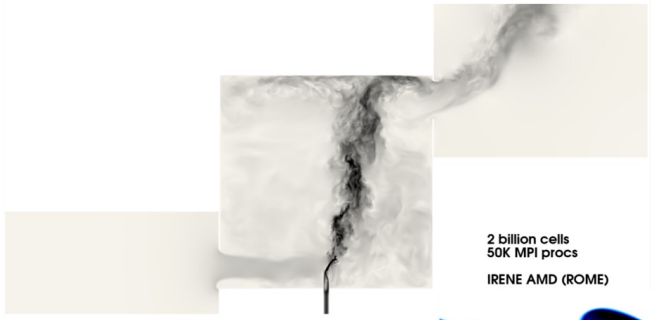
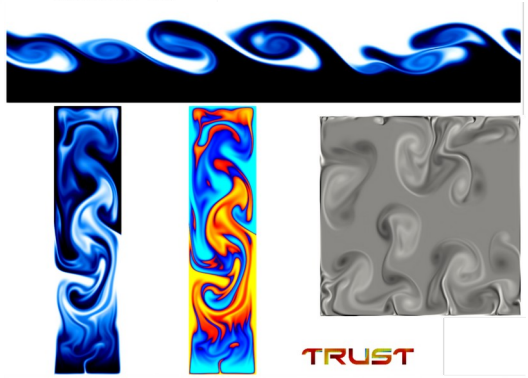
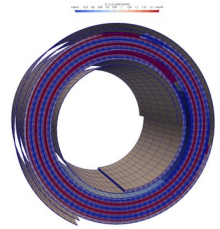
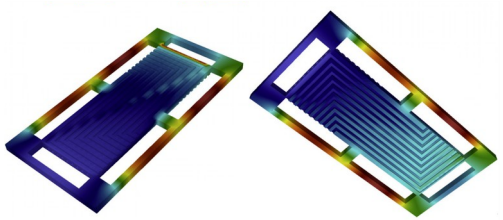








Plateforme TRUST
WP3: Solveurs
WP7: Benchmarking

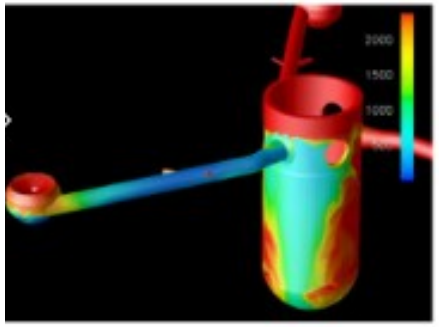
Pierre LEDAC + TRUST team

CEA (French Atomic Energy Commission), DES (Direction of Energy)

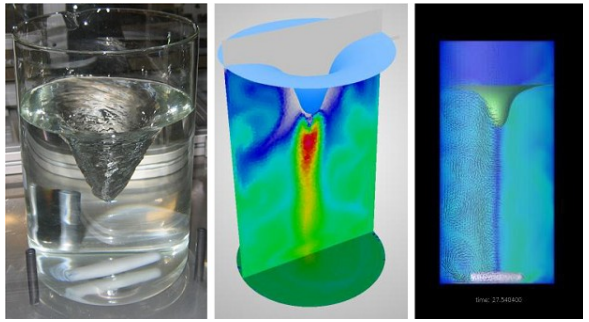
0 General presentation

TRUST Fluid mechanics platform (CEA)

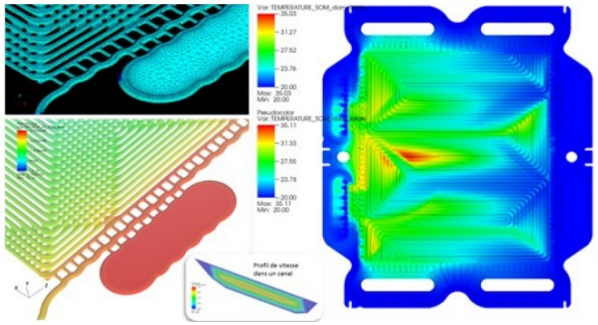
- Supports a dozen of research or industrial applications
- Demonstrator for internal **CExA** project (Kokkos based middleware for Exascale)
- Fluid mechanics
  - Incompressible/weakly compressible flows
  - Single or multi-phases flows
  - Turbulence models, Front tracking module (TrioCFD)
- C++ (300K LOC), MPI, OpenSource https://github.com/cea-trust-platform
- Modernized regularly by a growing team (~10) since 1994
- Simulation examples



PWR reactor

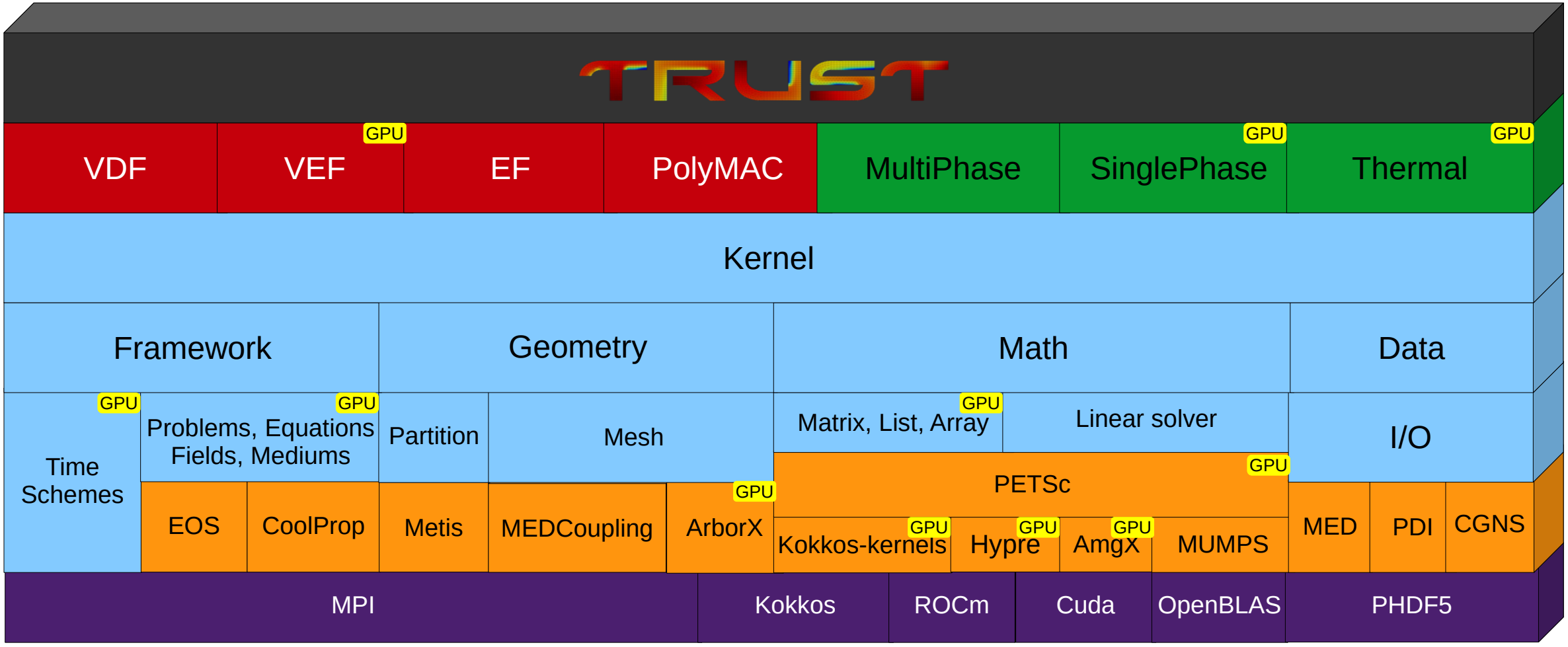


Vortex agitator



Fuel cell

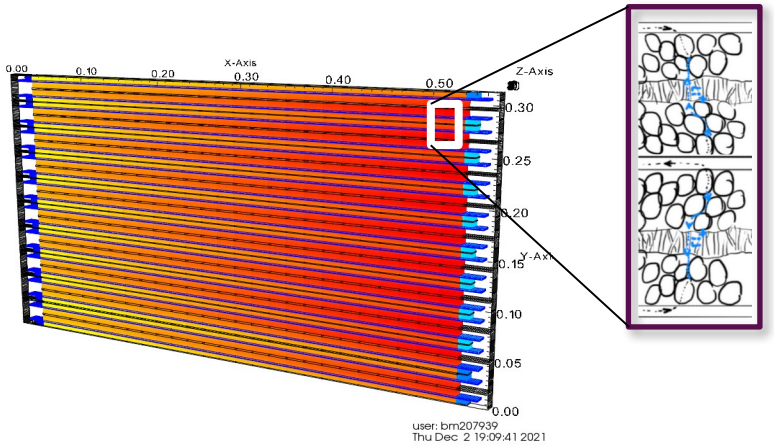
TRUST platform (1.9.8, 2026)

