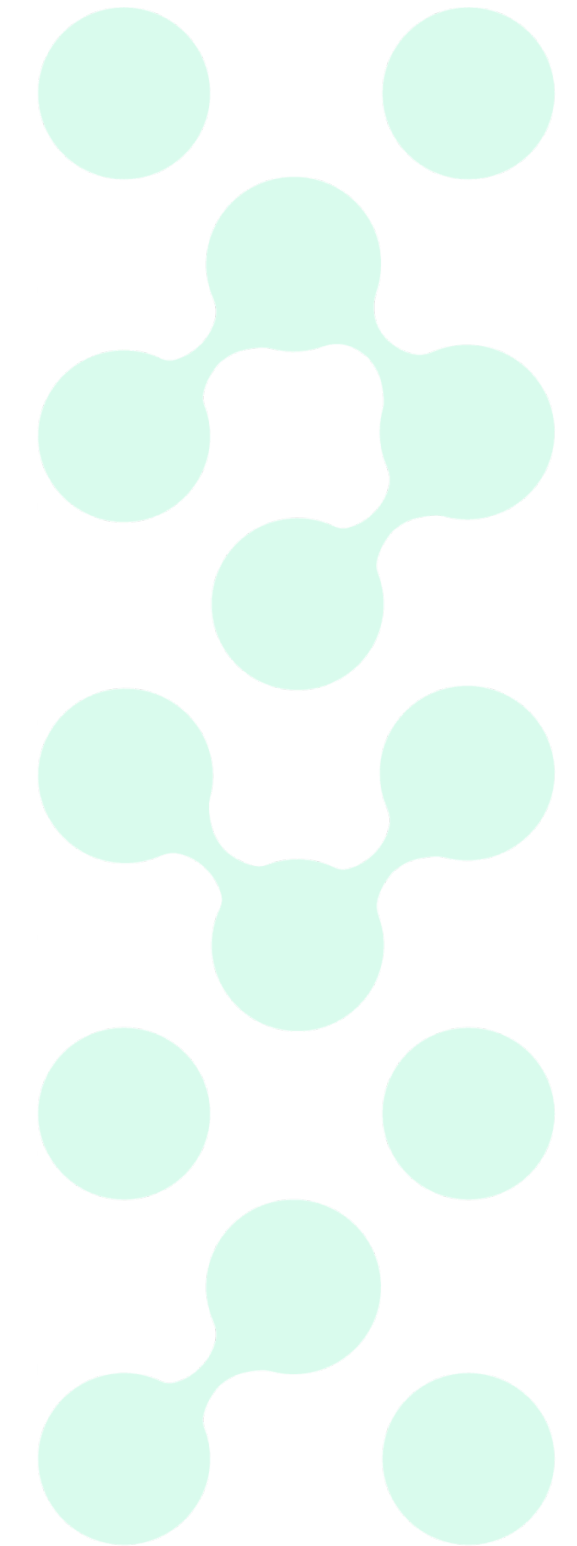


Pointer transfer pipeline for AMD



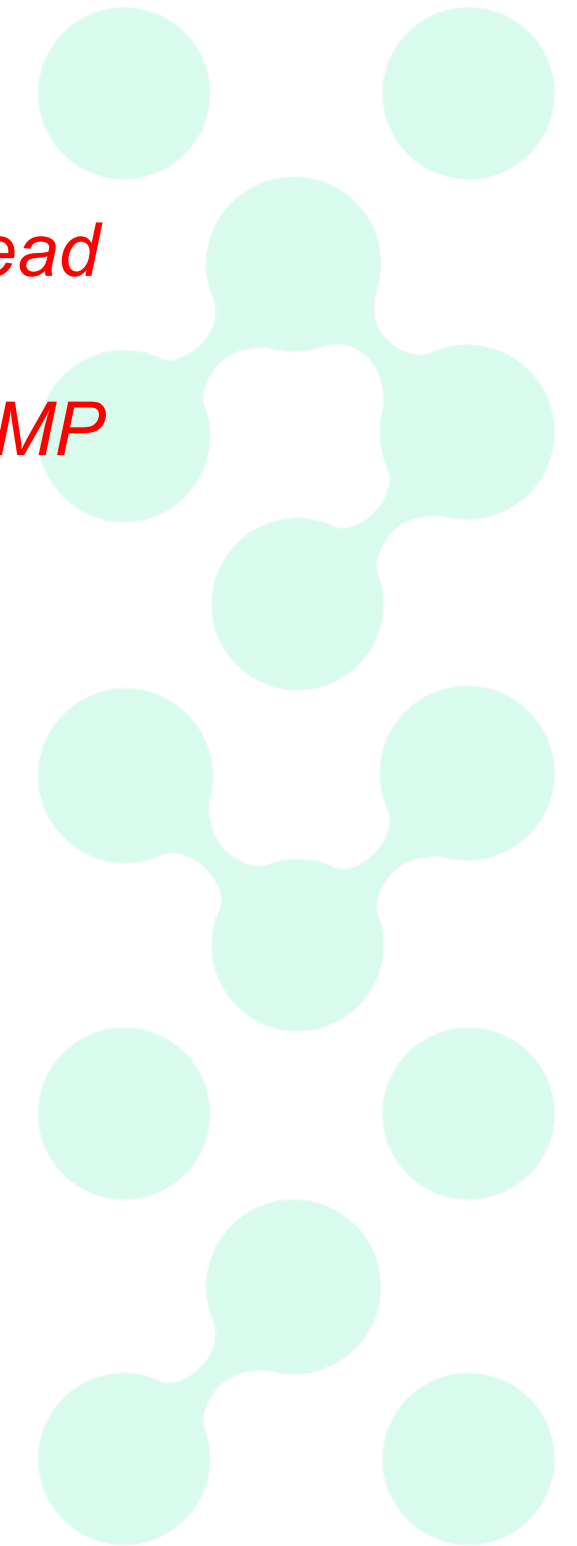
```

    Import cupy as cp
    mem = cp.cuda.UnownedMemory(ptr, size_bytes, owner=ptr, device_id=0)
    memptr = cp.cuda.MemoryPointer(mem, offset=0)
    cp_array = cp.ndarray(shape, dtype=_ctype2dtype[ctype], memptr=memptr)
    ga_dl = cp_array.toDlpack()
    gTensor = torch.utils.dlpack.from_dlpack(ga_dl)
    
```



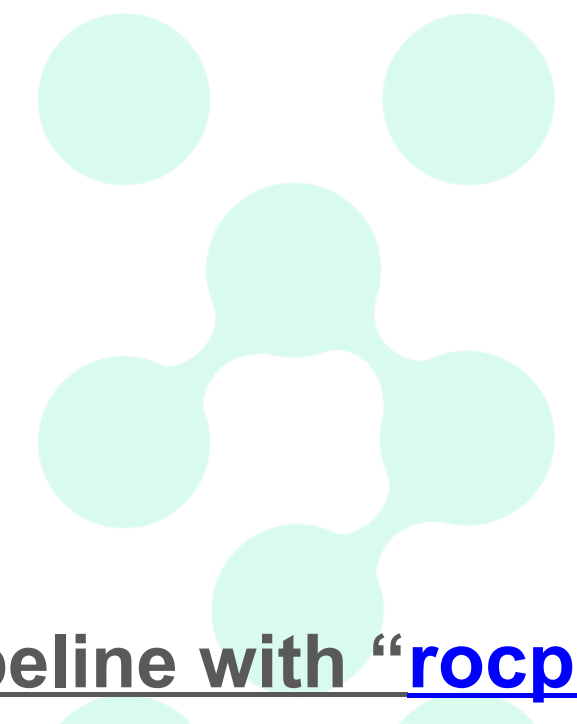
Pytorch's OMP issue with Tinker-HP

- Call to “`torch.autograd.grad()`”
 - error: *CCE OpenMP fatal error: omp_in_parallel attempted from non-OpenMP thread*
- Debug test : call to “`torch.set_num_threads(1)`”
 - error: *CCE OpenMP fatal error: omp_set_num_threads attempted from non-OpenMP thread*
- Possible reason:
 - incompatibility between two OMP libs present at runtime (Cray + GCC)
- Possible solutions:
 - compile pyTorch with CCE
 - compile pyTorch without OMP support with GCC
 - compile Tinker-HP with GCC
 - Introduce OMP threads in Python part (Cython etc.)

