EXCELLERAT P2

CFD Simulation of Steam Axial Turbine Stage on GPU

E. Fadiga, S. Bnà, G. Giaquinto, T. Zanelli - CINECA

P. Bottin, A. Dalla Rosa – De Pretto Industrie

This offering is not approved or endorsed by OpenCFD Limited, producer and distributor of the OpenFOAM software and owner of the OPENFOAM® and OpenCFD® trade marks.

Agenda

- Introduction and motivation
- **SPUMA**: a portable and non-invasive **GPU porting** of OpenFOAM
- Simulating a steam turbine stage on GPU: **missing features**
- Performance on an **industrial test case**
- Conclusions

Introduction

This presentation focuses on **compressible turbomachinery OpenFOAM** simulations, executed on both **CPUs and GPUs**, unlocking the full potential of **modern heterogeneous HPC** systems.

Turbines simulations are characterized by several complexities:

- Stator-rotor interaction
- Compressible high-speed flows
- Thermophysical modelling of steam

Images Copyright: 2025 De Pretto Industrie S.r.l.

Motivation

- OpenFOAM is widely regarded as the **most used open-source CFD** software.
- Specific **implementations tailored to turbomachinery** are often available only as third-party separate contributions.
- **Experienced developers are needed** to integrate these contributions and add new implementations.
- CINECA is currently putting a consistent effort in **OpenFOAM GPU porting activities**.

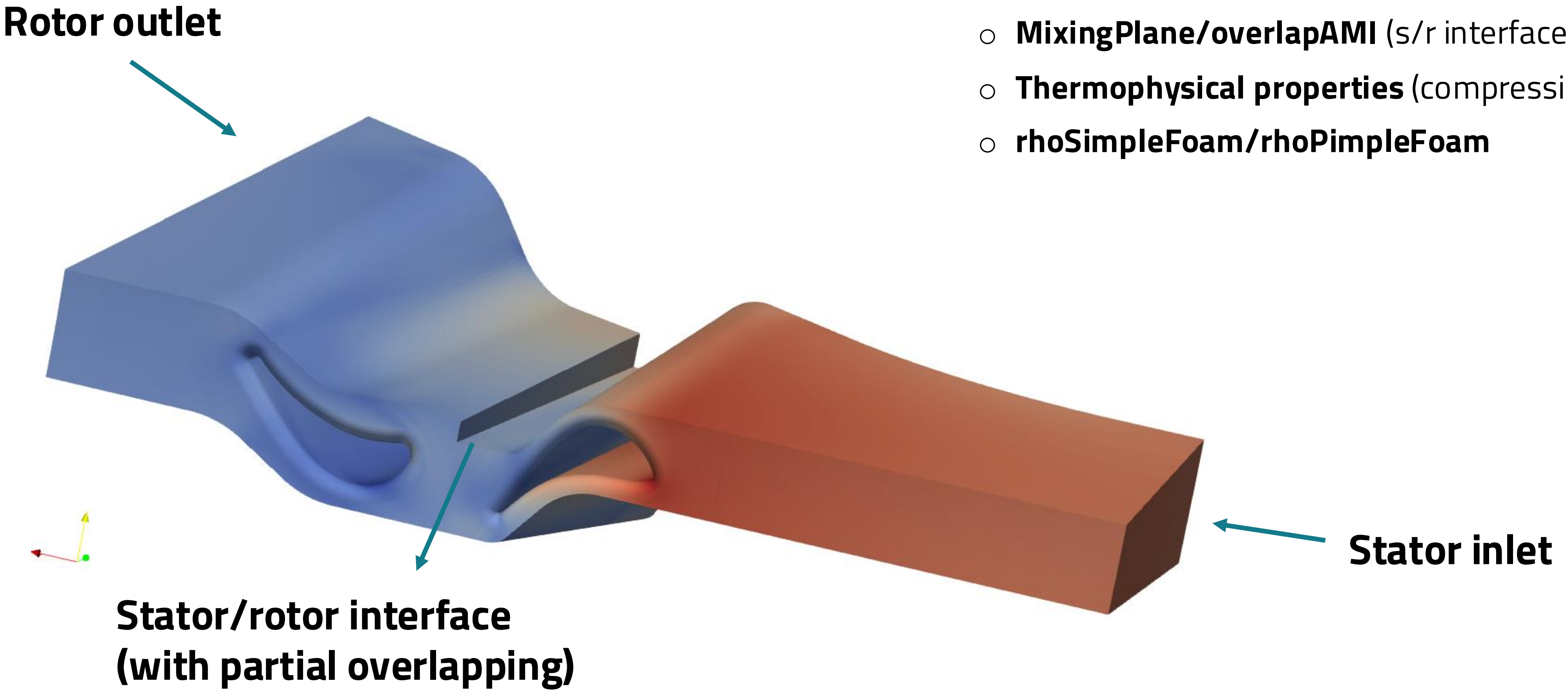
CINECA and De Pretto Industrie are collaborating to simulate turbines on CPUs

→

CINECA is porting the necessary features on GPU to execute a turbine stage pioneering analysis

Turbine stage analysis

- Simulation features:
  - **MRF** (rotor domain)
  - **MixingPlane/overlapAMI** (s/r interface)
  - **Thermophysical properties** (compressible cases)
  - **rhoSimpleFoam/rhoPimpleFoam**



