GPU porting of ASCOT5 code for Monte Carlo simulations in fusion plasmas

M. Peybernes, G. Fourestey, S. Äkäslompolo, K. Särkimäki, F. Spiga

5th Fusion HPC Workshop

EUROfusion Advanced Computing Hubs

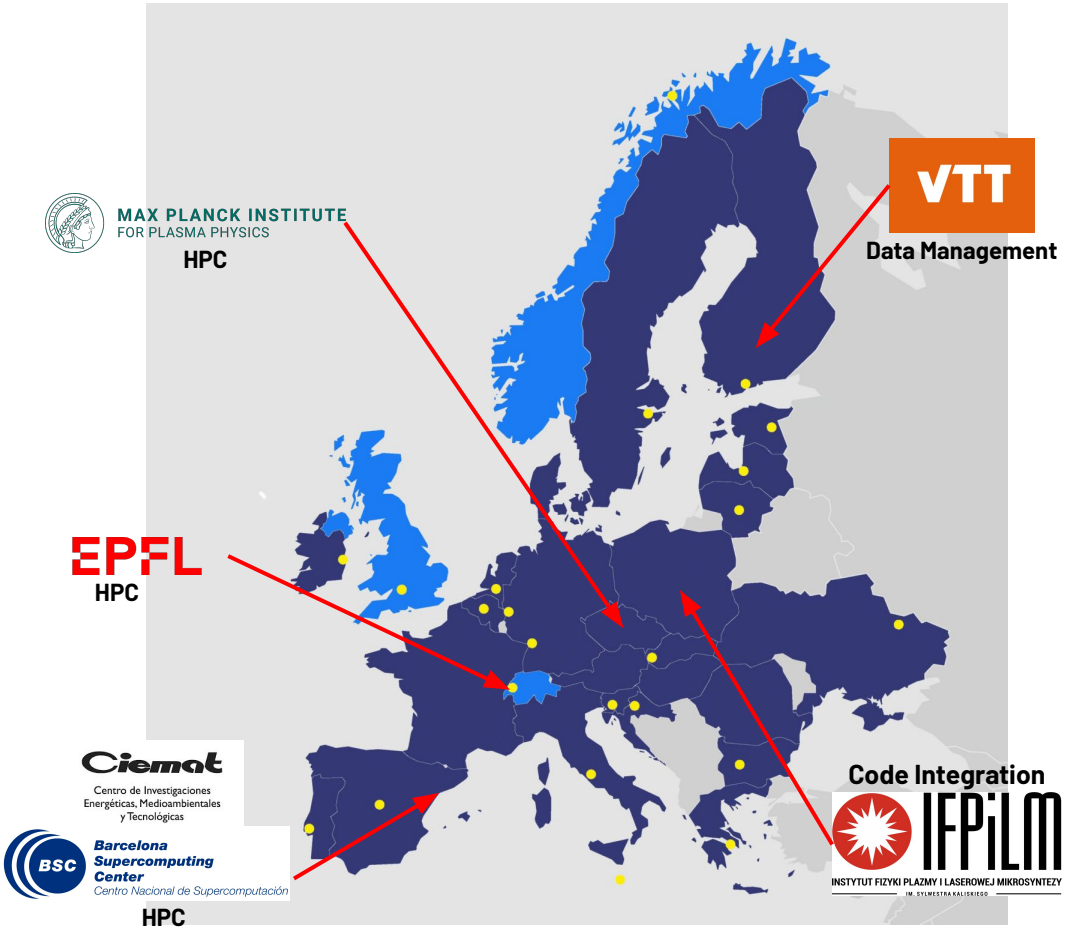
EUROfusion E-TASC – Theory and Advanced Simulation Coordination between:

- **14 TSVV** (Theory, Simulation, Validation and Verification) projects
- **5 ACH** (Advanced computing Hubs)

In particular, **3 HPC ACHs** were created in order **to help building power plants through numerical simulations**

