Spectral element methods on the Tenstorrent accelerator

International Workshop on RISC-V for HPC

Daniyal Arshad, Nick Brown

epcc

Table of Contents

What we run: Nekbone

- **Nek5000** — a Gordon-Bell-Prize-winning spectral-element CFD solver (Navier–Stokes), used across physics and engineering.
- **Nekbone** — its mini-app: the computational heart, easy to experiment with. Lessons transfer to a whole *class* of HPC codes (and to Xcompact3d's concerns).

The hot spot: the AX kernel

Nekbone solves Poisson via conjugate gradient. One operator — the **AX kernel** — consumes roughly **75%** of runtime and is *memory-starved* on a CPU.

What the AX kernel actually computes

For every element (a small 3D block of NX^3 grid points), AX applies the Poisson operator:

- Compute gradients in x, y, z (u_r, u_s, u_t) via the differentiation matrix D.
- Apply geometric factors (here, identity).
- Accumulate: $w = D^{\mathsf{T}} u_r + u_s D + u_t D$.

We fix $\mathbf{NX = 16}$ (the largest common size): **96 matrix multiplies per element**, $\sim$795,000 FLOPs each (with $G = I$), every element independent.

Our hypothesis

It's almost all matrix multiplication — so it should map naturally onto the Tensix **matrix engine**, with far less memory overhead than a CPU.

Why the CPU struggles

Profiling AX on a 24-core Xeon Platinum (Cascade Lake):[1]

- 1 core: only **45%** of cycles do useful work; **26%** stall on memory.
- All 24 cores: useful work falls to **27%**, stalls rise to **56%**, **memory bandwidth saturates at 100%**.

Diagnosis

AX is **memory-bound**. The CPU spends most of its time *waiting*, not computing — exactly the kind of inefficiency a specialised architecture can attack.

[1] Profiling figures from Brown (2020), "Exploring the acceleration of Nekbone on reconfigurable architectures".

Table of Contents

The Tenstorrent Wormhole accelerator

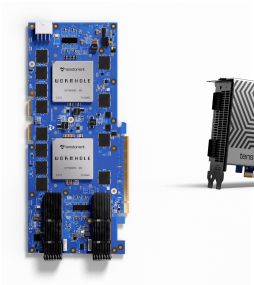


Figure: Wormhole™ n300 — a commercially available, open-stack PCIe card.[a]

- **Two** Wormhole chips, **128 Tensix cores** total, at 1 GHz.
- **24 GB** of GDDR6 (12 GB per chip).
- Host over **PCIe**; chip-to-chip and card-to-card over **standard Ethernet** (QSFP-DD) — no proprietary interconnect.
- Whole **software stack open-source**; Tenstorrent actively collaborates with researchers.

[a] Image: Tenstorrent product page / n300 datasheet.

Inside a Tensix core: 5 RISC-V cores + a co-processor

The 5 RISC-V cores **dispatch instructions** — they never touch the data:

- **2 data-movement** — NoC 0 ingests, NoC 1 egresses.
- **3 compute** — *Unpack*, *Math*, *Pack* — drive the co-processor.

The **co-processor** holds the horsepower:

- **FPU** (matrix) — 2048 multipliers; up to **4.1 TFLOPS**; BF16/TF32.
- **SFPU** (vector) — 32 lanes; richer maths (sqrt, sin, conditionals); FP32.

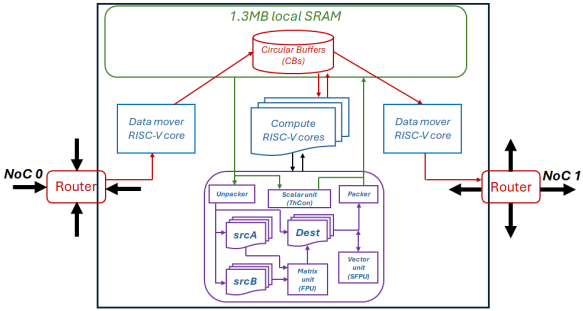


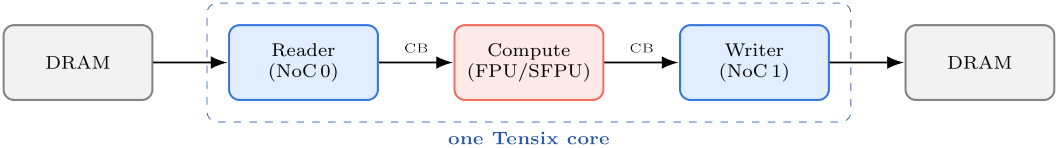
Figure: A single Tensix core in the Wormhole accelerator.[a]

