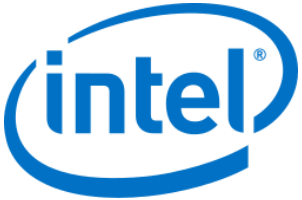
Toward End-to-End ML Support on Vortex

Shinnung Jeong, Chihyo Ahn, Sai Hemanth Reddy Bheemreddy,
Liam Cooper, Krishil Gandhi, Chulhyung Park, Vincent Pham,
Huanzhi Pu, Saurabh Singh, Mitul Tandon, Jisheng Zhao,
and Hyesoon Kim

Georgia Institute of Technology

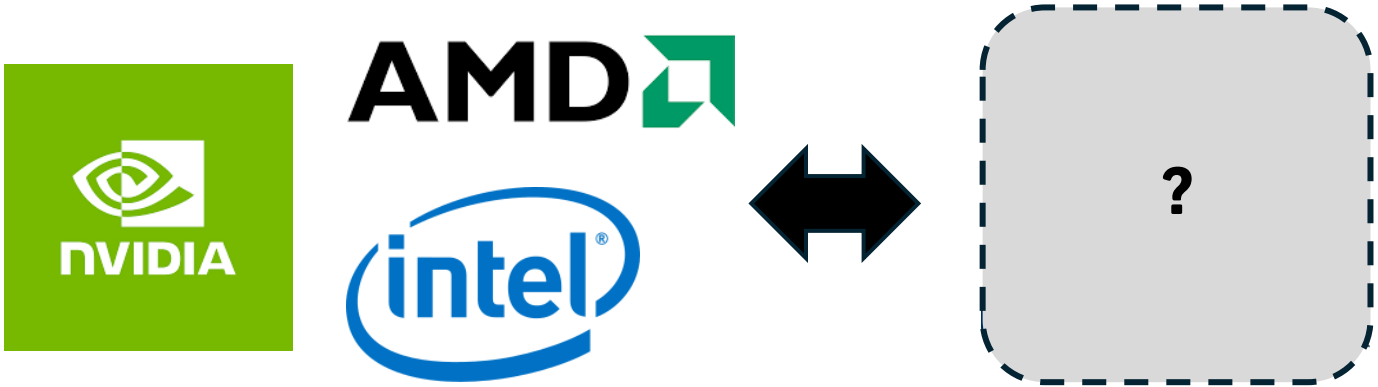
Georgia Tech

GPUs Power Modern Computing - Openness Remains Limited



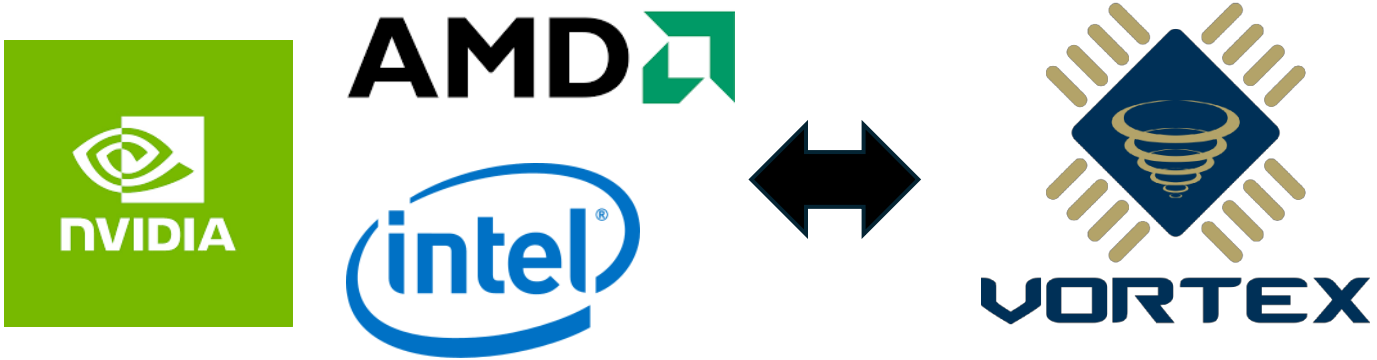
Core GPU microarchitecture and compiler internals remain largely inaccessible to researchers

