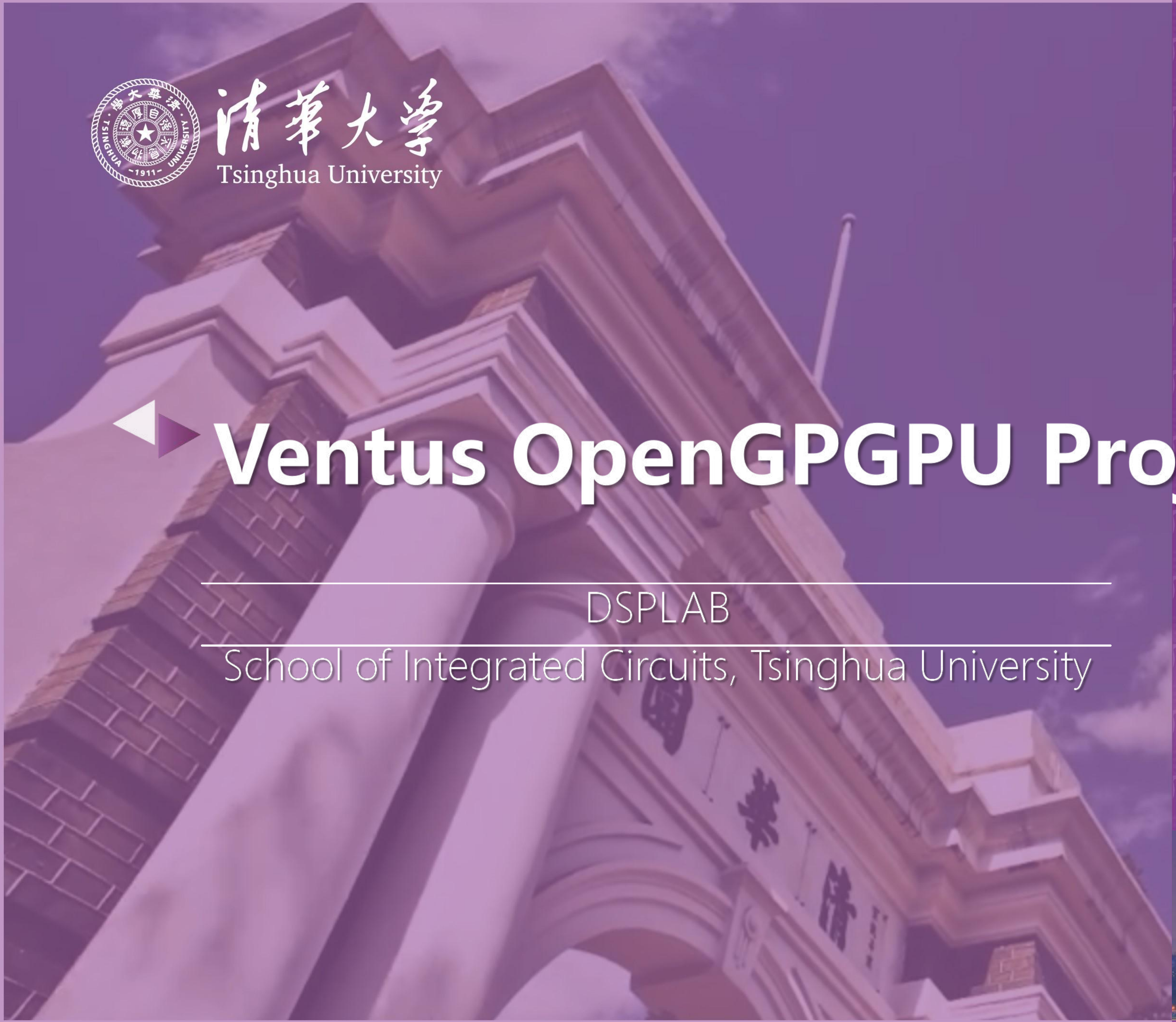
OpenGPGPU
乘影

清華大學
Tsinghua University



Ventus OpenGPGPU Project

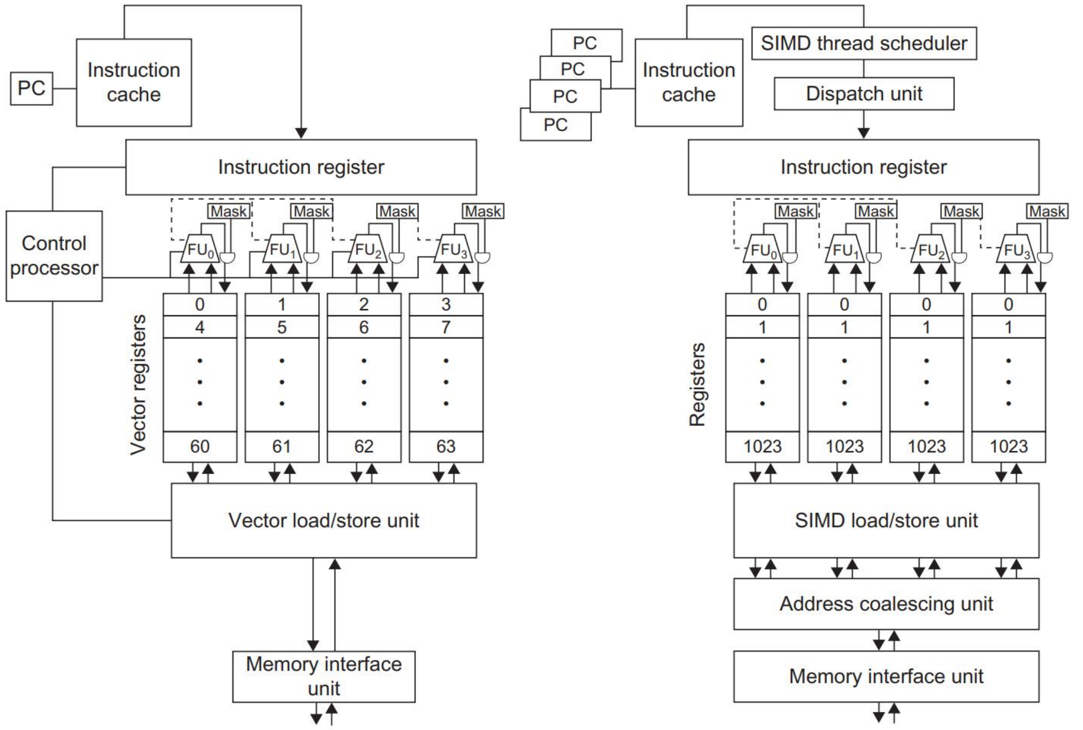
DSPLAB

School of Integrated Circuits, Tsinghua University

MICRO 2025
58th IEEE/ACM International Symposium on Microarchitecture®

Vector Processor vs. GPGPU

- Vector processors are similar to SIMT architectures in terms of computation, memory access, branch, task control, and register stack.
- Motivation:
  - GPGPUs offer much better scalability and compiler simplicity compared to SIMD.
- Map vector lanes to GPGPU threads

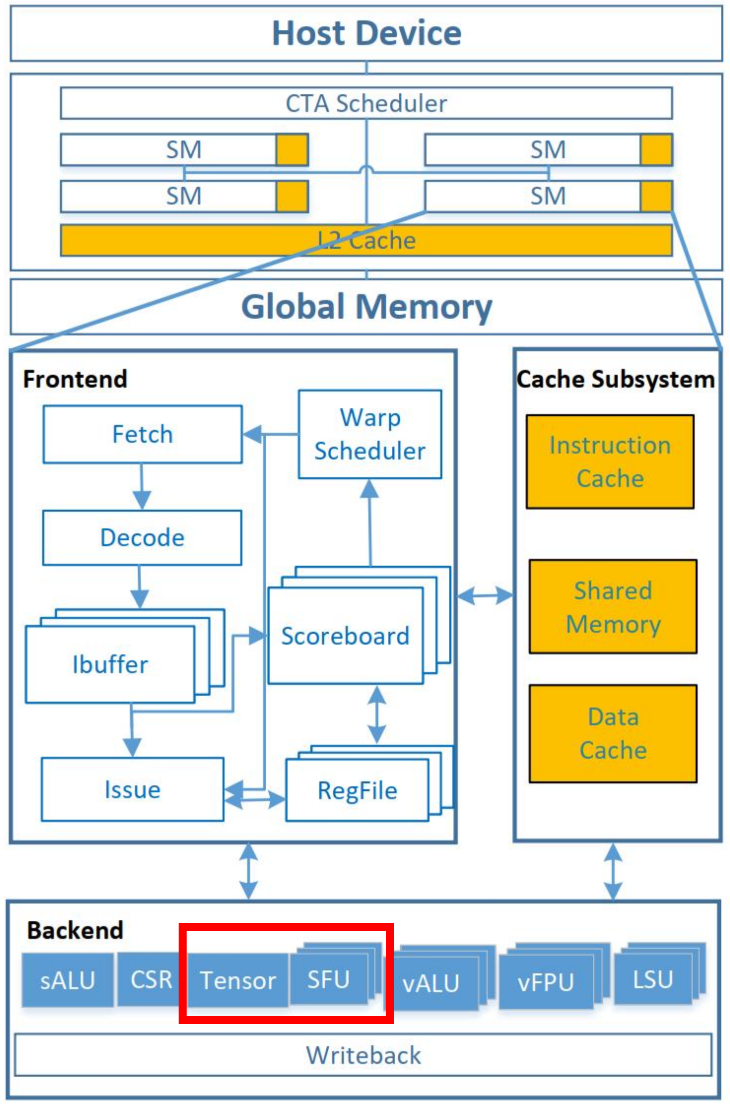


	Computation	Memory Access	Branch	Task control	Register	Architecture
Vector	Vector Lane	Gather/Scatter	Mask Registers	Control Processor	Vector Registers	Vector Processor
GPGPU	SIMD Lane	Global load/store	Predicate Registers	Thread Block Scheduler	SIMD Lane Registers	Multithreaded SIMD Processor

