

AI-assisted analysis of a flagship GPU code: the Tsunami-HySEA improvement roadmap

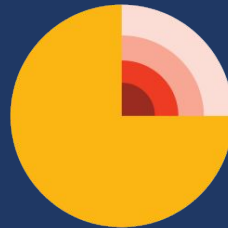
How a frontier agentic model produced a defects-and-improvements report for Tsunami-HySEA — and why expert feedback was decisive

ChEESE — WP2 / WP4 · EDANYA group, Universidad de Málaga · July 27th, 2026

J. M. González Vida, M. de la Asunción · analysis performed with Claude Fable 5 (Anthropic) in Claude Code



UNIVERSIDAD
DE MÁLAGA



ChEESE



Context and goal

Tsunami-HySEA: ChEESA flagship code for FTRT tsunami simulation and PTHA

CUDA + MPI, nested meshes (static AMR), production use in warning centres

Highly optimized over a decade — but documentation is sparse and deep analyses are costly

Goal (June–July 2026): an independent, in-depth IA-assisted analysis of the full code

What does the code actually do, kernel by kernel? Any latent defects?

Which optimizations are still on the table for modern GPUs (Hopper/Blackwell)?

Instrument: Claude Fable 5 (frontier LLM) running in Claude Code — an agentic environment

It does not just chat: it reads the whole repository, compiles, runs on a cluster over SSH, profiles, and writes/iterates the LaTeX report

Outcome: a 23-page technical report (rev. 5) — 2 defects found, 10 prioritized improvement proposals with implementation guides

Context and goal

2 Code overview and organization

2.1 Source tree

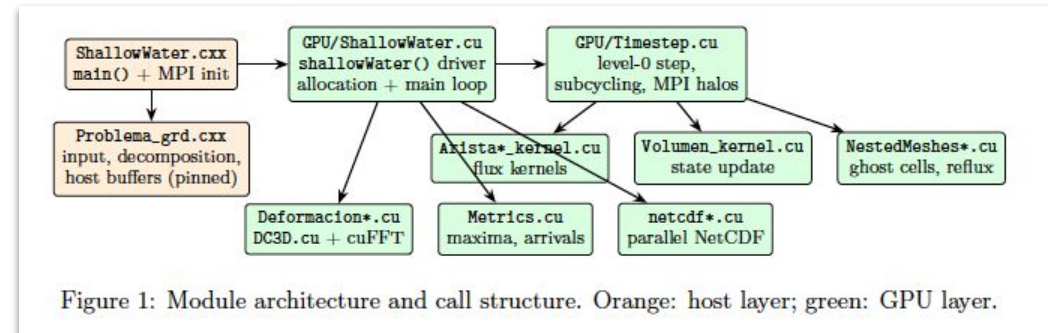
The code has two layers: a host layer that parses the problem definition, performs the domain decomposition and drives the simulation, and a GPU layer containing all computational kernels.

File	Role	Lines
ShallowWater.cxx	main(), MPI bootstrap, argument parsing	425
Problema_grd.cxx	Input parsing, domain decomposition, host allocation	~2,500
Constantes.hxx	Global constants, cluster/submesh data structures	129
OkadaTriangular.cxx	Host-side triangular Okada support	90
GPU/ShallowWater.cu	GPU driver: allocation, main loop, output orchestration	2,132
GPU/Timestep.cu	Level-0 step, nested-level subcycling, MPI halo exchange	1,163
GPU/AristaVer_kernel.cu	Vertical-edge flux kernels (both steps, all variants)	1,997
GPU/AristaHor_kernel.cu	Horizontal-edge flux kernels	2,006
GPU/AristaCommon.cu	Shared edge machinery: limiters, positivity, initial Δt	519
GPU/Volumen_kernel.cu	State update, sponge layer, friction, Δt per volume	428
GPU/Reduccion_kernel.cu	Minimum reduction (SDK reduce6 style [14])	179
GPU/NestedMeshesCommon.cu	Ghost-cell interpolation, flux correction, activation	790
GPU/NestedMeshesVal.cu	Ghost-cell values, conservative restriction	462
GPU/Metrics.cu	Max. amplitude/velocity/flux, arrival times	643
GPU/Deformacion.cu	Okada application, Kajiura filter (cuFFT), cropping	608
GPU/DeformacionTriangular.cu	Triangular dislocation (ported Fortran [12])	958
GPU/DeformacionDinamica.cu	Time-interpolated dynamic deformation	112
GPU/DC3D.cu	Okada DC3D dislocation code [10, 11]	869
GPU/FriccionVariable.cu	Variable Manning friction interpolation	180
GPU/netcdf*.cu	Parallel NetCDF output and time series [21]	1,830

Table 1: Main source files under `src/`.

