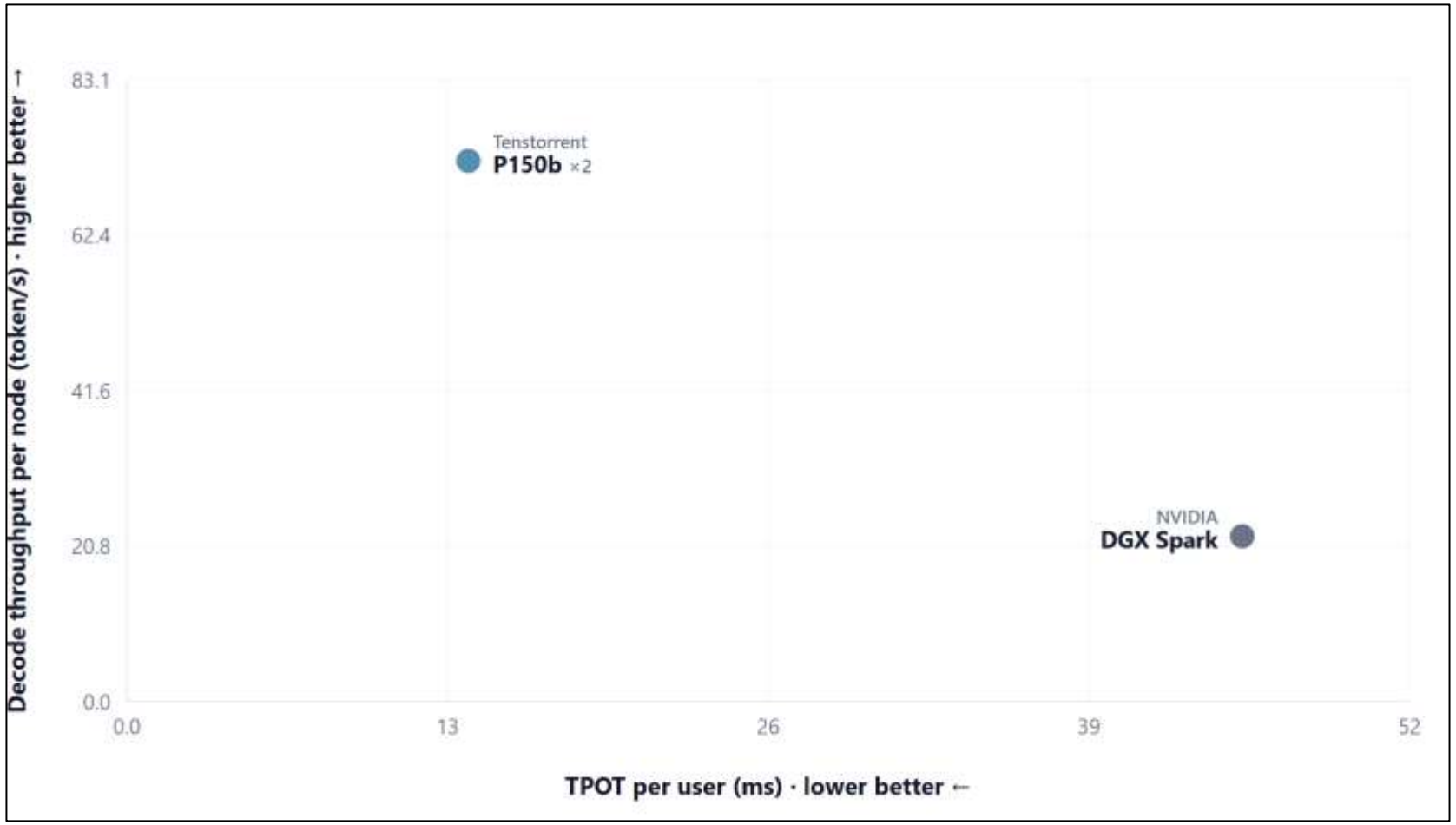
A RISC-V SIMULATOR FOR THE TENSTORRENT TENSIX ARCHITECTUR

Nick Brown, EPCC @ University of Edinburgh
Daniyal Arshad, EPCC @ University of Edinburgh

epcc

Tenstorrent is competitive for LLMs



- Comparison for Qwen3-4B from https://www.teasbench.com/
- Two p150 cards are $2800
- DGX Spark is $4699

- Compares Time Per Output Token and decode throughput between two cards

epcc

Nekbone on Tenstorrent

```
procedure AX(nx, D, G, u)
    // u is a field on an nx × nx × nx element grid
    // D is the 1D spectral differentiation matrix (nx × nx)
    // G is the 3 × 3 symmetric geometric factor matrix

    // Directional gradients
    for each z-layer:
        u_r(z) ← D · u(z)          // x-gradient: matrix × layer
        u_s(z) ← u(z) · D^T        // y-gradient: layer × matrix

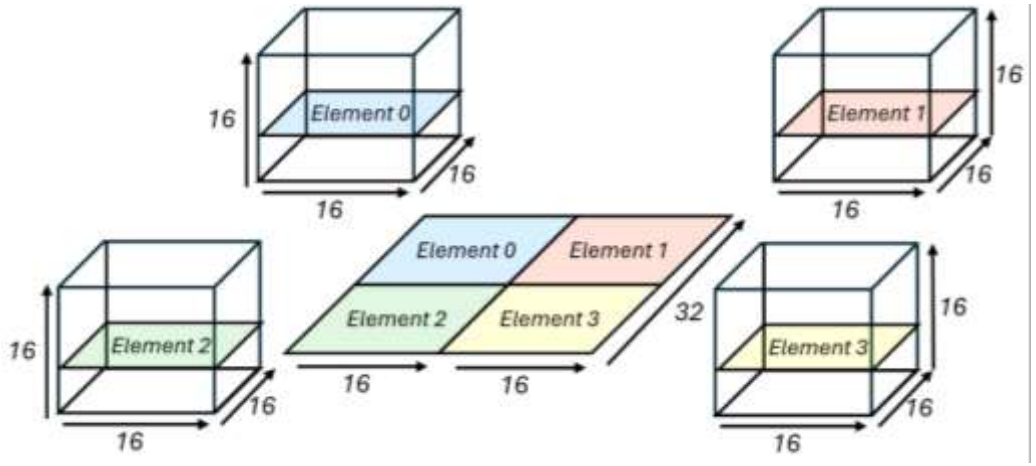
    for each (x,y) point:
        u_t(x,y,:) ← D · u(x,y,:)  // z-gradient: matrix × column

    // Apply geometric factors
    for each point (x,y,z):
        [u_r, u_s, u_t] ← G · [u_r, u_s, u_t]

    // Accumulate with transposed operator
    for each z-layer:
        w(z) ← D^T · u_r(z) + u_s(z) · D + u_t(z) · D
    return w
```