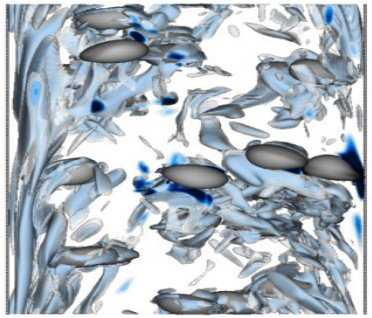


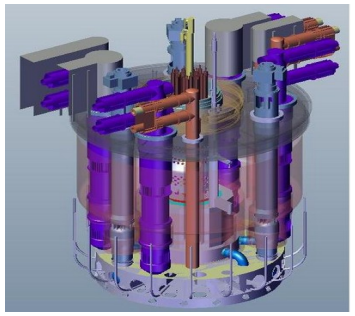
Discretizations modules | Generic modules | Physic modules | External libraries | HPC components

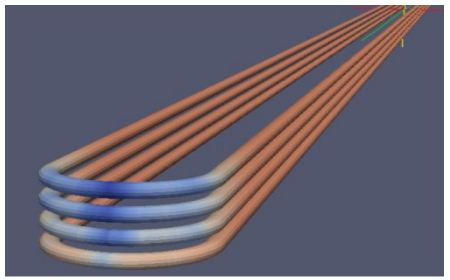
TRUST – PRINCIPALES APPLICATIONS

- TrioCFD : CFD monophasique + Front Tracking
- FLICA5 : code thermohydraulique diphasique niveau composant (code poreux)
- SympyToTRUST (STT) : code multiphysique pour la simulation (entre autres) des batteries et piles à combustible
- CATHARE3D: modélisation 3D du cœur réacteur couplée au système
- Trio-IJK : modélisation des écoulements diphasiques à l'échelle locale instantannée
- TrioMC : modélisation de la thermohydraulique du cœur des RNR
- GENEPI+ : modélisation de la thermohydraulique des GV
- PAREX+ : code de chimie 0D (pas de maillage)







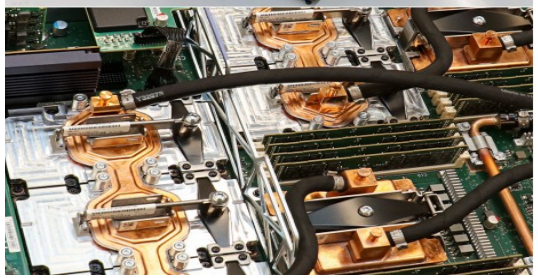


1 Exascale choices for TRUST

GPU in TRUST: a long story

2014
- First use of GPU in TRUST through PETSc

2020
- Use Nvidia AmgX GPU library

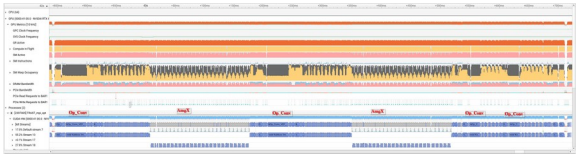


2021
- First kernels implemented with OpenACC (Nvidia Hackathon)

2022
- Partial port on AMD GPU with OpenMP-target

2023
- Select programming model Kokkos through CExA project

2024
- Migrate OpenMP-target kernels to Kokkos

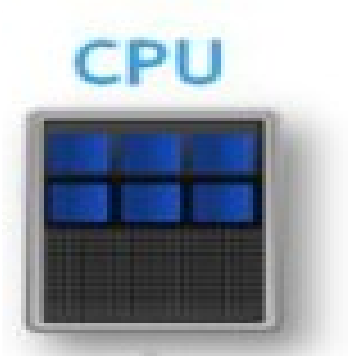


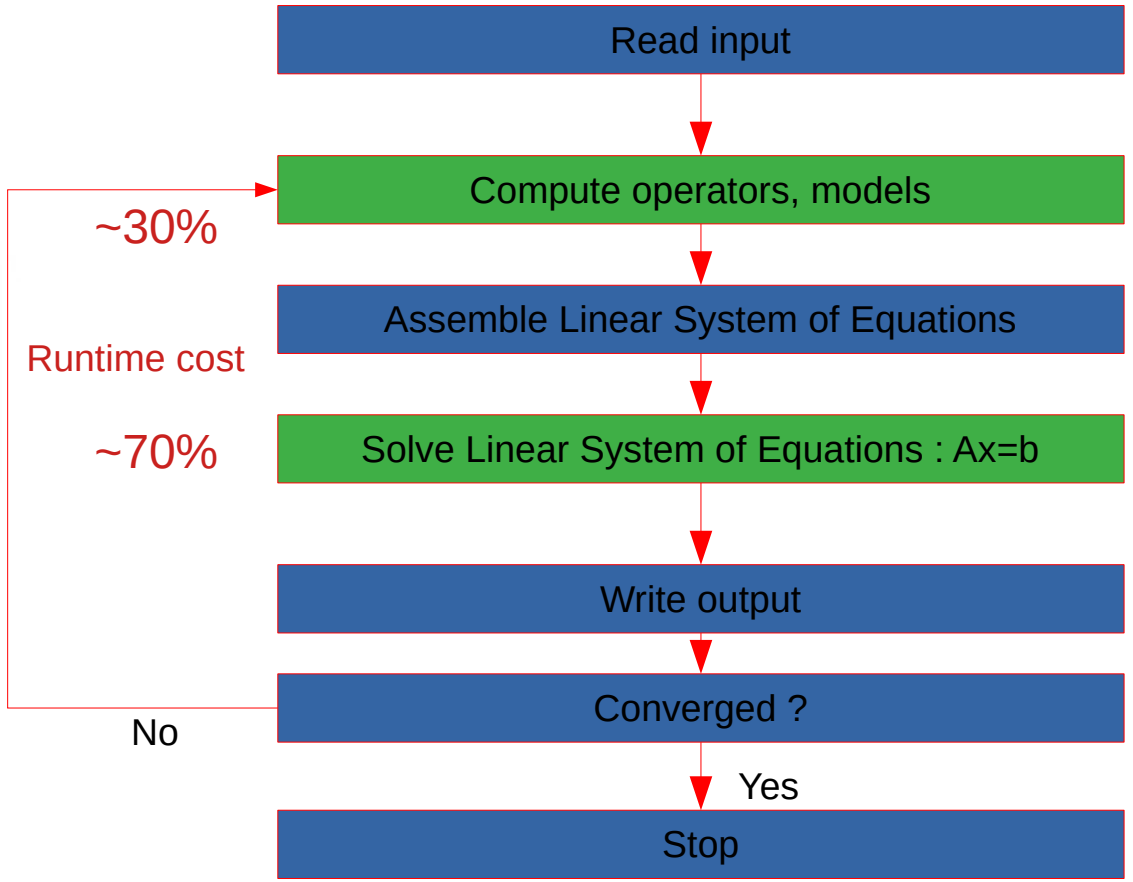
2025
- Kernels optimization + more Kokkos kernels

2026
- Simulations (~10 BDOF) on MareNostrum (H100), Lumi (MI250X), Jupiter (GH200)

2027
- TRUST on Alice Recoque : Simulation 90 BDOF (~1-2K GPUs)

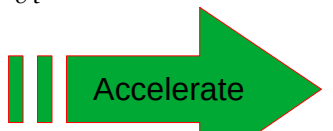
Strategy for GPU in TRUST ?