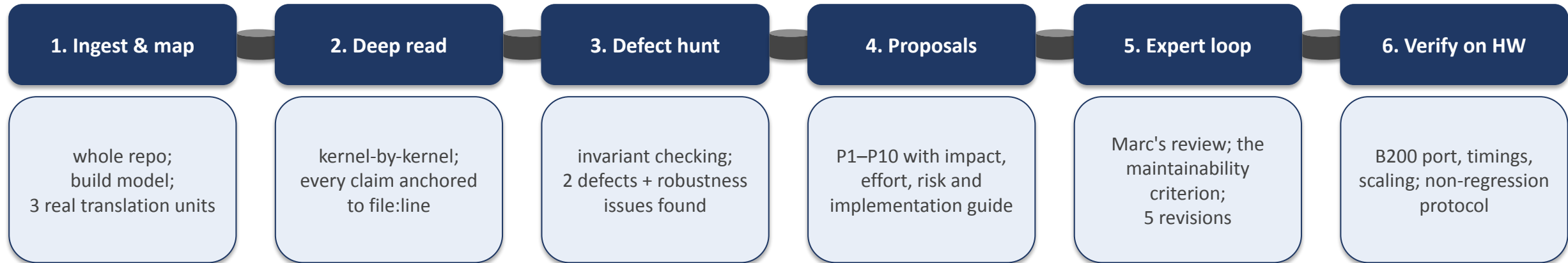
Code Architecture

- Two layers: host (C++) handles problem parsing, domain decomposition, and simulation orchestration; GPU (CUDA) contains all computational kernels.
- ~18,000 total lines of code, ~85% concentrated in the GPU layer.
- Kernel-level complexity: Okada dislocation model, Kajiura filtering, static AMR nested-mesh handling.
- This scale and specialization is what Fable 5 had to parse and understand in depth.



The roadmap at a glance

six phases, two loops



Loop A (fast): phases 2–4 iterate — every proposal must cite the exact lines it touches

Loop B (decisive): phases 4–6 iterate with the code developer —

Different proposals are: re-ranked, conditioned or discouraged based on maintainability and measured evidence

Everything logged: a changelog records every revision and why it happened (traceability)

Ground rules that made it trustworthy

1. Every technical claim anchored to file:line (e.g. src/GPU/AristaVer_kernel.cu:97,339)
Clickable, checkable, and it keeps the model honest — hallucinated claims cannot survive this rule
2. Uncertain interpretations explicitly flagged “To be confirmed by the authors”
The report **distinguishes what the code probably does from what it appears to intend**
3. Full-code coverage before any judgement
Compilation model mapped first (3 real translation units; #include-based single-TU CUDA) — many “obvious” refactors die here
4. Per-proposal validation plan defined up front
bitwise-identical where possible; tolerance-based where reductions/atomics reorder; regression suite as precondition
5. Living document with changelog — revisions traceable to specific feedback

Defects found (and patched)

Defect	Where	What happens	Status
D1: silent wrong minimum in reduction	src/GPU/Reduccion_kernel.cu:133	hand-written min-reduction returns a wrong value for non-power-of-2 sizes below the last block — affects the Δt global minimum in corner cases	trivial fix; unit test added
D2: wrong argument in final memory release	src/GPU/ShallowWater.cu:2121	last liberarMemoria call receives the wrong struct member; harmless today by luck of layout, latent crash on refactor	trivial fix
Robustness items (no observable failure)	various	unchecked CUDA/MPI return codes, magic numbers, fragile pointer aliasing between nesting levels	catalogued, prioritized in P9

How they were found: systematic kernel-by-kernel reading with invariant checking — not fuzzing, not static analysis

One proposal later coincided with a POP (Performance Optimization and Productivity) analyst finding; the defects and most proposals were new

The 10 proposals

impact alone does not decide: readability and long-term cost are first-class criteria

#	Proposal	Impact	Effort	Maintainability verdict
P1	eliminate full-grid D2D copies (double buffering)	high	medium	moderate cost; encapsulate
P2	coalesced access in vertical-edge kernels	high	med-high	variant A simplifies; variant C discouraged
P3	const __restrict__ qualification	moderate	trivial	improves (self-documenting)
P4	fuse the diagnostics kernels	moderate	low	net -440 lines of code
P5	CUB reductions replace hand-written ones	low-mod	low	net simplification (also fixes D1 class)
P6	device packing + CUDA-aware MPI	scaling	medium	neutral; isolate #ifdef
P7	CUDA Graphs for nested-mesh subcycles	moderate	medium	high cost; only if profiling justifies
P8	output path improvements	moderate	low	neutral / positive
P9	build system, affinity, error checking	enabling	low	improves by design
P10	minor kernel-level items	low	low	case by case

