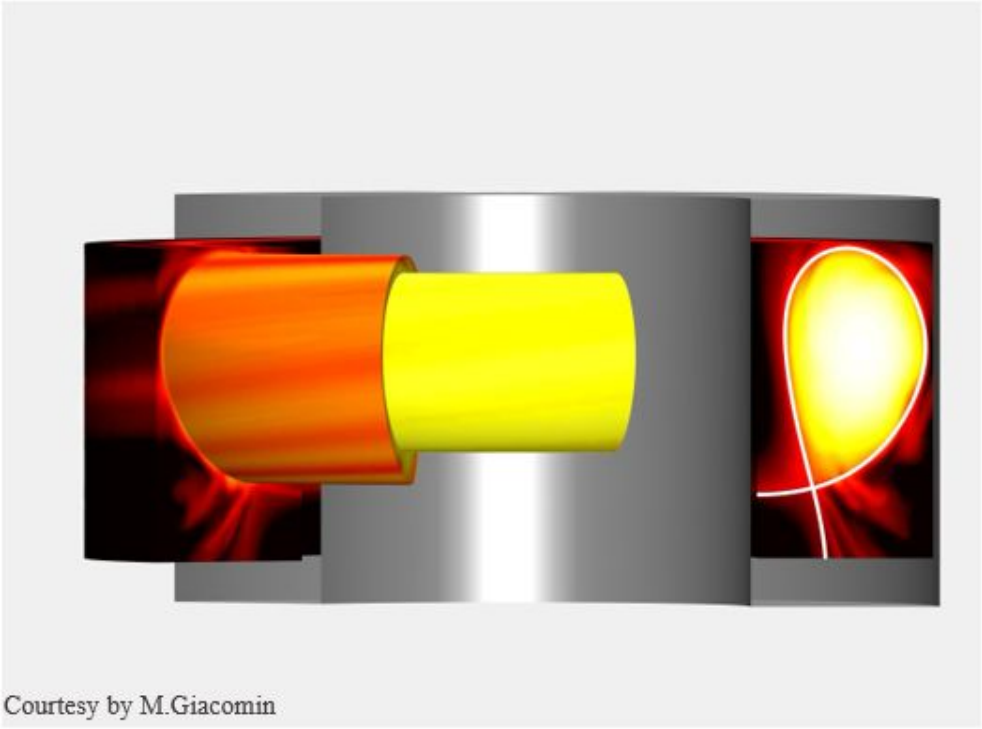
Implicit Solver Performance: Bottlenecks and optimizations in GBS

Goal: Improve numerical solution of the two-fluid plasma model



Courtesy by M.Giacomin

Numerical Challenges

- highly anisotropic turbulent phenomena
- multiple time scales
- presence of stiff and non-stiff terms

We investigate:

- **Implicit integration** of fast-oscillating phenomena
- advanced discretization methods to improve **numerical stability**

Two-fluid turbulent plasma model

No electromagnetic effects

Density

$$\frac{\partial n}{\partial t} = -\frac{\rho_*^{-1}}{B}[\phi, n] + C(p_e, \phi, n) - \nabla_\parallel(nV_{\parallel e}) + D_n\nabla_\perp^2 n + s_n$$

Vorticity

$$\frac{\partial \Omega}{\partial t} = -\frac{\rho_*^{-1}}{B}\nabla\cdot[\phi, n\nabla_\perp\phi] - \nabla\cdot(V_{\parallel i}\nabla_\parallel(n\nabla_\perp\phi)) + B^2\nabla_\parallel(j_\parallel) + D_\Omega\nabla_\perp^2\Omega + C(p_e, p_i) + s_\Omega$$

Electron Parallel Velocity

$$\frac{\partial V_{\parallel e}}{\partial t} = -\frac{\rho_*^{-1}}{B}[\phi, V_{\parallel e}] - V_{\parallel e}\nabla_\parallel V_{\parallel e} + \frac{m_i}{m_e}\left(\nu j_\parallel + \nabla_\parallel\phi - \frac{1}{n}\nabla_\parallel p_e - 0.71\nabla_\parallel T_e\right) + \frac{4}{3n}\eta_{0e}\nabla_\parallel^2 V_{\parallel e} + s_{V_{\parallel e}}$$

Ion Parallel Velocity

$$\frac{\partial V_{\parallel i}}{\partial t} = -\frac{\rho_*^{-1}}{B}[\phi, V_{\parallel i}] - V_{\parallel i}\nabla_\parallel V_{\parallel i} - \frac{1}{n}\nabla_\parallel(p_e + \tau p_i) + \frac{4}{3n}\eta_{0i}\nabla_\parallel^2 V_{\parallel i} + D_{V_{\parallel i}}\nabla_\perp^2 V_{\parallel i} + s_{V_{\parallel i}}$$

How can we efficiently integrate this system in time?

