EUROfusion

SPICE2d – ALYA – XTAPAS – XTORK

Alejandro Soba, Marc Vittolo, Joan Vinyals

HPC ACHs | 27-28 October 2026

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). 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 Commission. Neither the European Union nor the European Commission can be held responsible for them.

SPICE2

- **SPICE** (SheathParticle In CEll) package includes two codes: SPICE2 (2D3V) and SPICE3 (3D3V).
- PIC code for simulations of particles in a fixed magnetic and self-consistent electric field
- Written in Fortran 90, outputs in the Matlab MAT binary format. Right now, several python files are written as alternative.
- Parallelization implemented using domain decomposition principles and message passing interface (MPI).
- Now all internal routines are parallel or have the alternative to used a parallel subroutine.
- Bests solvers are SPARCE, based in UMFPACK and PETSC.

SPICE2

Requirements for 2026

- Include OPENMP in collision operators;
- Tests its performance in PITAGORA and MN5;
- Analize the possibility to portability of this subroutines to GPU.

Coulomb Collision operator

All are in the ***mod_collision.f90*** module.

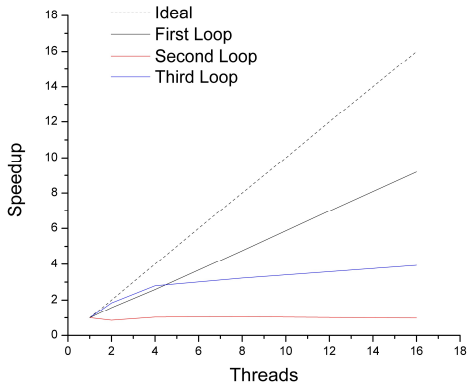
The most important is the ***coulomb_collision()*** subroutine.

Furthermore, the *cx_collisions_fluid()* subroutine handles charge-exchange collisions with a uniform "fluid" background of neutral particles, while the *di_collisions_fluid()* subroutine handles ionization collisions, also using a uniform "fluid" background of neutral particles.

SPICE2

coulomb_collision(): 3 main loops

-) First loop: Nothing particular to mention.
It's the best loop because go by all the particles.
Just need an ATOMIC call to order the write into the cell ar



Speedup of the three loops in coulomb_coliision() *subroutine. This patron is repeated in all cases*

-) The second and third loops are very difficult to optimize.
Three loops that run by grid and species. We introduce a schedule, dynamic call to avoid disbalance between the grids.
The number of the assigned dynamic task loads are now an input parameter. *Ndinamic*. The default value is 1000.

We fix repetition of code depending on the parity of the number of particles by portion of domain.

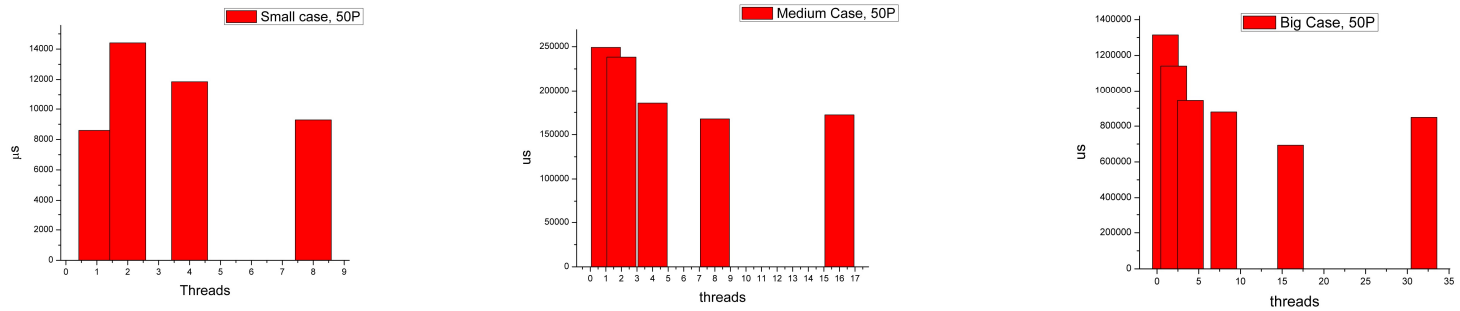
SPICE2

Inputs file: ***collisions.inp*** uses a grid of zxy (250x10) and Np=50 as key particle parameter.

We call this configuration *Small_case*. To increase more demanding cases, we create the *medium_case* (1250x50) and the *big_case* (2500x100) using Np=50, 100 and 200 to evaluate efficiency and performance.