What Research Needs Beyond Exposed APIs



GPU research would benefit from openness beyond exposed APIs

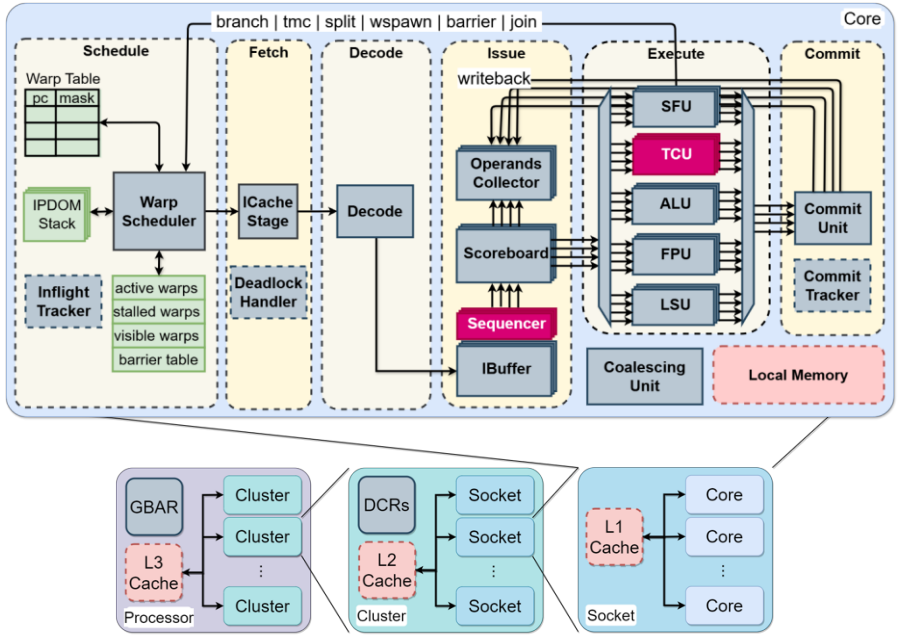
What Research Needs Beyond Exposed APIs



Vortex GPU provides a fully open-source GPU platform

Vortex GPU

- RISC-V based GPGPU
- Highly reconfigurable hierarchical architecture
- SIMT execution model
- Explicit hardware support for SIMT behavior, including control-flow divergence and barriers
- Drivers for host-device communication
- **In-core Tensor Core (TCU)[1]**

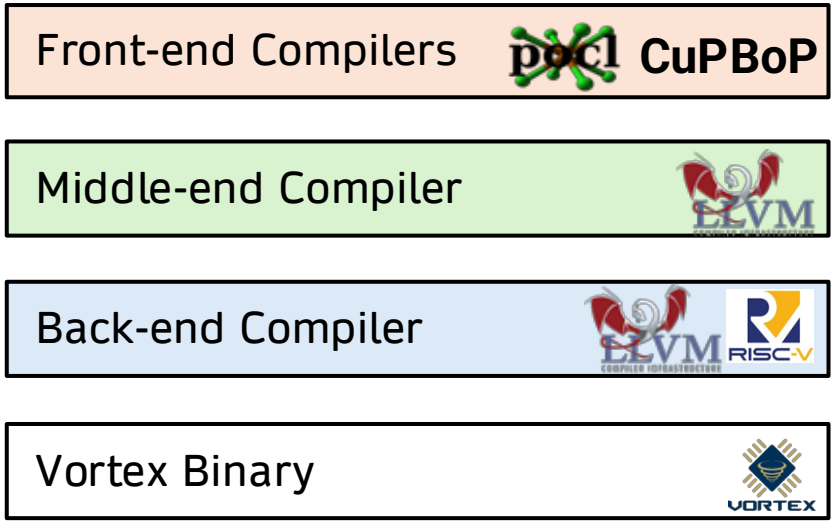


[1] N. Rout and B. Tine, “Ten-four: An open-source fused dot product unit for mixed-precision gpgpu tensor cores,” 2026, https://arxiv.org/abs/2512.00053

