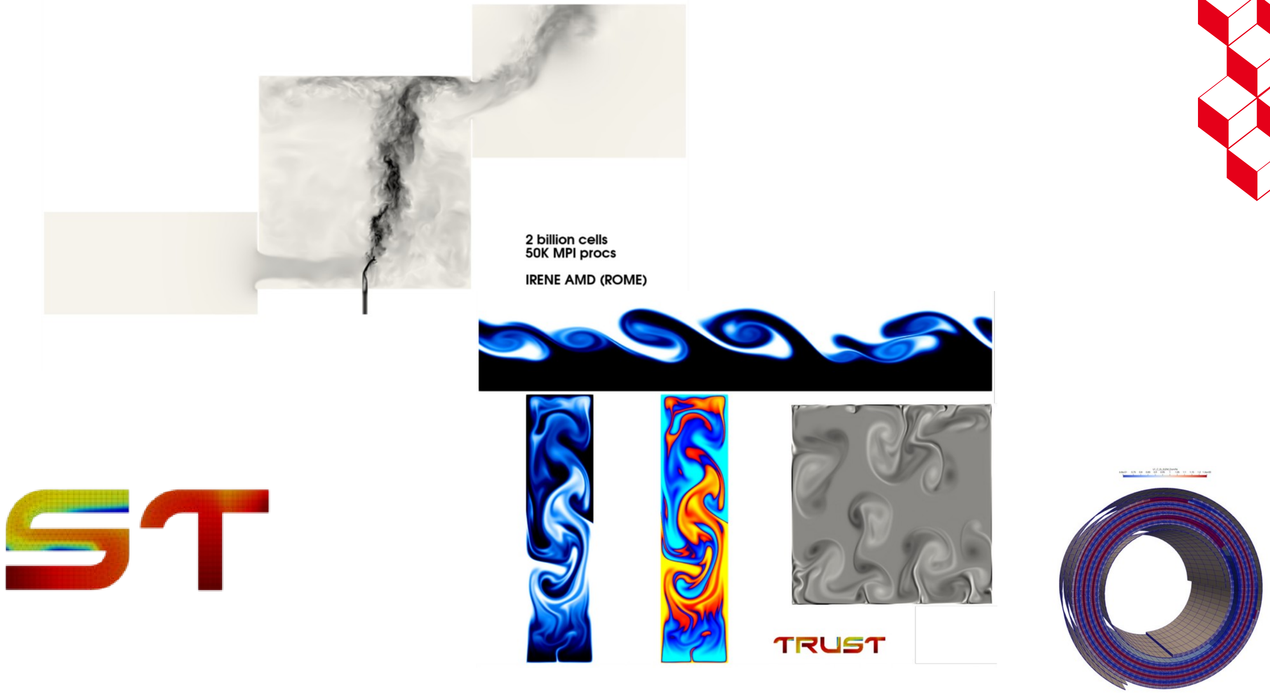
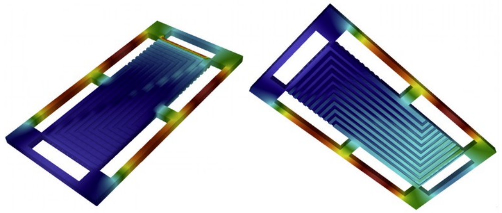


CExA project: DES TRUST/TrioCFD demonstrator



Pierre LEDAC

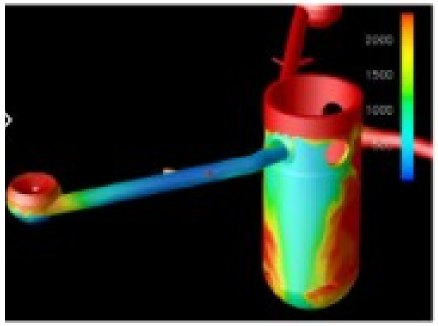
CEA/ISAS/DM2S/SGLS/LCAN

TRUST/TrioCFD

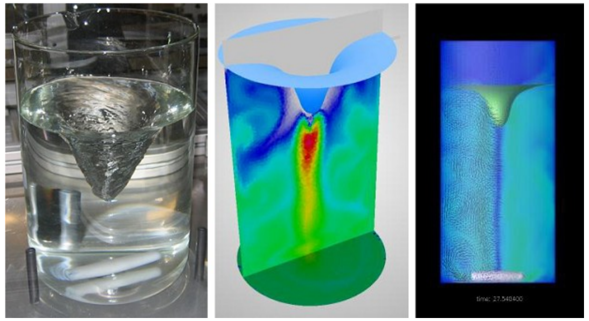
Fluid mechanics platform (CEA/DES/DM2S/SGLS/LCAN)

CFD application based on TRUST platform

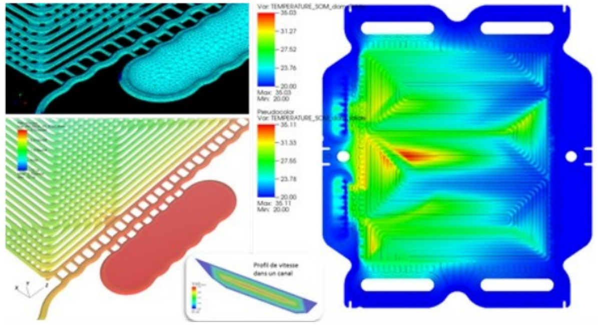
- Fluid mechanics
  - Incompressible/weakly compressible flows
  - Single or multi-phases flows
  - Front tracking module
- C++ (300K LOC), MPI, OpenSource https://github.com/cea-trust-platform
- Simulation examples:



PWR reactor



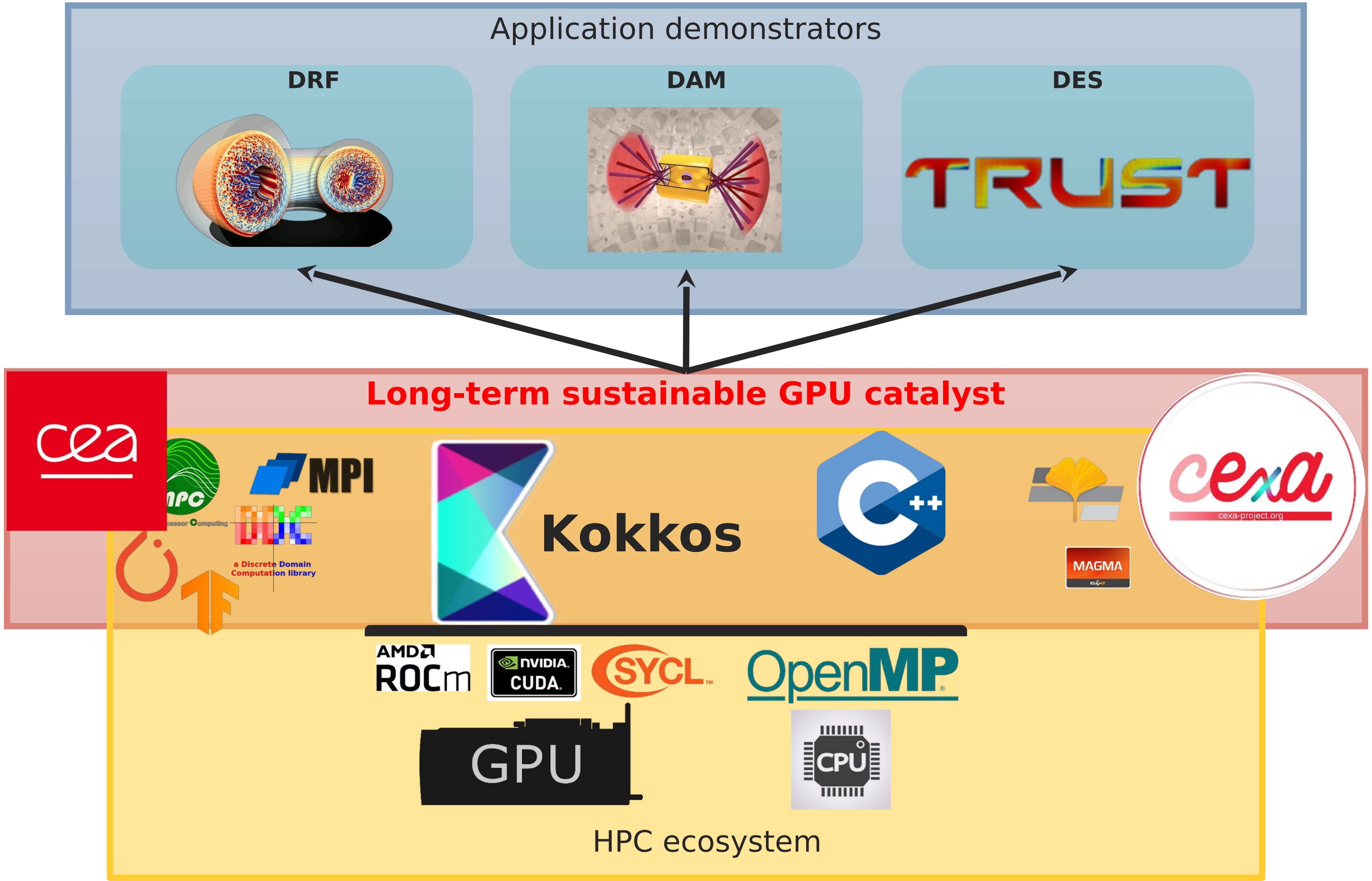
Vortex agitator



Fuel cell

1 CExA project

Kokkos choice in CExA (exascale project for CEA)



Application demonstrators

DRF

DAM

DES

Long-term sustainable GPU catalyst

HPC ecosystem

Disseminate and offer training in CEA and at large

Adapt application demonstrators

Provide a long-term sustainable software catalyst for GPU computing

Calendar of CExA project on TRUST

- Before 2023:
  - A GPU implementation was available in TRUST (OpenMP-target directives)
  - Limited to laminar incompressible flow (Poiseuille like)
- T4 2023:
  - CExA project kick-off
  - First implementation of views/kernels with Kokkos (A. Bruneton)
- T1 2024:
  - First verified run with Kokkos and OpenMP-target directives
- T2 2024:
  - Porting code, porting code, porting code,... with Kokkos !
  - Code regularly merged into TRUST releases (but still tagged ”experimental”)
- T3 2024:
  - First run on Nvidia H100 with all physical used kernels in Kokkos

CExA project

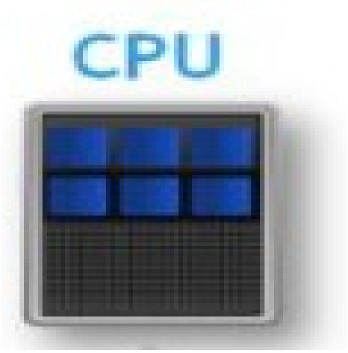
- TRUST team working on demonstrator (1 ETPT):
  - Pierre LEDAC, Rémi BOURGEOIS, Adrien BRUNETON, Thomas GONCALVES

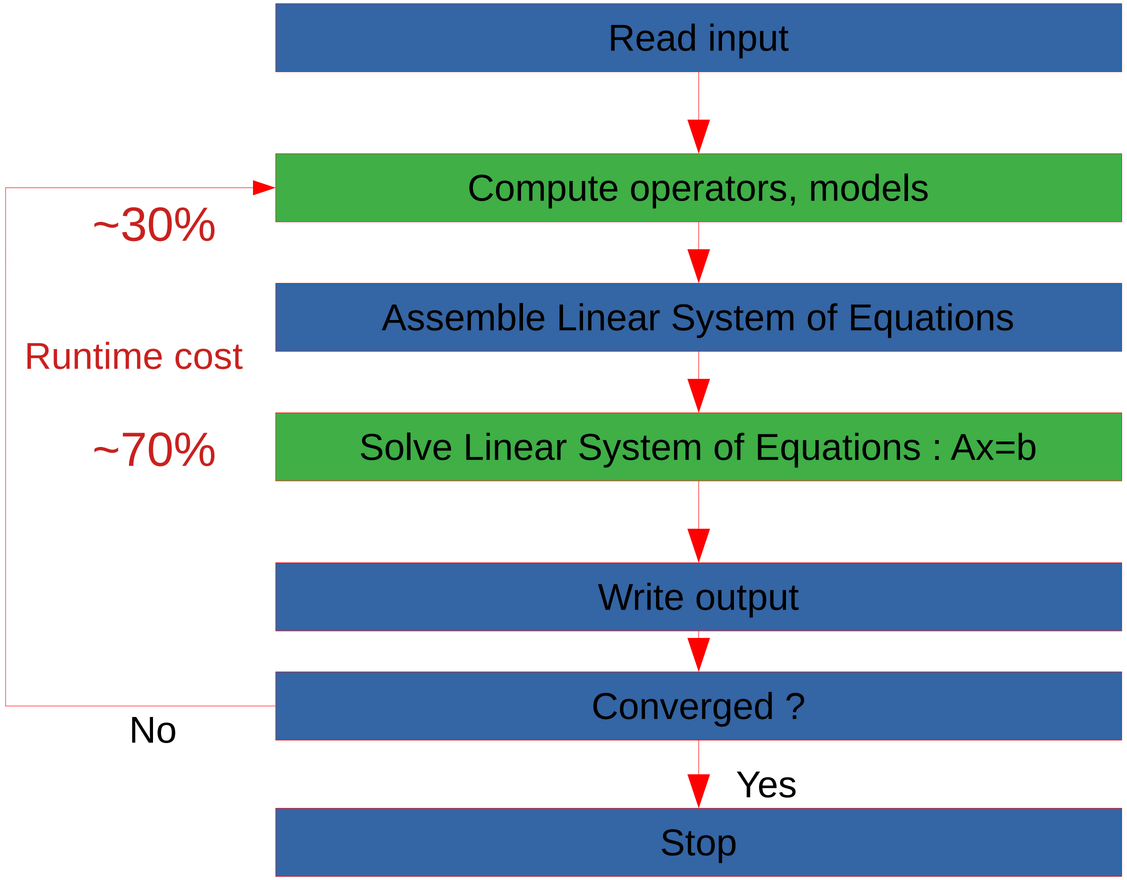
- CExA team for TRUST demonstrator (weekly/monthly meetings):
  - Paul ZEHNER, Rémi BARON, Hariprasad KANNAN, Mathieu LOBET

- Hot topics (solved, bypassed soon, current):
  - Kernel dynamic scheduling
  - Continous integration slowness
  - Virtual function in Kokkos regions
  - MI250 performance issue with OpenMPtarget backend
  - Kokkos SIMD (exp not available for GNU compiler)
  - MPI GPU-Aware robustness (not Kokkos related)

2 Kokkos porting status

Using Kokkos for GPU in TRUST ?





Read input

Compute operators, models

~30%

Runtime cost

~70%

Assemble Linear System of Equations

Solve Linear System of Equations : Ax=b

Write output

Converged ?