In the case of cx_collisions_fluid() and di*_collisions_fluid(). We introduce Openmp un the loops of the first one. Not in the second one because exist creation of particles.*

SPICE2

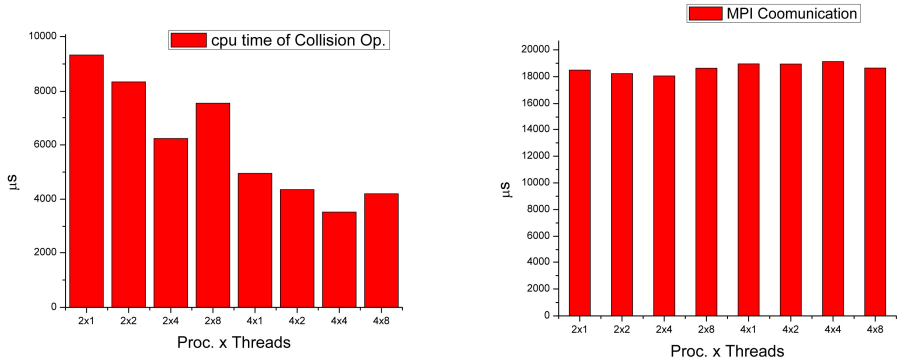


Comparison between results of the code in the small, medium and big domain with 50P.

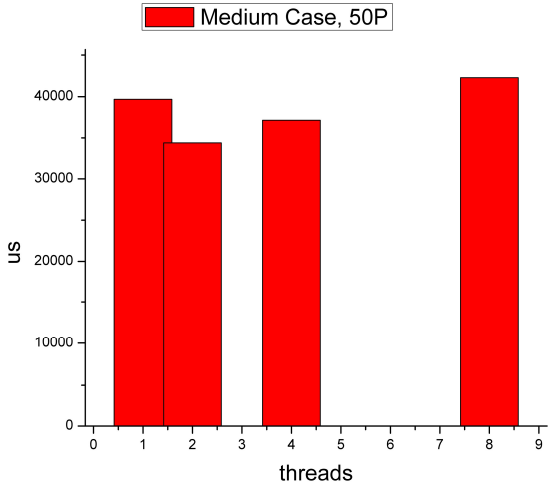
threads	50 p (μs)	100 p (μs)	200 p (μs)
1	138903	241868	468735
2	126556	224504	349642
4	99866	186687	276445
8	83174	163585	270784

Different number of particles that put a limit on the maximum number of particles allowed by cells. In the next table we show the collision operator times in microseconds of each case (50, 100 and 200) particles. We see the same patron, with a better Speedup when the particles increase, that shows that when we have more work by cell, we increase the efficiency of add more threads.

SPICE2



Cpu time of collision operator and MPI time communication for several hybrid runs.

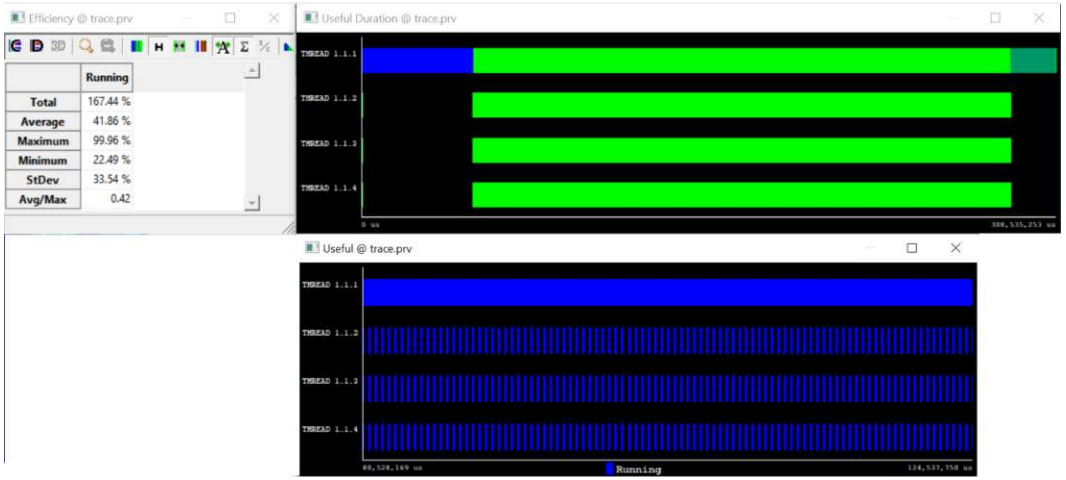


Cpu time of collision operator in PITAGORA cluster.

SPICE2

Paraver analysis

The BSC tools Paraver (https://tools.bsc.es/paraver) and Extrae (https://tools.bsc.es/extrae) used to analyze threads performance, memory consumption and communications environments.



Spice2 traces where we can verification the Openmp threads distribution when collision operator is called.

