2:4 Structure Sparsity Support for Tensor Core on Open RISC-V Vortex GPU

Yanggon Kim, Eric Song, Blaise Tine
Department of Computer Science
University of California, Los Angeles

Contents

Vortex GPGPU Platform

Hardware

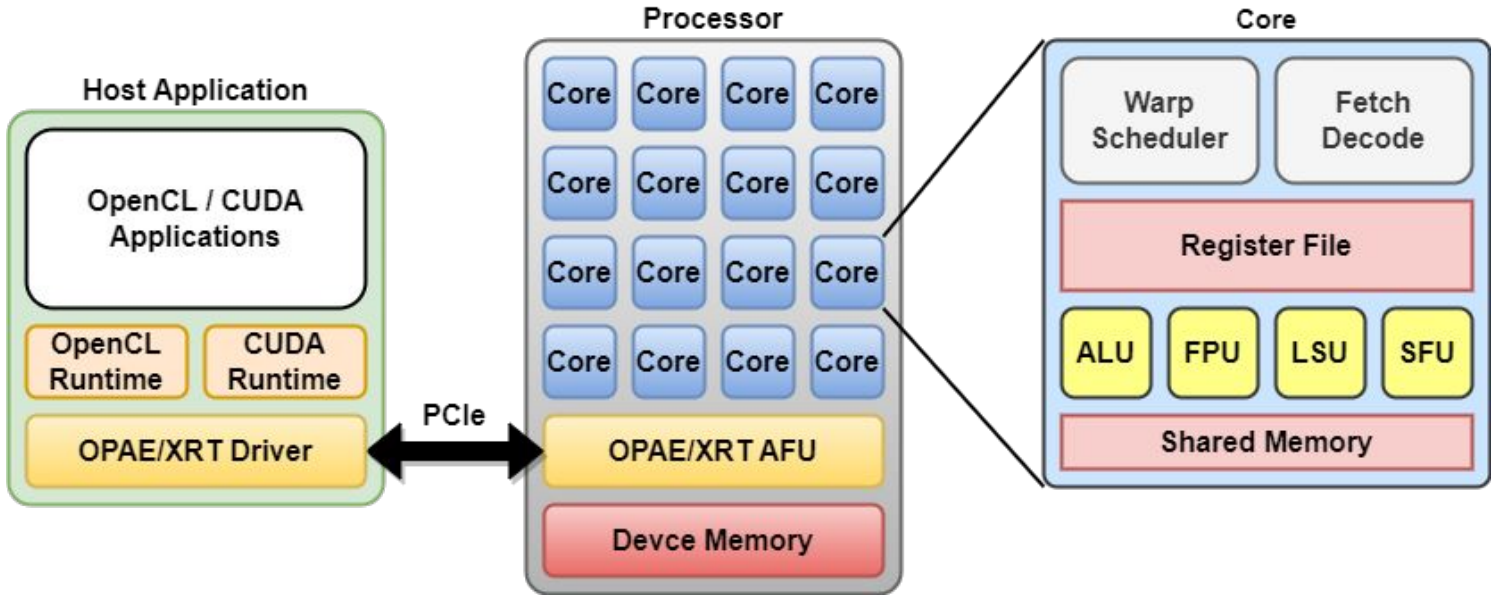
- Fully configurable multi-core GPU
- PCIe host-device interface
- Caches & shared Memory
- Floating-point Units
- 32 and 64-bit ISA

Open-source toolchain

- LLVM: Compiler
- POCL: OpenCL
- CuPBoP: CUDA
- OPAE: Intel FPGA API
- XRT: Xilinx FPGA API
- Verilator: RTL simulation
- Yosys: FPGA Synthesis

FPGA Synthesis

- Altera: Arria10, Stratix 10
- Xilinx: Alveo U50, U250, U280, U55C



RISC-V ISA Extension for Vortex

Threading model
- Wavefronts
- Threads

Thread Divergence Handling
- Split/Join stack based
- Thread predication

Memory model
- Global memory
- Shared memory

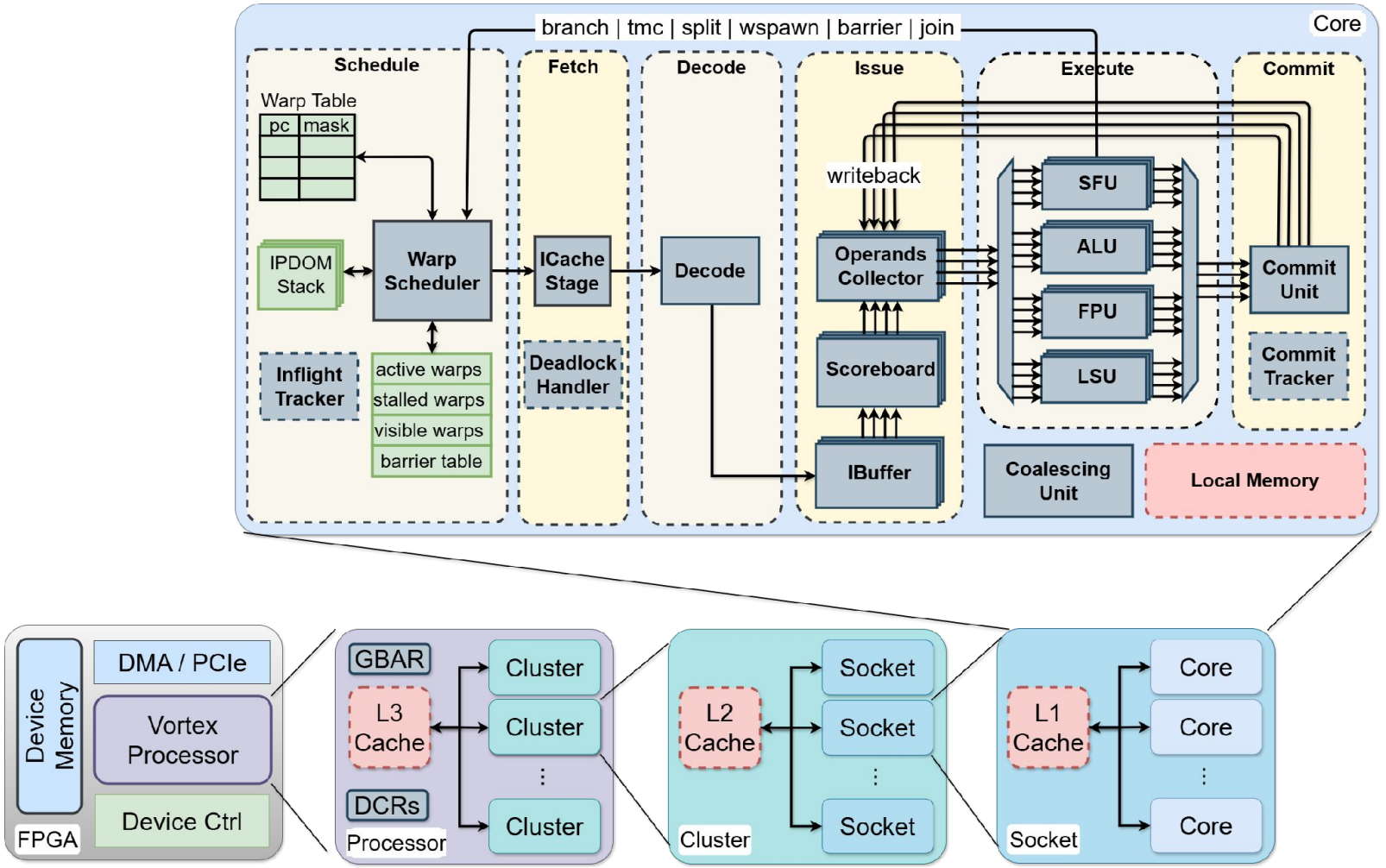
Register File
- RISC-V Integers and floats
- 32 * #warps * #threads regs

