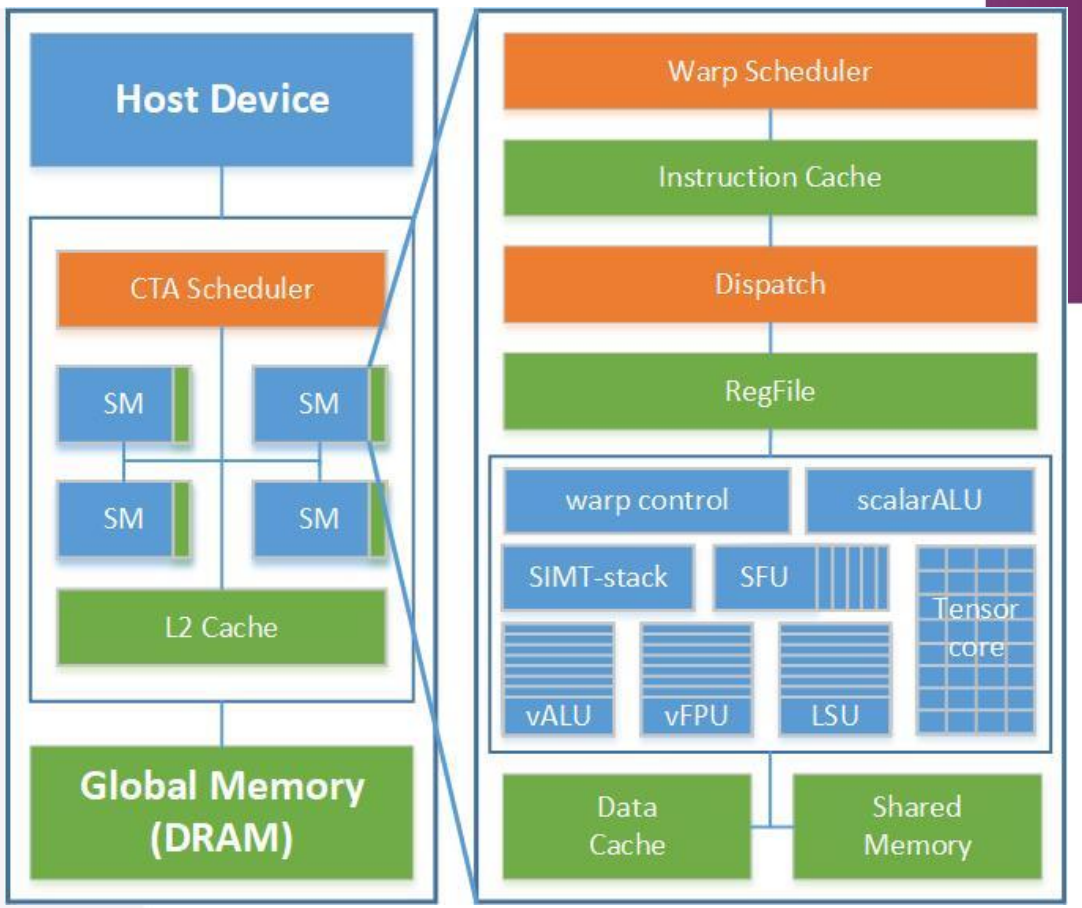
开源通用GPU“乘影”第二版硬件开发进展

于芳菲

清华大学集成电路学院

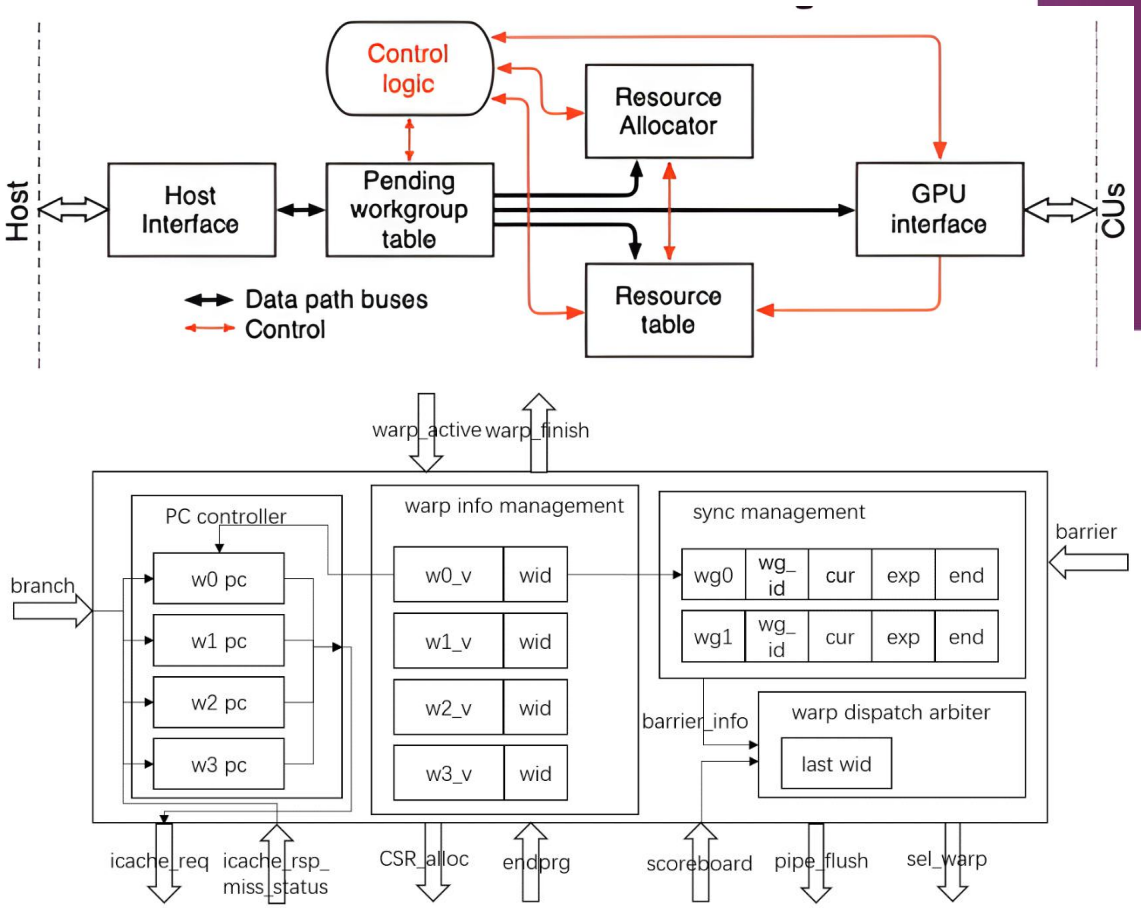
“乘影” GPGPU微架构

- 支持多线程束调度的向量处理器
- 计算任务的启动、分配、执行
- 自定义指令的支持



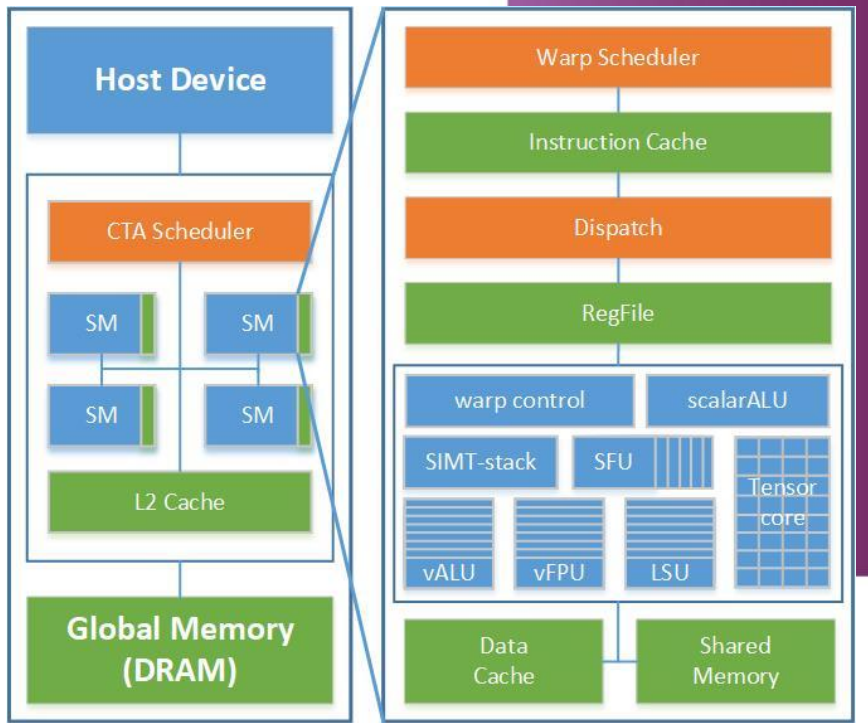