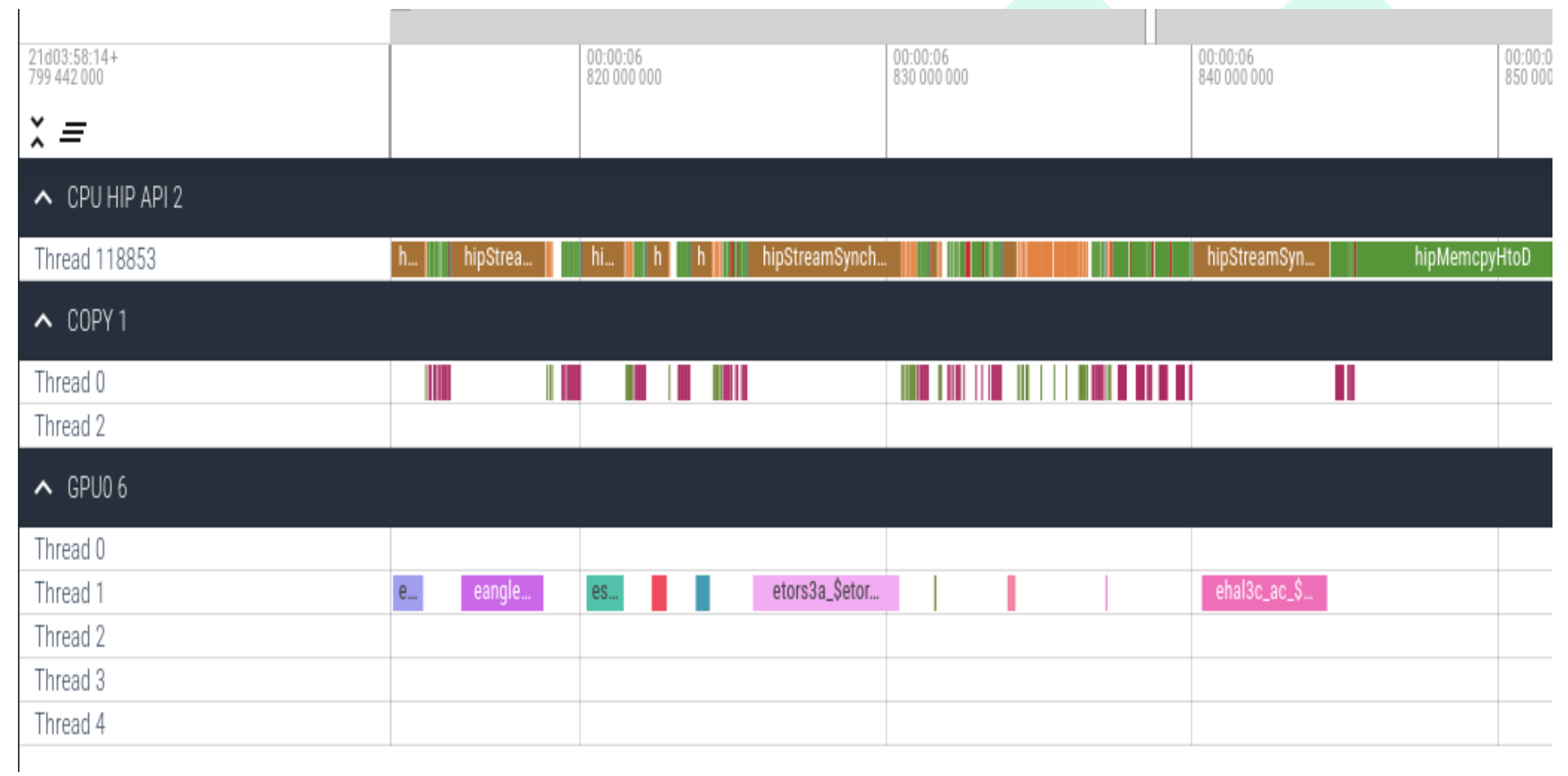


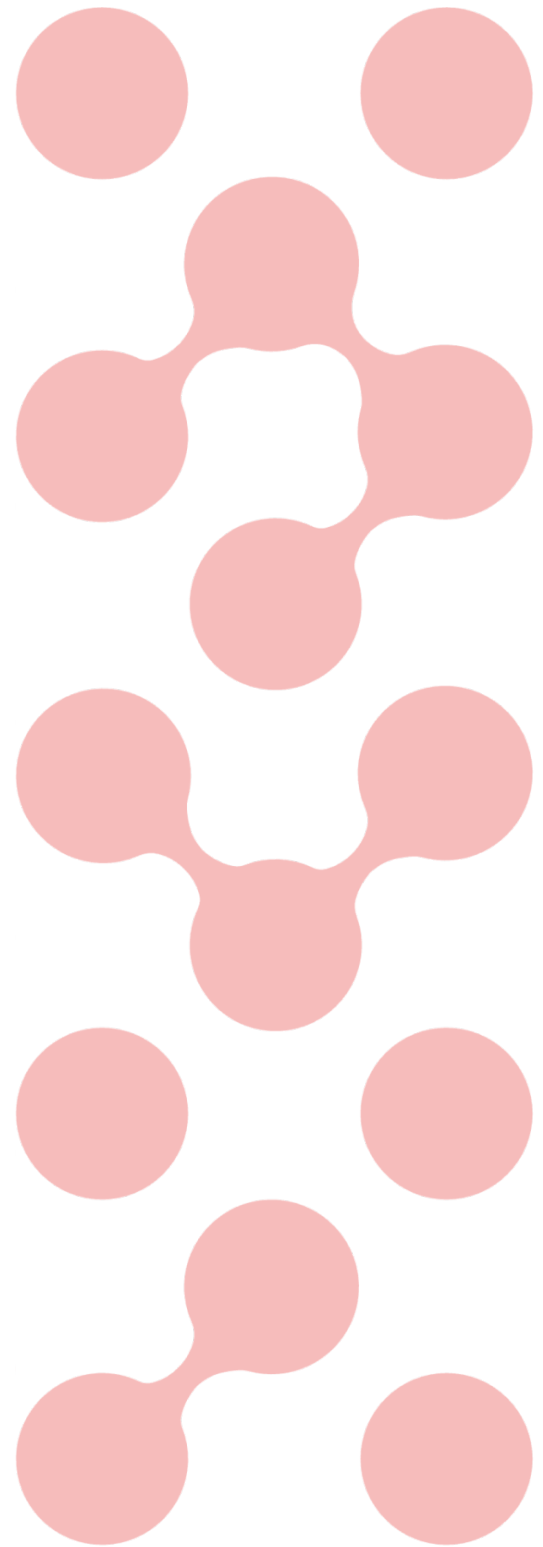
Concluding remarks

- Porting of a complicated scientific code is time taking
 - Difficult to follow a timeline
 - Rigorous project
- External factors that impacted this project
 - FS issues
 - Software issues
 - CCE (internal errors, OMP pragmas etc.)
 - lenient Nvidia compilers
 - Slurm CPU+GPU binding
 - Evolving ROCm/HIP environment
- Managing the software environment is not always easy
- WIP
 - Port rest of the code
 - Optimization • •



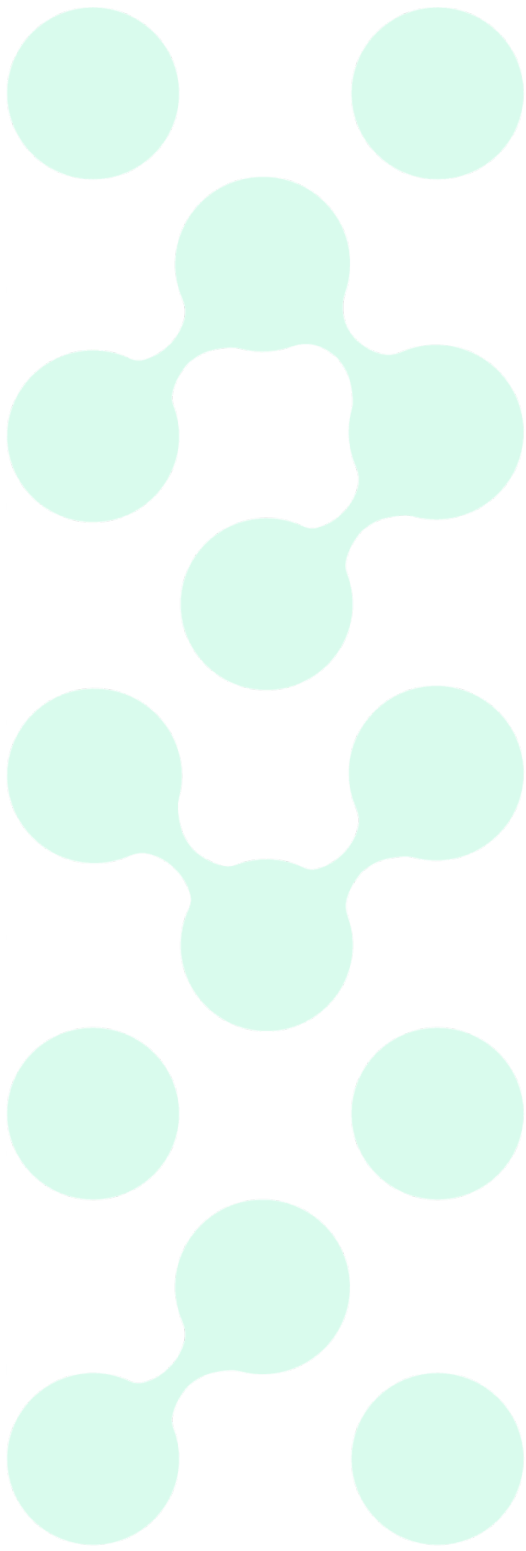
GPU execution pipeline with “rocprof”





Thanks

- EuroHPC : CSC (Supercomputer LUMI)
- CINES-GENCI (Supercomputer Adastr)
- To Prof. Jean-Philip Piquemal, Louis Lagardere & Olivier Adjoua of Sorbonne University, Paris.



OpenACC to OpenMP

- Explicit mention of reduction clause
- Explicit mention of mapping for serial OMP kernels on GPU
- CCE compiler only warns at some wrong OMP pragmas, where it should send errors
- Some kernels work only for a certain number of teams & threads
- Issue with collapse of 3 loops with omp
- Explicit handling of CUDA streams in ACC, feature not present in OMP

Scalar initialization on GPU

!\$omp target update from(A) → good

!\$omp **ta**get update from(A) → **no compiler error !**

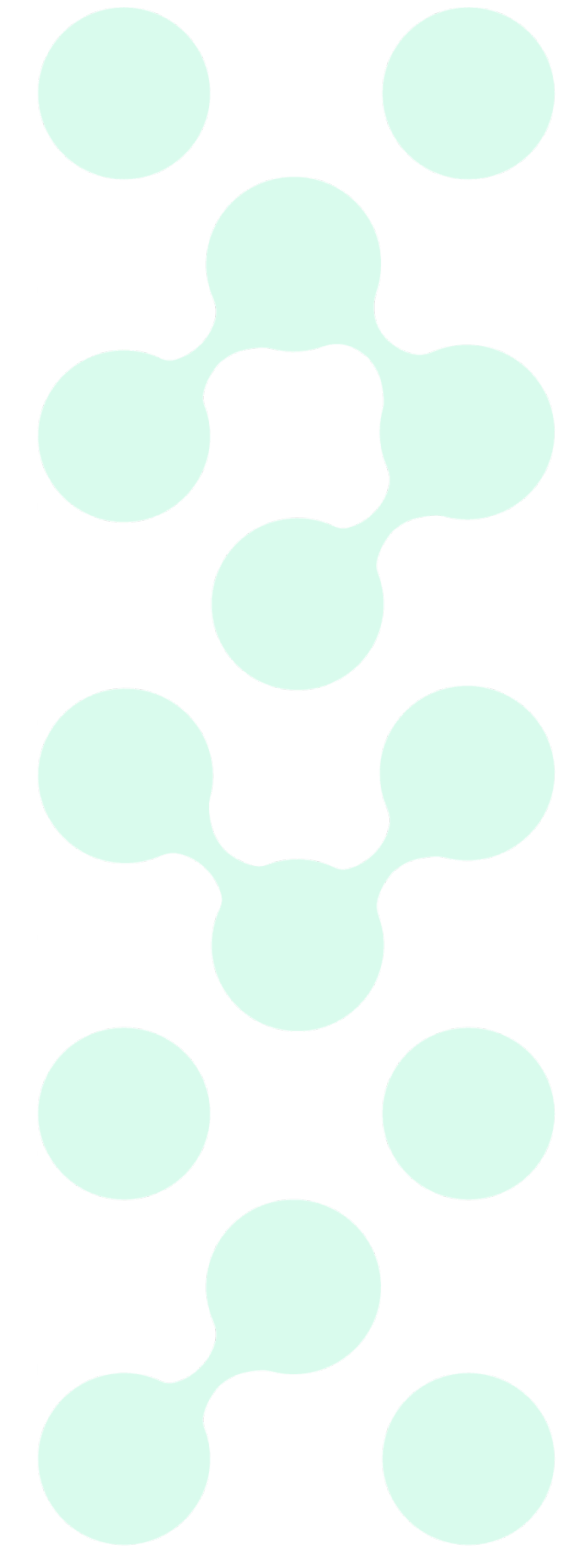
!\$omp target **map(A)**

A = 0.0d0

!\$omp end target

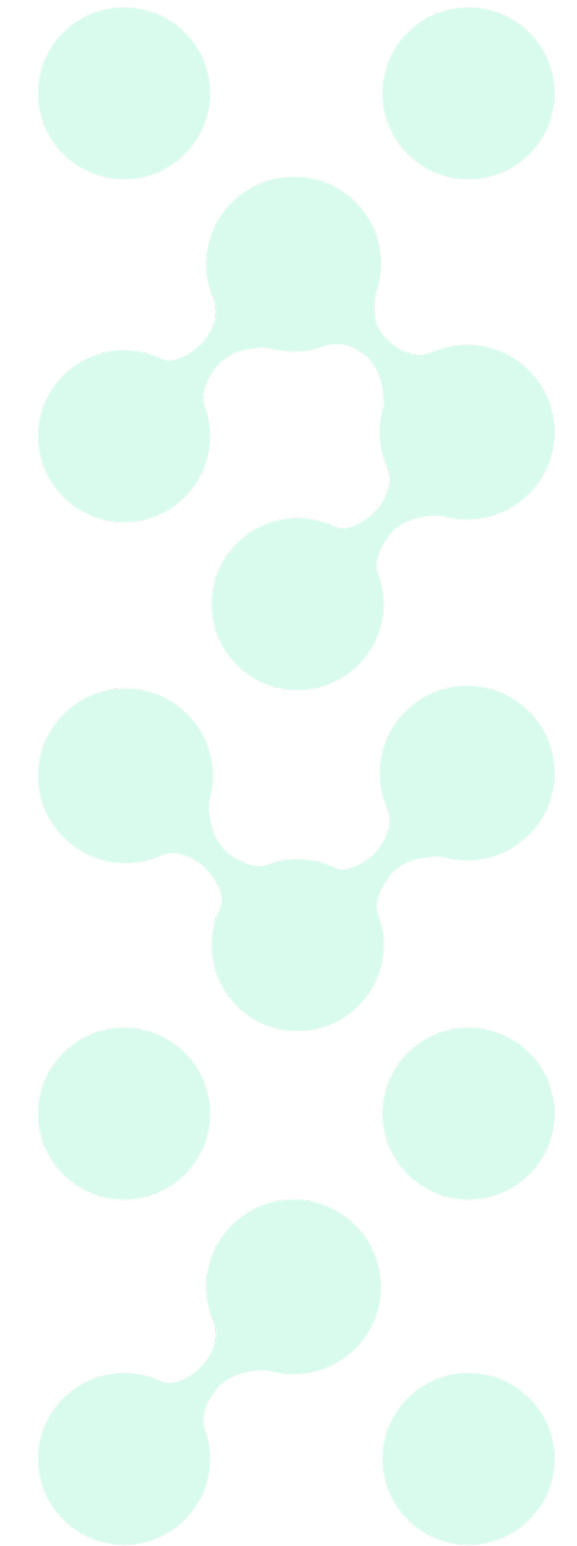
ftn-790 ftn: WARNING EMPOLE3CGPU, File = empole3gpu.f, Line = 117, Column = 7

Unknown or unsupported compiler directive or syntax error.