Synchronization
- Global and local barriers
- RISC-V fence

Instruction	Description
wspawn #warps, %PC	Wavefronts activation
tmc #threads	Thread mask control
#addr ← ***split*** #pred	Control flow divergence
join #addr	Control flow reconvergence
pred #pred, #mask	Thread predication
bar #id, #warps ***bar*** #id, #cores	Wavefront local barrier (positive id) Wavefront global barrier (negative id)

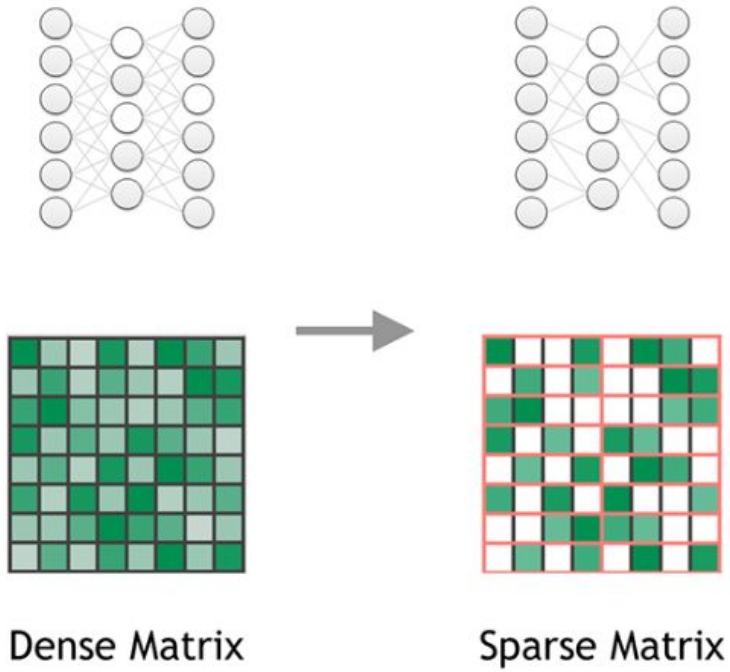
Vortex GPU microarchitecture

Six-stage RISC-V
in-order issue
out-of-order commit
Pipeline



Contents

2:4 Structure Sparsity

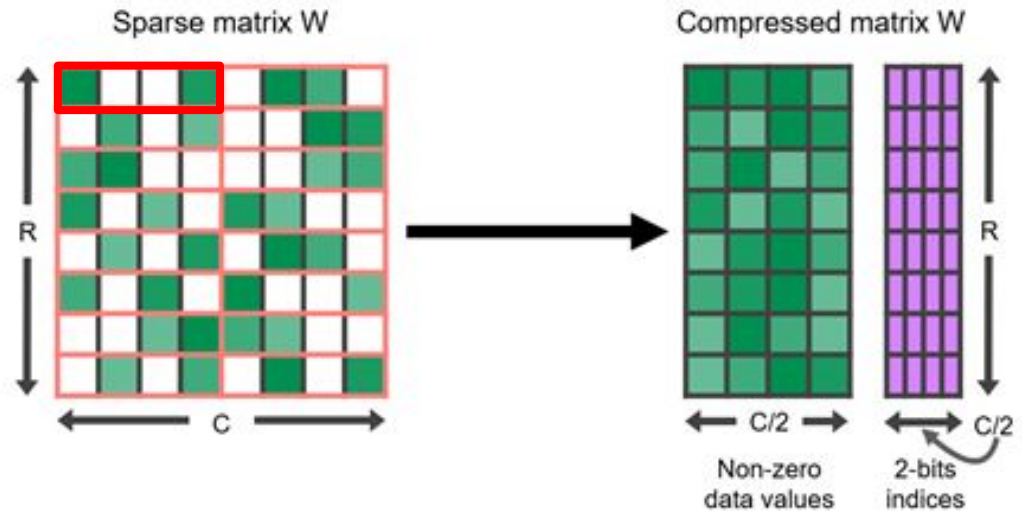


- For every four consecutive weights, two are retained and two are pruned.
- Weights are usually selected by magnitude to minimize accuracy loss.
- The predictable pattern enables efficient sparse-matrix acceleration.

image source : https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/

2:4 Structure Sparsity

metadata = 1001

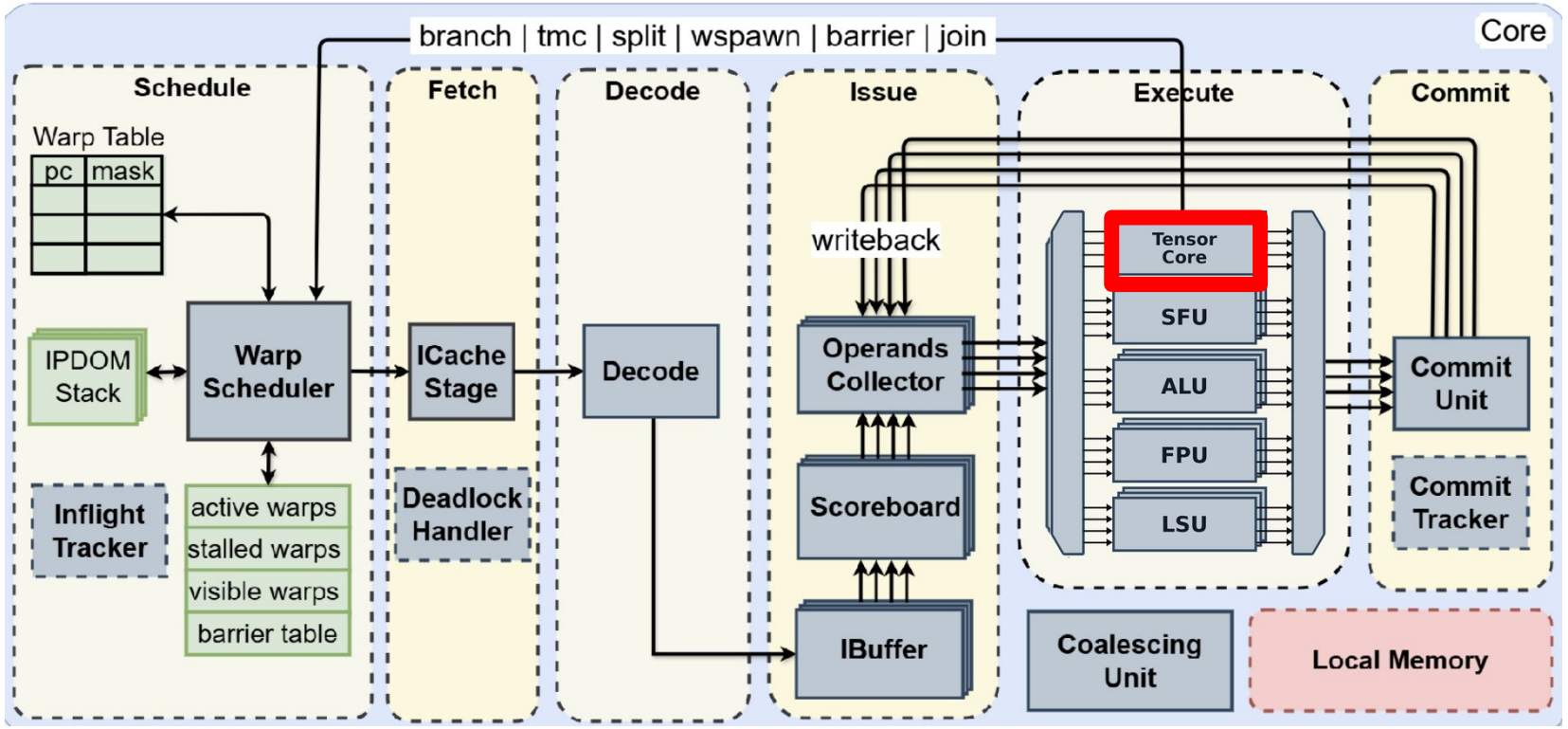


- Store only two nonzero values for each group of four weights.
- Metadata identifies where those values belong in the original matrix.
- Example: *1001* means that positions 1 and 4 are nonzero.
- The regular format reduces storage and enables hardware-efficient computation.