VOLT Compiler [CC'26]

- Compiler stack for Vortex GPU
  - Front-end: **OpenCL / CUDA**
  - Middle-end: LLVM
  - Backend: LLVM-RISCV extension

- Centralize SIMT-aware optimizations in the middle-end
  - Uniformity analysis
  - Thread divergence management

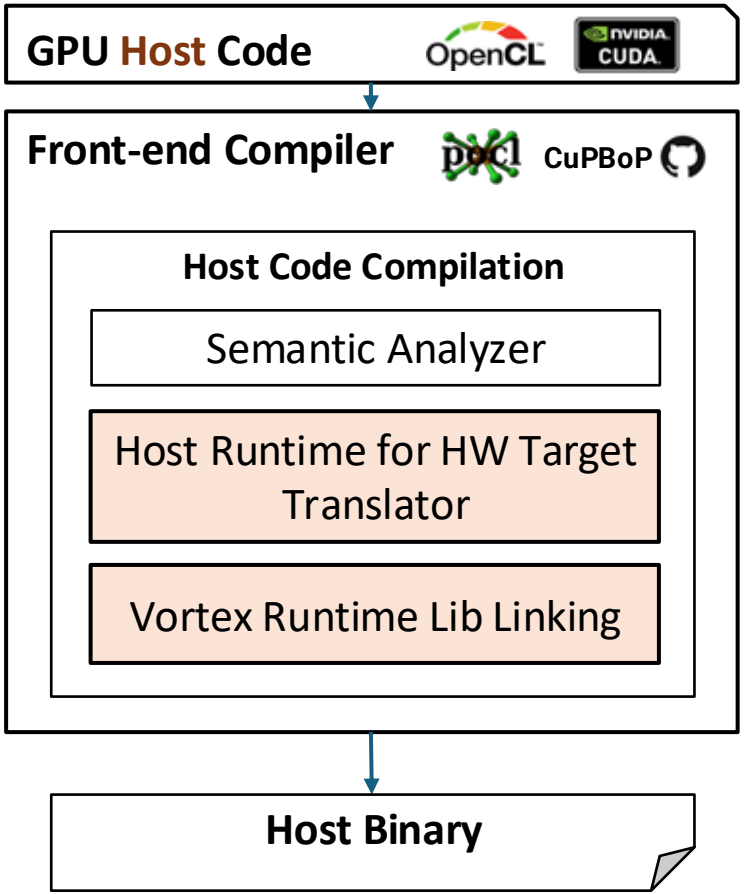


[2] Shinnung Jeong, Chihyo Ahn, Huanzhi Pu, Jisheng Zhao, Hyesoon Kim, and Blaise Tine. 2026. Inside VOLT: Designing an Open-Source GPU Compiler. CC '26