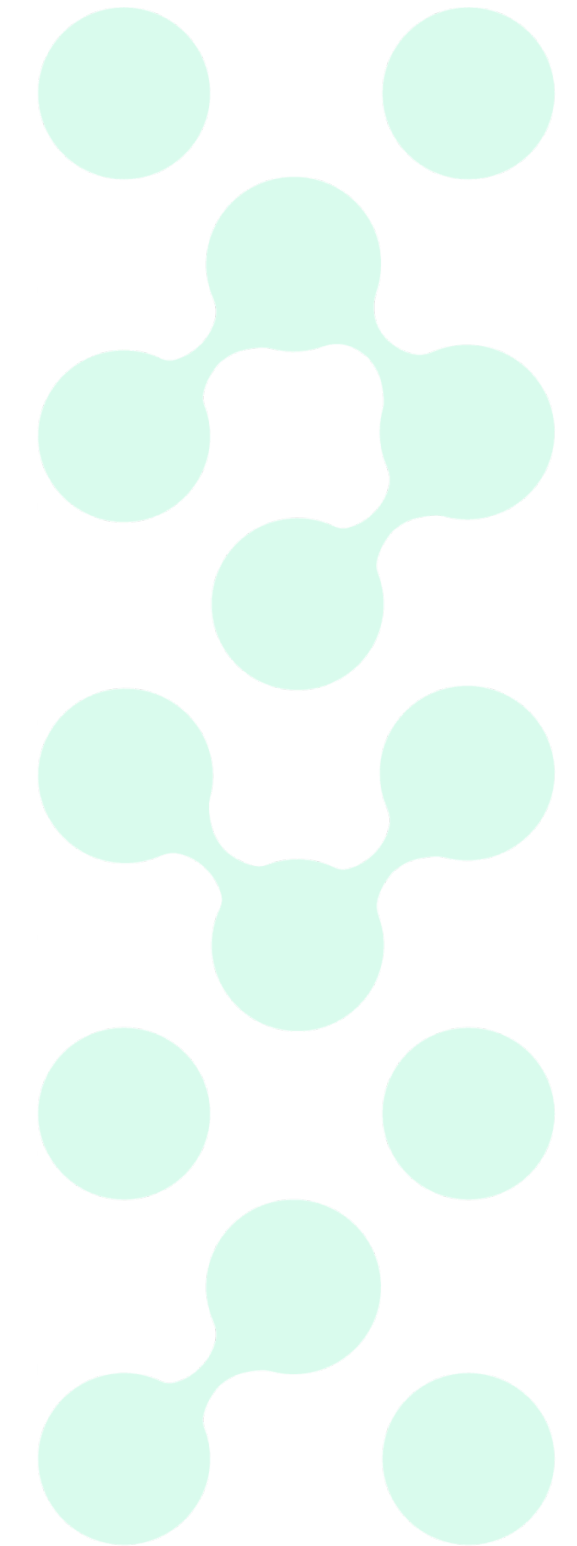
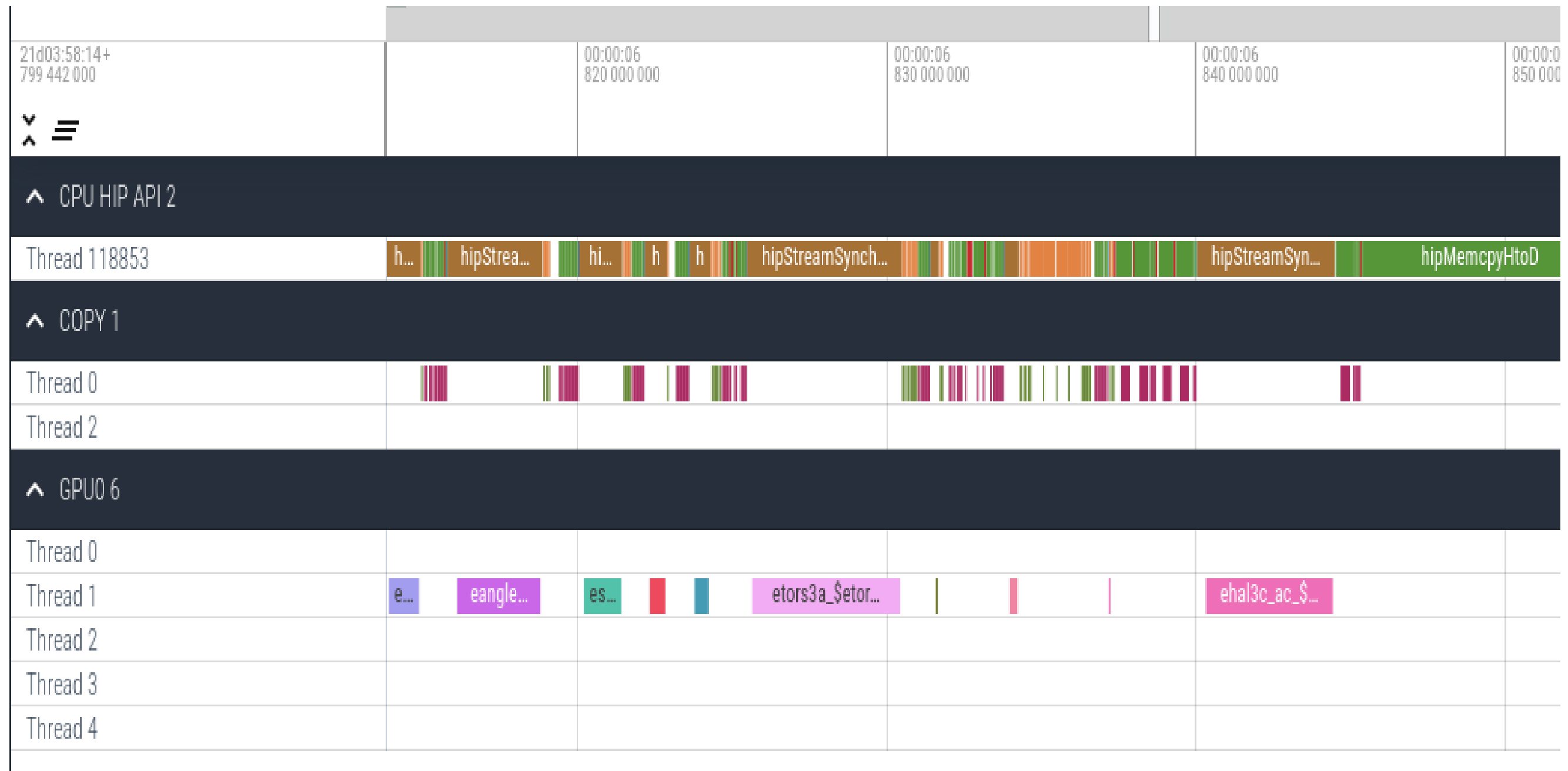


Porting Compiler environment

- Nvidia compilers (nvfortran,pgi,nvcc) to Cray + AMD compilers (cce,hipfc,hipcc)
- ~ 130 source files modified to get a working CPU version with CCE
- Porting of Makefiles
- Impact of compiler flags (-hipa0 etc.)
- Lenient Nvidia compilers Vs CCE
 - Explicit interfaces for subroutines
 - Module arrays
 - Scope (local scope variables are visible in other subroutines)
 - argument passing (module variables passed via subroutines arguments)



GPU execution pipeline with “[rocprof](#)”



A wireframe bear is depicted in a server room setting. The bear is composed of a network of blue lines forming a mesh, and it is positioned in the center-left of the frame. The background features a perspective view of server racks with glowing blue light strips, and scattered binary code (0s and 1s) in the air.

Workload Estimation and Load-Balancing of Discrete Element Method

**EUROHPC
USER DAY
2023** Brussels
11.12.23



Project: Workload Estimation and Load-Balancing of Discrete Element Method
EuroHPC used: MeluXina

Speaker: Xavier BESSERON (*Uni. of*

Workload Estimation and Load-Balancing of Discrete Element Method

Outline

Extended Discrete Element Method

- What is XDEM?

Parallelization of XDEM

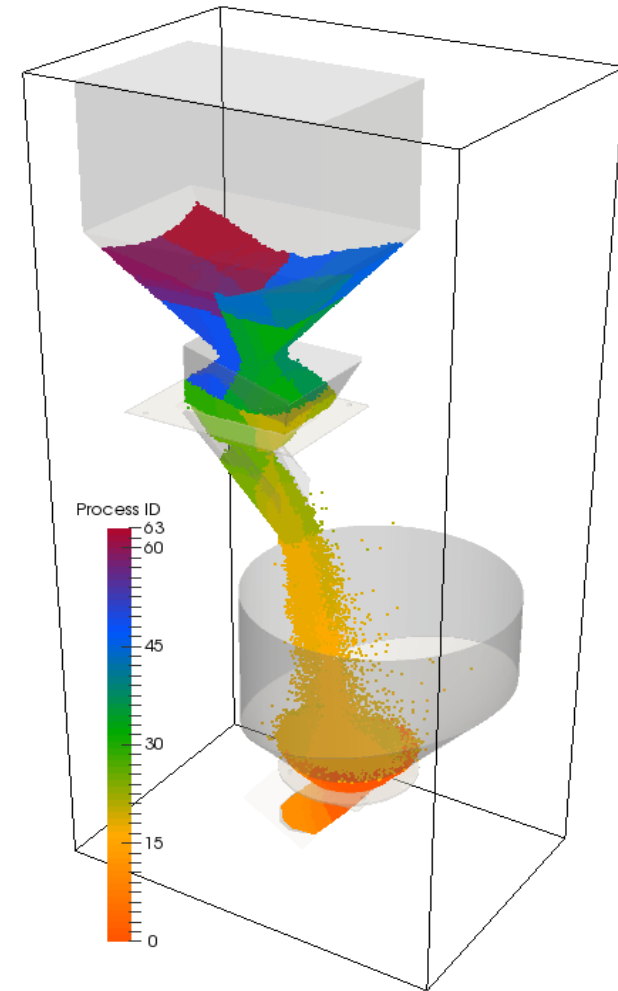
- Domain Decomposition with MPI
- Load-Balancing

Workload Estimation for XDEM

- Toward better Load-Balancing

Preliminary Results

- Load Estimation and Imbalance



Extended Discrete Element Method

What is XDEM?

eXtended Discrete Element Method

What is XDEM?