image source : https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/

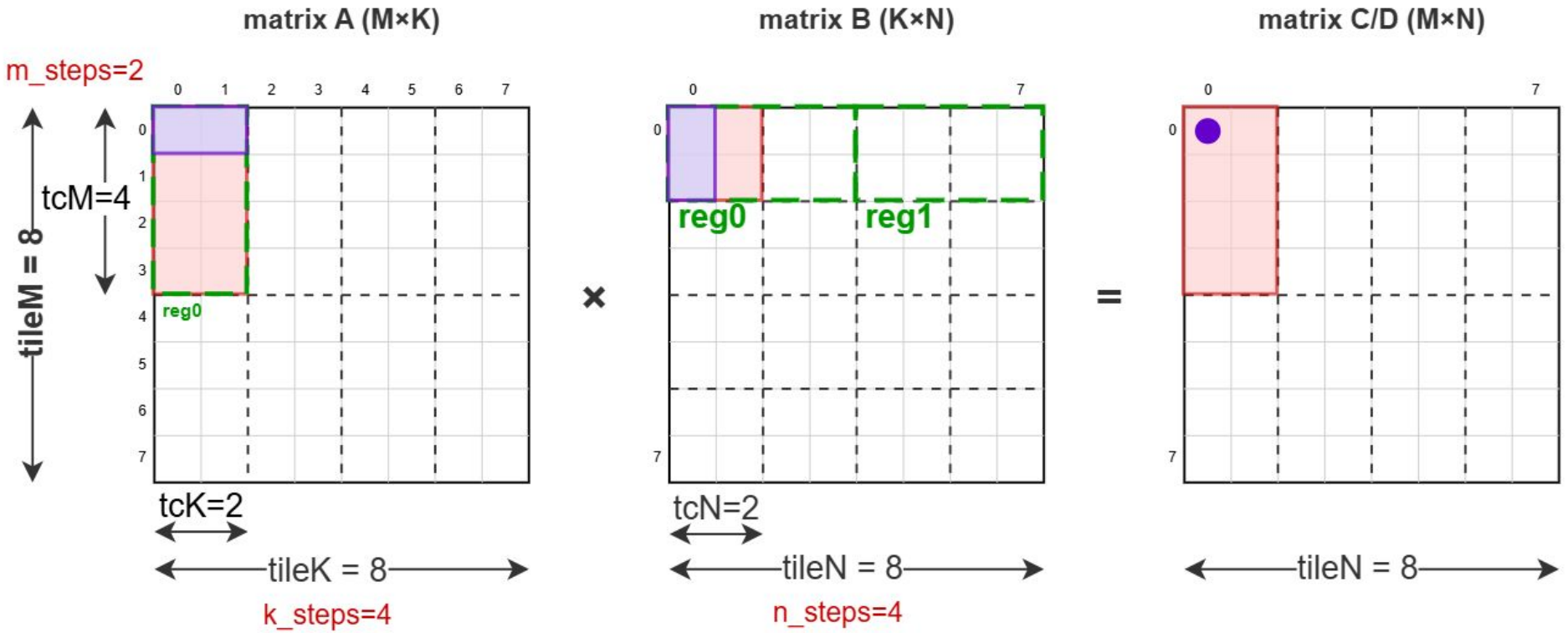
Contents

Tensor Core in Vortex



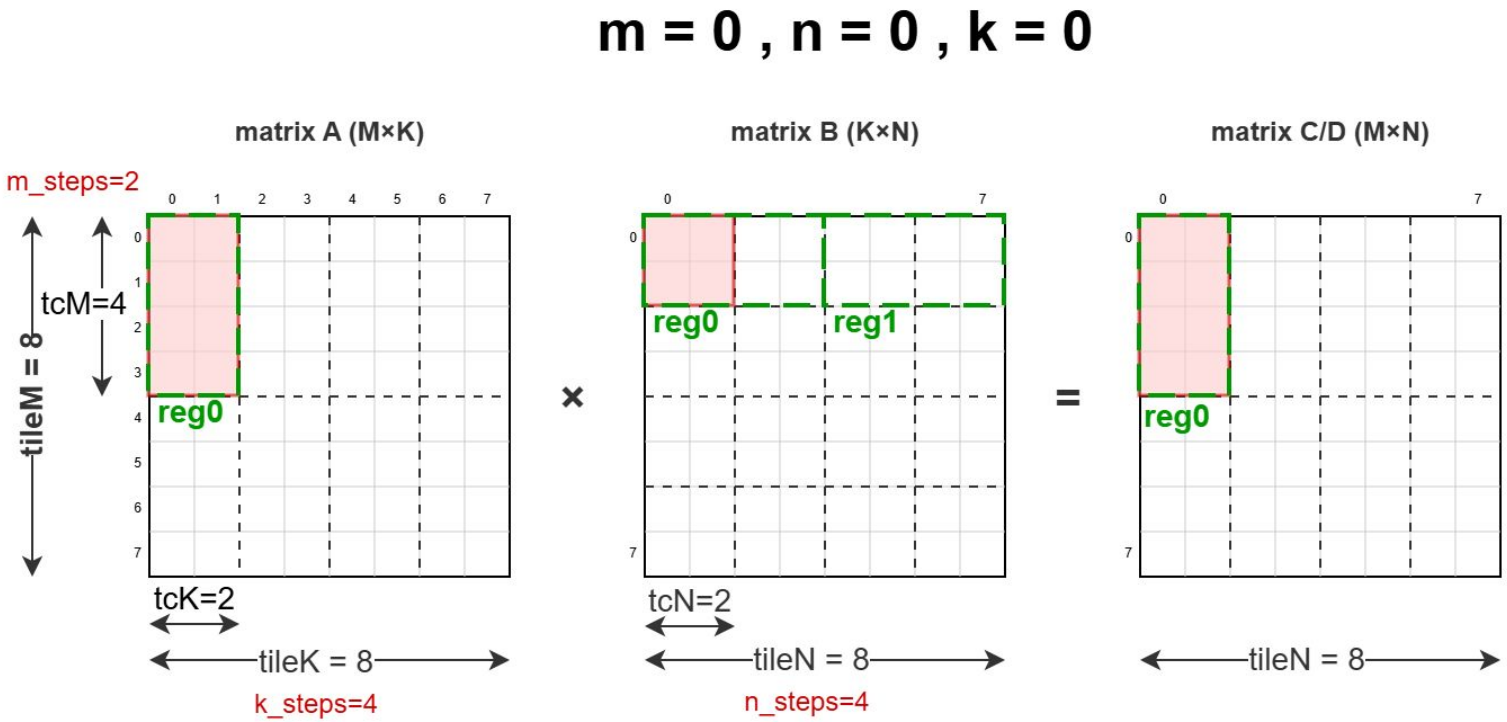
- TCU is added as a specialized matrix-compute unit in the execution pipeline.
- It reuses the existing warp scheduler, register file, and memory hierarchy.
- This integration provides high matrix-computation throughput with minimal changes to the core architecture.

