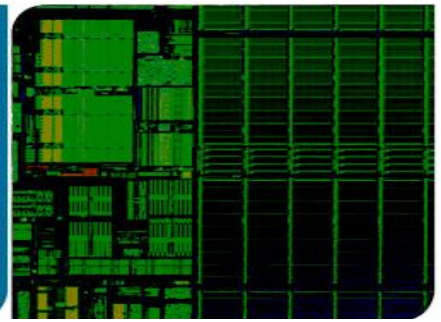
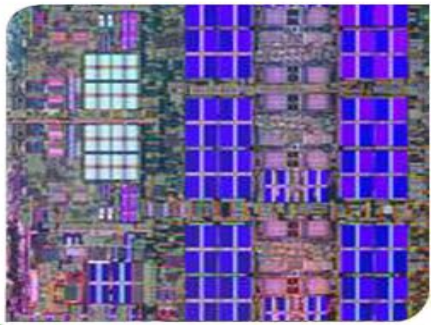
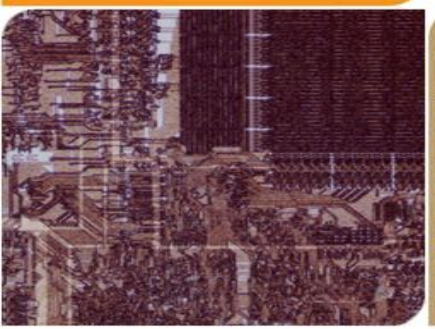




DNN Inference on RISC-V GPGPU

Varsha Singhania, Nayan Nair, Blaise Tine, Hyesoon Kim
Georgia Tech

Georgia Tech comparch



Outline

- Motivation
- Vortex GPU Platform
- Mapping software to hardware
- Tensor Core Exploration
- Comparison and Performance
- Future Work

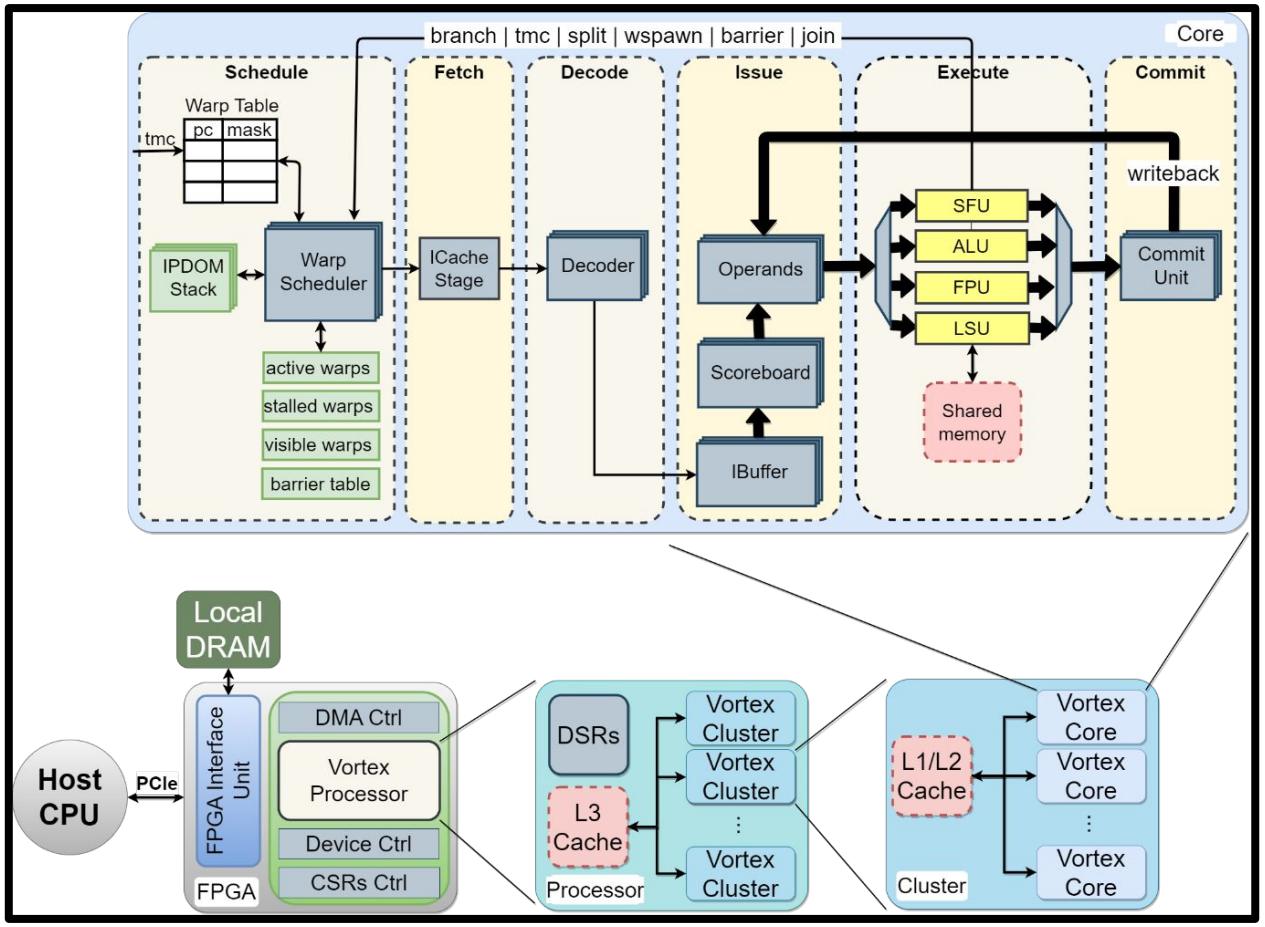
Motivation

- Vortex : Open Source RISC-V GPU ; no tensor cores
- Tensor Core integrated GPU architectures - proprietary
- Accelerate matrix multiplication on Vortex
- Full stack - platform for accelerator integration exploration

- Contributions:
  - ISA extension
  - DSE for tensor core integration
  - Software task mapping to threads for parallelization
  - Implementation on Vortex SimX (a cycle level simulator)

About Vortex (1)

- Full stack open-source RISC-V GPU
- ISA extension for GPGPU -
  - Thread scheduling
    - wspawn %waves, %PC
    - tmc %threads
  - Flow control
    - split %pred
    - join
  - Synchronisation
    - bar %bar, %waves
- Highly configurable number of -
  - Cores, warps, threads
  - ALU, FPU, LSU, SFU units per core
  - # tensor cores **(our addition)**
  - Size of tensor cores **(our addition)**