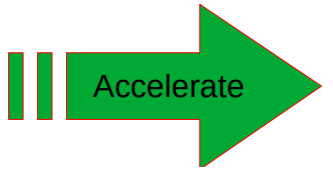


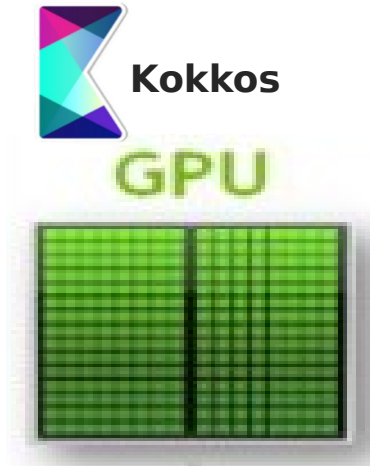


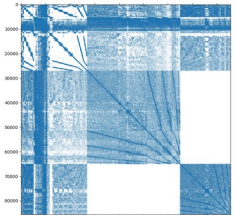
$$\rho(\frac{\partial u}{\partial t}+u\cdot\nabla u)=-\nabla p+\mu\nabla^2 u+f$$

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







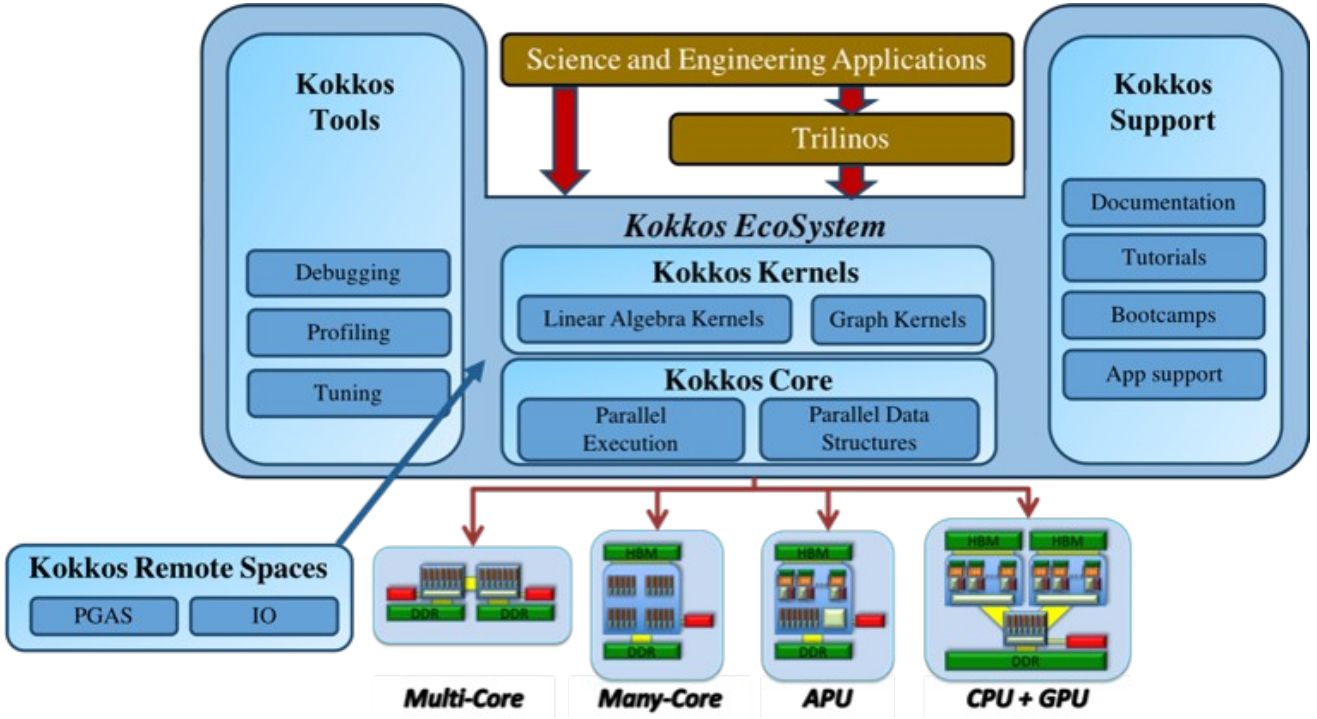


Nvidia: AmgX, PETSc/Hypre
AMD: PETSc/Hypre, rocALUTION

- 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

Sandia National Laboratories

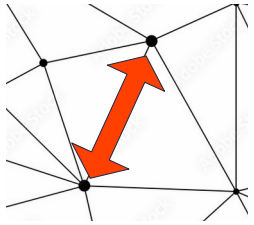


- C++ framework used in TRUST
- Provide performance portability on GPU: Unicity of source code. Several backends (Serial, Nvidia, AMD,...)
- Thanks to parallel data structures (« Views »)
- Several features for portable optimizations
  - Nested or Hierarchical //
  - Asynchronism
  - Scratch memory
  - ...