Tensor Core Integration in Vortex



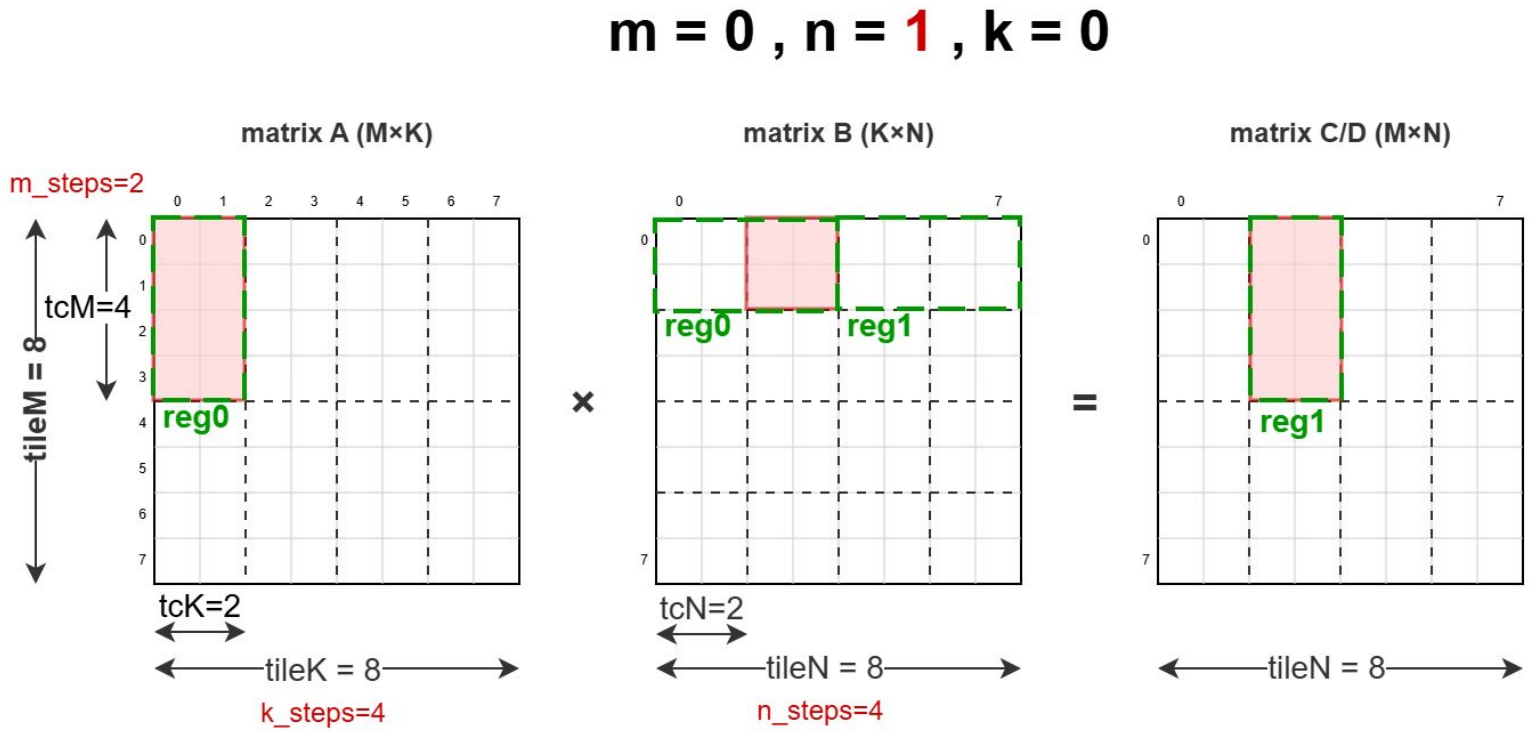
- TCU calculates matrix matrix multiplication via mma_sync instruction
- One instruction expands into multiple micro-ops
- Total number of micro-ops = m_steps * n_steps * k_steps
- Let's take a look at the 8 thread example

Tensor Core in Vortex



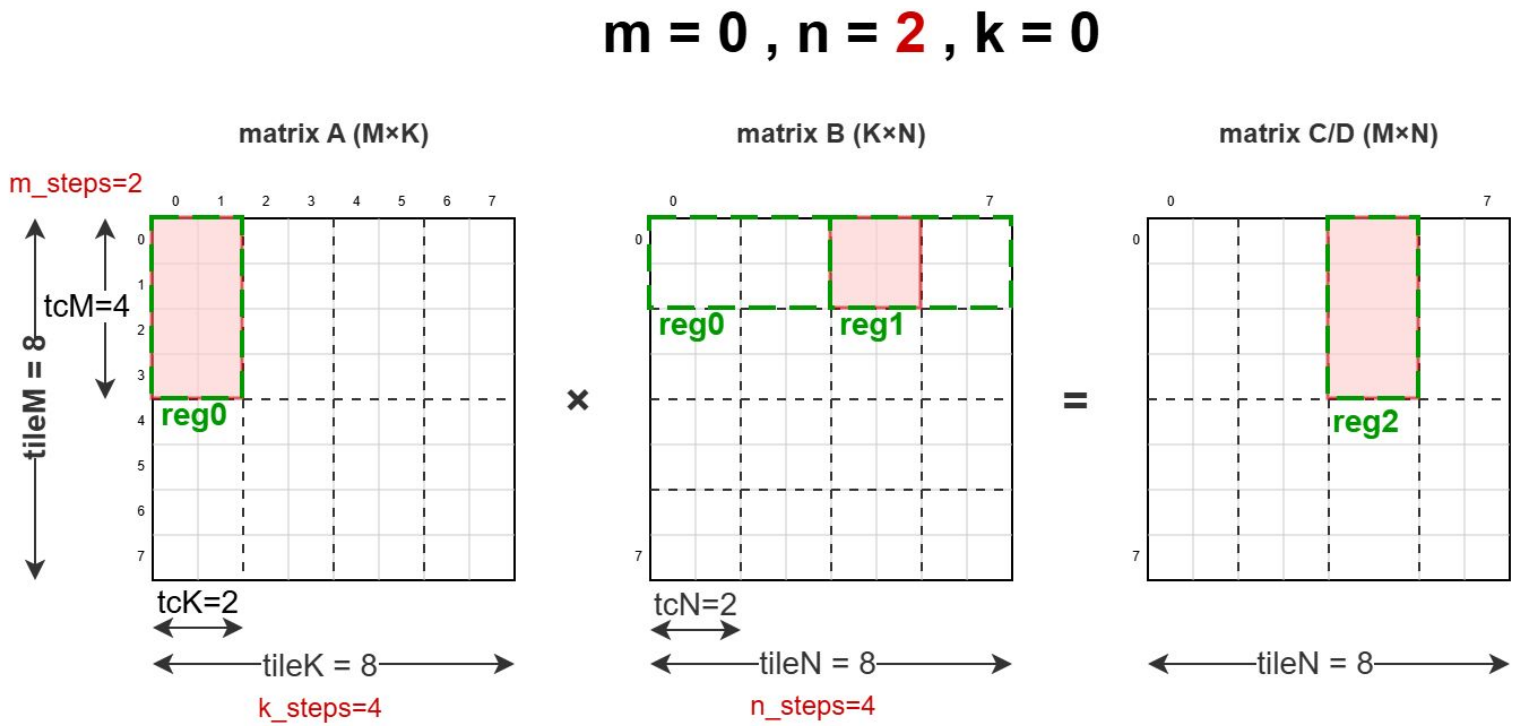
- At first micro-ops, we fetch a block from matrix A from register 0
- Also, fetch a block from matrix B from register 0
- And, do block-level matrix matrix multiplication and store results in register 0 in matrix C