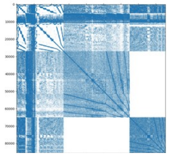
No

Yes

Stop

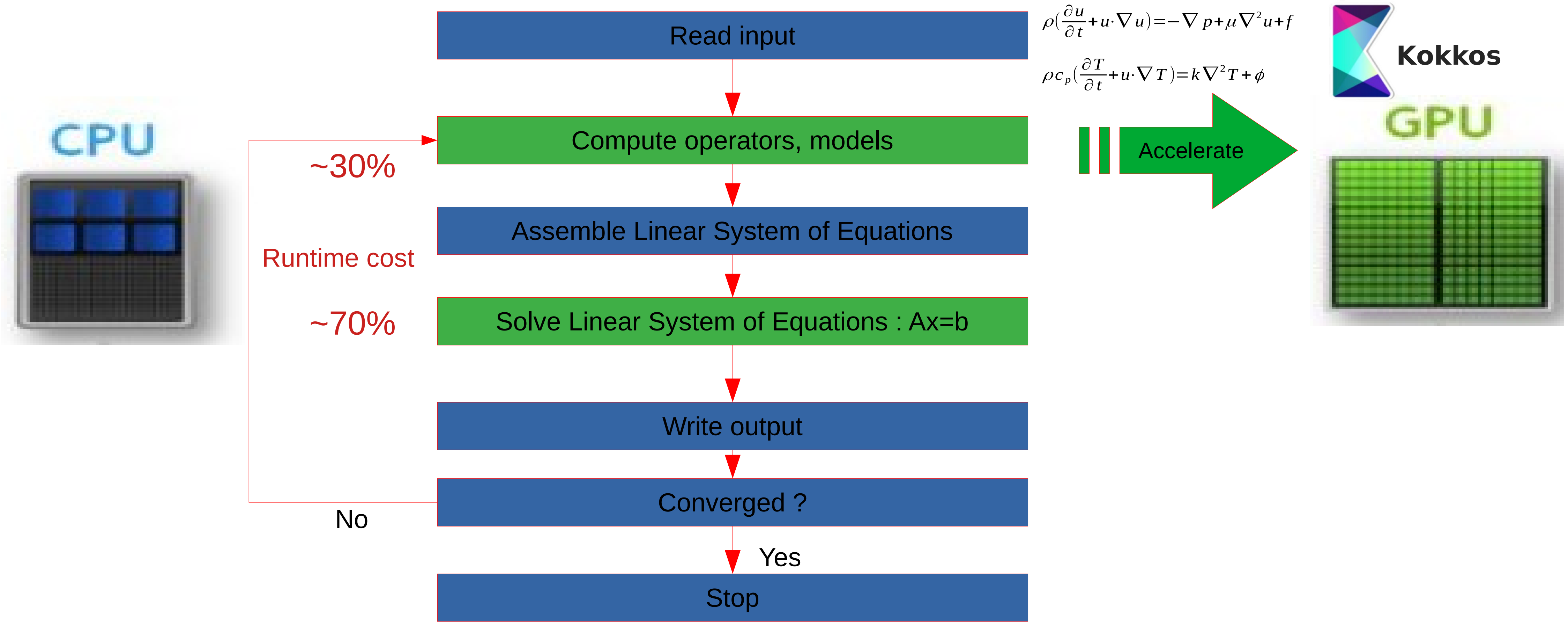
$$\rho\left(\frac{\partial u}{\partial t}+u\cdot\nabla u\right)=-\nabla p+\mu\nabla^2 u+f$$

$$\rho c_p\left(\frac{\partial T}{\partial t}+u\cdot\nabla T\right)=k\nabla^2 T+\phi$$



- Detect the most CPU expensive algorithms candidate to GPU

Using Kokkos for GPU in TRUST ?



Read input

Compute operators, models

Assemble Linear System of Equations

Solve Linear System of Equations : Ax=b

Write output

Converged ?

No

Yes

Stop

~30%

Runtime cost

~70%

$$\rho(\frac{\partial u}{\partial t}+u\cdot\nabla u)=-\nabla p+\mu\nabla^2 u+f$$

$$\rho c_p(\frac{\partial T}{\partial t}+u\cdot\nabla T)=k\nabla^2 T+\phi$$

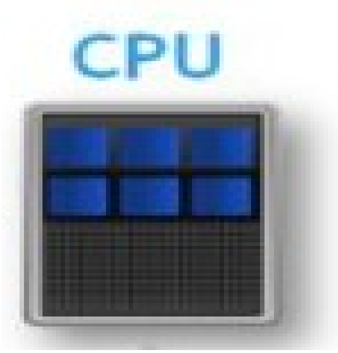
- Detect the most CPU expensive algorithms candidate to GPU
- Introduce parallelism on GPU for expensive loops (**Kokkos** framework)

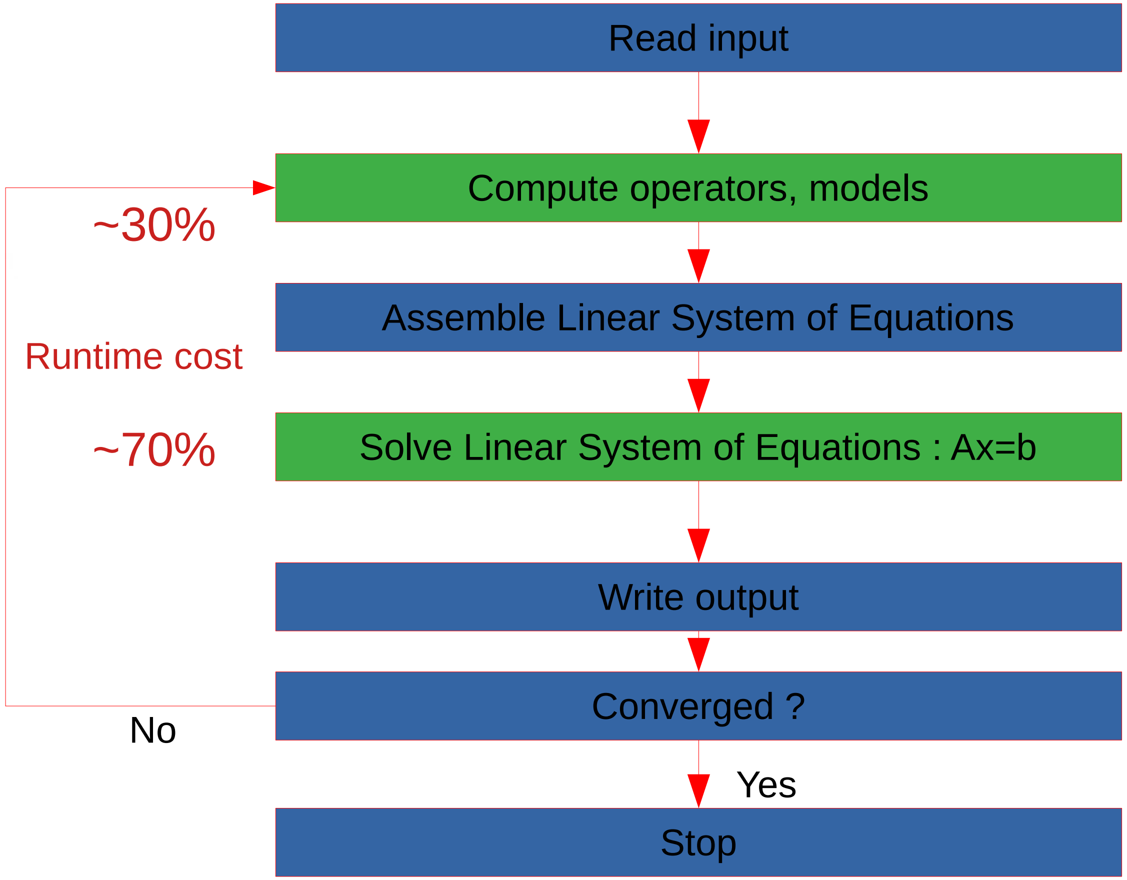
Using Kokkos for GPU in TRUST ?

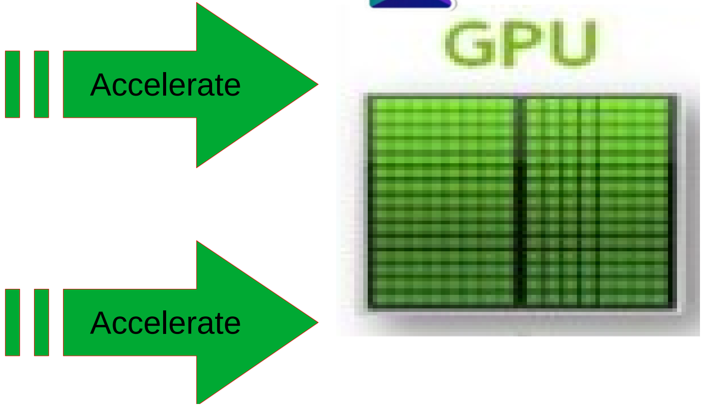
$$\rho(\frac{\partial u}{\partial t}+u\cdot\nabla u)=-\nabla p+\mu\nabla^2 u+f$$