	1	2	4	8	16
Global efficiency	99.98	51.15	29.04	15.25	6.21
-- Parallel efficiency	99.98	65.01	46.47	33.39	25.36
-- Load balance	100.00	65.04	46.49	33.40	25.37
-- Communication efficiency	99.98	99.97	99.96	99.96	99.95
-- Serialization efficiency					
-- Transfer efficiency					
-- Computation scalability	100.00	78.68	62.49	45.67	24.50
-- IPC scalability	100.00	59.42	34.68	23.16	11.94
-- Instruction scalability	100.00	214.07	226.93	218.82	204.01
-- Frequency scalability	100.00	82.88	42.69	104.90	116.64

Percentage(%): 100, 80, 60, 40, 20, 0

Conclusions

i) Poor efficiency in the OpenMP implementation in *coulomb _collision* operator, many deb to the second loop. Exist imbalance into the grid (that we try to avoid using schedule, dynamic) and the most probably case of file sharing, when multiple threads write to nearby memory locations, and constantly invalidate caches.

ii) The implementation of OpenMP in architecture like MN5 and PITAGORA cancel the vectorization with SIMD. In these architectures a single core might be running AVX-512 very efficiently, while multiple threads compete for memory bandwidth.

iii) However, the efficiency is good in the case of the loop over all the particles (first loop on *coulomb_collision* and in the fluid subroutines), this is a clue that the problem of file sharing can be the reason of the bad behavior of the other loops.

iv) A change in the creation of the *grids_d%cell_indices_count* and in the *cell-indices* arrays can help with this. This point needs to be discussed with the developers.

v) The porting these subroutines to GPU should not be difficult but to do that is necessary to put the particles information into the GPU from the beginning of the calculation.

ALYA (Marc Vittolo and Ezequiel Goldberg)

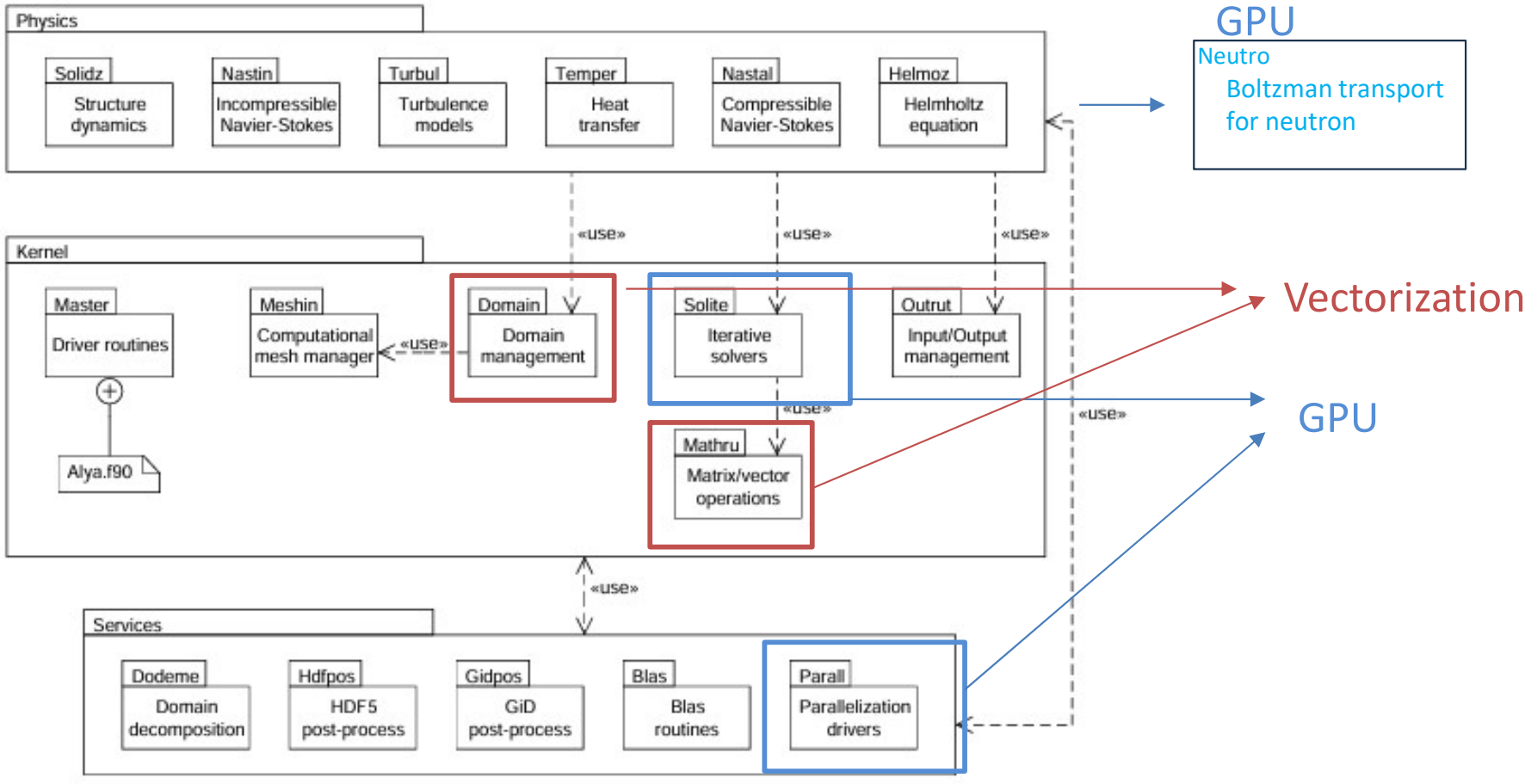
Alya is a finite element framework created and actively developed by the Physical and numerical Modelling group within the Computer Applications in Science and Engineering (CASE) department of BSC. It implements MPI parallelization and domain subdivision in its kernel to leverage using several supercomputer nodes simultaneously.