Tensor Core in Vortex



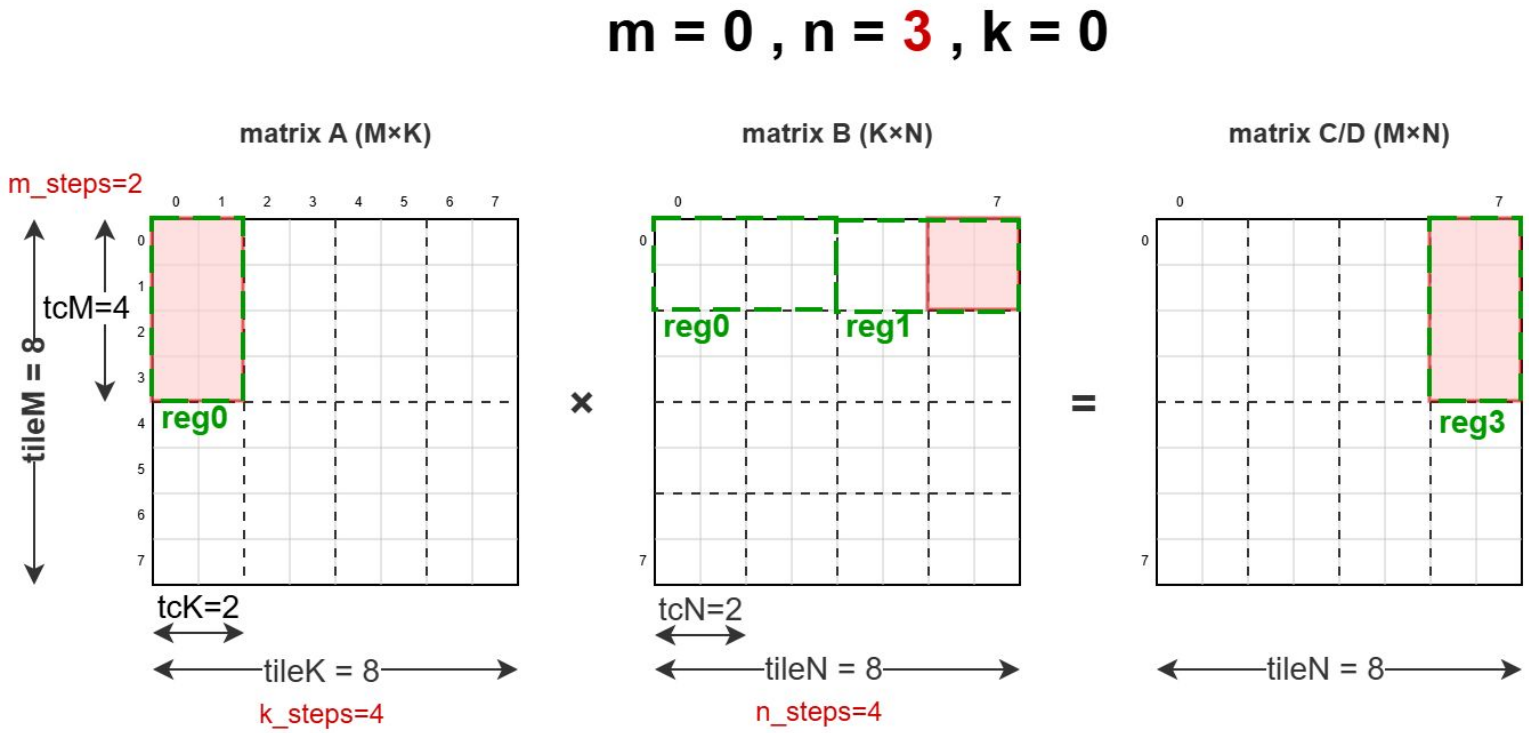
- At second micro-ops, we iterates over n_steps first.

Tensor Core in Vortex



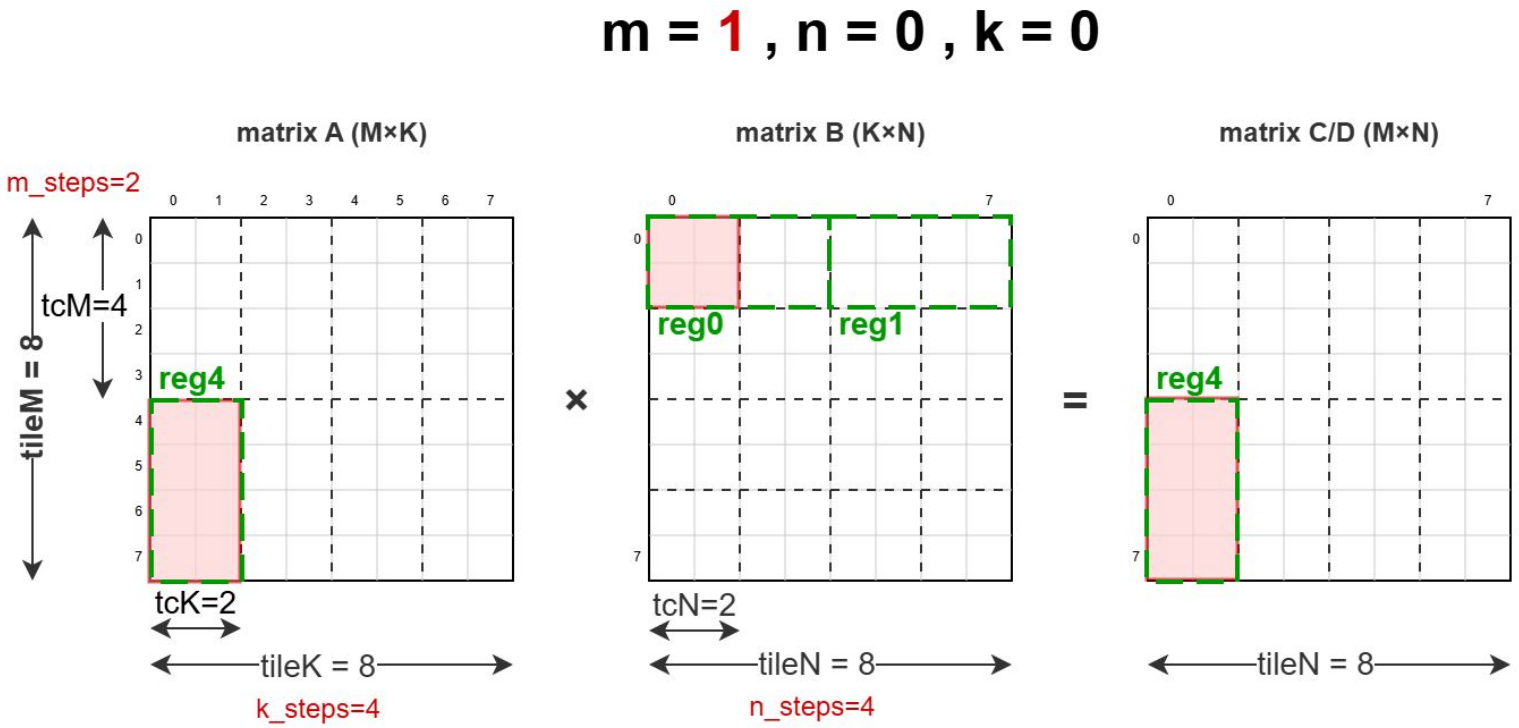
- At third micro-ops, we iterates over n_steps first.
- At every micro-op, we change the blocks by fetching from another registers.