Front-end Compiler

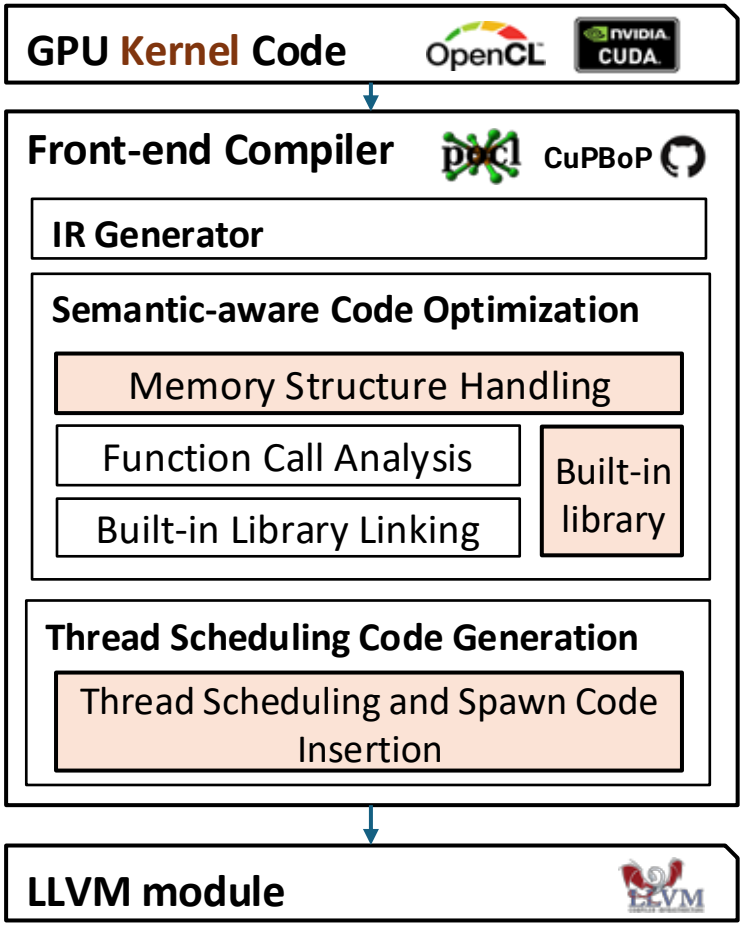
- Handles Both Host and Kernel Code

- **Host Code**
  - Compiled for the host hardware
  - Translates the frontend host functions using the Vortex runtime library



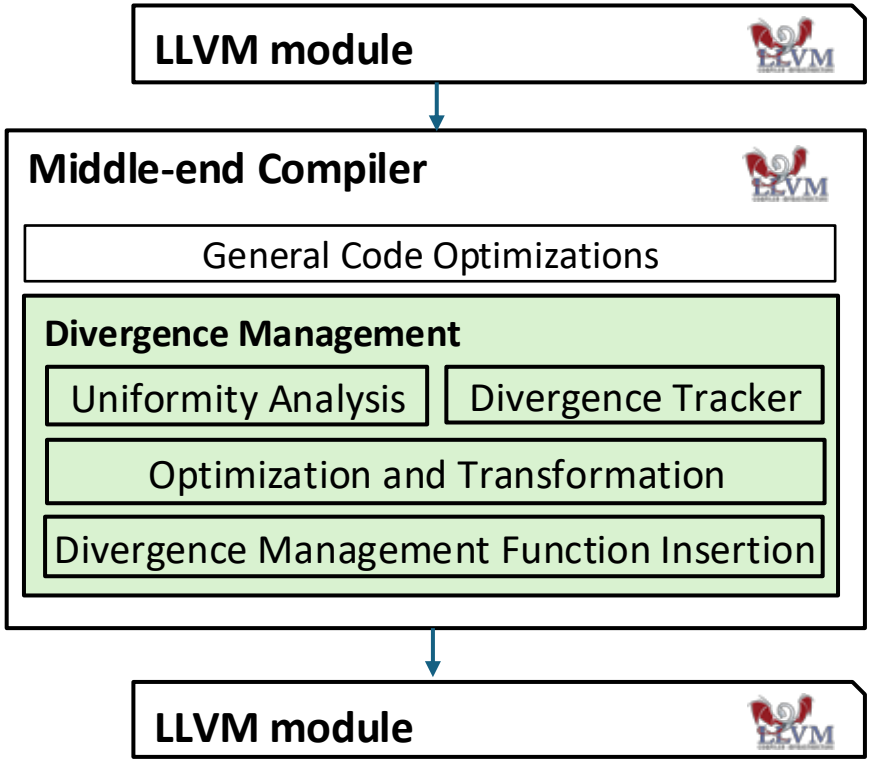
Front-end Compiler

- **Kernel Code**
  - Performs semantics-aware code opt
  - Transforms special function using built-in libraries
    - Lower math function using specific func
    - Lower NVVM IR to CuPBoP structure
  - Inserts thread scheduling and spawning code for device initialization



