CENTRE OF EXCELLENCE FOR HPC
ASTROPHYSICAL APPLICATIONS

gPLUTO LUMI Hackathon Status (Oslo May 2025)

S. Truzzi, A. Suriano University of Turin, Turin, Italy

A. Romeo, N. Shukla CINECA, Casalecchio di Reno Bologna, Italy

Co-funded by the European Union

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Belgium, Czech Republic, France, Germany, Greece, Italy, Norway, and Spain under grant agreement No 101093441.

EuroHPC Joint Undertaking

What is gPLUTO?

- GPU-enabled version of PLUTO: a multi-algorithm framework for solving the equations for gas and compressible plasma dynamics with high Mach numbers flows (i.e. compressible Navier-Stokes equation, ideal MHD, relativistic MHD (RMHD) and resistive relativstic MHD (ResRMHD)).
- It is a Godunov-type finite volume grid code solving hyperbolic and parabolic magneto-hydrodynamic conservation laws up to three dimensions on a static grid or mapped grids.
- Freely distributed PLUTO at *http://plutocode.ph.unito.it* (v. 4.4)
- Public gPLUTO at *https://gitlab.com/PLUTO-code/gPLUTO*
- gPLUTO is part of Scalable Parallel Astrophysical Codes for Exascale (https://www.space-coe.eu) European project

written in:

- ***C and C++ (core code)***
- *OpenACC for GPU shared memory*
- MPI for multiGPU / CPU support

Available Physics Modules (~60 % ported on GPU version)

Advection Physics (Hyperbolic PDE)
- Hydrodynamics (HD)
- Magnetohydrodynamics (MHD)
- Relativistic Hydrodynamics (RHD)
- Ideal and resistive relativistic MHD (RMHD - ResRMHD)

Source Terms
- Gravity / Body forces
- Cooling
- Heating / optically thin
- Chemical networks

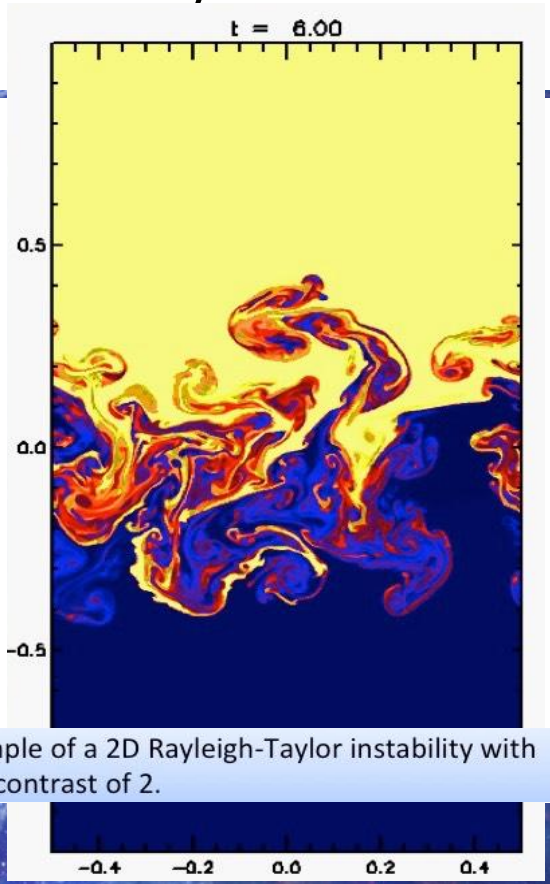
Geometry
- Cartesian
- Cylindrical
- Spherical

Dissipation Physics (Parabolic PDE)
- Viscosity (Navier-Stokes)
- Thermal conduction (hydro and MHD)
- Hall MHD, Ambipolar diffusion, Magnetic resistivity
- Radiation Hydrodynamics (FLD, 2 temp)