TRUST on GPU with Kokkos

Example : Density interpolation on faces from nodes

- **Loop** on mesh faces
- **In**: Density on faces+mesh connectivity
- **Out**: 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(__KERNEL_NAME__);
```

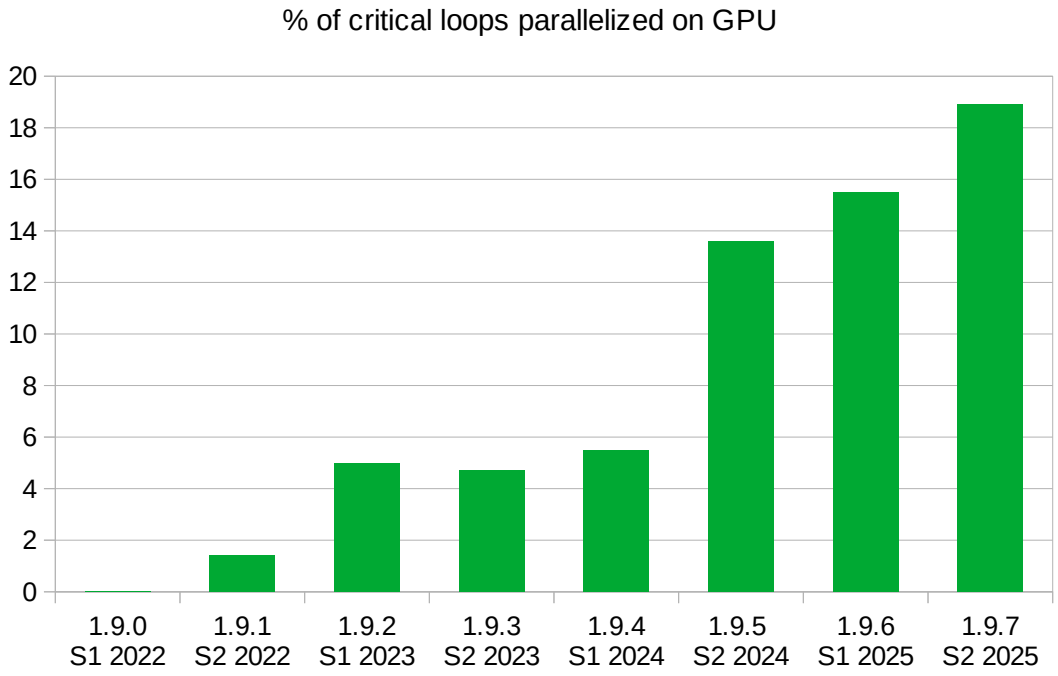
Minimal change of the initial CPU code

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

TRUST on GPU with Kokkos

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

TRUST porting status



TRUST/TrioCFD module available on GPU

Discretization	VEF	VDF
Flow	Incompressible Quasi-compressible Weakly-compressible	Incompressible Quasi-compressible Weakly-compressible Multiphase
Turbulence models	LES RANS	LES RANS
Time schemes	Explicit Semi-Implicit Implicit	Explicit Semi-Implicit Implicit
Spatial schemes	Muscl EF_stab Upwind	Quick Centre Upwind
Other	Front-Tracking Radiative heat transfer ALE	Front-Tracking Radiative heat transfer

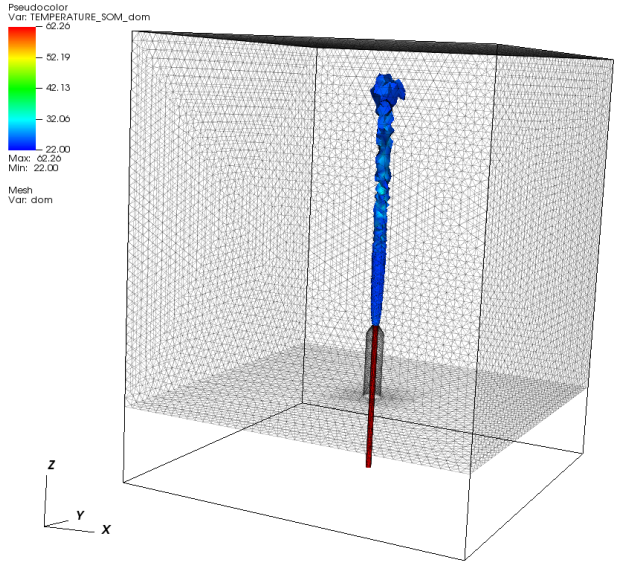
Hybrid **CPU/GPU** code:

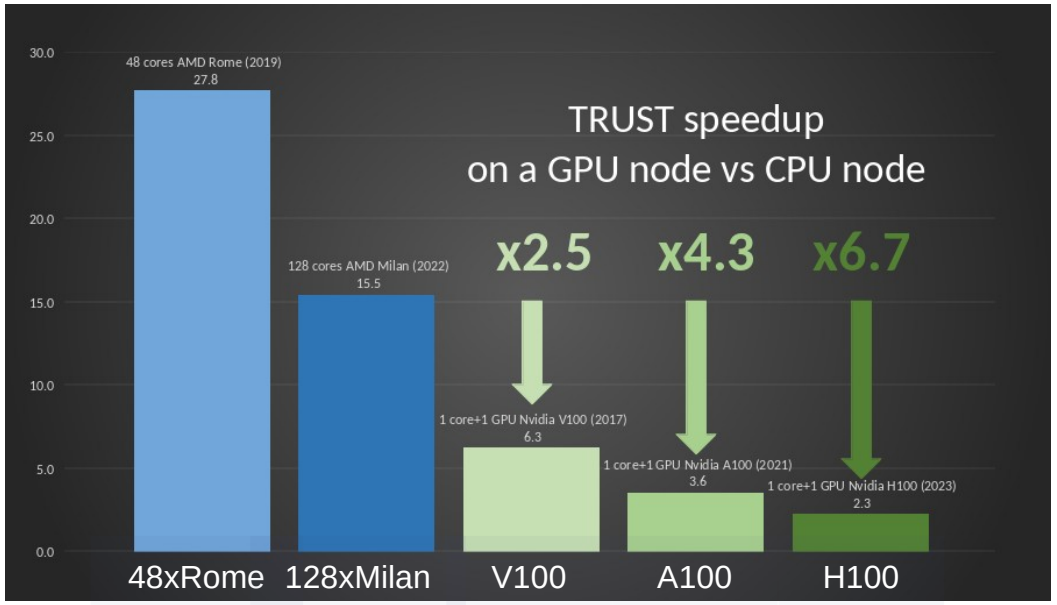
- Correctness if **CPU** modules still used
- Full performance only if **GPU** modules exclusively used (typical speed-up: x4 GPU vs CPU node)

2 GPU performance & solvers

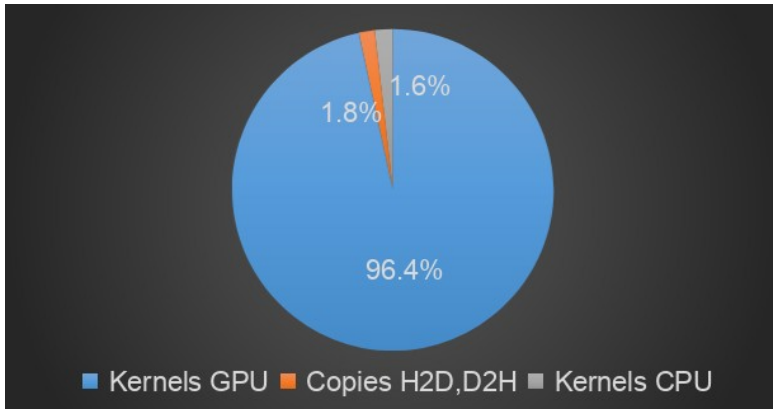
TRUST performance on GPU node

- 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

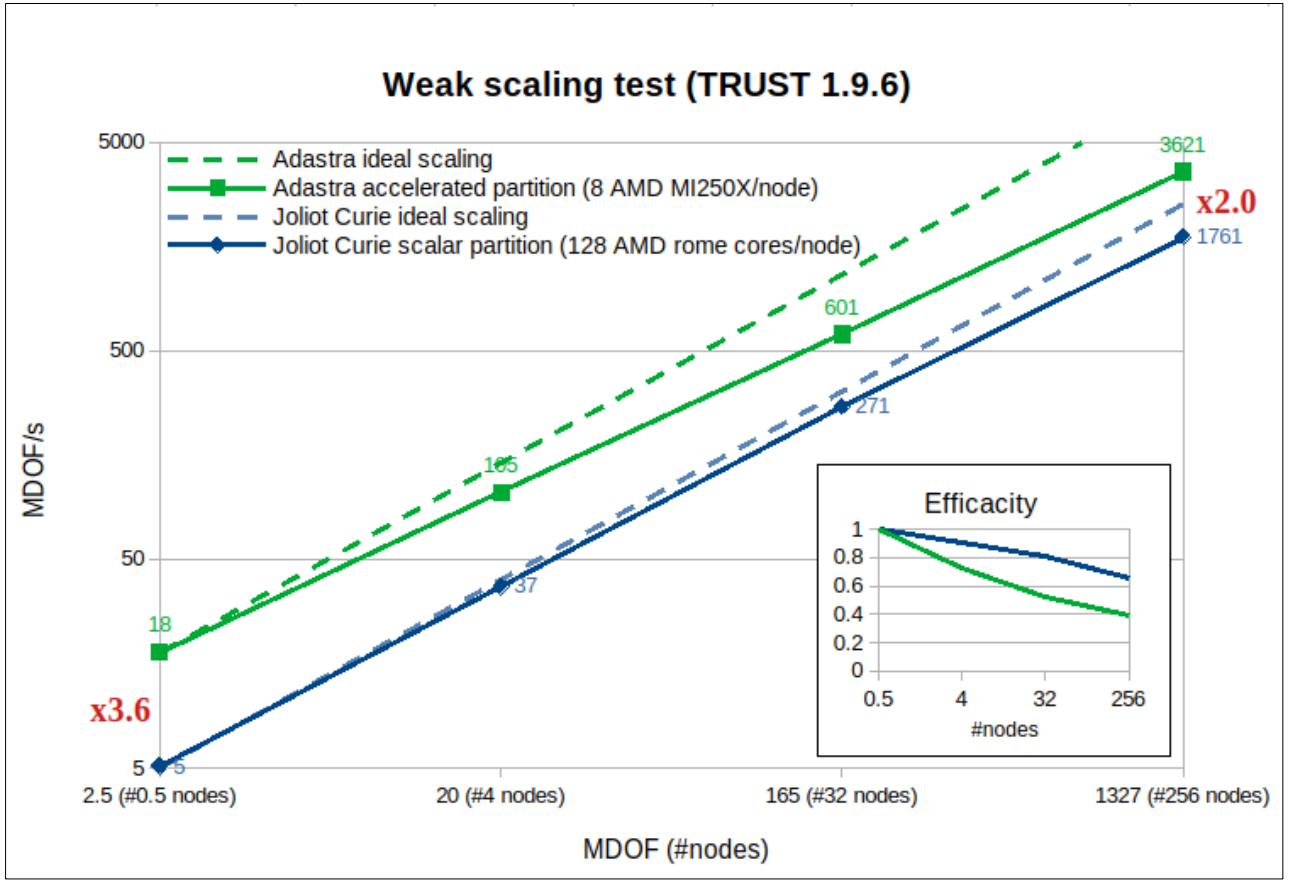




- **GPU** nodes (V100, A100, H100) compared to **CPU** nodes (48 AMD Rome & 128 AMD Milan cores)