*Computer Architecture, A Quantitative Appro

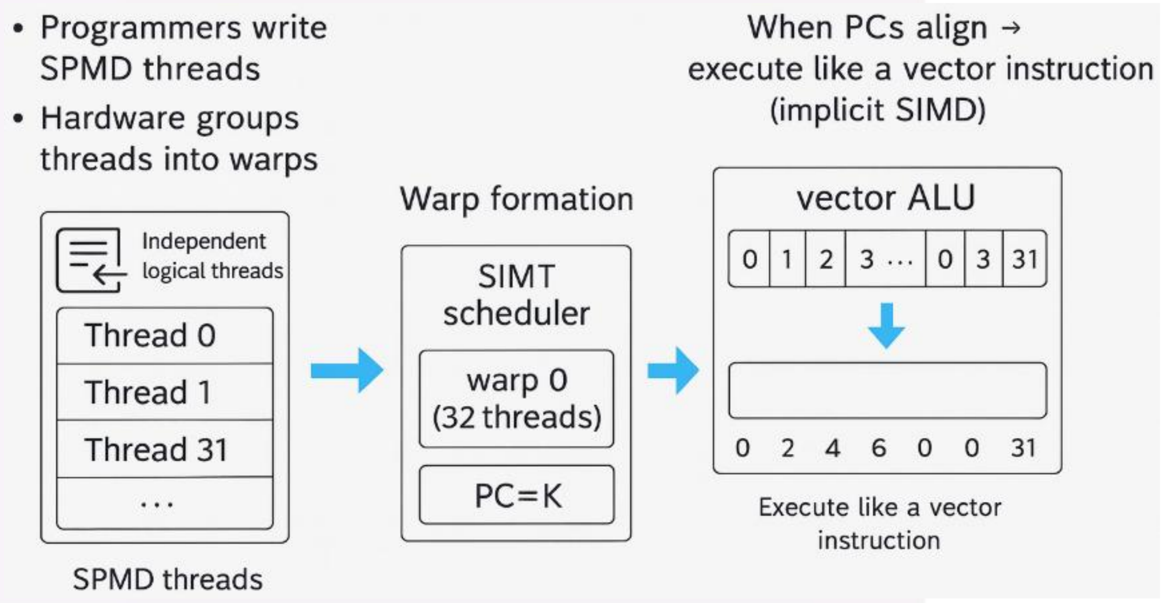
Ventus GPGPU Design

- RISC-V scalar and its vector extension are chosen to be the basic ISA of Ventus GPGPU
- The philosophy of converting a Vector Processor's ISA to a high-performance GPGPU is driven by the underlying architectural similarity between a vector processor's SIMD lanes and a GPGPU's datapath.
- An open-source, high-performance GPGPU hardware based on Chisel HDL.
- A holistic software toolchain from OpenCL to Ventus GPGPU ISA.



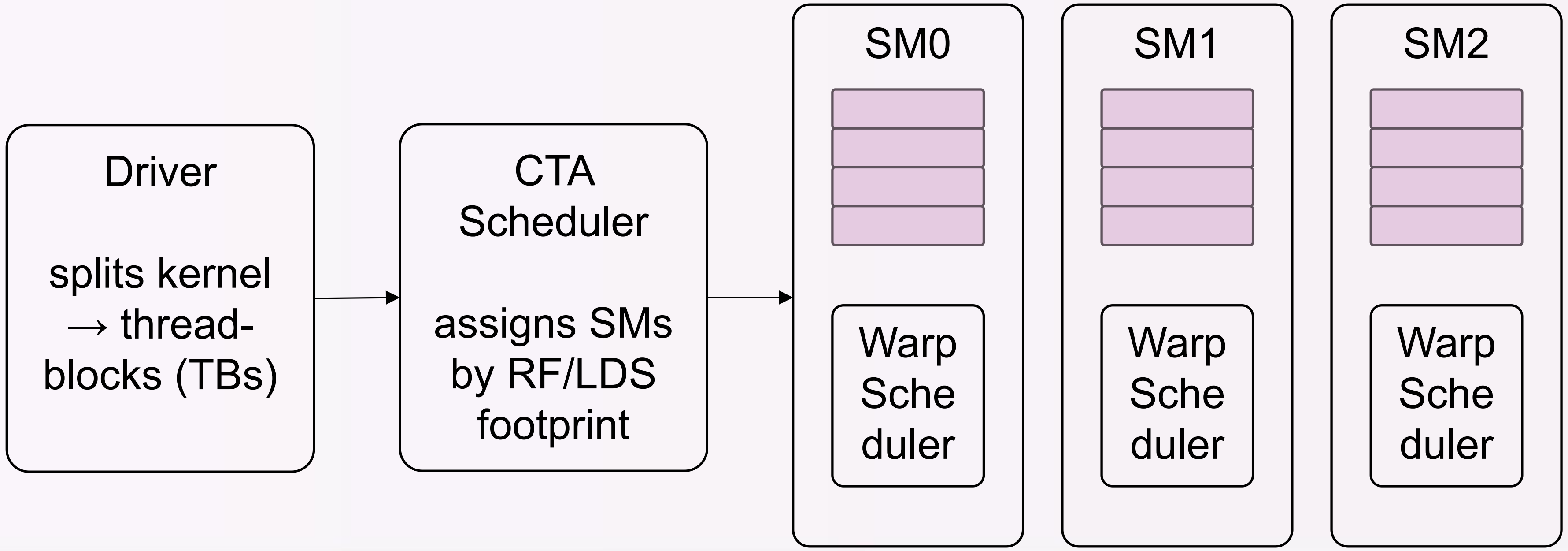
From SIMD to SIMT (Implicit SIMD)

- Programmers write **SPMD threads**
- Hardware groups threads into **warps**
- When PCs align → execute like a **vector** instruction (implicit SIMD)
- **Warp as a RISC-V V Program:**
- Choose **Zve32f**; fix **VL=32, SEW=32**


Programmers write SPMD threads
Hardware groups threads into warps
When PCs align → execute like a vector instruction (implicit SIMD)
Independent logical threads
Thread 0
Thread 1
Thread 31
...
SPMD threads
Warp formation
SIMT scheduler
warp 0 (32 threads)
PC=K
vector ALU
0 1 2 3 ... 0 3 31
0 2 4 6 0 0 31
Execute like a vector instruction