Simulation software for

Particles Dynamics

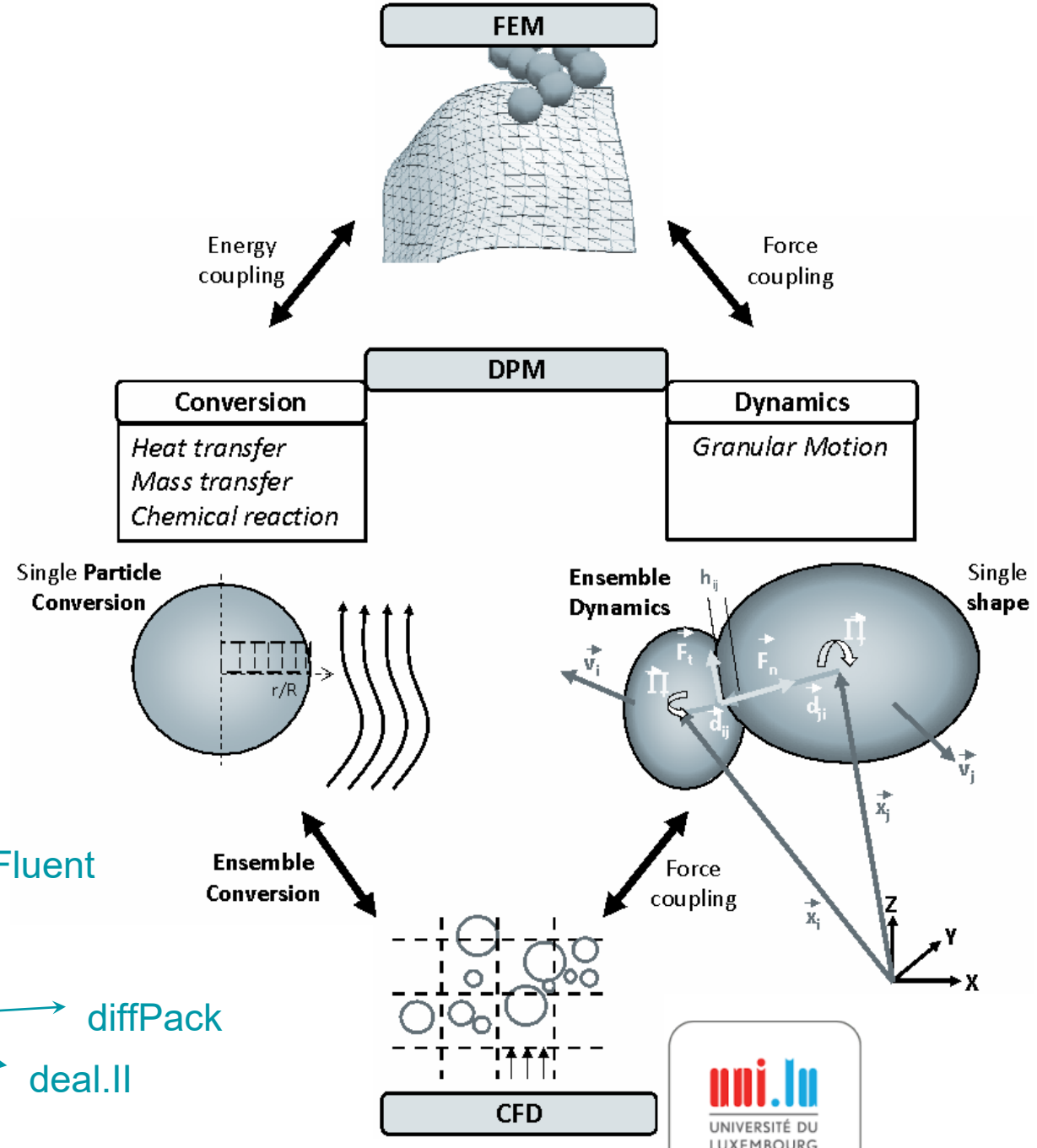
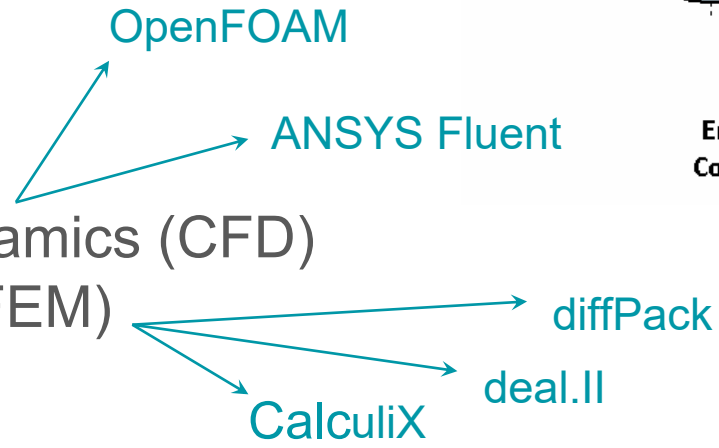
- Force and torques
- Particle motion

Particles Conversion

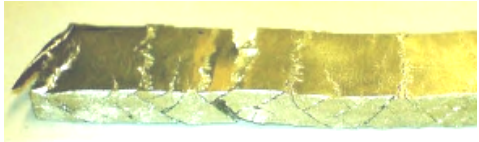
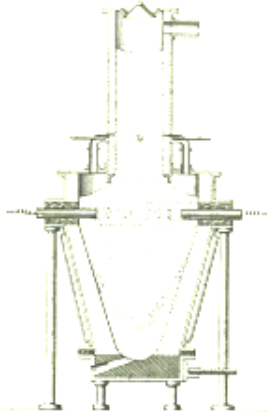
- Heat and mass transfer
- Chemical reactions

Coupled with

- Computational Fluid Dynamics (CFD)
- Finite Element Method (FEM)

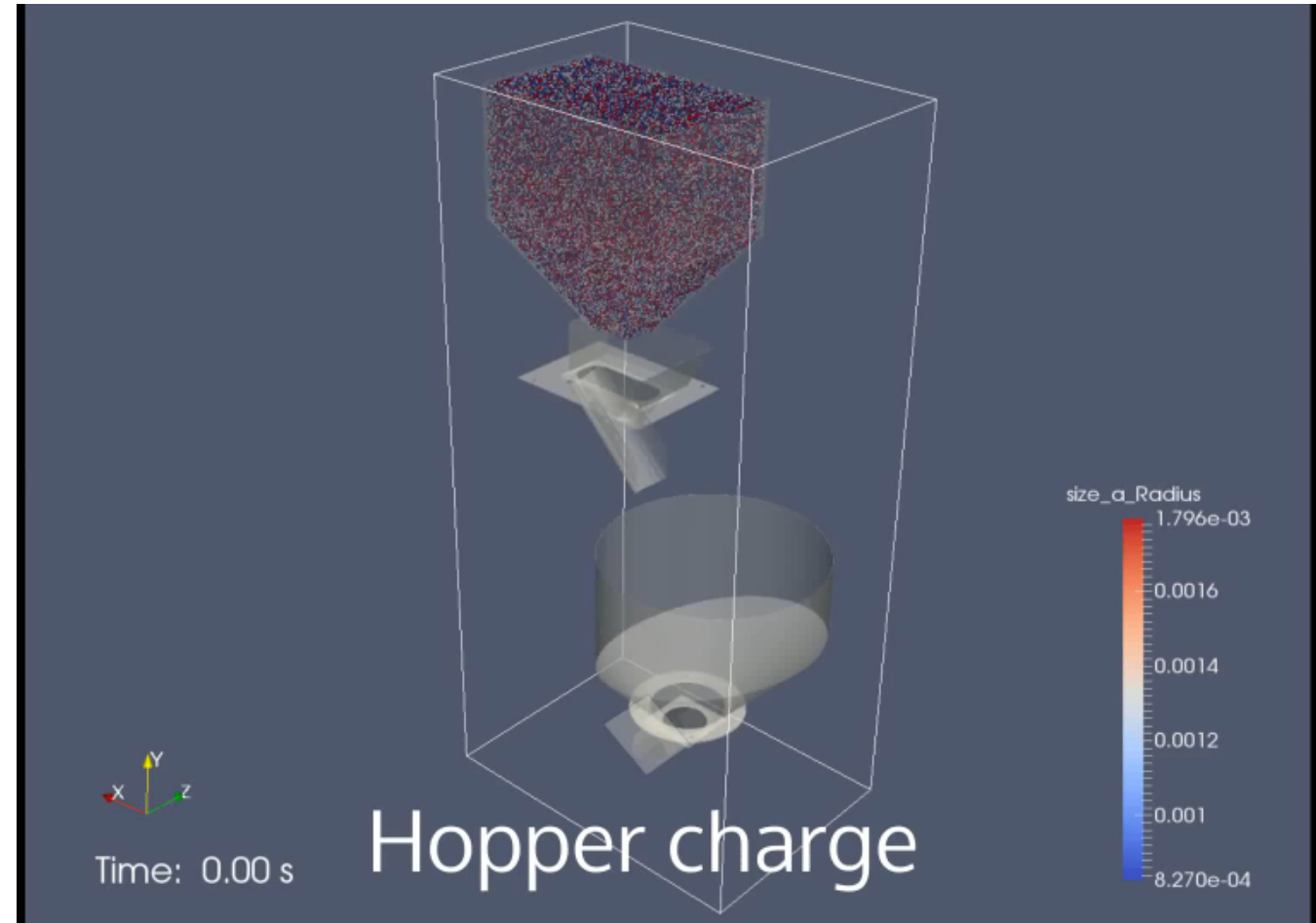
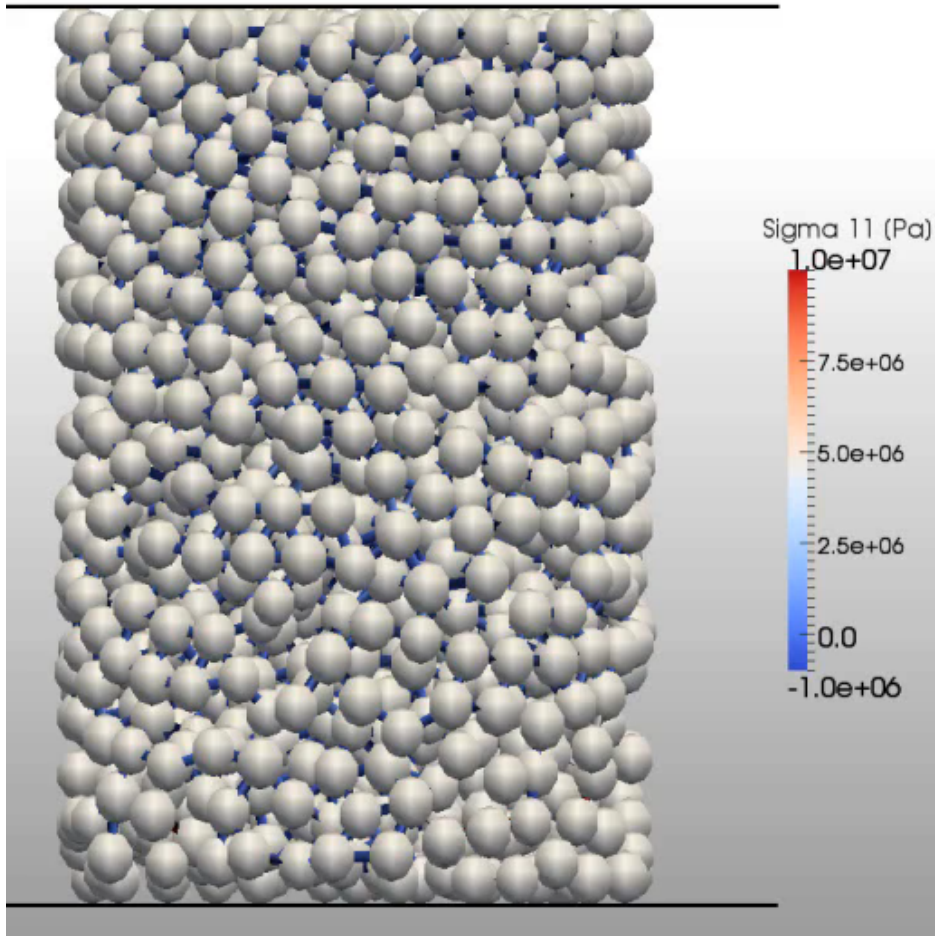