Middle-end Compiler

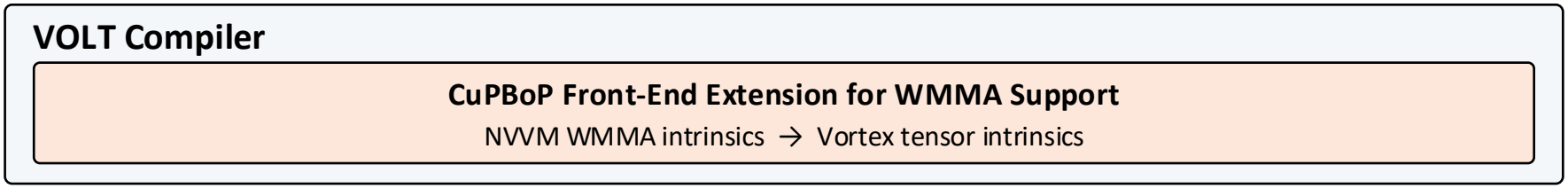
- **Target-independent optimizations**
- Minimal target-specific logic
- Thread Divergence management treated as a first-class concern
  - Uniformity analysis
  - Divergence tracking
  - Control-flow optimizations
  - Instruction count reduction optimization



End-to-End ML Support on Vortex

- Central goal: reuse existing ML application code assets
- Most ML code targets CUDA and NVIDIA Tensor Cores
- → Porting from CUDA is essential
- But CUDA support alone is not enough
- Extension needed at high-level layers
  - ML graph languages, benchmarks, and ML libraries
- Hardware-level exploration also needed
  - TCU location, in-cluster (DTCU), and sparse TCUs

End-to-End ML Support on Vortex



End-to-End ML Support on Vortex

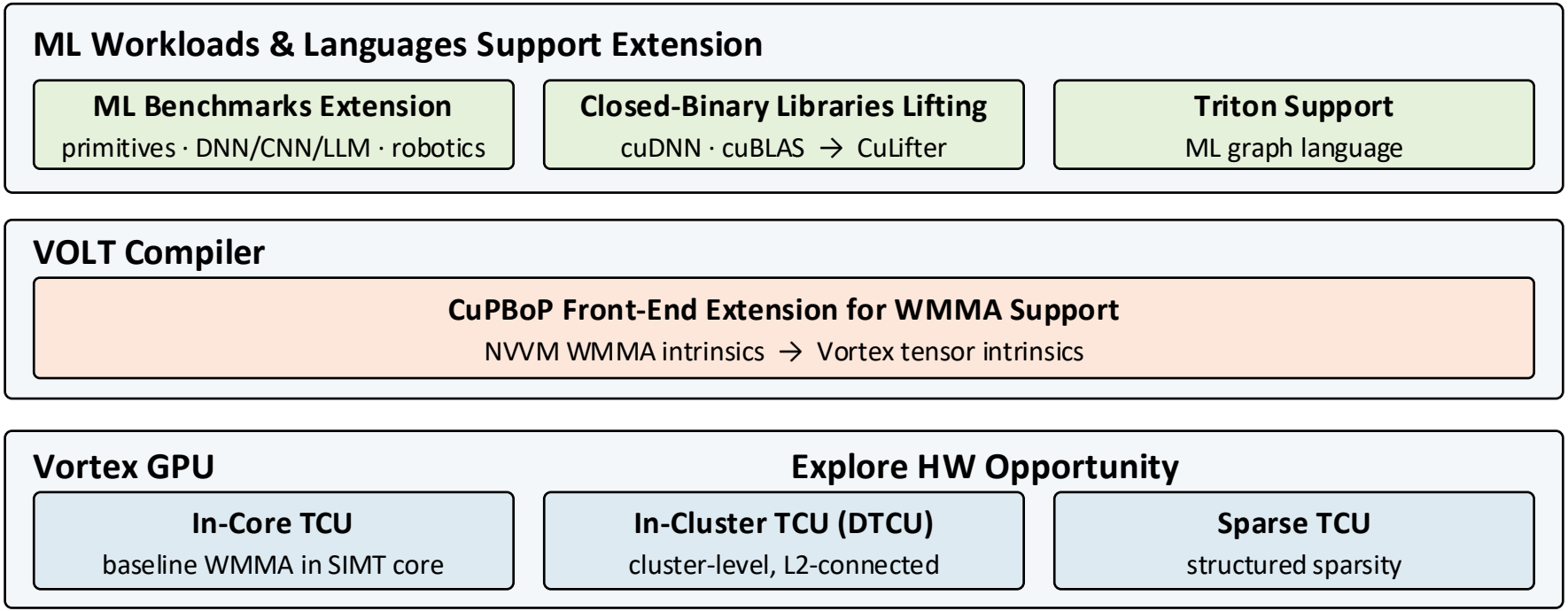
ML Workloads & Languages Support Extension