- Extension of **HLST** (Roman Hatsky/IPP)
- Develop **efficient, reliable** tools
- **Modernize and industrialize research codes**

In order to **gain insight and predict fusion experiments** (ITER, JT60-SA, DEMO…)



How to transition towards GPU codes

Plasma simulation codes are research codes:

- mostly **CPU-only**,
- written in **C,C++ and Fortran**,
- **MPI** and/or **OpenMP**,
- **under active development** by physicists, mathematicians, etc...
- main objective: GPU porting

General porting rules:

- **least modifications** of the code/no rewrite
- "maximum" **performance**
- **portability**/no specific target

3 approaches:

Library encapsulation (e.g. Kokkos, PETSc, AmgX, BLAS/Lapack...)

Cuda/ROCm

Pragma directives (OpenMP offload / OpenACC)

- GRILLIX — MAX PLANCK INSTITUTE FOR PLASMA PHYSICS
- GyselaX — cea irfm
- ORB5 — EPFL WARWICK MAX PLANCK INSTITUTE FOR PLASMA PHYSICS
- Soledge3X — cea irfm
- ASCOT5 — Aalto University
- CAS3D — MAX PLANCK INSTITUTE FOR PLASMA PHYSICS
- FELTOR — DTU
- GBS — EPFL
- GENE — MAX PLANCK INSTITUTE FOR PLASMA PHYSICS

ASCOT5 GPU porting

- **Outline**
  - **ASCOT5: particle orbit-following code**
  - **MPI + OpenMP CPU algorithm**
  - **GPU porting strategy**
  - **Benchmarks**
  - **Profiling**
  - **Conclusion**

ASCOT5

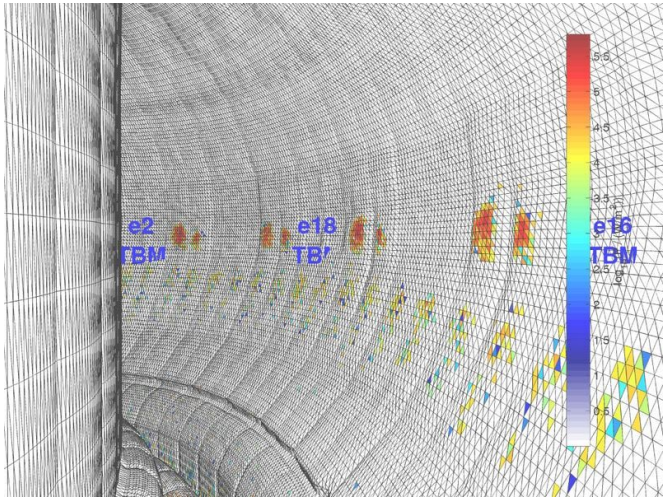
- ASCOT5 is a test **particle orbit-following** code for toroidal magnetically confined fusion devices
- The code uses the **Monte Carlo method** to solve the distribution of particles by following their trajectories.
  - The **evolution of the distribution function** for a test particle species *a* is described by the **Fokker-Planck equation**

$$\frac{\partial f_a}{\partial t} + \mathbf{v} \cdot \nabla f_a + \frac{q_a}{m_a}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \nabla_{\mathbf{v}} f_a = \sum_b -\nabla_{\mathbf{v}} \cdot [\mathbf{a}_{ab} f_a - \nabla_{\mathbf{v}} \cdot (\mathbf{D}_{ab} f_a)]$$

  and **approximated by the Langevin equation** for a large number of markers that represent the distributed function:

$$d\mathbf{z} = [\dot{\mathbf{z}} + \mathbf{a}(\mathbf{z}, t)]\, dt + \boldsymbol{\sigma}(\mathbf{z}, t) \cdot d\boldsymbol{\mathcal{W}}$$

- The particles undergo **collisions with a static Maxwellian background plasma**
- The detailed magnetic fields and the first wall can be **fully 3D**
- **MPI + OpenMP** (task-based) and **highly vectorized**