Electron Temperature

$$\frac{\partial T_e}{\partial t} = -\frac{\rho_*^{-1}}{B}[\phi, T_e] - V_{\parallel e}\nabla_\parallel T_e + \frac{2}{3}T_e\left[0.71\frac{\nabla_\parallel j_\parallel}{n} - \nabla_\parallel V_{\parallel e}\right] + C(T_e, n, \phi) + \nabla_\parallel(\chi_{\parallel e}\nabla_\parallel T_e) + s_{T_e}$$

Ions Temperature

$$\frac{\partial T_i}{\partial t} = -\frac{\rho_*^{-1}}{B}[\phi, T_i] - V_{\parallel i}\nabla_\parallel T_i + \frac{2}{3}T_i\left[j_\parallel\frac{\nabla_\parallel n}{n^2} - \nabla_\parallel V_{\parallel e}\right] + C(T_i, T_e, n, \phi) + \nabla_\parallel(\chi_{\parallel i}\nabla_\parallel T_i) + s_{T_i}$$

Electrostatic Potential

$$\nabla\cdot(n\nabla_\perp\phi) = \Omega - \tau\nabla_\perp p_i \quad \text{Algebraic constraint}$$

Different Time integration schemes

Explicit Time integration schemes:

- **Pros:**
  - Easy to implement
  - Each time step is computationally fast
  - Easy parallelizable
- **Cons:**
  - CFL condition on the Δt

Implicit Time integration schemes:

- **Pros:**
  - No CFL condition (most of them unconditionally stable)
- **Cons:**
  - Expensive per step
  - Complex implementation
  - Harder to parallelize

Implicit-Explicit (IMEX) time integration scheme:

- **Pros:**
  - Balances stability & efficiency: non-stiff terms are treated explicitly and stiff terms treated implicitly
- **Cons:**
  - Prior knowledge of the physics
  - Still needs implicit solver (implementation complexity)

IMEX-RK

$$\frac{\partial \mathbf{q}}{\partial t} = \mathbf{F}_S(\mathbf{q}, z) + \mathbf{F}_F(\mathbf{q}, z)$$
$$0 = g(\mathbf{q}, z)$$

- Fast terms
- Slow terms

$$\begin{cases} \mathbf{q}^{(i)} = \mathbf{q}^n + \Delta t \sum_{j=1}^{i-1} a_{ij} \mathbf{F}_S\left(\mathbf{q}^{(j)}, z^{(j)}\right) + \Delta t \sum_{j=1}^{i} \bar{a}_{ij} \mathbf{F}_F\left(\mathbf{q}^{(j)}, z^{(j)}\right), \\ 0 = g(\mathbf{q}^{(i)}, z^{(i)}), \end{cases} \quad \text{for } i = 1, \ldots, s$$

MEX chosen: BPR(3,5,3)[1]

n. of explicit stage | n. of implicit stage | combined order of accuracy

Properties of BPR(3,5,3)

- Globally stiffly accurate IMEX-RK
  $\Longrightarrow$ L-stable method (amplification factor goes to 0 as stiffness goes to infinity $\lim_{z\to\infty} R(z) = 0$)
- With Singly Diagonally Implicit Runge Kutta method

[1]Boscarino, S., Pareschi, L., & Russo, G. (2013). Implicit-explicit runge–kutta schemes for hyperbolic systems and kinetic equations in the diffusion limit. SIAM Journal on Scientific Computing.