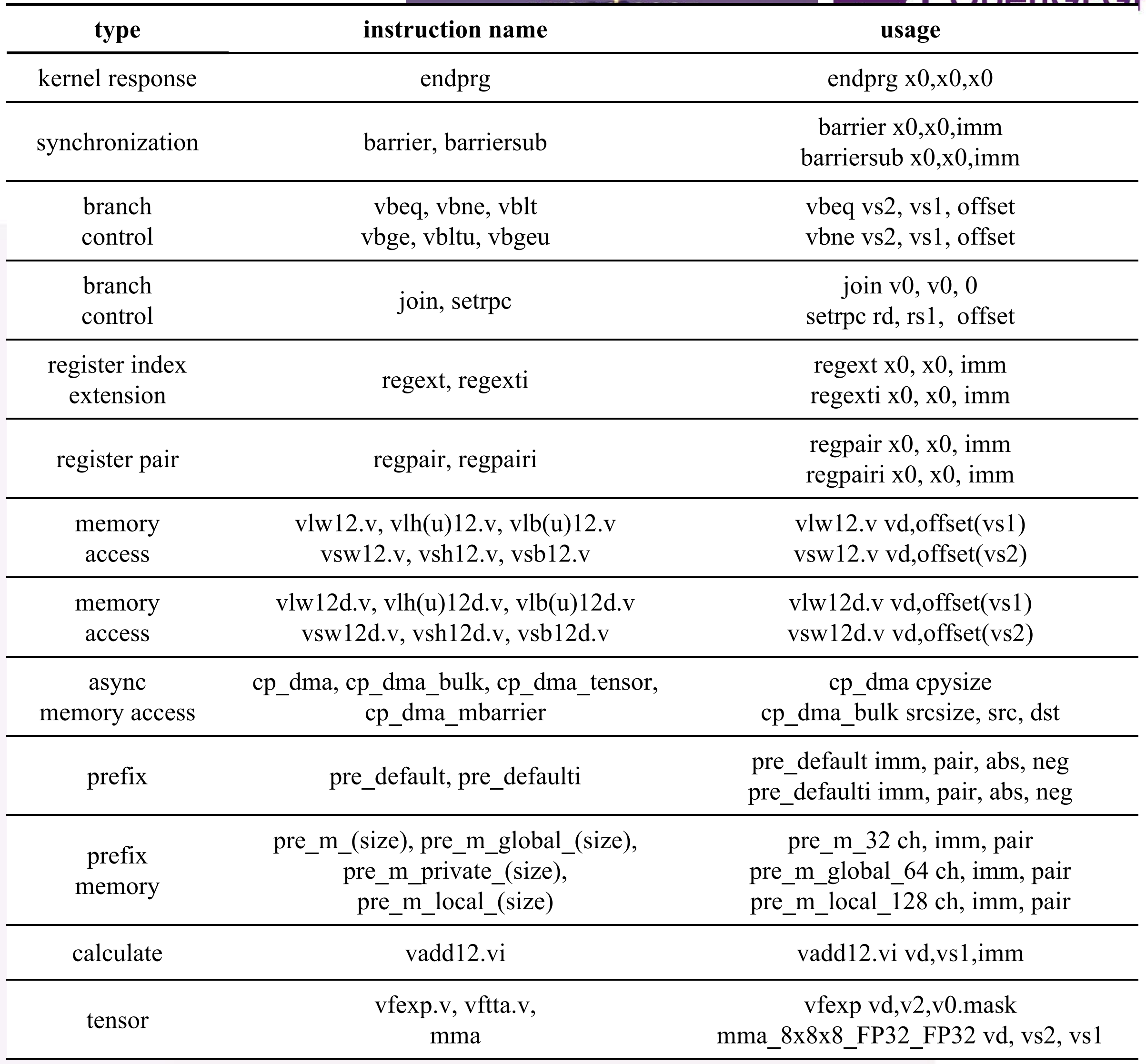
Two-Level Scheduling: Blocks and Warps

- **Driver** splits kernel → thread-blocks (TBs)
- **TB scheduler** assigns SMs by RF/LDS footprint
- **Warp scheduler** handles start/order/sync/exit + SIMT stack



Ventus GPGPU ISA

EXT	Inst Type	1.0 Ver	2.0 Ver	3.0 Ver
V	Configuration-Setting	partially supported	partially supported	partially supported
	Loads and Stores	supported	supported	supported
	Integer Arithmetic	supported	supported	supported
	Floating-Point	supported	supported	supported
	Reduction	×	added per requirements	added per requirements
	Mask	partially supported	added per requirements	added per requirements
I		partially supported	supported	supported
M		supported	supported	supported
F		supported	supported	supported
D		×	×	supported
A		×	supported	supported
RV64		×	×	Supported via transformation

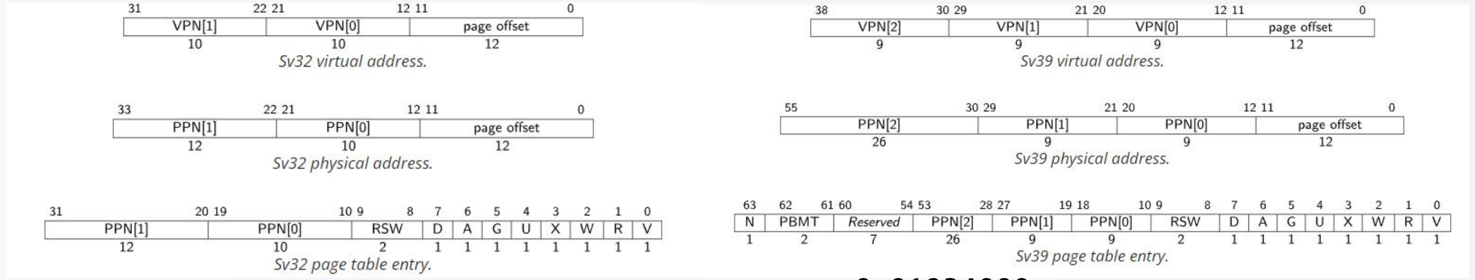


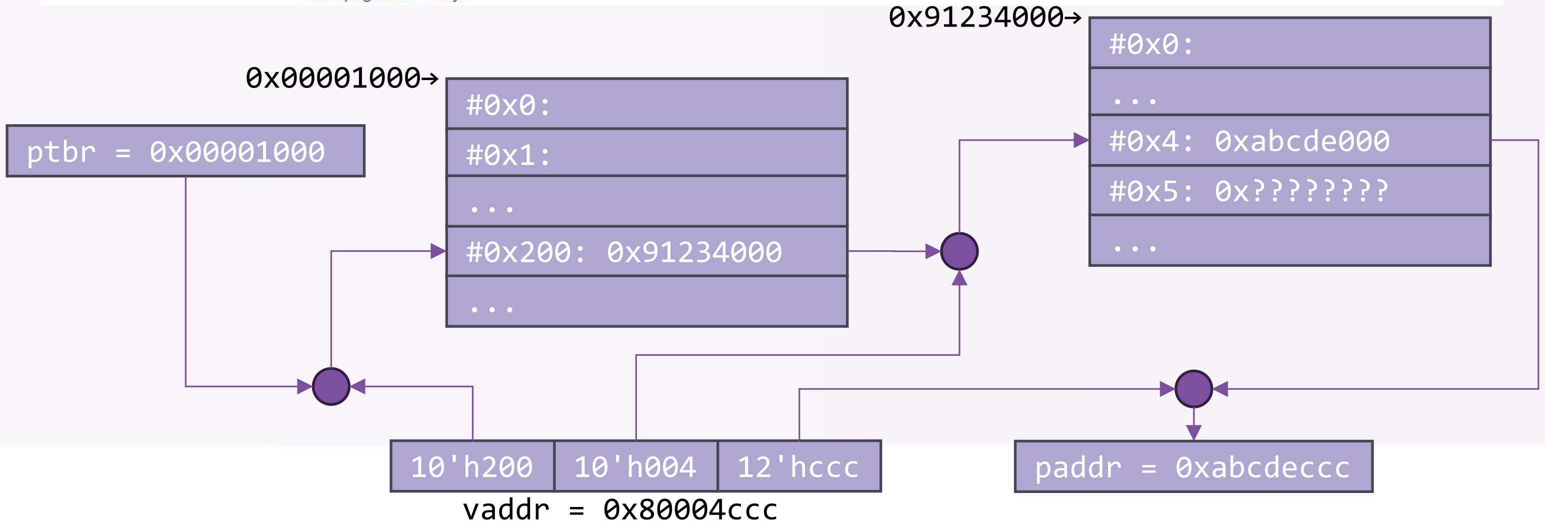
type	instruction name	usage
kernel response	endprg	endprg x0,x0,x0
synchronization	barrier, barriersub	barrier x0,x0,imm barriersub x0,x0,imm
branch control	vbeq, vbne, vblt vbge, vbltu, vbgeu	vbeq vs2, vs1, offset vbne vs2, vs1, offset
branch control	join, setrpc	join v0, v0, 0 setrpc rd, rs1, offset
register index extension	regext, regexti	regext x0, x0, imm regexti x0, x0, imm
register pair	regpair, regpairi	regpair x0, x0, imm regpairi x0, x0, imm
memory access	vlw12.v, vlh(u)12.v, vlb(u)12.v vsw12.v, vsh12.v, vsb12.v	vlw12.v vd,offset(vs1) vsw12.v vd,offset(vs2)
memory access	vlw12d.v, vlh(u)12d.v, vlb(u)12d.v vsw12d.v, vsh12d.v, vsb12d.v	vlw12d.v vd,offset(vs1) vsw12d.v vd,offset(vs2)
async memory access	cp_dma, cp_dma_bulk, cp_dma_tensor, cp_dma_mbarrier	cp_dma cpysize cp_dma_bulk srcsize, src, dst
prefix	pre_default, pre_defaulti	pre_default imm, pair, abs, neg pre_defaulti imm, pair, abs, neg
prefix memory	pre_m_(size), pre_m_global_(size), pre_m_private_(size), pre_m_local_(size)	pre_m_32 ch, imm, pair pre_m_global_64 ch, imm, pair pre_m_local_128 ch, imm, pair
calculate	vadd12.vi	vadd12.vi vd,vs1,imm
tensor	vfexp.v, vftta.v, mma	vfexp vd,v2,v0.mask mma_8x8x8_FP32_FP32 vd, vs2, vs1

Custom instructions
Branching, synchronization, and warp control — SIMT
Register/immediate extension — mitigates register spilling
Register-pair concatenation — enables 64-bit operations
Custom memory-access instructions — as required by the compiler
Tensor operations and exp — supported by the DSA