“乘影” GPGPU微架构

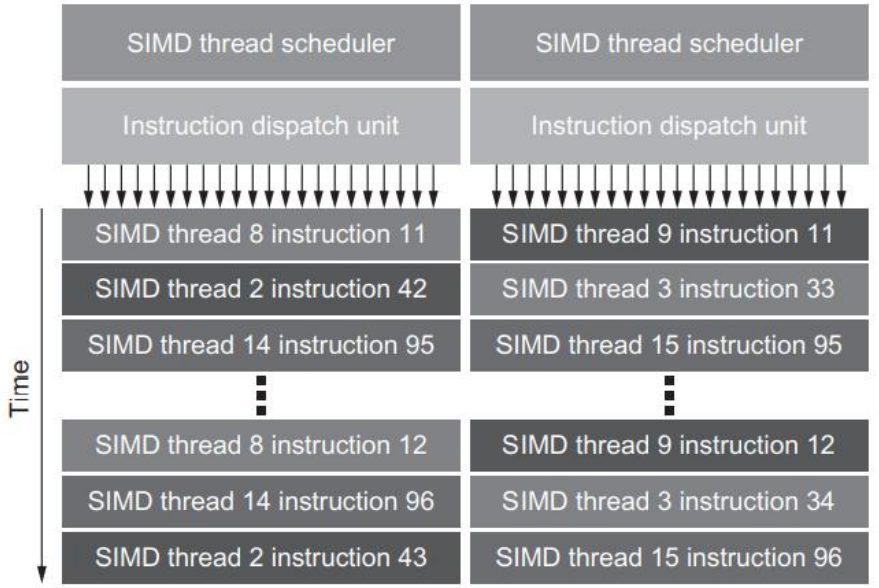
- **任务分配：** Host - CTA Scheduler - SM(streaming multiprocessor)
  - Host将任务以WorkGroup为单位发给GPGPU，包括metadata信息
  - 线程块调度器进行SM资源的管理，以warp为单位发给SM
  - 线程束调度器管理warp基本信息和执行情况
- 任务执行