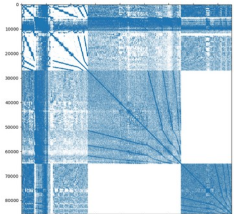
$$\rho c_p(\frac{\partial T}{\partial t}+u\cdot\nabla T)=k\nabla^2 T+\phi$$

Kokkos




Read input
Compute operators, models
~30%
Assemble Linear System of Equations
Runtime cost
~70%
Solve Linear System of Equations : Ax=b
Write output
Converged ?
No
Yes
Stop





Nvidia: AmgX, PETSc,
AMD: PETSc, rocALUTION

- Detect the most **CPU** expensive algorithms candidate to **GPU**
- Introduce parallelism on **GPU** for expensive loops (**Kokkos** framework)
- Benefit from **GPU** dedicated libraries (linear algebra)

Kokkos features TRUST use currently

- View types
  - Kokkos::Dualview
  - Unmanaged Dualview cause memory still allocated with OpenMP-target

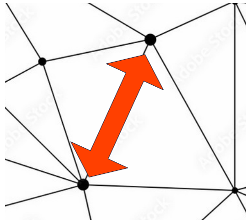
- Parallel execution
  - Kokkos::parallel_for, Kokkos::parallel_reduce

- Execution Spaces
  - Kokkos::Cuda, Kokkos::Serial, Kokkos::OpenMPtarget (for AMD)

Porting code on GPU with Kokkos

Example : Density interpolation on faces from nodes

- Loop on mesh faces
- **In**: rhonP1, rhonp1P1 (density on faces)+mesh connectivity
- **Out**: rhon_som, rhonp1_som (density on nodes)



```
const DoubleTab& rhonP1 = tab_rhonP1();
const DoubleTab& rhonp1P1 = tab_rhonp1P1();
DoubleVect& rhon_som = tab_rhon_som();
DoubleVect& rhonp1_som = tab_rhonp1_som();
for(int face=0; face<nb_faces_tot; face++)
{
  for (int som = 0; som < nsf; som++)
    {
      int som_glob = renum_som_perio(face_sommets(face, som));
      double pond = volumes_entrelaces(face) / volume_int_som(som_glob);
      rhon_som(som_glob) += rhonP1(face, 0) * pond;
      rhonp1_som(som_glob) += rhonp1P1(face, 0) * pond;
    }
}
```

```
CDoubleTabView rhonP1 = tab_rhonP1.view_ro();
CDoubleTabView rhonp1P1 = tab_rhonp1P1.view_ro();
DoubleArrView rhon_som = tab_rhon_som.view_rw();
DoubleArrView rhonp1_som = tab_rhonp1_som.view_rw();
Kokkos::parallel_for(start_gpu_timer(__KERNEL_NAME__), nb_faces_tot, KOKKOS_LAMBDA(const int face)
{
  for (int som = 0; som < nsf; som++)
    {
      int som_glob = renum_som_perio(face_sommets(face, som));
      double pond = volumes_entrelaces(face) / volume_int_som(som_glob);
      Kokkos::atomic_add(&rhon_som(som_glob), rhonP1(face, 0) * pond);
      Kokkos::atomic_add(&rhonp1_som(som_glob), rhonp1P1(face, 0) * pond);
    }
});
end_gpu_timer(Objet_U::computeOnDevice, __KERNEL_NAME__);
```

Minimal rewrite of the algorithm :

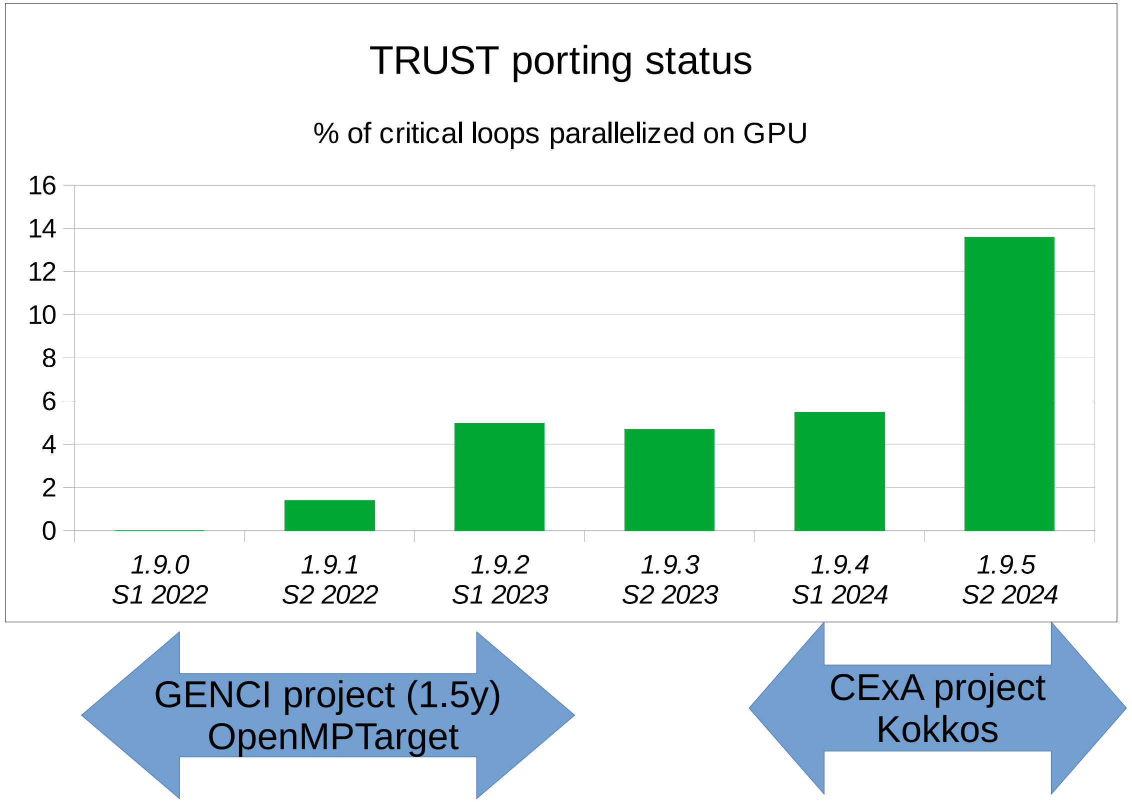
- Declare Kokkos **views** on TRUST arrays
- Decorate with Kokkos **macros**
- Add **atomics** (thread parallelism)

TRUST porting status

- 300K LOC and 1400 loops detected as critical for GPU
- Implementing Kokkos during CExA project dramatically speedup porting

TRUST porting status

- 300K LOC and 1400 loops detected as critical for GPU
- Implementing Kokkos during CExA project dramatically speedup porting

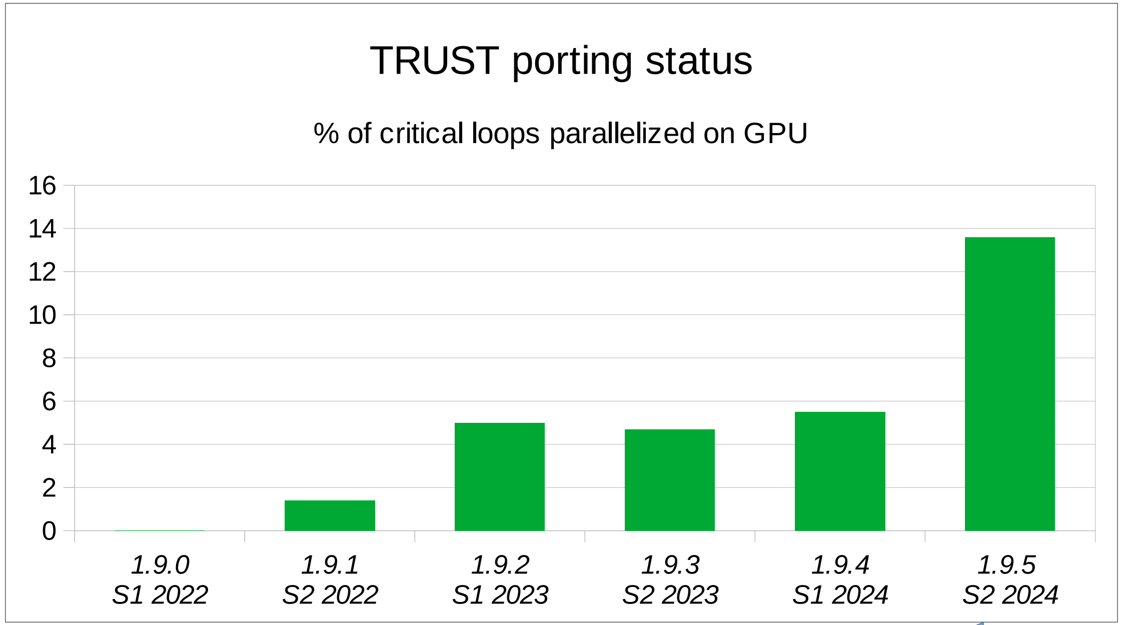


GENCI project (1.5y)
OpenMPTarget

CExA project
Kokkos

TRUST porting status

- 300K LOC and 1400 loops detected as critical for GPU
- Implementing Kokkos during CExA project dramatically speedup porting