Application Examples: XDEM

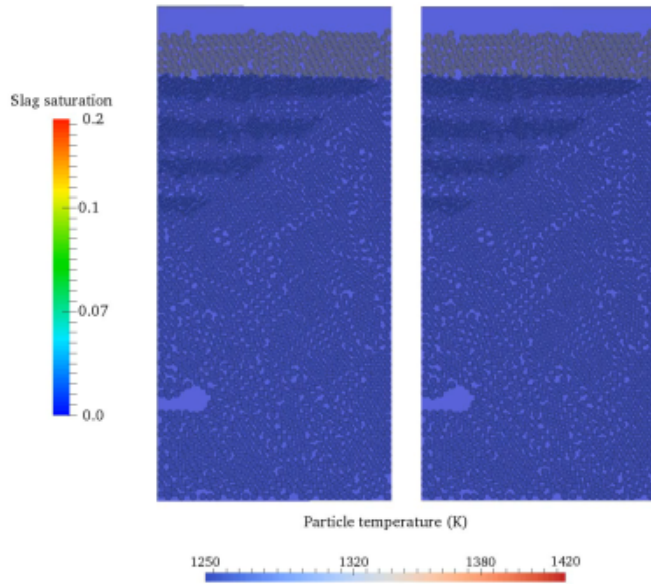


Brittle Failure

Hopper charge and discharge



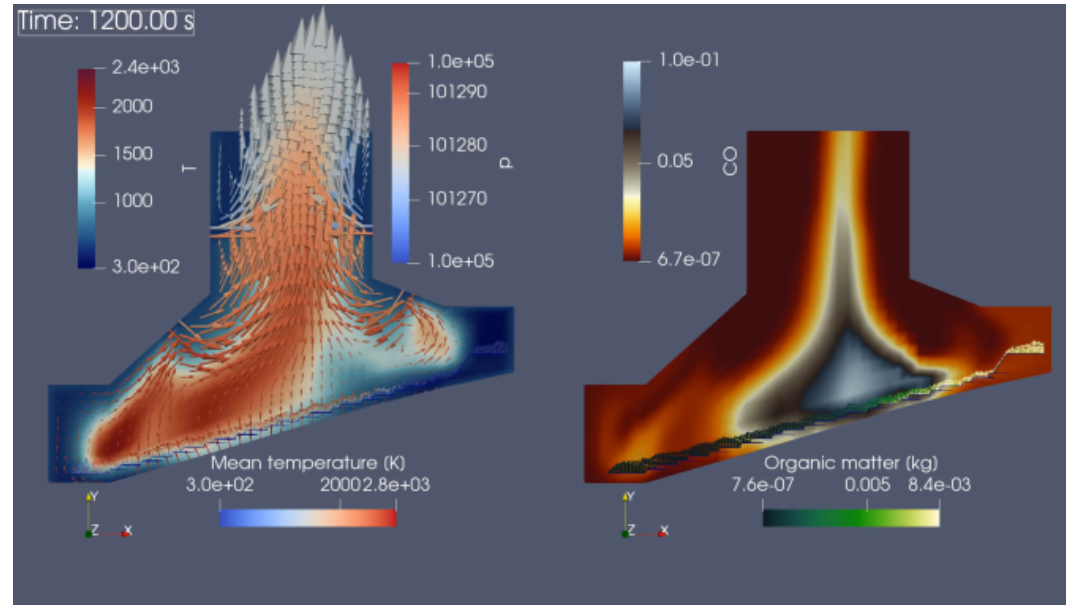
Application Examples: XDEM coupled with CFD



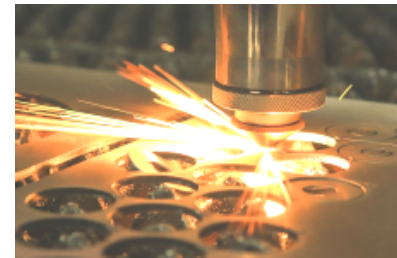
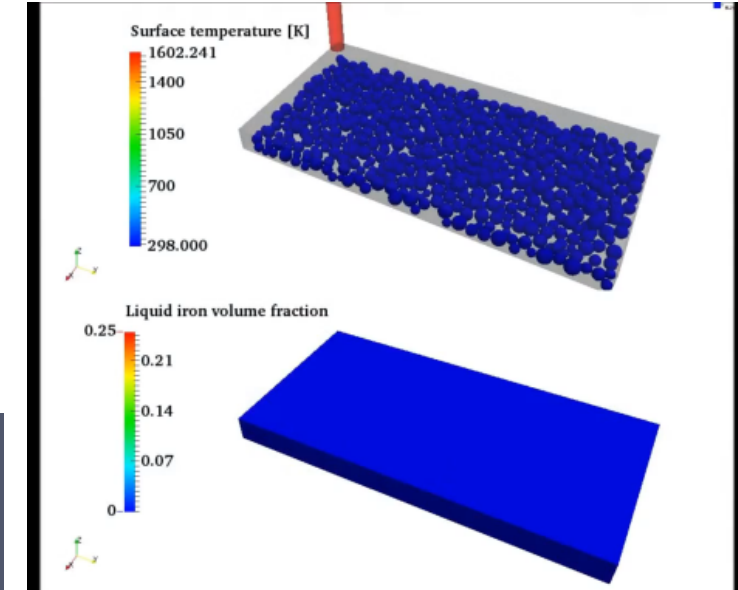
Iron & Slag production in a Blast Furnace



Wood Conversion in a Biomass Furnace



Selective Laser Melting in Additive Manufacturing



Parallelization of XDEM

Domain Decomposition with MPI
and Load-Balancing

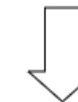
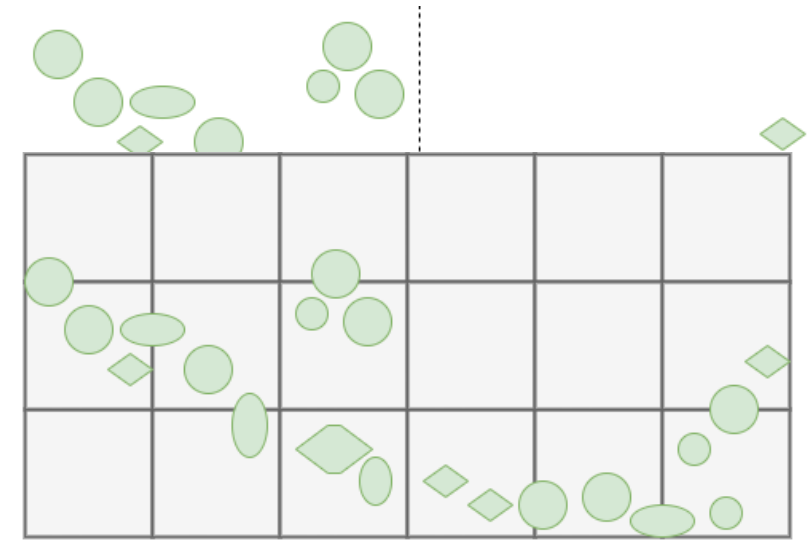
Domain Decomposition in XDEM

Decomposing the set of particles?

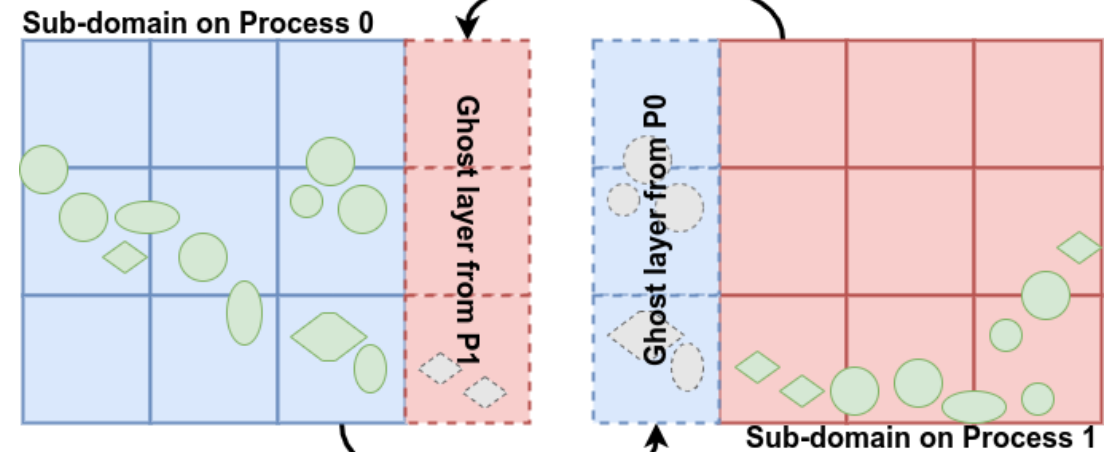
- Particles move during the simulation
 - Neighborhood relations change
 - Create undetected dependencies
- Would require frequent re-partitioning

Use a static regular grid to 'store' particles

- Find location of a particle in constant time
 - Size of grid cells adapted for collision detection
 - No missing communication
- Re-partitioning only required in case of imbalance