https://arxiv.org/pdf/2608.22964

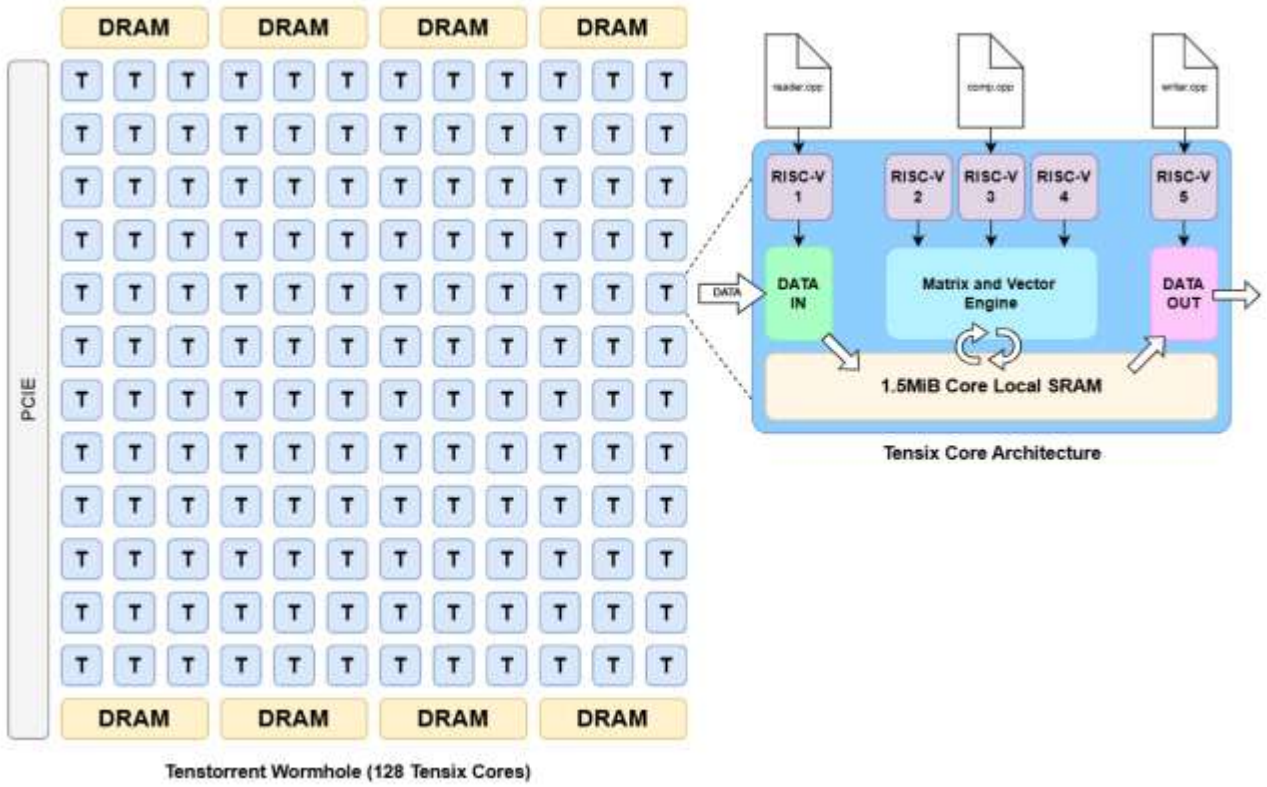
- Spectral element method, comprises lots of small matrix multiplications



Description	Elements	Performance (GFLOPS)	Power (Watts)	Efficiency (GFLOPS/W)
128-core Tensix	800	5.59	21.30	0.26
	10000	91.81	21.40	4.29
	100000	**242.97**	**21.40**	**11.35**
24-core Xeon CPU	800	159.36	114.70	1.39
	10000	161.04	137.25	1.17
	100000	161.93	141.34	1.15

E4 have ported astrophysics to Tenstorrent, 2x performance & energy efficiency https://arxiv.org/pdf/2509.19294