- TRUST speedup ~4 is typical when all modules used run on **GPU**



TRUST scalability on multi-GPU



- Degraded speedup, reasons:
  - Algebric multigrid in linear solvers (AmgX or Hypre) : lower convergence rate of GPU versions compared to CPU ones
  - MPI GPU-Aware enabled in TRUST, seems not beneficial in AmgX or PETSc/Hypre linear solvers

Linear solvers used

Compute time

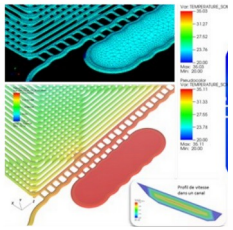
Days

Hours

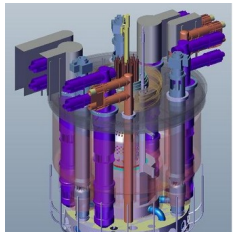
Minutes

PETSc LU (MUMPS)
PETSc LU
PETSc UmfPack (SuiteSparse)
cuDSS

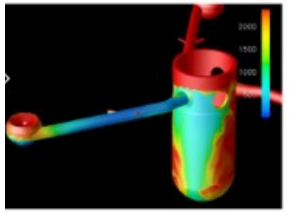
Direct linear solvers



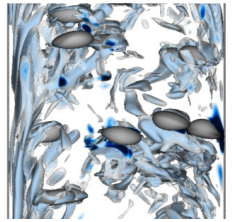
Fuel Cells



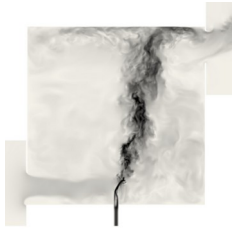
TrioMC



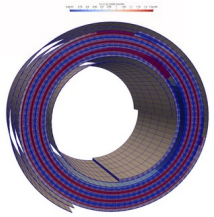
TrioCFD



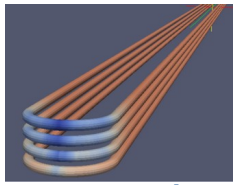
TrioIJK



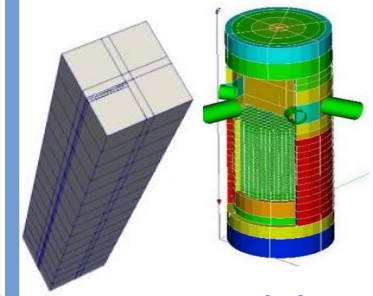
TRUST



Batteries



Genepi+



FLICA5 Module3D

Iterative linear solvers

PETSc solvers:

CG+BoomerAMG (Hypre)
CG+SSOR
GMRES+Jacobi

CG+AMG (AmgX)
CG+BoomerAMG (Hypre)
GMRES+Jacobi

TRUST solvers:

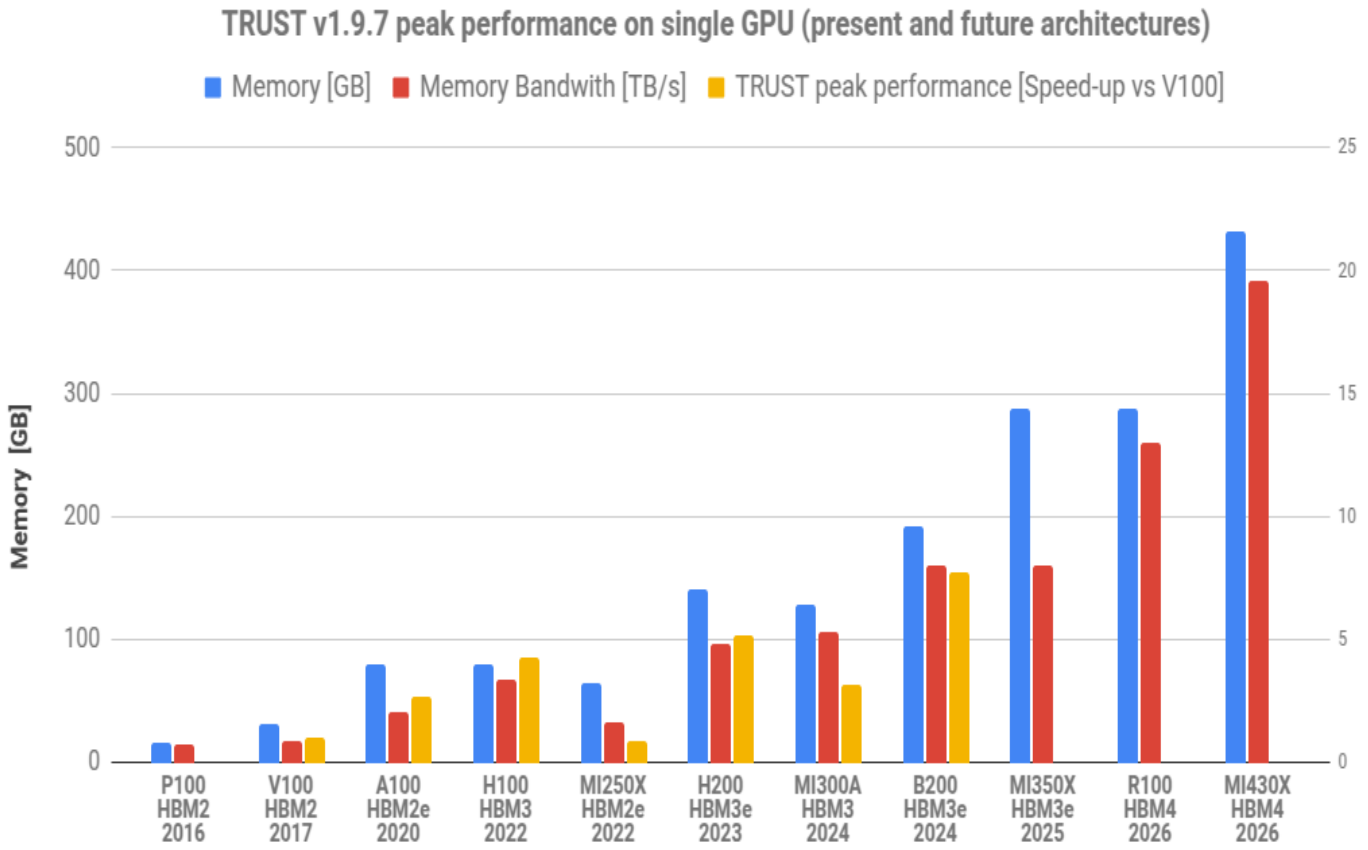
CG+SSOR
GMRES+Jacobi
GAMG

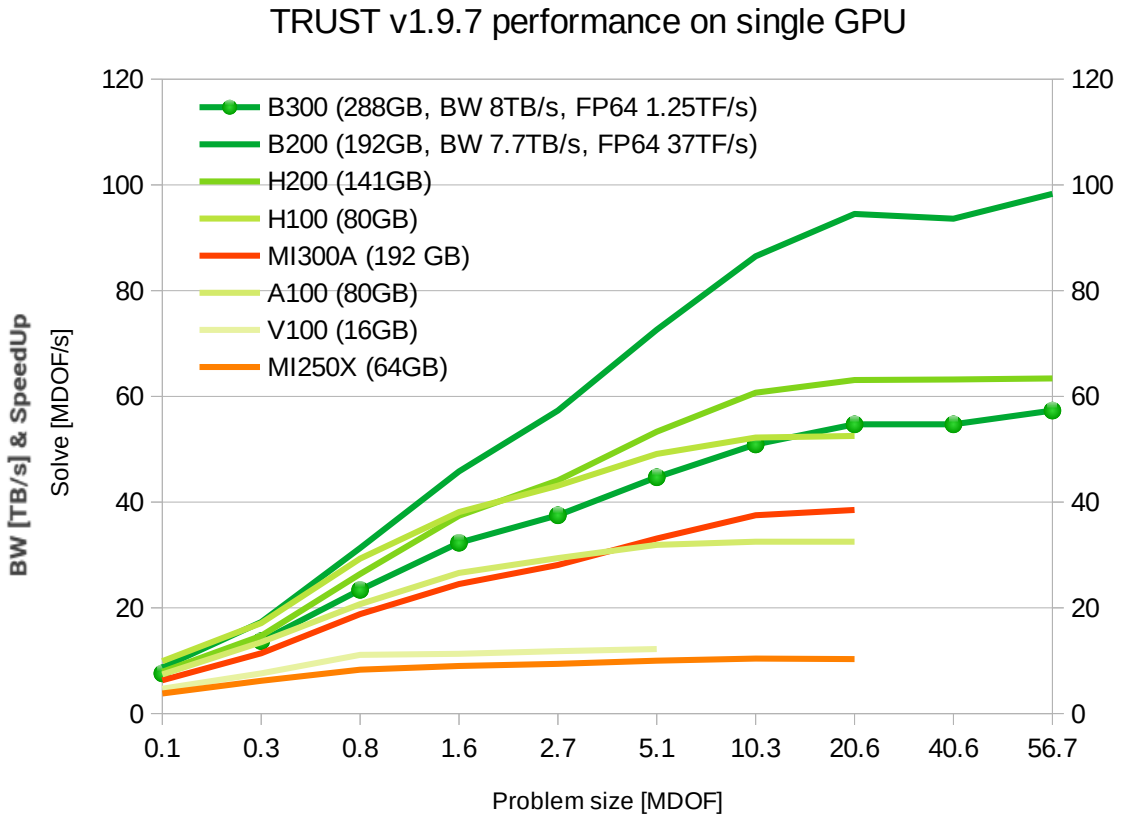
GMRES+Jacobi

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

Problem size (Cells)

Benchmarking GPUs





- GPU **memory** increase trend -> less MPI ranks used
- TRUST **peak performance** highly correlated with GPU **memory bandwith**
  - TRUST performance on **AMD** not in par compared to Nvidia yet