IMEX-RK implementation

At every internal stage:

Explicit

$$\theta^{(i)} = \theta^n + \Delta t \sum_{j=1}^{i-1} a_{ij} f_{n,S}\left(\theta^{(j)}, \phi^{(j)}, V_{\parallel e}^{(j)}\right)$$

$$n^{(i)} = \exp \theta^{(i)}$$

RHS

$$\mathrm{rhs}_1 = \Omega^n + \Delta t \sum_{j=1}^{i-1} a_{ij} f_{\Omega,S}\left(\theta^{(j)}, \phi^{(j)}, V_{\parallel e}^{(j)}\right) + \Delta t \sum_{j=1}^{i-1} \bar{a}_{ij} \left[-B^2 n^{(j)} \nabla_\parallel V_{\parallel e}^{(j)}\right]$$

$$\mathrm{rhs}_2 = V_{\parallel e}^n + \Delta t \sum_{j=1}^{i-1} a_{ij} f_{V_{\parallel e},S}\left(\theta^{(j)}, \phi^{(j)}, V_{\parallel e}^{(j)}\right) + \Delta t \sum_{j=1}^{i-1} \bar{a}_{ij} \left[\frac{m_i}{m_e} \nabla_\parallel \phi^{(j)} + \frac{4}{3n^{(j)}} \nabla_\parallel^2 V_{\parallel e}^{(j)}\right]$$

Implicit

$$\begin{bmatrix} \mathbf{I} & \bar{a}_{ii}\Delta t n^{(i)} \nabla_\parallel & \mathbf{0} \\ \mathbf{0} & \mathbf{I} - \bar{a}_{ii}\Delta t \frac{4}{3n^{(i)}} \nabla_\parallel^2 & -\bar{a}_{ii}\Delta t \frac{m_i}{m_e} \nabla_\parallel \\ \mathbf{I} & \mathbf{0} & -\nabla \cdot \left(n^{(i)} \nabla_\perp\right) \end{bmatrix} \begin{bmatrix} \Omega^{(i)} \\ V_{\parallel e}^{(i)} \\ \phi^{(i)} \end{bmatrix} = \begin{bmatrix} \mathrm{rhs}_\Omega \\ \mathrm{rhs}_{V_{\parallel e}} \\ \mathrm{rhs}_\phi \end{bmatrix} \longrightarrow \text{Algebraic constraint}$$

Physics-based preconditioners

For large linear system direct solvers are too expensive

↓

Need an **efficient preconditioner for GMRES** to transforms the system into a better-conditioned one.

$$\underbrace{\begin{bmatrix} \mathbf{I} & \bar{a}_{ii}\Delta t n^{(i)}\nabla_{\parallel} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} - \bar{a}_{ii}\Delta t \frac{4}{3n^{(i)}}\nabla_{\parallel}^2 & -\Delta t \bar{a}_{ii}\frac{m_i}{m_e}\nabla_{\parallel} \\ \mathbf{I} & \mathbf{0} & -\nabla\cdot\left(n^{(i)}\nabla_{\perp}\right) \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} \Omega^{(i)} \\ V_{\parallel e}^{(i)} \\ \phi^{(i)} \end{bmatrix} = \begin{bmatrix} \mathrm{rhs}_{\Omega} \\ \mathrm{rhs}_{V_{\parallel e}} \\ \mathrm{rhs}_{\phi} \end{bmatrix}$$

Important features of an efficient preconditioner

- $\mathbf{P}$ easy to invert
- $\mathbf{P} \approx \mathbf{A}$ ($\mathbf{P}^{-1}\mathbf{A} \approx \mathbf{I}$ or $\mathbf{A}\mathbf{P}^{-1} \approx \mathbf{I}$)

Different ways to construct a preconditioner:

- **Algebraic preconditioner based on the matrix A (ILU, Jacobi, GMG, AMG ...)**
- **Physics-based preconditioning strategy[2]: solving a wave equation for ϕ (idea inspired by MHD)**

[2]Chacón, Luis, Dana A. Knoll, and J. M. Finn. "An implicit, nonlinear reduced resistive MHD solver." *Journal of Computational Physics* 178.1 (2002): 15-36.