Code developers' requirements:

-) Porting NEUTRO module to GPU

-) Optimizations

ALYA


Physics
Solidz
Structure dynamics
Nastin
Incompressible Navier-Stokes
Turbul
Turbulence models
Temper
Heat transfer
Nastal
Compressible Navier-Stokes
Helmoz
Helmholtz equation
GPU
Neutro
Boltzman transport for neutron
«use»
«use»
«use»
Kernel
Master
Driver routines
Alya.f90
Meshin
Computational mesh manager
«use»
Domain
Domain management
Solite
Iterative solvers
Outrut
Input/Output management
Vectorization
«use»
Mathru
Matrix/vector operations
GPU
«use»
«use»
Services
Dodeme
Domain decomposition
Hdfpos
HDF5 post-process
Gidpos
GiD post-process
Blas
Blas routines
Parall
Parallelization drivers

Alya

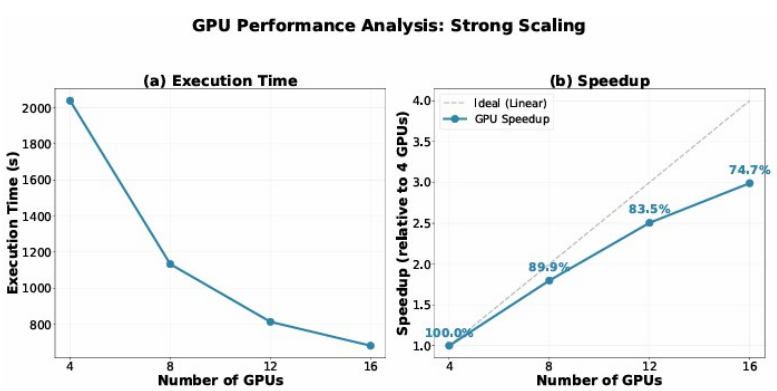
Algorithm 1 NEUTRO general algorithm

Input: Spatial mesh, material properties (Σ_t, Σ_s), external sources S
General initializations
while global convergence criteria not met **do**
  for $g = 1$ **to** G **do** ▷ Loop over energy groups
    for $m = 1$ **to** M **do** ▷ Loop over discrete directions
      Iteration initializations
      for $ielem \leftarrow 1, nelements$ **do** ▷ Loop over finite elements
        Element gather operations
        Compute coefficients (volumetric sources, flux, scattering)
        Element assembly
        Global matrix assembly
      end for
      for $iboun \leftarrow 1, nboundaries$ **do** ▷ Loop over domain boundaries
        Boundary gather operations
        Element assembly from boundary nodes (sources, reflective)
        Global matrix assembly
      end for
      Algebraic solver
      Update results for g, m iteration
    end for
  end for
  Check global convergence across all $G \times M$ degrees of freedom
end while
Post-processing: Calculate derived variables
Write output files