Tensor Core in Vortex



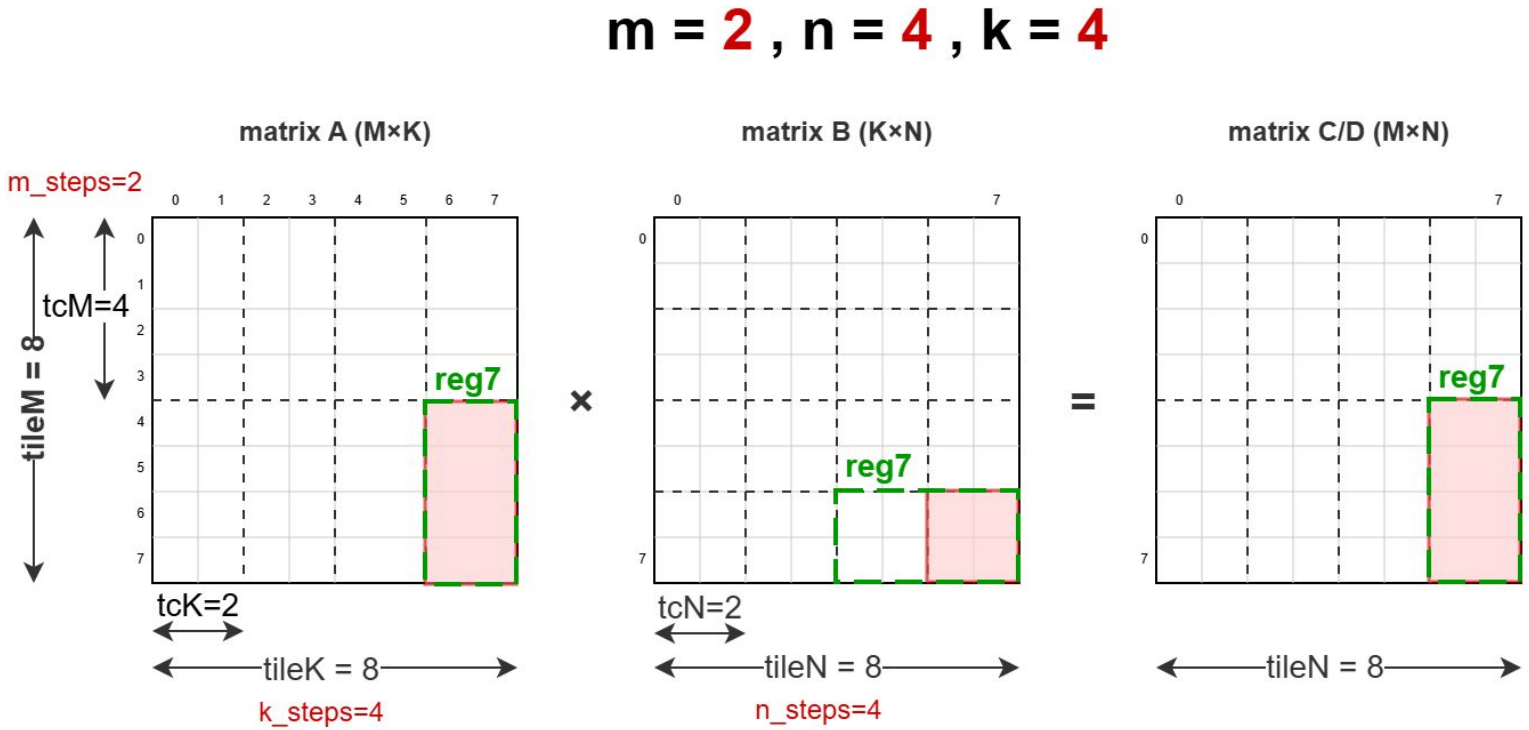
- At fourth micro-ops, we iterates over n_steps first.

Tensor Core in Vortex



- After n_steps, we go to m_step.
- So, n -> m -> k order iterations.

Tensor Core in Vortex



- Finally, we can finish the whole tile matrix matrix multiplication.
- This whole process take 32 micro-ops from one mma_sync instruction

Contents

Open Tensor-Core Platform Landscape

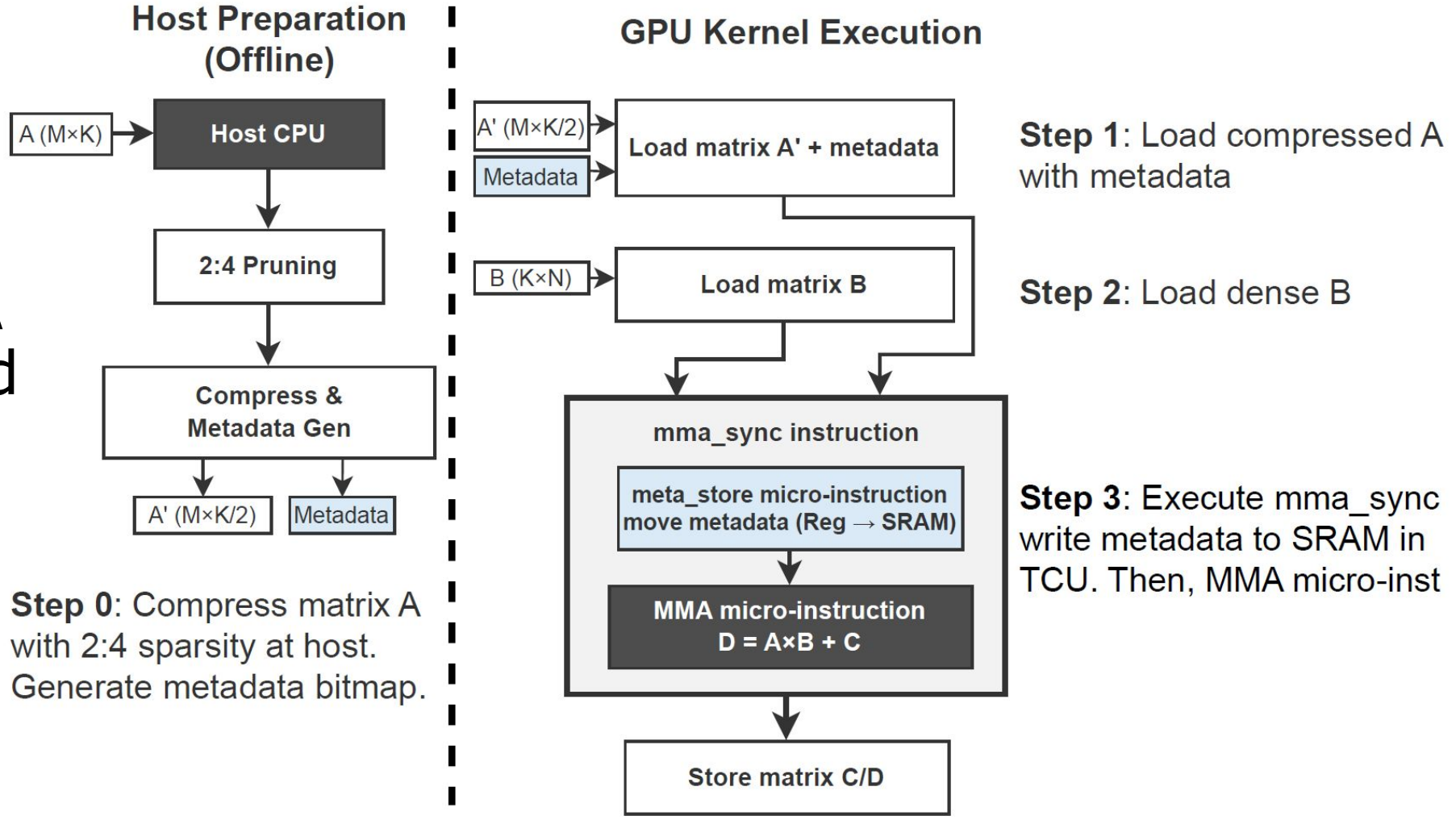
COMPARISON OF GPU TENSOR-CORE ARCHITECTURES AND SIMULATORS.