Two high-impact proposals were conditioned or discouraged by the human criterion — that is the point

The decisive ingredient: expert-in-the-loop

the model proposes, the code owner disposes — 4 traceable revisions

Marc's maintainability criterion, stated once, reshaped the whole document:

“every proposal must carry a readability/maintainability assessment; some optimizations do not pay for their complexity”

consequence: P2-variant-C and P7 are explicitly discouraged/conditioned despite performance upside

A concrete revision-loop example (from the report changelog, rev. 3 → rev. 4):

Marc: “equation (1) still shows the C-style case split you explained away in prose”

Fix: the flagged formula itself was rewritten (single sigmoid, $\alpha=0$ positional case), the code clamp re-described as branchless saturation — lesson: change the artefact, not the prose around it

Numerical-model chapter reviewed against the reviewer standard of the model's lead author (M. J. Castro): precise, complete, every formula anchored

Net effect: the report is co-authored in the real sense — AI throughput × expert judgement

Case study: P1 (double buffering) — AI archaeology before touching code

P1 looks trivial: replace full-grid device-to-device copies by pointer swap

The pre-implementation audit (a 'draft for Marc' document) found three hidden couplings that would silently break a naive swap:

- communication pointers of intermediate nesting levels mix the two buffers (harmless today only because staging runs after the D2D copy)
- a device-resident pointer table is uploaded once at start-up — it goes stale the moment buffers swap
- a seeding copy per step is NOT replaceable by a swap (ghost-frame freshness), only removable after a physical-boundary audit

Each finding cited with file:line (e.g. ShallowWater.cu:1237–1255, :1365; Timestep.cu:773–774)

Value: weeks of debugging avoided; the expert decides with the full coupling map on the table

Some questions asked (time, cost, prompting, tools, testing)

Question	Answer
How long?	25-page report: ~2-3 working days of interactive sessions (July 4–6), including the B200 port and timing campaign; revisions in minutes each
How expensive?	flat-rate subscription at that moment (no per-token billing); marginal cost per analysis ≈ 0 once subscribed; no fine-tuning, no special infrastructure. Now, Fable is available via API and with higher user plans. 10USD/1M upload tokens & 50USD/1M download tokens
Prompting practice	role and audience stated up front; hard rules: file:line anchoring, flag uncertainty, per-proposal maintainability assessment; 'document everything as you go' (changelog)
Tool integration	the agent itself runs nvcc/cmake/SLURM/ssh and reads profiler output; complements (not replaces) clang-tidy and POP static/dynamic analyses
Testing	per-proposal validation plan: bitwise-identical where possible, tolerance where reordering; regression suite as explicit precondition; verification runs on the target GPU

What worked, what to watch

Worked:

whole-repo reading depth no human review reaches in days: every kernel, every pointer path

the file:line rule converts 'AI opinions' into checkable statements

agentic verification: the same session compiles, runs and measures on the target machine

expert feedback loop: cheap for the expert (minutes per round), transformative for the output

Watch out for:

**plausible-but-wrong readings — mitigated by anchoring + 'to be confirmed' flags + expert review;
never merged unverified**

context limits on huge repos — mitigated by build-model mapping first, then targeted deep dives.

the model will happily optimize maintainability away if you do not make it a stated criterion

A recipe for other flagship codes

1. Give the agent the whole repo and ask for the build/compilation model first
- 2. Impose the ground rules: file:line anchoring, uncertainty flags, maintainability as explicit criterion**
3. Ask for defects and proposals with per-item validation plans (bitwise vs tolerance)
- 4. Review as the code owner: re-rank, condition, discourage — minutes per round, in writing**
5. Let the agent verify on the real machine (build, run, measure, A/B testing) before anything is claimed
6. Keep a changelog: the process itself becomes documentation (and a deliverable)

Cost of entry: a subscription (or via API -tokens-) and one afternoon of setup.

The expensive part — expert judgement — you already have.

Takeaway

A frontier agentic model turned a decade-old flagship GPU code into a mapped, audited, improvement-ready asset in days — but the report is only as good as the expert loop around it.

Contact: Marc de la Asunción (marcah@uma.es) Jose Manuel González-Vida (jgv@uma.es) · EDANYA, Universidad de Málaga

Thank you for your attention