ASCOT5 CPU version

- CPU: MPI - OpenMP - Vectorized implementation:
  - The time evolutions of each particle are independent from each other
  - One + two levels of parallelism:
  - MPI: Particles distributed among tasks, fields replicated
  - OpenMP: queue based approach
  - highly vectorized using the SIMD, originally developed for KNL manycore systems as target
  - to enable multithreading, a number of worker threads, each operating on a single set of N_{SIMD} arrays, are launched and allowed to perform their simulation independently
  - swapping mechanism
    - after each iteration, particles that have reached their end condition are stored in an array for completed particles
    - a fresh particle is retrieved from a queue to continue simulation in the particular slot in the N_{SIMD} arrays

Algorithm 1: CPU multithread vectorized algorithm

```
initialization;
#pragma omp parallel
while particles are alive in pack_{N_SIMD} do
    #pragma omp simd
    for particles ∈ pack_{N_SIMD} do
        move_particle;
    end
    #pragma omp simd
    for particles ∈ pack_{N_SIMD} do
        collisions;
    end
    #pragma omp simd
    for particles ∈ pack_{N_SIMD} do
        end_condition;
    end
    #pragma omp simd
    for particles ∈ pack_{N_SIMD} do
        diagnostics;
    end
    for particles ∈ pack_{N_SIMD} do
        if particle reached end condition then
            store particle and replace it by new one
        end
    end
end
```

ASCOT5 GPU version

- GPU porting strategy
  - Maintain a single version of the code
  - Ensure code portability and readability
  - Generic pragma for OpenMP/OpenACC

```
#ifndef gpu_commands
#define gpu_commands
/**
 * @brief Applies parallel execution to loops
 */
#if defined(GPU) && defined(OPENMP)
#define GPU_PARALLEL_LOOP_ALL_LEVELS\
        str_pragma(omp target teams distribute parallel for simd
#elif defined(GPU) && defined(OPENACC)
#define GPU_PARALLEL_LOOP_ALL_LEVELSstr_pragma(acc parallel loop)
#else
#define GPU_PARALLEL_LOOP_ALL_LEVELSstr_pragma(omp simd)
#endif


/**
 * @brief Maps variables to the target device
 */
#if defined(GPU) && defined(OPENMP)
#define GPU_MAP_TO_DEVICE(...) \
        str_pragma(omp target enter data map(to: __VA_ARGS__))
#elif defined(GPU) && defined(OPENACC)
#define GPU_MAP_TO_DEVICE(...) str_pragma(acc enter data copyin
(__VA_ARGS__))
#else
#define GPU_MAP_TO_DEVICE(...)
#endif
............

#endif
#endif
```

```
  GPU_LOOP_ALL_LEVELS
  for(i = 0; i < n_queue_size; i++) {
    if(p->running[i]) {
        posxyz[0] = posxyz0[0] + pxyz[0] * h[i] / (2.0 * gamma *
mass);
        posxyz[1] = posxyz0[1] + pxyz[1] * h[i] / (2.0 * gamma *
mass);
        posxyz[2] = posxyz0[2] + pxyz[2] * h[i] / (2.0 * gamma *
mass);
  }
  GPU_END_LOOP_ALL_LEVELS
```

ASCOT5 GPU version

- First implementation History-Based:
  - parallelism is expressed at a high level, emphasizing the independence of individual particles, allowing for concurrent execution of their respective histories from birth to death
  - each GPU processing unit is used to deal with the entire history of one or more particles until all of the particles have reached their end condition
  - this parallelism is implemented through a single monolithic GPU kernel

Algorithm 2: GPU algorithm - History-based

```
initialization;
#pragma acc parallel loop
for all particles ∈ {1...N_tot} do
    while particle is alive do
        move_particle;
        collisions;
        end_condition;
        diagnostics;
    end
end
```

ASCOT5 GPU version

- The original implementation is not GPU-friendly:
  - **one very large kernel**
  - events depend on the previous event
- Implement a new version by **splitting the initial kernel**:
  - **Parallelize over events** instead of particles
  - small kernels independent of each other

