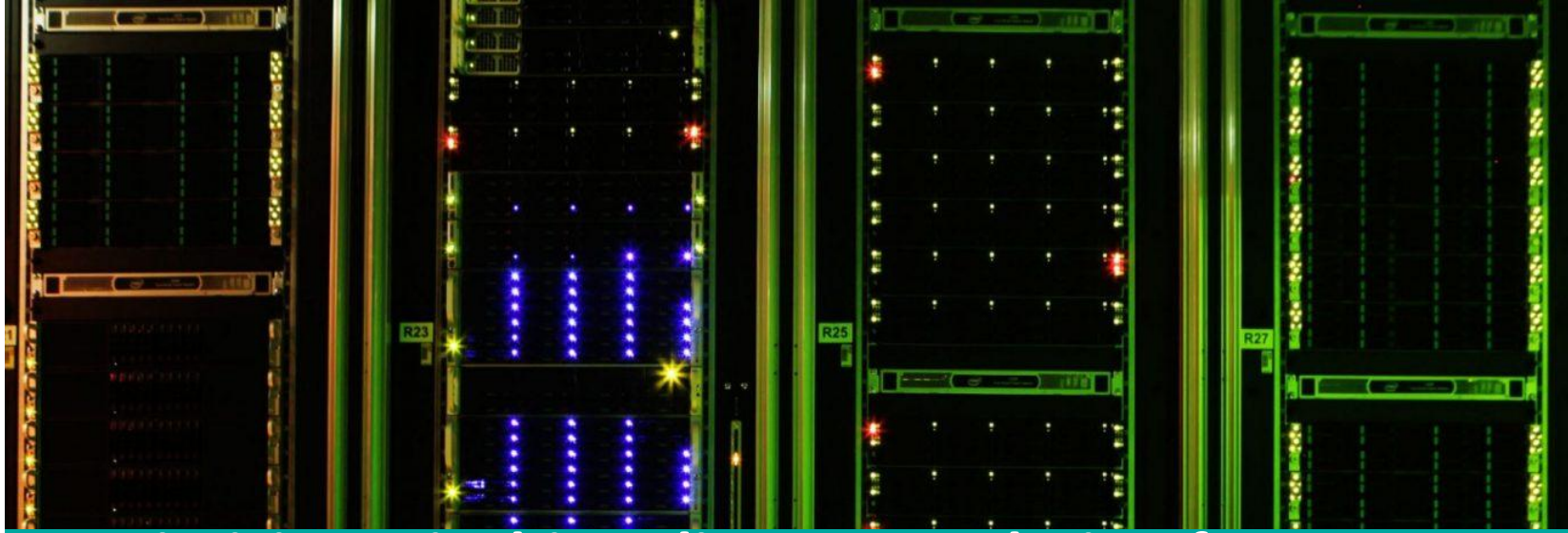


Optimizing Tricubic Spline Interpolation for ASCOT5 Using OpenCode

Seven sessions, two dead ends, one production kernel

Gilles Fourestey, Mathieu Peybernes, EPFL EUROfusion ACH
Simppa Äkäslompolo, Jari Varje (VTT)



ASCOT5

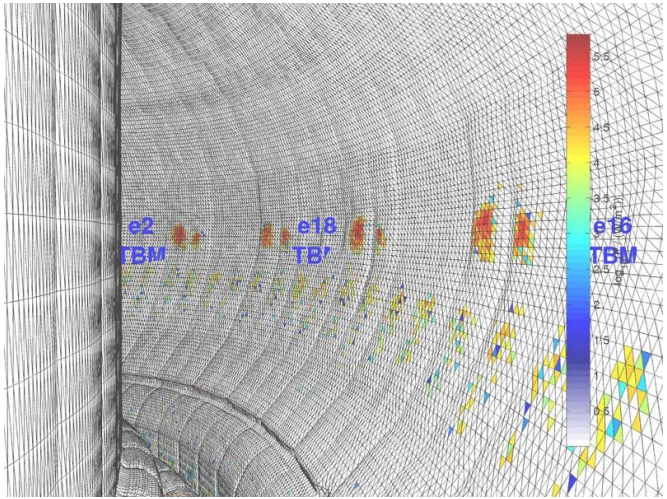
- ASCOT5 is a test **particle orbit-following** code for toroidal magnetically confined fusion devices
- The code uses the **Monte Carlo method** to solve the distribution of particles by following their trajectories.
  - The **evolution of the distribution function** for a test particle species *a*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



The problem: B-field evaluation dominates ASCOT5 GPU runtime

ASCOT5 full-orbit pusher (step_fo_vpa): every marker step evaluates the magnetic field B and its gradients