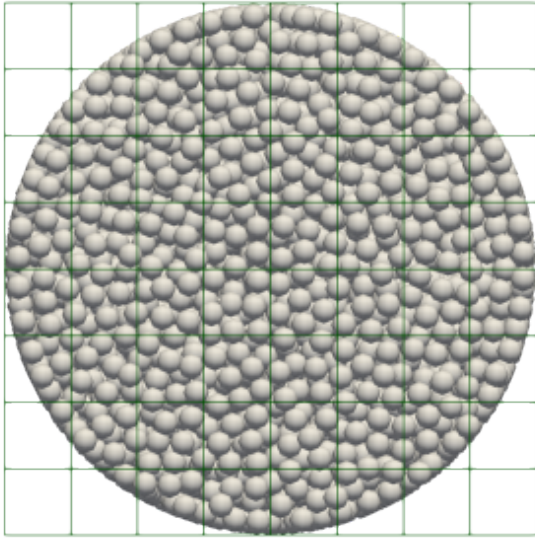


MPI communication from 1 to 0

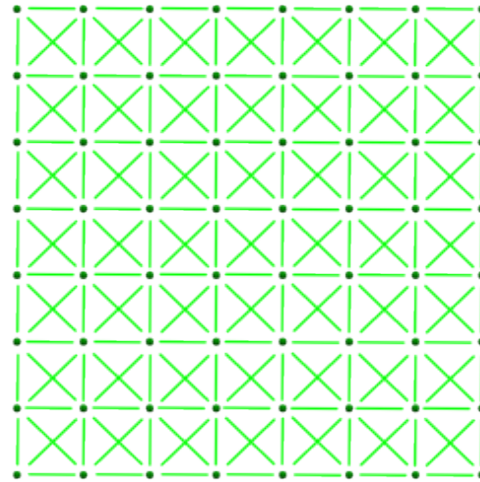


MPI communication from 0 to 1

Partitioning and Load-Balancing for XDEM

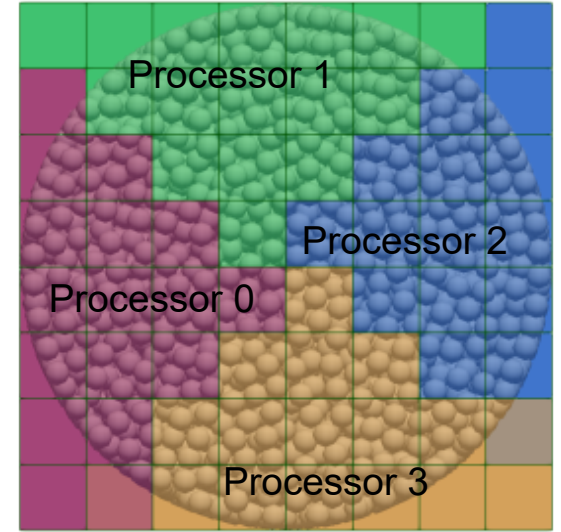


Particles in the cell grid



From grid to graph

- Node \leftarrow Cell
- Node weight $\leftarrow f(\text{nb particles})$
~ Computation cost
- Edge \leftarrow Neighborhood relation
- Edge weight $\leftarrow g(\text{nb particles})$
~ Communication cost
- Node Coordinates (topologic approaches)



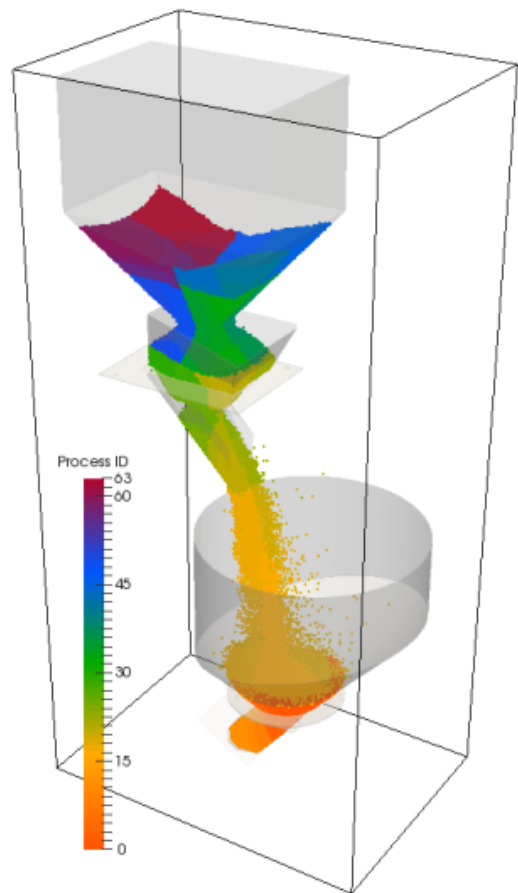
Partitioning algorithm

- Orthogonal Recursive Bisection
- METIS
- SCOTCH
- Zoltan PHG, RCB, RIB, ...
- etc.

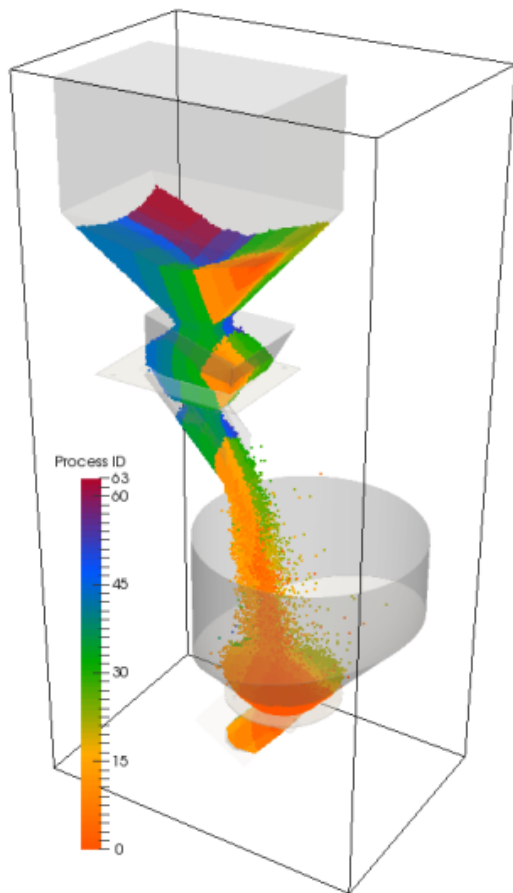
Objectives

- Balance the computation cost
- Minimize the communication cuts

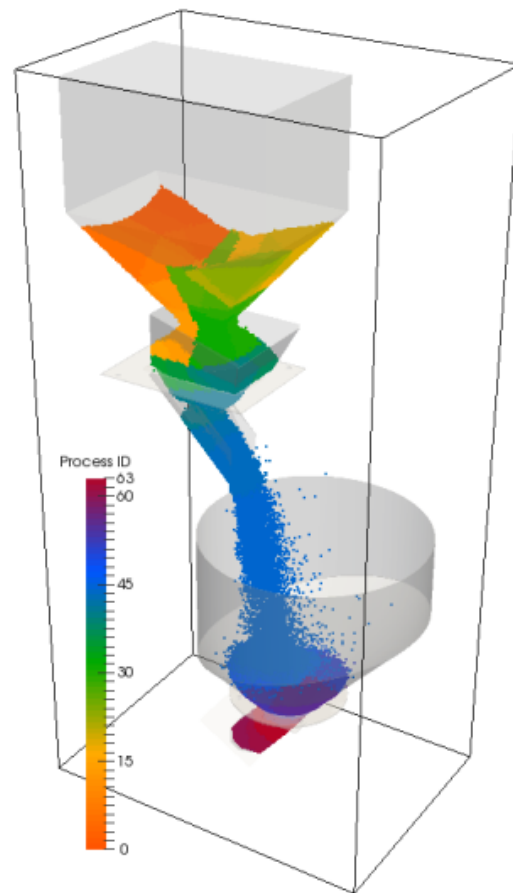
Example of Load-Balancing



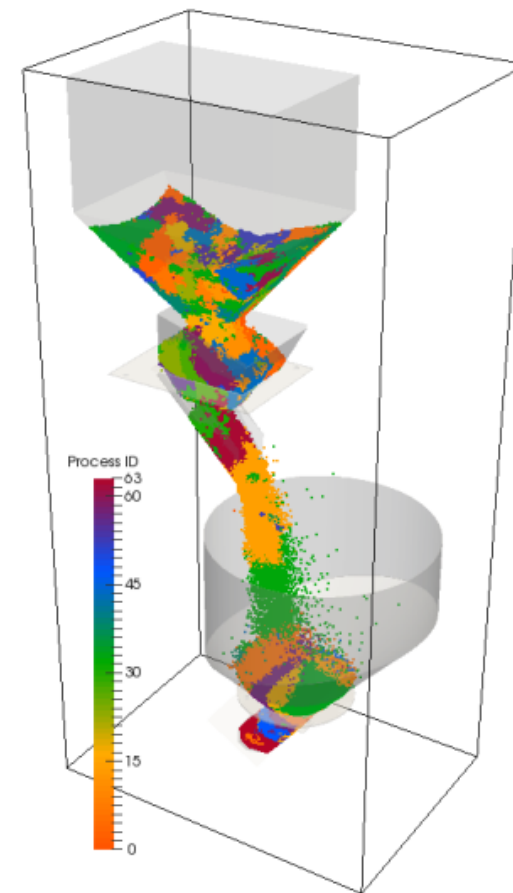
Zoltan RCB
(Recursive Coordinate Bisection)



ORB
(Orthogonal Recursive Bisection)



Zoltan RIB
(Recursive Inertial Bisection)



SCOTCH K-way



Workload Estimation for XDEM

Toward better Load-Balancing

Main Computations Phases in XDEM

Broad Phase: Fast but approximate scan to identify the pairs of particles that *could* interact

- uses an approximate shape (bounding volume)

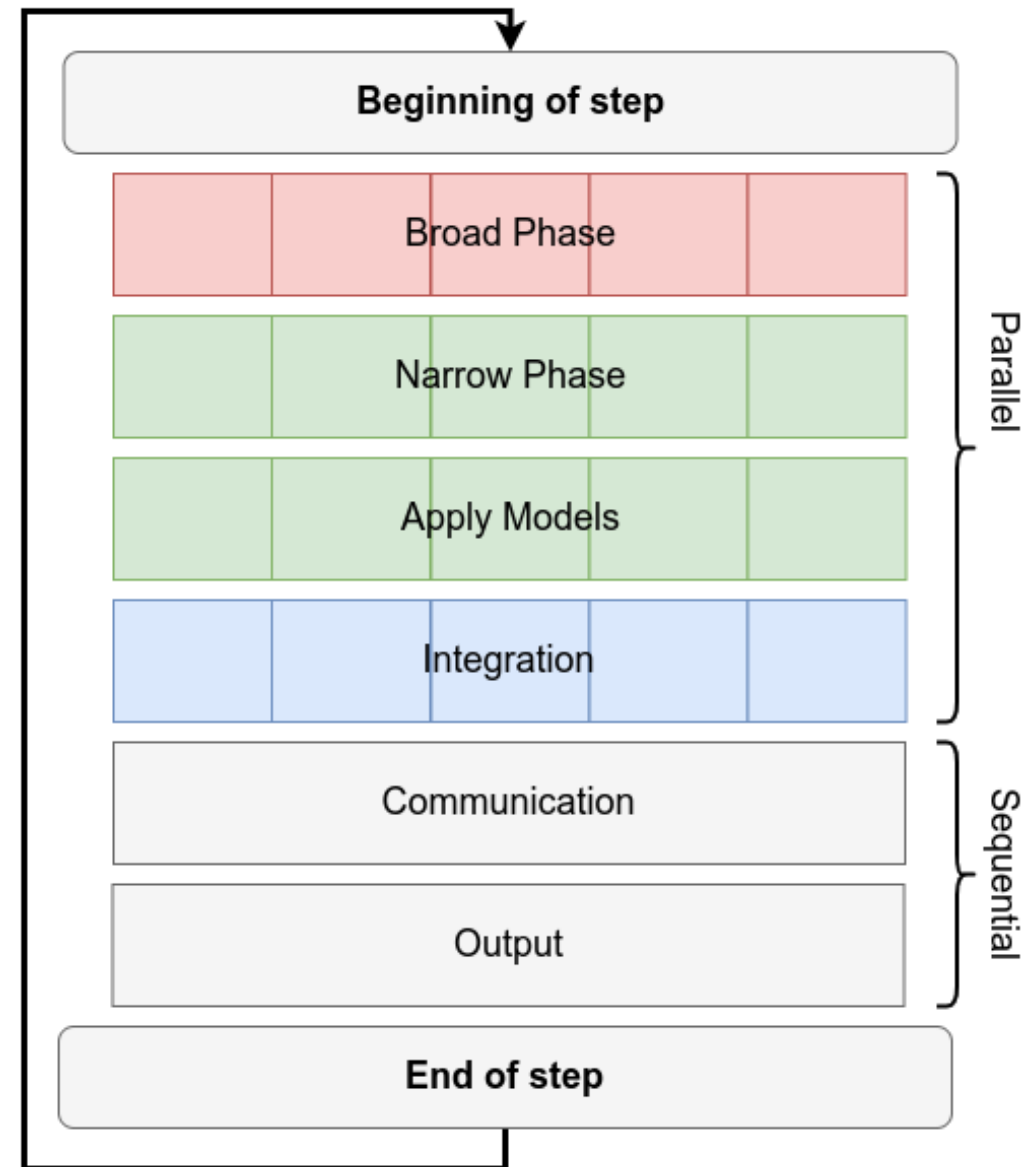
Narrow Phase: Precise collision detection on the particle pairs identified in the broad-phase