Algorithm 2: GPU algorithm - History-based

```
initialization;
#pragma acc parallel loop
for all particles ∈ {1...N_tot} do
    while particle is alive do
        move_particle;
        collisions;
        end_condition;
        diagnostics;
    end
end
```

Algorithm 3: GPU algorithm - Event-based

```
initialization;
while number of particles alive > 0 do
    #pragma acc parallel loop
    for all particles ∈ {1...N_tot} do
        if particle alive then
            move_particle;
        end
    end
    #pragma acc parallel loop
    for all particles ∈ {1...N_tot} do
        if particle alive then
            collisions;
        end
    end
    #pragma acc parallel loop
    for all particles ∈ {1...N_tot} do
        if particle alive then
            end_condition;
        end
    end
    #pragma acc parallel loop
    for all particles ∈ {1...N_tot} do
        if particle alive then
            diagnostics;
        end
    end
end
```

ASCOT5 GPU version

- Implement a new version by **splitting the initial kernel**:
  - **parallelize over events** instead of particles
  - small kernels independent of each other
  - pack particles

Algorithm 2: GPU algorithm - History-based

```
initialization;
#pragma acc parallel loop
for all particles ∈ {1...N_tot} do
    while particle is alive do
        move_particle;
        collisions;
        end_condition;
        diagnostics;
    end
end
```

Algorithm 3: GPU algorithm - Event-based

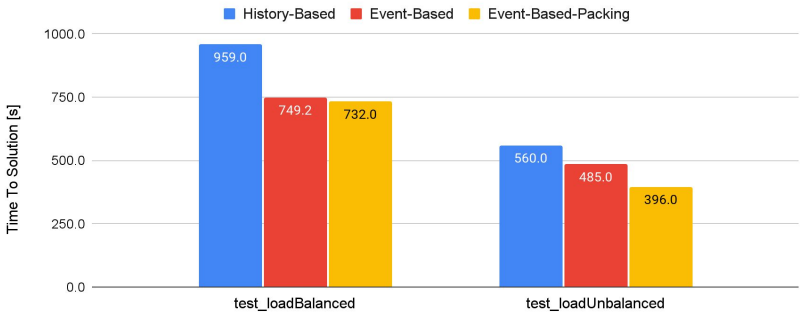
```
initialization;
while number of particles alive > 0 do
    #pragma acc parallel loop
    for all particles ∈ {1...N_tot} do
        if particle alive then
            move_particle;
        end
    end
    #pragma acc parallel loop
    for all particles ∈ {1...N_tot} do
        if particle alive then
            collisions;
        end
    end
    #pragma acc parallel loop
    for all particles ∈ {1...N_tot} do
        if particle alive then
            end_condition;
        end
    end
    #pragma acc parallel loop
    for all particles ∈ {1...N_tot} do
        if particle alive then
            diagnostics;
        end
    end
end
```

Algorithm 4: GPU algorithm - Event-based - packing

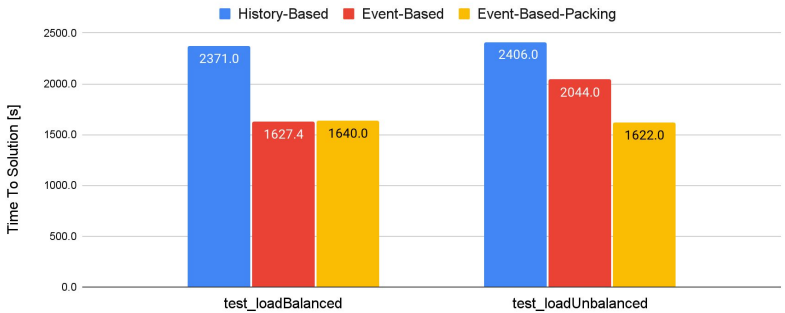
```
initialization;
N_pack ← N_tot;
while number of particles alive > 0 do
    #pragma acc parallel loop
    for packed particles still alive ∈ {1...N_pack} do
        move_particle;
    end
    #pragma acc parallel loop
    for packed particles still alive ∈ {1...N_pack} do
        collisions;
    end
    #pragma acc parallel loop
    for packed particles still alive ∈ {1...N_pack} do
        end_condition;
    end
    #pragma acc parallel loop
    for packed particles still alive ∈ {1...N_pack} do
        diagnostics;
    end
    if (N_pack − N_running > α · N_tot) then
        pack particles;
        N_pack ← N_running;
    end
end
```