Key idea: the co-processor reads SRAM *directly*; the RISC-V cores just choreograph it.

[a] Brown et al (2025), "Exploring FFTs on the Tenstorrent Wormhole".

Programming it: TT-Metalium

tt-metal is the open low-level SDK. You write **three kernels per core**, coordinated by **circular buffers** (CBs) in SRAM:



CBs are FIFO producer–consumer queues; each stage blocks until data is ready — giving safe, asynchronous, **pipelined** execution across the cores.

Native tiles

Everything is a **32×32 tile** (BF16 = 2048 bytes) — the hardware's native unit, matching the FPU. No cache hierarchy: data placement is **explicit**, giving deterministic, predictable performance.

Table of Contents

The central challenge

AX needs gradients in three directions — but the hardware only does **tile-based matrix multiplies on a fixed 32×32 grid**.

- x, y: a matmul along tile columns / rows — fine.
- z: D must contract **across** tiles — no direct hardware path.

On a CPU the z case is trivial — just reorder the loop (but that's what causes the memory stalls). On the Tensix there is no such flexibility.

A hard rule

Avoid shuffling data using the baby RISC-V cores — far too slow. Use the NoC and the FPU's hardware transpose for reordering.

Approach: Pack 4 elements/tile

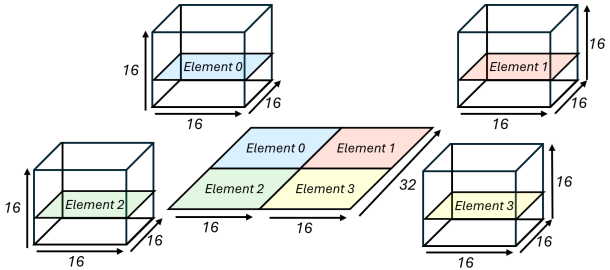


Figure: Each element's slice is only 16×16, but the tile is 32×32. Packing **four** elements per tile wastes no compute — the FPU works all four elements at once.

Table of Contents

Initial results: a reality check

Configuration	Performance (GFLOPS)	Power draw (Watts)
1 Tensix core	2.68	13.20
64 Tensix cores	3.50	15.70
128 Tensix cores	2.90	21.40
1 CPU core	5.38	65.16

Table: Initial port: performance and power comparison between Wormhole (BF16) and the CPU (FP32) with NX=16 and 800 elements. Wormhole: host code Clang 20, device code TT-Metal v0.65; CPU: GCC 7.4.

Slower than a *single* CPU core — even with 128 Tensix cores

1.5–2× slower, and adding cores barely helps. The naive port leaves the architecture's advantages on the table — **something is holding us back.**

Examining the bottleneck

Elements	Phase 1 (ms)	Phase 2 (ms)	Phase 3 (ms)
800	18	177	23
10 000	19	1324	15
100 000	32	13058	23

Table: Runtime for each phase of the AX kernel on 128 Tensix cores (BF16) for different problem sizes.

Phase 2 dominates — and explodes with size. Phases 1 and 3 are flat and fast. The z-gradient is the problem.

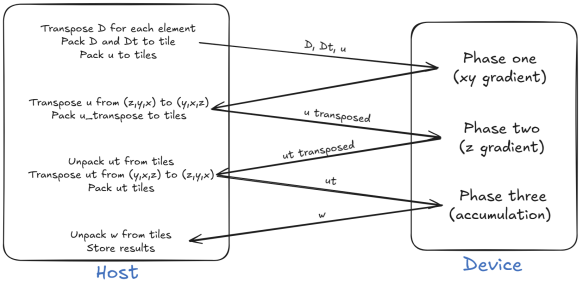


Figure: The 3-phase pipeline — interaction between the host and device.

