EUROfusion

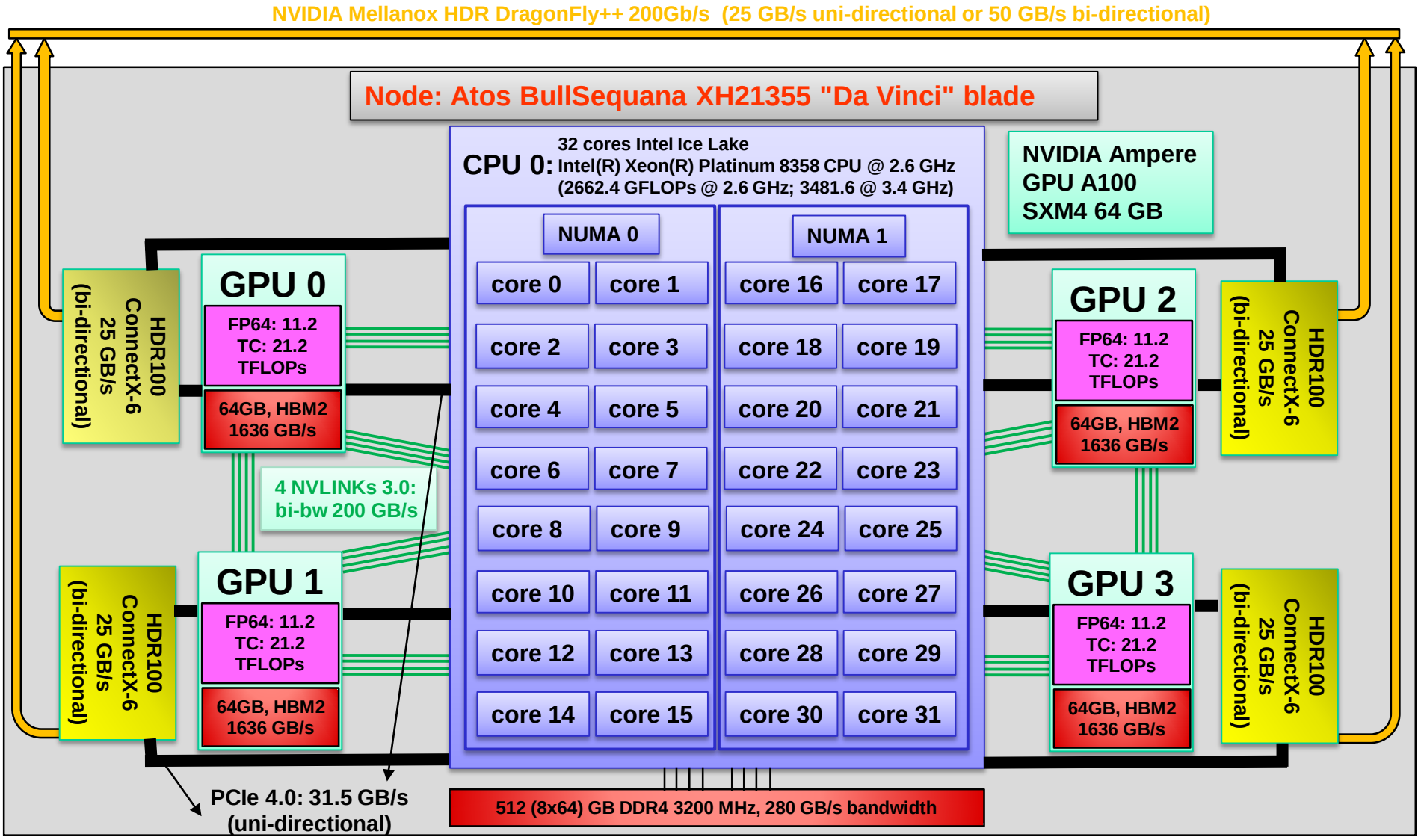
CINCOMP project: Benchmarks and validation of the EUROfusion HPC system

Serhiy Mochalskyy

Joint ACH HPC meeting
November 15-16, 2023

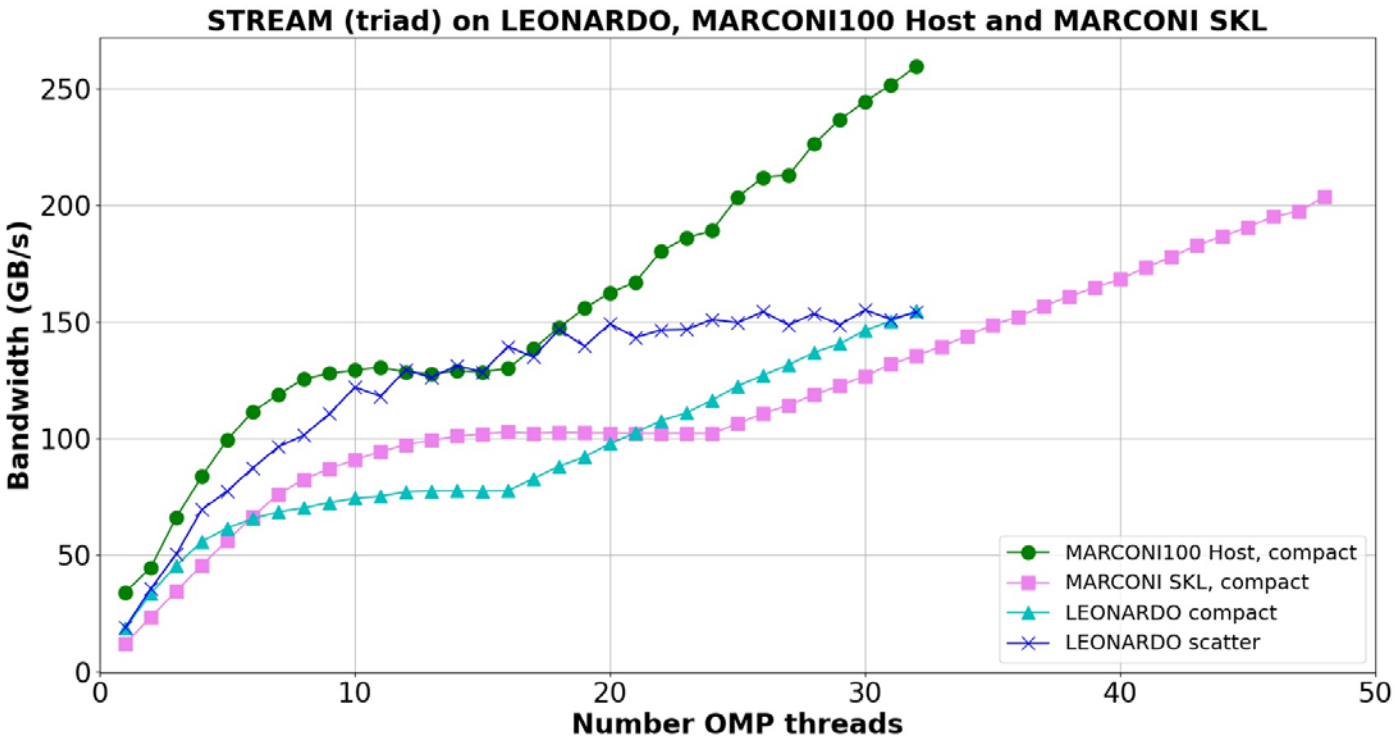
Advanced Computing Hub Garching
Max-Planck-Institut für Plasmaphysik
Boltzmannstr. 2, D-85748 Garching, Germany