Paper	Simulation	Thread Cfg.	Sparsity	TC Style
Ours	Cycle-sim, RTL	4, 8, 16, 32	2:4 structured	SIMT
Virgo [11]	Cycle-sim, RTL	Fixed 32	✗ No	Systolic
CoopWarp [15]	Cycle-sim	4, 8, 16, 32	✗ No	SIMT
RM-STC [13]	Cycle-sim	Fixed 32	Any degree	SIMT
VEGETA [9]	Cycle-sim, RTL	No SIMT	N:M flexible	Systolic
SIMD^2 [14]	HW emulation	Fixed 32	✗ No	SIMT
AccelSim [6]	Cycle-sim	Fixed 32	✗ No	SIMT
Sparse TC [12]	Cycle-sim	Fixed 32	Vector N:M	SIMT
GPGPUSim [7]	Cycle-sim	Fixed 32	✗ No	Modeled

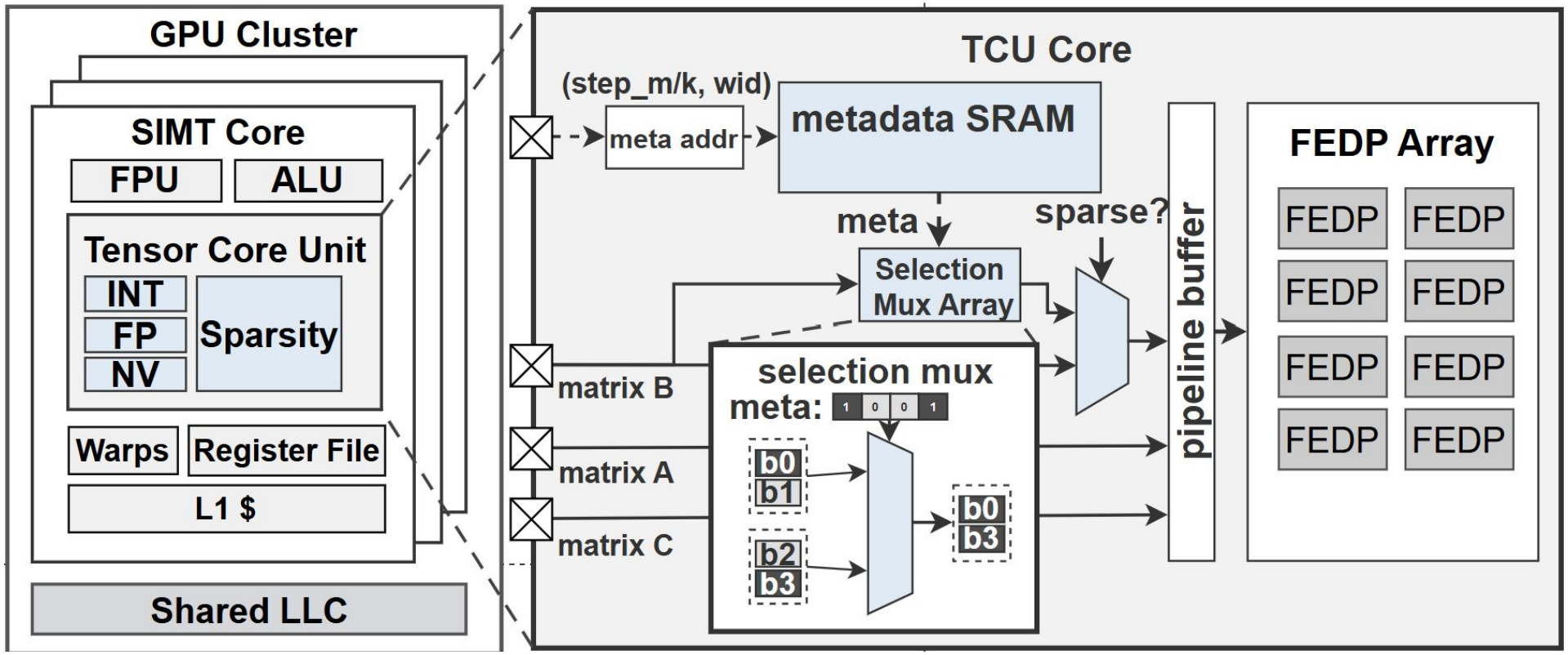
- Prior platforms provide hardware fidelity, thread scalability, or sparsity—but not all three.
- Most SIMT tensor-core designs assume a fixed 32-thread warp and dense computation.
- Our work unifies thread-scalable SIMT and 2:4 sparsity in both simulation and RTL.

2:4 Sparse Tensor-Core Execution Flow

- Offline: prune and compress matrix A, then generate 2:4 metadata.
- Load compressed A with metadata; load B and C normally.
- Use metadata to locate nonzero values without reconstructing dense A.

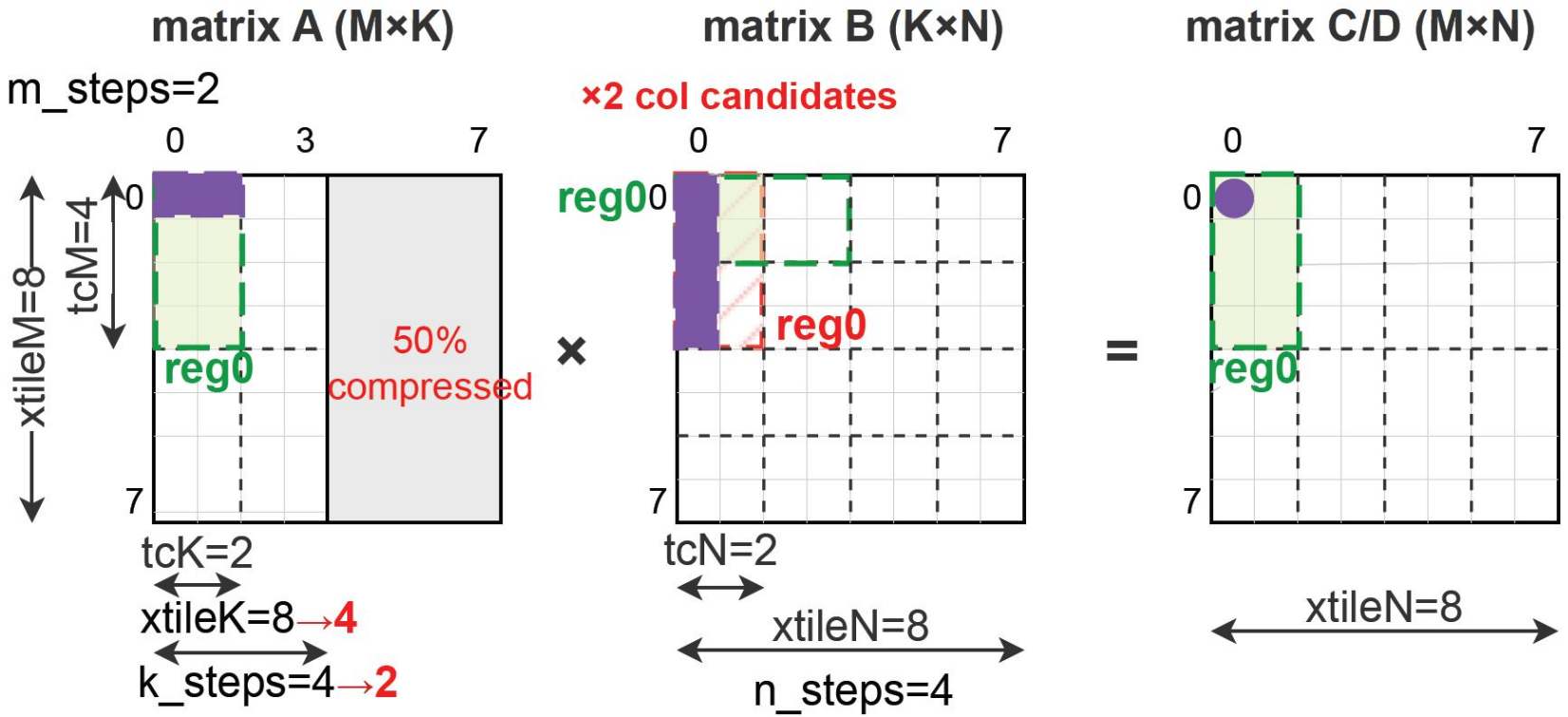


Micro-architecture



- Store 2:4 sparsity patterns in a dedicated metadata SRAM.
- Use metadata to select the matching elements from matrix BBB.
- Align and buffer the selected operands before FEDP execution.
- Reuse the same compute array for both sparse and dense modes.