Virtual Memory: Sv32/Sv39 Specifications

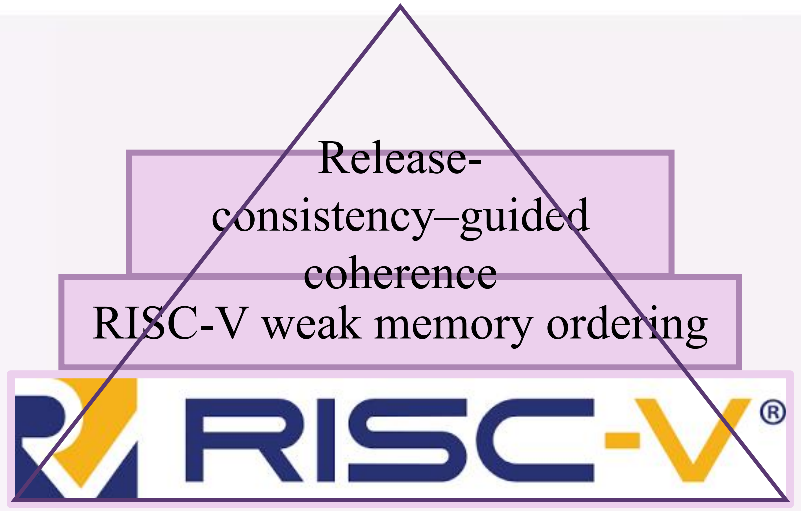
- RISC-V defines a variety of virtual addresses of different lengths and different page table systems
  - Sv32: virtual address 32 bits / physical address 34 bits, two-level page table (10+10)
  - Sv39: virtual address 39 bits / physical address 56 bits, three-level page table (9+9+9)





Ventus GPGPU Cache Coherence

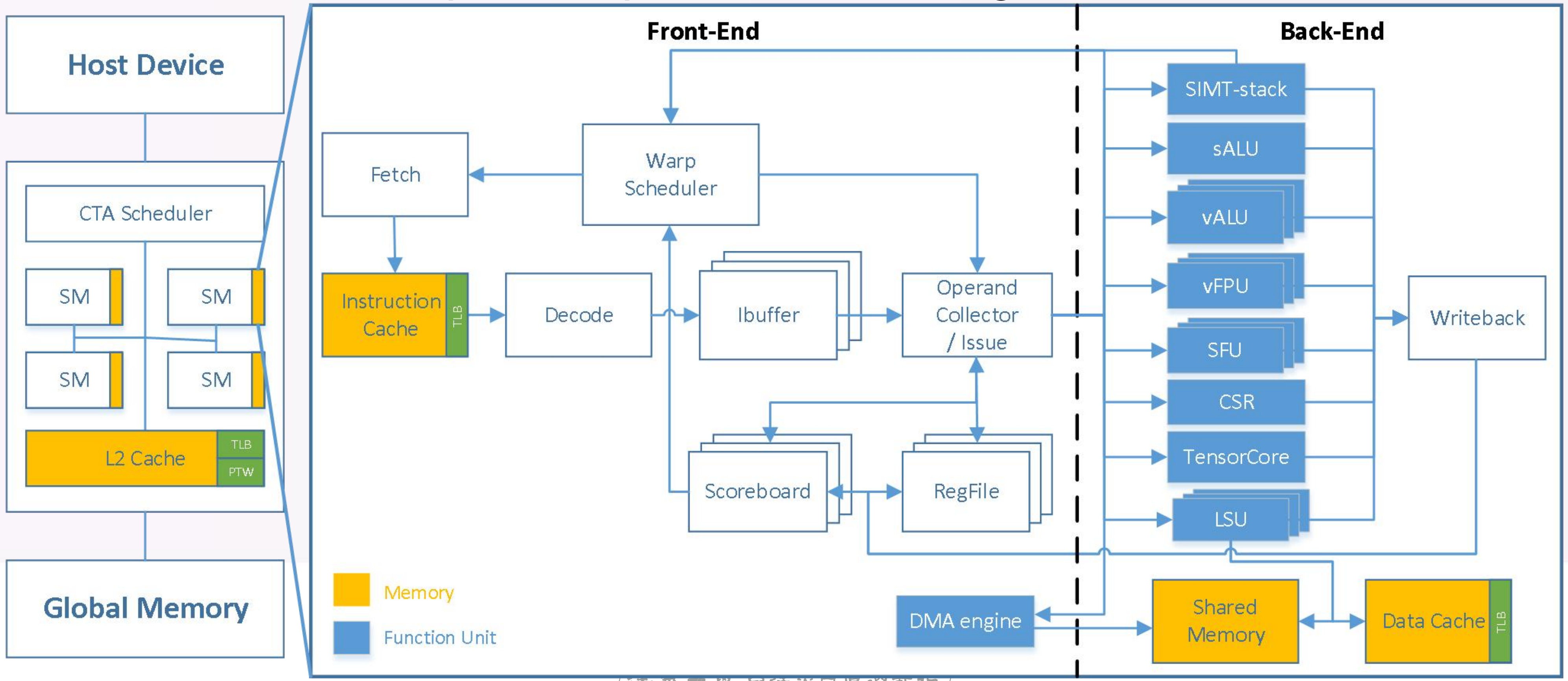
- Built at op the RISC-V weak memory model (RVWMO), **ventus** deploys **release-consistency–guided cache coherence (RCC)**. This enables coherence among SM (streaming multiprocessor) private caches while avoiding the high hardware complexity and runtime bandwidth overhead of full hardware coherence protocols. **Atomic instructions are specified as per-thread behaviors.**


Release-consistency–guided coherence
RISC-V weak memory ordering
RISC-V

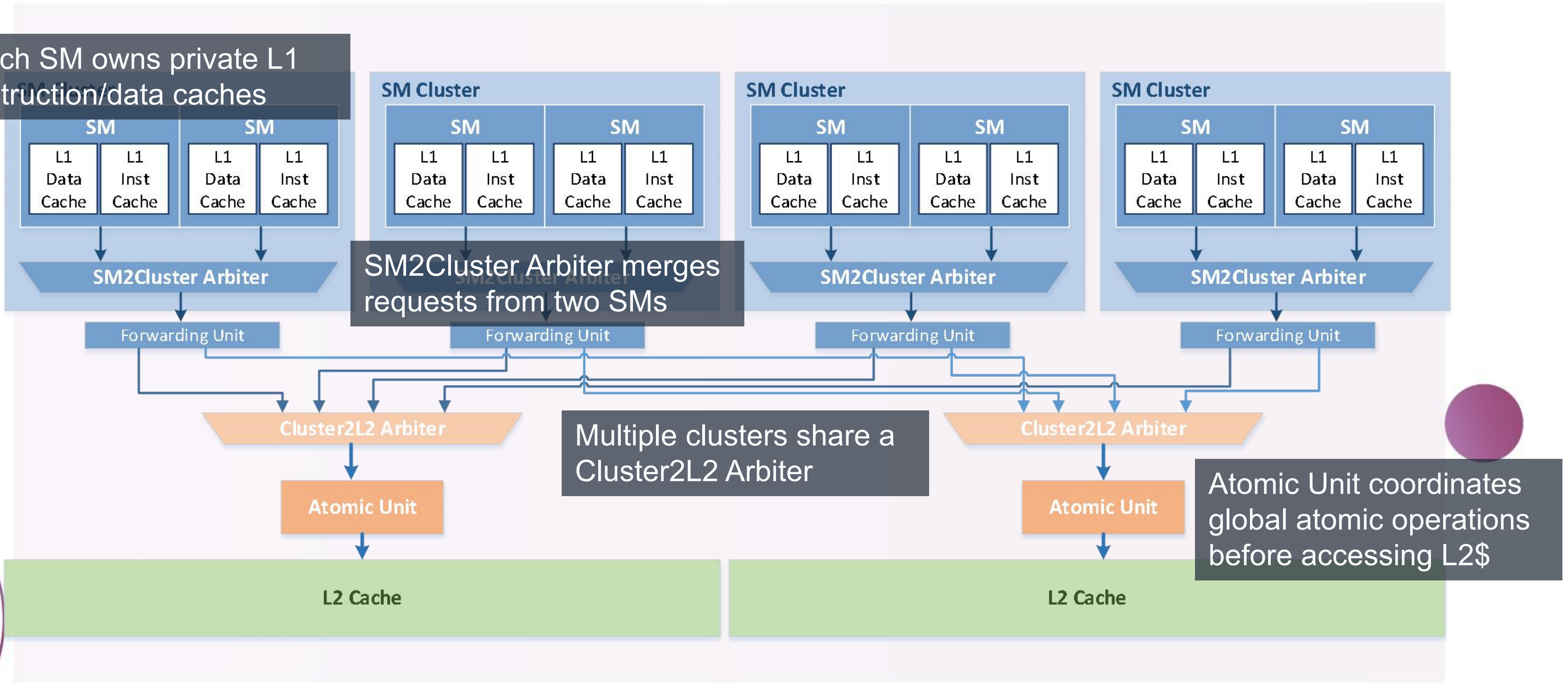
Microarchitectural action	(4) global invalidation	(3) global invalidation	(4) global invalidation
RVWMO marker	.aq qualifier	.rl qualifier	FENCE
RCC coherence operation	“acquire” and “release”	“release”	“acquire” and “release”

SIMT Pipeline Overview

- Classic SM flow: **Fetch → Decode → I- Buffer → Scoreboard → Dispatch → Operand Collect → Issue → Execute → Memory → Writeback**.
- Every in- flight op is tagged with **WID**; front- end is SIMT- aware; back- end treats ops as a pool with WID tags.