- B_3DS field = 3 tricubic splines on a 120 x 360 x 200 grid (R, phi, z) + one 2D psi spline

  4.4 GB of double coefficients per spline — far beyond cache; random 64-coefficient gathers per eval
- Profile at 1M markers (PGI_ACC_TIME): step_fo_vpa = 362 s of 546 s total = **66% of time-to-solution**
- Roofline: ~2.1 KB moved and ~2000 FLOPs per marker-step, that is **~1 FLOP/byte, deeply memory-bound on GH200**

  FP64 ridge ~8.5 FLOP/B → ceiling ~12% of peak FLOPs; measured ~1.3 of ~4 TB/s effective

Goal: keep bit-comparable physics while cutting the memory traffic

This is a memory-hierarchy problem

Setup

- Simple-spline-interpolation **reproducer code** for optimization then inserted in the ASCOT5 (git@gitlab.com:ursache/simple-spline-interpolation.git)
- **OpenCode** 1.15.10 with
  - GLM 4.3 on 8xB200@SCITAS
  - GLM 5.3 flash on 4xH100@SCITAS
- Running on one Grace-Hopper **GH200** node @EPFL
- ~**1 week of work**, no coding myself, only prompts and checks

Starting point:

- OpenACC and CUDA version available
- **OpenACC version faster than the CUDA version** (developed using OpenCode and DeepSeek v4 lite)

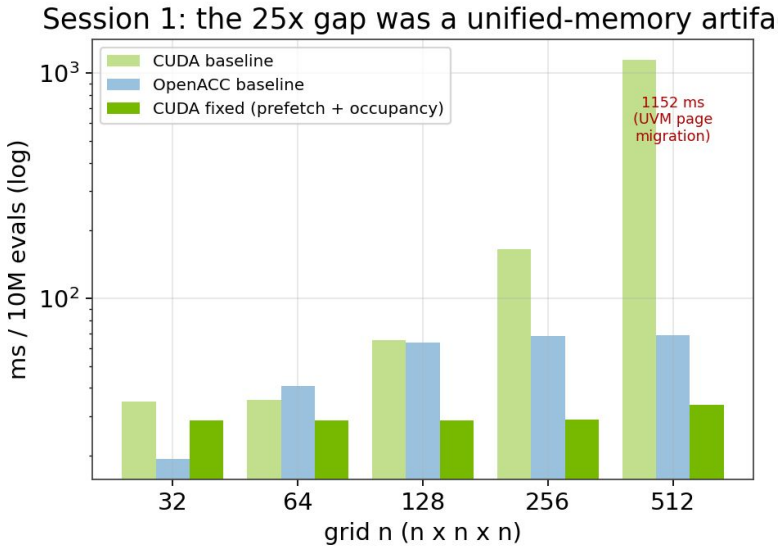
Benchmark discipline (session 1)

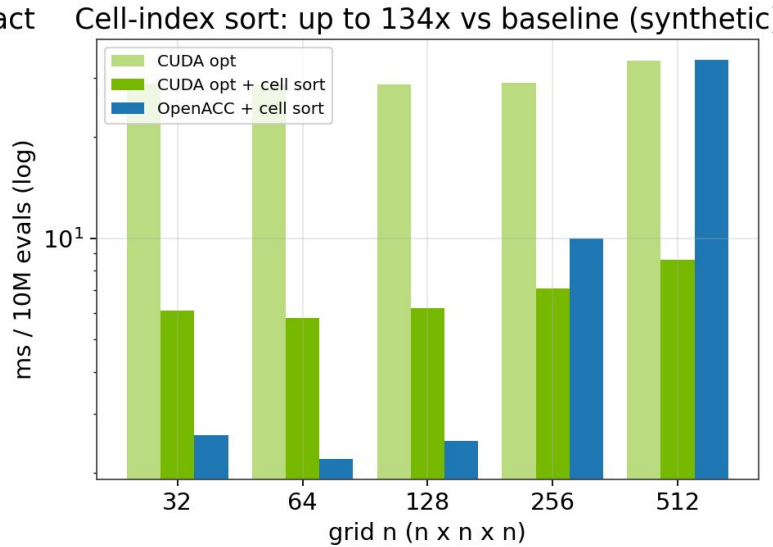
- Standalone harness:
  - 10M random eval points,
  - grids n = 32…512,
  - f + first derivatives only (ASCOT5 never uses the Hessian — removed, saves 60% of arithmetic)
- Timing covers the GPU eval kernel only: spline init (15–17 s, host) is out of scope — built once in production
- CUDA vs OpenACC, algorithmically identical kernels — only the backend differs
- Correctness gate after EVERY change: errors must be bit-identical (or rounding-level, if summation order changes)
  This gate is what kept 7 sessions of kernel surgery honest