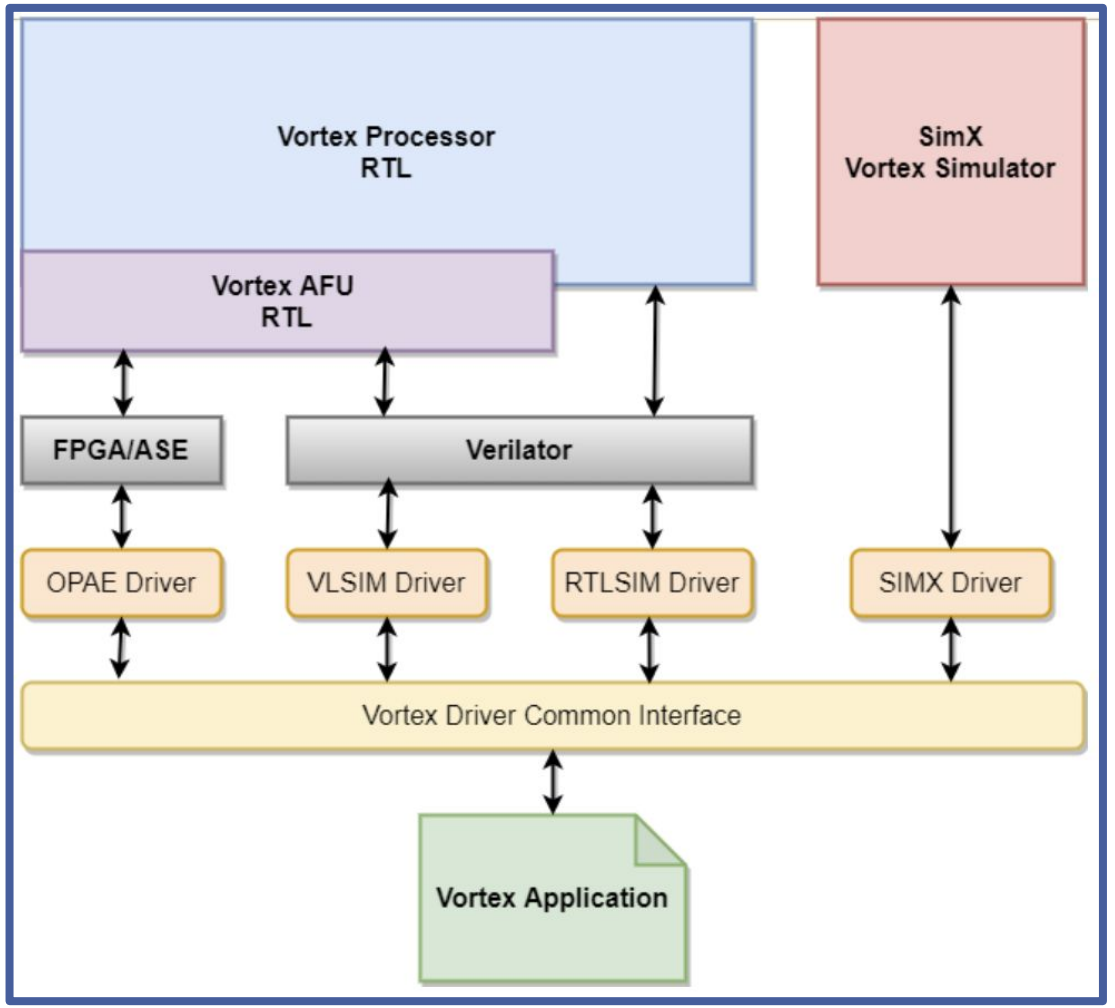


About Vortex (2)

- Key Features
  - PCIe-based Host communication
  - High-bandwidth Cache sub-system
  - Multi-channel memory system
  - Pipeline elasticity

- Cross platform simulation
  - FPGA
  - RTL
  - Cycle-Level Simulation

- Hardware Extensions
  - 3D Graphics
  - Graphics Analytics
  - Ray Tracing
  - Custom Extension



Mapping S/W to H/W (1)

- 1 warp works on 1 output tile at a time
- Task = Unit of work that gets assigned to a thread
- Total #tasks for a MatMul = #Output Tiles x (#Threads/Tensor Cores per Warp)

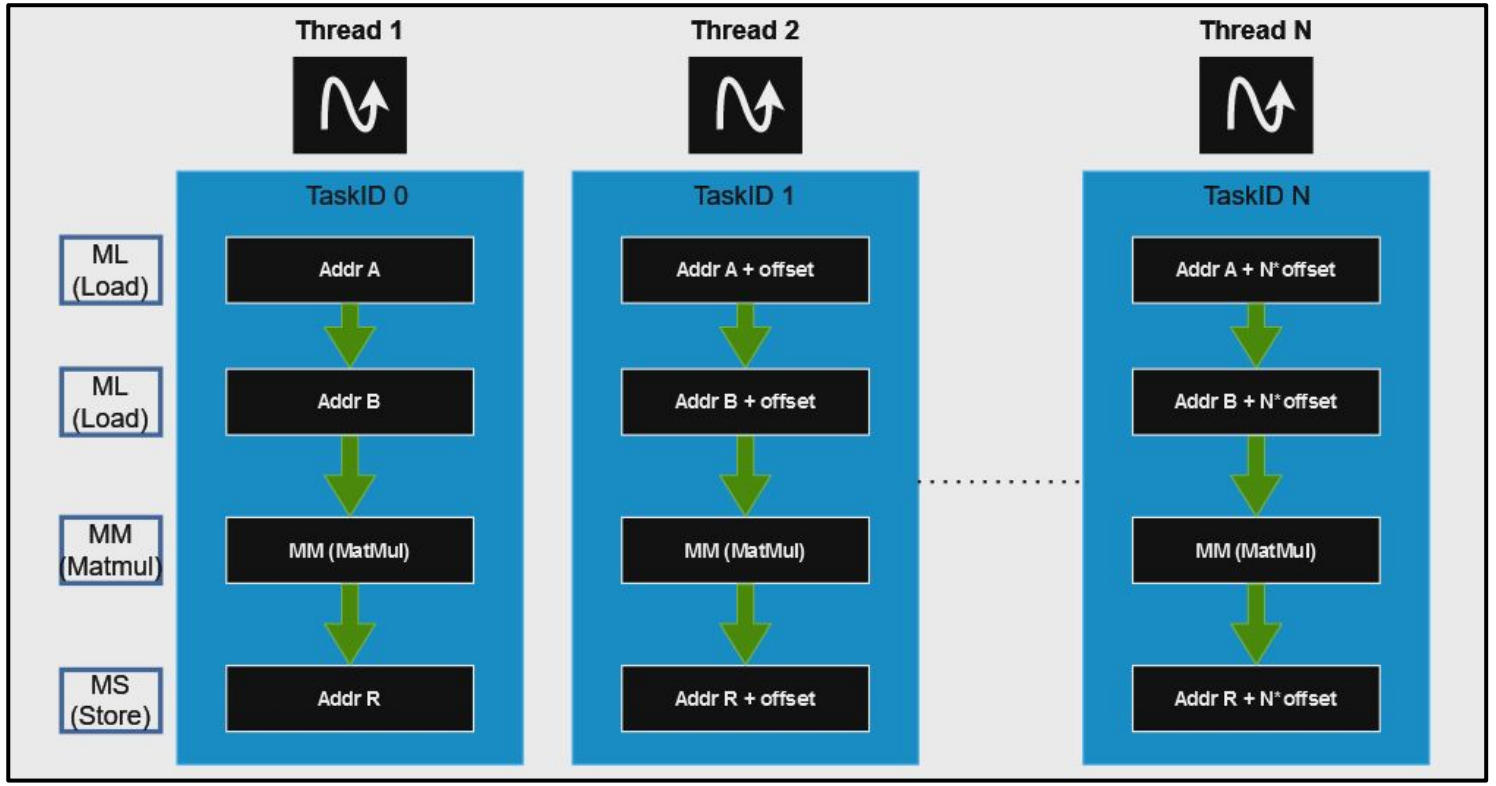
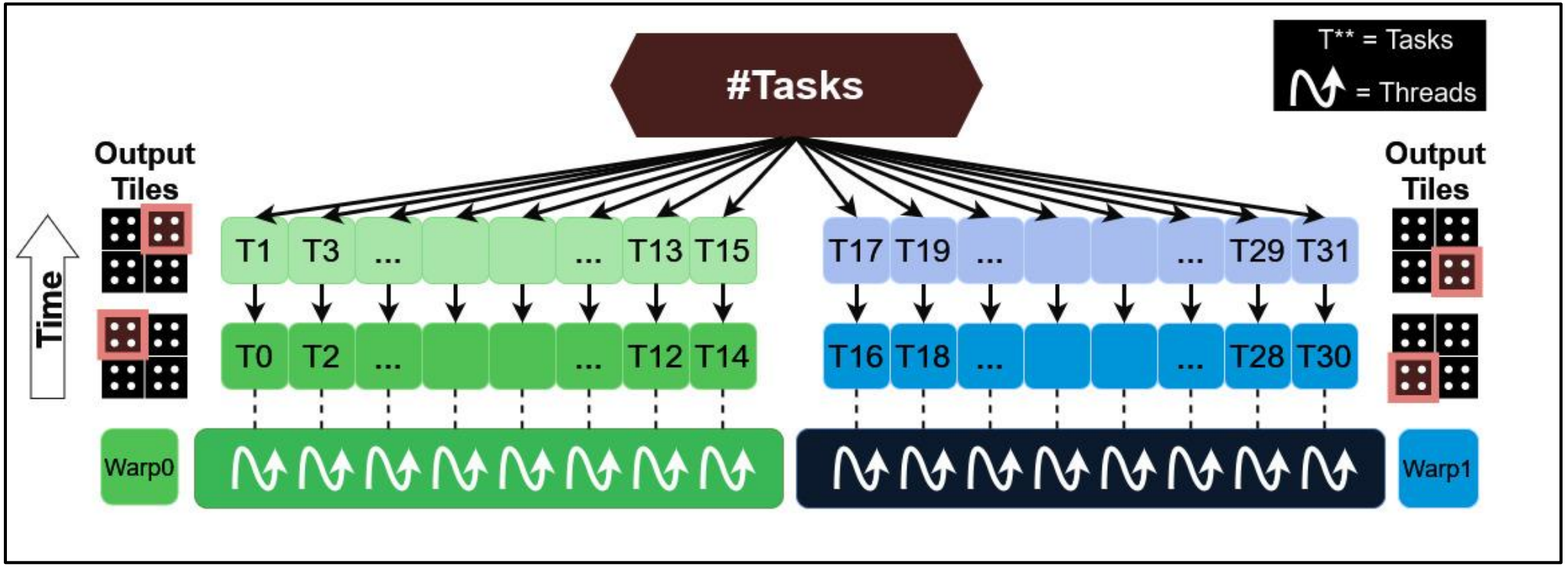