EPFL

GPU porting

Bringing the physics-based IMEX solver to NVIDIA GH200 (Alps)

GPU porting strategy

Solver port (OpenACC + PETSc)

- Explicit RHS, parallel-gradient & auxiliary-field kernels offloaded with OpenACC
- PETSc GPU backend: aijcusparse matrices, CUDA vectors
- Hypre BoomerAMG preconditioners run on the GPU (wave / diffusion / temperature systems)
- COO device matrix assembly — values built on device, no per-step host refill
- Physics-based PC matrix–vector product moved to the device

Target platform & memory

- NVIDIA GH200 Grace-Hopper, coherent C2C — 4 GPUs / node
- Unified / managed memory with cudaMemAdvise residency hints (−90% coherence faults)
- GPU-aware MPI halo exchange over NVLink (P2P)
- Same source & input as CPU — single code base

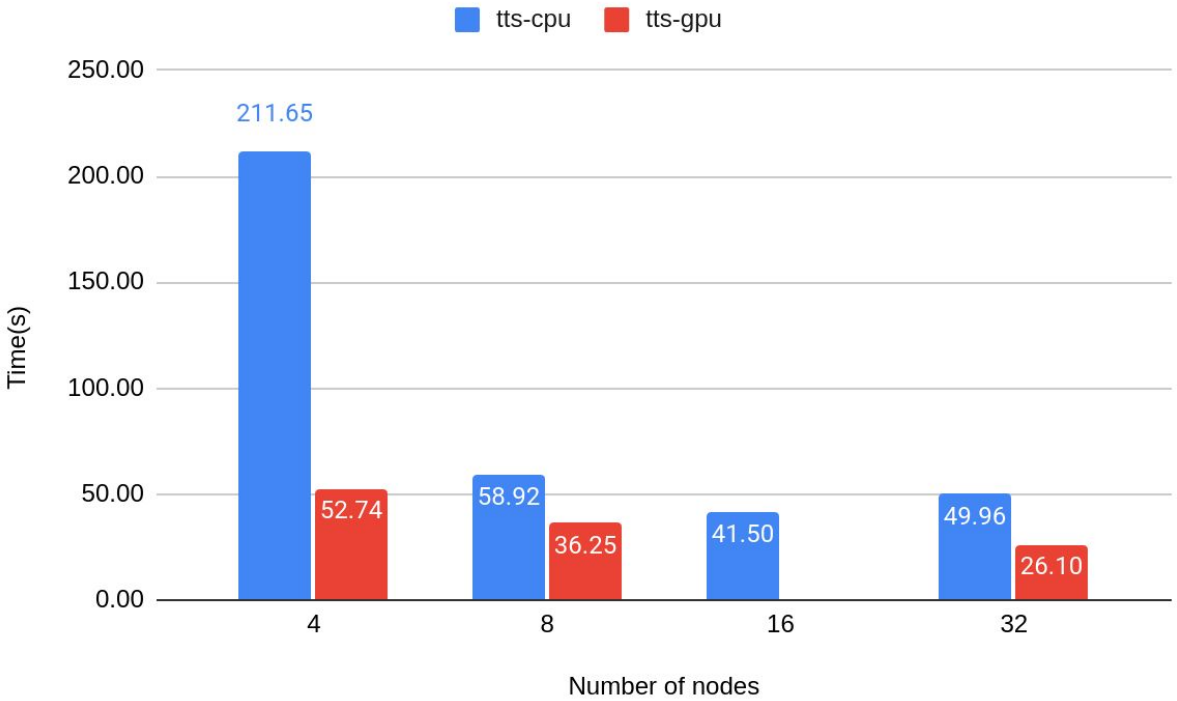
Scaling study(TCV) — test case & method

480×640×128	39.3 M	10
grid points	cells (fixed size)	time steps (NRUN)

Methodology

- Strong scaling: identical problem, increasing resources (CPU cores / GPUs)
- Platform: Alps / Daint GH200 — GPU: 4 GPUs per node · CPU: 256 MPI ranks per node
- Metric: “Main loop” wall-clock time (setup/prologue excluded)
- Each point = best of three domain-decomposition permutations (x/y/z split)
- Identical numerics on both backends: physics-based IMEX, Hypre BoomerAMG solves
- Synthetic simulations