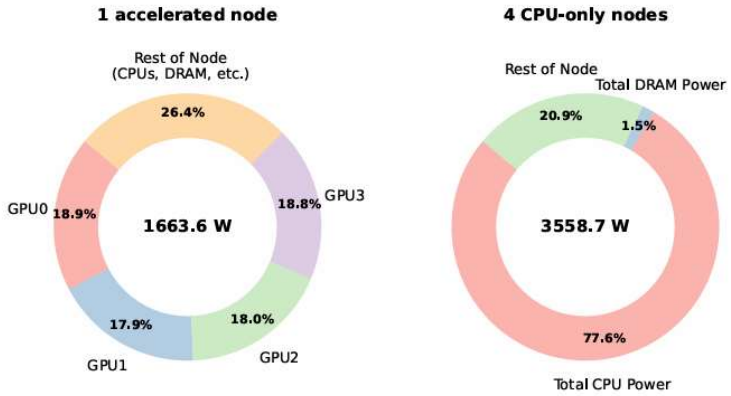
- Initial GPU profiling revealed bottlenecks caused by monolithic OpenACC kernels and inefficient memory access patterns.
- Kernel decomposition split large computations into smaller kernels, reducing register pressure and exposing finer-grained parallelism. (execution time from 21.62 ms to 17.9 ms, a 17.5% improvement).
- Loop reordering was applied to promote stride-one traversal and improve memory locality and cache efficiency.
- arrays multidimensional were restructured so the innermost loop variable became the fastest-varying dimension in Fortran memory layout.
- This enabled coalesced GPU memory accesses, reducing global-memory traffic and improving data-fetch efficiency. (execution time from 17 min 24 s to 15 min 7 s (13%))
- Scattering calculations were also restructured by precomputing invariant terms outside inner loops, reducing unnecessary floating-point operations.
- Overall execution time decreased from 15 min 7 s to 5 min 57 s, a 60.6% improvement and the largest performance gain achieved.

Alya

- Strong scaling results: parallel efficiency degrading to 74.8% as the number of GPUs increases to 16.
- Cause: insufficient work per GPU. Global domain is subdivided and with 16 processes, each GPU receives roughly 366K elements.
- Not enough concurrent element assembly and gather operations to hide global memory latency and fully saturate the massive Streaming Multiprocessor (SM) architecture of the NVIDIA Hopper GPUs.
- To evaluate the energy efficiency of the accelerated solver, power consumption was monitored using the Energy Aware Runtime (EAR) software (https://lenovopress.lenovo.com/lp2541-optimizing-power-energy-and-performance-in-hpc-and-ai-data-centers-with-ear)



a) Execution time varying the number of processes/GPUs; b) Speedup relative to the time taken with 5 MPI processes and 4 GPUs



Time-averaged total power used (W) from the Energy Aware Runtime (EAR) for one accelerated node (left, 4 GPUs) and 4 CPU-only nodes (right).

Alya

Conclusions

- **3.58× speedup** achieved with the optimized OpenACC GPU implementation.
- Kernel decomposition, memory optimization, coalesced access, and scalarization improved key components by **24–92%**.
- GPU performance depends on maintaining a sufficiently high **element-to-GPU ratio**.
- GPU execution reduced total **energy consumption by 62%** compared with the CPU implementation.
- The iterative solver is now the main bottleneck (**~60%** of runtime), motivating future **solver and multi-GPU optimization**.

XTAPAS (Marc Vittolo)

First version

eXtended Toroidal Accelerated PArticle Simulator (XTAPAS) is a hybrid simulation code conceived to describe the interaction between macroscopic magnetohydrodynamic (MHD) activity and energetic particles in magnetically confined plasmas. The model couples a reduced MHD solver to a full-F, full-orbit kinetic module (the previously TAPAS code) that follows the energetic particle trajectories without perturbative assumptions.

boris_push_particles kernel →96%



Time	Total Time	Instances	Avg	Med	Min	Max	StdDev	Name
96.8%	527.823 s	5000	105.565 ms	109.766 ms	52.997 ms	139.158 ms	26.102 ms	boris_push_particles_238_gpu
3.1%	16.976 s	251	67.634 ms	67.677 ms	64.271 ms	69.156 ms	525.356 µs	fluid_moments_compute_fluid_moments_41_gpu
0.1%	378.132 ms	26	14.544 ms	15.088 ms	10.500 ms	15.602 ms	1.299 ms	diagnostics_2d_compute_densrth_27_gpu
0.0%	31.147 ms	26	1.198 ms	1.191 ms	1.171 ms	1.314 ms	30.170 µs	diagnostics_2d_compute_f2d_112_gpu
0.0%	9.309 ms	26	358.025 µs	343.872 µs	331.201 µs	572.864 µs	47.492 µs	diagnostics_2d_compute_densbtbz_70_gpu

