EPFL-ACH

Pr. Paolo Ricci (Academic Director & PI)
Dr. Gilles Fourestey (Operations Director)

EPFL-ACH in a nutshell

SCITAS
Scientific IT and Application Support

Support center for HPC applications and provider of advanced computing platforms (~30 people)

Experimental Museology +

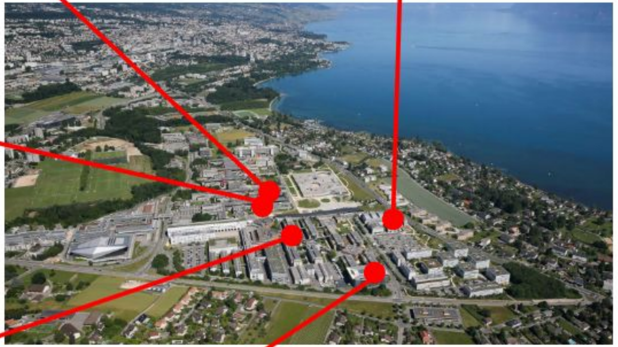
Virtual, augmented, mixed reality, through advanced computer science and state-of-the-art visualization facilities (~10 people)

MATHICSE

Computational Science and Engineering Mathematics group (~70 people)

Swiss Data Science Center, national institute for artificial intelligence and machine learning techniques (~50 people)

Swiss Plasma Center (~ 200 people, theory group: ~ 40 people)

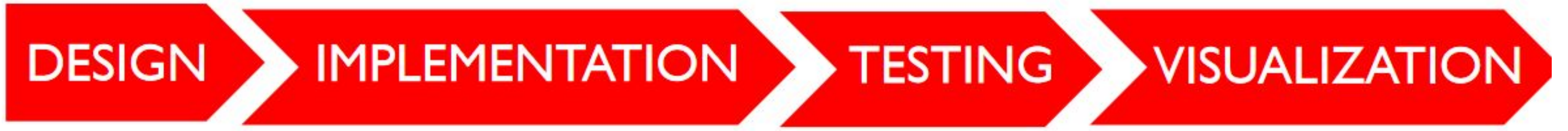


EUROfusion

Swiss Plasma Center

A comprehensive support, from HPC code design to visualization

We are a competence center for
- methods, providing specific support to specific needs
- applications, developing and maintaining EUROfusion software