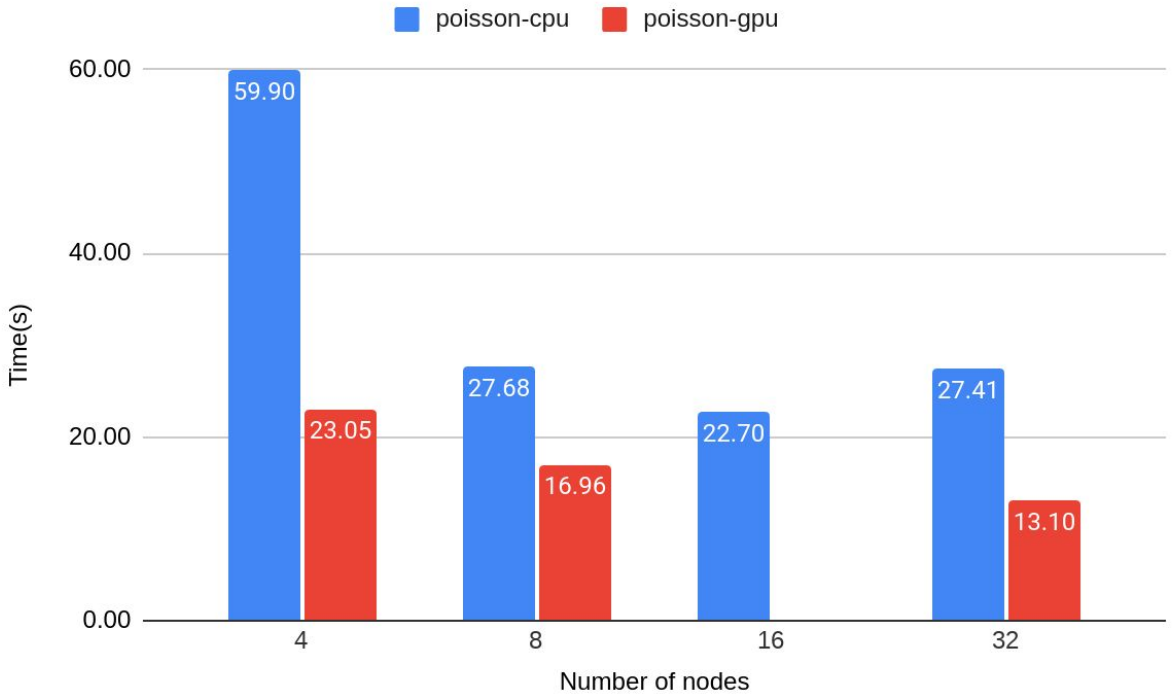
CPU vs GPU - Total time to solution



- 4 nodes CPU: memory capacity regime
- GPU outperform the CPU at every node count
- At 16 nodes the GPU run failed

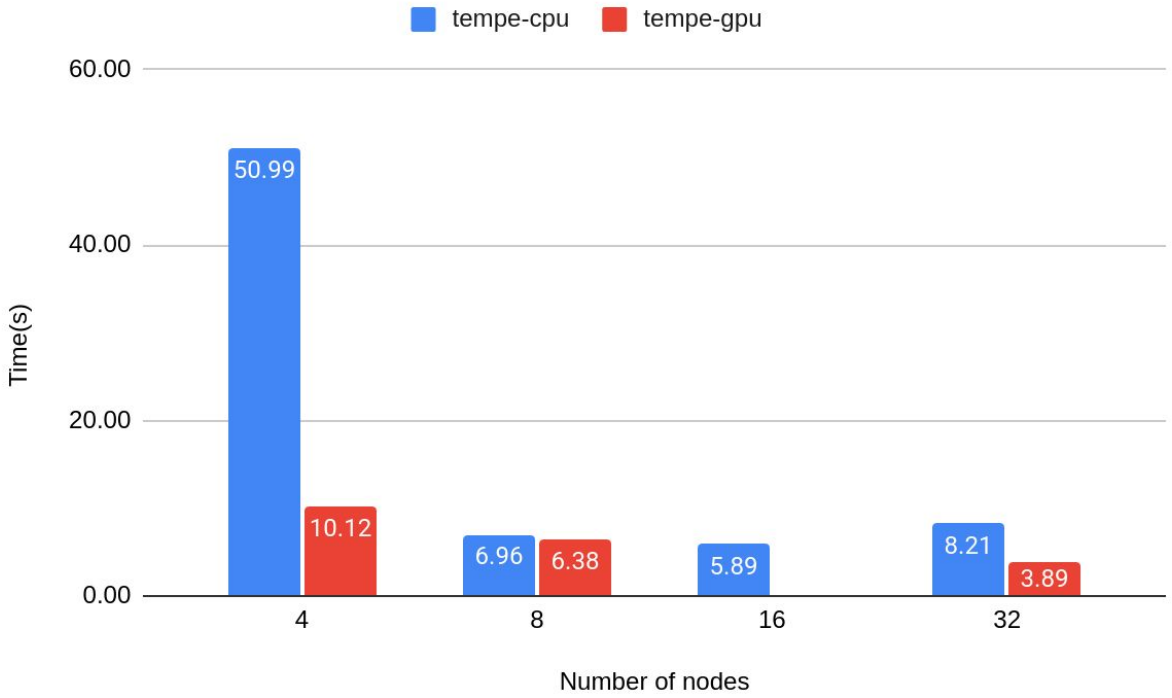
GPU = 4 GPUs/node · CPU = 256 ranks/node

