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

Abstract

Coherence and consistency are critical factors in multi-core processors and parallel programming models. In this paper, we present an RTL implementation of an L1 vector cache that supports Release Consistency-directed Coherence (RCC) based on RVWMO for open-source GPGPU, the RISC-V ISA provided consistency model. We propose a methodology for transforming axiomatic rules into hardware design guidance. Our design aims to achieve a good performance-cost trade-off for GPGPU cache design. We are currently conducting litmus tests and performance evaluations to further validate our proposed design.

Introduction

Coherence protocols such as MESI are commonly used in multi-core CPU processors. However, creating coherence protocols for General Purpose Graphics Processing Units (GPGPUs) can be difficult due to the large number of private L1 caches and the bottleneck in L1-L2 bandwidth. The Release Consistency-directed Coherence (RCC)[1] addresses these challenges by performing coherence operations alongside consistency operations. This eliminates the need for costly hardware controllers and reduces the bandwidth required for status transitions.

The open-source ISA RISC-V provides a well-defined CPU-oriented consistency model called RVWMO[2], which is a variant of release consistency that is suitable for implementing RCC. However, this model is primarily described in an axiomatic manner, which poses a challenge in guiding hardware implementation.

In this work, we address the gap between axiomatic RVWMO rules and microarchitecture design guidance for GPGPUs by proposing an open-source RTL implementation of an L1 cache that supports RCC. Our proposed architecture adapts the CPU-centric RVWMO to the GPGPU cache and leverage RCC to provide coherence functionality.

GPGPU L1 Cache Design & Architecture

The proposed L1 cache is a component of the open-source GPGPU named “Ventus”. Within the GPGPU, each streaming multiprocessor has a dedicated L1 memory subsystem consisting of an instruction cache, scratchpad memory, vector data cache, and constant cache. All SMs share a common L2 memory subsystem (L2). This section takes L1 vector data cache as an example.

To fully leverage the benefits of the RCC, the "write back" policy has been adopted to minimize the write bandwidth consumption between L1 and L2. Similarly, the "write non-allocate" policy has been set to avoid fetching missing cache lines when a write miss occurs.

To support the large parallelism requirement of GPGPUs, we employ a non-blocking cache, which allows for multiple concurrent outstanding misses. This functionality is facilitated by a module called the Missing Status Holding Register (MSHR), which manages regular read misses, load reserved, store conditional, and atomic operations that are sent to the L2 cache. The MSHR is also responsible for recovering the LSU response when the corresponding L2 request returns. Furthermore, the MSHR is capable of merging upcoming read misses access the same cache line.

To enable data SRAM vector element access in "Ventus", vector memory access instructions are checked and split into coalescing cache requests in the LSU. Coalescing requests only access elements that fall into a single cache line.

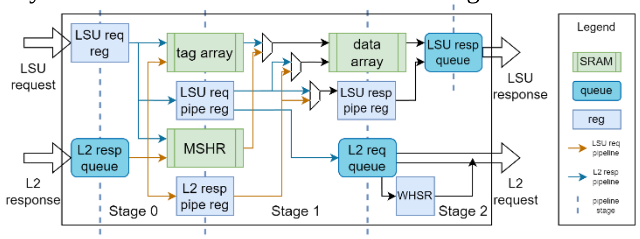

Figure 1 Simplified cache timing behavior

As Figure 1 shows, The SRAM access cannot be completed within a single clock cycle. This characteristic dominant the pipeline design of cache microarchitecture. Here is an abstract for LSU request pipeline:

- Stage0: Send probe to tag array and MSHR;
- Stage1: According to request type and probe result: new entry in L2 request queue and MSHR, access data array;
- Stage2: New entry in LSU response queue from data array.

RCC Design Methodology under RVWMO

Release Consistency provides “Acquire” or “Release” semantics to indicate its constraint. In Release Consistency-directed Coherence[2], the combination of “Release” with dirty cache line flush and “Acquire” with L1 global invalidation satisfies the coherence requirement.

This section will begin by breaking down RVWMO rules into a more hardware-friendly form. Next, we will scrutinize and refine the hardware design to ensure RVWMO compliance in the absence of auxiliary consistency operations. Finally, we will concentrate on translating these consistency operations (combined with coherence operations) into microarchitectural behaviors to complete the RCC cache design under RVWMO.