Tenstorrent's RISC-V accelerator is very interesting



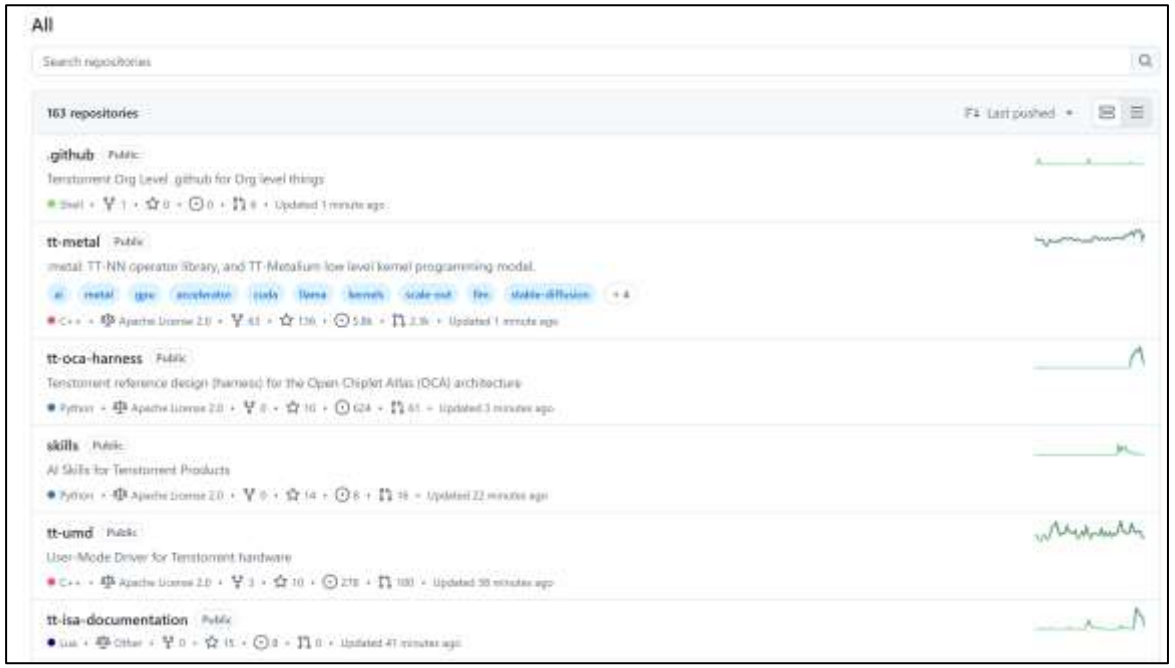
Courtesy of Jake Davies, EPCC

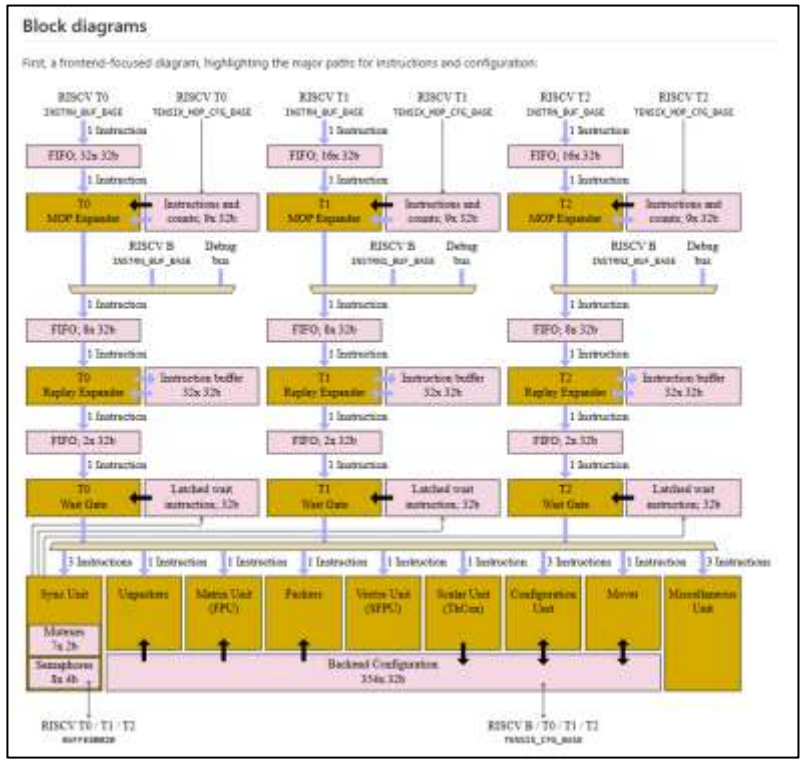
- Built around Tensix cores, each of these contains five RISC-V *baby* cores
- Drive a co-processor which contains dedicated matrix and vector units
- RISC-V core(s) feed local SRAM from external DDR
- Programmed with TT-Metal which is open source
- Detailed ISA documentation also available

epcc

Committed to open source

- Tenstorrent release their software stack open source on GitHub
  - And have even released their own simulator (released after ours)
- They have written detailed ISA documentation at https://github.com/tenstorrent/tt-isa-documentation





From Tenstorrent ISA documentation

- RISC-V itself is open so it is clear exactly what is implemented here

Therefore, based on all this open, public, information can someone external develop a simulator for the architecture that will run binaries?

epcc

The Wolfpine community Tensix simulator



Built only from public information
ISA docs, open source software and hardware measurements we have done

Modular and extensible in Python
Provides hackability and transparency

Wormhole and Blackhole
Run unmodified TT-Metal programs seamlessly

Introspection for performance tracking
Opt-in cost model, event streams, tracing, timelines to help programmers understand performance and debug