Implicit solver



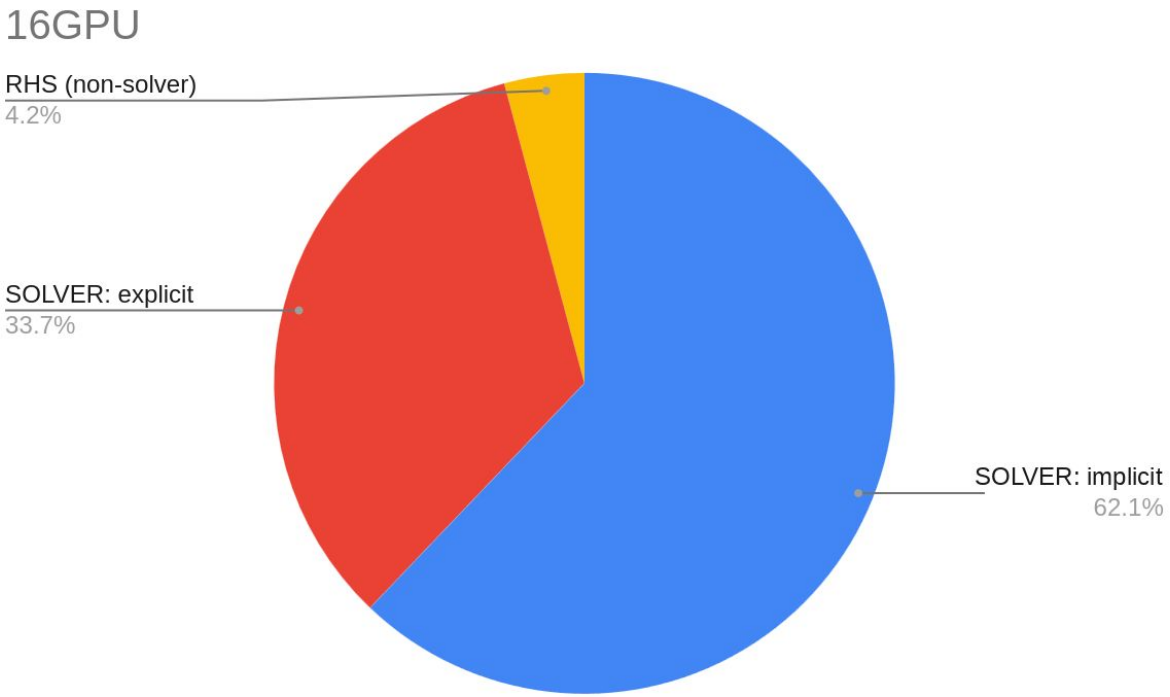
- 4 nodes CPU: memory capacity regime
- GPU outperform the CPU at every node count
- At 16 nodes the GPU run failed

Explicit solvers



- 4 nodes CPU: memory capacity regime
- GPU outperform the CPU at every node count
- At 16 nodes the GPU run failed

GPU timers breakdown



- Poisson + explicit solvers account for 95% of the time. Any optimization effort should go there.
- The RHS computation is successfully offloaded through OpenACC

%GPU busy

bucket	16GPU	32GPU	128GPU
SOLVER: poisson_strmf (phi)	5.7	2.2	0.8
SOLVER: IMEX explicit solves	17	8.5	3.3
RHS (non-solver)	6.1	3.7	1.2

quantity	16GPU	32GPU	128GPU
grid_points_per_gpu	2,457,600	1,228,800	307,200
footprint_per_gpu_(MB)	118	59	14.7