Lesson 0 *(found on day 1): a local Makefile edit had* ***silently switched -O2 to -O0****;*
One 'no effect' full-code benchmark was measuring unoptimized SASS. ***Always double check****.*

5 sessions over 5 days

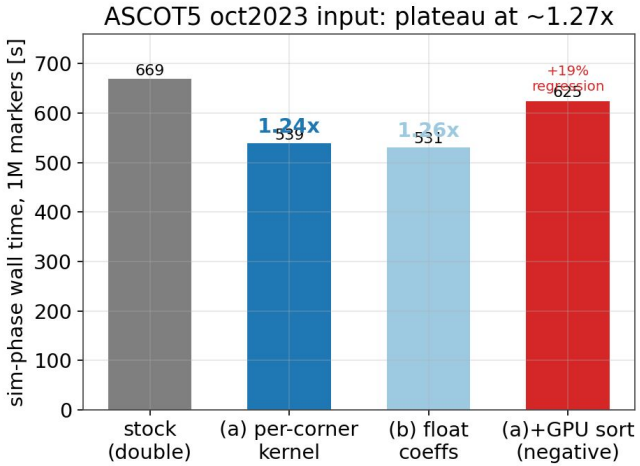
- **Session 1 (Sept 19–20): The 25x that wasn't there**
  - Built a standalone CUDA-vs-OpenACC benchmark of the tricubic spline eval (10M points, grids 32 to 512 points).
  - Fixed the unified-memory artifact (25× phantom gap), vectorized loads, occupancy tuning, and cell-index sort: up to 134× on synthetic grids, bit-identical errors.

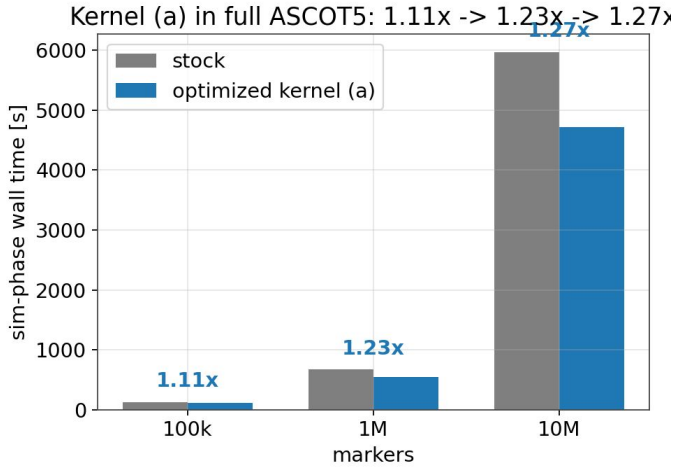




5 sessions over 5 days

- **Session 2 (Sept 22):**
  - **Validated on the real ASCOT5 code** (5.6.4 reproducer): kernel win is real (1.8×), **cell sort is NOT worth it** (qsort cost),
  - found a **silent -O0 flag bug**, then ~4–7% total-runtime gain.
- **Session 4 (Sept 23):** Restarted integration ASCOT main code:
  - trimmed the 10-output eval to 4 outputs (saves 60% of arithmetic),
  - fixed Makefiles,
  - benchmarks: **1.11×/1.23×/1.27× at 100k/1M/10M markers**.





EPFL 5 sessions over 5 days

- Session 5 (Sept 24): **plateau confirmed at ~1.27×**
  - Roofline analysis (deeply memory-bound, ~1 FLOP/B).
  - Tried float coefficients (only +1.5%, latency-bound) and GPU sort + shared memory (+19% regression)