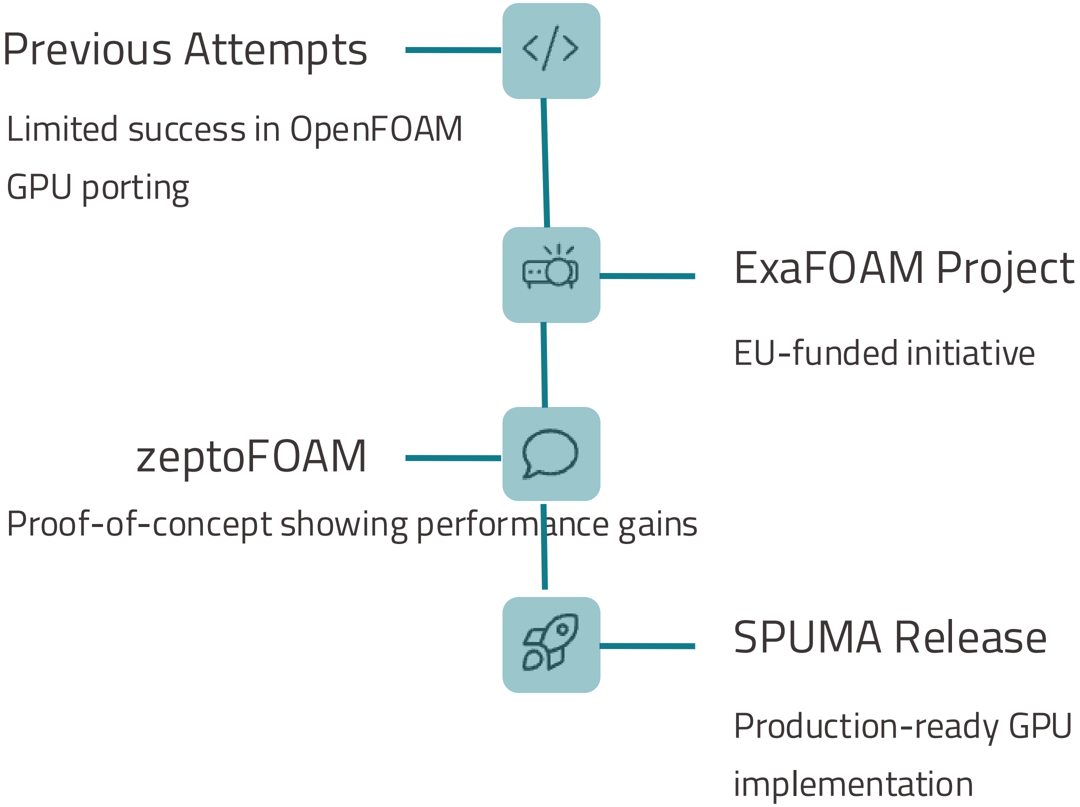
CPU runs – current status

- **Final goal** -> simulating the last stages of an axial steam turbine in wet operating conditions.
- **Main third party features:**
  - **Mixing Plane** from the turboWG (credits to Hakan Nillson, Martin Beaudoin & foamExtend devs)
    https://sourceforge.net/p/turbowg/mixingPlane/ci/master/tree/
  - **OverlapAMI** from ICSFoam (credits to Stefano Oliani & foamExtend devs)
    https://github.com/stefanoOliani/ICSFoam
  - **wetSteamFoam** (credits to Jiří Fürst et al.)
    https://github.com/furstj/wetSteamFoam/tree/main
- R&D project ongoing -> **refactoring, bug-fixing and integration** of those libraries already led to satisfying results.

Agenda

- Introduction and motivation
- **SPUMA**: a portable and non-invasive **GPU porting** of OpenFOAM
- Simulating a steam turbine stage on GPU: **missing features**
- Performance on an **industrial test case**
- Conclusions

SPUMA: GPU-Ready Solution



Previous Attempts

Limited success in OpenFOAM GPU porting

ExaFOAM Project

EU-funded initiative

zeptoFOAM

Proof-of-concept showing performance gains

SPUMA Release

Production-ready GPU implementation

Minimal Code Impact

Avoid radical changes to maintain community acceptance

Efficient Memory Management

Use memory pools for GPU optimization

No External Dependencies

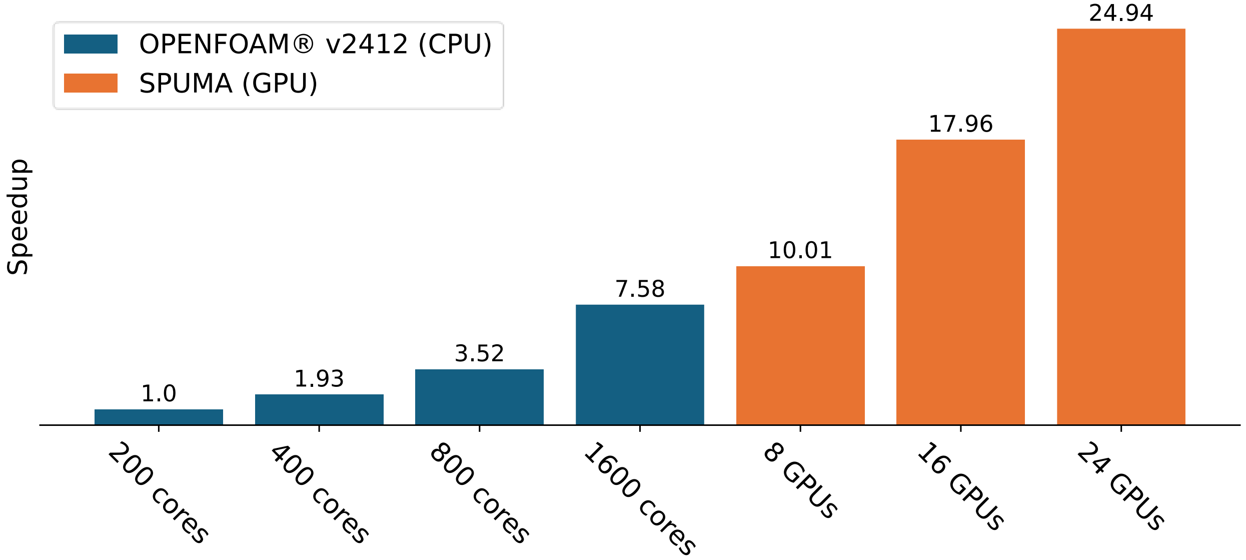
Third-party frameworks are not mandatory,

but possible (e.g., AmgX compatibility)