LEONARDO Atos BullSequana XH21355 "Da Vinci" blade node



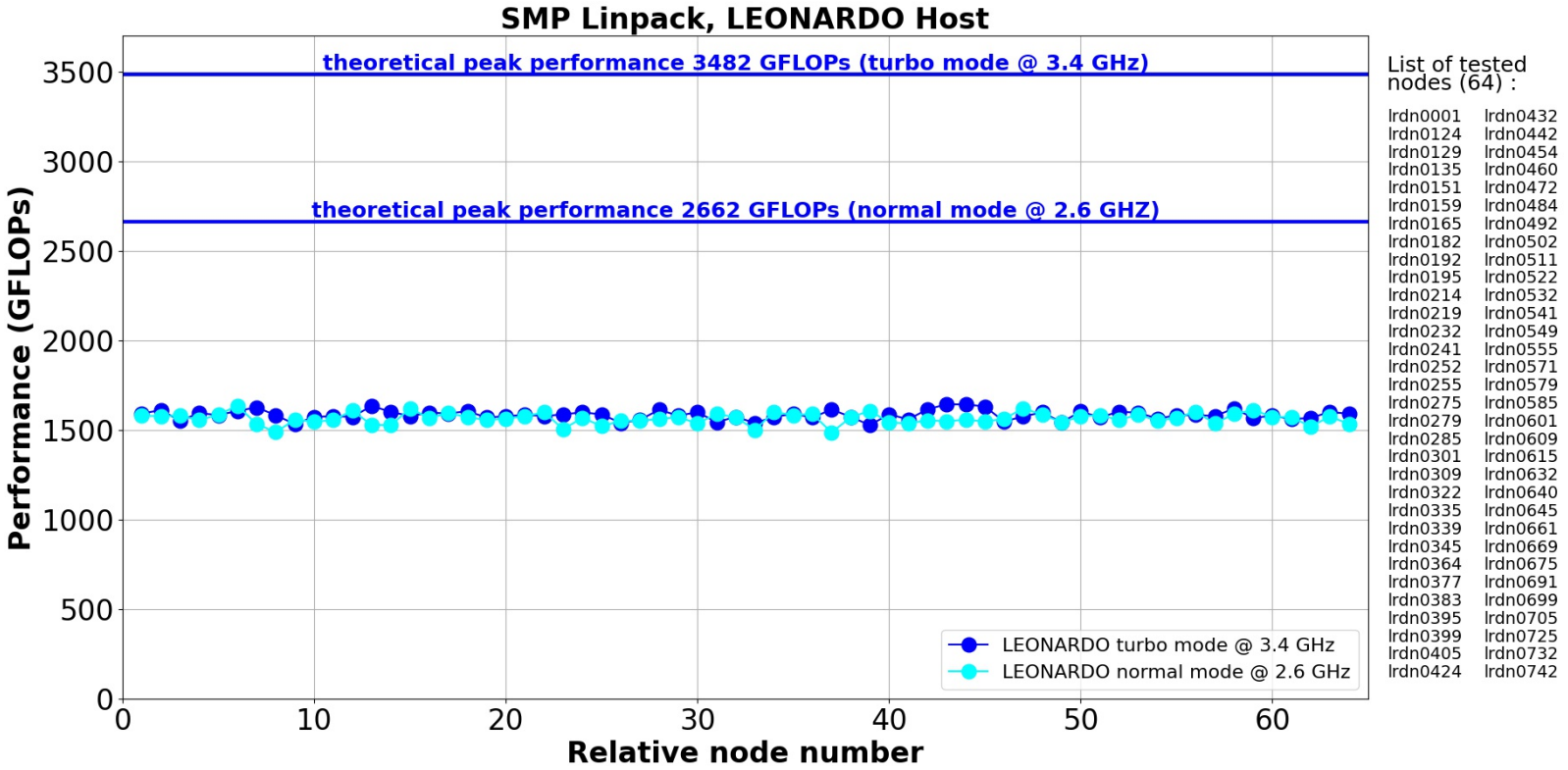
STREAM on a single node

```
icx -O3 -qopenmp -mcmodel=medium -qopt-streaming-stores=always -mtune=icelake-client -xHost
-DSTREAM_ARRAY_SIZE=400000000 -DVERBOSE  -DNTIMES=100 stream.c -o stream_intel.x
```



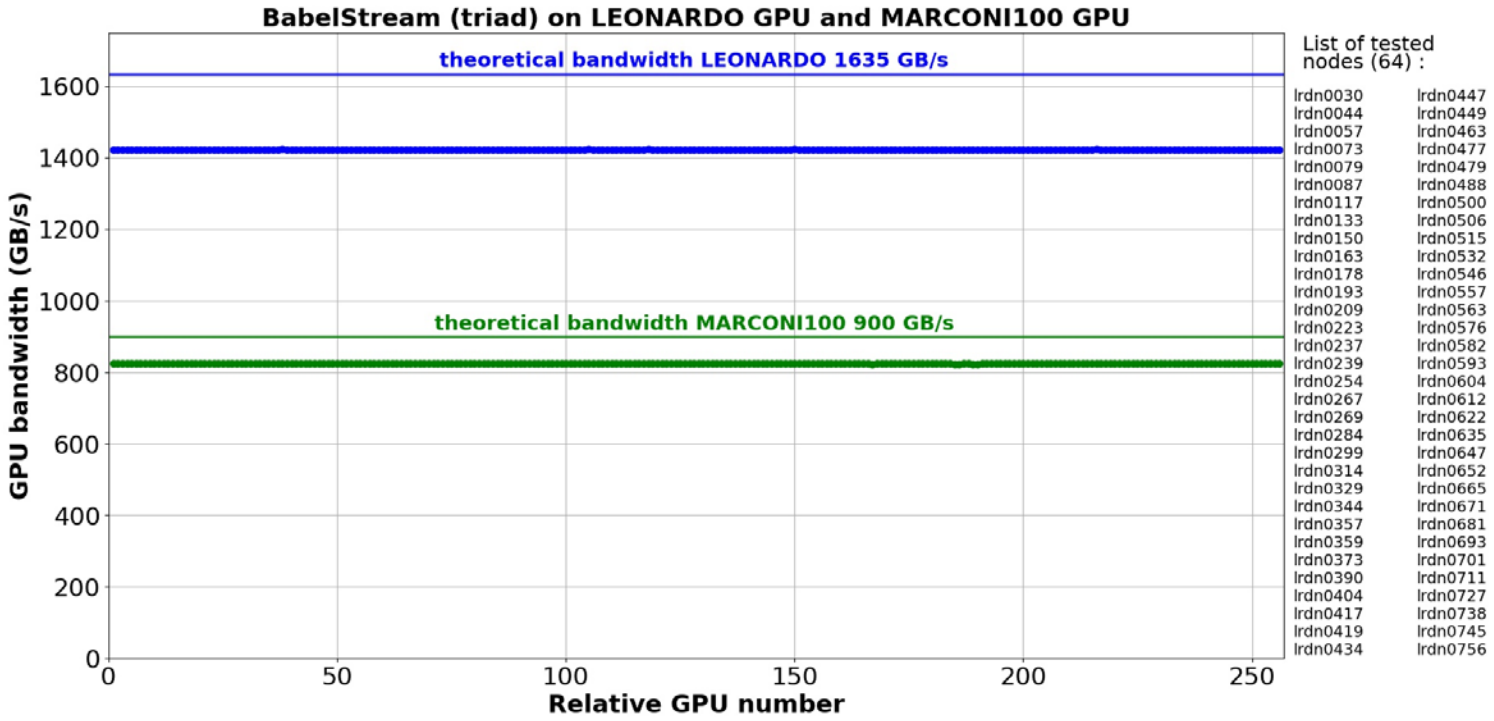
- MARCONI SKL: 203 GB/s from 255.94 GB/s theoretical (80%).
- MARCONI100: 266 GB/s from 280 GB/s theoretical (95%).
- LEONARDO: 153 GB/s from 205 GB/s theoretical (75%).