Benchmarks

- Benchmark:
  - Collisional full-orbit simulation of prompt-losses of fusion alpha particles
  - 2D wall; ITER-like but circular equilibrium interpolated with cubic splines
  - 2D wall rectangular, coulomb collisions, gyro orbit, simulation time = 0.0001s, fixed time step
  - **Leonardo**: A100, nvhpc/23.1
  - Comparison of three GPU implementations on GPU A100
    - Event-based packing algorithm is most efficient in all cases
    - Impact of Packing:
      - test_loadBalanced: Minimal impact due to majority of particles reaching end of simulation
      - test_loadUnbalanced: Significant impact with speedup of up to 1.41 compared to history-based algorithm and up to 1.22 compared to event-based one.



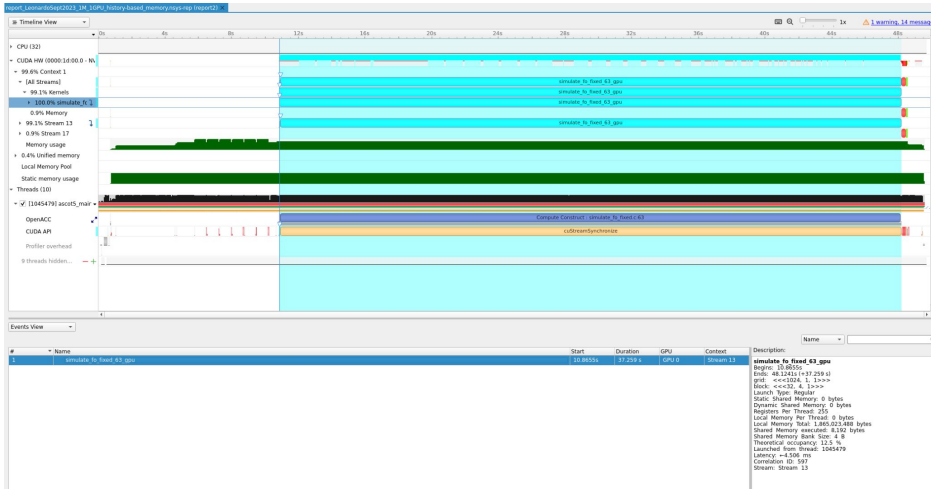
Comparison of the 3 particle-following GPU implementations - 1 Millions markers - 1 A100