Multi-Platform Compatibility

Vendor-agnostic approach for wider accessibility

SPUMA performance (incompressible)



DrivAer Test Case

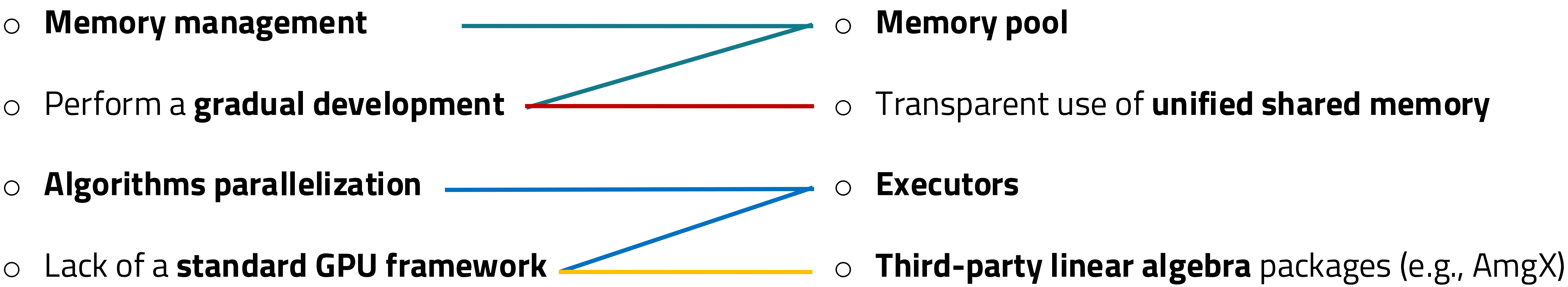
236 million cell automotive steady state simulation

1 GPU equals 220 CPU cores (1600 cores vs 24 GPUs)

This equivalence leads to **an 80% reduction of the energy** consumption

SPUMA: key features

- Major issues during a GPU porting:
- SPUMA key features



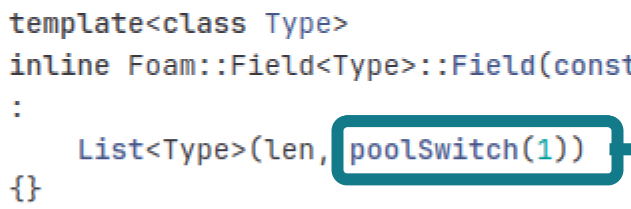
- **Memory management**
- Perform a **gradual development**
- **Algorithms parallelization**
- Lack of a **standard GPU framework**

- **Memory pool**
- Transparent use of **unified shared memory**
- **Executors**
- **Third-party linear algebra** packages (e.g., AmgX)

CINECA laid the foundations of the GPU porting by **implementing these key features**, as well as most of the **operators needed during an OpenFOAM run** (e.g., operations between Fields, GeometricFields and so on)

SPUMA: memory pool

- Foam::**Field objects** are **automatically** allocated through our memory pool APIs.



```
template<class Type>
inline Foam::Field<Type>::Field(const label len)
:
    List<Type>(len, poolSwitch(1))
{}
```

Field class constructors automatically allocate data on the pool via poolSwitch

- Unified shared memory is handling CPU->GPU and GPU->CPU copies implicitly:
  - We can **spot unported regions** of the code by profiling **CPU and GPU page faults**
  - If there are too many page faults, a **relevant slowdown** is experienced, but **the simulation is not crashing!!**

- Some OpenFOAM features use **Lists combined with Fields in computing algorithms** -> those **Lists must be allocated explicitly on the memoryPool** to avoid sudden crash of the calculation.

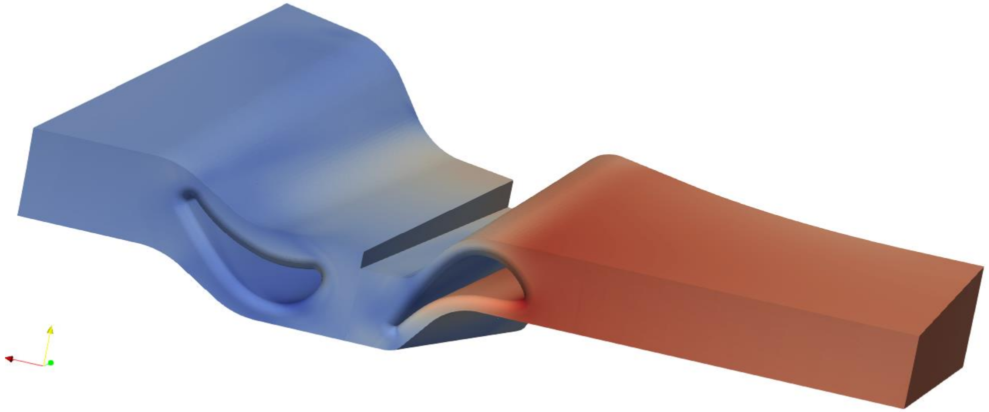
  - These crashes (due to invalid pointers) are <u>guiding us to spot the List</u> to be taken care of.