- uses the actual shape (sphere, cube, cylinder, etc.)
- calculates the distance/overlap between particles

Apply Models: Apply the physics models to each pair of interacting particles

- accumulate contributions to each particle:
Contact → *force, torque, ...*
Conduction/Radiation → *heat flux, ...*

Integration: Update the particle states by integrating the contributions from all the interacting partners



Weight estimation for load-balancing

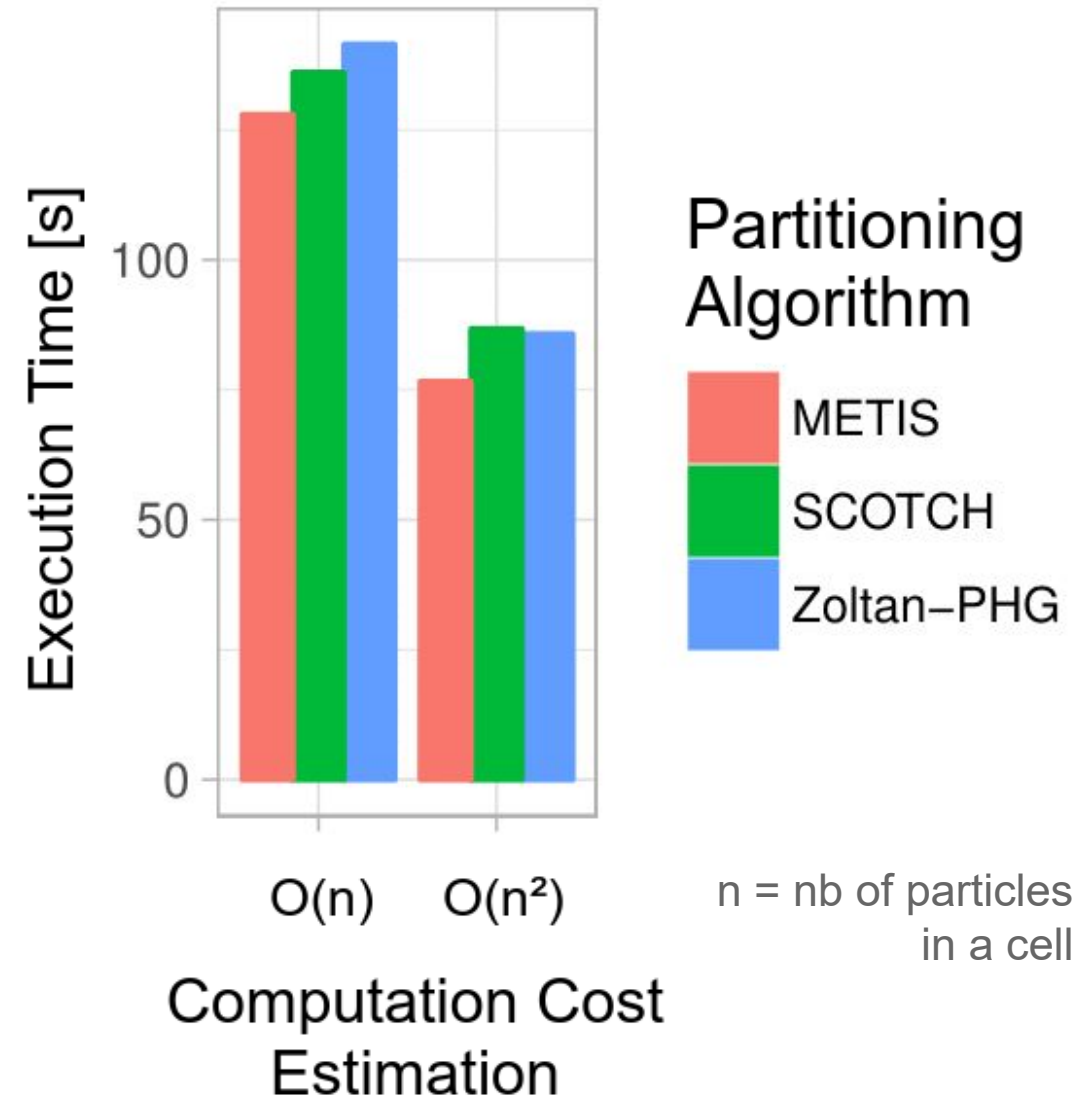
How to estimate the computing cost ?

- Difficult to measure at the level of a single cell
- Multiple phases and different complexities

<i>Computation Phase</i>	<i>Complexity</i>
Broad-phase	$O(\text{nb particles in cell}^2)$
Narrow-phase	$O(\text{nb interactions})$
Apply Models	$O(\text{nb interactions})$
Integration	$O(\text{nb particles})$

- Nb of interactions is difficult to estimate

→ **Workload estimation** has a significant impact on the **load-balancing** and on the **performance**

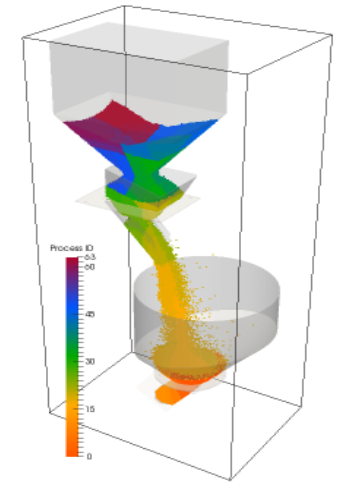
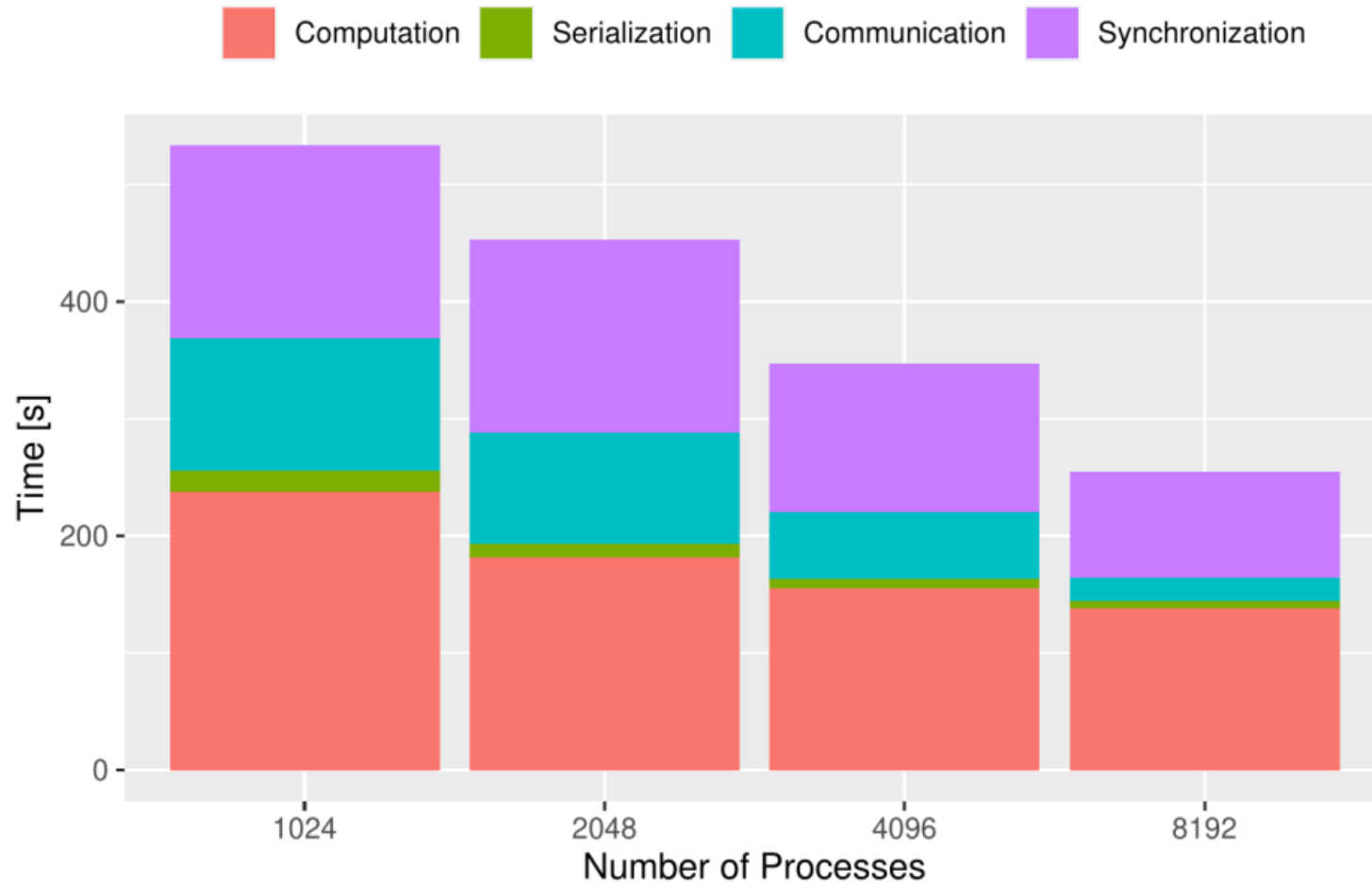