- **TCV is still small at scale**
- **The GPU is becoming less and less busy when the size increase**

Conclusions & outlook

Conclusions

- IMEX solver ported to GPU (OpenACC + PETSc/Hypre on device)
- Validated: iteration-10 fields match the CPU reference
- ~1.7× faster than CPU at 4 nodes; parity at 8–32 nodes
- Best time 46 s (128 GPU) ≈ 8192 CPU cores

Outlook

- Overlap communication with compute; pipelined Krylov to beat the MPI wall
- Better preconditioner reuse for the Tempe/Tempi solves
- Resolve the 64-GPU divergence
- Extend device residency to solver work buffers

EPFL

Beyond compute: communication profiling and optimization in Feltor

FELTOR benchmark: diverted-tokamak edge turbulence

Physics

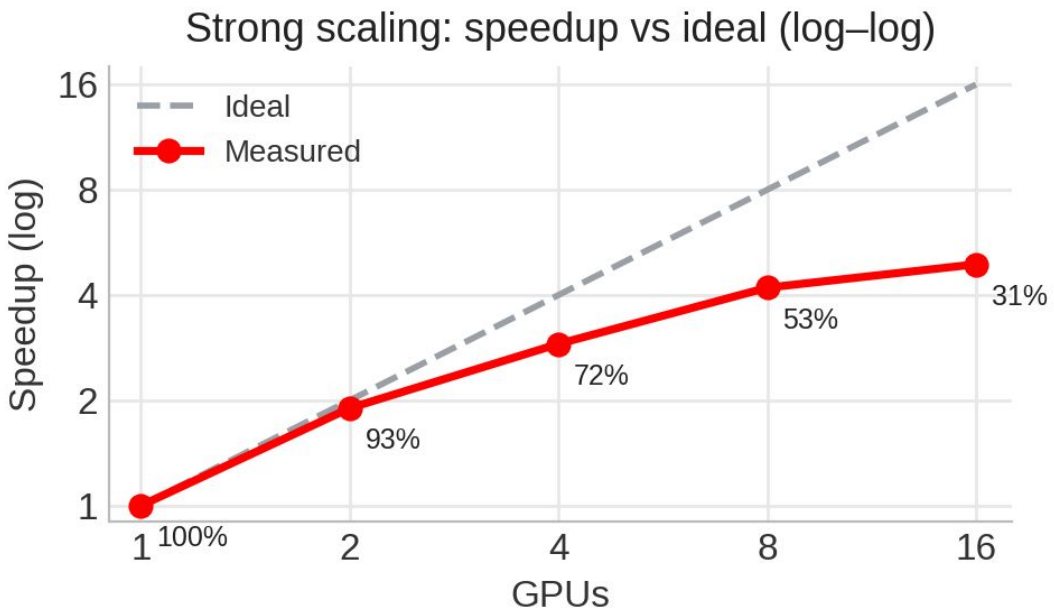
- Electromagnetic gyrofluid: n_e, N_i, $u_{/\!/}$, $A_{/\!/}$ (4 evolved fields)
- Solovev X-point equilibrium, COMPASS-like (κ=1.44, δ=0.3)
- Deuterium, $m_i/m_e \approx 3670$, $\beta = 10^{-4}$; insulating sheath + wall penalization

Numerics

- Discontinuous Galerkin n=3, 192×336×32 ≈ 18.6 M nodes/field
- 3 elliptic solves/step: polarisation (nested), Ampère, Γ
- z-only split on 16 H100 → ~2 planes/GPU → latency-bound

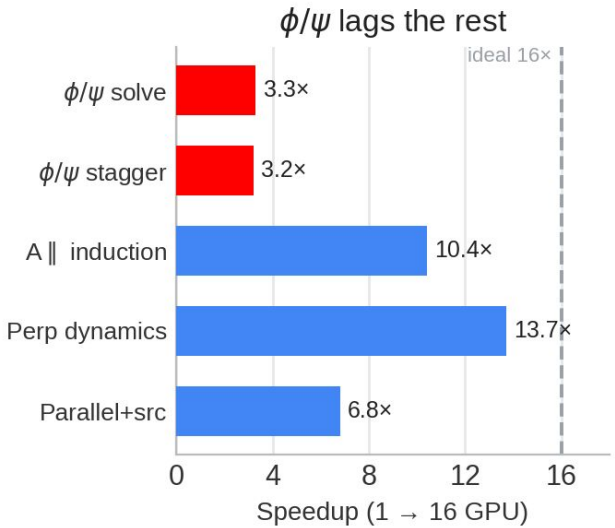
Strong scaling

GPUs	s / loop	speedup	efficiency
1	101.9	1.0×	100%
2	54.8	1.9×	93%
4	35.5	2.9×	72%
8	24.2	4.2×	53%
16	20.7	4.9×	31%



