A Micro Arch Design of L1 Cache for GPGPUs Supporting Release Consistency-directed Coherence Based on RVWMO

Jin Chufeng [1,2], Yang Kexiang [1,2], Li Jingzhou [1,2] and He Hu [1,2]
1 School of Integrated Circuits, Tsinghua University 2 International Innovation Center of Tsinghua University, Shanghai

Background & Novelty

On-chip memory pursues many desirable characteristics such as high throughput, low latency, and low cost. However, without **functional correctness** as a prerequisite, these characteristics are merely illusory.
This work:
1. the 1st open-source microarchitectural solution for GPGPU cache that take **consistency and coherence** into design consideration.
2. Adapts the RISC-V CPU-centric consistency model **RVWMO** to GPGPU.
3. Leverage **Release Consistency-directed Coherence(RCC)** to provide coherence functionality, reducing both the burden on programming frameworks and the complexity of hardware implementation.

Release Consistency-directed Coherence

RISC-V Weak Memory Ordering

RISC-V®

Intro to Memory Consistency & GPU Cache Coherence

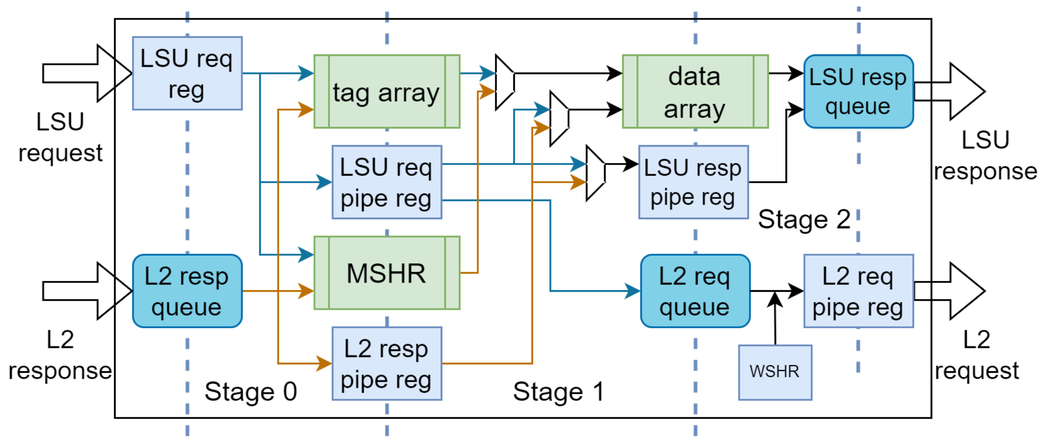
In multi-core/multi-thread scenarios, the consistency problem can arise from out-of-order (OoO) execution. Different consistency models with varying degrees of strictness can lead to different valid execution results for the same multi-threaded program.

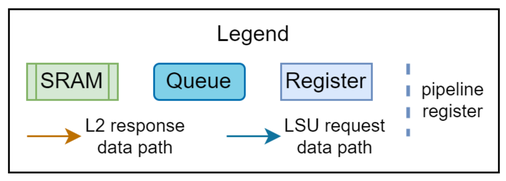
```
/*A, B, Flag with init value 0
assume 0 and 1 share memory directly*/
//HW thread 0   //HW thread 1
A = 1;          while(Flag==0){};//spin
B = 1;          printf("A=%d\n",A);
Flag = 1;       printf("B=%d\n",B);
```

	Sequential Consistency	Relaxed Consistency
Legal result	A=1 B=1	A=1 B=1
		A=1 B=0
		A=0 B=1
		A=0 B=0