- **ML Benchmarks Extension**
  primitives · DNN/CNN/LLM · robotics
- **Closed-Binary Libraries Lifting**
  cuDNN · cuBLAS → CuLifter
- **Triton Support**
  ML graph language

VOLT Compiler

- **CuPBoP Front-End Extension for WMMA Support**
  NVVM WMMA intrinsics → Vortex tensor intrinsics

End-to-End ML Support on Vortex



Extend CuPBoP Support: WMMA

- **Enable CUDA Tensor Core semantics to run on Vortex Tensor Cores**
  - NVVM IR expresses Tensor Core use as intrinsics (llvm.nvvm.wmma.*)
- Extend CuPBoP-Vortex to **translate NVVM WMMA intrinsics into Vortex intrinsics**
  - Types: F16, F32, unsigned/signed char, int
  - Variadic fragment kinds and tile sizes
- Tile-size-based Vortex intrinsic insertion
  - Vortex supports reconfigurable tile sizes, while NVVM IR fixes tile counts per target GPU
  - Smaller tiles → padding, larger tiles → tiling
- Direct intrinsic call support
  - Expose Vortex direct intrinsic calls to CUDA code for diverse sizes

Extend Benchmark Support

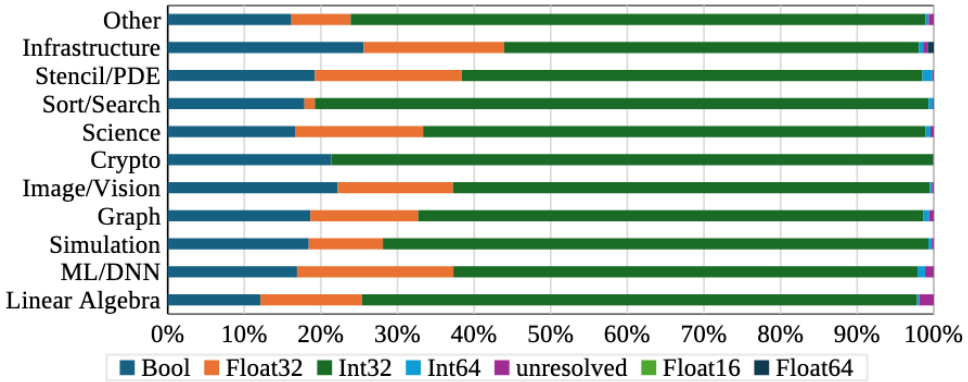
- Broader ML support needs workloads beyond small unit tests
- Extended Vortex benchmark suite now covers:
  - ML primitives, fused operators, DNN/CNN pipelines
  - Neural rendering and graph workloads
  - KernelBench
  - Robotics and robotics-ML kernels, with SIMT and TCU variants
- Implemented as native Vortex C++ on the Vortex build and simulator infrastructure
- Future work: Extend support to CUDA and Triton kernels

Extend ML Application Support: CuLifter

- Many GPU ML libraries are closed-source
  - e.g., cuDNN, TensorRT, cuBLAS, NCCL
  - Real ML applications depend heavily on them, so support is critical
- Distributed as binaries in SASS assembly, not high-level NVVM IR
- Accurate support is impossible without lifting SASS to LLVM/NVVM IR

Extend ML Application Support: CuLifter

- Framework that recovers analyzable LLVM IR from SASS binaries[3]
- Solves the key GPU lifting challenge: Type recovery
- Supports SM75 through SM120, lifting all kernels to valid IR with 96.2% execution correctness
- **Supports diverse benchmarks**
  - HeCBench, CUDA SDK
  - cuBLAS
  - CUTLASS
  - FlashAttention
  - cuDNN CNN/Ops/Adv