Breaking Down Axiomatic Consistency Rules

RVWMO defines three axioms[1], one of which is the PPO Axiom consisting of 13 specific PPO rules. In the "Ventus" architecture, PPO rules 9-13 are resolved by the front-end scoreboard before instructions reach the LSU of execution stage.

Consistency control operations in RVWMO include the FENCE instruction in RV32I, as well as the .aq (Acquire) and .rl (Release) qualifiers in RV32A[1], which together form PPO rules 4-8.

FENCE R,RW has close semantics to .aq qualifier, but stronger. FENCE RW,W and .rl have a similar relationship. Therefore, the use of successive FENCE R,RW can serve as a straightforward implementation of .aq, thereby the use of preceding FENCE RW,W for release semantics.

Without taking device I/O into account, it is possible to derive all variants of FENCE from 4 fundamental types: FENCE R,R ; FENCE R,W ; FENCE W,R ; FENCE W,W.

Hardware Operational Model without Auxiliary Consistency Operations

In the design process of cache microarchitecture, the prioritization of memory fetch functionality over the ordering of return values is paramount, particularly when selecting a relaxed model that imposes minimal constraints on memory orders. Consequently, the hardware framework must first be established before embarking on a consistency-conformant design phase. During this phase, meticulous attention is devoted to conforming with the regular memory access orders, in accordance with PPO rule 1-2.

Table 1 Hardware operational model, regular W&R

*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

Operational and axiomatic models differ in their focus: the former specifies "which actions can" be performed, while the latter specifies "which actions cannot" be performed. Table 1 shows the operational model of our proposed cache design. A module called Write Status Holding Register (WSHR) is used to prevent erroneous reordering of L2 access when accessing the same cache line, thereby enforcing both the PPO rule 1-2 and the Load Value Axiom.

Mapping Consistency Operations to Microarchitectural Behaviors

To support the 4 fundamental variants of FENCE for enforcing PPO rule 4-7, as well as to provide coherence operations for RCC, it is necessary to leverage all the microarchitectural operations that can be provided:

- MSHR drain: Wait all regular miss, LR, SC and AMO return from L2. Necessary for FENCE R,R, FENCE R,W;
- WSHR drain: Wait all write return from L2. Necessary for FENCE W,R, FENCE W,W;
- Flush: Write back all dirty lines, contains a precede WSHR drain. Necessary for FENCE W,R, FENCE W,W;
- Global Invalidation: invalidate all lines, contains precede flush and MSHR drain. Sufficient for all FENCE type.

Table 2 presents the mapping from microarchitectural operations to FENCE variants to implement PPO rule 4-7. After this step we have successfully deployed PPO rule 1,2,4-7,9-13, Load Value Axiom, and RCC. The remain PPO rule 3 and 8 can be enforced by additional L2 request checking.

Table 2 Final mapping between HW and RVWMO

RVWMO operation	.aq	.rl	FENCE R,R	FENCE W,W	FENCE W,R	FENCE R,W
HW	Invalidation	Flush	Invalidation	Flush	Invalidation	MSHR drain

Evaluation

Our proposed design is simulated using a C++ cycle-accurate model (https://github.com/Auyuir/cacheCmodel) and implemented using the Chisel HDL (https://github.com/THU-DSP-LAB/ventus-gpgpu). When synthesized using SMIC40 process and specialized SRAM, our proposed design(configurated as 64KB) achieves a frequency of 370MHz and an area of 1319901.1 μm^2, of which 736150.7 μm^2 are SRAM(worst PVT variation).

Validating that the implementation satisfies the promised consistency model is a challenging task, ranging from formal proofs to litmus tests. The derivation process in last section is also a type of manual checking process. To further ensure the rigor of our design, we are also subjecting it to the RVWMO litmus test set.

Conclusion

We present a microarchitecture design of an L1 vector cache for open-source GPGPU "Ventus" that supports Relaxed Consistency-directed Coherence based on the RVWMO. Additionally, we provide a systematic approach for decomposing the axiomatic rules of RVWMO into hardware design specifications for RCC.

References

[1] Nagarajan, V., Sorin, D. J., et al. A primer on memory consistency and cache coherence. 2nd. Morgan & Claypool Publishers. 2020

[2] Andrew W., Krste A.. The RISC-V Instruction Set Manual, Volume I: User-Level ISA. V20191213. RISC-V Foundation. December 2019