Ventus Cache Subsystem Overview



Each SM owns private L1 instruction/data caches

SM2Cluster Arbiter merges requests from two SMs

Multiple clusters share a Cluster2L2 Arbiter

Atomic Unit coordinates global atomic operations before accessing L2$

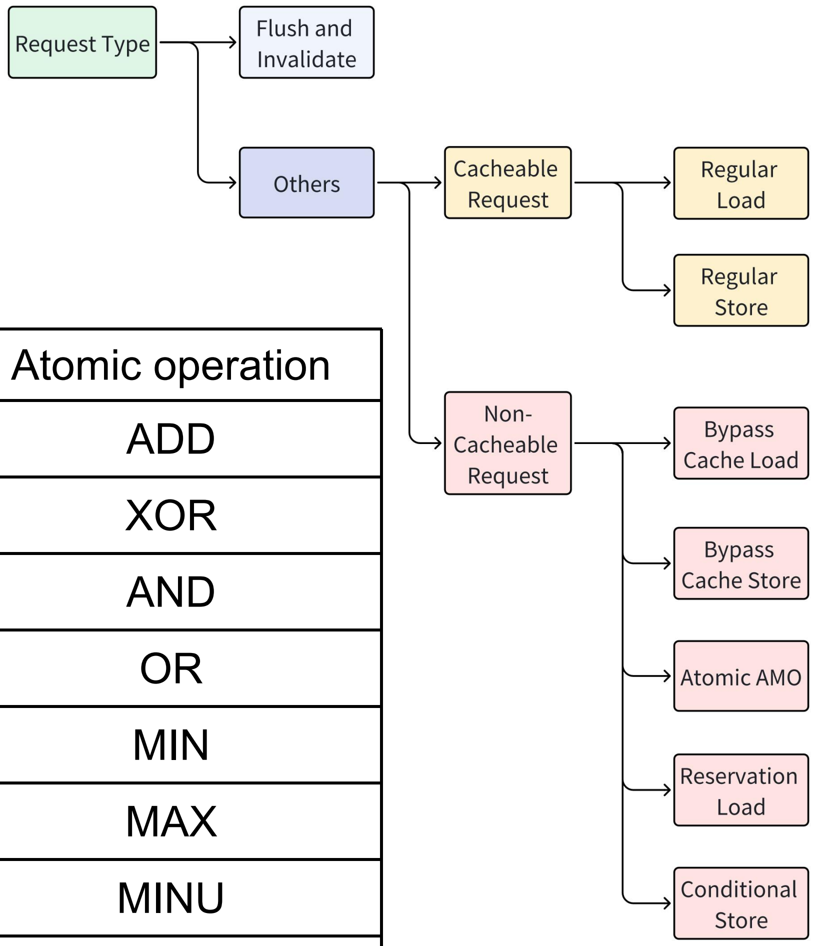
MICRO 2025
58th IEEE/ACM International Symposium on Microarchitecture

OpenGPGPU 乘影

L1 DCache Supported Operations

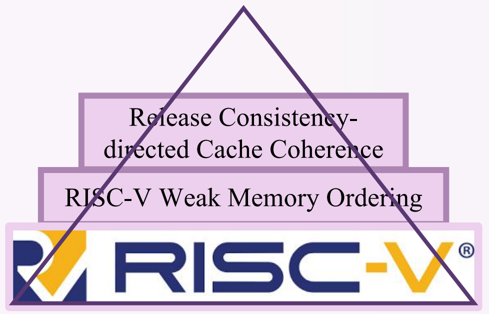
Opcode	Parameter	Operation
0	0	Normal load
0	1	Load-Reserved (LR)
0	2	Uncached load (bypass L1)
1	0	Normal store
1	1	Store-Conditional (SC
1	2	Uncached store (bypa
2	0-7	Atomic AMO
3	0	Global invalidate (L2)
3	1	Global flush (L2)
3	2	Wait until MSHR empt
3	3	L1 invalidate
3	4	L1 flush

Parameter	Atomic operation
0	ADD
1	XOR
2	AND
3	OR
4	MIN
5	MAX
6	MINU
7	MAXU

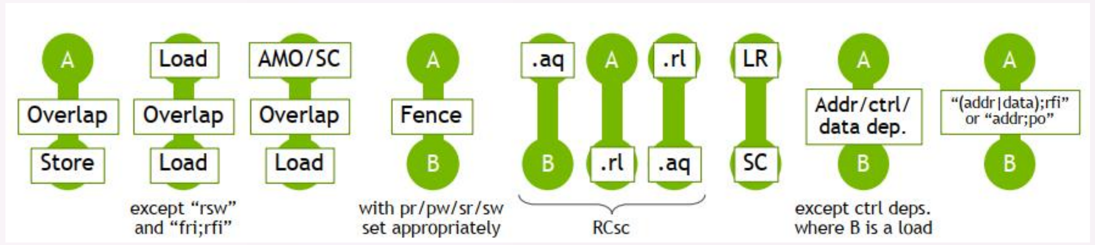


MICRO 2025
58th IEEE/ACM International Symposium on Microarchitecture
OpenGPGPU 乘影

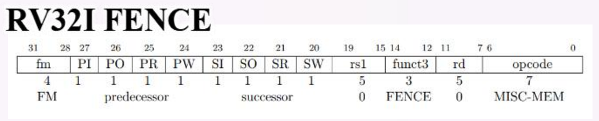
Implementing PPO in Ventus


Release Consistency-directed Cache Coherence
RISC-V Weak Memory Ordering
RISC-V

RVWMO

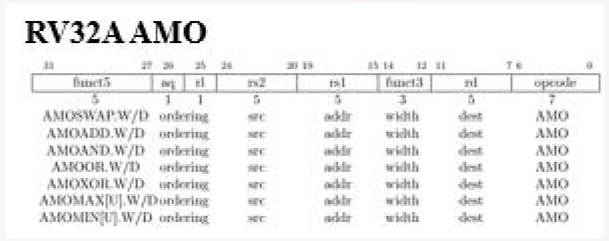

A
Overlap
Store
Load
Overlap
Load
except "rsw" and "fri;rfi"
AMO/SC
Overlap
Load
A
Fence
B
with pr/pw/sr/sw set appropriately
.aq
B
A
.rl
.rl
.aq
RCsc
LR
SC
A
Addr/ctrl/ data dep.
B
except ctrl deps. where B is a load
A
"(addr|data);rfi" or "addr;po"
B

A Visualized Explanation of PPO Rules



RV32I FENCE

31 28	27	26	25	24	23	22	21	20	19 15	14 12	11 7	6 0
fm	PI	PO	PR	PW	SI	SO	SR	SW	rs1	funct3	rd	opcode
4	1	1	1	1	1	1	1	1	5	3	5	7
FM	predecessor				successor				0	FENCE	0	MISC-MEM



RV32A AMO

31 27	26	25	24 20	19 15	14 12	11 7	6 0
funct5	aq	rl	rs2	rs1	funct3	rd	opcode
5	1	1	5	5	3	5	7
AMOSWAP.W/D	ordering		src	addr	width	dest	AMO
AMOADD.W/D	ordering		src	addr	width	dest	AMO
AMOAND.W/D	ordering		src	addr	width	dest	AMO
AMOOR.W/D	ordering		src	addr	width	dest	AMO
AMOXOR.W/D	ordering		src	addr	width	dest	AMO
AMOMAX[U].W/D	ordering		src	addr	width	dest	AMO
AMOMIN[U].W/D	ordering		src	addr	width	dest	AMO