Particle Physics
- Lagrangian particles
- Cosmic Ray
- Dust

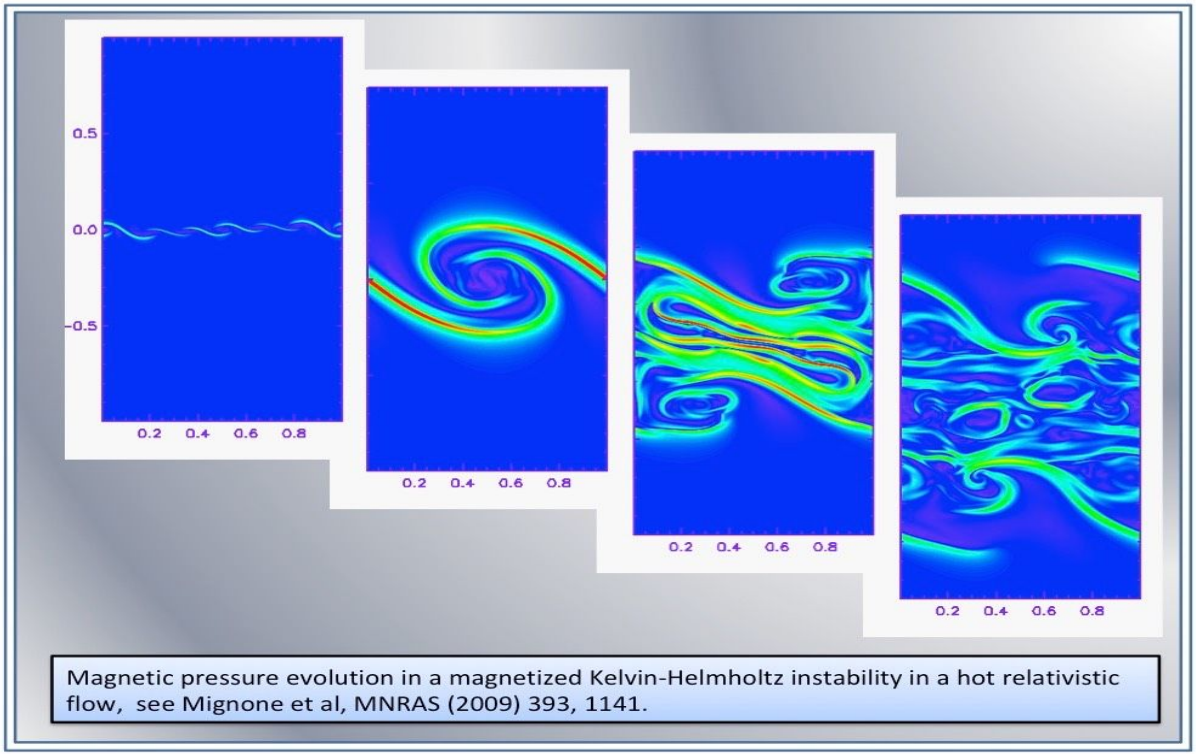
Thermodynamics
- Ideal
- Isothermal
- Non-Constant gamma
- Synge Gas (relativistic)

LEGEND
- Ported
- in progress
- Not ported



An example of a 2D Rayleigh-Taylor instability with density contrast of 2.

gPLUTO application Example

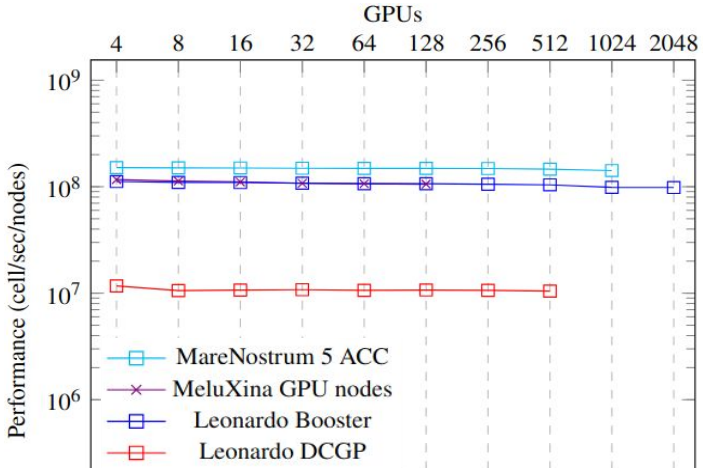


Magnetic pressure evolution in a magnetized Kelvin-Helmholtz instability in a hot relativistic flow, see Mignone et al, MNRAS (2009) 393, 1141.

gPLUTO Benchmark results

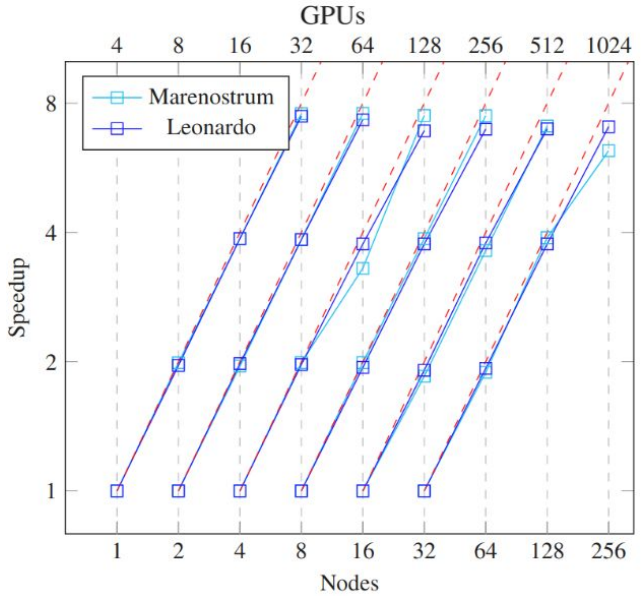
- Tested on EUROHPCs NVIDIA GPU HPCs (CINECA - Leonardo, BSC - Marenostrum)
- **speedup** factors (tCPU/tGPU) from **10 to 50** (depends on test)