Estimate ~50%
S2 2026

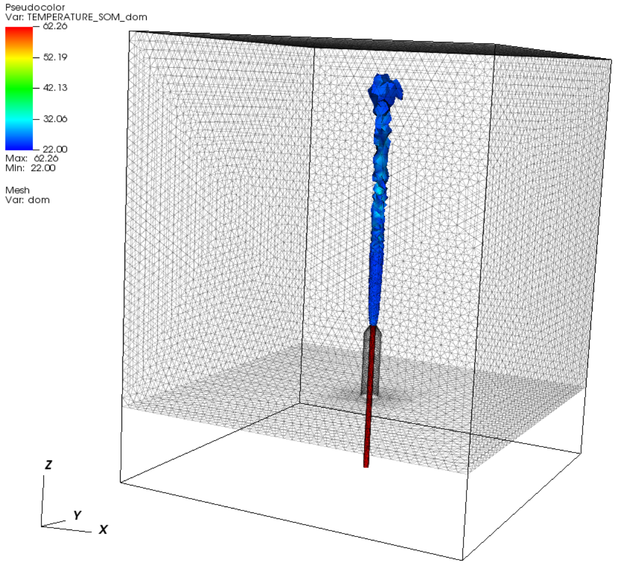
GENCI project (1.5y)
OpenMPTarget

CExA project (3y)
Kokkos

3 Kokkos performances on GPU

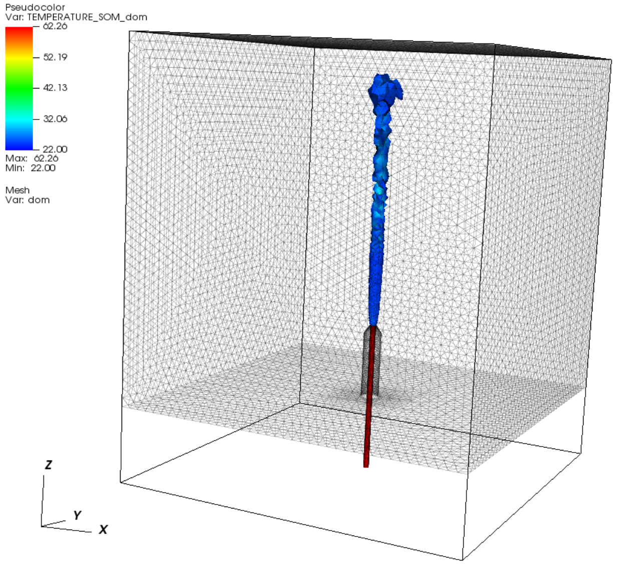
TRUST performance on GPU node

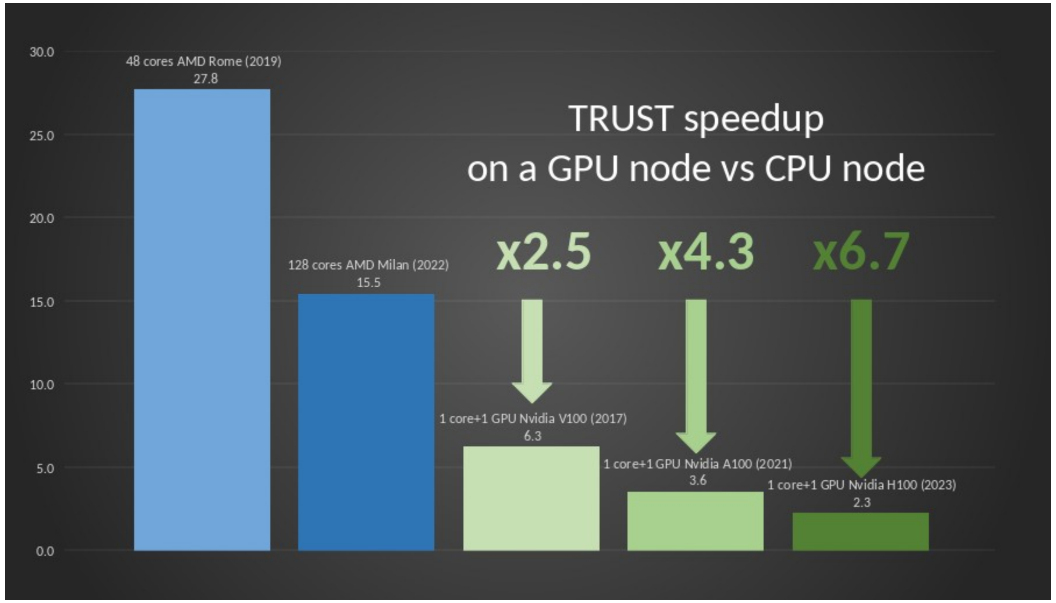
- Flow simulation:
  - Injection of turbulent jet of hot water into cold water (Re~3000)
  - LES calculation, Boussinesq hypothesis, semi-implicit scheme
  - 2.5e6 tetras mesh



TRUST performance on GPU node

- Flow simulation:
  - Injection of turbulent jet of hot water into cold water (Re~3000)
  - LES calculation, Boussinesq hypothesis, semi-implicit scheme
  - 2.5e6 tetras mesh