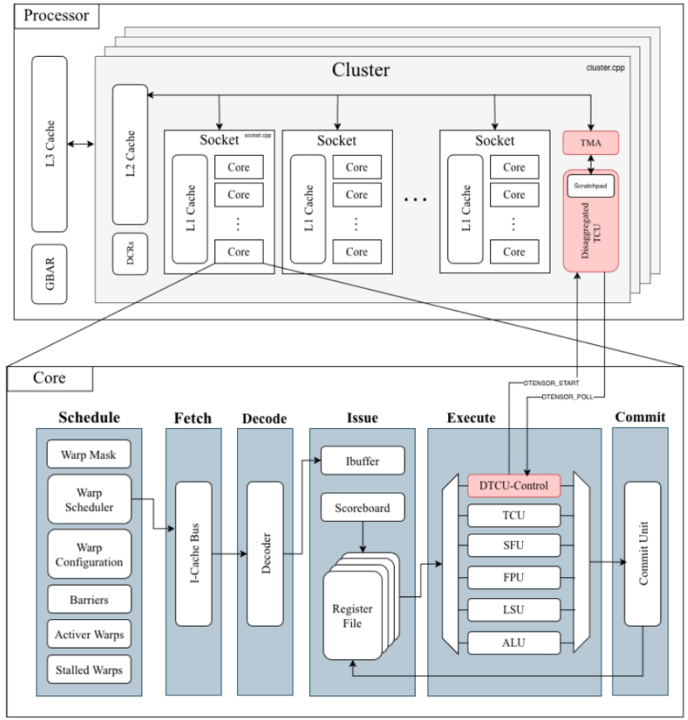
[3] Jisheng Zhao and Huanzhi Pu and Shinnung Jeong and Chihyo Ahn and Hyesoon Kim. 2026, CuLifter: Lifting GPU Binaries to Typed IR, Arxiv, https://arxiv.org/abs/2604.27486

Extend Graph Language Support: Triton

- Triton is a widely used ML graph language with growing adoption
  - Lowers operators to CUDA code or to LLVM IR
- Setup Pipeline: Triton to CUDA and compile with CuPBoP
  - Generate CUDA code through Triton's CUDA lowering
  - Lower that CUDA code through CuPBoP
- Simple operators such as GEMM lower successfully today
- Future work: support diverse real ML models

Explore HW Opportunity: TCU Location

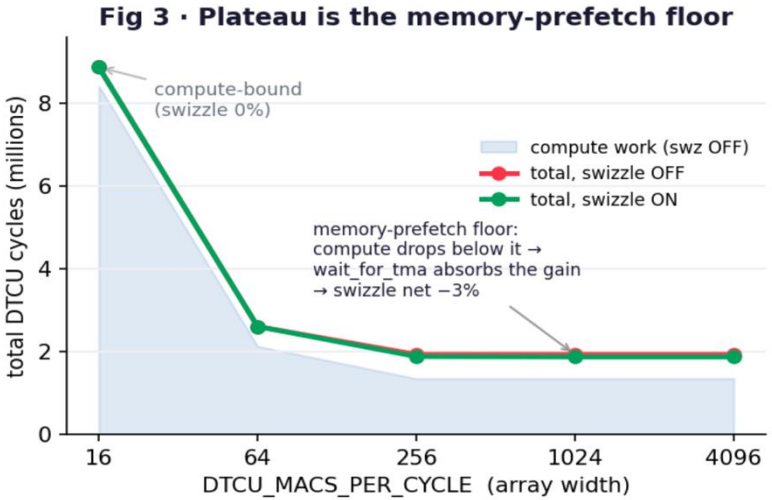
- In-core TCU leans on the Register File for operand/accumulator storage
- High register pressure limits operand/accumulator bandwidth
- Explore architecture opportunities for diverse TCU locations and scale
  - Inside core vs inside cluster, inspired by Virgo[4]



[4] Hansung Kim, et. al. 2025. Virgo: Cluster-level Matrix Unit Integration in GPUs for Scalability and Energy Efficiency, ASPLOS