Agenda

- Introduction and motivation
- **SPUMA**: a portable and non-invasive **GPU porting** of OpenFOAM
- Simulating a steam turbine stage on GPU: **missing features**
- Performance on an **industrial test case**
- Conclusions

Our GPU test case

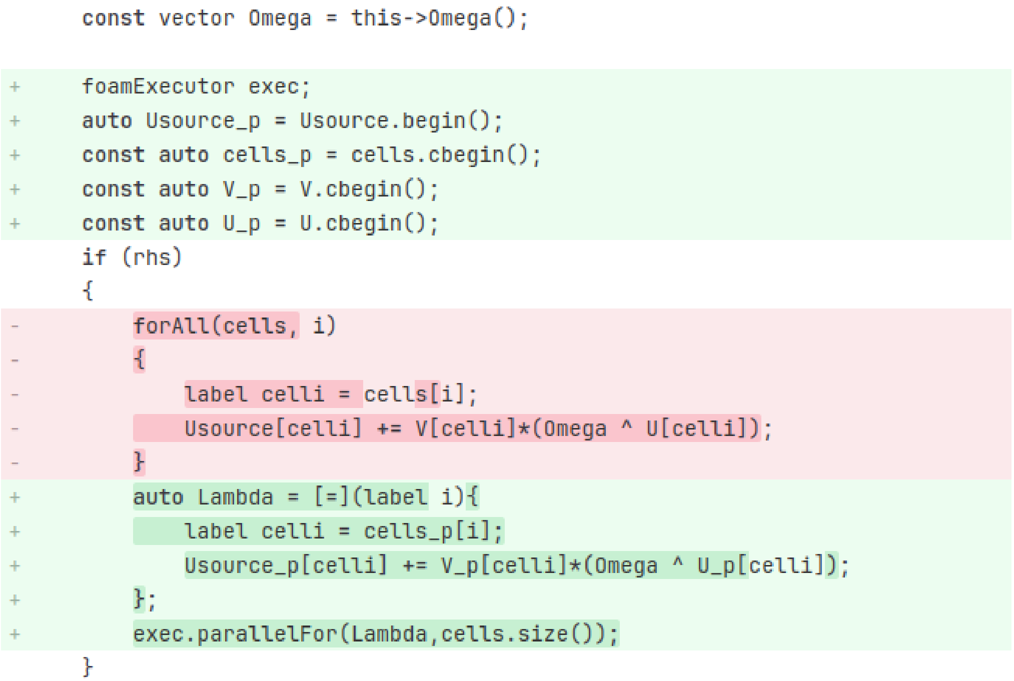
- Axial turbine – low pressure stage exploiting rotational periodicity. **Real operating conditions** (high Re, transonic, fully turbulent)
- Finite volume, second order, mesh size -> **more than 20 million cells (for a single blade periodic segment).**
- Superheated steam operating conditions (approximation)
- Needs the following features to be ported on GPU:
  - **MRF**
  - **Thermophysical models**
  - **cyclicAMI**
  - **Mixing plane/ partially overlapping interfaces.**



MRF: porting GPU

The MRF classes contain a typical example of **how we can remove page faults and speed up** the calculation.

MRFZone::addCoriolis (example)

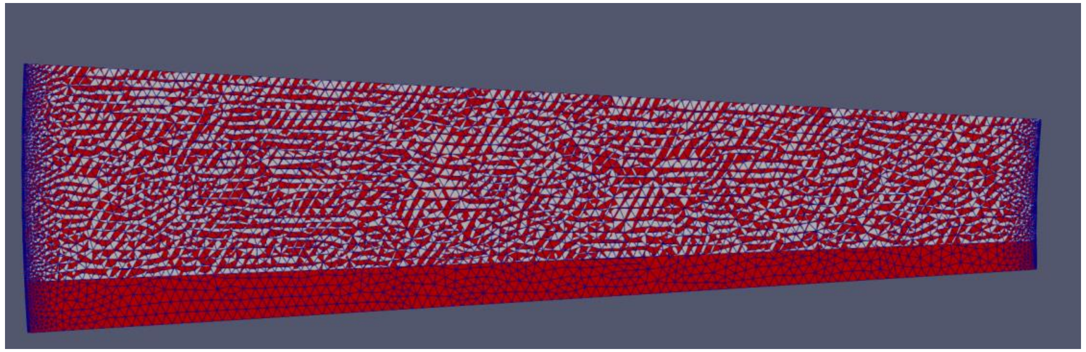


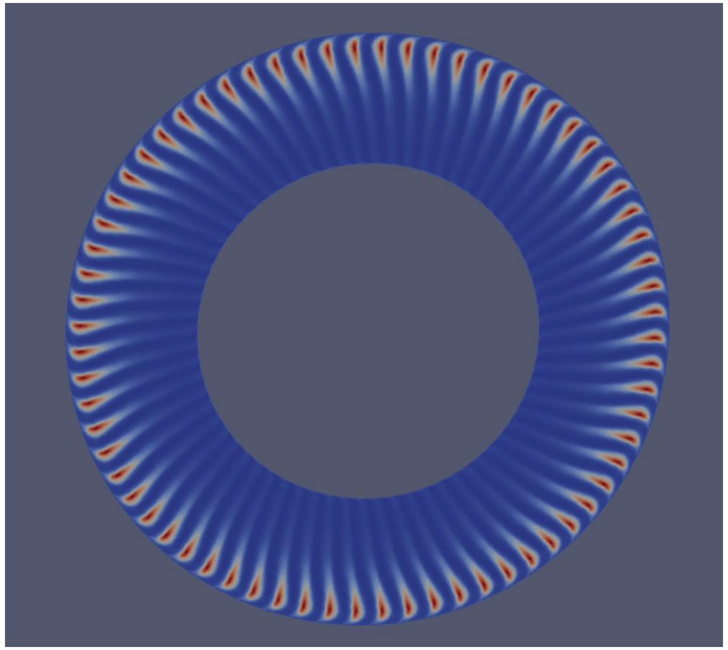
```
    const vector Omega = this->Omega();

+   foamExecutor exec;
+   auto Usource_p = Usource.begin();
+   const auto cells_p = cells.cbegin();
+   const auto V_p = V.cbegin();
+   const auto U_p = U.cbegin();
    if (rhs)
    {
-       forAll(cells, i)
-       {
-           label celli = cells[i];
-           Usource[celli] += V[celli]*(Omega ^ U[celli]);
-       }
+       auto Lambda = [=](label i){
+           label celli = cells_p[i];
+           Usource_p[celli] += V_p[celli]*(Omega ^ U_p[celli]);
+       };
+       exec.parallelFor(Lambda,cells.size());
    }
```