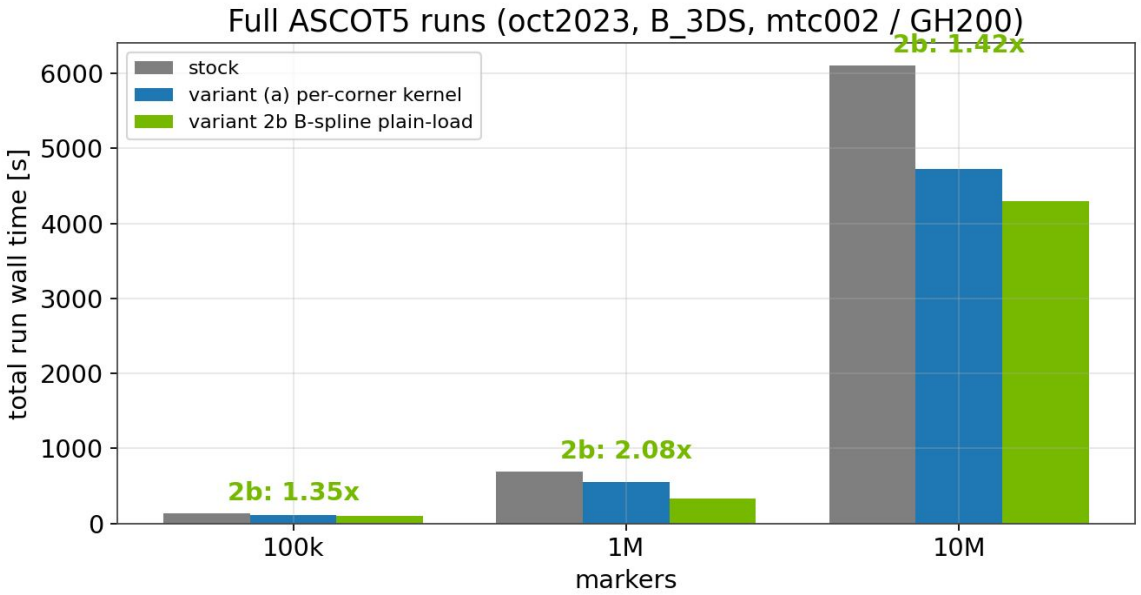
- Session 6 (Sept 24): Built the CUDA texture variants
  - hardware trilinear + full Boris push in one kernel,
  - solved OpenACC/CUDA interop
  - architecturally working but accuracy-failed (1e-3 error)

Variants

- Stock: compact tricubic spline (8 coeff/cell, double, 2.2 GB) — OpenACC spline calls per particle — **baseline**
- (a) per-corner kernel (real4 vectorized loads): same representation, restructured arithmetic (−31% instr, −52% FMA) — bit-identical to stock — **kept for bit-comparable fates**
- (b) float coeffs: same, coefficients stored as float (1.1 GB) — only +1.5% over (a): latency-bound, not bandwidth — **dropped**
- texture filtering: B-spline coefficients + hardware trilinear (Sigg-Hadwiger) — float32 coordinate ulp perturbs the weights → error 3e-3 — **dead**
- 2b B-spline (**shipped**): 1 coeff/point, ghost-extended float (35.7 MB) — 64-gather + β^3 weights in double — accuracy ~1e-6, L2-resident → **the 1.42× winner**

	stock	(a) per-corner	(b) float coeffs	texture filtering	2b B-spline
Accuracy vs stock	–	bit-identical	~1e-7	3e-3 ✗	~1e-6 ✓
Time to solution (10M)	101m39s	78m44s	–	–	71m38s
Speedup vs stock	1.00×	1.29×	1.31× (1M only)	✗ rejected	1.42×
Memory/spline	2.2 GB	2.2 GB	1.1 GB	35 MB	35.7 MB (L2)

Variants



- 10M markers: stock 101m39s, (a) 78m44s (1.29x), 2b 71m38s (1.42x)
- 1M: 2b is 1.66x vs (a) — the eval gain partially masks at 10M (L2 pressure)
- Memory: 107 MB vs 6.6 GB of coefficients (fits L2)
- Fates: 2b 217k wall hits vs stock 284k, chaos decorrelation?

Speed-up evolution

Micro-optimization was exhausted at 1.27x. The speedup came from memory behavior modification, in order of impact:

- **The B-spline representation** (biggest lever, **~1.3× of the 1.42×**):
  - stock reads ~4^3 = 64 double knot coefficients per eval from 2.2 GB/spline of compact tricubic coefficients, all streaming from HBM every step.
  - The B-spline prefilter stores 1 float coefficient per grid point: 35.7 MB/spline → 3 splines = 107 MB, which fits in GH200's 126 MB L2 cache.
  - **30–60× less DRAM traffic**; gathers hit cache instead of memory.
- **One fused CUDA kernel for the entire Boris push**:
  - stock calls OpenACC-compiled spline code per particle, per evaluation.
  - The single kernel does momentum transform → B eval → rotation → position update → B+dB eval with no launch/argument overhead and better locality, plus weights computed in double (64-gather + β^3 contraction).
- **Per-corner accumulation + 4-output trim** (session 2–4 work, **1.11–1.29×**):
  - restructured the eval (–31% instructions, –52% FMAs in SASS)
  - dropped the 6 unused second-derivative outputs (60% of the arithmetic).