Benefits of 2:4 Sparse Execution

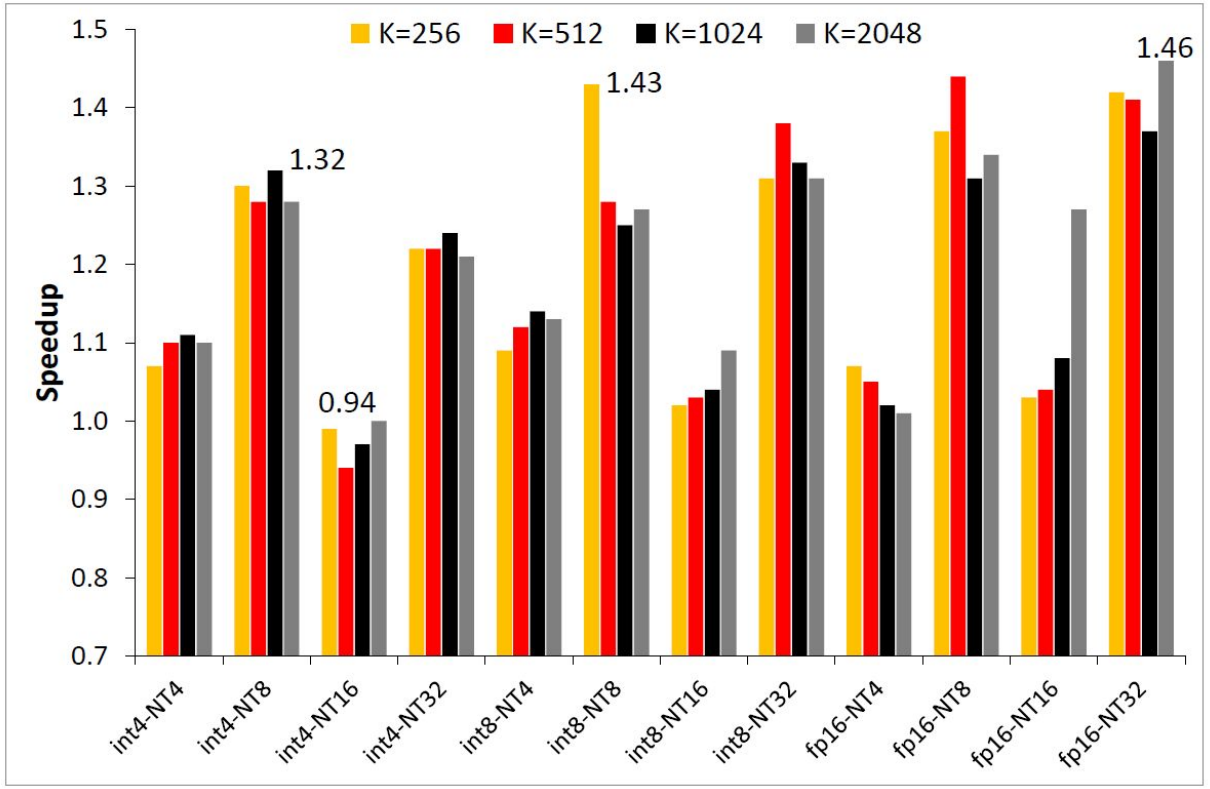


#Thread = 8
Green : dense
Red : sparse
Purple : one thread compute for sparse

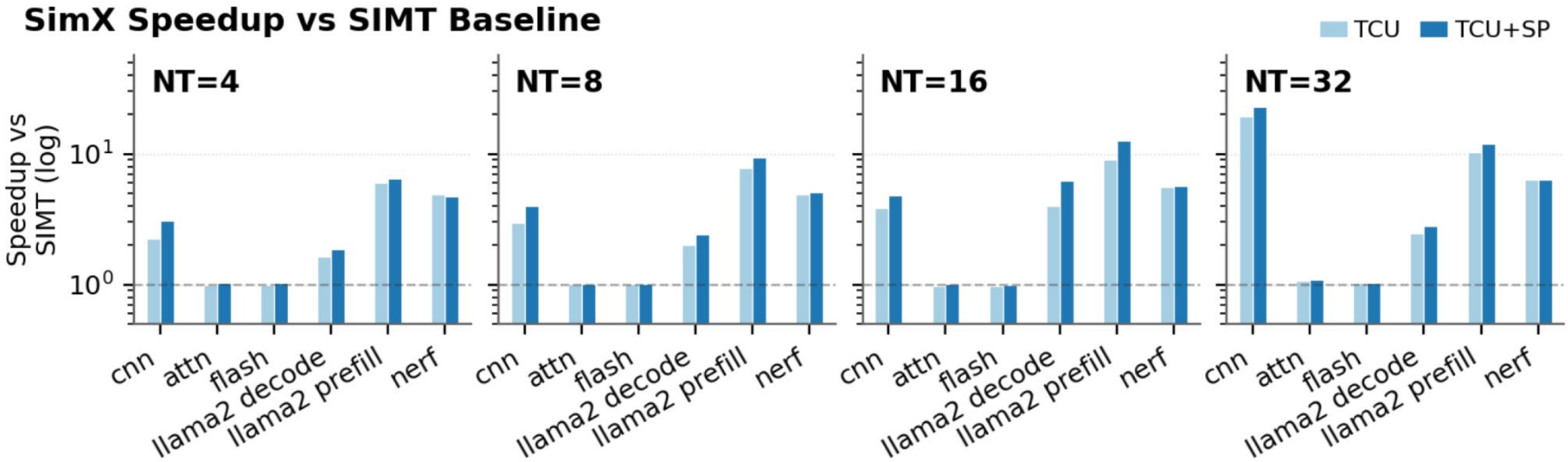
- Memory benefit: load only 50% of matrix (A).
- Compute benefit: (xTileK) is reduced from 8 to 4.
- Therefore, (k_steps) decreases from 4 to 2.
- Since total micro-ops are proportional to (m_steps x n_steps x k_steps), the total micro-op count is halved.

End-to-End Sparse TCU Speedup

- Sweep #Thread, datatype, and K dimension to evaluate full-kernel sparse GEMM compared to dense.
- Sparse speedup includes load, WMMA, and store execution.
- #Thread=8 and #Thread=32 show the strongest scaling across configurations.
- The sparse TCU achieves up to 1.46x end-to-end speedup, demonstrating that reduced matrix A traffic and fewer K-step micro-operations.



End-to-End DNN Speedup over SIMT



- Sparse TCU gives the largest gains on memory- and GEMM-heavy workloads compared to dense
- At #Thread=8, it reaches 5.9x on Llama2 decode and 11.71x on prefill, compared with 2.98x and 6.11x for the dense TCU.
- Attention workloads see little gain because Softmax remains SIMT-bound. (un-optimized softmax kernel)

Summary & Questions

- Configurable 2:4 sparse TCU controlled by a single compile-time thread parameter.
- Validated across cycle-accurate simulation, synthesizable RTL, and FPGA deployment at 300 MHz.
- Structured-sparsity support adds only (+0.07%) overhead.
- RTL and SimX extensions are released through a public Vortex fork.