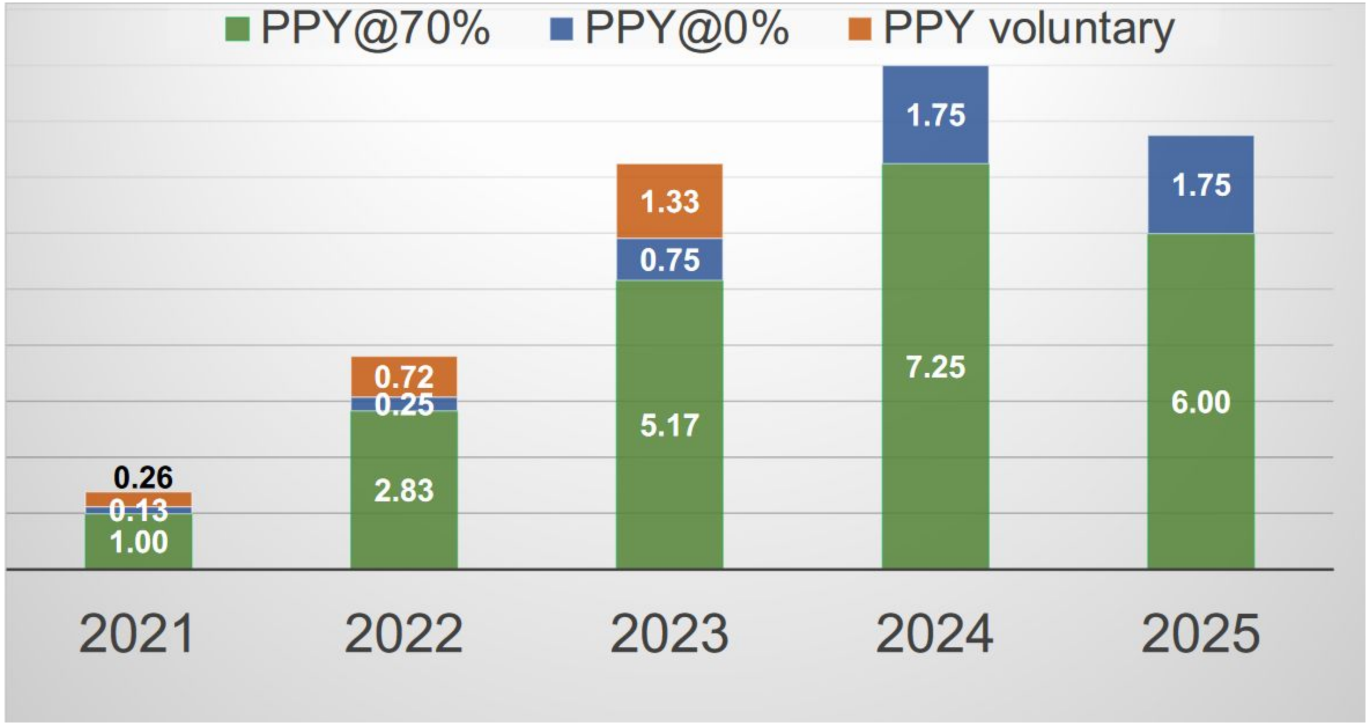


An attractor of new expertise to fusion...

... to an even larger involvement

Swiss Plasma

Large voluntary contribution



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

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

HPC ACH F2F Meeting

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**

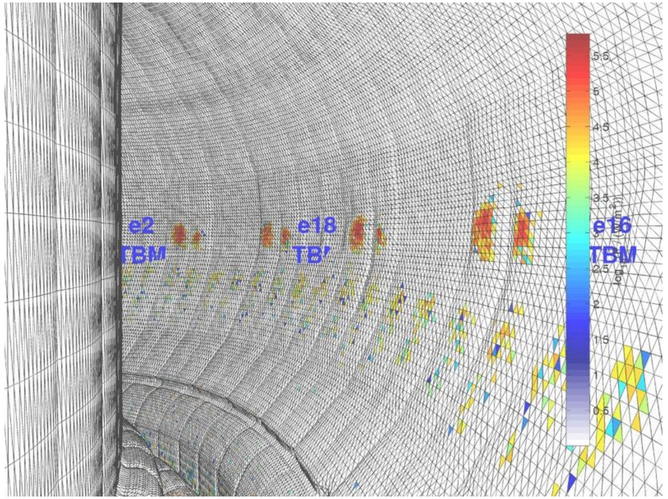
$$\frac{\partial f_a}{\partial t} + \mathbf{v} \cdot \nabla f_a + \frac{q_a}{m_a}(\mathbf{E} + \mathbf{v} \times \mathbf{B}) \cdot \nabla_{\mathbf{v}} f_a = \sum_b -\nabla_{\mathbf{v}} \cdot [\mathbf{a}_{ab} f_a - \nabla_{\mathbf{v}} \cdot (\mathbf{D}_{ab} f_a)]$$