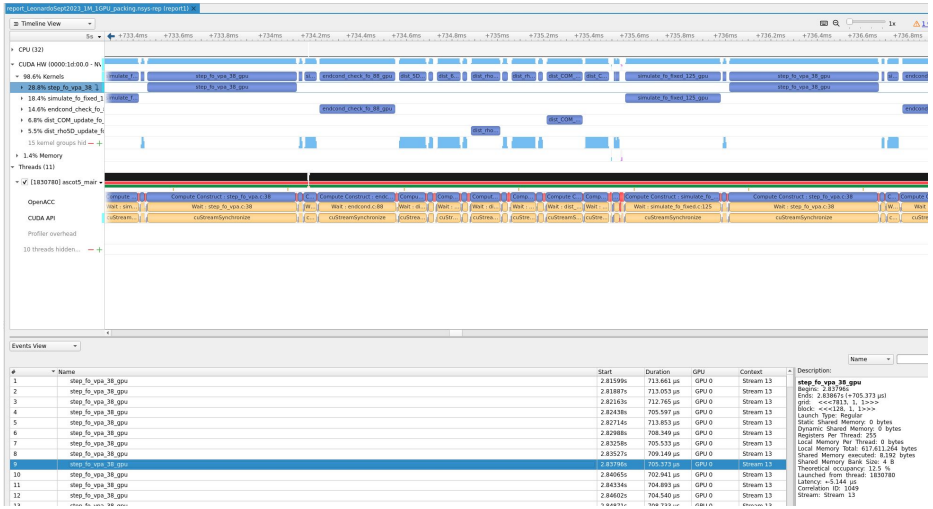


Comparison of the 3 particle-following GPU implementations - 10 Millions markers - 4 A100

Profiling Nsys

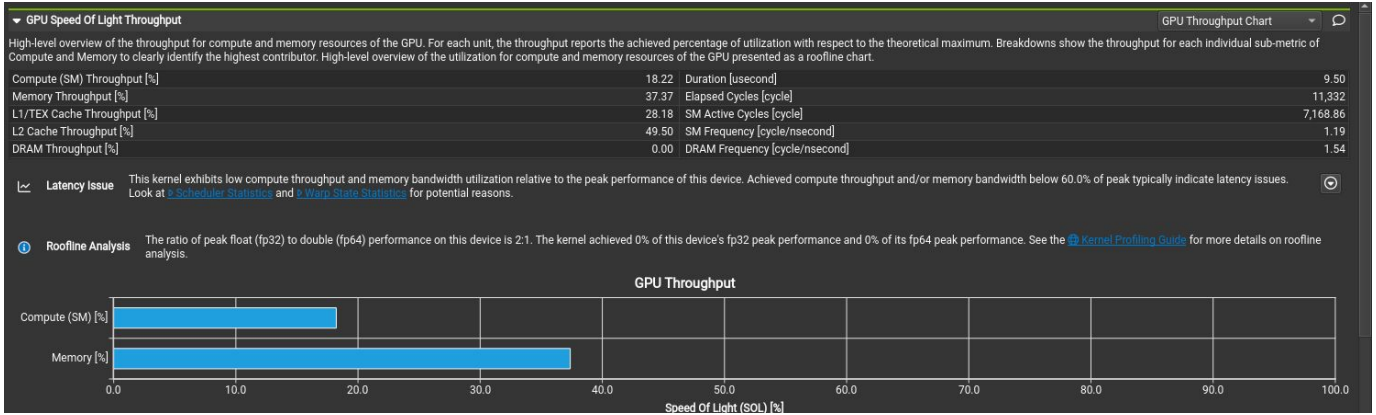
- **Lower Local Memory Use**: Event-based packing uses multiple smaller kernels, reducing local memory demands versus the history-based version.
- **Efficient Data Transfer**: Minimal data transfer overhead as all kernels run on the GPU.
- **Optimized Memory Access**: Contiguous, coalesced memory access through packing enhances efficiency.
- **Reduced Loop Bounds**: Through packing step, dynamic loop bounds improve runtime performance, with only ~30% particles active per timestep.