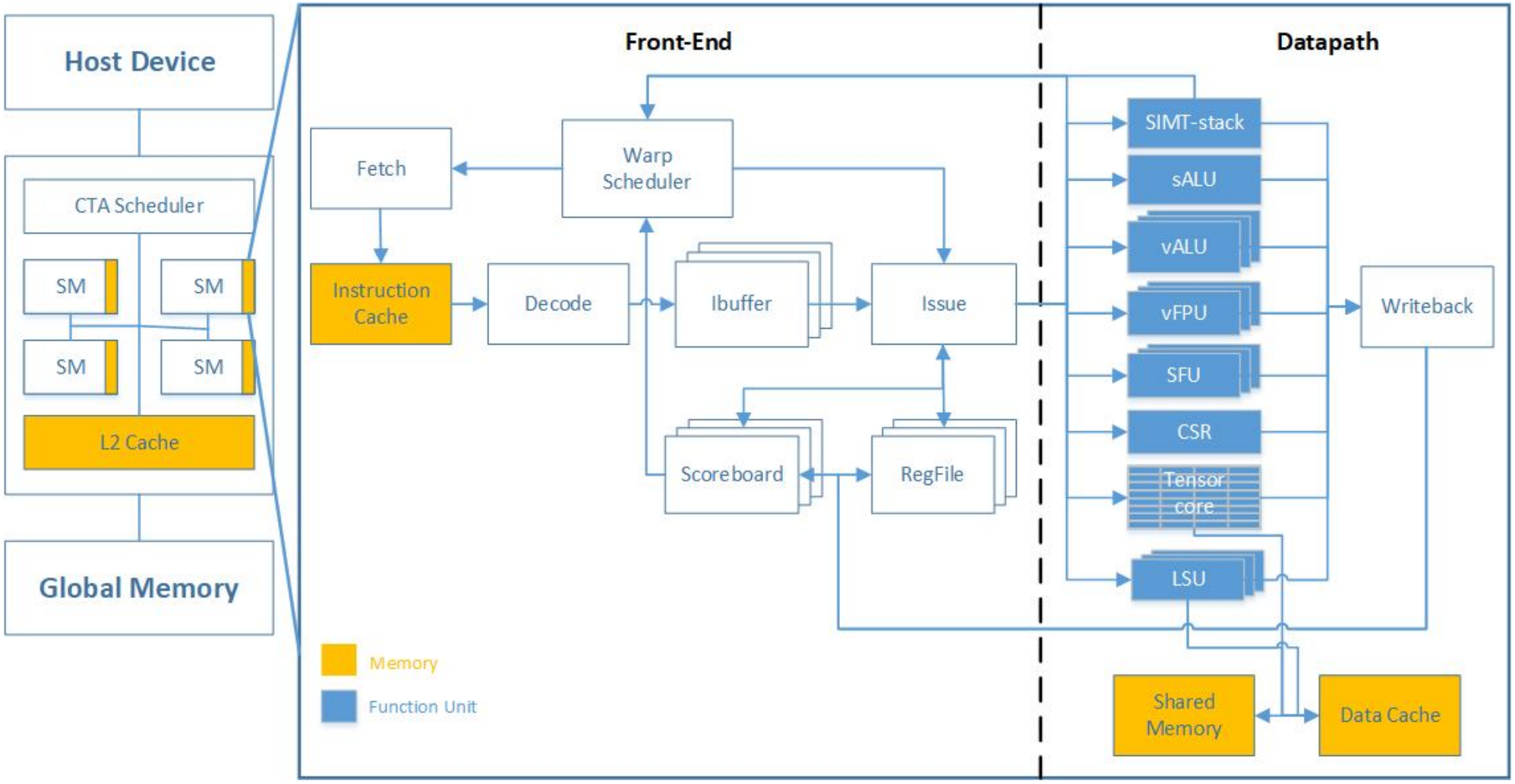
“乘影” GPGPU微架构

- 任务分配：Host - CTA Scheduler - SM(streaming multiprocessor)
- **任务执行：**
  - 每个SM可视为一个支持多warp调度的RISC-V向量处理器
  - 每个warp可视为一段RVV程序，经由取指、译码、发射，执行后写回寄存器
  - warp切换是类似hyper-threading的周期级调度