Performance on Host, stability test



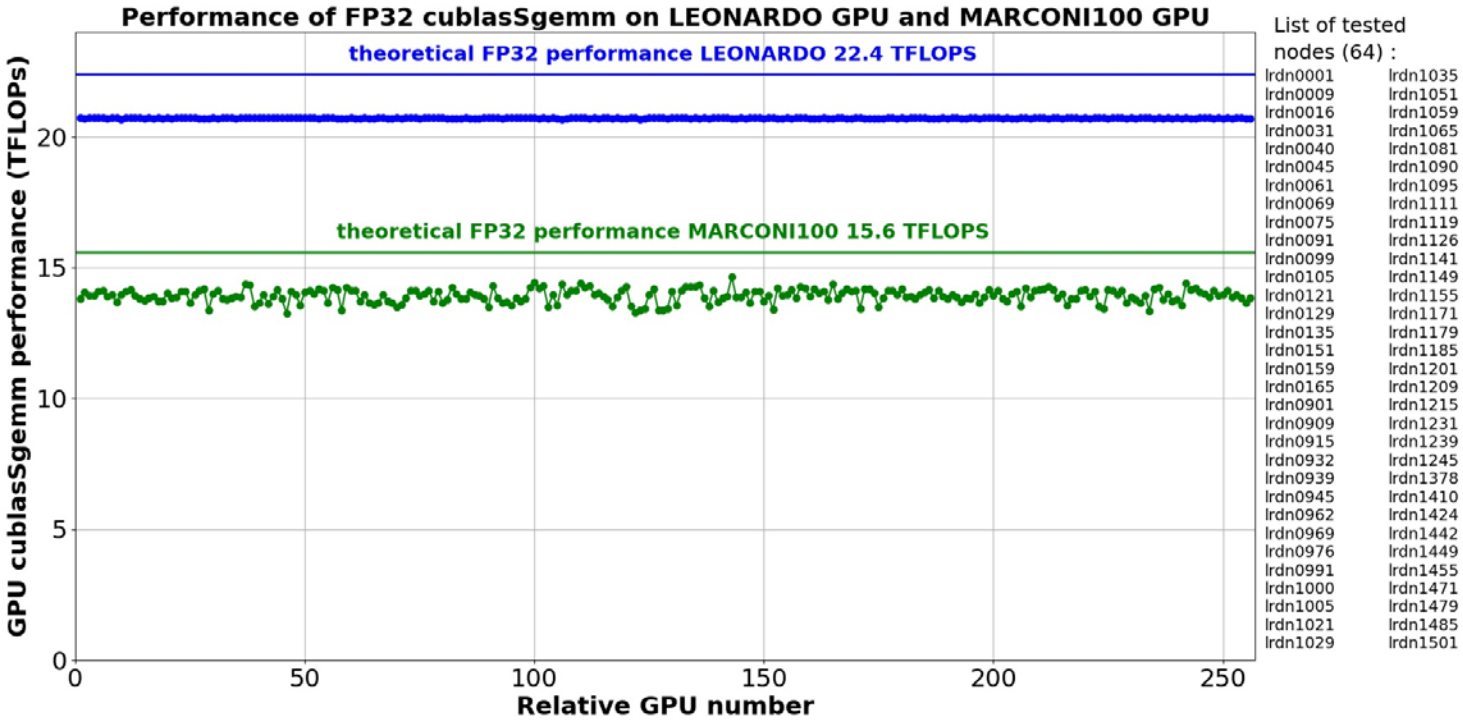
- All nodes provide stable performance.
- The performance is relatively low:
  - turbo mode: mean=1584 GFLOPs (theoretical peak 3482 GFLOPs ~45.5%).
  - normal mode: mean=1563 GFLOPs (theoretical peak 2662.4 GFLOPs ~58.7%).
- MARCONI SKL = 2028.28 GFLOPs (theoretical peak 3211 GFLOPs ~63%).

BabelStream benchmark on LEONARDO GPU



- All GPUs provide high, stable and symmetric bandwidth close to the theoretical value.
- No difference between GPUs on different nodes or GPUs inside one node.
- LEONARDO: 1423.5 GB/s from 1635 GB/s theoretical (87%).
- MARCONI100: 845 GB/s from 900 GB/s theoretical (94%).

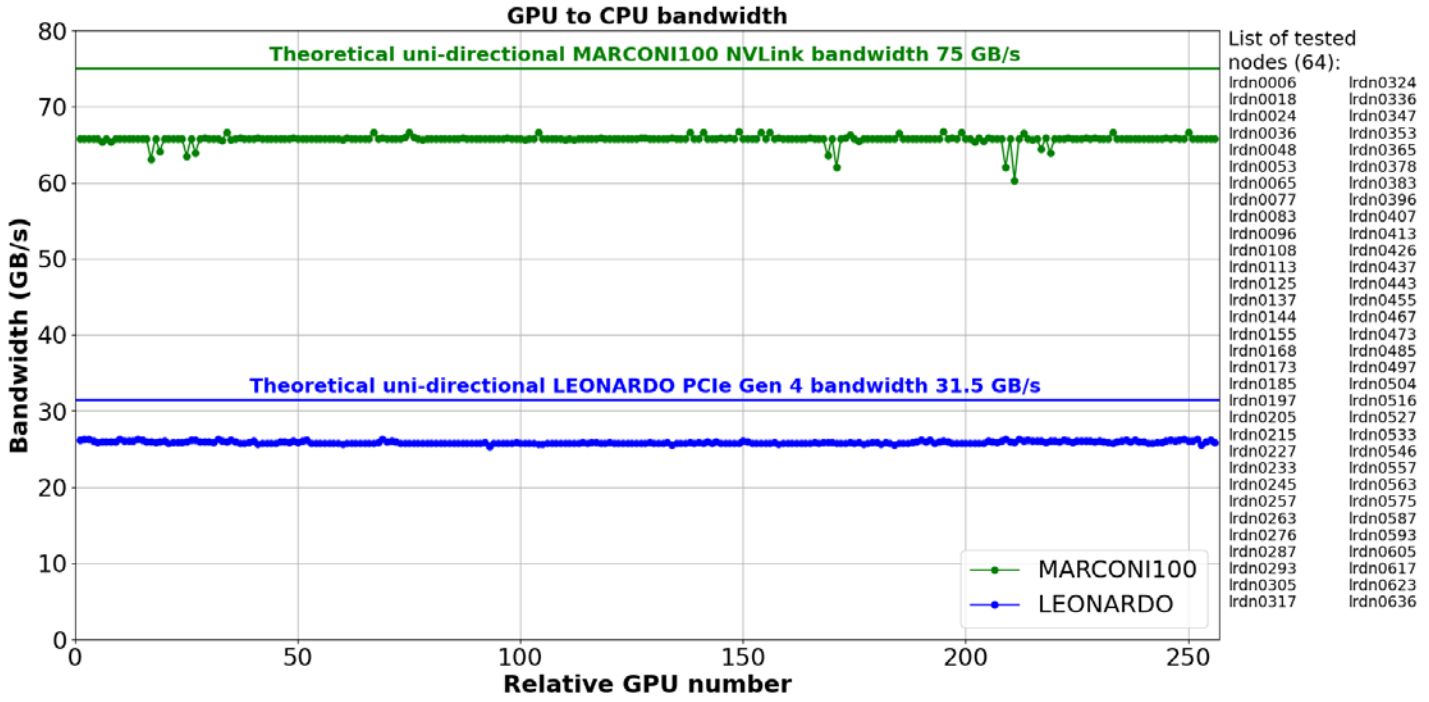
SGEMM (cublasSgemm) benchmark on LEONARDO GPU