- Explicit **forAlls are originally executed** on the CPU, producing unwanted copies.
- We construct a **lambda expression**, copy pass all the pointers and ship the content to the SPUMA executor.
- In this case, the forAll corresponds to a **parallelFor** method of the executor.

overlapAMI: optimize the code while porting it

- OverlapAMI can be exploited to run **frozen rotor** simulations when **domains are partially overlapped.**
- Example:
  - Rotor -> 66 blades
  - Stator -> 51 blades

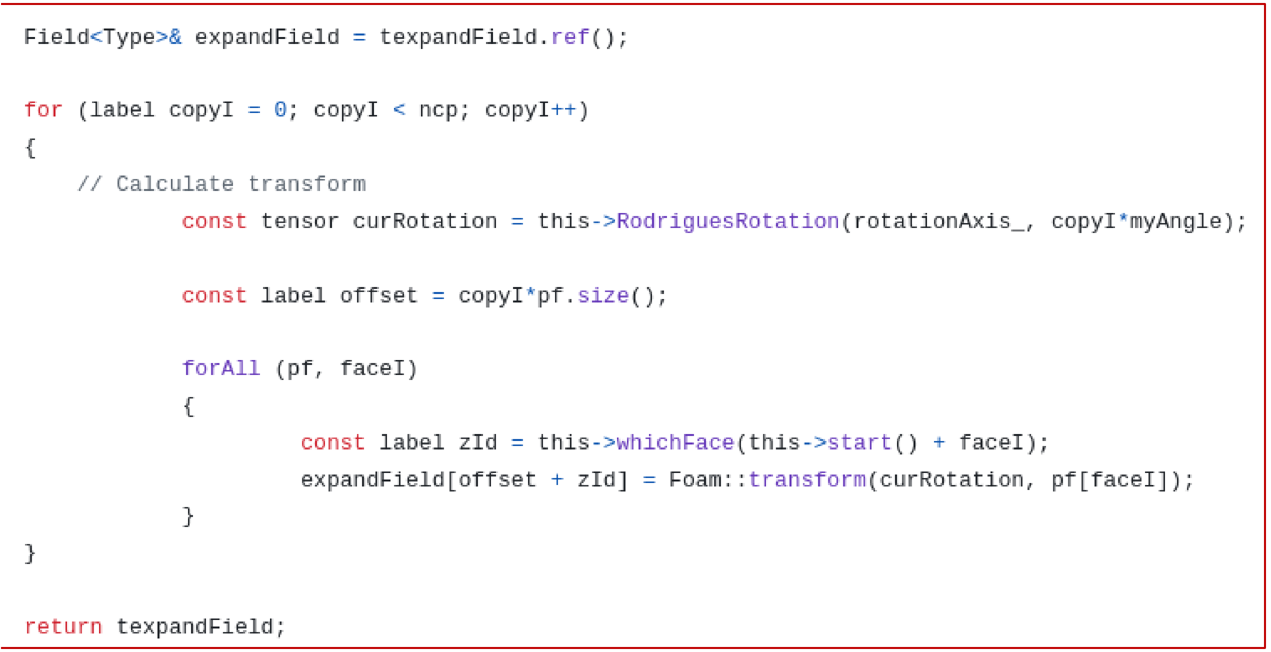




Both patches are expanded periodically to obtain a 360° patch, then the AMI interpolation is performed.

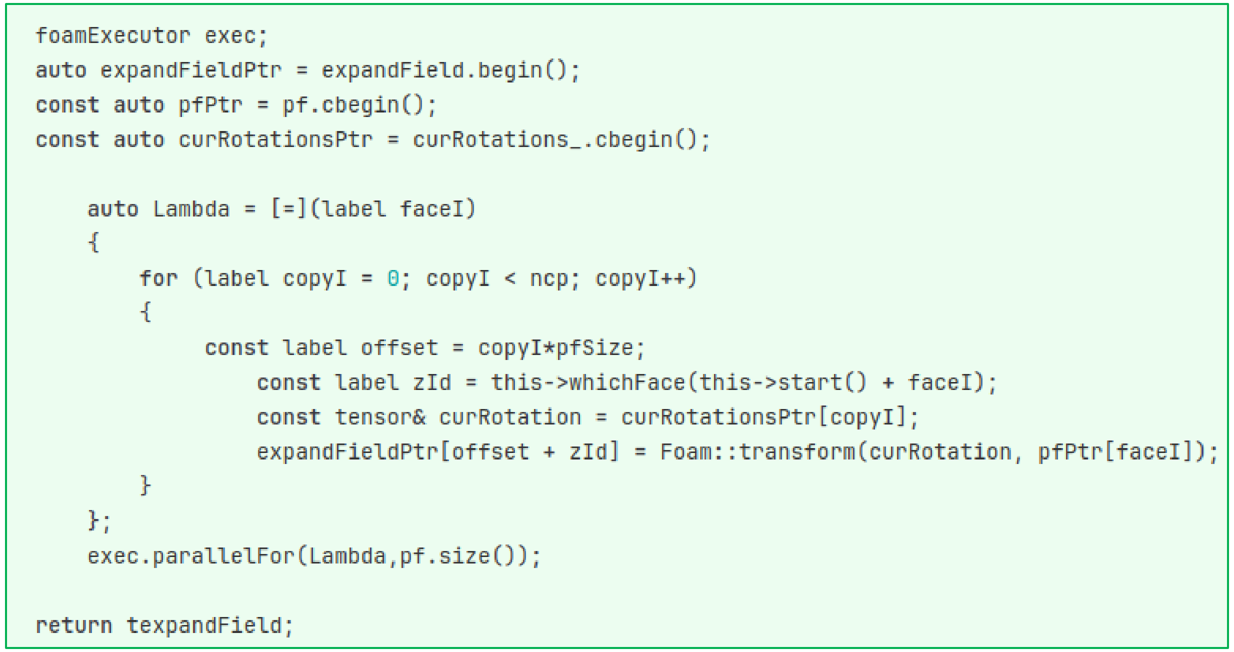
overlapAMI: optimize the code while porting it