Figure:
Parallel task execution by threads

Mapping S/W to H/W (2)

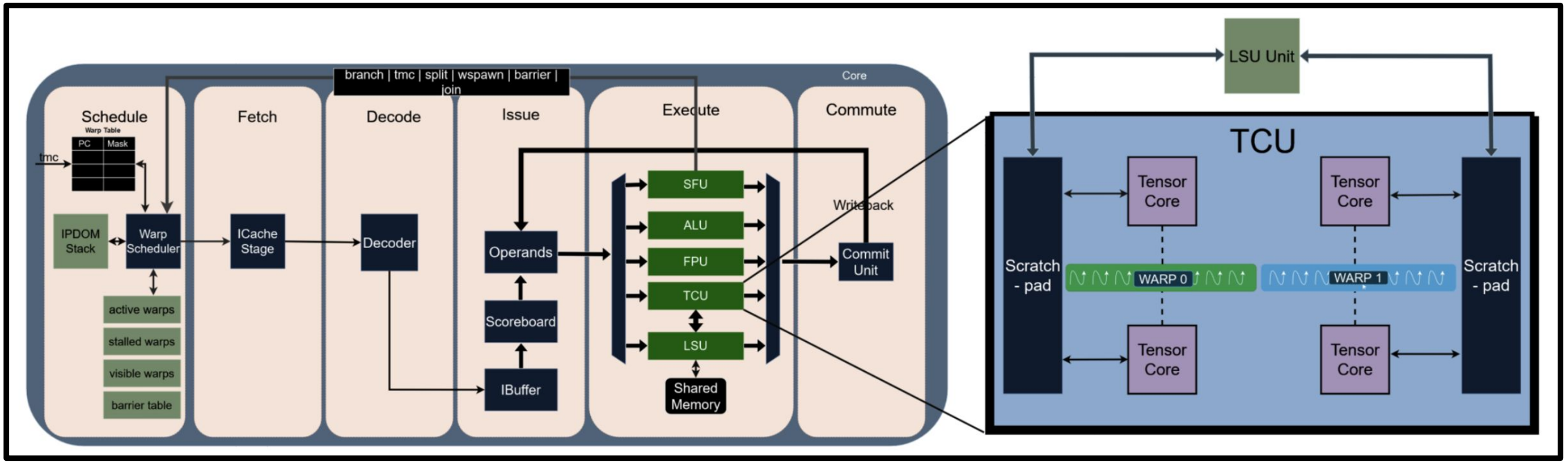
- Tasks are distributed first over all warps, then over threads within a warp.
- Threads will work on tasks sequentially in time. (For Ex. 2 tasks per thread in fig.)



Task Distribution for 8 threads, 2 warps

Warp Level Integration of TC

- Interface with the LSU unit
- Interface with the Vortex pipeline (decode, execute …)
- Scratchpad integration



ISA Extension

- Divided into three parts – Matrix Load, Matrix Multiply and Matrix Store.
- Implemented using three I-type instructions

imm[11:0]	rs1	funct3	rd	opcode	I-type

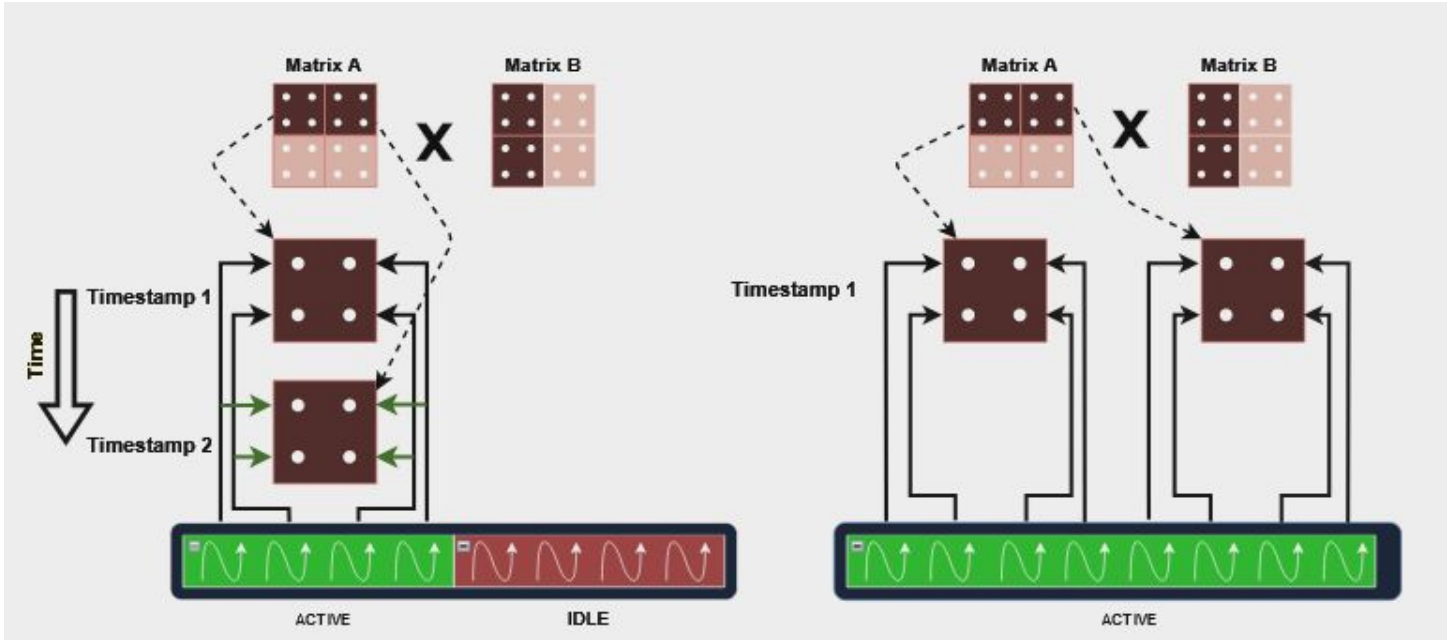
- Spare Opcode – 0x7B
- Funct3 fields used to distinguish between ML, MS, MM