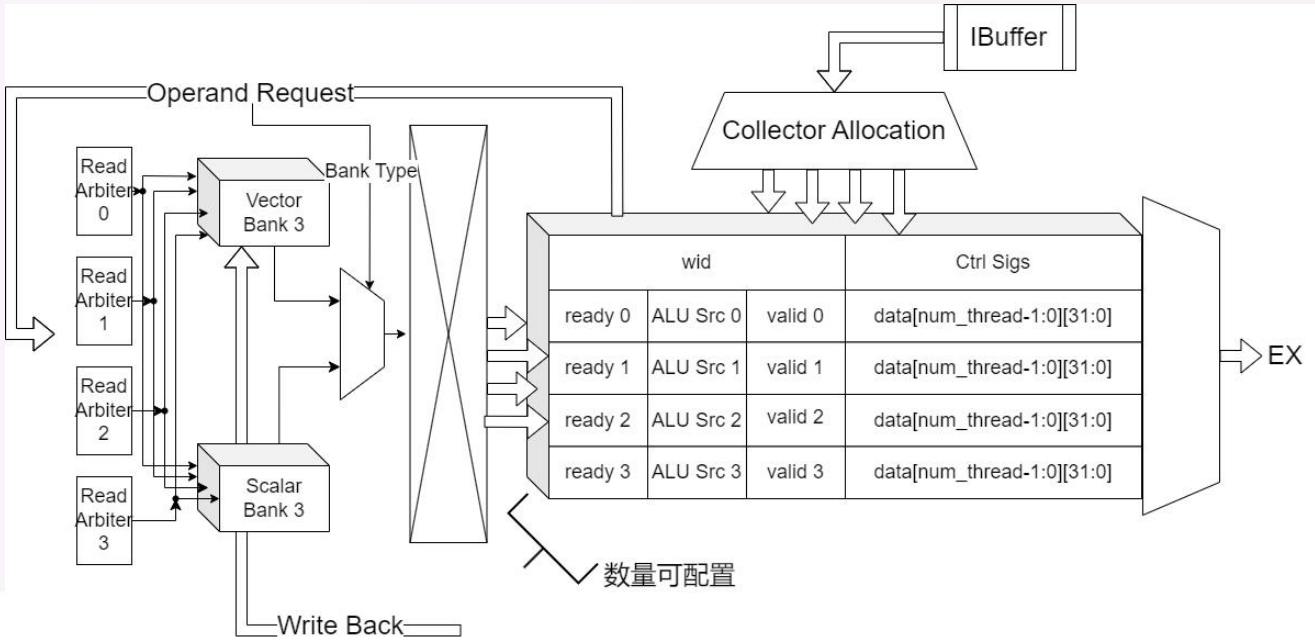


“乘影” GPGPU微架构


Host Device
CTA Scheduler
SM
SM
SM
SM
L2 Cache
Global Memory
Front-End
Datapath
Fetch
Warp Scheduler
Instruction Cache
Decode
Ibuffer
Issue
Scoreboard
RegFile
SIMT-stack
sALU
vALU
vFPU
SFU
CSR
Tensor core
LSU
Writeback
Shared Memory
Data Cache
Memory
Function Unit

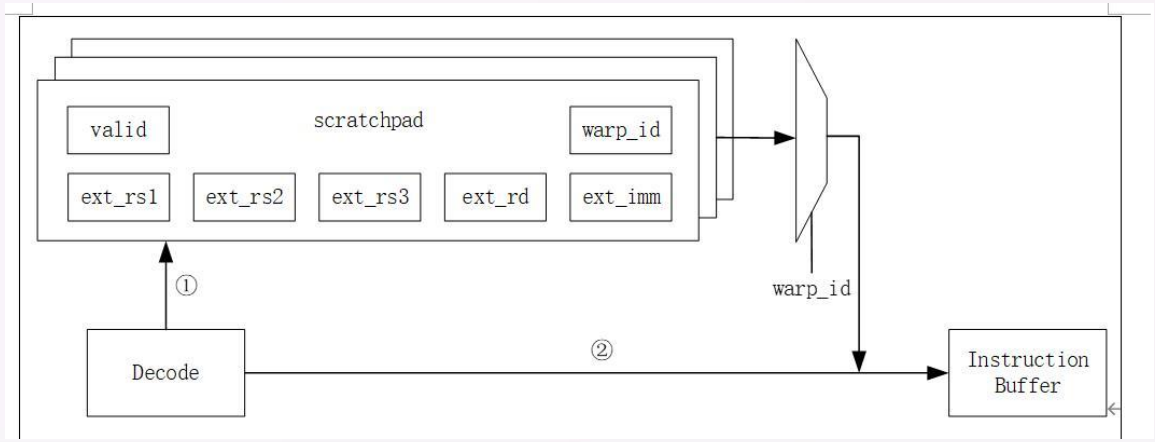
“乘影” GPGPU微架构 - 寄存器堆

- 寄存器堆为4-bank SRAM，1r1w， interleave（交织），且为unified结构（可根据warp的实际使用数目进行分配）
- 每个warp至多有
  - 256个 num_thread * 32bit vGPR
  - 64个 32bit sGPR
  - CSR



“乘影” GPGPU微架构 – 寄存器扩展处理

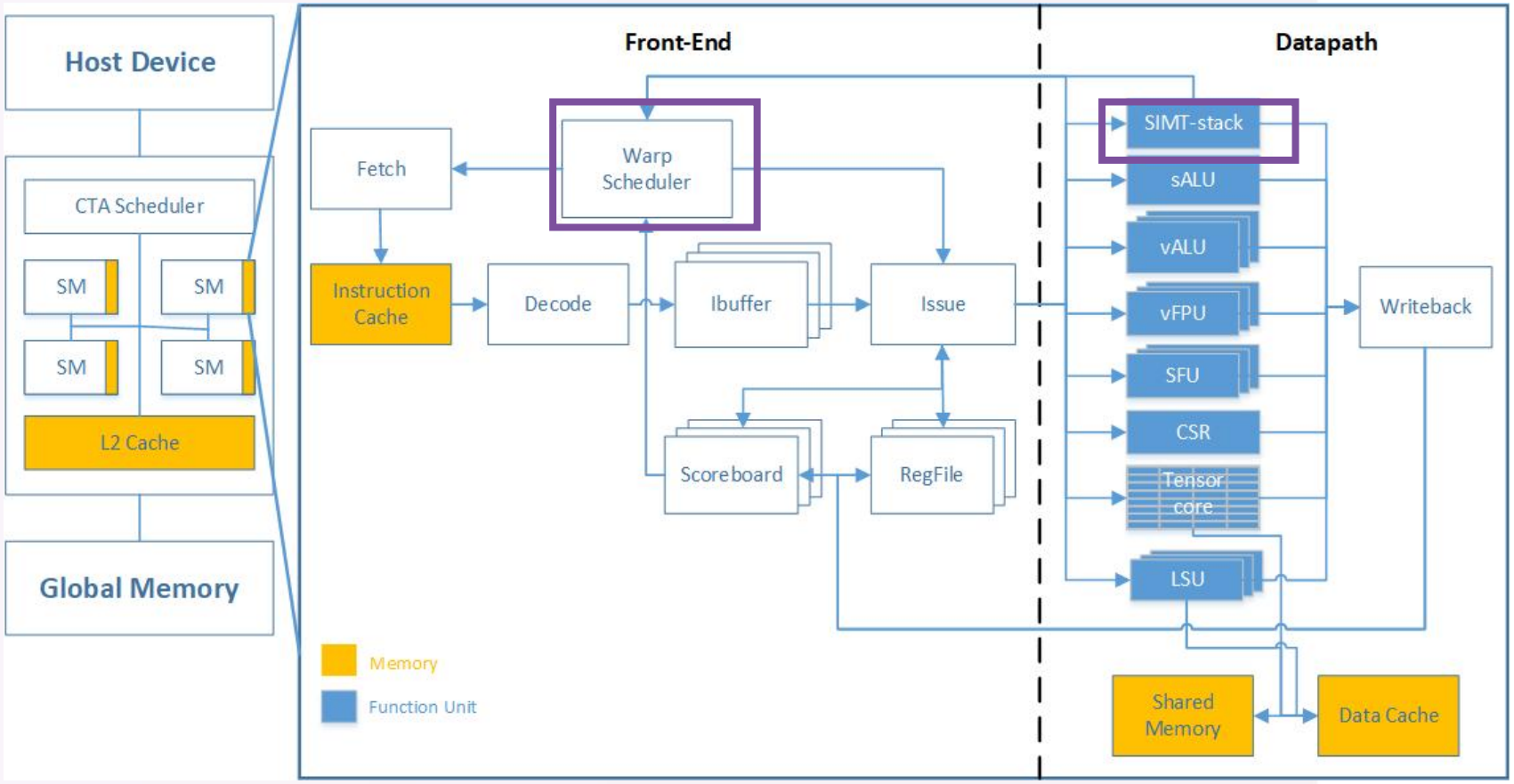
- 寄存器位宽扩展指令
  - 译码时若为寄存器扩展指令，其包含的寄存器扩展信息将暂存在译码级中
  - 接续的有效指令进行译码时，取出扩展信息合并处理
- 实现效果
  - regext指令：扩展四个5位寄存器编码至8位
  - regexti指令：扩展两个5位寄存器编码至8位，扩展5位立即数至11位