- All GPUs provide high, stable and symmetric performance close to the theoretical value.
- No difference between GPUs on different nodes or GPUs inside one node.
- LEONARDO: 20.7 TFLOPs per GPU from 22.4 TFLOPs theoretical (92%).
- MARCONI100: 14 TFLOPs per GPU from 15.6 TFLOPs theoretical (90%).

GPU to CPU bandwidth

using bandwidthTest benchmark from NVIDIA samples

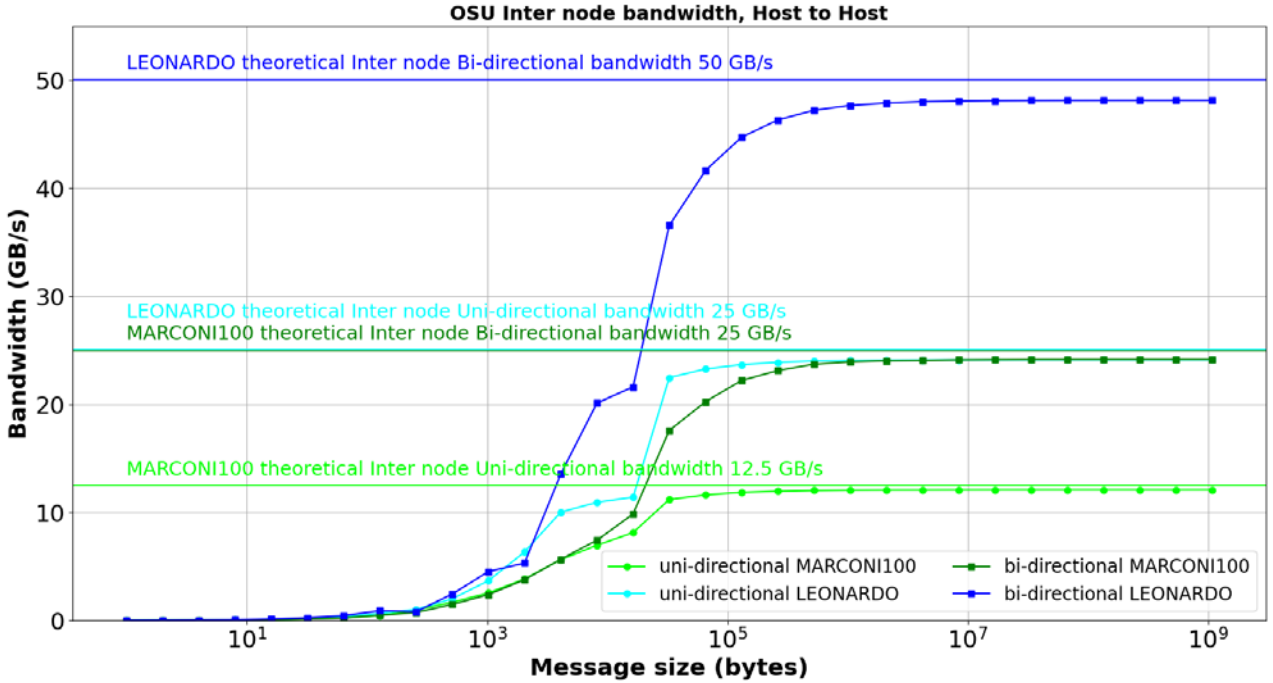


- The results are stable on LEONARDO and stable on MARCONI100.
- LEONARDO: the mean uni-directional bandwidth of ~26 GB/s from 31.5 GB/s of the theoretical value (83 %).
- MARCONI100: the mean uni-directional bandwidth of ~66 GB/s from 75 GB/s of the theoretical value (88 %).

Inter node network bandwidth, Host to Host

NVIDIA Mellanox HDR DragonFly++ 200Gb/s (25 GB/s) uni-directional or 50 GB/s bi-directional

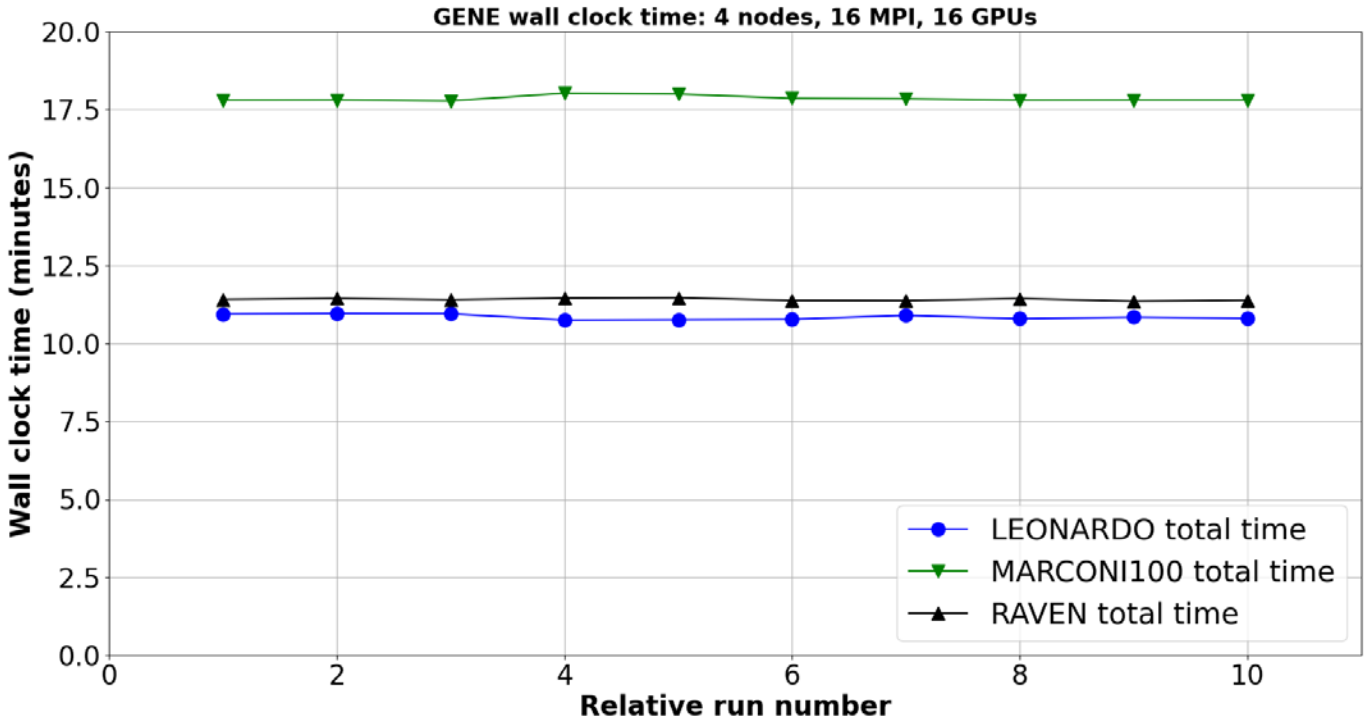
using osu_bw and osu_bibw benchmarks from OSU microbenchmark