3D MHD Orsag Tang <u>weak</u> scaling

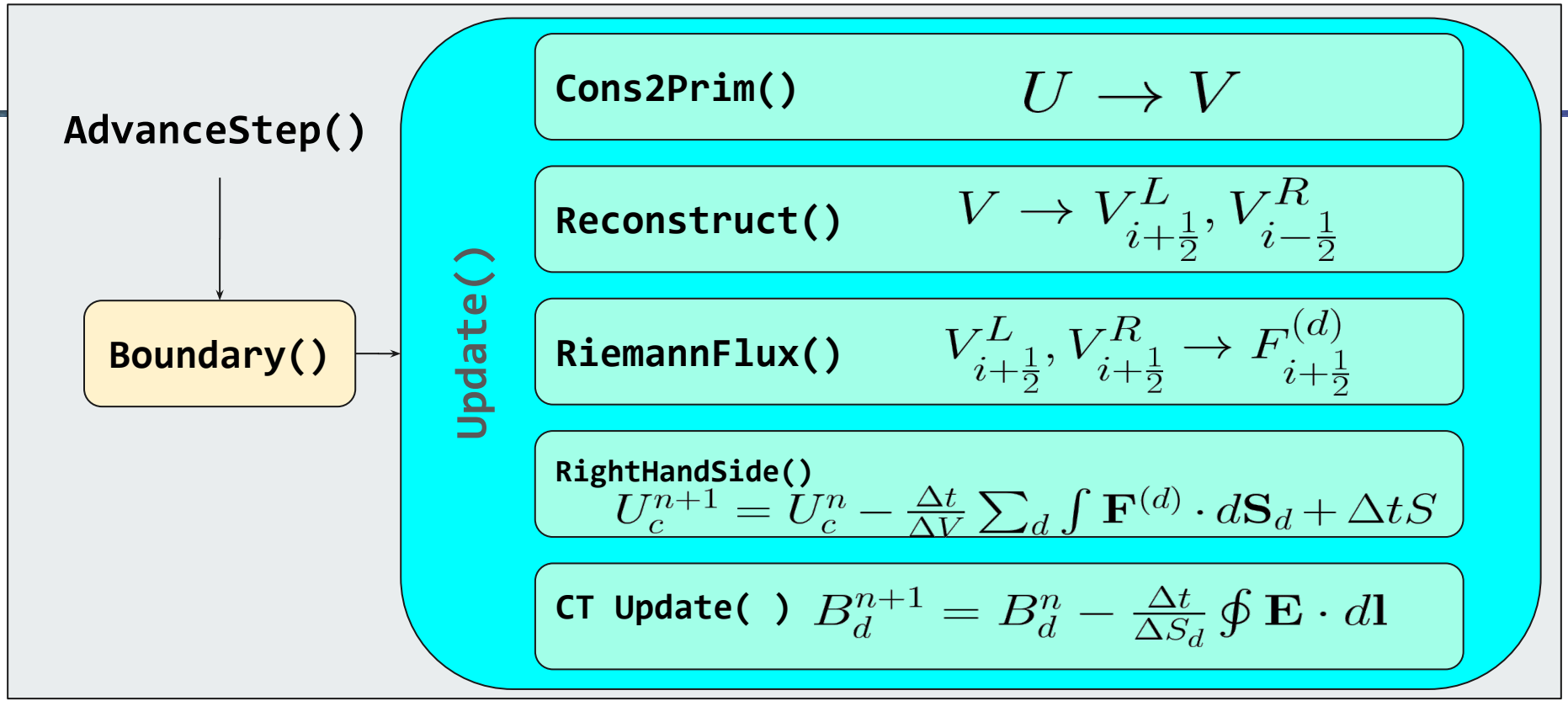


Nodes = GPUs / 4

3D MHD Orsag Tang <u>strong</u> scaling



gPLUTO - (GPU) MAIN KERNELS

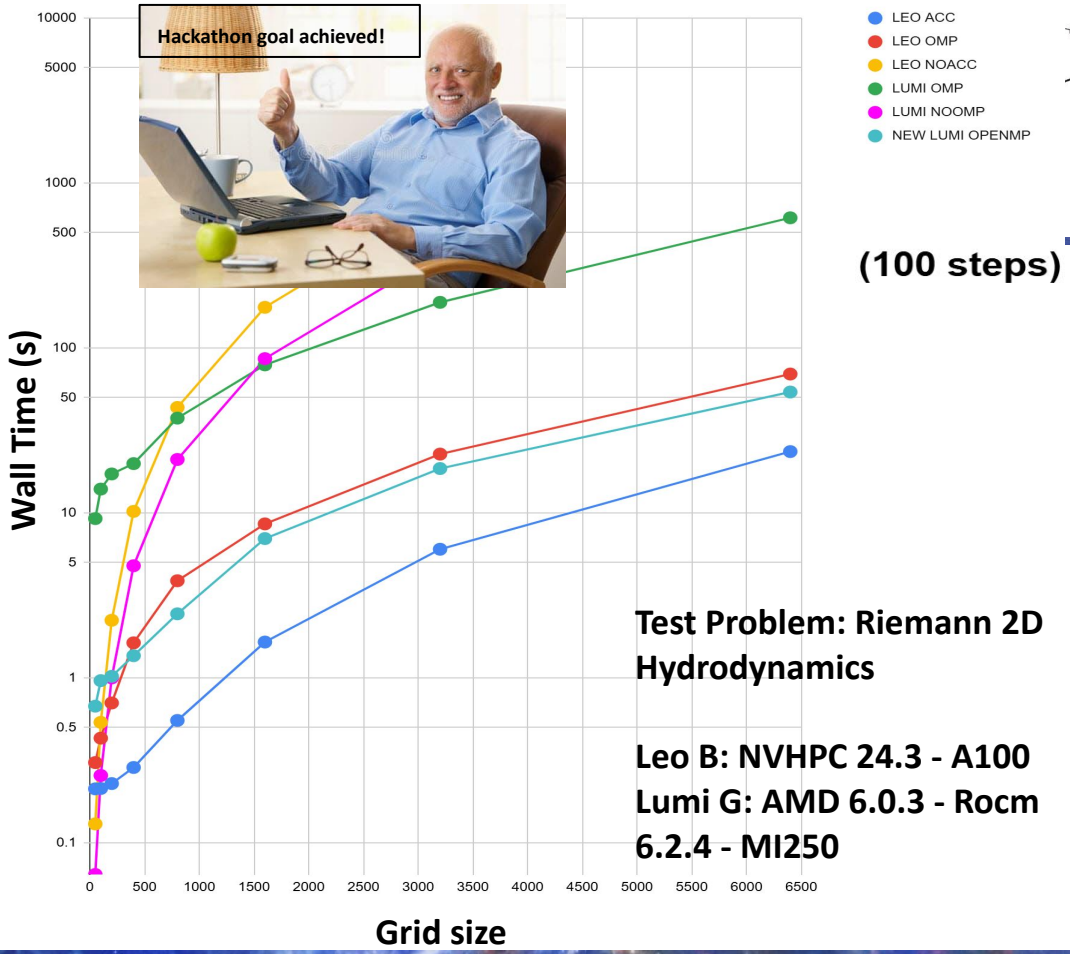


STATUS on LUMI

Mini-app OMP runs on LUMI with comparable performance wtr to Leonardo Booster OMP

- Simple HD test problem:

  Riemann2D
- **OpenMP** still loses against OpenAcc
- **UNIFIED MEMORY** is on
- Gained **15x** in average wtr to old OMP implementation on LUMI



OMP is a standard, but…

- Some directives are preferred for some compiler/GPU (*omp distribute teams loop* vs *omp distribute teams parallel for*): **standard can be implemented differently by specific compiler**

```
#pragma acc parallel loop collapse(2) present(d, run_config, Dts, grid)
//#pragma omp target
//{
#pragma omp target teams distribute parallel for collapse(3) //before: target teams loop collapse
for (k = kbeg; k <= kend; k++){
for (j = jbeg; j <= jend; j++){
  #pragma acc loop
  //#pragma omp simd
  for (i = ibeg; i <= iend; i++){ //Ary definitions moved inside innermost loop to parallelize it

  long int offset  = ni*(j + nj*k);
  long int offset1 = NVAR*ni*(j + nj*k);

  Ary1D cmax(&Dts->cmax[offset],ni);
  Ary1D cs2(&d->sweep.cs2[offset],ni);
  Ary1D press(&d->sweep.press[offset],ni);
  Ary2D vL(&d->sweep.vL[offset1],ni,NVAR);
  Ary2D vR(&d->sweep.vR[offset1],ni,NVAR);
  Ary2D flux(&d->sweep.flux[offset1],ni,NVAR);
```