“乘影” GPGPU微架构 - 分支处理

SIMT-stack硬件管理

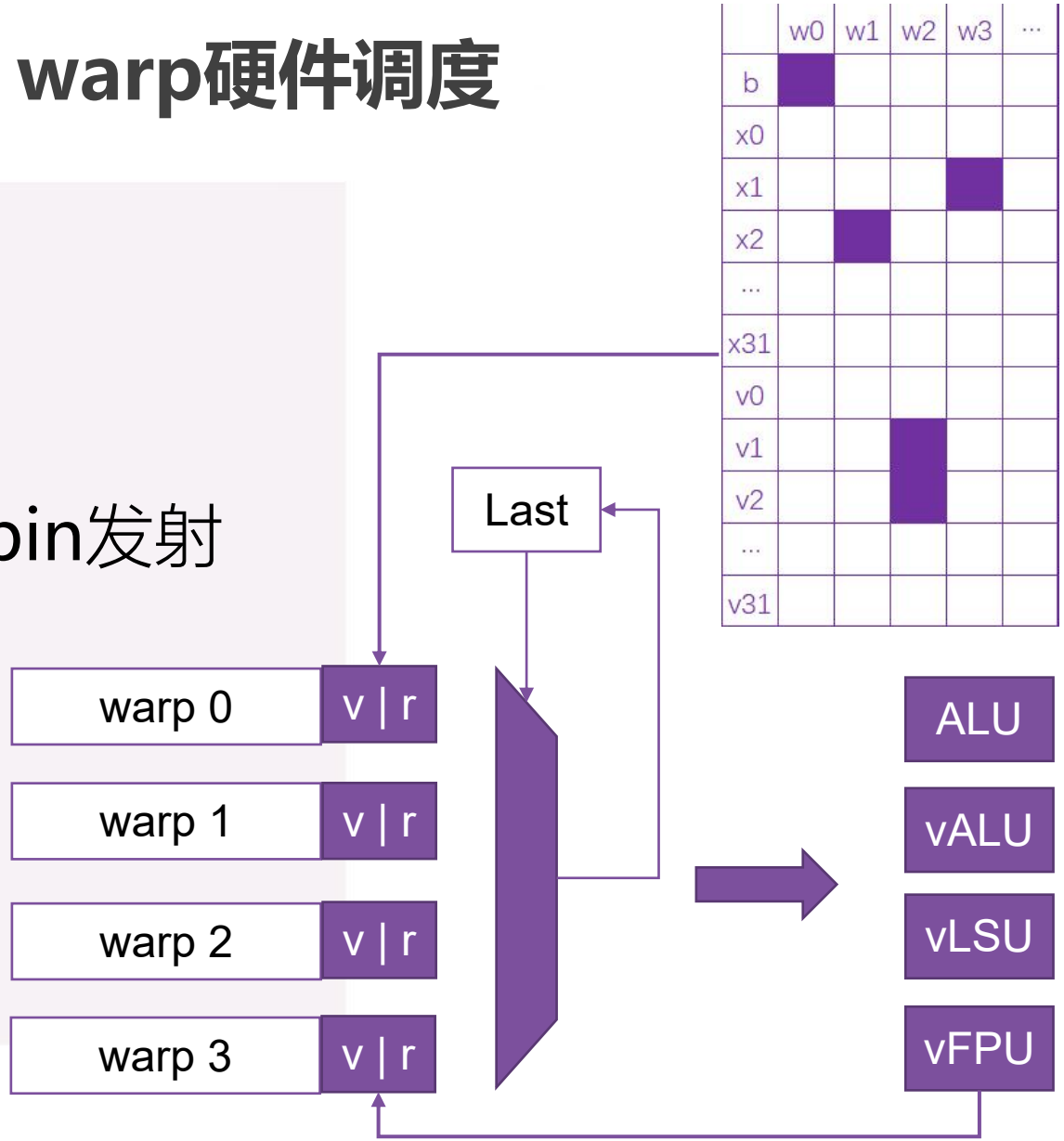
- 不占用通用寄存器，减少流水线停顿，嵌套分支和循环也能快速处理
- 仅用四条指令完成一次完整分支-合并操作