- **Stable and high bandwidth for uni- and bi-directional data transfer.**
- **LEONARDO: bi-directional bandwidth ~48 GB/s from 50 GB/s of the theoretical value (96 %).**
- **LEONARDO: uni-directional bandwidth ~24 GB/s from 25 GB/s of the theoretical value (96 %).**
- **MARCONI100: bi-directional bandwidth ~24.2 GB/s from 25 GB/s of the theoretical value (97 %).**
- **MARCONI100: uni-directional bandwidth ~12.1 GB/s from 12.5 GB/s of the theoretical value (99 %).**

GENE performance

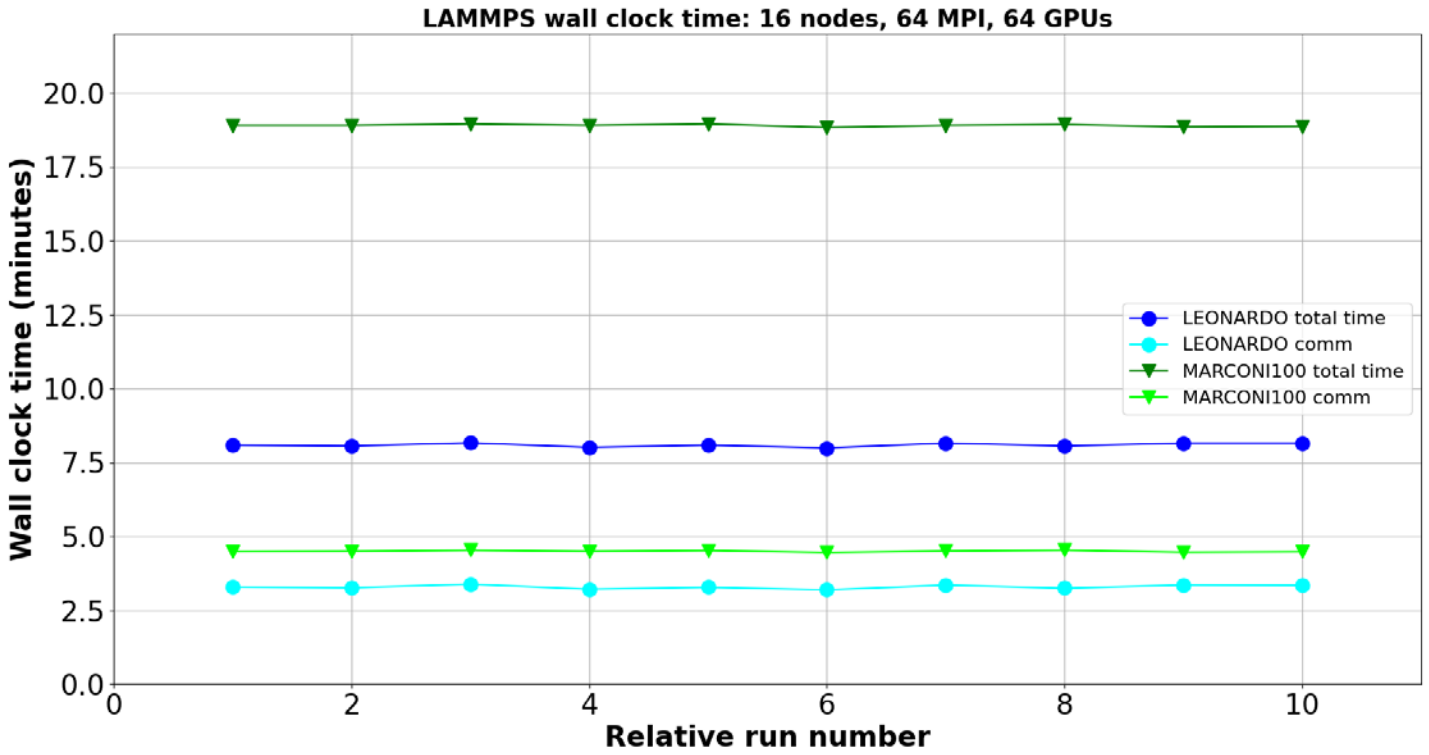
4 nodes, 16 MPI, 16 GPUs



- The execution time is stable on all supercomputers.
- The code is faster on LEONARDO (factor of ~1.6) in comparison to MARCONI100.
- LEONARDO and RAVEN provide similar wall clock time.