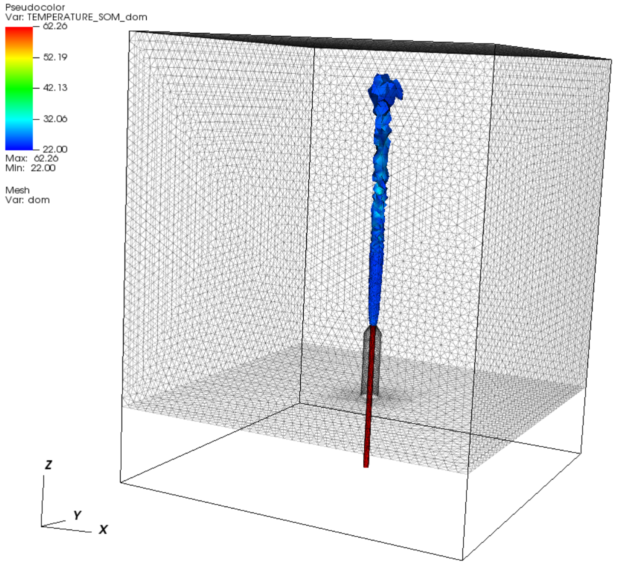


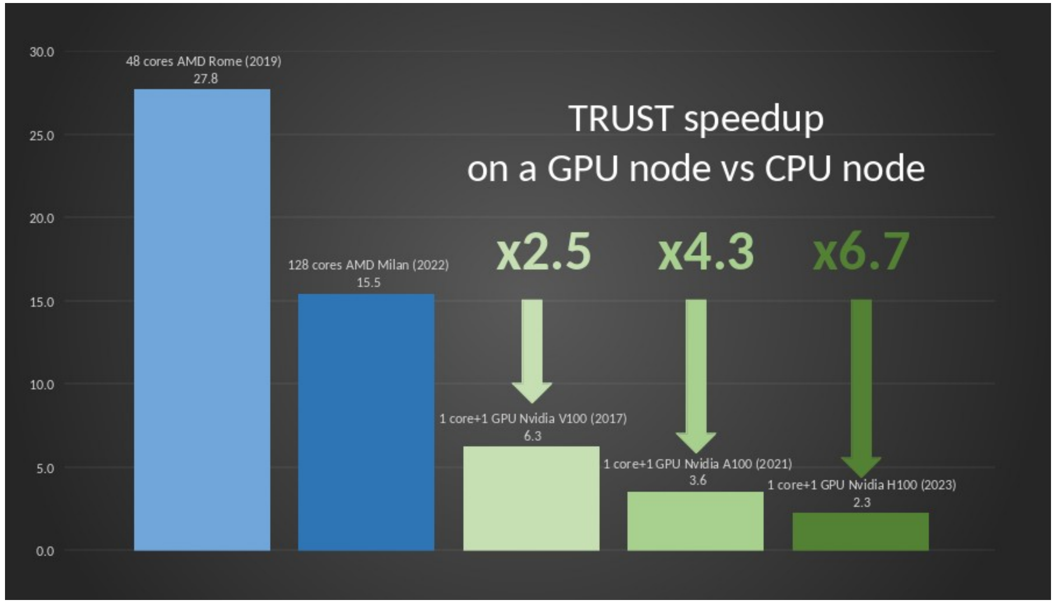


- GPU nodes (V100, A100, H100) compared to CPU nodes (48 AMD Rome & 128 AMD Milan cores)

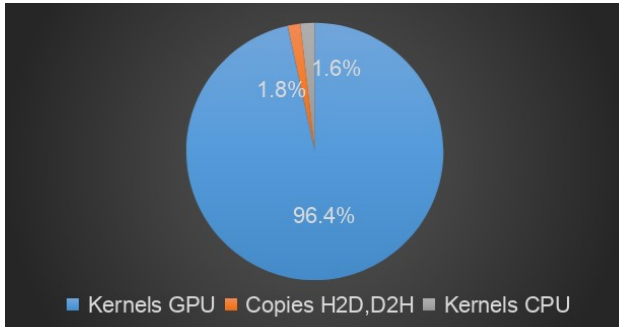
TRUST performance on GPU node

- Flow simulation:
  - Injection of turbulent jet of hot water into cold water (Re~3000)
  - LES calculation, Boussinesq hypothesis, semi-implicit scheme
  - 2.5e6 tetras mesh





- GPU nodes (V100, A100, H100) compared to CPU nodes (48 AMD Rome & 128 AMD Milan cores)
- TRUST speedup of 4 is typical when all modules used run on GPU



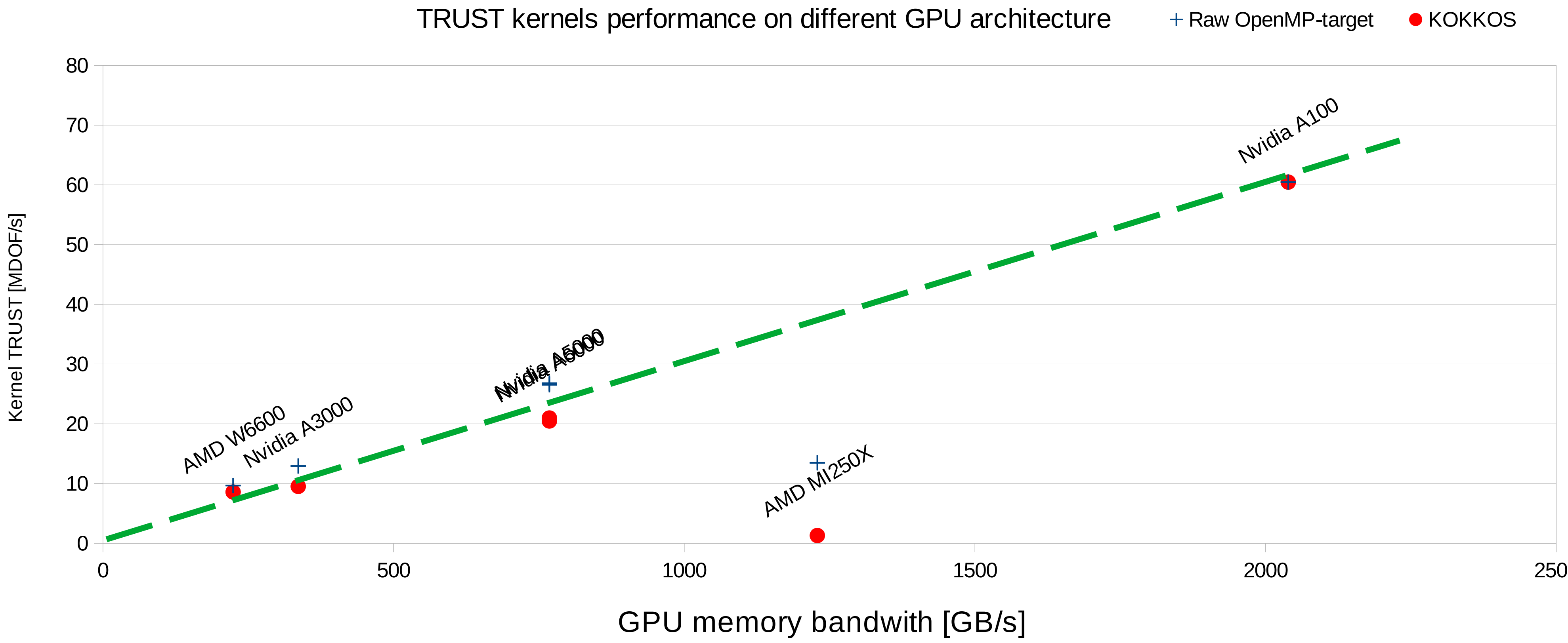
TRUST performance on GPUs

- All the heavy loops **must run** on GPU during a simulation. Why ?

- A heavy loop not ported on GPU dramatically slow down on CPU, with 4 overheads !
  - Expensive synchronization detection for arrays
    - method access (TRUSTArray::operator[]) is not inlined anymore
  - Expensive copy from **device** to **host** memory
  - Fewer CPU cores used (no GPU oversubscribing by MPI ranks)
    - 4 cores on an Nvidia 4*H100 **GPU** node
    - 128 cores on AMD Milan **CPU** node
  - Expensive copy from **host** to **device** memory later

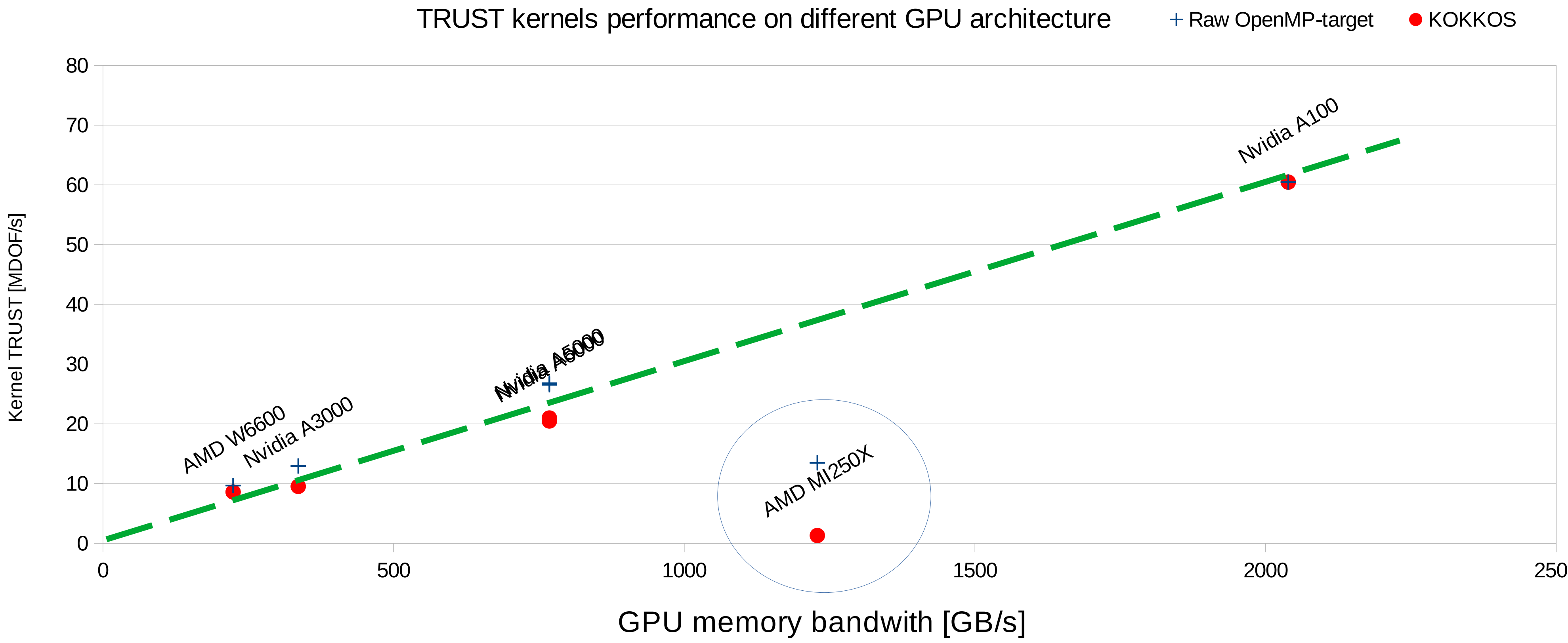
- So, 95% activity rate on **GPU** should be the main goal for each simulation for optimal performance