overlapAMIPolyPatch::expandData



```
Field<Type>& expandField = texpandField.ref();

for (label copyI = 0; copyI < ncp; copyI++)
{
    // Calculate transform
            const tensor curRotation = this->RodriguesRotation(rotationAxis_, copyI*myAngle);

            const label offset = copyI*pf.size();

            forAll (pf, faceI)
            {
                    const label zId = this->whichFace(this->start() + faceI);
                    expandField[offset + zId] = Foam::transform(curRotation, pf[faceI]);
            }
}

return texpandField;
```

Original

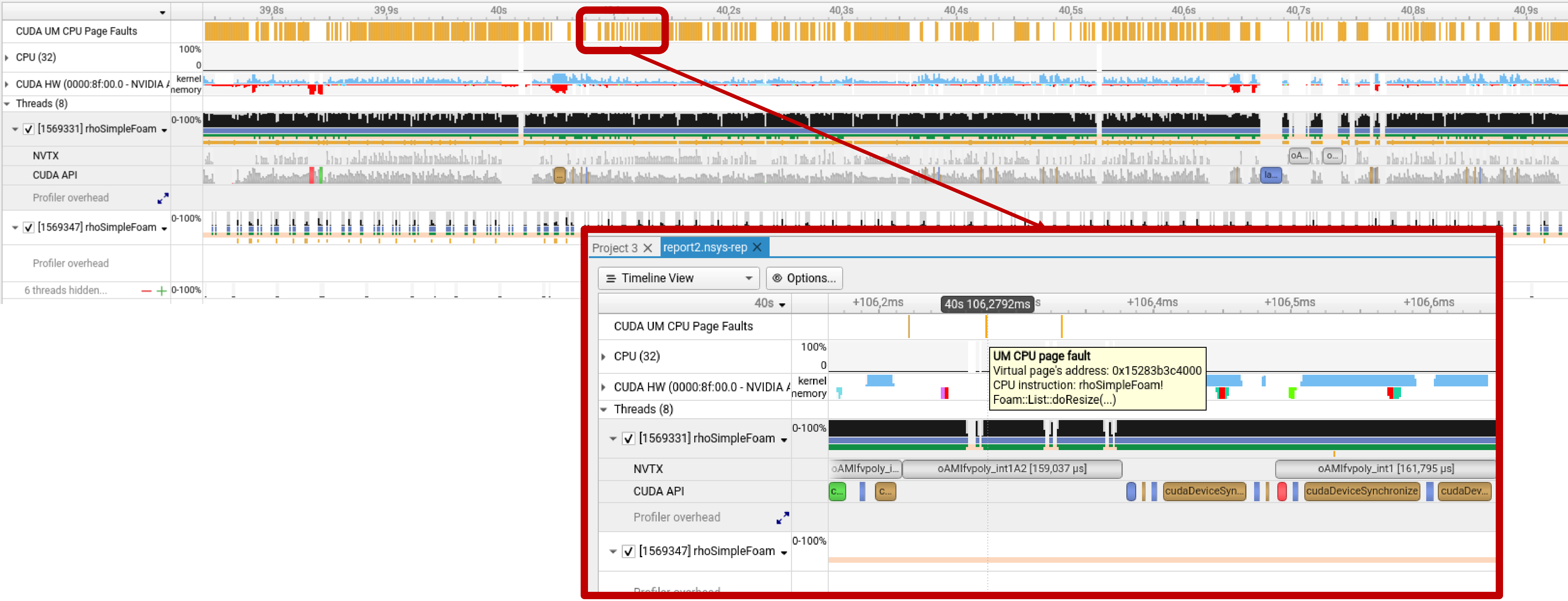


```
foamExecutor exec;
auto expandFieldPtr = expandField.begin();
const auto pfPtr = pf.cbegin();
const auto curRotationsPtr = curRotations_.cbegin();

    auto Lambda = [=](label faceI)
    {
        for (label copyI = 0; copyI < ncp; copyI++)
        {
             const label offset = copyI*pfSize;
                 const label zId = this->whichFace(this->start() + faceI);
                 const tensor& curRotation = curRotationsPtr[copyI];
                 expandFieldPtr[offset + zId] = Foam::transform(curRotation, pfPtr[faceI]);
        }
    };
    exec.parallelFor(Lambda,pf.size());

return texpandField;
```