OMP is a standard, but…

- not all NVIDIA levels of parallelization are available for clang compiler (SIMD ?)

```
#pragma acc parallel loop collapse(2) present(d, run_config, Dts, grid)
//#pragma omp target
//{
#pragma omp target teams distribute parallel for collapse(3) //before: target teams loop collapse
for (k = kbeg; k <= kend; k++){
for (j = jbeg; j <= jend; j++){
  #pragma acc loop
  //#pragma omp simd
  for (i = ibeg; i <= iend; i++){ //Ary definitions moved inside innermost loop to parallelize it

  long int offset  = ni*(j + nj*k);
  long int offset1 = NVAR*ni*(j + nj*k);

  Ary1D cmax(&Dts->cmax[offset],ni);
  Ary1D cs2(&d->sweep.cs2[offset],ni);
  Ary1D press(&d->sweep.press[offset],ni);
  Ary2D vL(&d->sweep.vL[offset1],ni,NVAR);
  Ary2D vR(&d->sweep.vR[offset1],ni,NVAR);
  Ary2D flux(&d->sweep.flux[offset1],ni,NVAR);
```

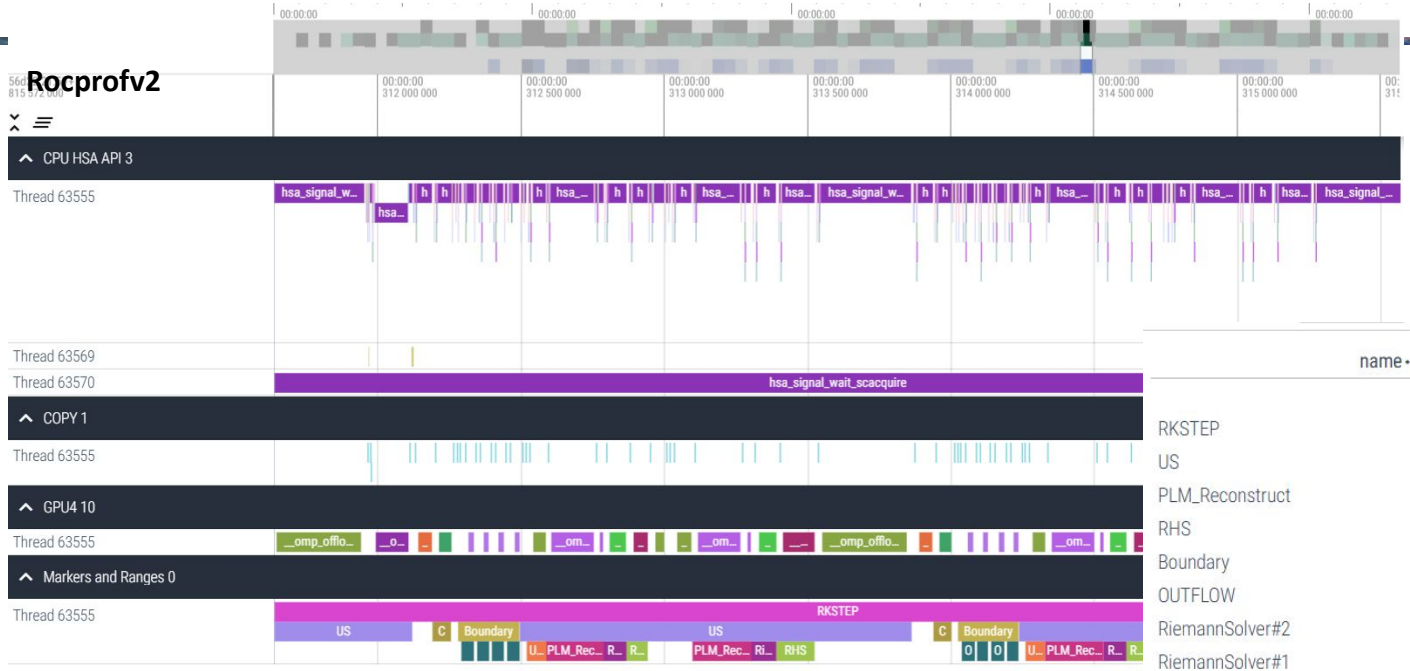
```
#pragma omp target teams distribute parallel for collapse(2)
for (k = kbeg; k <= kend; k++){
for (j = jbeg; j <= jend; j++){
  long int offset  = ni*(j + nj*k);
  long int offset1 = NVAR*ni*(j + nj*k);

  Ary1D cmax(&Dts->cmax[offset],ni);
  Ary1D cs2(&d->sweep.cs2[offset],ni);
  Ary1D press(&d->sweep.press[offset],ni);
  Ary2D vL(&d->sweep.vL[offset1],ni,NVAR);
  Ary2D vR(&d->sweep.vR[offset1],ni,NVAR);
  Ary2D flux(&d->sweep.flux[offset1],ni,NVAR);
  #pragma acc loop
  #pragma omp simd  //not working, as well as omp parallel for or omp loop
  for (i = ibeg; i <= iend; i++){
```