Per-phase scaling

Phase	MPI op	1	2	4	8	16	1→16
φ/ψ field solve	Allreduce	37.8	22.5	15.0	12.5	11.3	3.3×
φ/ψ stagger	Sendrecv	3.9	2.2	2.6	1.2	1.2	3.2×
A // induction	Allreduce	50.3	23.5	12.9	7.0	4.8	10.4×
Perp dynamics	Sendrecv	1.9	1.0	0.5	0.3	0.1	13.7×
Parallel + sources	Sendrecv	8.5	4.5	3.4	1.7	1.2	6.8×
STEP (sum)	mixed	102.3	53.6	34.4	22.7	18.7	5.5×

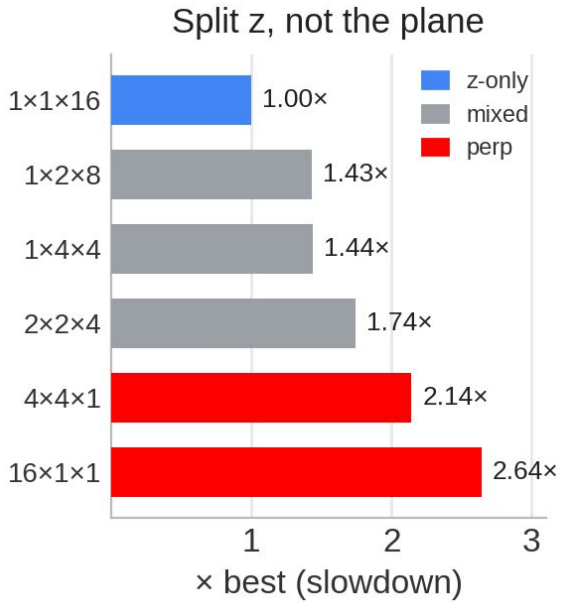


Anatomy of one timestep: five phases

Phase	What it does	Solves	Main MPI
1 · φ/ψ field solve	Invert polarisation for φ, gyro-average ψ	3 PCG	Allreduce
2 · φ/ψ stagger	Interpolate fields onto field-aligned grid	—	Sendrecv
3 · A // induction	Solve Ampère for A // , recover u //	1 PCG	Allreduce
4 · Perp dynamics	E×B advection + magnetic-drift fluxes	—	Sendrecv
5 · Parallel + sources	Streaming, resistivity, sheath, sources	—	Sendrecv

Split z, not the perpendicular plane

Layout npx×npy×npz	s / loop	× best	class
1×1×16	20.7	1.00×	z-only
1×2×8	29.5	1.43×	mixed
1×4×4	29.7	1.44×	mixed
2×2×4	36.0	1.74×	mixed
4×4×1	44.3	2.14×	perp
16×1×1	54.6	2.64×	perp



- **z > y > x — pure-z wins**
- **any x/y split adds a halo per CG iteration**
- **off-z layouts 1.4–2.6× slower**
- **scale via a bigger grid, not more ranks**

Future work - in progress

Phase 1 · φ/ψ solve — cut the collectives

- Communication-avoiding s-step CG: fold s iterations into one Allreduce (stabilize the scaled basis that fails at solve 47)
- Fewer iterations: stronger preconditioner + warm-start from the previous step (recycle the Krylov basis)
- Revisit the reduction path on this fabric (NCCL disabled today)

Phase 2 · φ/ψ stagger — hide the z-halo

- Overlap the field-aligned halo with the interpolation (Isend/Irecv; compute the interior while planes are in flight)
- Fuse the stagger into neighbouring phases; batch the per-field planes into one message
- More work per GPU: bigger grid / fewer z-ranks to climb off the ~2-planes/GPU floor
- CUDA-aware point-to-point over NVLink for the nearest-plane exchange