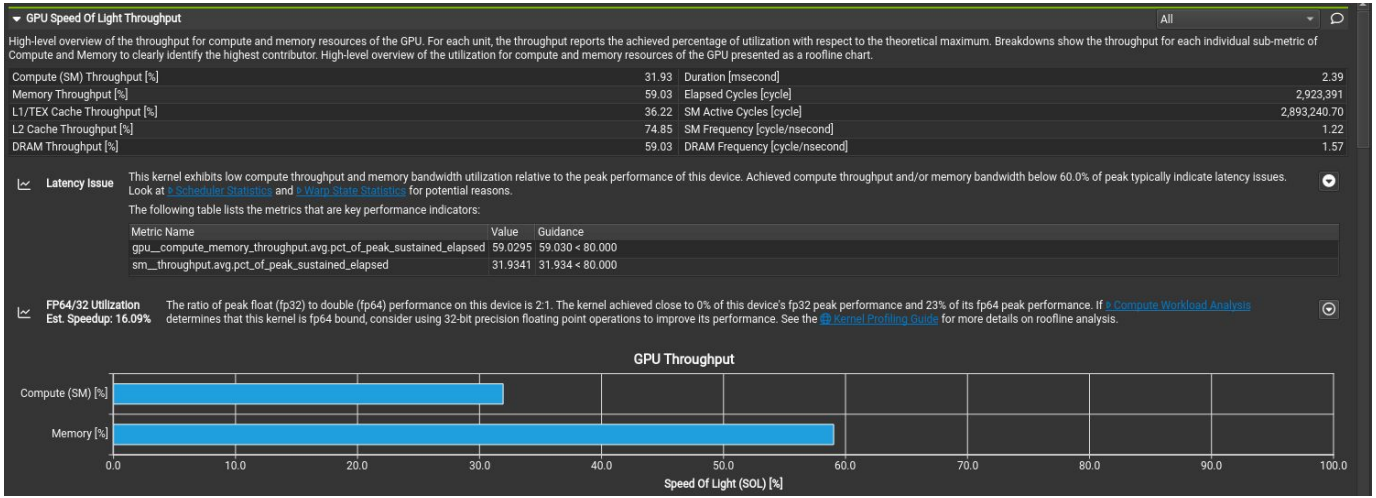


Profiling Nsight-Compute

- HistoryBased

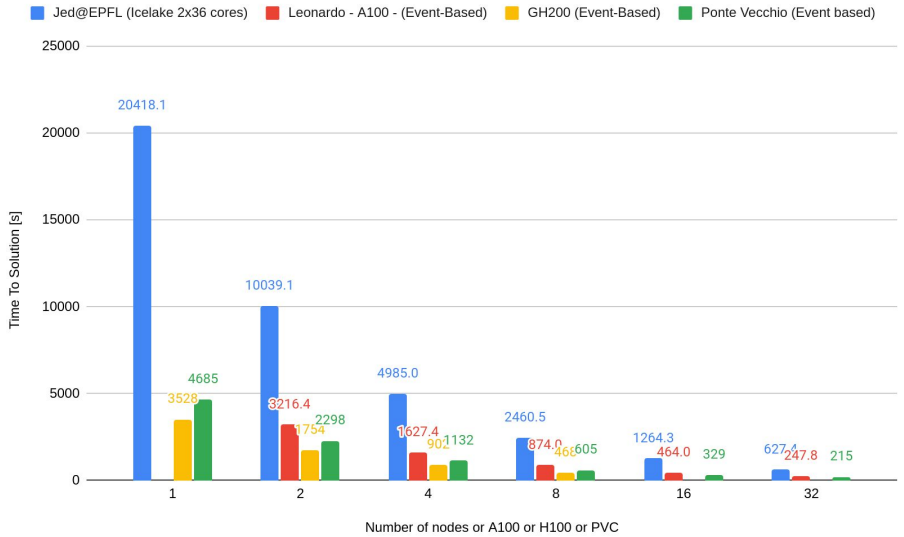


- EventBased



Benchmarks

- 10M markers Benchmark:
  - Collisional full-orbit simulation of prompt-losses of fusion alpha particles
  - 2D wall; ITER-like but circular equilibrium interpolated with cubic splines
  - 2D wall rectangular, coulomb collisions, gyro orbit, simulation time = 0.0001s, fixed time step
  - **Jed**: 2x Platinum 8360Y, intel/2021.6.0
  - **Leonardo**: A100, nvhpc/23.1
  - **NVIDIA Grace Hopper Superchip engineering sample early access courtesy of NVIDIA**
  - **Intel Ponte-Vecchio 600W (2 tiles) engineering sample early access courtesy of INTEL**



Profiling

- EventBased version:
  - kernels mostly memory-bound
  - multiple branch divergences in end_condition kernel involving lower Memory SOL due to thread divergence