Always check coalescence for performance boost and cache lines optimizations (some loops' order has been inverted)

Proficient users of AMD profiling tools!

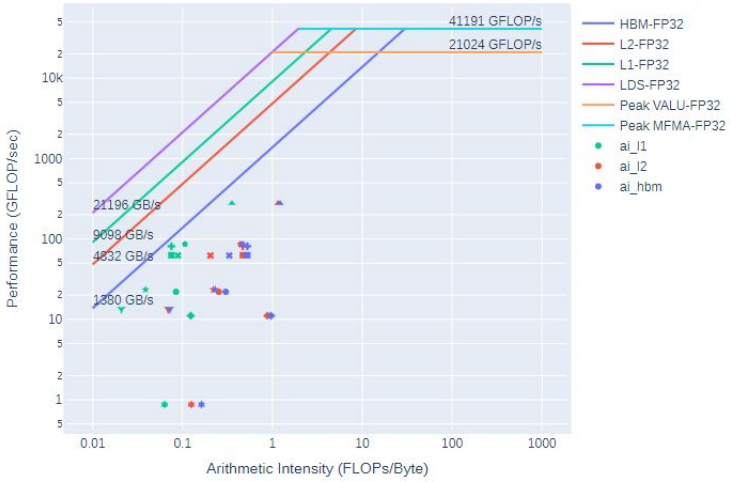
Rocprofv2



name	SUM(dur)
Total values:	5ms 426us
RKSTEP	5ms 426us
US	1ms 534us
PLM_Reconstruct	403us
RHS	219us
Boundary	212us
OUTFLOW	167us
RiemannSolver#2	83us
RiemannSolver#1	83us
USIDIR	68us
ConsToPrim3D	65us

Proficient users of AMD profiling tools!

OmniPerf



Proficient users of AMD profiling tools!

7.2 Wavefront Runtime Stats

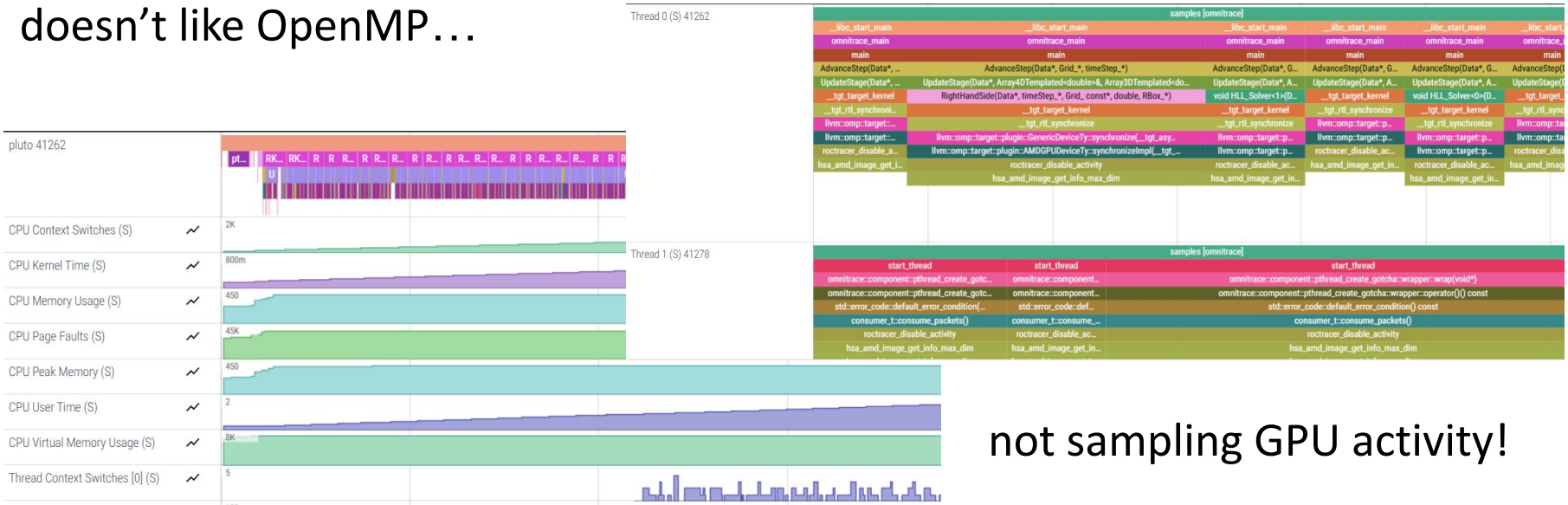
Metric_ID	Metric	Avg	Min	Max	Unit
7.2.0	Kernel Time (Nanosec)	80577.12	3840.00	4021000.00	Ns
7.2.1	Kernel Time (Cycles)	143675.06	16311.00	7087415.00	Cycle
7.2.2	Instructions per wavefront	1010.35	20.50	6881.00	Instr/wavefront
7.2.3	Wave Cycles	93003.44	1123.00	7030613.83	Cycles per wave
7.2.4	Dependency Wait Cycles	73046.99	1014.00	3516654.06	Cycles per wave
7.2.5	Issue Wait Cycles	12363.20	5.00	3121700.43	Cycles per wave
7.2.6	Active Cycles	3873.61	102.00	24340.35	Cycles per wave
7.2.7	Wavefront Occupancy	935.00	0.27	2618.85	Wavefronts