	Inst	Opcode	Funct3	Imm	rs1	rs2
Load	ml	0x7B	0	0/1	src base addr	X
Mult	mm	0x7B	2	X	X	X
Store	ms	0x7B	1	X	dest base addr	X

Thread Utilization Schemes

- Work distribution among threads inside a warp
- For 1 output tile computation, multiple input tiles are required

#elements fetched by each thread = # elements required per output tile / # threads per tensor core



Option 1 : Threads Underutilization (smaller scratchpad requirement)

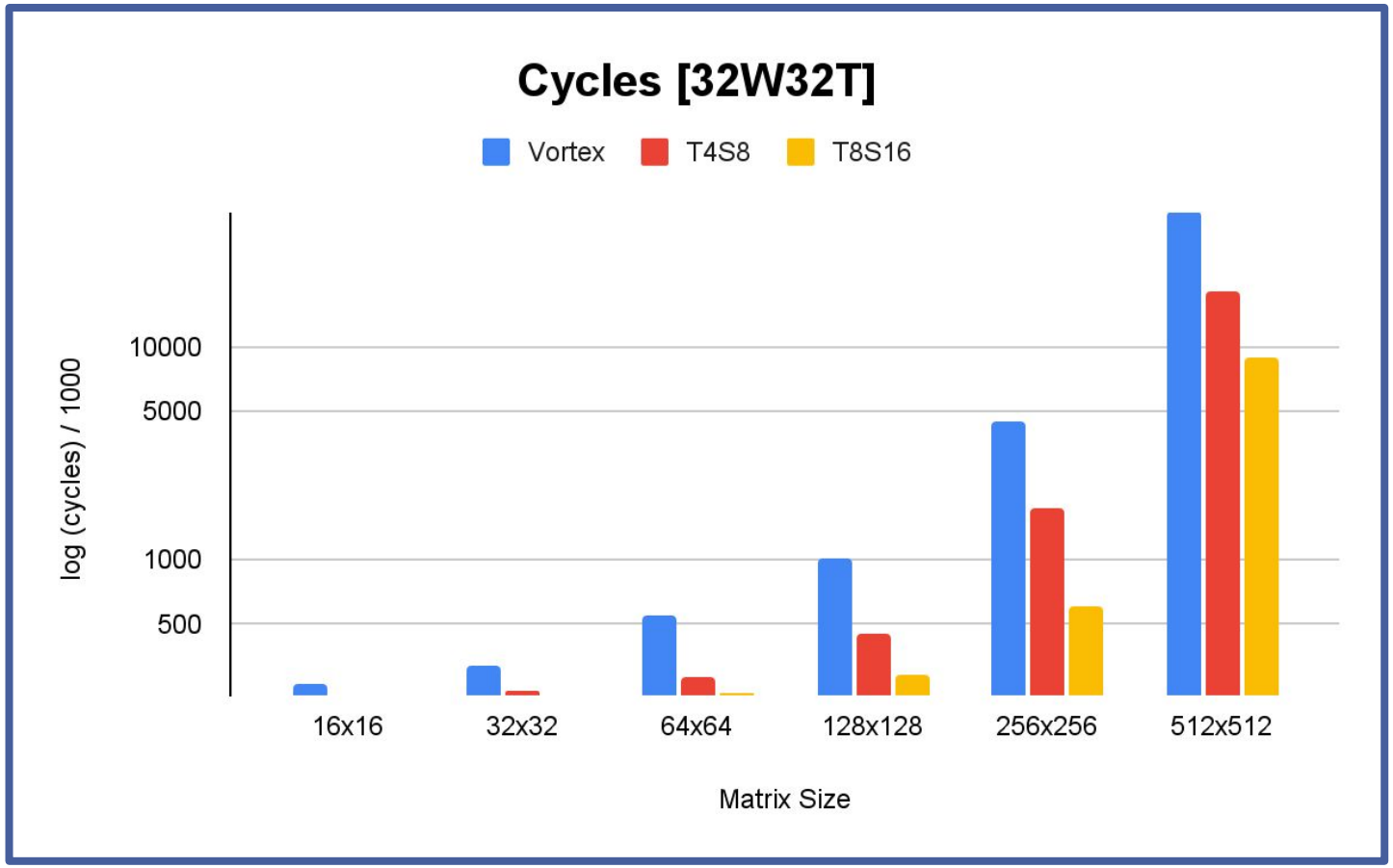
Option 2 : Efficient Thread Utilization



Further Design Exploration

- Data reuse
  - Order of output tile being processed
  - Tile assignment to a warp
  - Reuse across warps
- Number of Instructions
- Accelerator Dataflows
  - Weight Stationary, Input Stationary
- Tradeoff evaluation of Scratchpad size
- Granularity of TC integration

Vortex With and Without TC

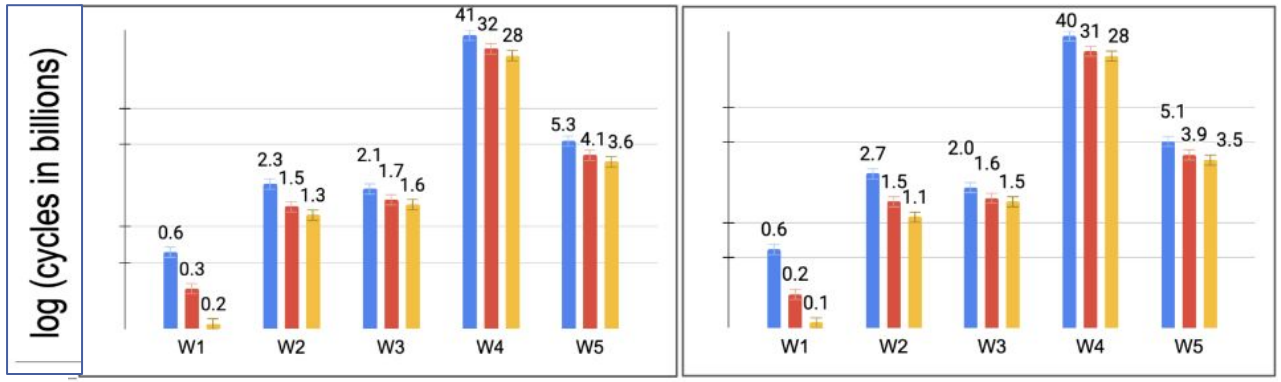


- Without TC, matrix multiplication by ALU/FPU Units
- Increasing hardware resources per warp for tensor cores further decreases cycles.