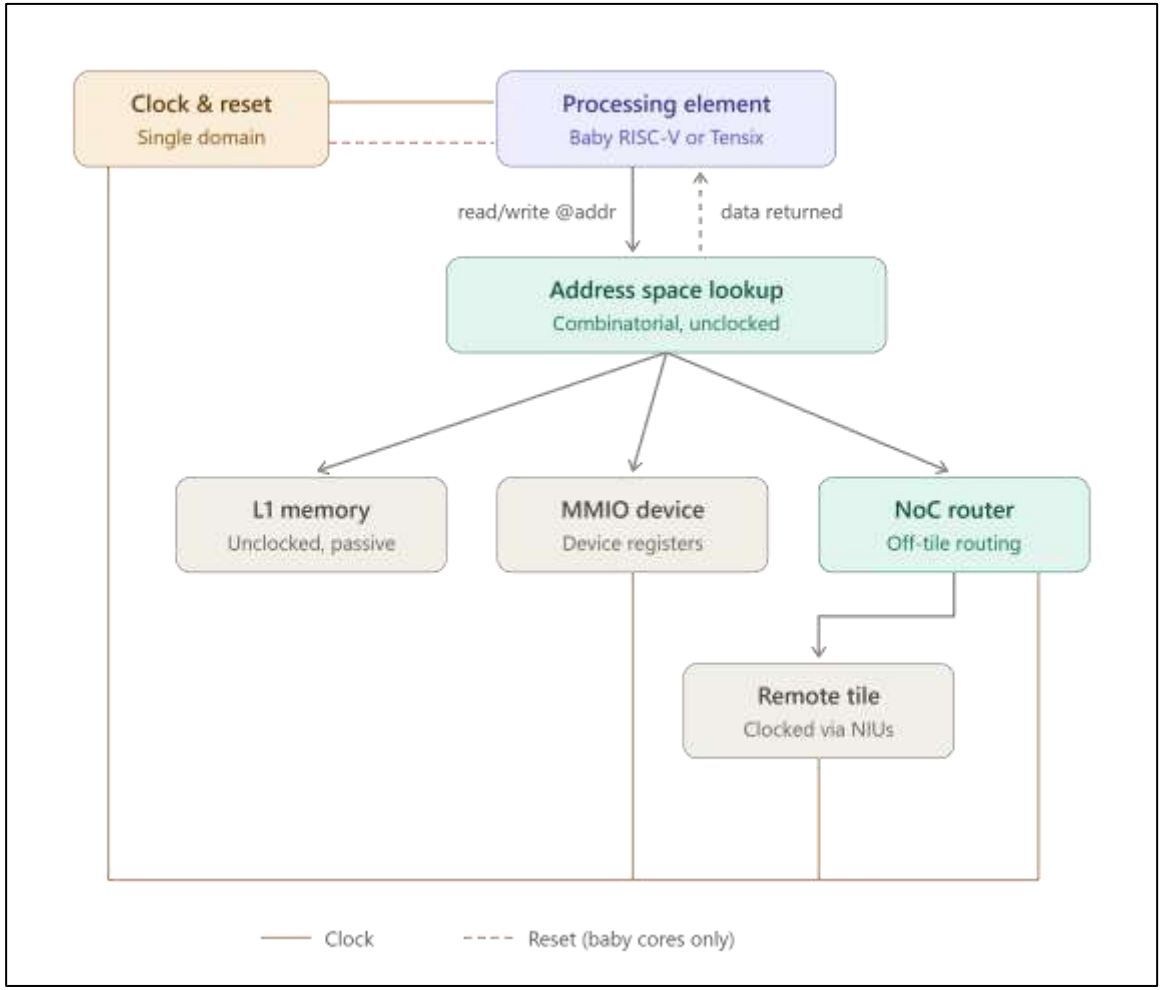
	ttsim (Tenstorrent)	Wolfpine
Developed by	Vendor	Community
Contributions	Filtered releases, none accepted	Welcomed!
Language	C++	Python
Aim	Golden reference, bit-exact	Debugging and performance analysis
Timing insight	Limited	Provenance-tracked cost model