Instruction Encodings for Memory Consistency and Synchronization in RISC-V

➢ Cache Microarchitecture Design

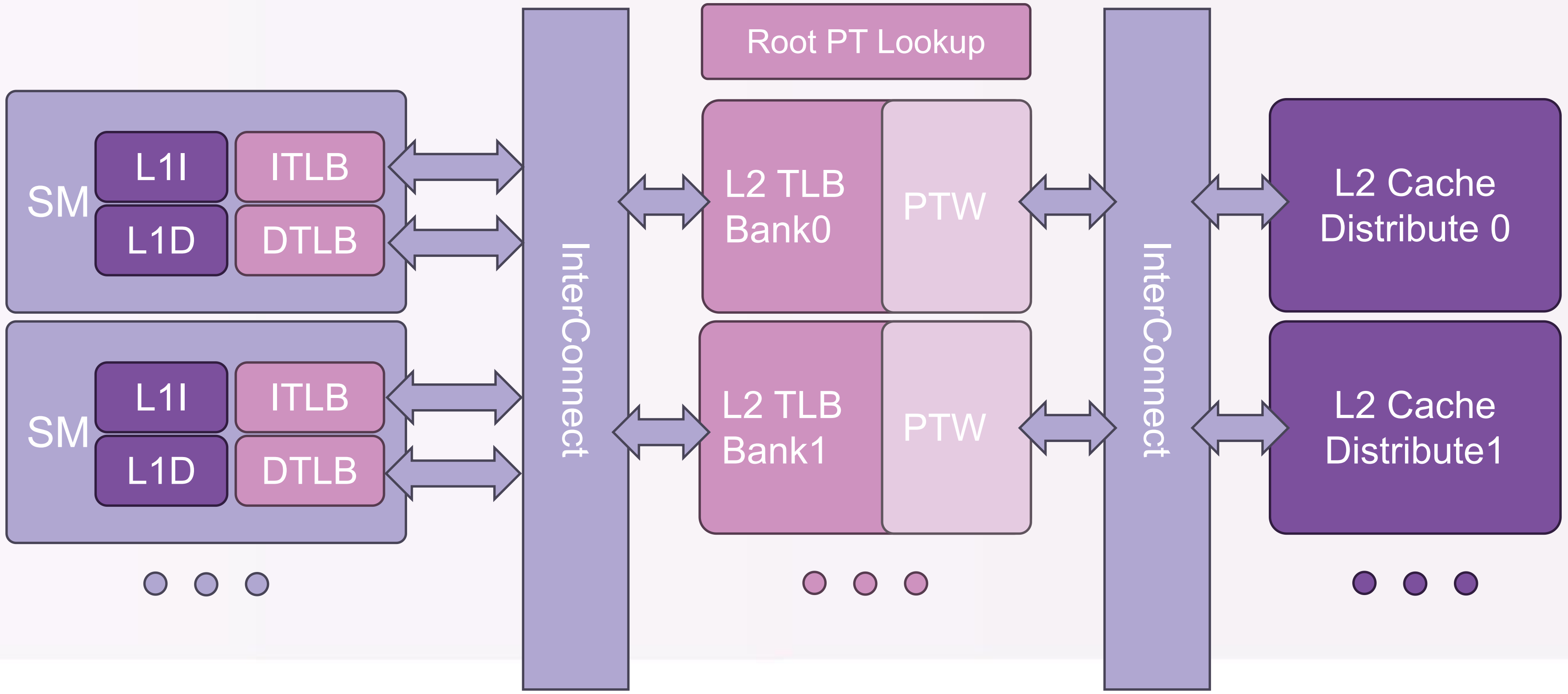
a. Use Miss Status Holding Registers (MSHRs) and Write Status Holding Registers (WSHRs) to address **PPO1 and PPO2**

b. **PPO4–7** correspond to coherence operations such as Fence and acquire/release, which are supported via microarchitectural mechanisms like draining MSHRs, global flushes, and global invalidations.

c. **PPO9–13** are already resolved within the pipeline, while PPO2 and PPO8 are handled by inserting additional checks in the cache pipeline

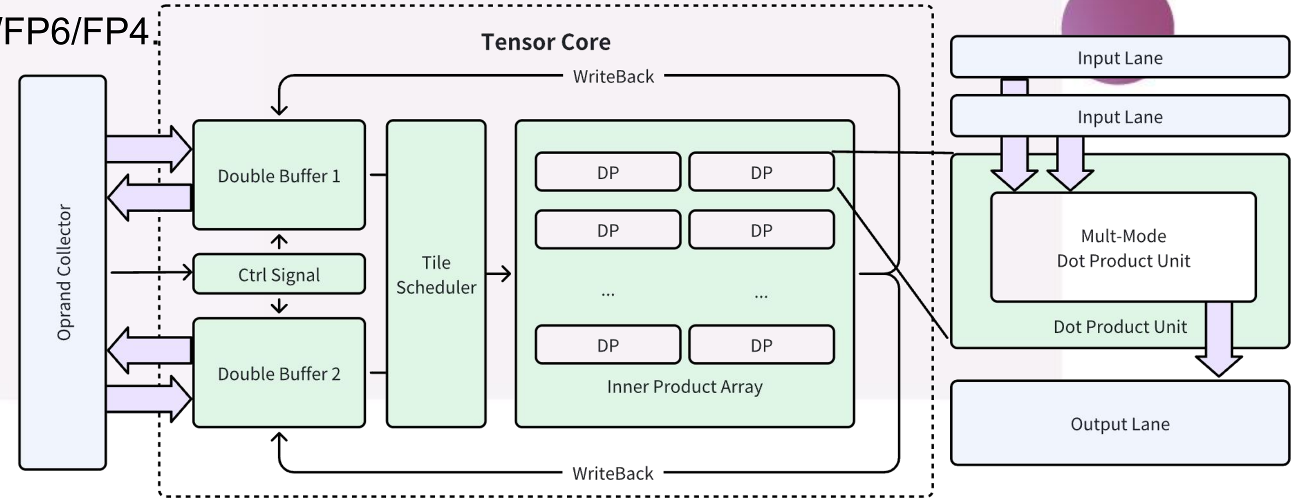
Virtual Memory: Architectural Design

- L1 cache switched to VIVT; L2 remains PIPT
- Hierarchical crossbar interconnection for access



Ventus Tensor Core Overview

- The new version of Ventus with multi-precision tensor core features is as follows:
  - **Computation Scale Expansion:**
    - FP16 matrix multiplication computation scale expanded up to 16×16×16.
    - Low-bit precision computation scales naturally with the k-dimension.
  - **Computation Resource Reuse:**
    - FP16/INT8/INT4 precision dot product units reuse hardware resources.
  - **Increased Computation Precision:**
    - Fine-grained quantization: Supports OCP MX-FP8/FP6/FP4.
    - Low-bit floating-point support: Supports FP8/FP6/FP4.
  - **Supports sparse computation.**



Multi-precision

- Supports multi-precision matrix multiply-and-accumulate (MMA) calculations:
  - **FP16**: Supports both mixed precision and accumulation in FP16 precision.
  - **INT8/INT4**: Reuse dot product units with FP16, accumulation in INT32 precision.
  - **FP8/FP6/FP4**: Supports accumulation in FP32 precision.
  - **MX FP8/FP6/FP4**: Supports fine-grained quantization factors according to the OCP MX standard.

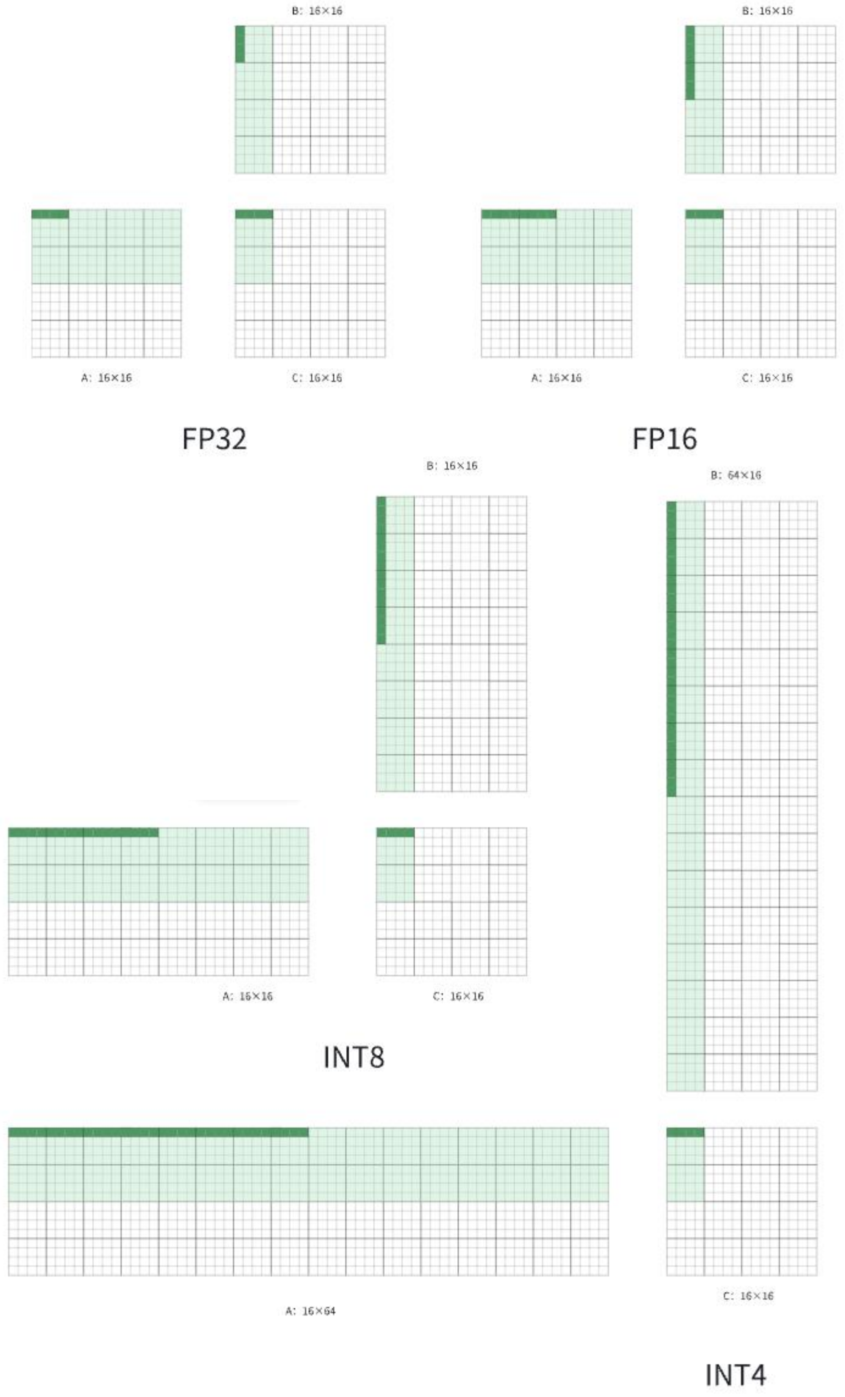
[1] "IEEE Standard for Floating-Point Arithmetic," in IEEE Std 754-2019 (Revision of IEEE 754-2008) , vol., no., pp.1-84, 22 July 2019.
[2] Google, "The Bfloat16 Numerical Format", https://cloud.google.com/tpu/docs/bfloat16.
[3] Micikevicius, Paulius, et al., "OCP 8-bit Floating-Point Specification", June 2023
[4] Bita Darvish Rouhani, et al., "OCP Microscaling Formats (MX) Specification", Sep 2023