Running Workloads

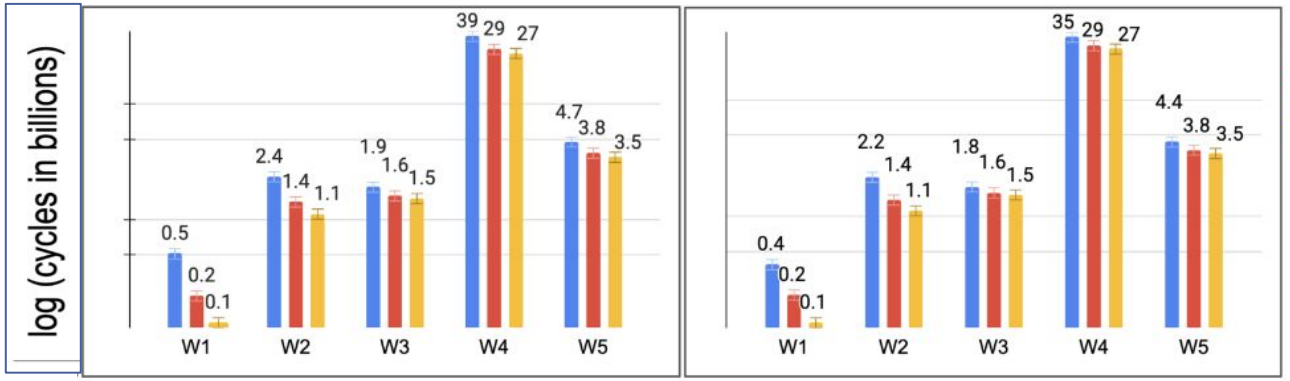
- Layer specificifications from **ScaleSim***
- **TC Sizes** : 8 = Blue ;
  16 = Red ; 32 = Yellow

Tag	Workload
W1	Bert
W2	AlexNet
W3	GoogleNet
W4	Transformer
W5	ResNet-50



TC_NUM = 4

TC_NUM = 8



TC_NUM = 16

TC_NUM = 32

* A. Samajdar, J. M. Joseph, Y. Zhu, P. Whatmough, M. Mattina and T. Krishna, "A Systematic Methodology for Characterizing Scalability of DNN Accelerators using SCALE-Sim," 2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Boston, MA, USA, 2020

Layer Evaluation Example

1st layer of Alexnet :

- **Step 1** : Im2col

Layer	IfMap H	IfMap W	Filter H	Filter W	Channel	Num Filters	Stride
Conv1	227	227	11	11	3	64	4

Conv Layer

Ax	Ay	Bx	By
363	64	3025	363

GEMM Layer

- **Step 2** : Tiling and/or padding to form standard matrix sizes (16, 32, 64, 128, 256..)

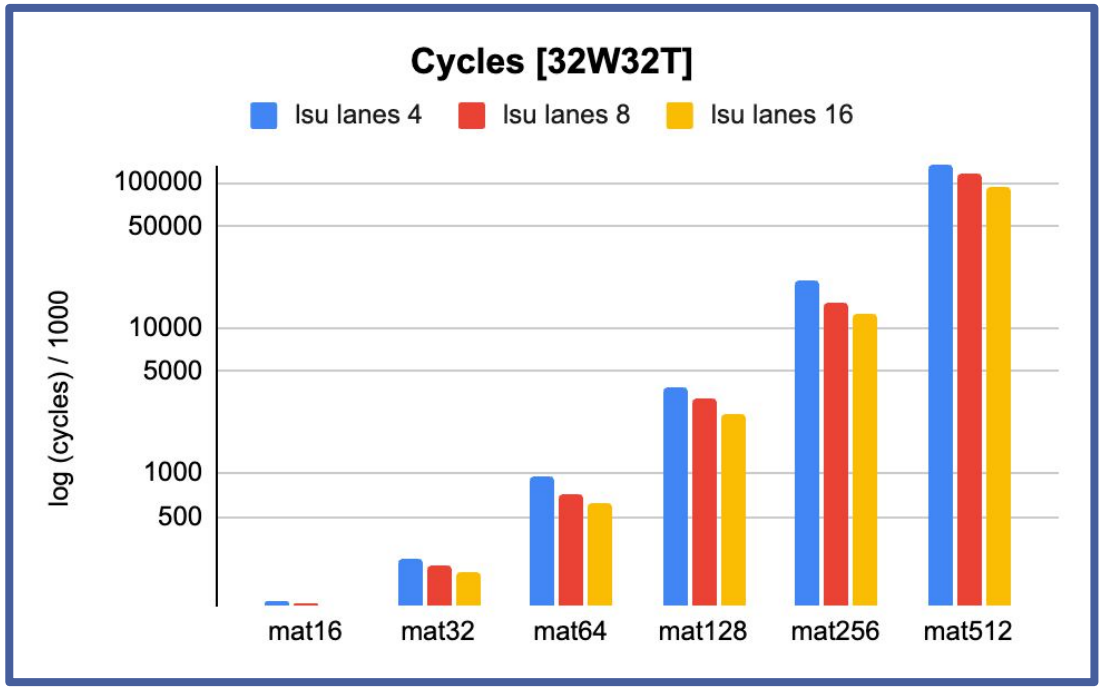
Ax	Ay	Bx	By
512	64	4096	512

- **Step 3** : Determine # kernel launches and size of matrix multiplication on each

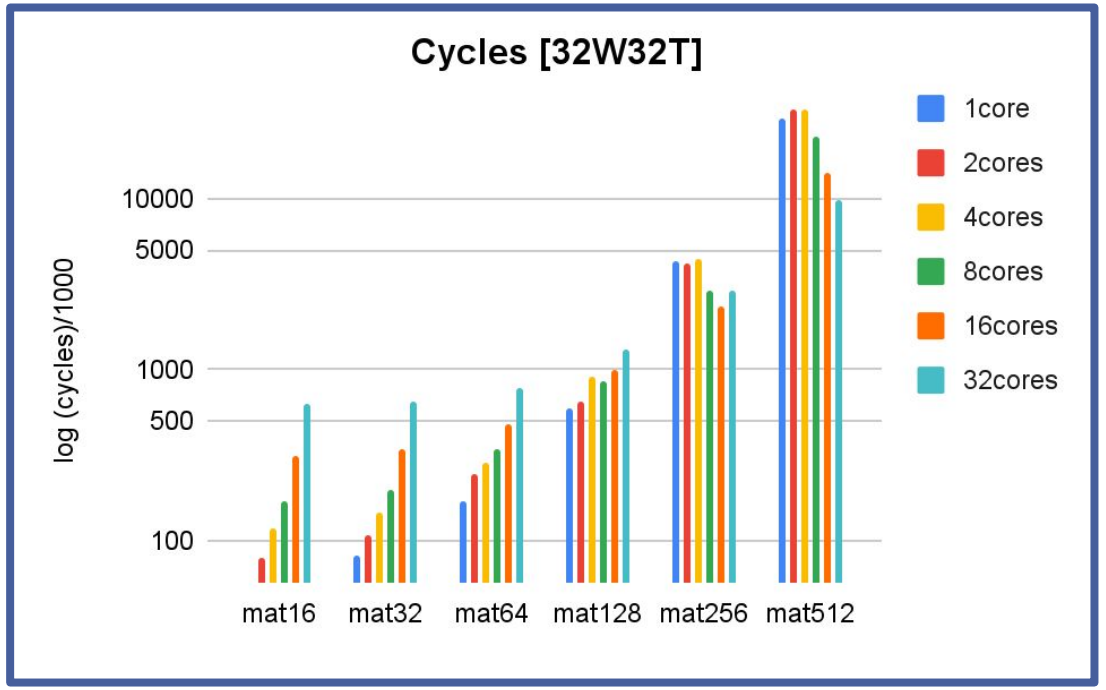
Min Dimension	Ax tiles	Ay tiles	Bx tiles	By tiles	#kernel launch
64	8	1	64	8	512