“乘影” GPGPU微架构 - warp硬件调度

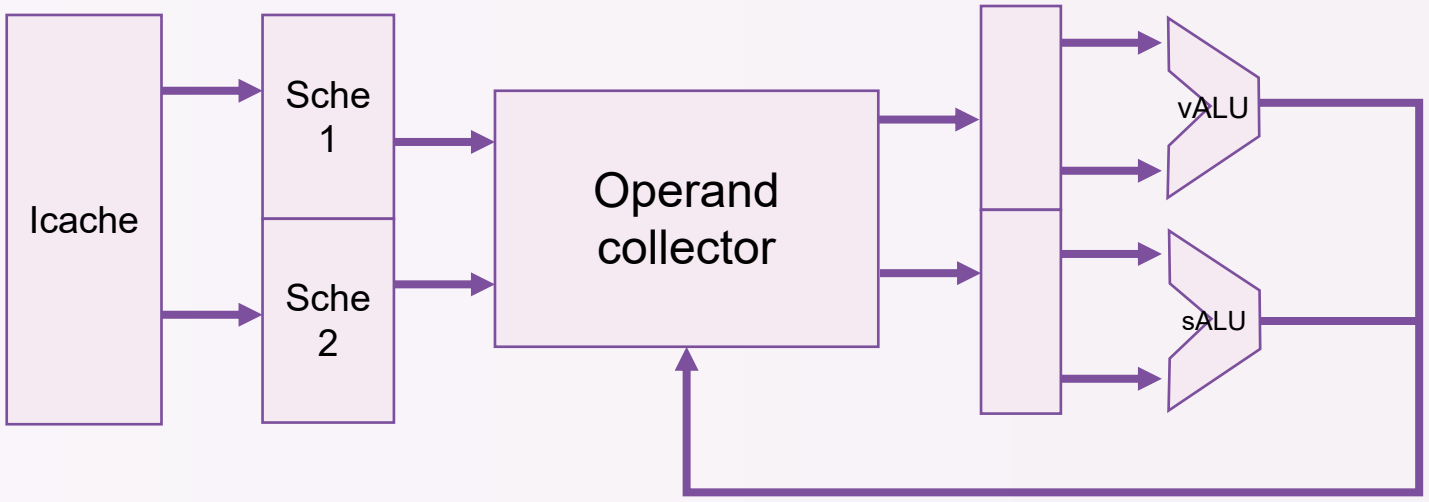
硬件调度发射执行

- ibuffer暂存warp信息
- Scoreboard记录依赖
- 每周期切换，Round-Robin发射



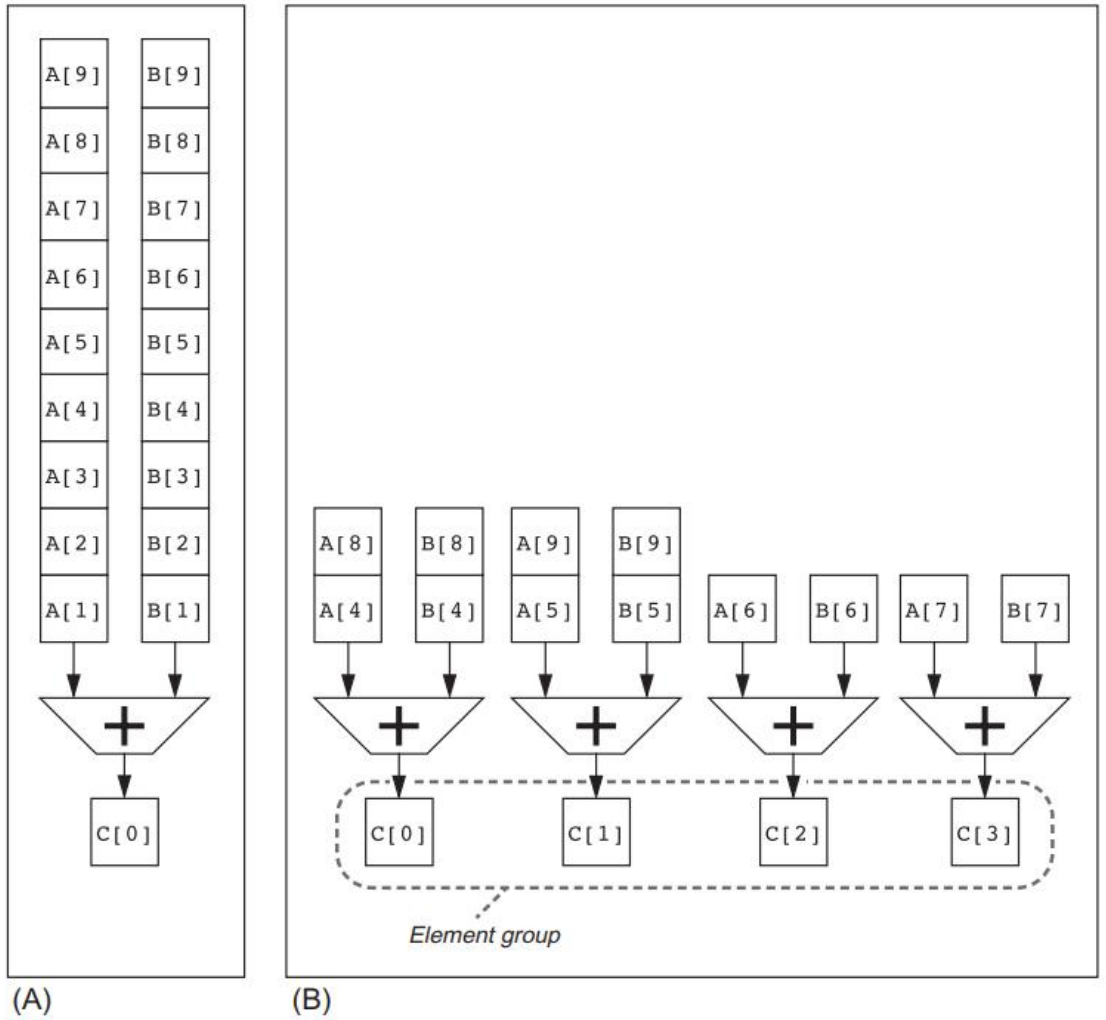
“乘影” GPGPU微架构 – 双发射

- 增加额外的线程束调度器，进一步发掘并行度
  - 不同线程束的标量指令和向量指令之间并行调度，理想情况下每周期发射一条标量指令和一条向量指令
  - 在保持低硬件复杂度的情况下提高整体性能



“乘影” GPGPU微架构 – 执行单元

- 功能单元数目可配置
- 目前vALU、vFPU、vSFU、vMUL单元均为可折叠、全流水配置，可通过num_lane参数配置硬件单元数目，结合num_thread可自由控制每类向量指令的执行cycle数。