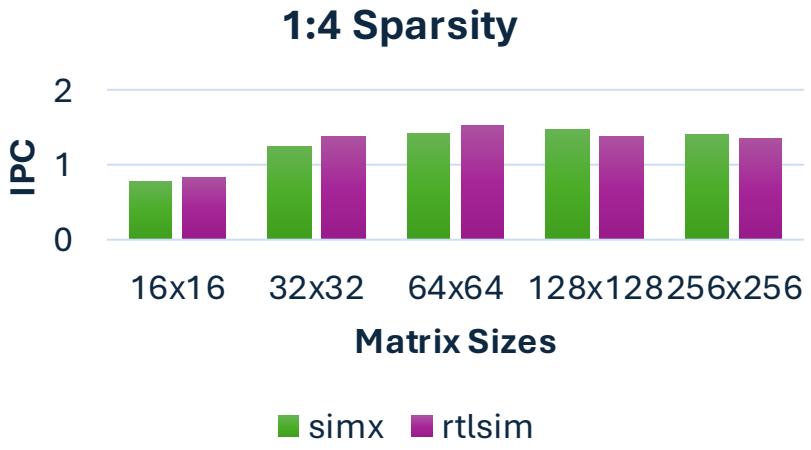
Explore HW Opportunity: TCU Location

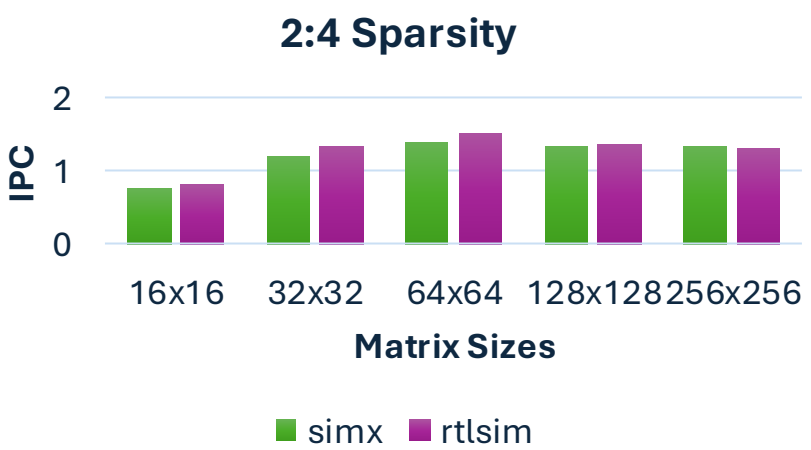
- DTCU
  - Directly connected to the L2 cache
  - Overlaps data movement with compute and memory operations
  - TMA engine with swizzling
  - Targets large-GEMM support
- Comparison to in-core TCU on Simx
  - 1024×512×256, fp16→fp32, NT=4)
  - Cycles: 130.2M → 8.87M (≈15× speedup)
  - Instructions: 20.6M → 0.81M (≈26× reduction)



Explore HW Opportunity: Sparsity

- Extend TCU support for varying degrees of structured sparsity
- Extend SimX / RTL support for sparse WMMA





Cost and Power Modeling

- Highly configurable: cores, threads, warps, and TCU params can reshape area, timing, and power
- Evaluate tensor extensions as real architecture tradeoffs, not only features
- Two-step path: RTL power analysis → analytical model in SimX
  - Synthesis + PrimeTime extract ground-truth power per component
  - Toolchain validated end-to-end on a parameterized FIFO (PrimeTime PX)
  - Baseline power generated for Vortex configs (1c4w4t, 4c4w4t)
- Next: VCD(Value Change Dump)-driven dynamic power → per-component coefficients for fast SimX estimation

Conclusion

- Vortex GPU has a tensor core and a compiler stack (VOLT)
- However, End-to-End ML support is inherently cross-layer, spanning languages, compiler, ISA, and tensor hardware
  - Starting by extending the VOLT Compiler stack to lower CUDA WMMA onto Vortex tensor cores
  - Exploring high-level layers: Benchmarks, CuLifter binary lifting, and Triton support
  - Exploring diverse tensor hardware options: in-cluster (DTCU) and sparse TCUs

Thank You!