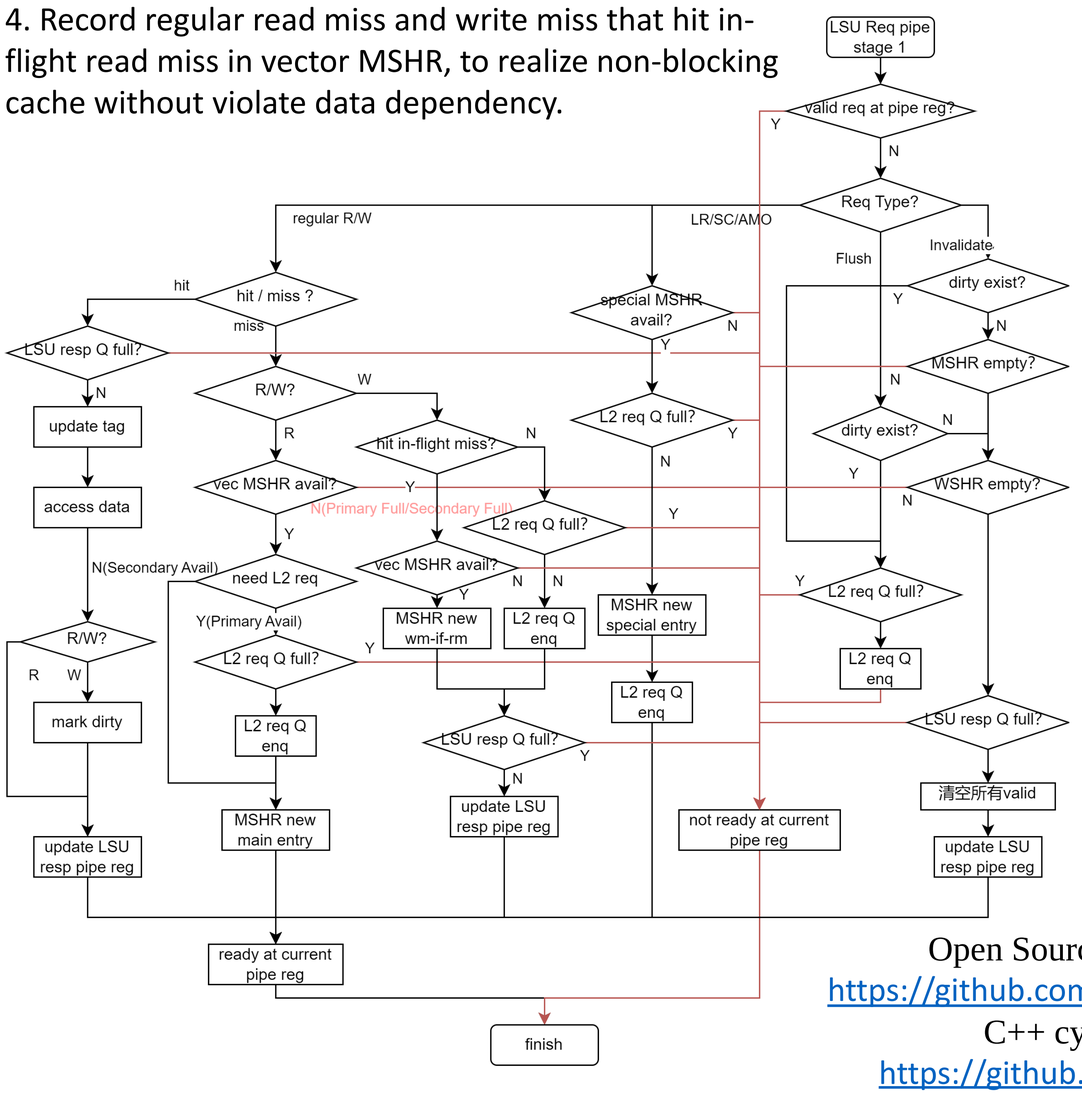
GPU caches typically do not implement hardware coherence due to the significant area cost and bandwidth consumption. However, this does not preclude the need for synchronization between private caches, which can be accomplished using explicit cache operations such as flush or invalidate.

GPGPU L1 Cache Design & Architecture



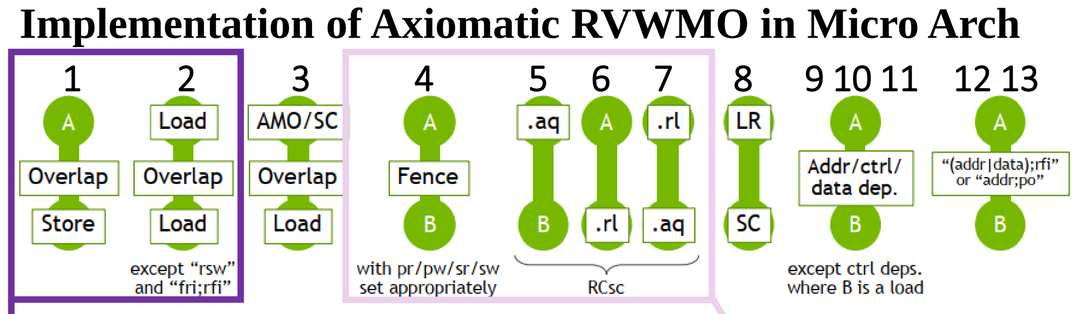


1. Vector cache receive coalesced request. LSU split uncoalesced instructions into multiple coalesced cache request.
2. Write-back and write-around.
3. Forward AMO/LR/SC to L2, record in-flight request in special MSHR.
4. Record regular read miss and write miss that hit in-flight read miss in vector MSHR, to realize non-blocking cache without violate data dependency.