Summary

OpenCode/GLM has improved TTS by 1.42x:

- Result: 2b shipped → 1.42× vs stock at 10M markers, memory cut ~60×
- Tradeoff: ~1e-6 error and decorrelated fates, **this needs validation from field scientists**

A few comments/recommendations:

- AI coding agents are amazing 95% of the time, catastrophic 5% of the time
- Ask the agents to explain every step and check the results as it is very easy to lose understanding of what is being done
- Involvement of the field scientist will become more and more important as domain knowledge will be need to check physical validity of the results
- Large context might be a bad idea: dump the session in a file and re-inject it when the context is reset (it cost tokens though, need a good tradeoff)
- Cost can explode very quickly if agent is not local

Summary

Full breakdown since Sept 19:

conversation	input	output	reasoning	cache rd	cache wr
Cleaning repo for CUDA vs OpenACC (09-19)	171,797,131	287,862	341,778	0	0
Importing session_rescue.json (09-25)	93,286	292	286	0	0
Continuing OPTIMIZATION_LOG.txt work (this one)	42,587,446	117,103	146,138	0	0
Survey ASCOT5 B-field integration points (09-28)	397,667	3,325	843	0	0
New session 2026-10-07	31,650,398	79,004	55,135	0	0
TOTAL	246,525,928	487,586	544,180	0	0

Grand total: 247,557,694 tokens, cost $0.00 (your GLM-5.3 provider reports no pricing).

Next step

- We are going to submit a proposal in order to develop a surrogate model for spline interpolation using torch torchfort in collaboration with Nvidia.
- First results show standard MLP models provide faster evaluation but high error.

Mixture	Share
Uniform interior points	55%
Grid-jittered points	15%
Exact grid nodes	10%
Boundary planes + φ seam	10%
Near-axis Gaussian (clustered)	10%

Original ASCOT5 B-field evaluation — 3D spline interpolation

Native B_3DS path for one particle query; no neural network

ONCE PER FIELD Read gridded B_R, B_φ, B_Z and axisymmetric ψ(R,Z); precompute spline coefficients and offload field data.

REPEATED FOR EACH PARTICLE: POSITION (R, φ, Z)

1 Locate cell

Wrap φ periodically; find local R–φ–Z cell.

2 Interpolate 3D B

For each component: 8 cell corners × 8 stored coefficients = 64 local values.

3 Evaluate splines

Evaluate B and first spatial derivatives from the local spline.

4 Add ψ correction

2D ψ spline supplies poloidal B_R/B_Z and derivative corrections.

Poloidal correction (axisymmetric ψ)

$B_R \leftarrow B_R - (\partial\psi/\partial Z)/R$ $B_Z \leftarrow B_Z + (\partial\psi/\partial R)/R$

Returned by B_field_eval_B_dB

(B_R, B_φ, B_Z) + 9 spatial derivatives

Key cost per query: repeated local spline-coefficient accesses for 3D B, plus a 2D ψ evaluation.

Source: ASCOT5 B_field.c → Bfield/B_3DS.c → spline/interp3Dcomp.c (native 3DS evaluator).

Timing — Native ASCOT vs neural inference

H100 • same B_3DS queries • GPU-resident buffers • 100 repeats × 5 trials

ASCOT native — B_field_eval_B_dB

54.2 µs / 2,048 queries
26.5 ns/query • 37.8 M queries/s

Current TorchFort model — B + dB

2.019 ms / 2,048 queries
986 ns/query • about 37× slower

For one query: native 34.6 µs vs NN 1.762 ms (about 51× slower).

What each timing configuration measures — batch of 2,048

Configuration	Time / batch	vs native	What is being timed
Native ASCOT	**54.2 µs**	**1.0×**	**B_field_eval_B_dB: native 3D splines plus ψ correction; returns B and dB.**
TorchFort + autograd	2.019 ms	37.2×	Current MLP; autograd computes dB and create_graph=True builds the training graph.
No higher-order graph	1.486 ms	27.5×	Still uses autograd for dB w.r.t. coordinates; skips the graph for training gradients.
Manual Jacobian	0.717 ms	13.3×	Same weights; propagates derivatives by the chain rule, without autograd at inference.
CUDA Graph replay	0.319 ms	5.9×	Captured Python/JIT manual-Jacobian call to reduce launches; not yet integrated into TorchFort.

TF32 gave no speedup. Direct B+dB head reached 171 µs, but its dB/divergence accuracy degraded.

Best tested prototype remains ~5.9× slower than native; graph replay is not yet an ASCOT speedup.