Type	Sign	exporent	matissa	Total	Description
FP16	1	5	10	16	From IEEE 754 standard.
BF16	1	8	7	16	A new data type introduced by Google in TensorFlow, commonly used in deep learning training. It better matches the distribution of neural network weights and addresses the issue of FP16's limited expressiveness range during model training.
FP8 E4M3	1	4	3	8	Defined in September 2022 by NVIDIA, Arm, and Intel, FP8 offers a more nonlinear representation range compared to INT8.
FP8 E5M2	1	5	2	8	
INT8	1	N/A	7	8	Quantization precision commonly used in the industry
FP6 E3M2	1	3	2	6	Based on the OCP MX standard
FP6 E2M3	1	2	3	6	
FP4 E2M1	1	2	1	4	
INT4	1	N/A	3	4	Quantization precision.

Multi-size

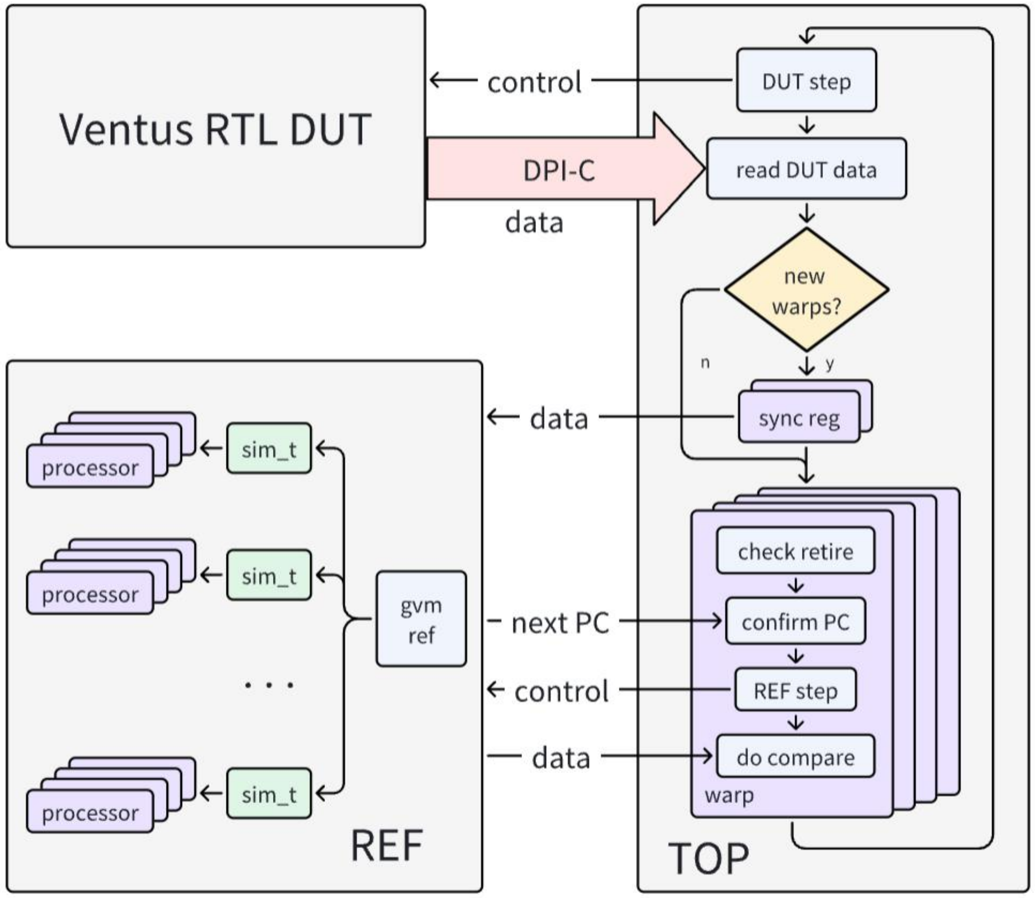
- Supports multi-size matrix multiply-and-accumulate (MMA) calculations:
  - **FP32/FP16/FP16 mixed precision/TF32/BF16/INT8/INT4**: Supports m16n16k16, m32n8k16, m8n32k16.
  - **INT8/FP8**: Supports m16n16k32, m32n8k32, m8n32k32.
  - **INT4**: Supports m16n16k64, m32n8k64, m8n32k64.
- For **INT8/INT4/FP8** and other precisions, the array supports shape expansion in the k-dimension.
- NVIDIA 2:4 sparse acceleration support enables further extension in the k-dimension, doubling theoretical peak performance.

[1] A. Mishra et al., "Accelerating Sparse Deep Neural Networks," Apr. 16, 2021, arXiv: arXiv:2104.08378. doi: 10.48550/arXiv.2104.08378.
[2] C. Zhang et al., "DSTC: Dual-Side Sparse Tensor Core for DNNs Acceleration on Modern GPU Architectures," IEEE Trans. Comput., pp. 1–14, 2024, doi: 10.1109/TC.2024.3475814.



GPU Verification Model（GVM)

- GVM is an UVM-like verification model for GPGPU written in Chisel or Verilog
- Single-cycle step DUT, with instruction dispatch, completion, and other events captured by the top-level simulation program.
- To address the absence of hardware ROB in GPGPUs, which is different from CPU and complicates synchronization between the DUT and Reference, a warp-level software ROB is constructed.
- GVM helps users to debug software and hardware efficiently.



Simulation Framework Update and Toolchain Integration

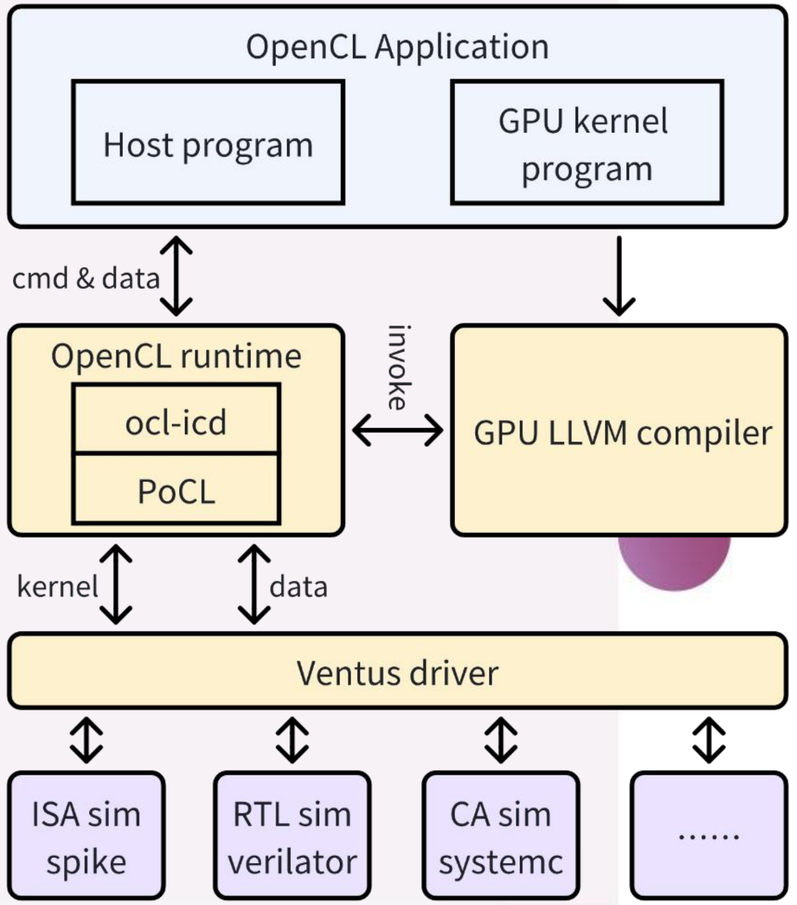
Integration of Chisel RTL Verilator simulation framework and cycle-level simulator into the Ventus toolchain

- Define simulation framework API and package dynamic libraries
- Virtual memory management driver: SV32/SV39
- Multiple simulation backends adopt a unified interface for invocation, facilitating switching

Advantages:

- Simplify the simulation process without manually generating intermediate files for test cases
- Able to verify result correctness in OpenCL programs
- Provide one-click deployment scripts + regression test scripts
- Significantly improved simulation efficiency compared to the chiseltest solution (RTL)
- Avoid repeated compilation and allow multi-threaded parallel simulation