Results[1] : Scaling Vortex HW

Number of LSU Lanes



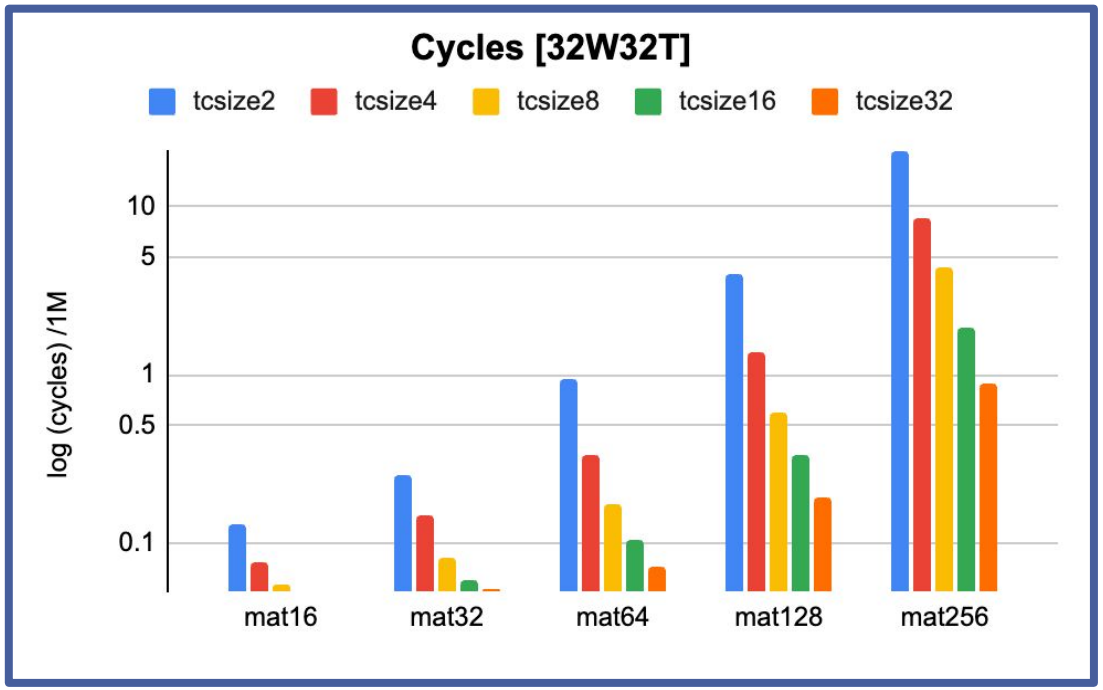
Number of Vortex Cores

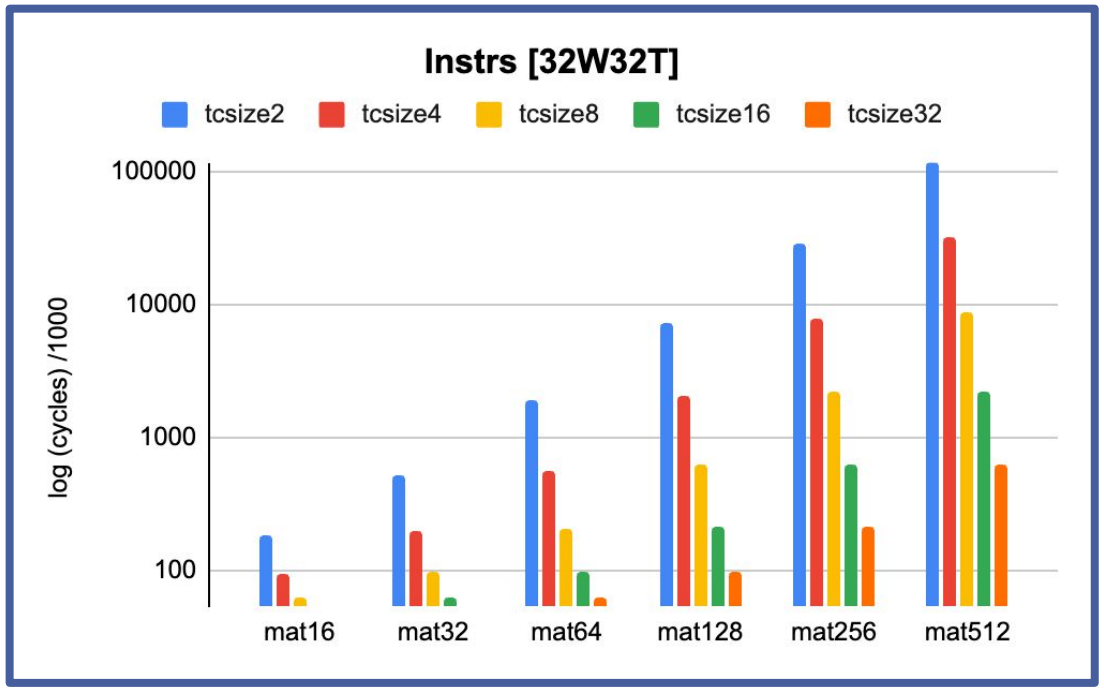


- #LSU Lanes⬆️ : Fewer memory request cycles
- [smaller matrices] #Cores⬆️→ #cycles⬆️ : Communication overhead > Compute efficiency

Results[2] : Scaling TC Size

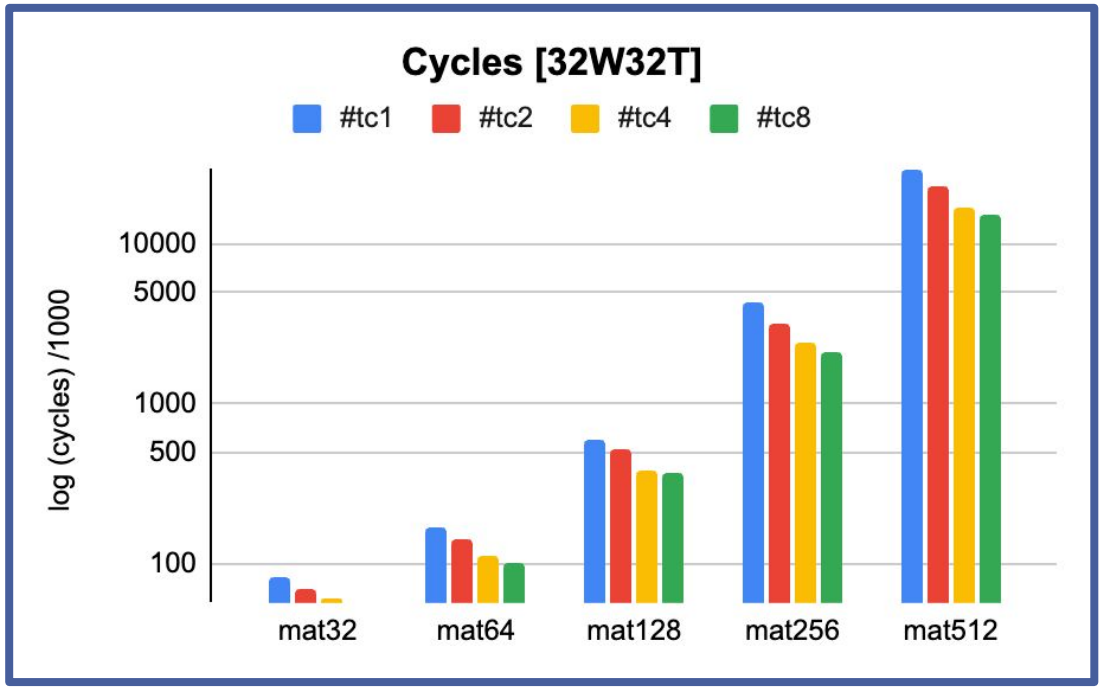
- For bigger TCs → need fewer output tiles → fewer instructions
- More data fetched per instr → #instr⬇️→ pipeline overhead⬇️→ #cycles⬇️