Diagnosis and fix: kill the host round-trip

Baseline phase 2 bounces u through the host: transpose (z,y,x)→(y,x,z), pack, BF16↔FP64 conversions — all in single-threaded host loops — plus **three full-field PCIe Gen3 transfers** (6.4 MB each at 800 elements). **The bottleneck isn't the maths; it's data movement.**

1. Host fusion

Fuse the data-shuffles to operate *in place*; stay in BF16 throughout (no FP64 round-trip). ∼2× performance but the three PCIe transfers remain.

2. Device-side transpose

u is already on-device — transpose on-chip via the FPU's hardware transpose. **Eliminates all three PCIe transfers** and the host loops.

The shift

Bottleneck moves from **PCIe bandwidth** to on-chip **NoC bandwidth** — an order of magnitude faster.

The three versions, side by side

Elements	Baseline (GFLOPS)	Host fusion (GFLOPS)	Device transpose (GFLOPS)
800	2.90	5.54	5.59
10 000	5.81	11.96	91.81
100 000	6.02	13.69	**242.97**

Table: Performance comparison for baseline and two optimised versions for different problem sizes across 128 Tensix cores (BF16).

Device transpose scales dramatically: the architecture **favours large, pipelined problems**. At 100 000 elements, it is **40**× the baseline.

Final results: the payoff

Hardware	Precision	Elements	Performance (GFLOPS)	Power (W)	Efficiency (GFLOPS/W)
128-core Tensix	BF16	800	5.59	21.30	0.26
		10 000	91.81	21.40	4.29
		100 000	**242.97**	21.40	**11.35**
24-core Xeon CPU	FP32	800	159.36	114.70	1.39
		10 000	161.04	137.25	1.17
		100 000	161.93	141.34	1.15
	FP64	800	106.22	131.79	0.81
		10 000	110.69	143.49	0.77
		100 000	110.90	144.07	0.77

Table: Final performance and power comparison between Wormhole (BF16) and CPU (FP32 & FP64) with NX=16 across 800, 10 000, and 100 000 elements. Wormhole: host code Clang 20, device code TT-Metal v0.65; CPU: GCC 10.2.[2]

[2] Wormhole power: *tt-smi* ASIC core telemetry; CPU power: Intel RAPL.

Final results: the headline

The headline

At 100k elements, Wormhole delivers $\sim$**10**$\times$ **the power efficiency** of the 24-core CPU (BF16 vs the CPU's fastest format, FP32 — the Xeon has no native BF16) — widening to $\sim$**15**$\times$ against FP64 — while *outperforming* the CPU on raw GFLOPS (1.5–2.2$\times$), at a near-constant $\sim$**21 W**.

- Tensix efficiency **scales with the workload** — 0.26 $\rightarrow$ 11.35 GFLOPS/W.
- Both CPU configurations are **flat** across all problem sizes — memory-bandwidth saturation, exactly as the profiling predicted.
- The gap is widening, not closing.

Table of Contents

Conclusions & what's next

- We ported the Nekbone AX kernel to a Tenstorrent RISC-V accelerator; the z-gradient bottleneck was solved by keeping data on-device via the FPU's hardware transpose.
- At scale it **matches or beats** a 24-core CPU at a **fraction of the power**.
- **Caveats**: BF16 only — FP32 via the SFPU is future work; only the 100k-element run beats the CPU on raw GFLOPS, and at 800 elements the Wormhole loses on power efficiency too.

Insights for the RISC-V-for-HPC community

Hardware: the architecture rewards *large, pipelined* problems. **Software:** keep data resident; exploit on-chip features (NoC, hardware transpose, register accumulation).

Next: SFPU vector unit for FP32; multi-card scaling; next-gen **Blackhole** (work already underway).

Acknowledgements

- This research was funded by the **HPC-R project** (EP/Z533701/1).
- CPU runs were performed on **NextGenIO**, funded by the EU Horizon 2020 research and innovation programme under grant agreement No. 671591.

Thank you!