Preliminary Results

Load Estimation and Imbalance

Profiling large scale execution

- Use 'extra' synchronizations to isolate the phases in the execution



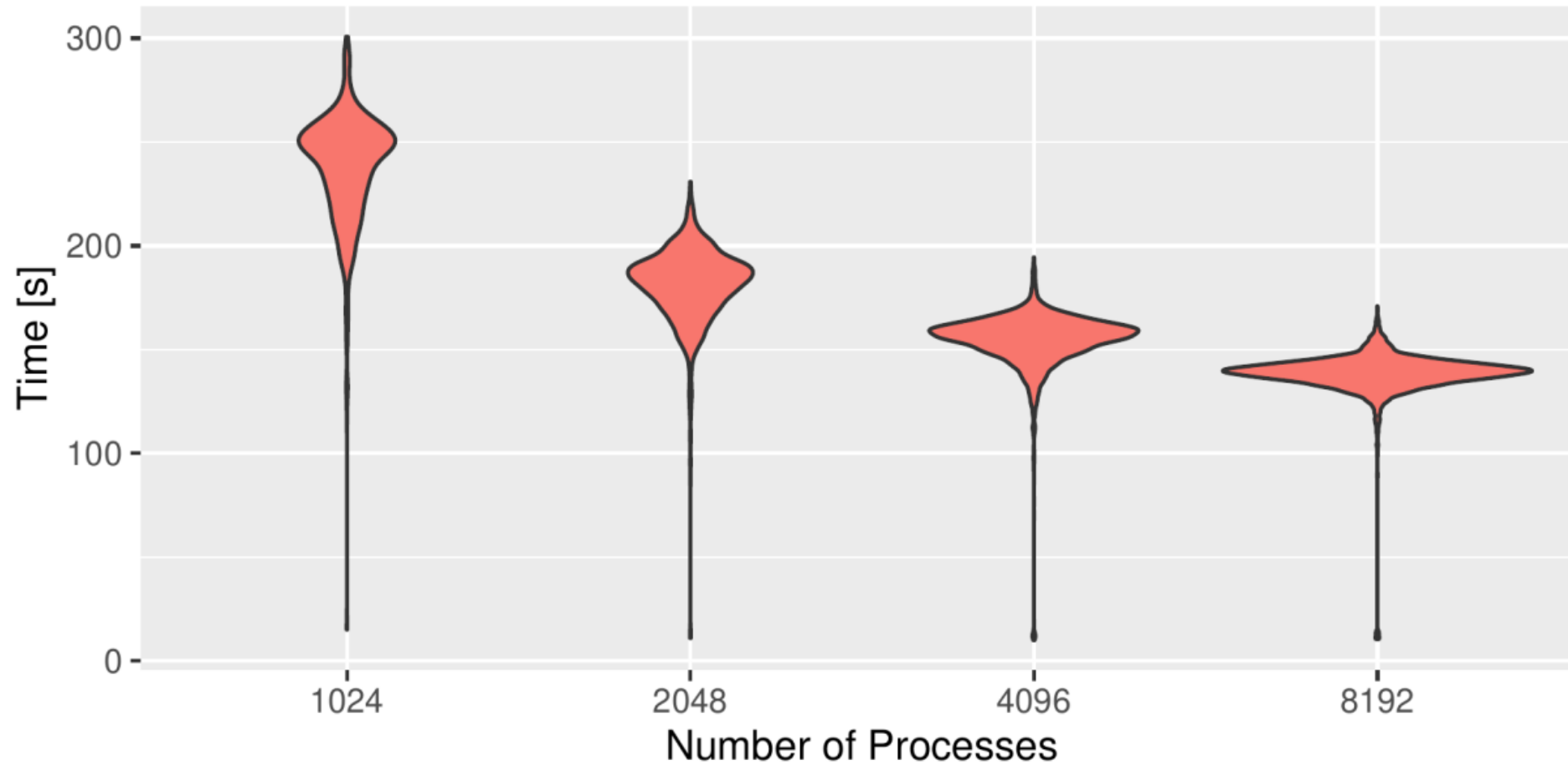
Hopper discharge with 5.5M particles

1000 timesteps
Partitioner: Zoltan-RCB
Cost function: $1+n^2$

→ Time spent in synchronization indicates imbalance between the processes

Measured Imbalance

Distribution of the computation time (excluding communication-related time)



Measured
Imbalance:

1.26

1.27

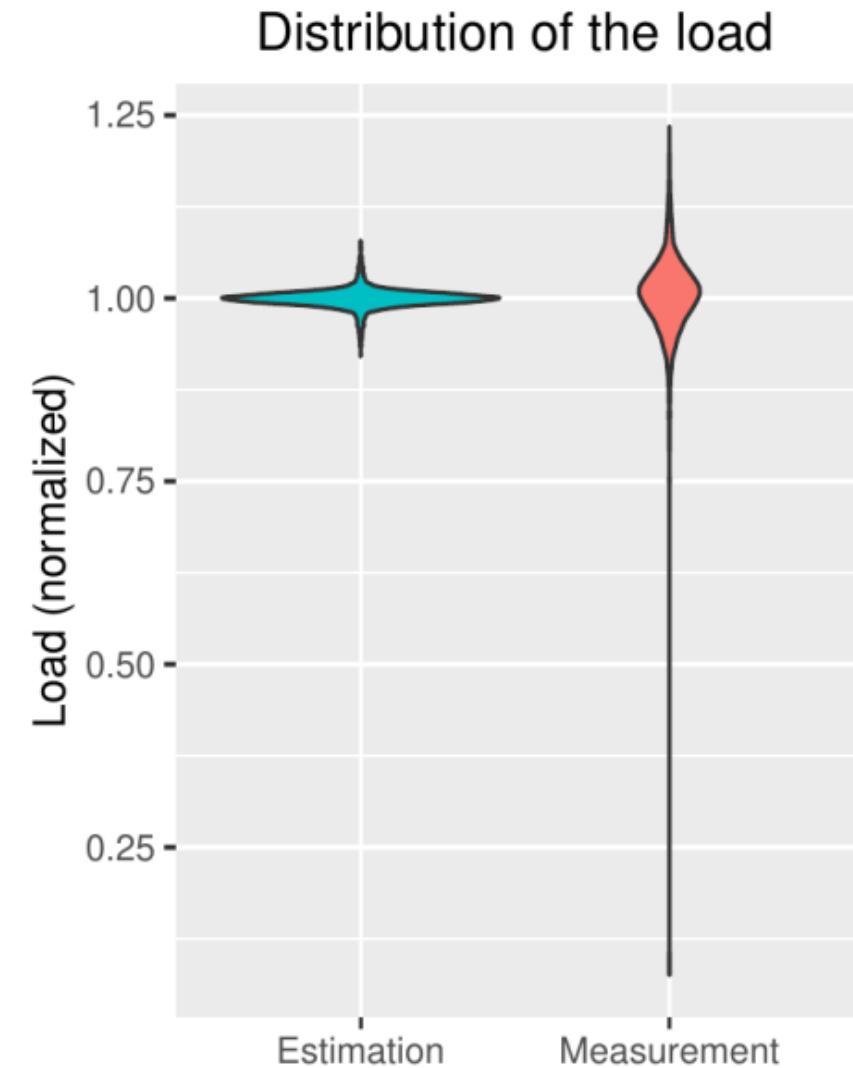
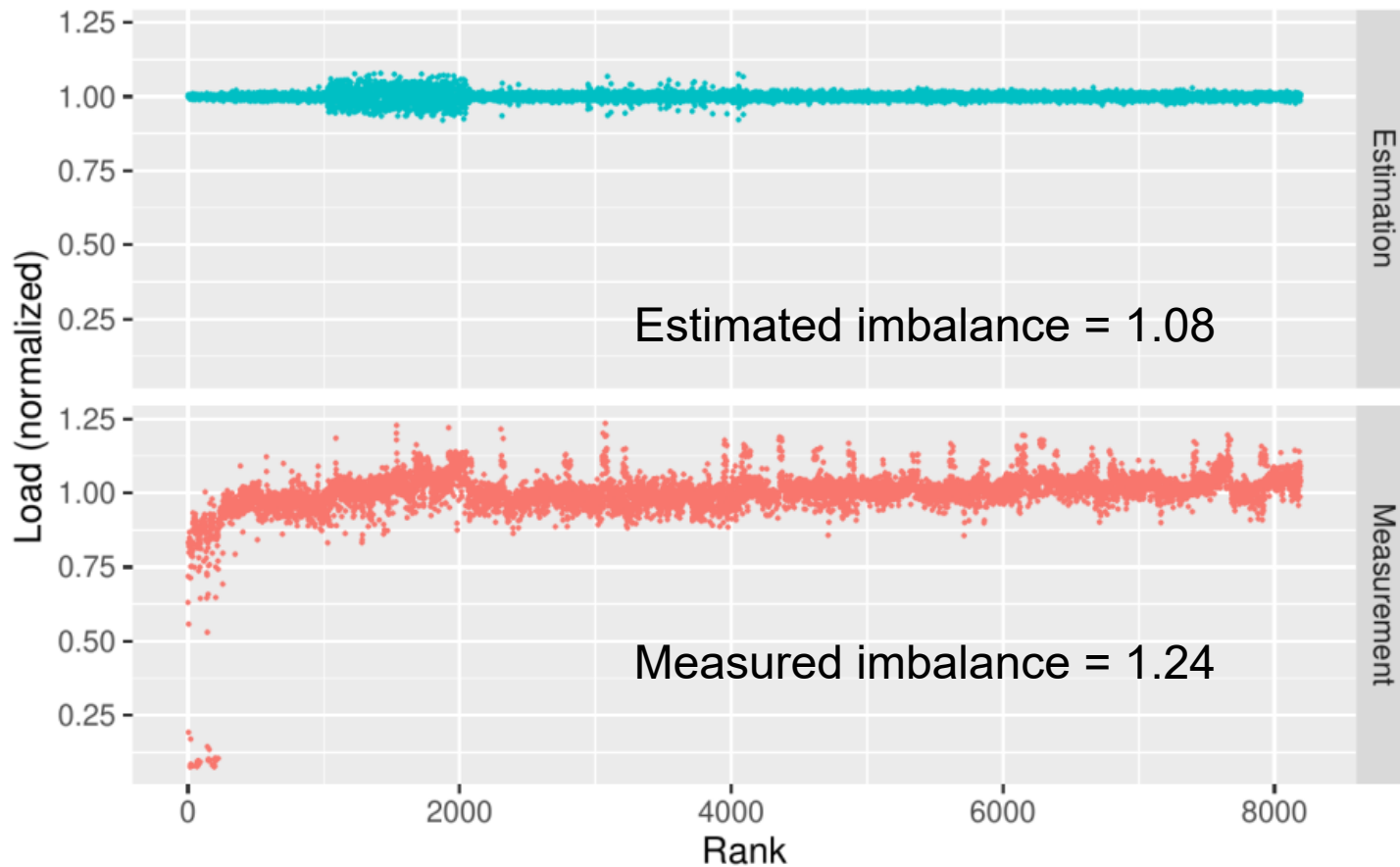
1.26

1.24



Estimated Load vs Measured Load

Load for each process



- Discrepancy between the estimated and the measured load
- The load-balancing depends on a good load estimation

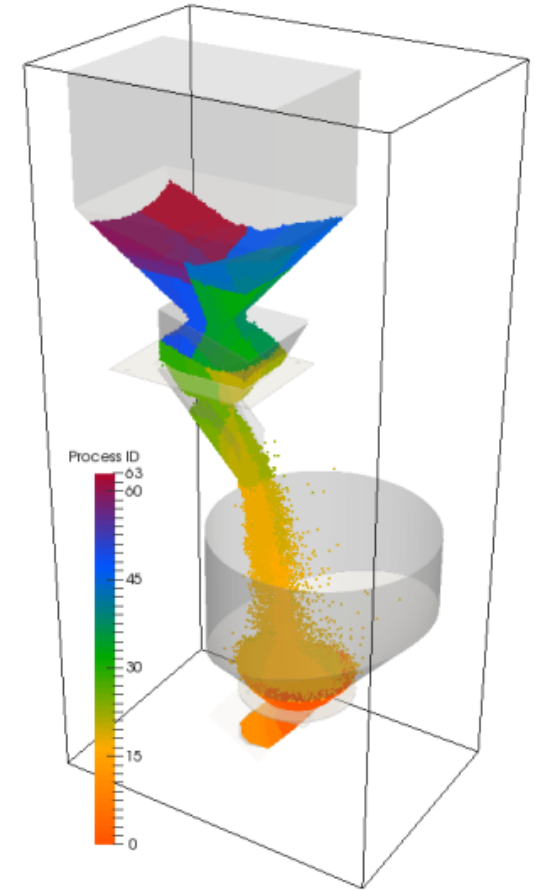
→ **Propose an accurate load estimation function for XDEM** (work-in-progress)

Thank you for your attention !

Question?

Xavier Besseron
University of Luxembourg

LuXDEM Research Team
<https://luxdem.uni.lu>



We acknowledge EuroHPC JU for awarding this project access to MeluXina.



Project: Workload Estimation and Load-Balancing of Discrete Element Method
Period: Sep. 2023 – Aug. 2024
EuroHPC used: MeluXina



Some results presented in this research were carried out using the HPC facilities of the University of Luxembourg