TRUST performance status



- TRUST kernels are memory-bound

TRUST performance status



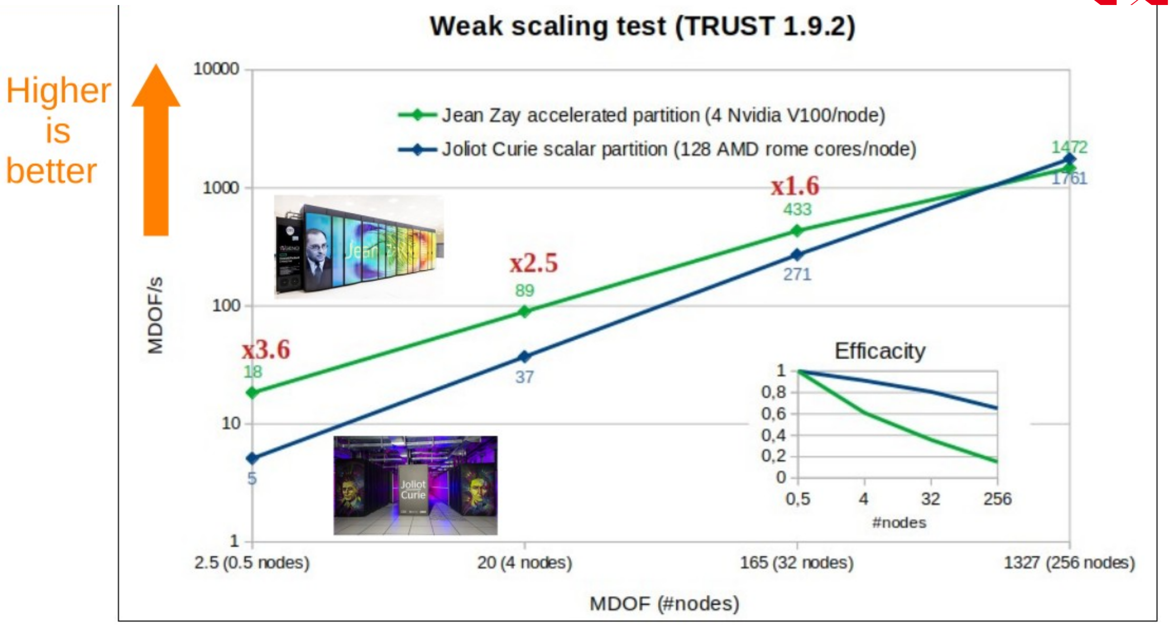
- TRUST kernels are memory-bound
- Atomics with OpenMPtarget are slow on MI250X
  - OK for other AMD cards, but MI300?
- Our raw OpenMP-target (+) kernels ran faster than Kokkos (●) ones (on low BW cards)
  - Solved since (memory access is critical !)

GPU scalability

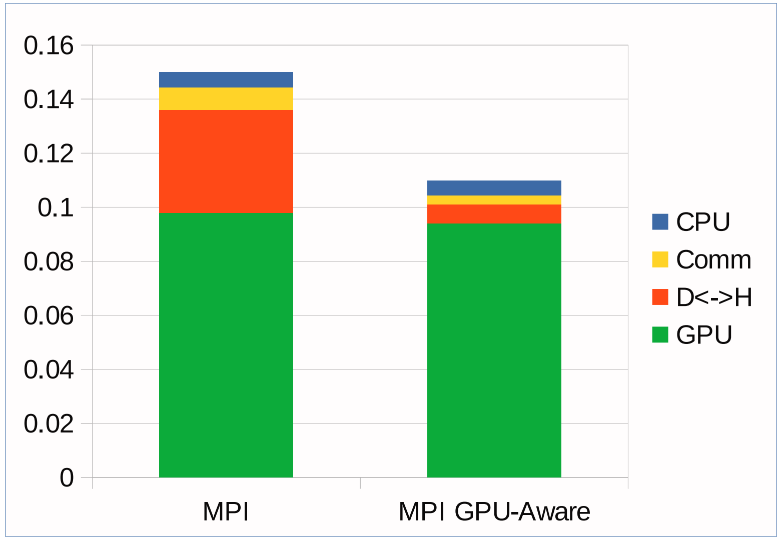
- Weak scaling test (2023, 2->1024 V100)
  - Degraded speedup, reasons:

    1) AMG linear solvers: lower convergence rate of GPU versions compared to CPU ones

    2) MPI GPU-Aware not enabled



- MPI GPU-Aware now supported in TRUST
  - Still robustness issues in TRUST dependencies (PETSc AMG preconditioner)
    - KSP_DIVERGED with OpenMPI-cuda 4.x
    - Convergence with OpenMPI-cuda 5.x !
  - When available and if it works
    - Reduced number of D<->H memory copies
    - Reduced time in communications



x1.4 on 2xV100 (Jean-Zay)

Kokkos features to evaluate

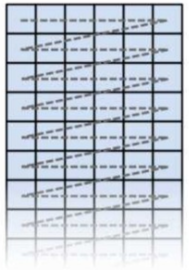
- Launch asynchrone Kokkos kernels
  - Goal: keep the **GPU** busy
  - Examples in the code:
    - Apply equations boundary condition
      - Each kernel work on a set of independant boundary faces -> easy
    - Compute equation operators
      - Currently synchrone compute, could be asynchone -> need atomic or final sum

$$\rho(\frac{\partial u}{\partial t} + \boxed{u \cdot \nabla u} = -\nabla p + \boxed{\mu \nabla^2 u} + \boxed{f}$$

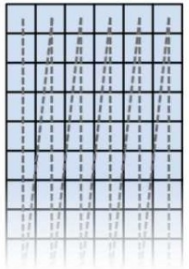
$$\rho c_p(\frac{\partial T}{\partial t} + u \cdot \nabla T) = k \nabla^2 T + \phi$$

Kokkos features to evaluate

- Optimal Kokkos view layout for multi-dimension arrays (e.g. tab(i,j)):
  - Currently using LayoutRight on **CPU** and **GPU**
    - Imposed by TRUST arrays still using OpenMP-target directives



LayoutRight



LeftLayout

  - Once OpenMP-target removed from TRUST (T4 2024)
    - LayoutLeft for arrays on **GPU** (optimal, performance should improve)
    - Add a layout transpose operation to deal with some GPU libraries