GPU ported

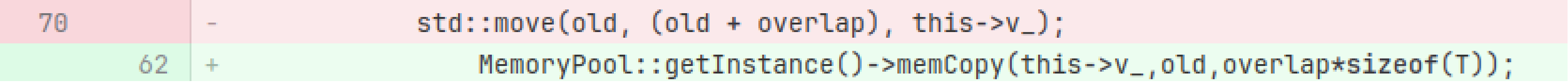
- The baseline kernel is again a **parallelFor.**
- We precalculate a rotation tensorField (curRotations_) and move the copyI loop inside the kernel, strongly **reducing the number of GPU kernels launched during the calculation**.

overlapAMI: optimize the code while porting it



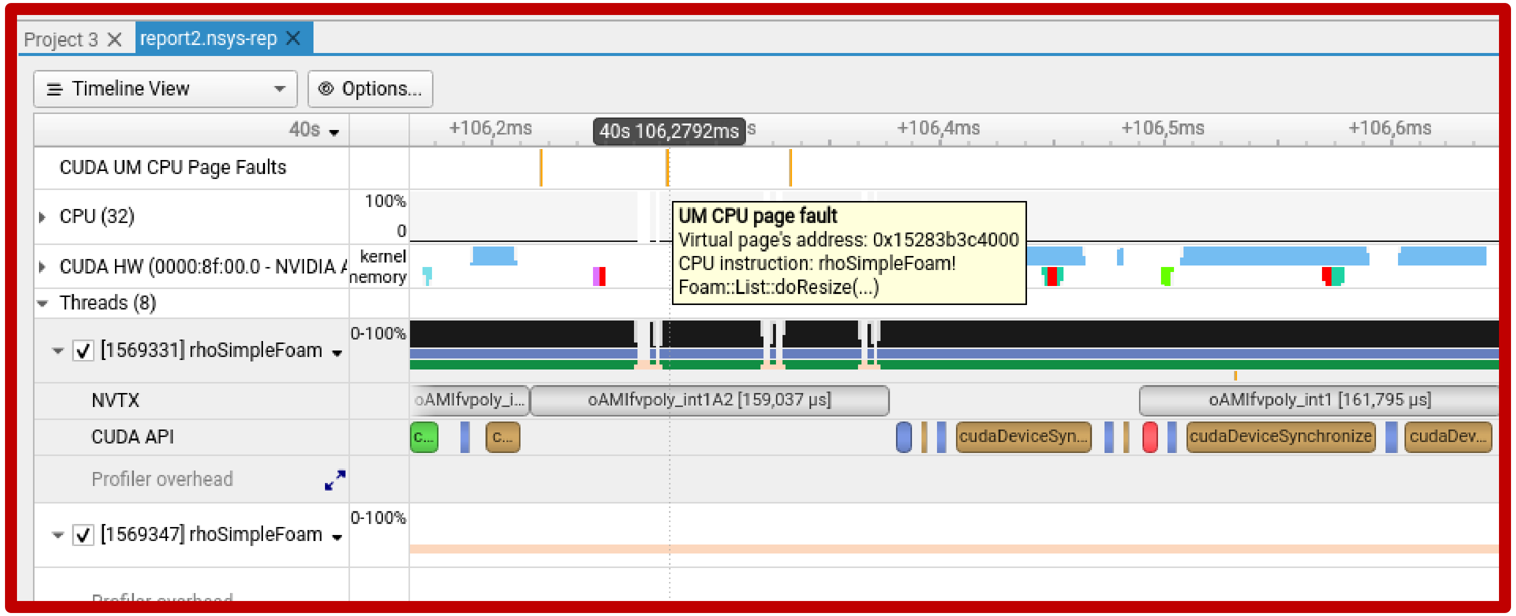
UM CPU page fault
Virtual page's address: 0x15283b3c4000
CPU instruction: rhoSimpleFoam!
Foam::List::doResize(...)

overlapAMI: optimize the code while porting it



```
-            std::move(old, (old + overlap), this->v_);
+                MemoryPool::getInstance()->memCopy(this->v_,old,overlap*sizeof(T));
```