- AI cloud provider: **helpful** to benchmark latest device (e.g. B200/B300)

3 Benchmarking

Preparing for Alice Recoque : stress test #1

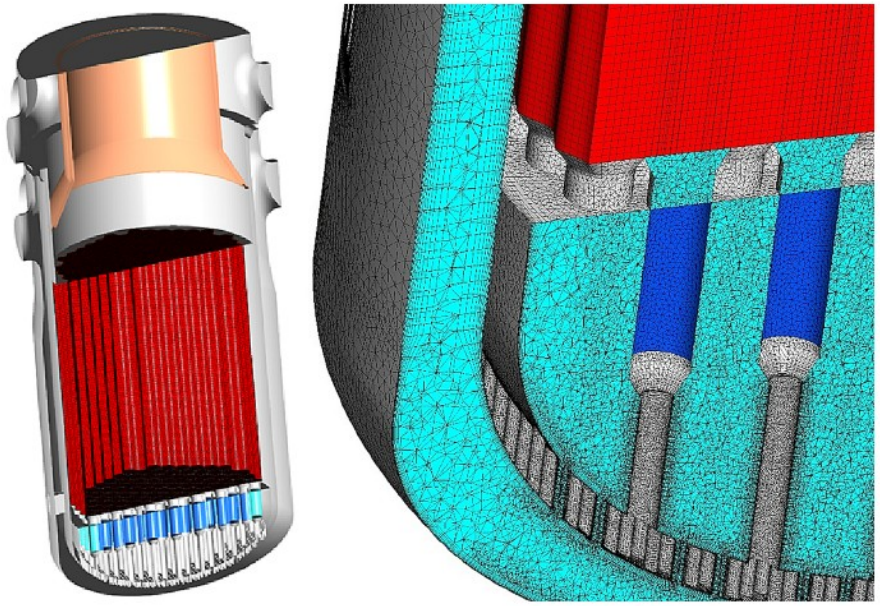


Fig. 5. The model by KIT with meshing detail in the lower plenum.

TrioCFD simulation (VVER-1000):

- Mesh: **2.6B tetras**
- Model: Incompressible flow + RANS
- Solver: GC + SSOR

  2016: 12000 Intel Xeon Cores (solver divergence)

- Solver: GC + AMG per block (Hypre/Boomeramg)

  2025: 128 Nvidia A100 GPUs (solver convergence ~250it/dt)

Preparing for Alice Recoque : stress test #1

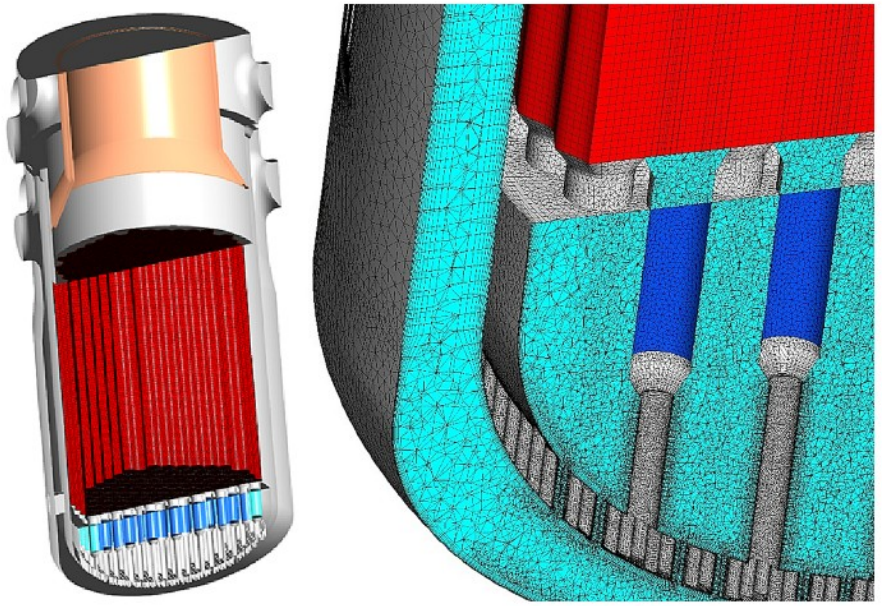


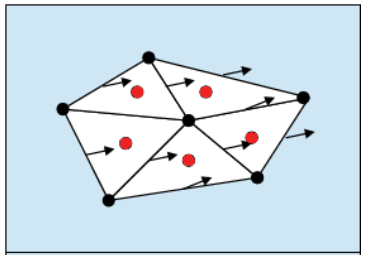
Fig. 5. The model by KIT with meshing detail in the lower plenum.

TrioCFD simulation (VVER-1000):

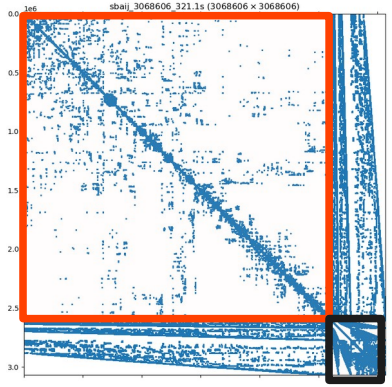
- Mesh: **2.6B tetras**
- Model: Incompressible flow + RANS
- Solver: GC + SSOR
  2016: 12000 Intel Xeon Cores (solver divergence)

- Solver: GC + AMG per block (Hypre/Boomeramg)
  2025: 128 Nvidia A100 GPUs (solver convergence ~250it/dt)

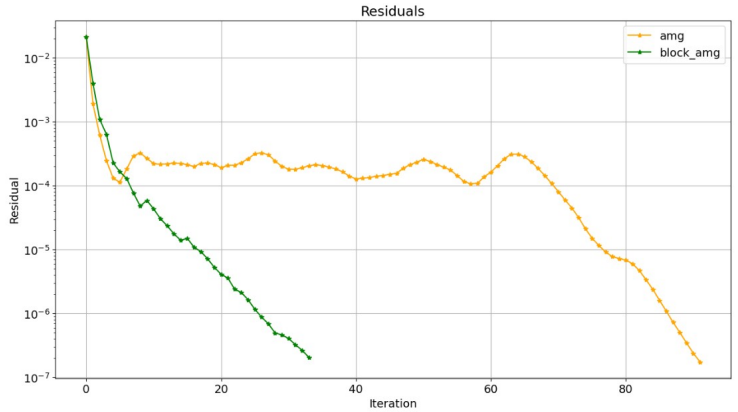
- Renovated multigrid solver. Efficient numerical methods matter!
  - Block preconditioner (PCFieldpslit PETSc)



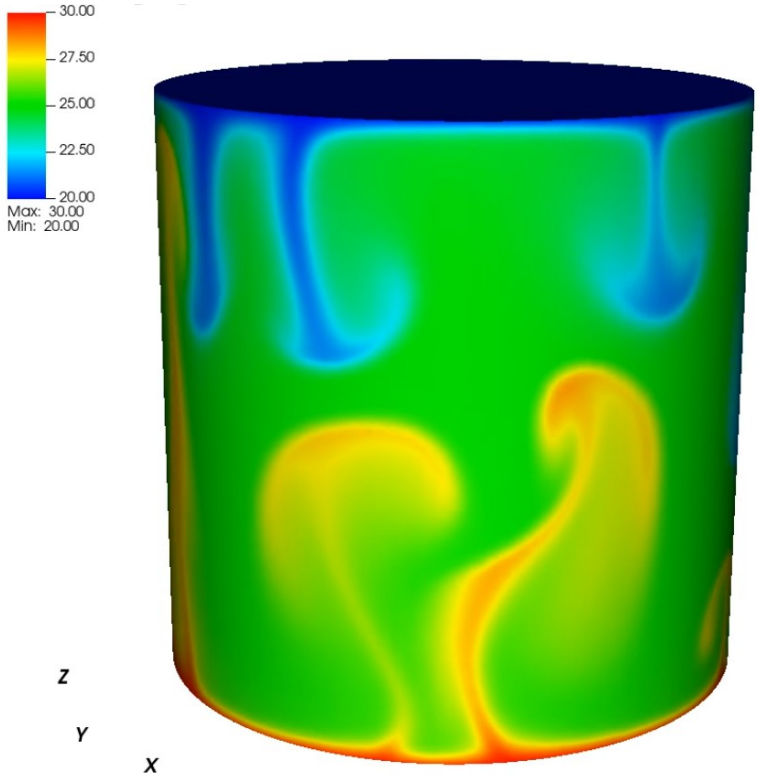
$$A = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix}$$



$$P^{-1} = \begin{pmatrix} A_{00} & 0 \\ 0 & A_{11} \end{pmatrix}^{-1}$$



Preparing for Alice Recoque : stress test #2



Rayleigh Benard simulation (Ra 1e7):

- Direct numerical simulation (DNS)