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

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

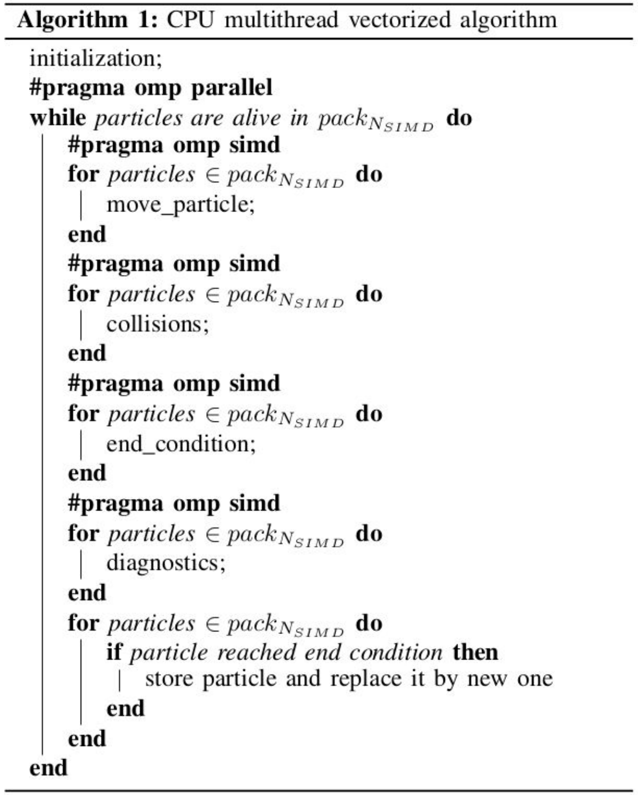
- The particles undergo **collisions with a static Maxwellian background plasma**
- The detailed magnetic fields and the first wall can be **fully 3D**

- **MPI, TLP** (OpenMP, task-based), **DLP (SIMD)**



ASCOT5 CPU version

- MPI - OpenMP - SIMD implementation:
  - The time evolutions of each particle are independent from each other, particles having different lifetimes
  - One + two levels of parallelism:
    - MPI: Particles distributed among tasks, fields replicated
    - OpenMP: queue based approach
    - SIMD: each lane handles a particule during its lifetime (events) 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

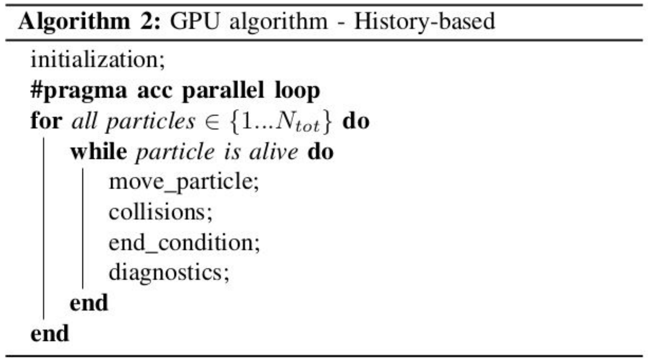


Algorithm 1: CPU multithread vectorized algorithm

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

ASCOT5 GPU version

- First implementation History-Based:
    - parallelism is expressed at a high level, emphasizing the independence of individual particles
    - each GPU thread deals 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 $\in \{1...N_{tot}\}$ do
    while particle is alive do
        move_particle;
        collisions;
        end_condition;
        diagnostics;
    end
end
```

Results:
May2022 Benchmark Comparison

GPU and CPU versions have similar TTS (in general)

ASCOT5	TTS [s]	may2022_2dwall_go_analyticB			
	markers:	10000	100000	Platform	Compiler
m100@CINECA	**OpenMP Offload**	46	473	Power9 + v100	XL compilers
Phoenix@EPFL	**OpenMP Offload**	232	2143	6138 gold + v100	gcc 11
Phoenix@EPFL	**OpenACC**	**48**	**261**	6138 fold + v100	gcc 11
Helvetios@EPFL	**OpenMP**	87	860	2x Gold 6140	intel compilers
Jed@EPFL	**OpenMP**	31	318	2x Platinum 8360Y	intel compilers

ASCOT5 GPU version

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

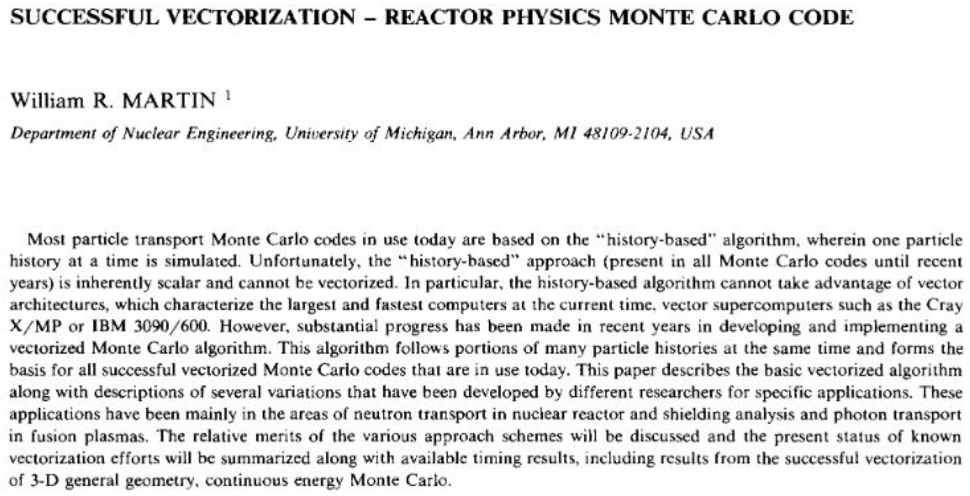
```
#ifndef gpu_commands
#define gpu_commands
/**
 * @brief Applies parallel execution to loops
 */
#if defined(GPU) && defined(OPENMP)
#define GPU_PARALLEL_LOOP_ALL_LEVELS\
        str_pragma(omp target teams distribute parallel for simd)
#elif defined(GPU) && defined(OPENACC)
#define GPU_PARALLEL_LOOP_ALL_LEVELSstr_pragma(acc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