Open Source with Ventus GPGPU
https://github.com/THU-DSP-LAB/ventus-gpgpu
C++ cycle accurate model
https://github.com/Auyuir/cacheCmodel

Implementation of Axiomatic RVWMO in Micro Arch



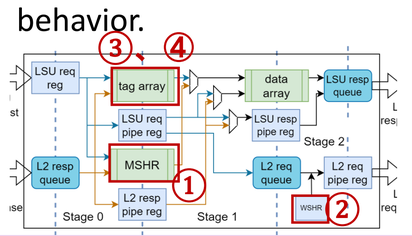
PPO Rule 1-2: For hardware operational model without auxiliary consistency operations, use Write Status Holding Register(WSHR) to check and prevent regular memory operations from violating PPO 1-2 or Load Value Axiom.

*W for Write, R for Read; iO for in-ordered, OoO for out-of-order; H for cache hit, M for cache miss.

	Access the same cache line				Access different cache lines			
Cases	R-R	W-W	W-R	R-W	R-R	W-W	W-R	R-W
RVWMO PPO	Rule 2	Rule 1	No	Rule 1	No	No	No	No
HW behavior	iO	iO	OoO	iO	OoO	OoO	OoO	OoO
WSHR guard	No	Yes	Yes	Yes				
OoO/guard cases		Except H-H	M/H-M	M-M	M-M/H	H-M/H	H-M	M-M/H

Micro arch behavior	Description	Effect ("F" for FENCE) (N for Necessary, S for Sufficient)
① Drain MSHR	Wait L2 response for all regular read misses, LR, SC and AMO	N but not S for F R,R S for F R,W
② Drain WSHR	Wait L2 response for all write	N but not S for F W,R N but not S for F W,W
③ Global Flush	(include a subsequent ②) write all dirty cache line to L2	N but not S for F W,R S for F W,W
④ Global Invalidate	(include a preceding ③ and ①) invalidate all cache line	S for all FENCE, include F R,R; F W,R

PPO Rule 4-7: For RVWMO consistency operations(RV32I FENCE, RV32A .aq & .rl identifier), mapping them to corresponding microarchitectural behavior.



Implementation of RCC in Micro Arch

Release Consistency provides "Acquire" or "Release" semantics to indicate its constraint. In Release Consistency-directed Coherence, the combination of "Release" with dirty cache line flush and "Acquire" with L1 global invalidation satisfies the coherence requirement.

Micro arch behavior	④Global Invalidate	③Global Flush	④Global Invalidate	①Drain MSHR	④Global Invalidate	③Global Flush
RVWMO Consistency Semantics	.aq identifier	.rl identifier	FENCE R,R	FENCE R,W	FENCE W,R	FENCE W,W
Attached RCC Coherence Function(s)	"Acquire" and "Release"	"Release"	"Acquire" and "Release"	None	"Acquire" and "Release"	"Release"

Comparison & Result

feature	MICRO22 Vortex	HotChips15 MIAOW	HotChips10 NV Fermi	HotChips17 NV Volta	HotChips12 AMD GCN	This work
Vector access	√	√	√	√	√	√
Non-blocking cache	√	No RTL micro arch	√	√	√	√
Atomic instruction			√	√	√	√
Release consistency				√	√	√
Invalidate			√	√	√	√
Flush	√	√	√	√	√	√

In the table below, the number on the left of each column indicates the typical delay cycles for the corresponding request type when there is no blocking in the corresponding signal path. The numbers on the right of each column indicate potential blocking scenarios.

req type / Data path	Regular read miss		Regular read hit		Regular write miss		Regular write hit		LR/SC/AMO		Wait MSHR		Flush		Invalidate	
c Req – c Rsp			4	a	4	ab	4	a			4	ae	4	a h-b	4	a h-b
c Req – m Req	4	bdf			4	bcd			4	bdfgh			4	h-bcd	4	h-bcd
c Req – data							3									
m Rsp – c Rsp	4	aij							4	a						
m Rsp – m Req	4	j-bcd														
m Rsp - data	3															

blocking scenarios for LSU req and L2 resp		blocking scenarios only for LSU req		blocking scenarios only for L2 resp	
LSU resp Q full	a	MSHR not empty	e	MSHR subentry ≥ 2	i
L2 req Q full	b	MSHR full	f	Replacement occur	j
Write status holding register full	c	LR – SC	g		
WSHR protection	d	Dirty cache line exist	h		

Synthesis using SMIC 40nm process and specialized SRAM. worst PVT variations.

	Max frequency	Area
SRAM	420MHz	736,150.7$\mu m2$
Others	320MHz	538,390.2$\mu m2$
Total	320MHz	1319,901.1$\mu m2$