Available on PyPi, *pip install wolfpine*
https://github.com/hedgehope/wolfpine

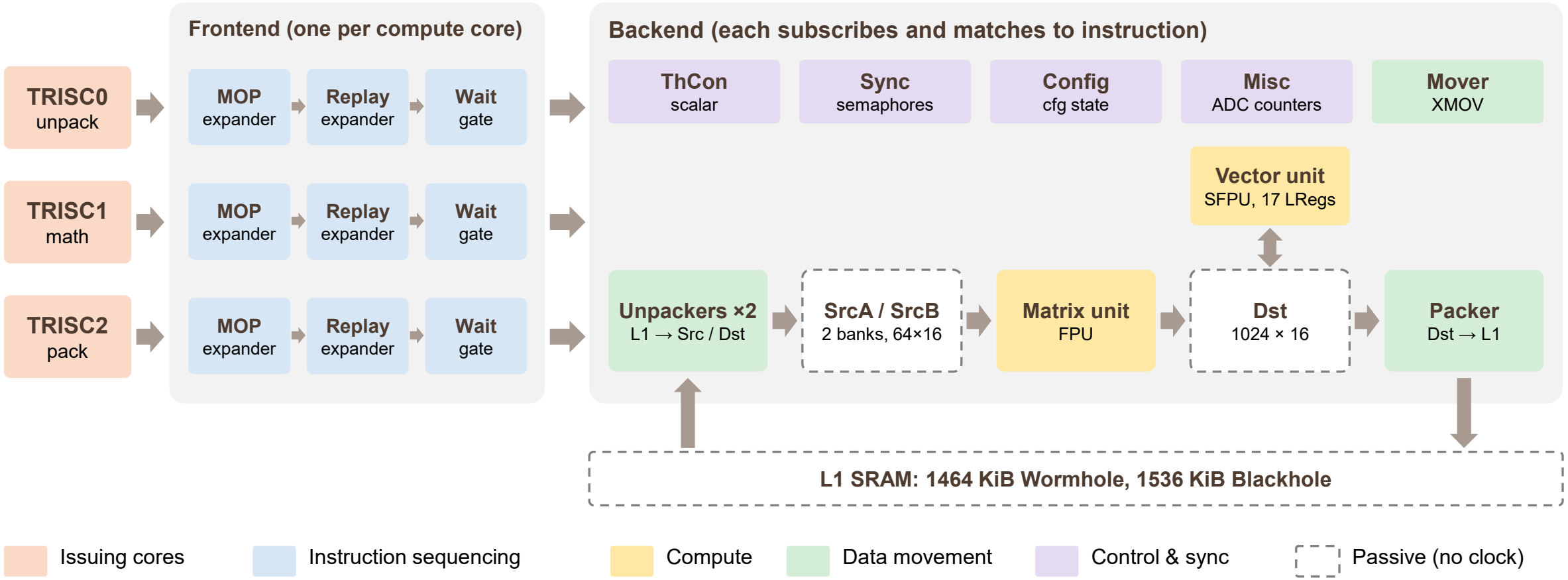
epcc

Overarching architecture view

- **Active** blocks (baby cores, Tensix) are driven by clock and reset. **Passive** blocks (e.g. L1) act on demand
- Blocks register address ranges and a lookup arbitrator routes each access
  - Layered views with per-core private memory plus tile-shared L1, NIU and Tensix config
- A target can return a stall, so the core retries and this is how mailboxes and sync are modelled
- The NoC is just another block
  - Remote accesses re-enter the same lookup on the far tile
- All the memory infrastructure and RV baby core support was around 2000 LoC (the core 1700 LoC)



Simulating the Tensix co-processor model

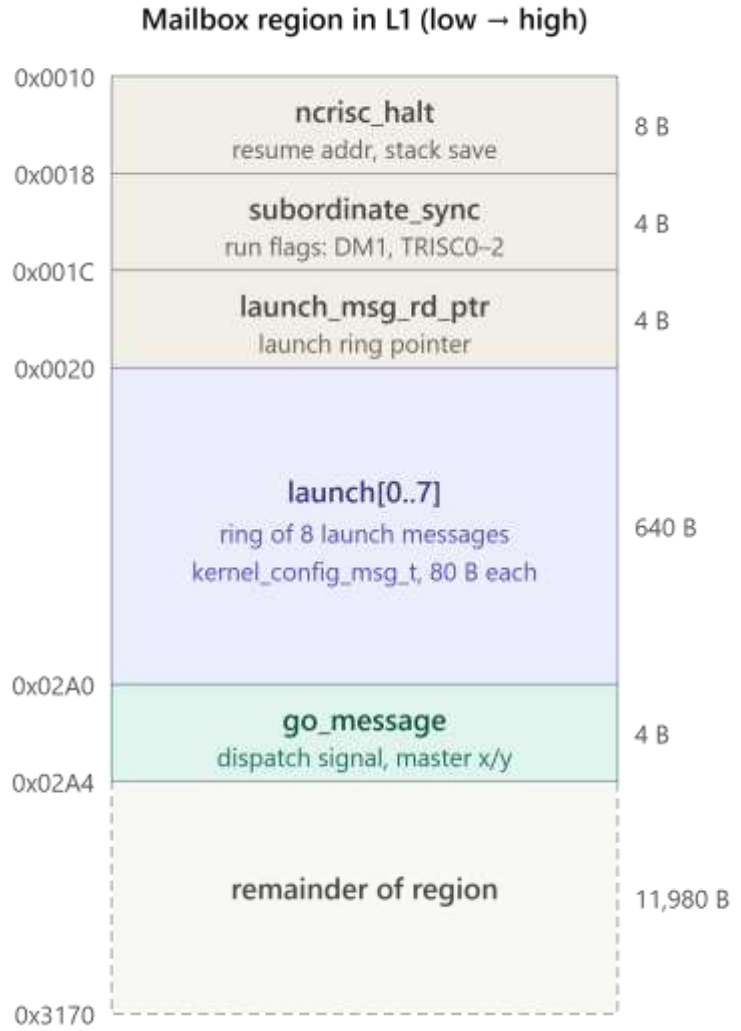


- Cores push instructions via bespoke RV extension with ttinsn instruction
- 137 Tensix instructions and we implement 111
  - 103 for Wormhole and Blackhole, 34 are Blackhole specific
  - Support done on demand, remaining ones never emitted by tt-llk
  - Approx. 9000 lines of Python code here

epcc

Integrating with TT-Metal

- TT-Metal first launches firmware on the Tensix cores
  - These spin waiting for commands and kernels
  - TT-Metal then JIT compiles code into binaries which are launched
- Initially was very hacky as we dumped out the mailbox memory addresses manually and hardcoded these
  - As Tenstorrent's simulator was released we were able to hook into that same mechanism