- The original implementation is not GPU-friendly:
  - **one very large kernel** (1000+ threads/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

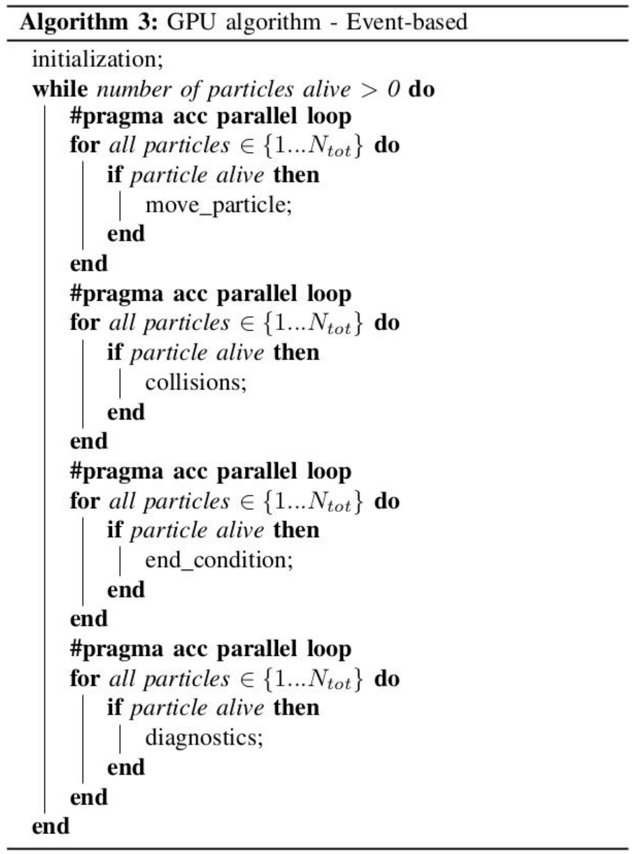


SUCCESSFUL VECTORIZATION – REACTOR PHYSICS MONTE CARLO CODE

William R. MARTIN [1]

Department of Nuclear Engineering, University of Michigan, Ann Arbor, MI 48109-2104, USA

Most particle transport Monte Carlo codes in use today are based on the "history-based" algorithm, wherein one particle history at a time is simulated. Unfortunately, the "history-based" approach (present in all Monte Carlo codes until recent years) is inherently scalar and cannot be vectorized. In particular, the history-based algorithm cannot take advantage of vector architectures, which characterize the largest and fastest computers at the current time, vector supercomputers such as the Cray X/MP or IBM 3090/600. However, substantial progress has been made in recent years in developing and implementing a vectorized Monte Carlo algorithm. This algorithm follows portions of many particle histories at the same time and forms the basis for all successful vectorized Monte Carlo codes that are in use today. This paper describes the basic vectorized algorithm along with descriptions of several variations that have been developed by different researchers for specific applications. These applications have been mainly in the areas of neutron transport in nuclear reactor and shielding analysis and photon transport in fusion plasmas. The relative merits of the various approach schemes will be discussed and the present status of known vectorization efforts will be summarized along with available timing results, including results from the successful vectorization of 3-D general geometry, continuous energy Monte Carlo.