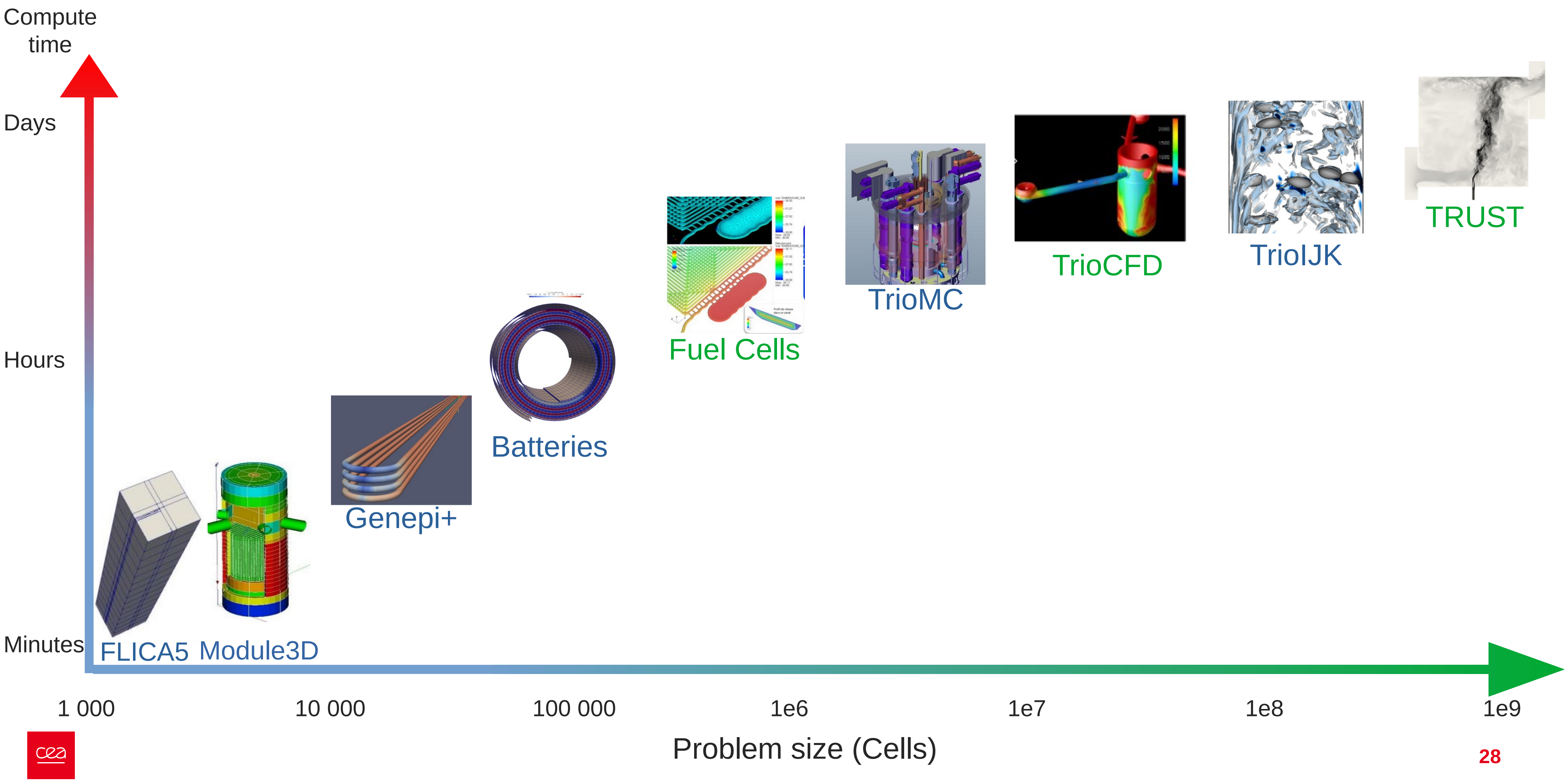
4 Conclusions & future works

CExA/Kokkos feedback

- Great collaboration with CExA team
  - Reactive, helpful, motivating

- Great choice of Kokkos
  - Easy learning curve if your C++ code has some specific patterns:
    - Modular
    - Intensive computation loops on arrays
  - In this case, porting code on GPU is:
    - Incremental
    - Faster than using a directive (OpenACC, OpenMP) or specific (CUDA, HIP) programming model
  - Lot of documentation, important community (still growing)
    - But some features undocumented
    - Missing some (trivial?) samples. E.g in Kokkos regions:
      - Handle C++ objects
      - Handle virtual function or class static attributes

Kokkos for TRUST based apps ?



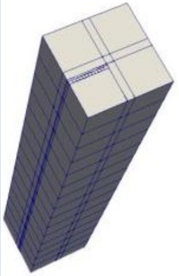
Kokkos for TRUST based apps ?

Compute time

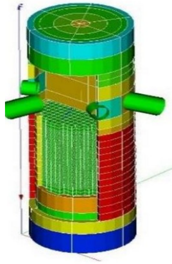
Days

Hours

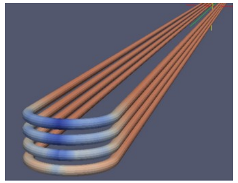
Minutes



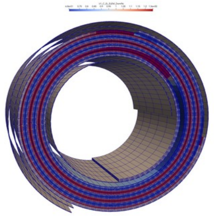
FLICA5



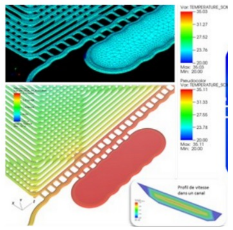
Module3D



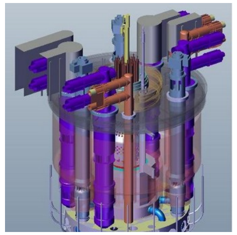
Genepi+



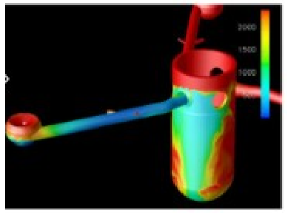
Batteries



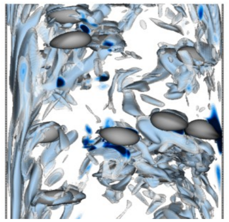
Fuel Cells



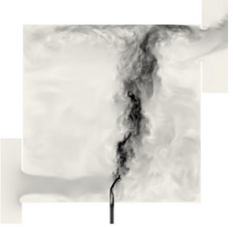
TrioMC



TrioCFD

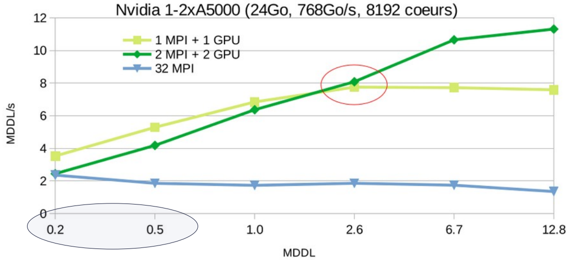


TrioIJK



TRUST

- Warning, not all apps will be **GPU** accelerated !
- Tipping point between 1e5 and 1e6 cells according to our tests



- Kokkos may help to exploit parallelism (OpenMP and/or SIMD).

1 000 CPU 10 000 CPU 100 000 CPU GPU 1e6 GPU 1e7 GPU 1e8 GPU 1e9

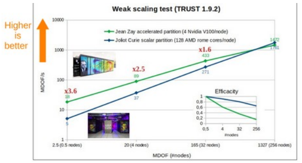
n size (

TRUST/TrioCFD roadmap for Alice Recoque

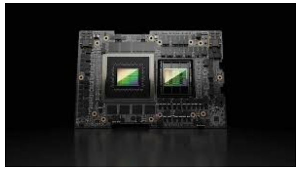
- Unknown CPU/GPU architecture yet for Alice Recoque : anticipate !
  - Contribute to **Jules Verne**, **NumPex ExaMa**, **CExA** projects

- 2025 technical roadmap for TRUST code :

  - T1 : OpenMP-target fully replaced by Kokkos framework
  - T1 : Benchmark on Adastra (AMD/MI300)



  - T2 : Improve GPU scalability



  - T3 : Benchmark on Jupyter (GH200)

  - All year : More physical modules available on GPU

    Kernel fine-tuning (layout, asynchronism, memory access,...)

Thanks. Any questions ?