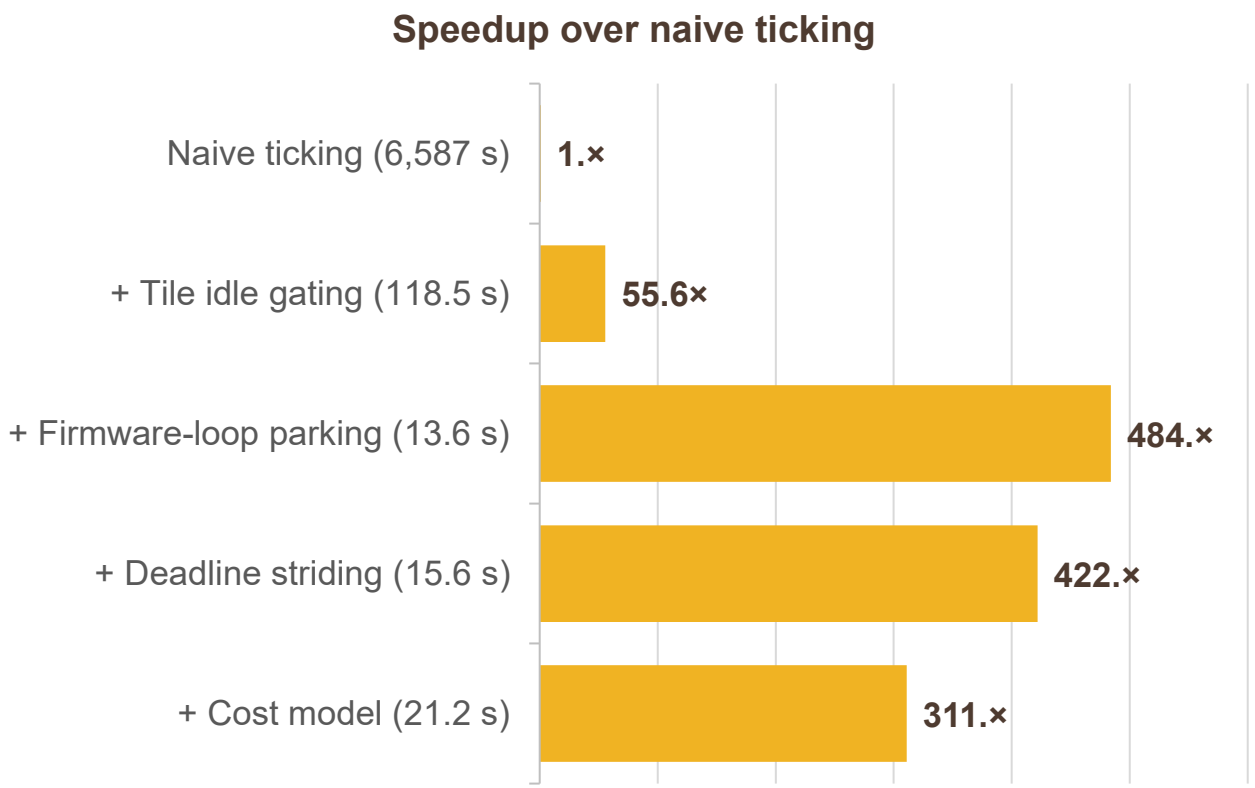
export TT_METAL_SIMULATOR=/path/to/wolfpine




```
wolfpine run wormhole  ./metal_example_add_2_integers_in_compute   # simulated Wormhole
wolfpine run blackhole ./metal_example_add_2_integers_in_compute   # simulated Blackhole
wolfpine run hw        ./metal_example_add_2_integers_in_compute   # the same binary on hardware
```

epcc

Python is great, but slow!



The pad_multi_core benchmark on a simulated p150 Blackhole with four Tensix cores. Rows are cumulative

- Assumed would simply multi-thread & Python JIT compilation to speed things up but not enough
- **Tile idle gating:** Each block in a tile reports whether it is idle or not, if all are idle then the tile is idle (this is a very common state)
  - Therefore don't do any clock ticking, instead wake up only on message on the NoC, a read or write, or a reset
- **Firmware-loop parking:** Recognises firmware spinning on the go-message and instead parks the core to idle (as above)
- **Deadline striding:** Each tile reports the next cycle that is needed for internal processing. If all are idle then the entire device can stride to this
  - This comes from the cost model, which will make specific units idle for N cycles e.g. waiting on NoC or memory transfers

epcc

Perfbench microbenchmark

Characterise the silicon in cycles

tensixbench | riscvbench | nocbench | nocreadbench | dramratebench

Tensix instruction costs including baby core issue, branches, fetch and instruction queue. NoC latency against hops and shared links. Read-rate limits. DRAM channel rates

Compare mechanisms: silicon vs Wolfpine

mechbench | nocevbench | retirebench

Decomposed a program identically on both and measure Tensix performance counters, TT-Metal NoC event traces

Energy usage

energybench

Steady-state board power of repeated kernels against an idle baseline

- Can't get everything from the documentation
- Run a single program on silicon and Wolfpine
- Average over many input sizes and many runs
- Also used to validate what is in the documentation

Raw measurement data and scripts at https://github.com/hedgehope/wolfpine

epcc

Cost model for performance insights

- Declarative per-unit cycle tables
  - Tensix back-end
  - NoC
  - DRAM
  - baby cores
  - L1
- Costs are spent where the hardware stalls
- NoC: endpoint cost + per-hop cost on the unidirectional torus; flits occupy links, so transfers contend