- 典型latency为：alu 1cycle, mul 2cycle, fmul 3cycle, fadd 3cycle, fmacc 5cycle



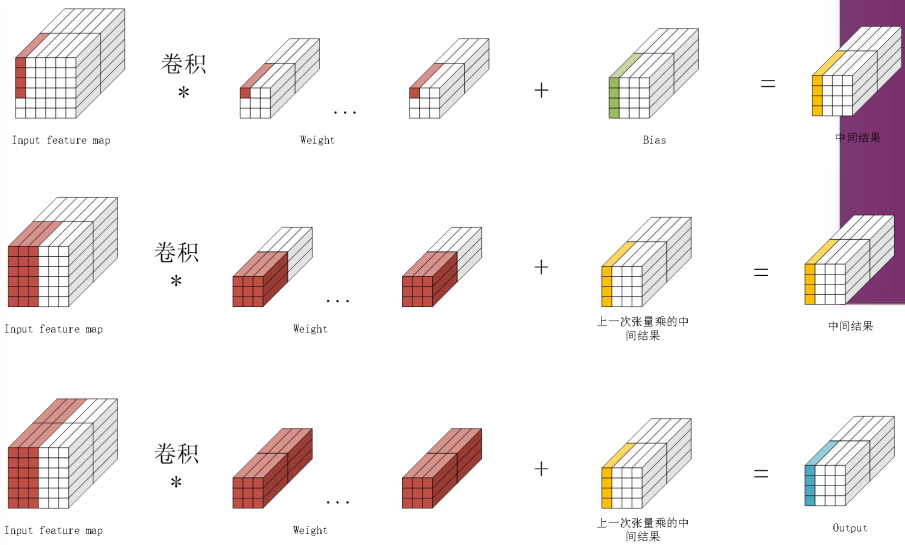
“乘影” GPGPU微架构 – Transformer支持

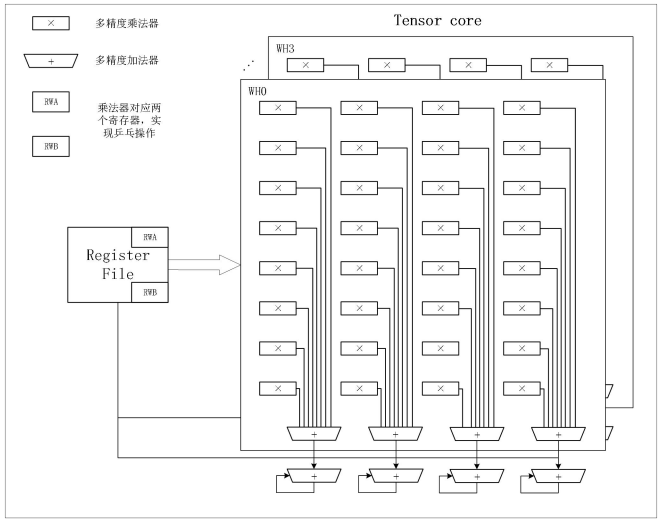
- 硬件实现张量计算TensorCore和exp函数，由此可实现对transformer运算的支持
- TensorCore可用于卷积和矩阵运算，支持4*8*4 fp32的运算
- 借助自定义指令使用通用数据通路完成
- 底层硬件通过fpu阵列实现矩阵乘加，作为功能单元接入流水线