3. Memory Chart
3.1 Memory Chart

Metric_ID	Metric	Value
3.1.0	Wavefront Occupancy	16.0
3.1.1	Wave Life	93003.0
3.1.2	SALU	268.0
3.1.3	SMEM	7.0
3.1.4	VALU	492.0
3.1.5	MFMA	0.0
3.1.6	VMEM	74.0
3.1.7	LDS	81.0
3.1.8	GWS	0.0
3.1.9	BR	72.0
3.1.10	Active CUs	58.0
3.1.11	Num CUs	110.0

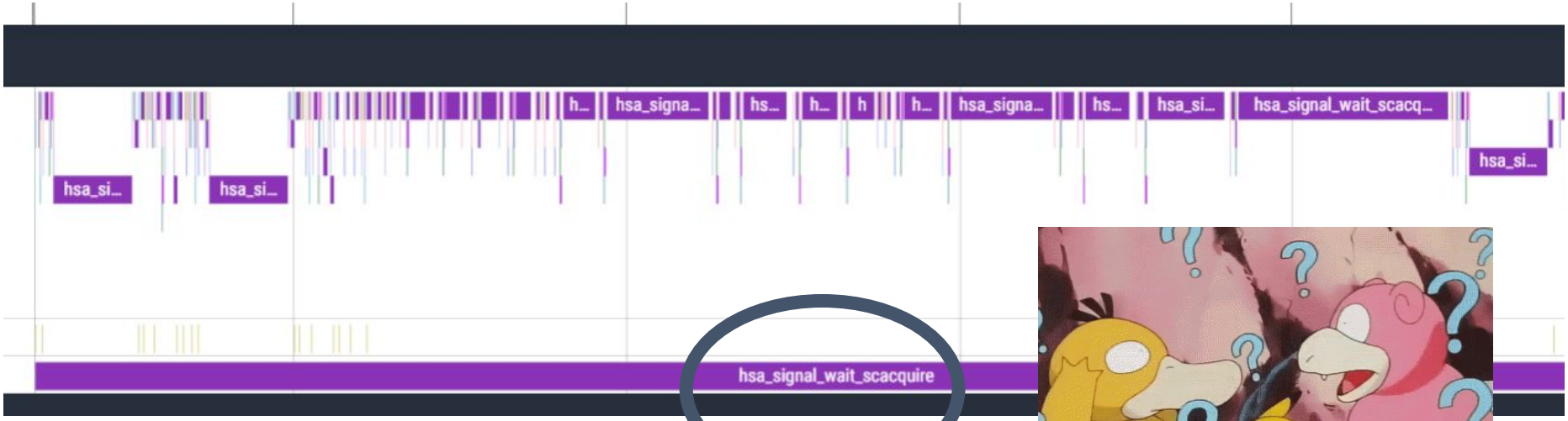
Proficient users of AMD profiling tools… ?

It’s very likely that **OmniTrace** doesn’t like OpenMP…



not sampling GPU activity!