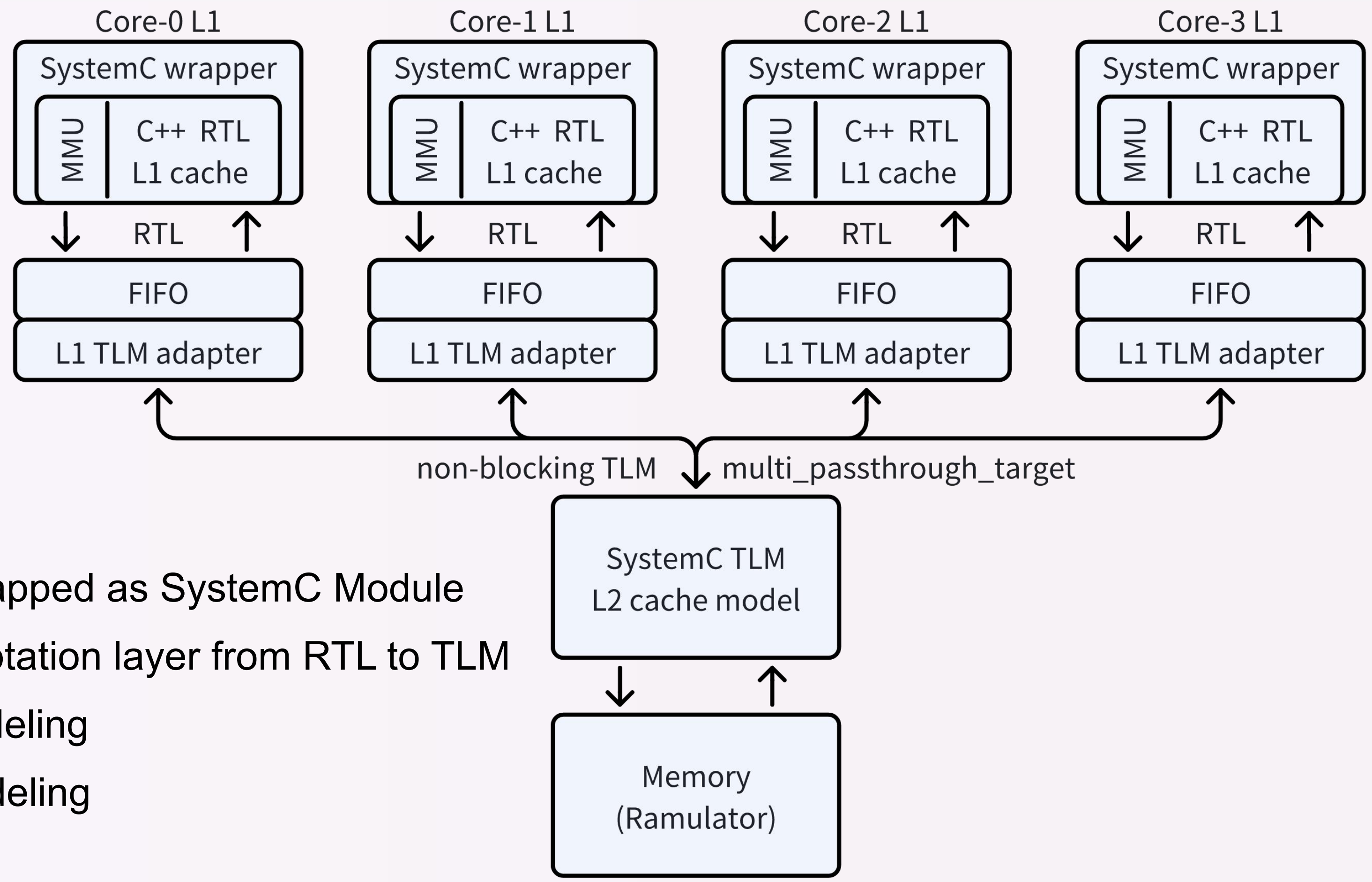
```
VENTUS_BACKEND=spike    ./a.out # Using the Spike Instruction-Level Simulator
VENTUS_BACKEND=rtlsim   ./a.out # Using Chisel RTL Simulation
VENTUS_BACKEND=cyclesim ./a.out # Usingcycle-level Simulator
```



SystemC-based Cycle-Level Simulator

Cache System Modeling

- Atomic Operation Support
- Hit and Latency Statistics

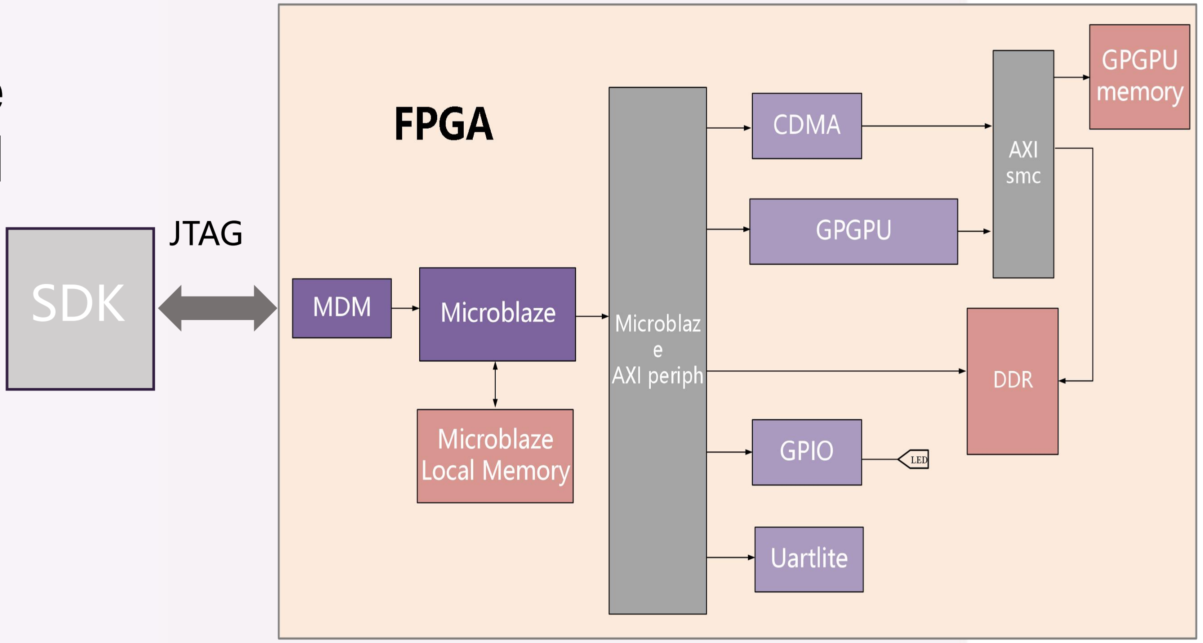


- L1: C++ RTL modeling, wrapped as SystemC Module
- Adapter: Intermediate adaptation layer from RTL to TLM
- L2: Transaction queue modeling
- MMU: Functional-level modeling

Ventus-Community: FPGA Test Architecture

- Control Core: MicroBlaze processor executing testbench logic
- Debugging and Interaction Interface: The .elf file is uploaded to MicroBlaze memory via SDK to control peripheral
- Memory Subsystem: AXI-interfaced DDR for high-capacity data storage

Virtex Ultrascale + HBM VCU128

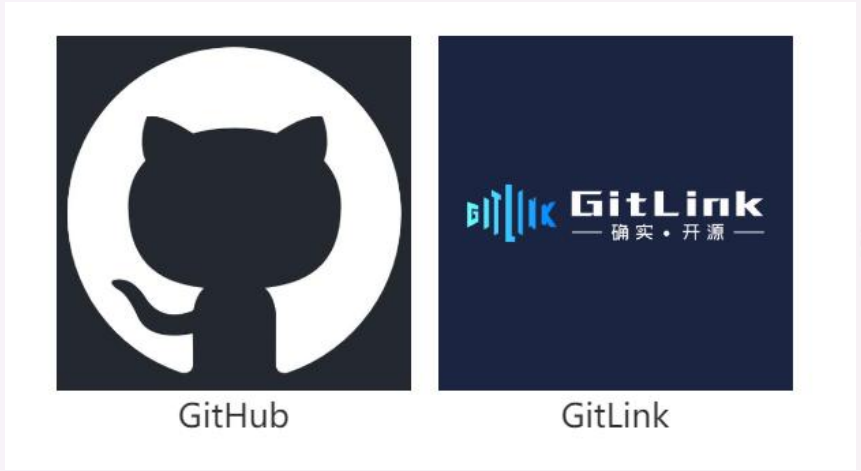


This platform employs a MicroBlaze soft-core processor to manage peripherals (GPGPU, DDR), featuring JTAG debugging and AXI bus communication for streamlined development and testing of hardware acceleration tasks.

https://opengpgpu.org.cn/

- Code repo

https://github.com/THU-DSP-LAB
https://www.gitlink.org.cn/THU-DSP-LAB



THANK YOU

OpenGPGPU 乘影

清華大學教育基金会
TSINGHUA UNIVERSITY EDUCATION FOUNDATION