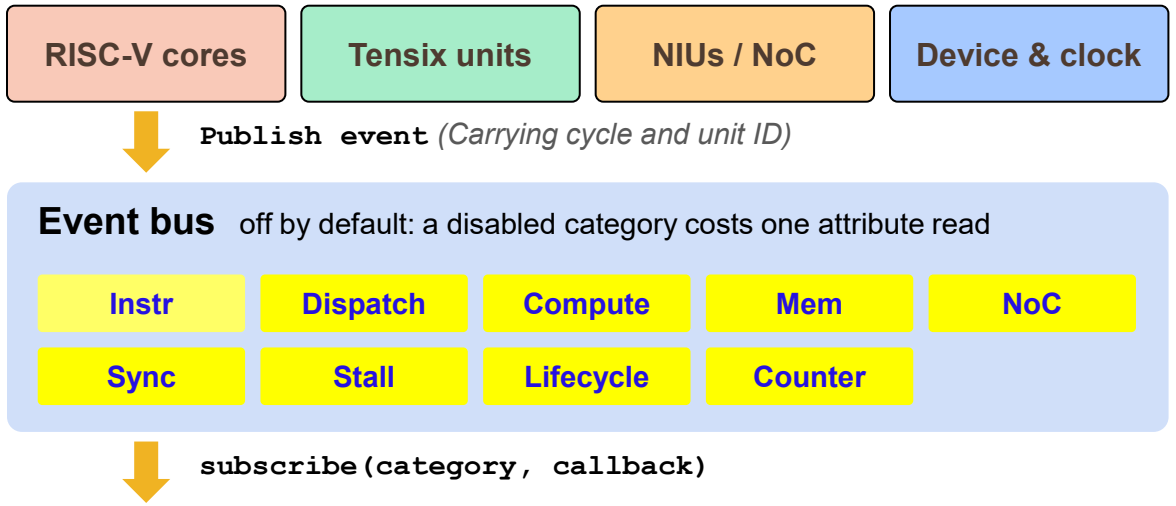
- Every number comes from the documentation or our own measurements
  - We developed a range of microbenchmarks that ran on Wormhole and Blackhole to record these measurements

Operation	p150 cycles	Wolfpine	Error %
FPU MVMUL	1.061–1.067	1.00	6
THCON ADDDMAREG	2.973	3.00	0.9
RISC-V multiply	1.985	2.00	0.8
RISC-V L1 load	8.098	8.00	1.2
RISC-V divu	**33.001**	**6.00**	**82**
NoC latency per hop	8.85–9.03	9.00	0.3–1.7

divu takes 6–33 cycles depending on operands; the model currently assumes the low end

epcc

Exposing the cost model



Switched on by	Subscribes to	Output
TT_SIM_TRACE_PERFETTO	7 categories	Perfetto timeline
TT_SIM_TRACE_COMMITLOG	Instr	Spike commit log
TT_SIM_TRACE_LCOV	Instr	LCOV coverage
TT_SIM_TRACE_MEMORY	Mem	Cachegrind
TT_SIM_TRACE_NOC	NoC	Parquet, per packet
TT_SIM_TRACE_COUNTERS	All → Counter	Parquet, sampled
TT_SIM_TRACE	All	JSON lines

Capability	HW	ttsim	Wolfpine
Per-instruction retirement trace	✗	✗	✓
Register writeback log (Spike diff)	✗	✗	✓
NoC transaction stream	P	✗	✓
Timeline export (Perfetto)	✓	✗	✓
Source-level cycle attribution	P	✗	✓
Memory-range snooping	✗	✗	✓
Deadlock diagnosis	✓	✗	✓
Conformance / UB flagging	✗	✓	✓

P = partial. ttsim deliberately exposes no trace, dump or timeline entry point

epcc

Evaluation



- TT-Metal v0.74, Wolfpine v2.0 and ttsim v1.9.6
- Simulator results are measured on a 12th Gen Intel Core i7-12700H with 48 GiB of RAM, and CPython 3.12.13

- Hardware results are measured on a Blackhole accelerator
  - 120 Tensix tiles organised as a 12×10 logical worker grid,
  - Clock frequency of 1350 MHz E
  - Eight DRAM banks
  - Firmware bundle 19.6.0 and KMD 2.10.0

Benchmark	Size	Subsystems introduced
`add_2_integers_in_riscv`	1 uint32	Baby cores, L1, launch/go path
`loopback`	51000 bf16	NIU programming, NoC, DRAM tiles
`eltwise_binary`	64 tiles, 32x32, bf16	Unpack → FPU → pack pipeline, circular buffers
`matmul_single_core`	128^3 bf16	Matrix unit, Dst accumulation
`matmul_multi_core`	640^3, bf16, 40 Tensix	Multi-tile dispatch, inter-tile NoC, interleaved DRAM

Result accuracy and performance

Benchmark	Result accuracy (PCC) Wolfpine	ttsim	p150
add_2_integers	1	1	1
loopback	1	1	1
eltwise_binary	1	1	1
matmul_single_core	0.9982	0.9981	0.9833
matmul_multi_core	0.99997	0.99995	0.9821

- Comparing against the program's host-computed golden reference and reported as the Pearson Correlation Coefficient (PCC) where a value of 1 represents an exact match

- Wallclock of Wolfpine and ttsim simulators
- Slowdown compared to running on actual p150 hardware