List::doResize



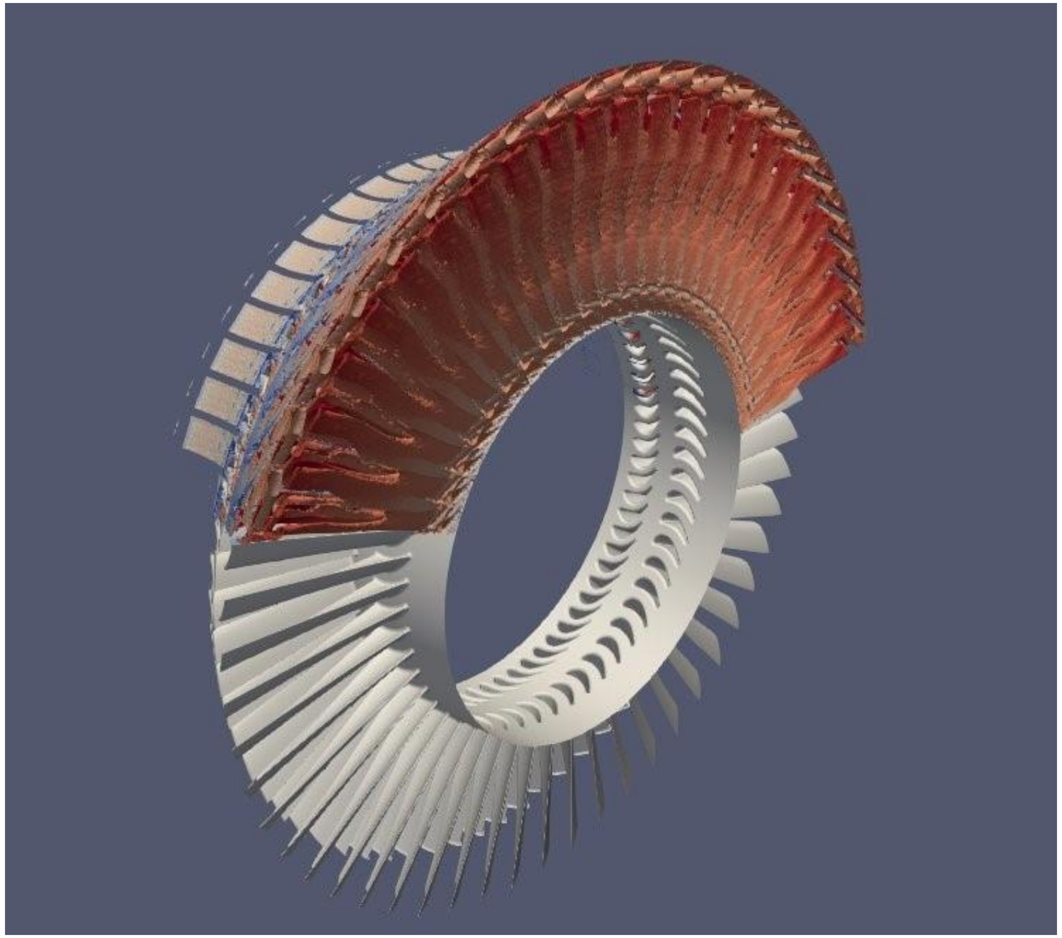
UM CPU page fault
Virtual page's address: 0x15283b3c4000
CPU instruction: rhoSimpleFoam!
Foam::List::doResize(...)

Agenda

- Introduction and motivation
- **SPUMA**: a portable and non-invasive **GPU porting** of OpenFOAM
- Simulating a steam turbine stage on GPU: **missing features**
- Performance on an **industrial test case**
- Conclusions

CPU runs

- Test case executed on both Galileo100 and Leonardo DCGP HPC systems.
- The simulations converge smoothly at the testing conditions provided by the industrial end-user.
- Steady RANS (Reynolds Averaged Navier Stokes) simulation wall time ranges from 5 to 8 hours on 2 nodes of G100
- 1 G100 node = 2 x CPU Intel CascadeLake 8260, with 24 cores each)



Performance comparison

EXCELLERAT P2

GPU run

- Time per iter = 2.07 s
- N GPUS = 1 NVIDIA A100

CPU run #0

- Time per iter = 8.625 s
- N cores = 48 (Leonardo DCGP)

CPU run #1

- Time per iter = 6.755 s
- N cores = 96 (Leonardo DCGP)

CPU run #2

- Time per iter = 6.135 s
- N cores = 192 (Leonardo DCGP)

Cores per GPU ratio

*timeCPU*coresCPU / timeGPU* nGPU*

Cores per GPU ratio (run #0) = 200

Cores per GPU ratio (run #1) = **313**

Cores per GPU ratio (run #2) = 569

Warning: the coresPerGPU indicator can be easily altered by variating the CPU conditions

Performance breakdown

GPU run

Step	**% of Total Time**
Simple loop	**100.00%**
Uassembly	**23.17%**
Upredictor	**1.23%**
Usolve	**7.75%**
Eassembly	**20.60%**
Esolve	**1.80%**
ThermoCorrect	**0.05%**
Passembly	**2.82%**
Psolve	**7.52%**
Turbo correct	**26.74%**

CPU run #1

Step	**% of Total Time**
Simple loop	**100.00%**
Uassembly	**25.00%**
Upredictor	**1.29%**
Usolve	**5.33%**
Eassembly	**20.48%**
Esolve	**1.49%**
ThermoCorrect	**0.09%**
Passembly	**2.34%**
Psolve	**3.78%**
Turbo correct	**31.37%**

Performance vs mixingPlane

EXCELLERAT P2

GPU run - overlapAMI

Step	% of Total Time
Simple loop	**100.00%**
Uassembly	**23.17%**
Upredictor	**1.23%**
Usolve	**7.75%**
Eassembly	**20.60%**
Esolve	**1.80%**
ThermoCorrect	**0.05%**
Passembly	**2.82%**
Psolve	**7.52%**
Turbo correct	**26.74%**

CPU run - mixingPlane

Step	% of Total Time
Simple loop	**100.00%**
Uassembly	**22.25%**
Upredictor	**1.23%**
Usolve	**6.09%**
Eassembly	**20.28%**
Esolve	**1.84%**
ThermoCorrect	**0.15%**
Passembly	**2.89%**
Psolve	**4.67%**
Turbo correct	**32.45%**

Cores per GPU ratio = **180**

Conclusions

- The compressible axial turbine test case **run successfully on CPU** after the required **software integration/development activities**.

- **SPUMA**, a recent development by CINECA, already supported most of the OpenFOAM features needed to simulate the test case.

- **GPU porting of missing features has been performed** working on:
  1. MRF
  2. Thermophysical models
  3. cyclicAMI and overlapAMI
  4. Velocity and temperature limiters

- The resulting **cores to GPU ratio (approx. 2-300)** is in line with results obtained in incompressible cases.

EXCELLERAT P2

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Germany, Italy, Slovenia, Spain, Sweden, and France under grant agreement No 101092621.

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European High Performance Computing Joint Undertaking (JU) and Germany, Italy, Slovenia, Spain, Sweden, and France. Neither the European Union nor the granting authority can be held responsible for them.

Co-funded by
the European Union

EuroHPC
Joint Undertaking

EXCELLERAT P2

Thank you for your attention!