Main kernels	*%*
move_particle	64.8
diagnostics	9.6
end_condition	6.5
collisions	5.8
copy_particles_structures	5.5
sorting	< 0.1
packing	< 0.1

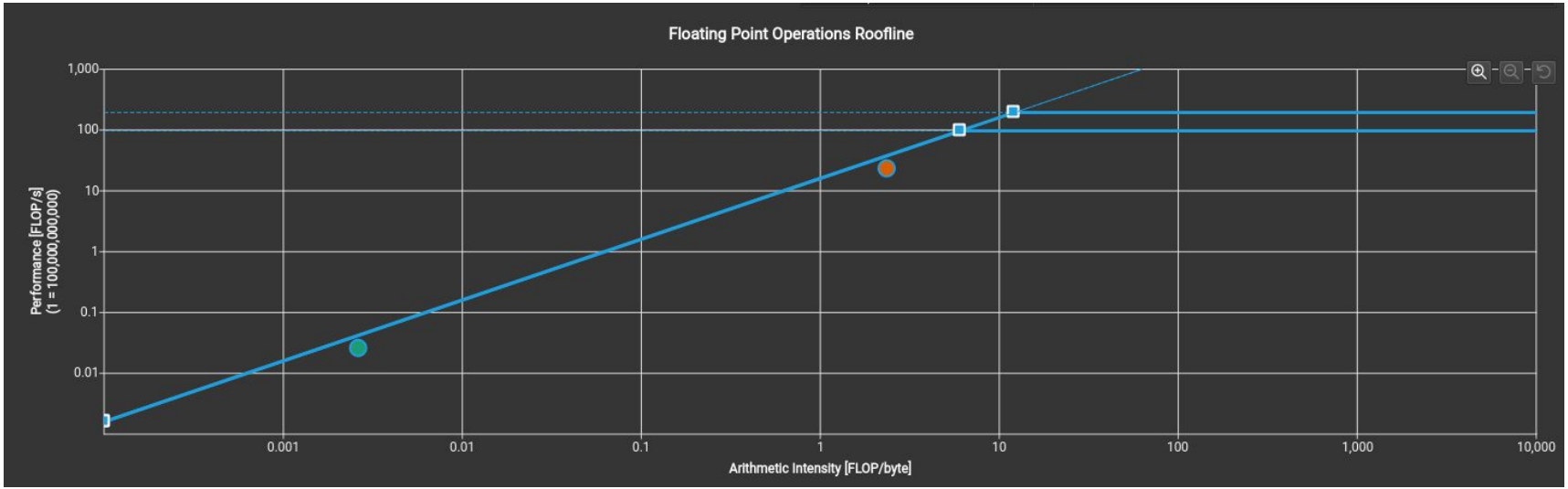
TABLE I. RELATIVE WEIGHTS OF THE DIFFERENT STEPS OF THE SIMULATION ON A100. % VALUES ARE AVERAGED SIMULATING 1 MILLION PARTICLES WITH THE ASCOT5 EVENT-BASED-PACKING ALGORITHM

Main kernels	*Memory SOL (%)*	*Compute SOL (%)*
move_particle	68	30
diagnostics	80	26
end_condition	36	12
collisions	40	56

TABLE II. TEST_LOADBALANCED, SPEED OF LIGHT - 1 MILLION PARTICLES WITH THE ASCOT5 EVENT-BASED-PACKING ALGORITHM

Profiling

- Roofline



Conclusion

- **Successful GPU Transition**: ASCOT5 was efficiently ported from CPU to GPU using a directive-based strategy, ensuring code consistency.
- **Optimized Algorithms**: Three strategies were tested, with event-based-packing achieving the best performance due to improved load balancing and reduced thread divergence.
- **Significant Speedup**: Event-based-packing on H100-96GB shows up to 6x speedup over a dual Intel Xeon CPU node.
- **Future Work**: Conduct new tests incorporating enhanced physical models.