Op1

Op2

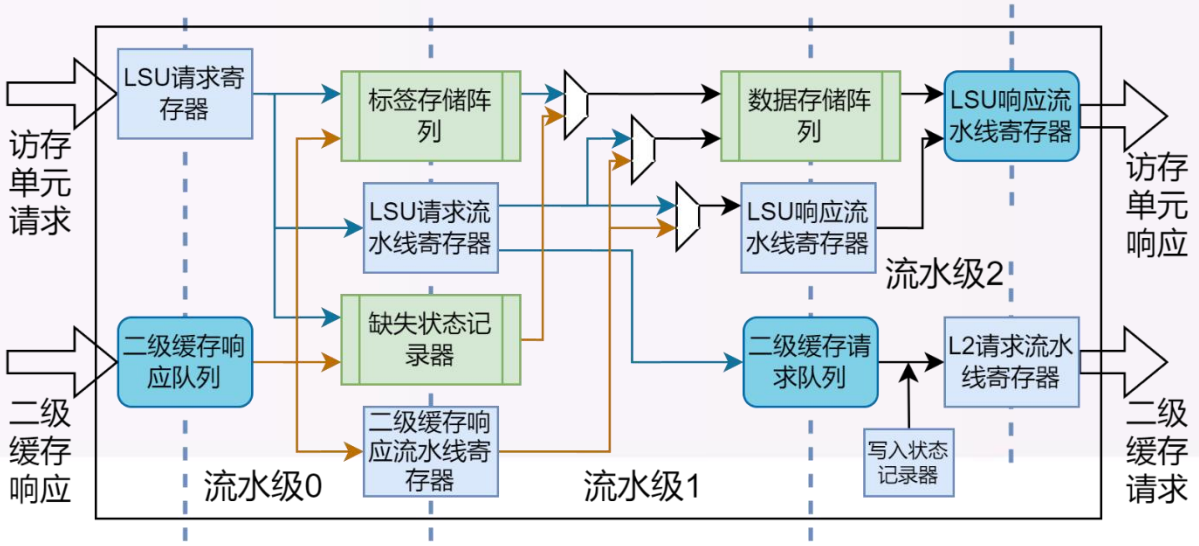
Op3

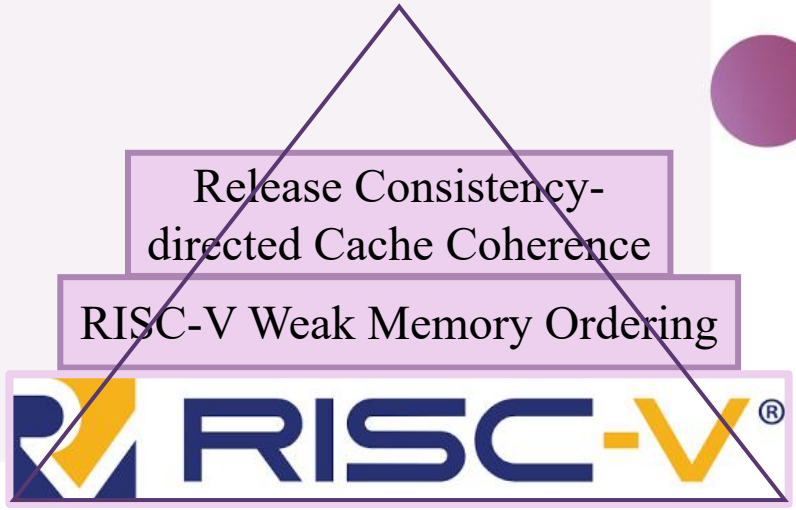




“乘影”GPGPU微架构 – 新版缓存设计

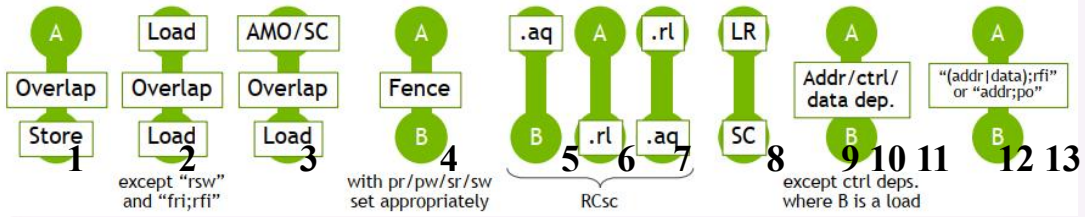
- 乱序机制（指令级并行/ILP）和多级缓存对性能至关重要，但会对线程间同步（线程级并行/TLP）产生阻碍
  - 连贯性问题：流水线中的乱序因素
  - 一致性问题：缓存内容的不同步
- 将RISC-V宽松连贯性模型RVWMO映射到GPGPU，并实现连贯性指导的缓存一致性（RCC）
  - 相比CPU硬件一致性，降低L1-L2带宽开销和硬件复杂度
  - 相比GPU软件一致性，减少编程框架的额外负担





“乘影”GPGPU微架构 – 新版缓存设计

- RVWMO通过显式指令控制严格的数据同步行为
  - RV32I的FENCE指令
  - RV32A AMO原子指令、LR/SC指令的acquire和release标识符



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

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 LR SC

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
LR.W/D	ordering		0	addr	width	dest	AMO
SC.W/D	ordering		src	addr	width	dest	AMO

- 缓存微架构设计
  - 使用缺失状态记录器（MSHR）、写入状态记录器（WSHR）解决PPO1、2
  - PPO4~7对应连贯性操作Fence和acquire/release，通过清空MSHR、全局冲刷、全局无效化等微架构操作予以支持
  - PPO 9~13在流水线中已经解决，2和8在缓存流水线中插入额外检查解决

“乘影” GPGPU指令集 – 新版缓存设计

- 实现：对acquire、release添加额外的冲刷（flush）和无效化（invalidate）操作规定
  - “释放”：release发生时，L1所有脏缓存行写回L2（即flush*），然后release对应的写入再写回L2；
  - “获取”：acquire发生时，无论是否hit，直接从L2读取，然后无效化cache里该line之外的所有line（即invalidate）。

微架构操作	④全局无效化	③全局冲刷	④全局无效化	①清空MSHR	④全局无效化	③全局冲刷
RVWMO连贯性语义	.aq标识符	.rl标识符	FENCE R,R	FENCE R,W	FENCE W,R	FENCE W,W
附带的RCC一致性操作	“获取”和“释放”	“释放”	“获取”和“释放”	无	“获取”和“释放”	“释放”

- 不同地址空间的数据采用不同缓存策略

	private mem	global mem
写命中	write-back	write-through
写缺失	write non-allocate	write non-allocate

THANK YOU

OpenGPGPU
乘影

上海清华国际创新中心
集成电路研究平台
International Innovation Center of Tsinghua University Shanghai
Integrated Circuit Research Platform