LAMMPS performance

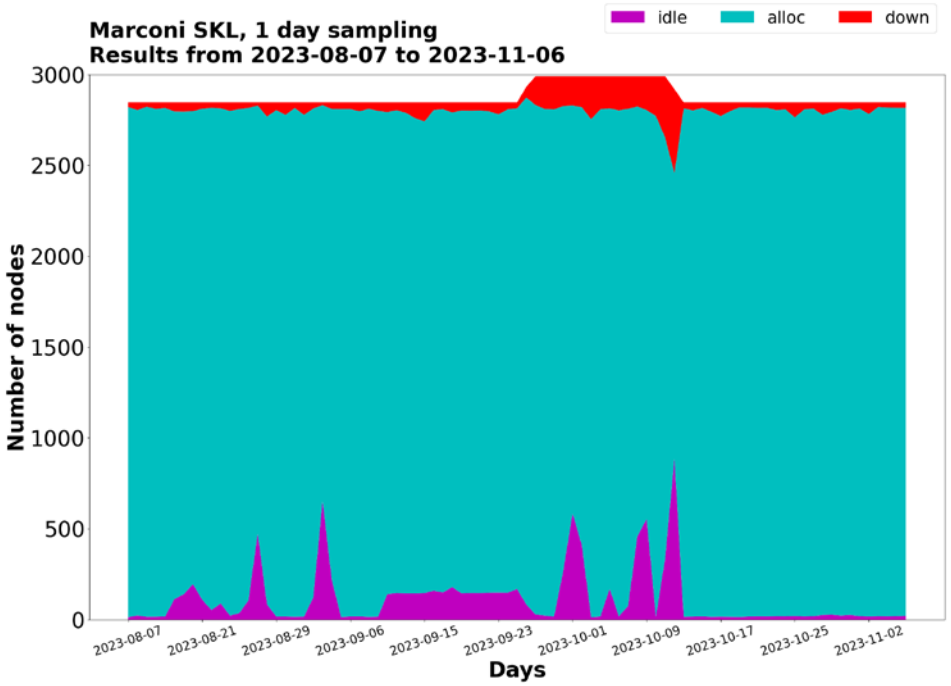
16 nodes, 64 MPI, 64 GPUs



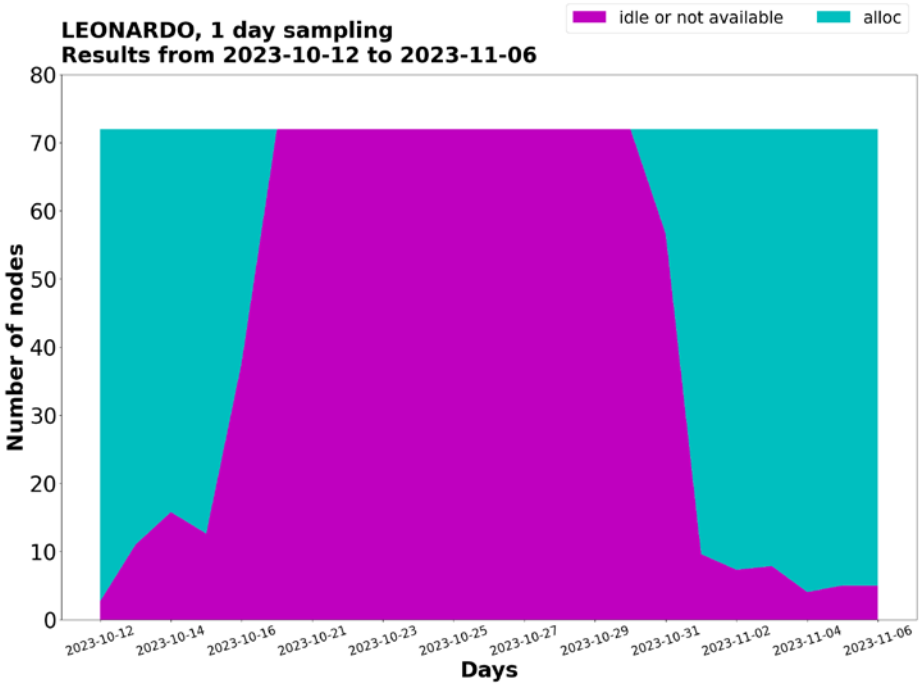
- The execution time is stable on both supercomputers.
- The code is faster on LEONARDO (factor of ~2.3).
- The communication time is similar.

Supercomputer recourses usage

Marconi SKL

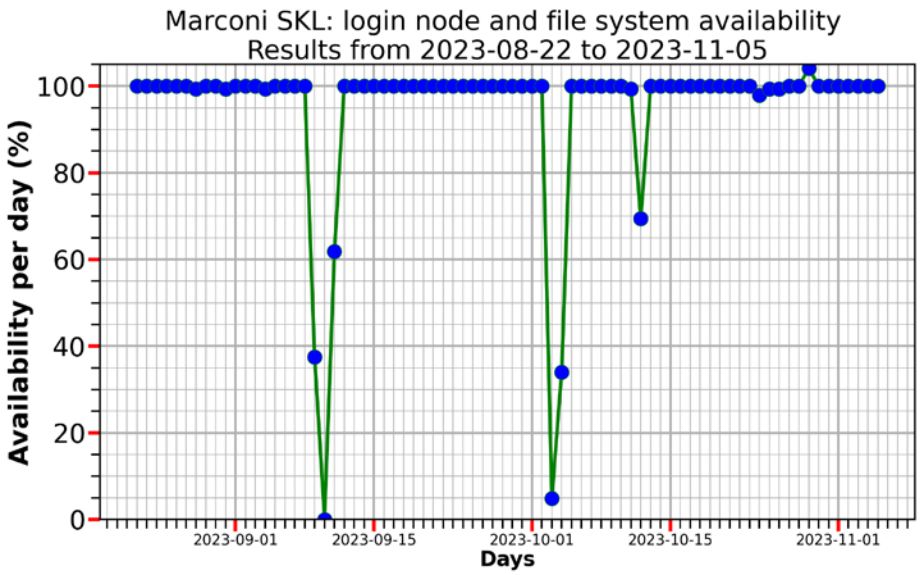


LEONARDO

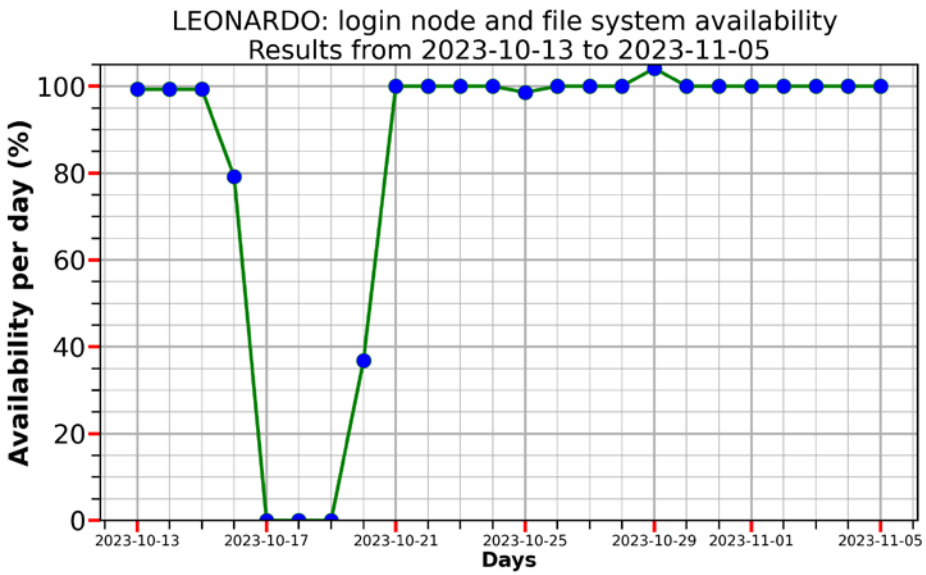


Supercomputer login node accessibility

Marconi SKL



LEONARDO



EUROfusion

GENE project: Porting of the GENE code on GPU

Serhiy Mochalskyy

Joint ACH HPC meeting
November 15-16, 2023

Advanced Computing Hub Garching
Max-Planck-Institut für Plasmaphysik
Boltzmannstr. 2, D-85748 Garching, Germany