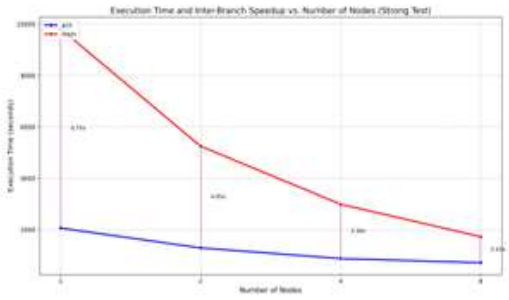
XTAPAS

First version

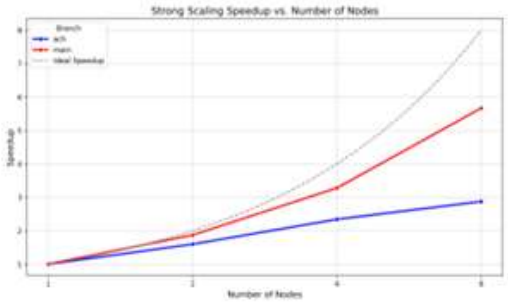
Kernel Name	Time % of execution	Instances	Average time	Total time
boris_push_particles_430_gpu	90.1%	400	352.3 ms	140.9 s
boris_push_particles_359_gpu	7.0%	400	27.4 ms	11.0 s
fluid_moments_compute_fluid_moments_42_gpu	2.4%	11	345.3 s	3.8 s
boris_push_particles_416_gpu	0.3%	400	1.2 ms	477.3 ms
diagnostics_2d_compute_densrth_27_gpu	0.1%	2	102.7 ms	205.3 ms

Table 3: Data obtained by profiling test case test_JET_99896 doing 10 iteration in 1 full node (80 CPUs and 4 GPUs).

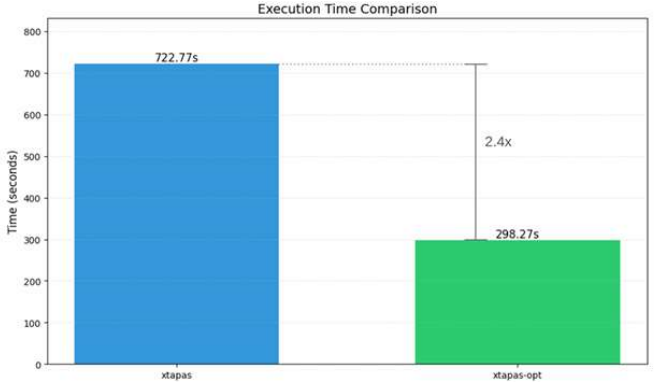
- Initial OPENACC implementations was improved
- Main loop reordered
- Change from 4D arrays to 1D arrays
- The changes was actualized in the new version of the code.



(a) Execution time scaling

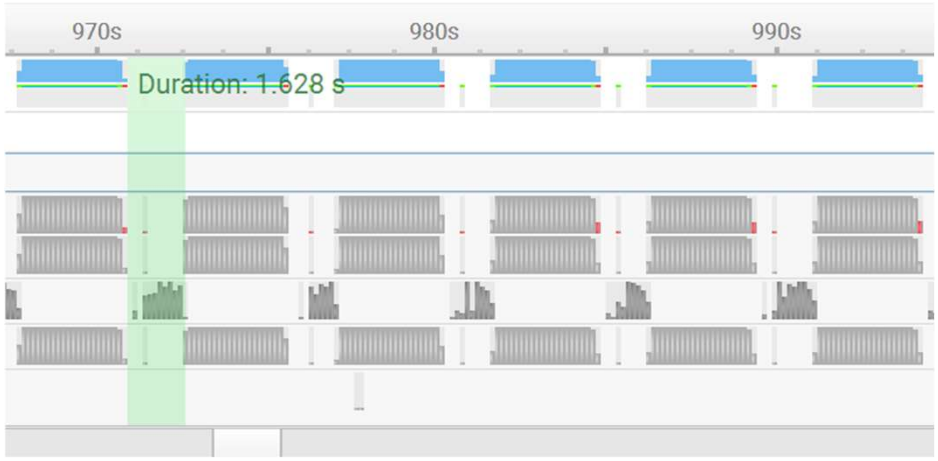


(b) Speedup

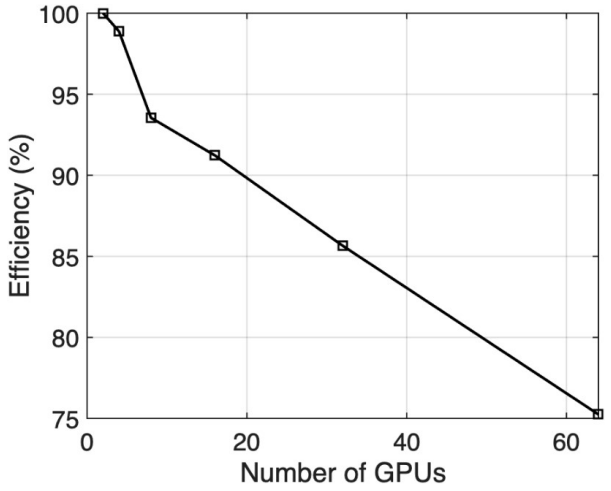


Execution time before and after optimization (as measured for: 25 iterations; 32 nodes; 4 GPUs per node)