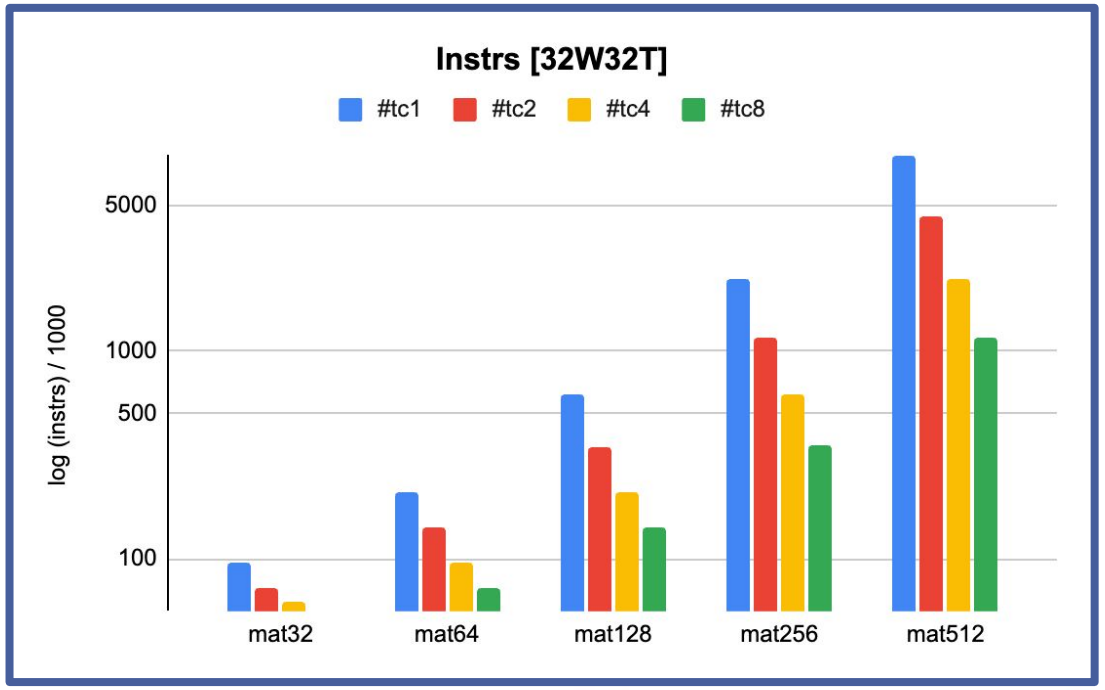




Results[2] : Scaling Number of TCs

- TCs per warp
- Per thread is a special case of per warp granularity
- 1 instr for multiple TCs → thus as #TCs⬆️ , #instrs⬇️ (more for bigger matrices)
- amount of data fetched per instr increases → more cycles per instr
- Overall, total cycles **reduces** as #instr have reduced much more





Running Tensor Core Tests

- **Step 1:** Clone Vortex repo : https://github.com/vortexgpgpu/vortex
- **Step 2:** Pull the **"tensor-core"** branch
- **Step 3:** Install required toolchains and build Vortex (Steps : README)
- **Step 4a:** Run **matmul_regression.sh** inside vortex/tests/regressions/matmul to run a sweep of #Tensor Cores, Tensor Core Sizes and Matrix Sizes
- **Step 4b:** To run for a single configuration, run the command below from vortex/build dir

  tcnum → # of TCs, tc_size → Size of TC, matsize → matrix size

```
CONFIGS="-DTC_NUM=tcnum -DTC_SIZE=tc_size" ./ci/blackbox.sh --cores=4 --app=matmul --driver=simx
--threads=32 --warps=32 --args="-nmatsize -d1" --rebuild=1 --perf=1
```

Kernel Programming

```
kernel_body:
    taskID = blockIdx.x
    addr_offset, res_addr_offset  = f( taskID )
    matrix_a_addr_offset = f( a_base_addr , addr_offset )
    matrix_b_addr_offset = f( b_base_addr , addr_offset )
    result_matrix_addr_offset = f( res_base_addr , res_addr_offset )
    vx_matrix_load( 0 , matrix_a_addr_offset )
    vx_matrix_load( 1 , matrix_b_addr_offset )
    vx_fence()
    vx_matrix_mul()
    vx_fence()
    vx_matrix_store( result_matrix_addr_offset )
    vx_fence()
```

Future Work

- Explore other possibilities in the GPU+accelerator design space.
- Explore different data movement strategies based on:
  - Tensor core scratchpad space
  - Shared memory utilization
- Explore warp collusion through shared memory to allow sharing of input data across multiple output tiles.
- Warp specialization
- Include compiler support to translate matrix multiplications to MatMUL instructions.
- Try out support for more involved tensor cores.
- Integrate other accelerators with GPU and implement a general GPU-accelerator interface



Thank You!

Processing on Host v/s Device

- Vortex has 8GB global memory.

- Demand matrix → Matrix Format suitable for streaming into a Systolic Array
  - For 1024x1024 matrix multiplication, with 2x2 tile sizes, 4B elements
  - Size of A, B, C = 4GB each.

- For larger matrices (CPU side) -
  - Use larger sizes of tensor cores.
  - Use fewer bits for elements of matrices to be multiplied.

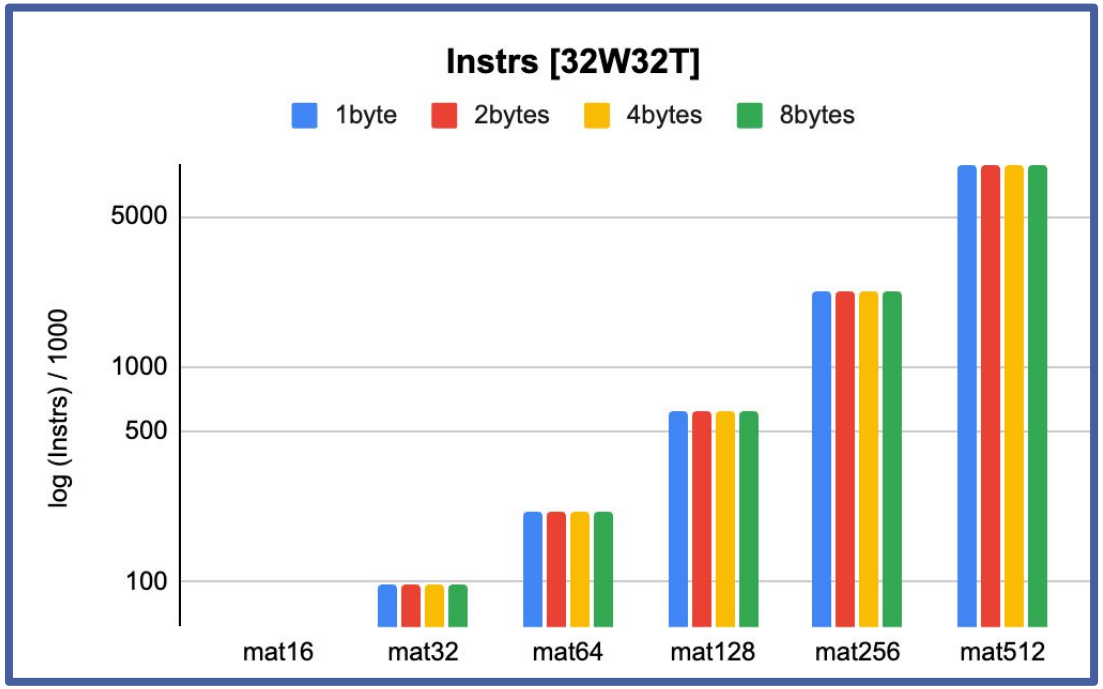
Global Memory Req = 2 × demand matrix size × #output tiles × data size