Some strange overhead in the background…



Unified memory can soothe you, but…

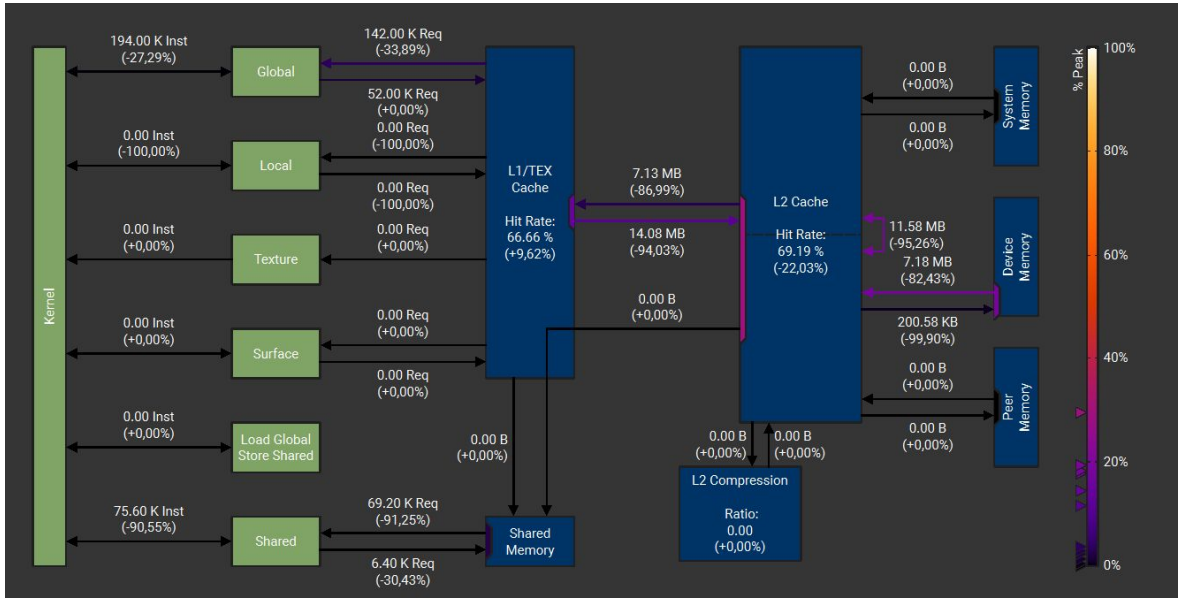
- Complicated structures and classes in gPLUTO require **mappers** to carefully move data from host to device (and viceversa)
- **Compiler clang bug** if mapping members of structures
- Turnaround: using pointers to structures' members, but…

```
4   struct Array1DTemplated {
38
39    T  *data_;
40    int n0_;
41    int stride0_;
42    int total_size_;
43  };
44  using Ary1D=Array1DTemplated<double>;
45  #pragma omp declare mapper(Ary1D arr) map(arr, arr.data_[0:arr.total_size_], arr.stride0_)
46
47  struct Data{
48      Ary1D A;
49      Ary1D B;
50      Ary1D C;
51  };
52  #pragma omp declare mapper (Data d) map(d, d.A, d.B, d.C)
```

Moving towards full app

- Same optimizations seems to hold for full-app;
- 50 steps of 6400x6400 HD Riemann 2D displays a **14x speed-up** (but still losing **3x** wtr OpenACC);
- Compared to OpenMP implementation (nvhpc/24.3) on Leonardo Booster we gain a **2.7x;**
- Not sure if we can keep same modifications for more complicated cases (loop ordering changing) like 3D MHD (i.e. Orszag-Tang test);
- We started to implement multi-GPU case with MPI (synchronous boundaries exchange).

Limits of OMP: strange memory movements within the GPU



- Full-app Reconstruct kernel shows more register spilling and less arithmetic intensity wtr to OpenACC on Leonardo B.
- Need to check this on LUMI, but generally these issues may depend on compilers and can be handled only with lower level paradigms…

Future outlooks

- Implement asynchronous kernels and complete multi-GPU porting;
- Check optimizations are still valid for other cases;
- Check if OpenACC code does not lose too much performance with these changes;
- Comparison with other compilers/GPUs (intel one-api ?);
- **Pray almighty AMD engineers do their job in solving issues and keep implementing OMP standards**.

According to ChatGPT this is what happens if CLANG compiler wins the fight

Acknowledgement & Disclaimer

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Belgium, Czech Republic, France, Germany, Greece, Italy, Norway, and Spain under grant agreement No 101093441.

Thank You