Second version, new inputs

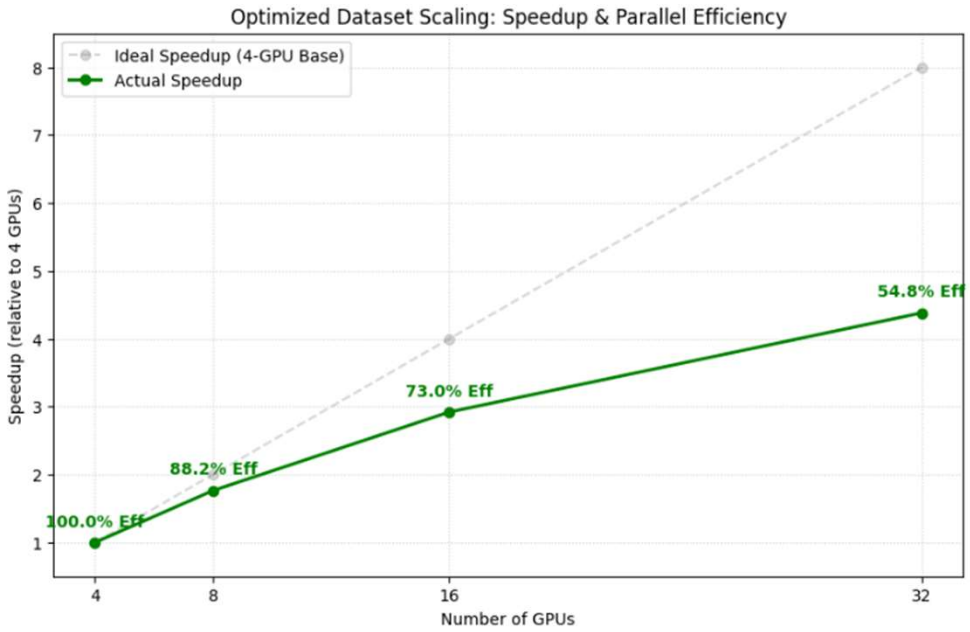




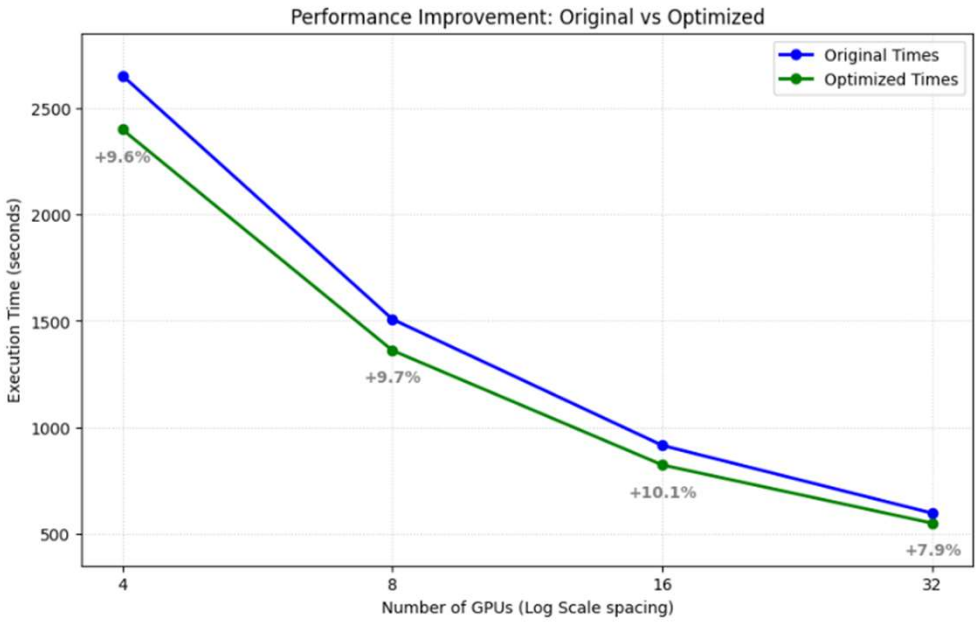


Jean Zay **HPE SGI 8600** part and a **Bull BullSequana XH3000** part, forming a total of five compute partitions: one partition containing cpu nodes and four partitions containing accelerated nodes. GPU NVIDIA V100/A100