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** to avoid thread divergence and unbalance

Algorithm 2: GPU algorithm - History-based

```
initialization;
#pragma acc parallel loop
for all particles $\in \{1...N_{tot}\}$ do
    while particle is alive do
        move_particle;
        collisions;
        end_condition;
        diagnostics;
    end
end
```

Algorithm 3: GPU algorithm - Event-based

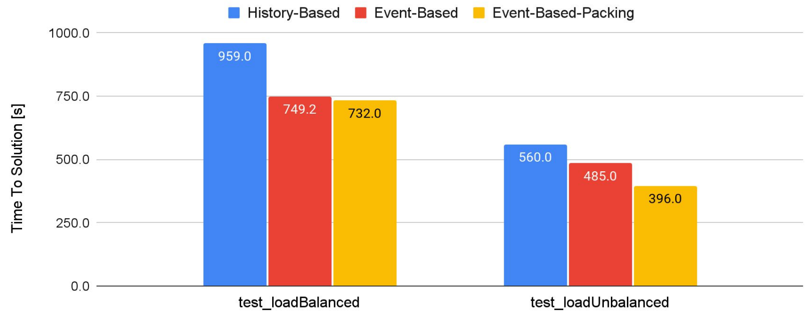
```
initialization;
while number of particles alive $> 0$ do
    #pragma acc parallel loop
    for all particles $\in \{1...N_{tot}\}$ do
        if particle alive then
            move_particle;
        end
    end
    #pragma acc parallel loop
    for all particles $\in \{1...N_{tot}\}$ do
       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

Algorithm 4: GPU algorithm - Event-based - packing

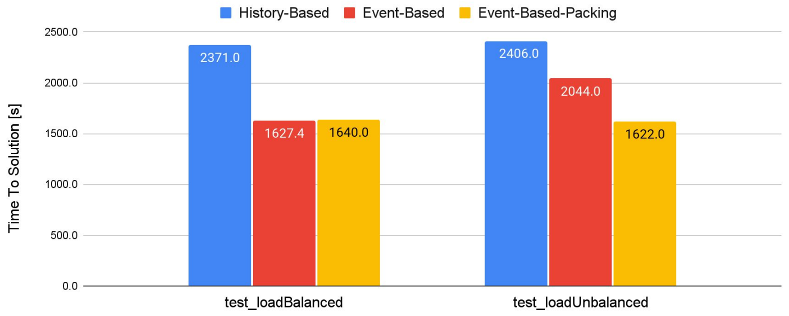
```
initialization;
$N_{pack} \leftarrow N_{tot}$;
while number of particles alive $> 0$ do
    #pragma acc parallel loop
    for packed particles still alive $\in \{1...N_{pack}\}$ do
        move_particle;
    end
    #pragma acc parallel loop
    for packed particles still alive $\in \{1...N_{pack}\}$ do
        collisions;
    end
    #pragma acc parallel loop
    for packed particles still alive $\in \{1...N_{pack}\}$ do
        end_condition;
    end
    #pragma acc parallel loop
    for packed particles still alive $\in \{1...N_{pack}\}$ do
        diagnostics;
    end
    if $(N_{pack} - N_{running} > \alpha \cdot N_{tot})$ then
        pack particles;
        $N_{pack} \leftarrow N_{running}$;
    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.