The GENE code

The Gyrokinetic Electromagnetic Numerical Experiment

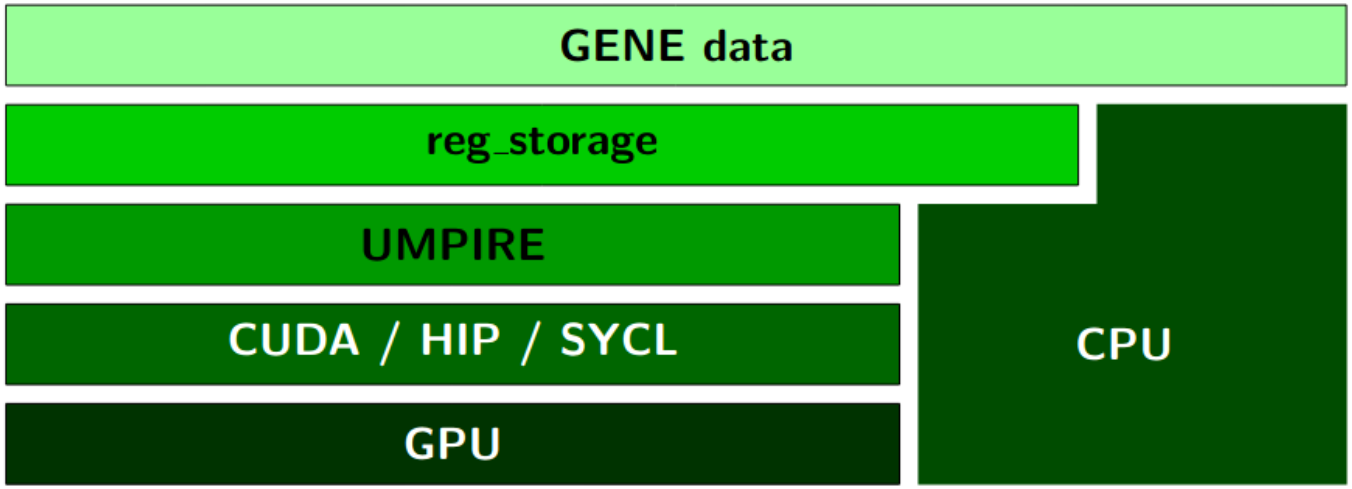
- has been developed for 20 years at the IPP, Garching;
- solves the Maxwell-Vlasov system of integro-differential equations;
- able to consider Non-Maxwellian background distributions;
- is one of the leading gyrokinetic turbulence codes in the world;
- is written in Modern Fortran: object oriented approach;
- Massively parallelized (MPI), well-benchmarked and validated in several scenarios
- one of the top users of our HPC systems.

- master branch: 135k lines
- cuda_under_the_hood branch: 181k lines
- in 409 files (362 Fortran, 40 C/C++)
- 372 Fortran MODULEs
- over 100 publications related to the GENE code;
- has many (> 150) users all over the world.

Additional libraries for GPU (*reg_storage*)

***reg_storage*: the specially developed library for GENE is used to facilitate sub-division and/or reshaping of a multi-dimensional array with a custom memory allocation.**

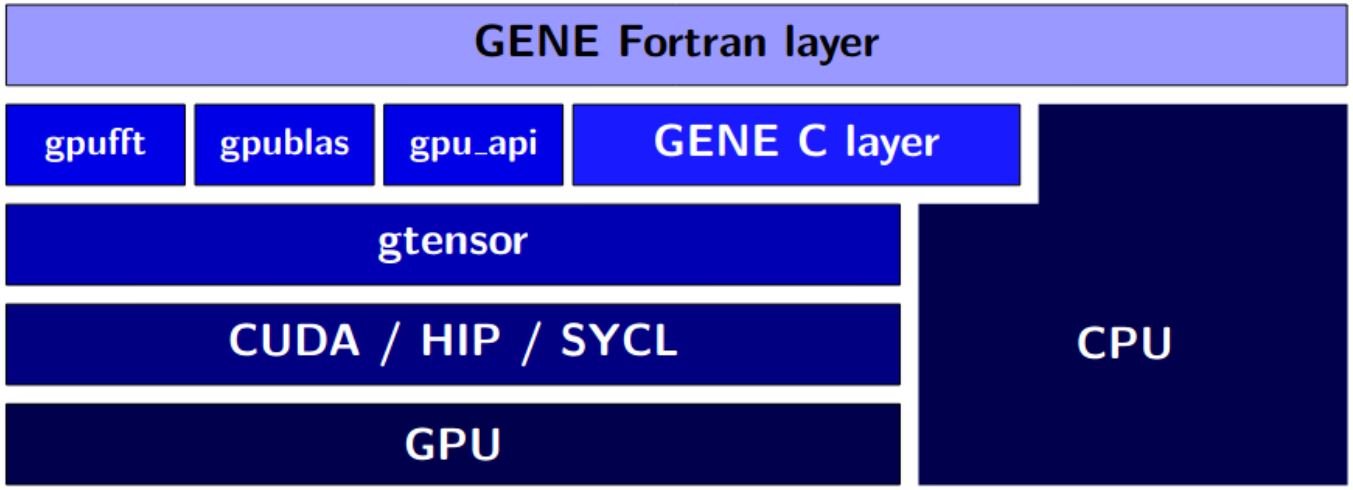
The library encapsulates a defined region of contiguous memory, and its interpretation as multi-dimensional arrays of varying dimensionality. The library can also deal with the managed memory that automatically migrates a memory region between the host (CPU) and device (GPU) memory depending on where it is accessed from.



Additional libraries for GPU (GTENSOR)

In order to simplify the passing of multi-dimensional arrays from Fortran to C/C++ and CUDA, the GTENSOR library was specifically developed.

This library significantly simplifies the generation of CUDA kernels using C++ lambda functions.