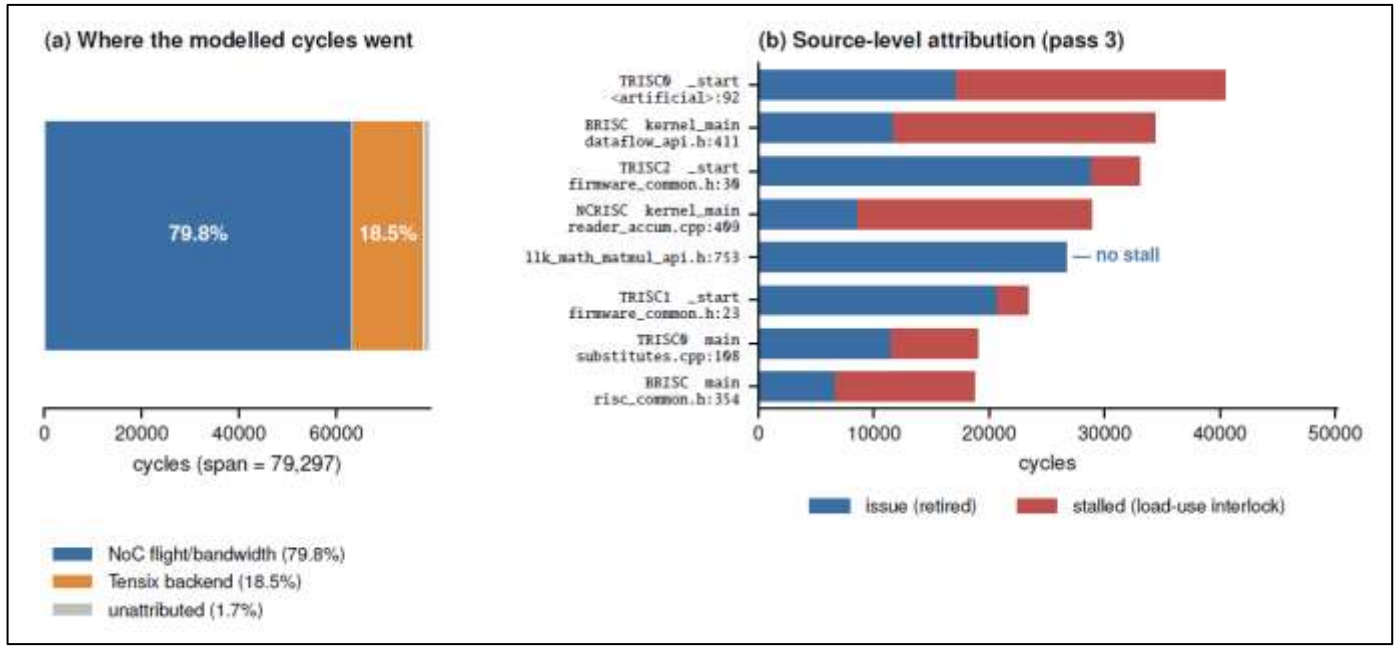
Benchmark	Wall-clock (s) Wolfpine	ttsim	Slowdown vs p150 Wolfpine	ttsim
add_2_integers	8.4	1.8	3.5×	0.8×
loopback	8.8	2.4	3.7×	1.0×
eltwise_binary	14.2	2.4	3.8×	0.6×
matmul_single_core	32.4	7.2	9.3×	2.1×
matmul_multi_core	786	37.8	192×	9.2×

Nekbone

	Wolfpine		p150 Blackhole		Error
Pass	Cycles	Runtime (μs)	Cycles	Runtime (μs)	%
1	18,304	13.56	16,605	12.30	10.2
2	11,926	8.83	13,120	9.72	9.1
3	29,836	22.10	32,784	24.28	9.0
Total	**60,066**	**44.49**	**62,509**	**46.30**	**3.9**

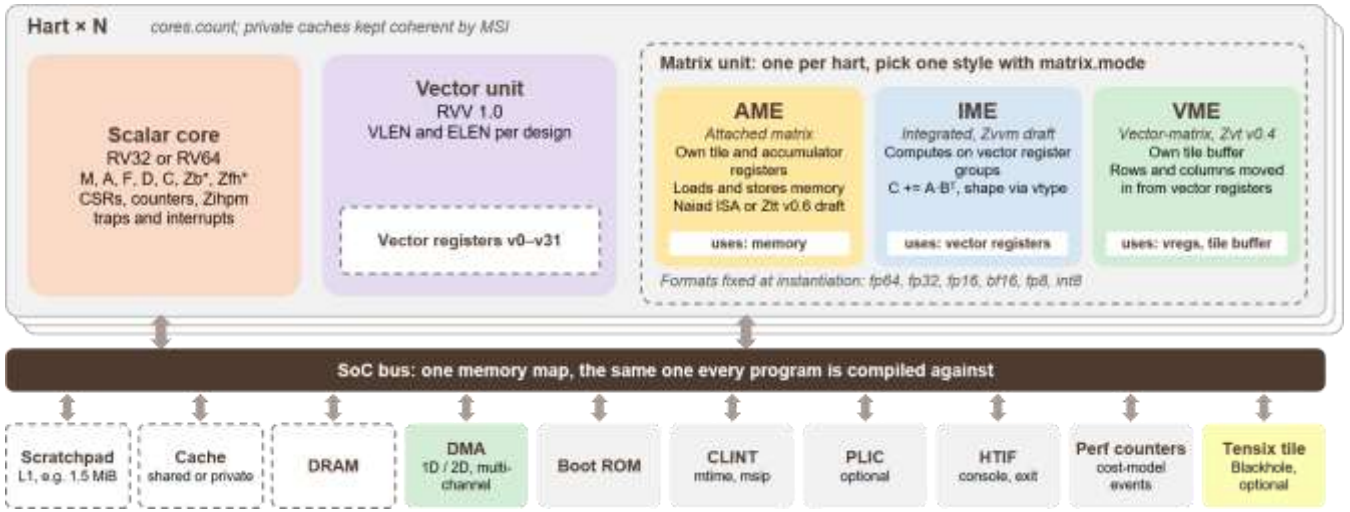
- Ran the basic version of work previously ported to n300 on p150 and Wolfpine
  - N=16, 16384 elements
- Three phases to the CG kernel involving vector and matrix-matrix operations

- Around 10% difference of (predicted) cycle count between p150 and Wolfpne
- To the right is performance analysis report showing what time is spent where



Conclusions and future work

- It is absolutely possible to build an independent simulator based on the openness of RISC-V and Tenstorrent's public documentation/code
- And we can go further to bring in a cost model based on documentation/hardware measurements
  - Absolutely key for HPC, and potentially less of a priority for Tenstorrent to provide in their own simulator
- Wolfpine is much simpler than Gem5



- Not just Tenstorrent
  - Have AME/IME/VME blocks here now too, and are using these to explore HPC workloads on RISC-V matrix extensions
  - Simply another target in Wolfpine

epcc