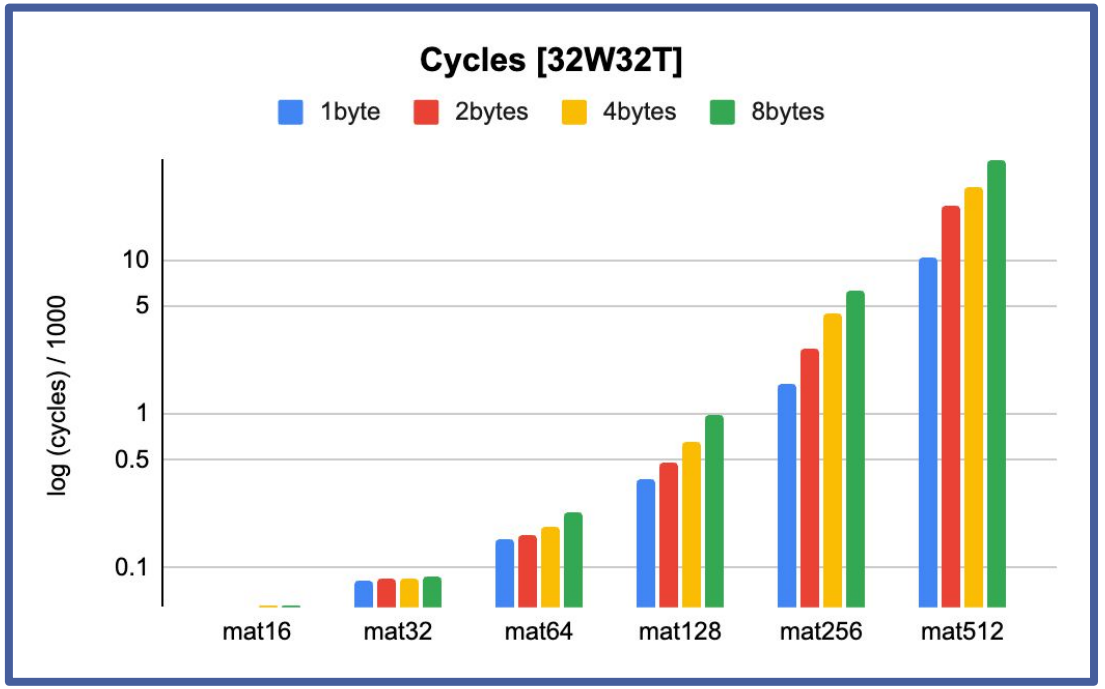
- Demand matrix creation on GPU allows –
  - Larger matrix computation
  - Cost – extra cycles on the device, lower IPC

Design Space Exploration (Results)

Data Types

- #instr is same for different data sizes
- #cycles⬆️ as data size⬆️ → more data to be fetched per instr.





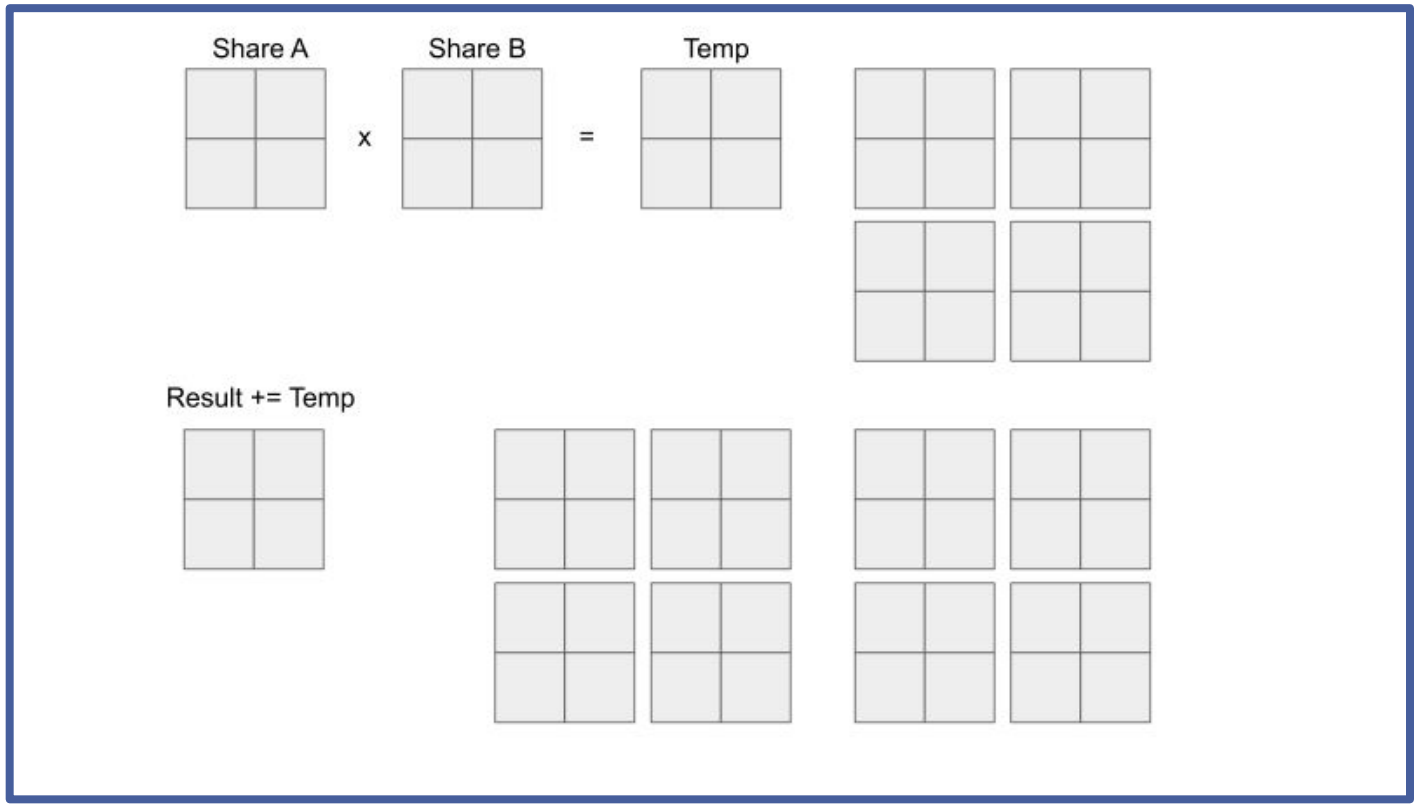
Related Work

- Tensor Core implementation is inspired by TCs of Google and Nvidia and instructions of Intel, IBM and ARM
  - Intel AMX instructions → TMUL (Tiled Matmul Unit)
  - IBM Power10 MMA
  - Armv9-A architecture with Scalable Matrix Extension (SME)

- Scale-SIM, Accel-Sim, Mgpusim
- *Implementing Hardware Extensions for Multicore RISC-V GPUs*
- Enabling Domain-Specific Architectures with an Open-Source Soft-Core GPGPU

Tiled Matrix Multiplication

- More performant to perform tiled matrix multiplication for big matrices in a SIMT processor
- Fewer number of global memory accesses
- Data reuse



https://penny-xu.github.io/blog/tiled-matrix-multiplication

Performance Model

- Tensor Core (multiply)
  - Output Stationary Dataflow
  - Latency of 1 MATMUL op = Systolic Array Load + Streaming + Drain Time

- Memory Requests (load and store)
  - Number of Requests Sent via LSU per thread
  - #Requests depends on data size

Conclusions

- Tensor Core improves MatMul Performance on Vortex
  - Thus improving neural network workload performance.

- Design Space Exploration demonstrates flexibility of Vortex to integrate with an accelerator

- Vast DSE space for optimizing GPU+accelerator system.
  - Many more HW and SW parameters to explore.