GTENSOR function

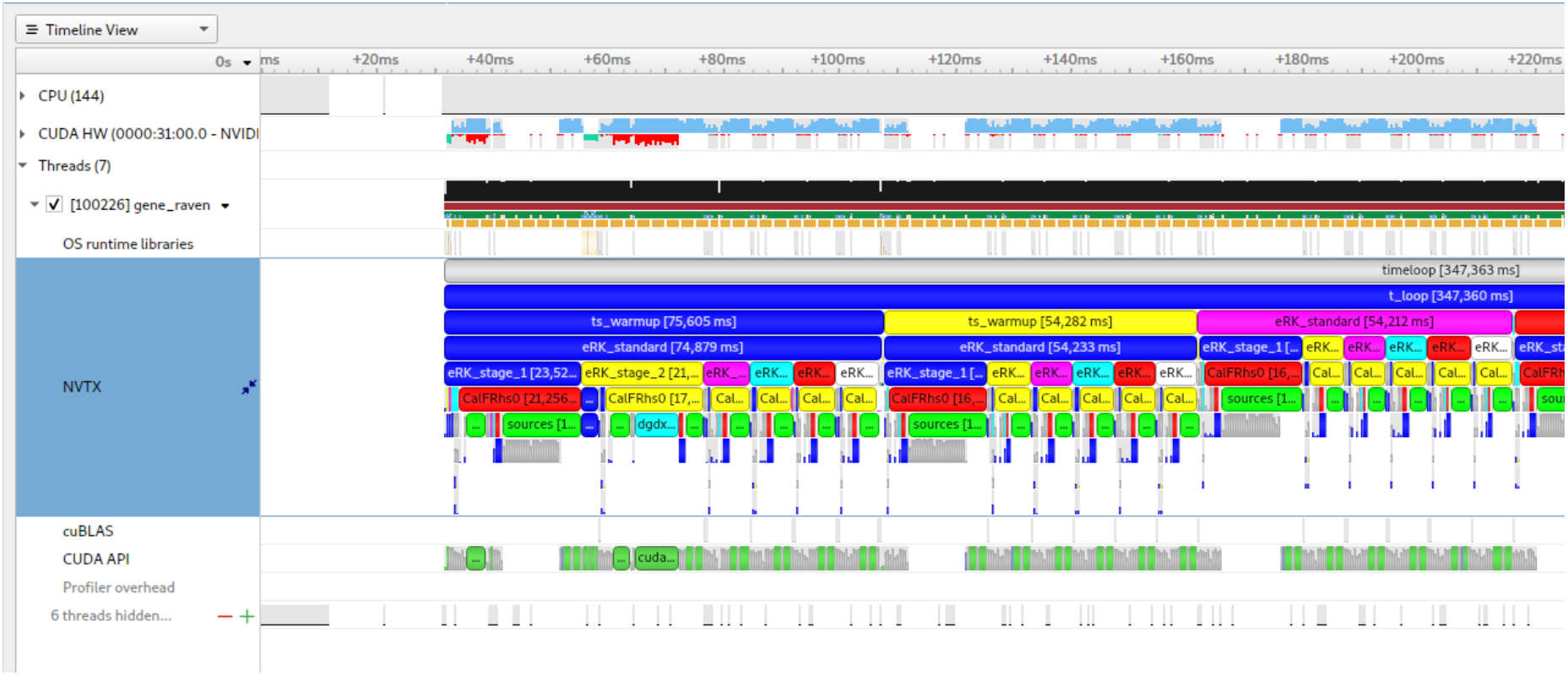
```
extern "C" void krookheat_wrapper(
  real_t* _krookheat_src, const int* krookheat_src_shape, const real_t* _f_vabs,
  const int* f_vabs_shape, const real_t* _k_heat_prefac,
  const int* k_heat_prefac_shape, const real_t* _int_vel_f_vabs,
  const int* int_vel_f_vabs_shape)
{
  auto krookheat_src = gt::adapt_device<4>(_krookheat_src, krookheat_src_shape);
  auto f_vabs = gt::adapt_device<4>(_f_vabs, f_vabs_shape);
  auto k_heat_prefac = gt::adapt_device<4>(_k_heat_prefac, k_heat_prefac_shape);
  auto int_vel_f_vabs = gt::adapt_device<2>(_int_vel_f_vabs, int_vel_f_vabs_shape);

  krookheat_src = f_vabs - k_heat_prefac * int_vel_f_vabs.view(_all, _newaxis,
                                                  _newaxis, _all);

  gt::synchronize();
}
```

```
f_block = f.view(_all, _all, _s(nbz, -nbz), _s(nbv, -nbv), _s(nbw, -nbw), _s(n_b, n_e));
```

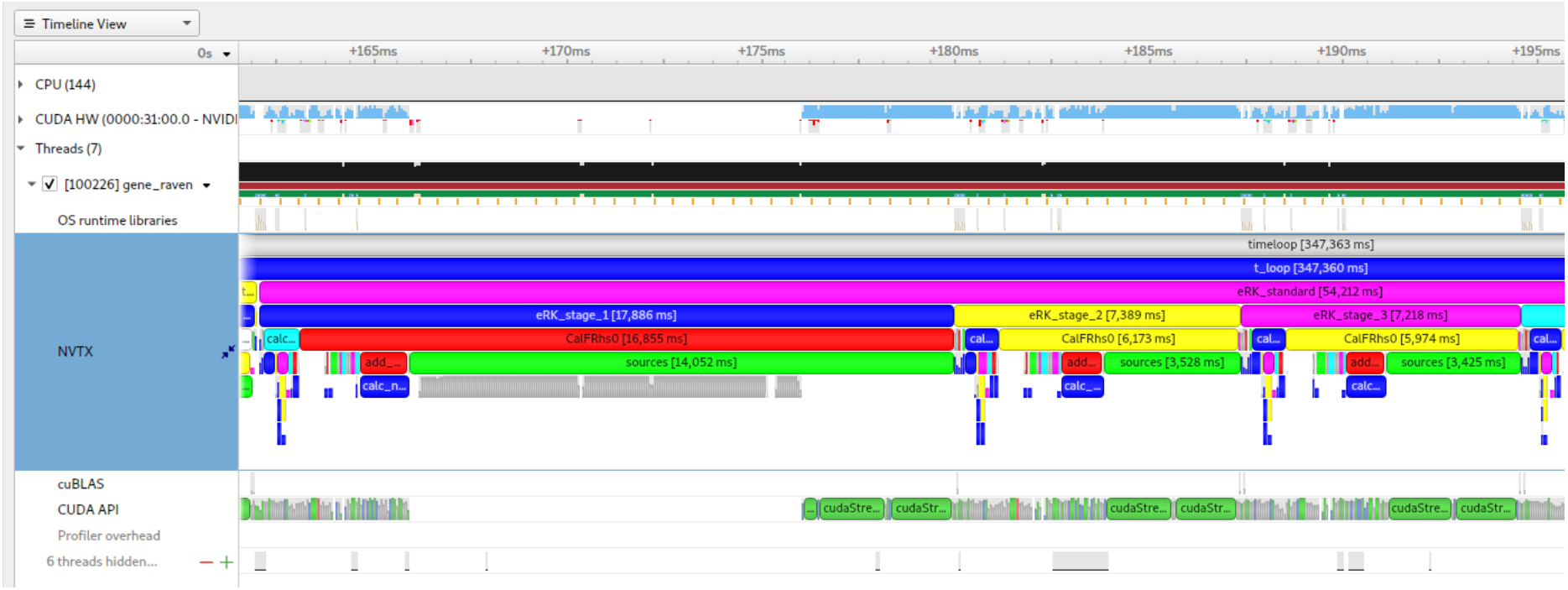
GTENSOR kernel

```
extern "C" void z_deriv_gpu_wrapper(complex_t* dydz, const int* dydz_shape,
                                    const complex_t* y, const int* y_shape,
                                    const real_t* par_sten,
                                    const int* par_sten_shape)

{
  auto gt_dydz = gt::adapt_device<5>(dydz, dydz_shape);
  auto gt_y = gt::adapt_device<5>(y, y_shape);
  auto gt_par_sten = gt::adapt_device<1>(par_sten, par_sten_shape);

  int sten_bound = (gt_y.shape(2) - gt_dydz.shape(2)) / 2;

  gt::launch<4>(
   {gt_dydz.shape(0), gt_dydz.shape(1), gt_dydz.shape(3), gt_dydz.shape(4)},
   GT_LAMBDA(int i, int j, int m, int n) {
    for (int k = 0; k < gt_dydz.shape(2); ++k) {
      gt_dydz(i, j, k, m, n) = 0.0;

      for (int sten = -sten_bound; sten <= -1; ++sten) {
        gt_dydz(i, j, k, m, n) =
          gt_dydz(i, j, k, m, n) + gt_par_sten(sten + sten_bound) *
                                   gt_y(i, j, k + sten_bound + sten, m, n);
      }

      for (int sten = 1; sten <= sten_bound; ++sten) {
        gt_dydz(i, j, k, m, n) =
          gt_dydz(i, j, k, m, n) + gt_par_sten(sten + sten_bound) *
                                   gt_y(i, j, k + sten_bound + sten, m, n);
      }
}
```

Nsight systems profiler



Nsight systems profiler zoom



Nsight systems profiler final

Thank you for our attention!