- Model: Incompressible flow + Boussinesq source term
- Scheme: Muscl + explicit Euler
- Pressure solver: GC + Boomeramg

Mesh **1.8M cells**: 1 Nvidia H100 (0.08s/dt)

Currently evaluated:

Mesh **7.7B cells:** 2024 AMD MI250X (2.8s/dt)

I/O (PDI+pHDF5, checkpoint/restart): validated

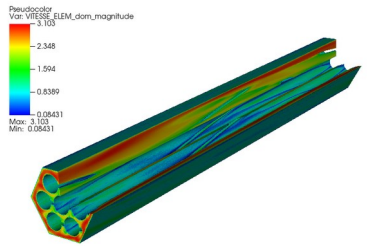
I/O (CGNS, results): validated

In-situ visualization mandatory !

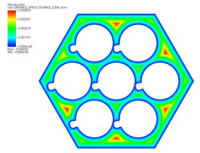
Preparing for Alice Recoque : stress test #3

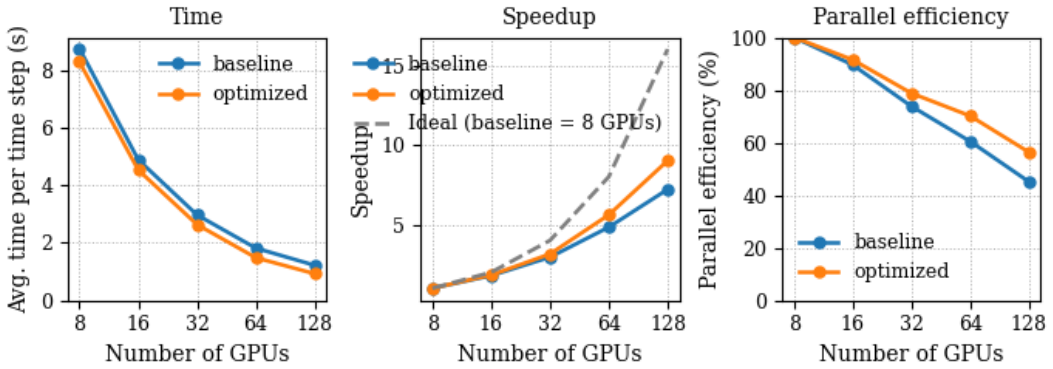
Flow simulation (LMFR) with TrioCFD on Lumi

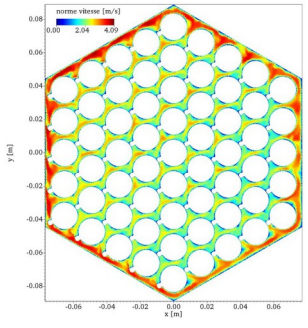
- Model: Low Reynolds RANS k-ω
- Scheme: Muscl + Euler implicit (GMRES/Jacobi)
- Pressure solver: GC (PETSc) + Boomeramg (Hypre)



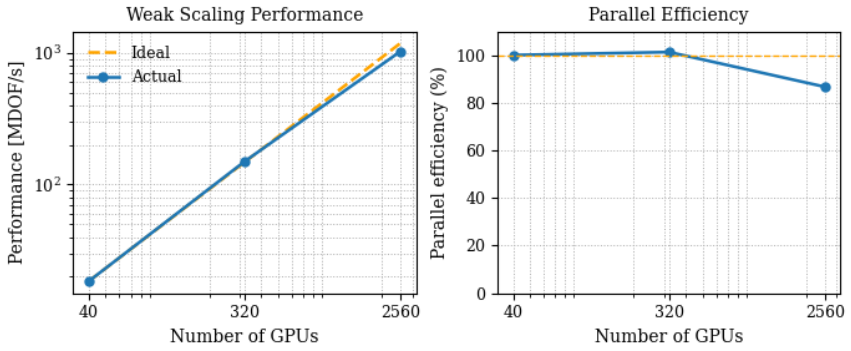
- **Strong scaling test**
  - 7 pins configuration, 32MDOF
  - Reduced MPI communication
  - Distances to wall computed by ArborX:



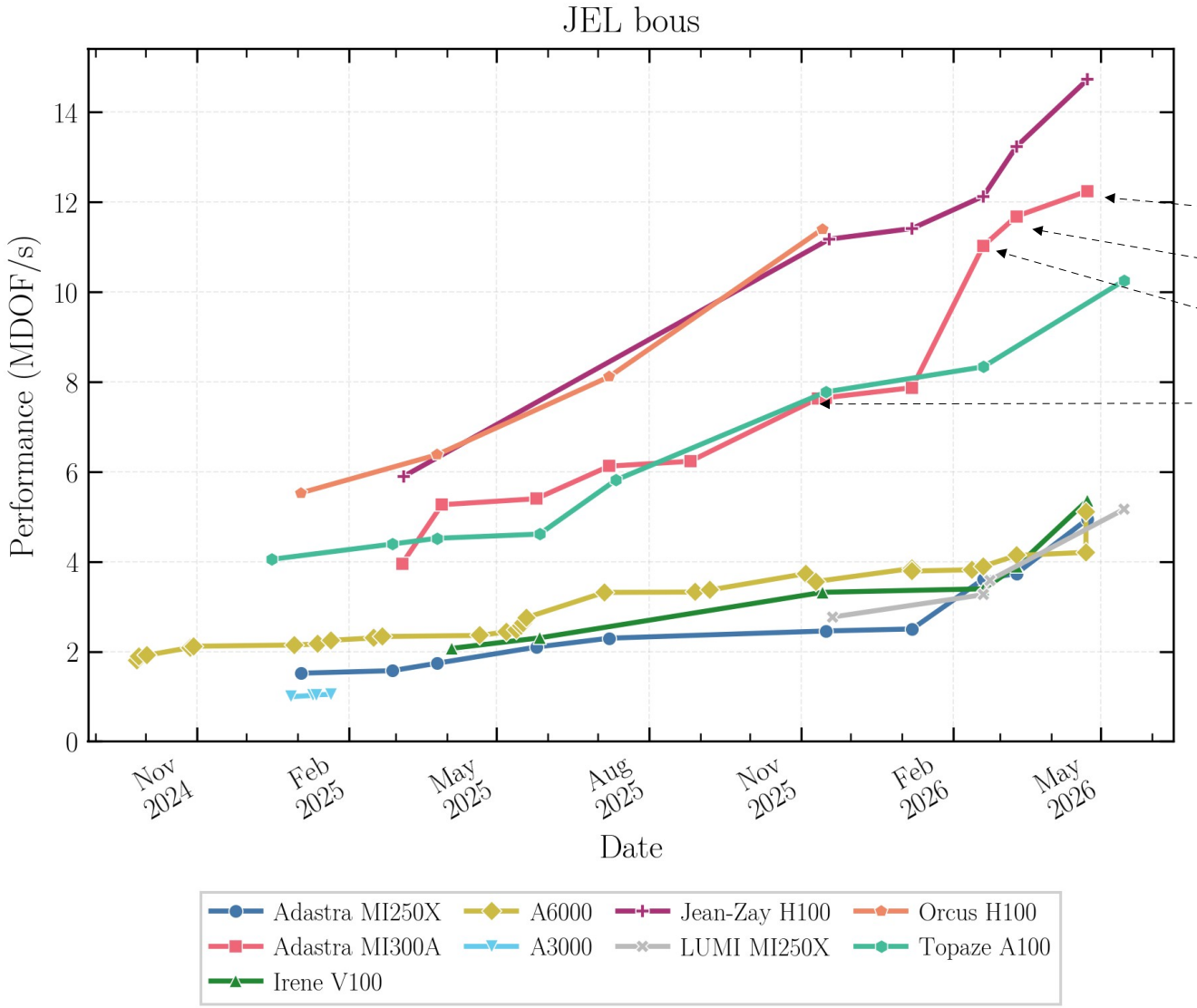




- **Weak scaling test**
  - 61 pins configuration, 2.7MDOF/GPU
  - ArborX (Kokkos based geometric library) to accelerate nearest point algorithm for k-ω model (distance to wall)
  - Up to 18 BDOF for now…



Continous integration testing



- Performing non regression checks help tracking optimizations over time
- E.g. AMD MI300A (speed-up x3):
  - Mesh reordering (Hilbert, SFC)
  - Kokkos 5.1 update (MDRangePolicy opt)
  - ROCm update (6.x -> 6.4.3)
  - Convection kernel optimization
- Future gains:

-Optimal memory access for MD array (LeftLayout view, WIP)

-Mixed precision (linear solver, kernels ?)

4 Roadmap

TRUST roadmap for Alice Recoque

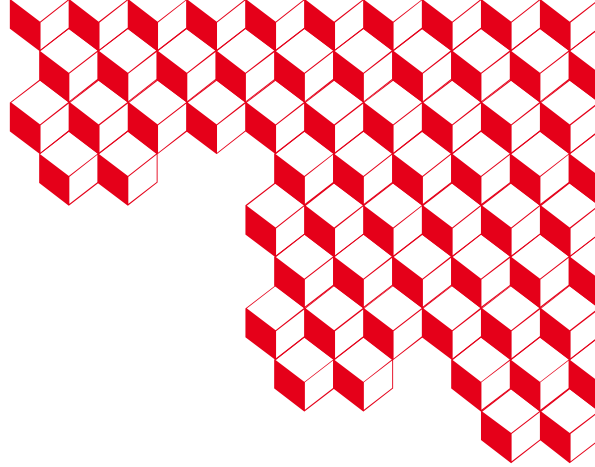
- **Our concerns**
  - Host Memory (1TB) < Device Memory (1.7TB)
  - Friendly profiler tools
  - ROCm evolutions (HPC libraries should follow)
  - Early access to MI430X

Technical roadmap for our code

- Parts of code now needs **long** instead of int to benefit from the huge device memory
- **Port** loops with Kokkos on un-accelerated part of the code
- Optimizations
  - **Reduce** host memory footprint (Host alloc->Device copy->Host dealloc strategy or Device alloc strategy ?)
  - **Reduce** device memory footprint of some of our algos (2-12KB/cell) with matrix-free strategy ?
  - Kernels **tuned** for MI300A, MI355X, MI430X AMD chips
  - **Reduced** p2p MPI communications (overlap computation with communication? NCCL/RCCL? reduce ghost cells ?)
  - **Coalescent** memory access on multiD array (LeftLayout Kokkos views)
  - **Reordered** mesh items (Hilbert/Morton algorithm)

TRUST roadmap

- **Other current projects**
  - Suitable numerical methods for GPU (Discontinous Galerkin, WIP)
  - Modern delivery strategy (cmake for whole build + spack for deployment, WIP)
  - Visualization in-situ for Exascale simulations (Catalyst, Ascent ?)
  - More linear solver availables (Ginkgo, Muelu from Trilinos ?)
  - GPU usage for all TRUST applications...
  - ...



Thanks. Any questions ?