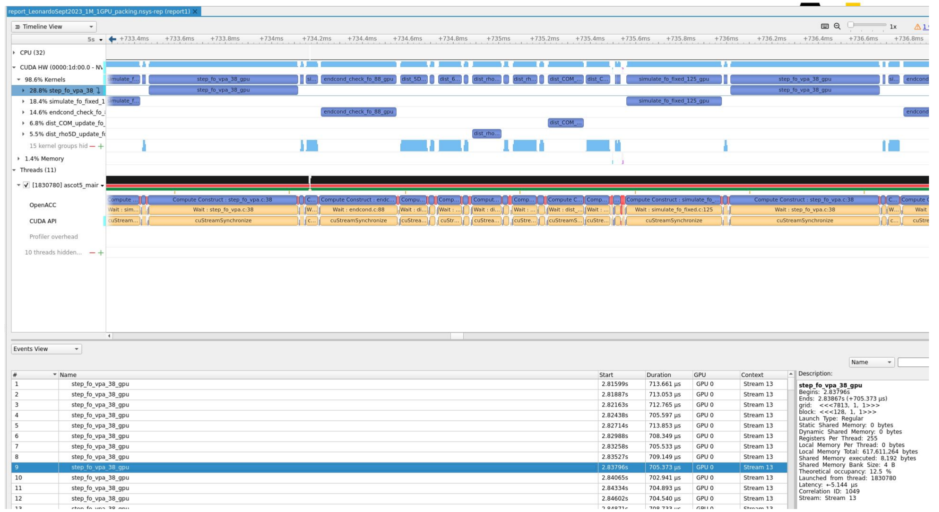
Comparison of the 3 particle-following GPU implementations - 1 Millions markers - 1 A100



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

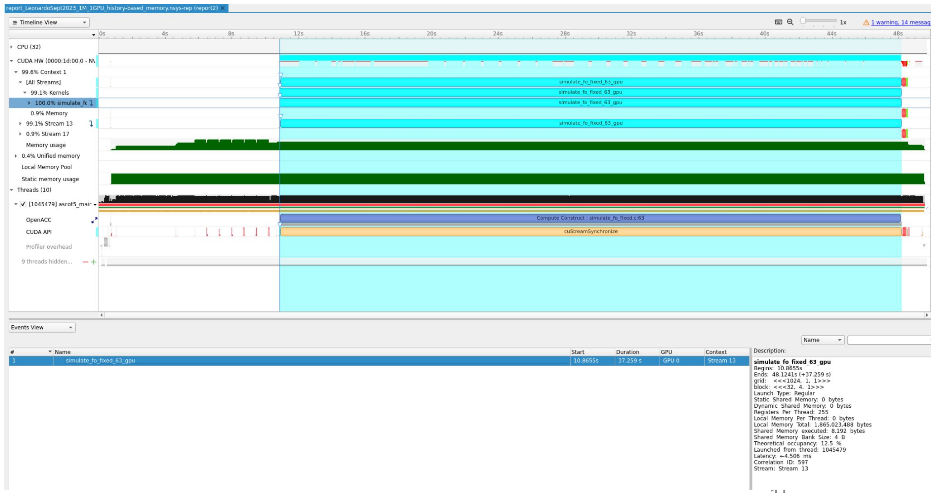
Profiling Nsys

- **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.


report_LeonardoSept2023_1M_1GPU_packing.nsys-rep (report1)

#	Name	Start	Duration	GPU	Context
1	step_fo_vpa_38_gpu	2.81599s	713.661 μs	GPU 0	Stream 13
2	step_fo_vpa_38_gpu	2.81887s	713.053 μs	GPU 0	Stream 13
3	step_fo_vpa_38_gpu	2.82163s	712.765 μs	GPU 0	Stream 13
4	step_fo_vpa_38_gpu	2.82438s	705.597 μs	GPU 0	Stream 13
5	step_fo_vpa_38_gpu	2.82714s	713.853 μs	GPU 0	Stream 13
6	step_fo_vpa_38_gpu	2.82988s	708.349 μs	GPU 0	Stream 13
7	step_fo_vpa_38_gpu	2.83258s	705.533 μs	GPU 0	Stream 13
8	step_fo_vpa_38_gpu	2.83527s	709.149 μs	GPU 0	Stream 13
9	step_fo_vpa_38_gpu	2.83796s	705.373 μs	GPU 0	Stream 13
10	step_fo_vpa_38_gpu	2.84065s	702.941 μs	GPU 0	Stream 13
11	step_fo_vpa_38_gpu	2.84334s	704.893 μs	GPU 0	Stream 13
12	step_fo_vpa_38_gpu	2.84602s	704.540 μs	GPU 0	Stream 13

```
Description:
step_fo_vpa_38_gpu
Begins: 2.83796s
Ends: 2.83867s (+705.373 μs)
grid:  <<<7813, 1, 1>>>
block: <<<128, 1, 1>>>
Launch Type: Regular
Static Shared Memory: 0 bytes
Dynamic Shared Memory: 0 bytes
Registers Per Thread: 255
Local Memory Per Thread: 0 bytes
Local Memory Total: 617,611,264 bytes
Shared Memory executed: 8,192 bytes
Shared Memory Bank Size: 4 B
Theoretical occupancy: 12.5 %
Launched from thread: 1830780
Latency: ←5.144 μs
Correlation ID: 1049
Stream: Stream 13
```