Second version in MN5



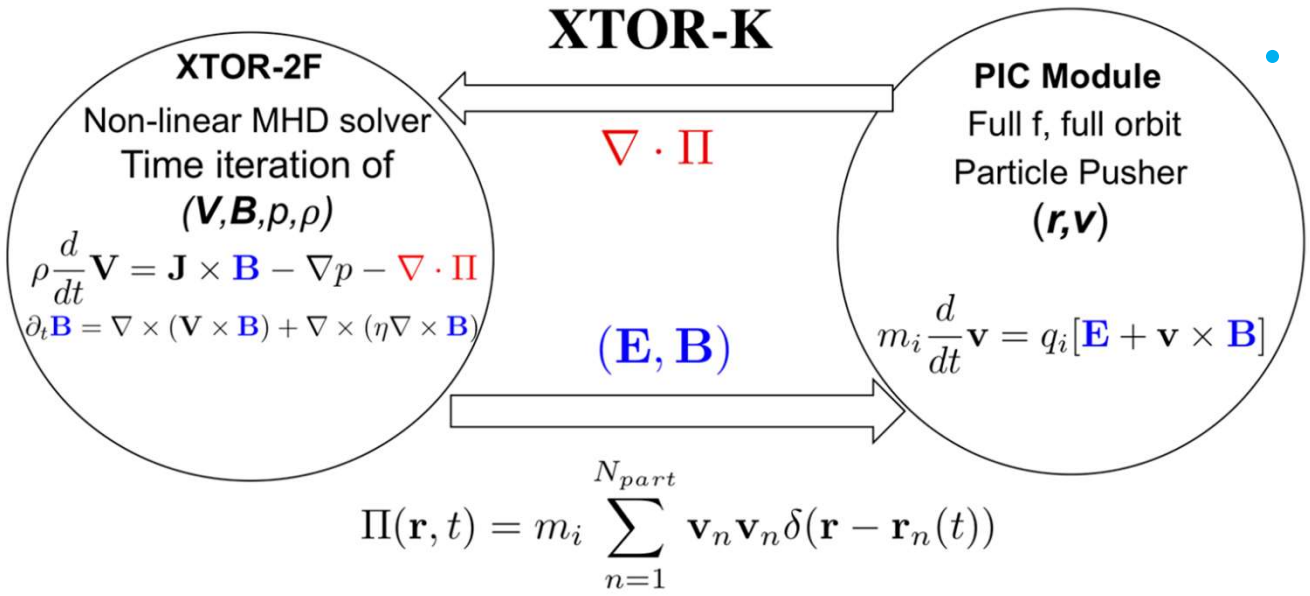
Speedup strong scaling of the new optimized version.



Execution Time

Xtor-k (Eduardo Cabrera)

Hybrid time evolution of global instabilities in magnetically confined tokamak fusion plasmas including 2-fluid and kinetic effects



- GPU-2025: The GPU porting of the kinetic routines has enabled improved performance and, critically, the capability to simulate particle numbers on the order of 100 million per GPU

Xtor-k

But… in the mind time… New version of Xtor-k! Right now, we are in train to received it.

- **Memory deallocation errors**: The *compute-sanitizer* tool reported approximately 30 missing deallocations, which can be addressed through routine corrections. This is mainly for the CPU part. However, it could create some portability issues. Done
- **Argument-passing inconsistency**: A more critical issue involves subroutine interfaces where a three-dimensional array is passed and received as a four-dimensional array with incompatible shapes. In Fortran, explicit shape matching is required unless assumed-shape arrays are used. Both the number of dimensions and their extents must match when passing arrays. Reinterpreting memory is possible but error-prone and should only be attempted with care.
- **Clarification** has been requested from the developers regarding whether this approach was intentional, potentially as a performance optimisation or as part of a storage layout strategy. At present, this inconsistency constitutes the primary obstacle to completing the GPU port.
- **Porting FFT subroutine**. This task requires the integration of NVIDIA's cuFFT and cuBLAS libraries.

General Conclusions

- The work with SPICE2 is close to reach a final point. The requirements of developers were done and the recommendation of ACH-BSC almost done (the final decision need to be taken by the developers.)
- Almost all the NEUTRO module into Alya was porting to GPU. Some optimization in the solver are needed.
- We tests two versions of XTAPAS and optimized the fluid_xx subroutine obtaining efficiencies of double of the initial tests.
- We suffer a delayed in Xtor-k code, we starting in September. We are waiting the last version of the code, but we initiated the GPU analysis of the fluid subroutines.

EUROfusion

SPICE2d – ALYA – XTAPAS – XTORK

Many Thanks!

HPC ACHs | 27-28 October 2026

This work has been carried out within the framework of the EUROfusion Consortium, funded by the European Union via the Euratom Research and Training Programme (Grant Agreement No 101052200 — EUROfusion). 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 Commission. Neither the European Union nor the European Commission can be held responsible for them.