report_LeonardoSept2023_1M_1GPU_history-based_memory.nsys-rep (report2)

#	Name	Start	Duration	GPU	Context
1	simulate_fo_fixed_63_gpu	10.8655s	37.259 s	GPU 0	Stream 13

```
Description:
simulate_fo_fixed_63_gpu
Begins: 10.8655s
Ends: 48.1241s (+37.259 s)
grid:  <<<1024, 1, 1>>>
block: <<<32, 4, 1>>>
Launch Type: Regular
Static Shared Memory: 0 bytes
Dynamic Shared Memory: 0 bytes
Registers Per Thread: 255
Local Memory Per Thread: 0 bytes
Local Memory Total: 1,865,023,488 bytes
Shared Memory executed: 8,192 bytes
Shared Memory Bank Size: 4 B
Theoretical occupancy: 12.5 %
Launched from thread: 1045479
Latency: ←4.506 ms
Correlation ID: 597
Stream: Stream 13
```

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

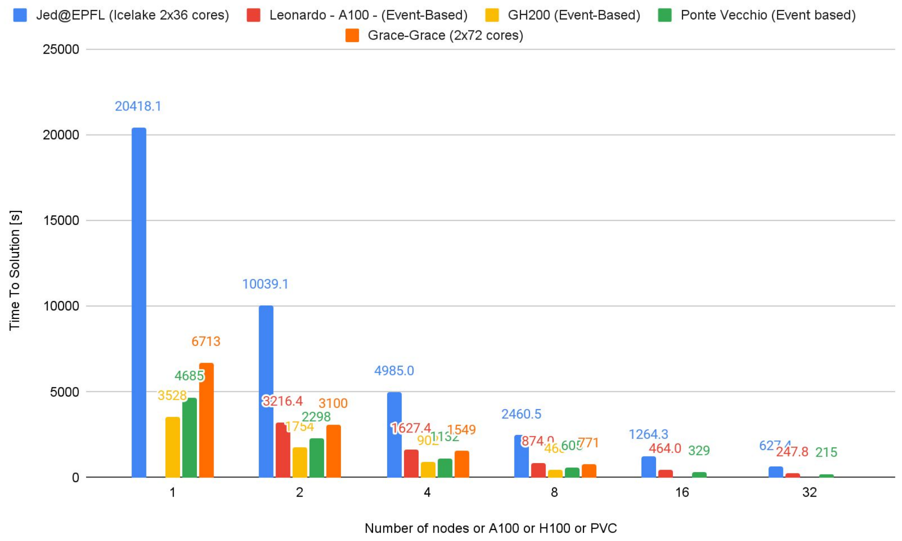
TABLE I. Relative Weights Of The Different Steps Of The Simulation On A100. % Values Are Averaged Simulating 1 Million Particles With The Ascot5 Event-Based-Packing Algorithm

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

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**
  - **NVIDIA Grace-Grace**
  - **Intel Ponte-Vecchio 600W engineering sample early access courtesy of INTEL**



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.
- ACOT5-GPU is now fully imported into the master version
- Several groups have started using it
- Future Work: Conduct new tests incorporating enhanced physical models.