清華大學
Tsinghua University

GPU中RV赋能的宽松存储连贯性指导的缓存一致性

2022.08.23

报告人：THU DSPLAB 金楚丰

目录

存储连贯性模型：乱序多核系统之基石

Memory Consistency Model

- 乱序+多核在处理器会发生什么？

```
/*A, B, Flag初值均为0*/

//HW thread 0
A = 1;
B = 1;
Flag = 1;

//HW thread 1
while(Flag==0){}//自旋
printf("A=%d\n",A);
printf("B=%d\n",B);
```

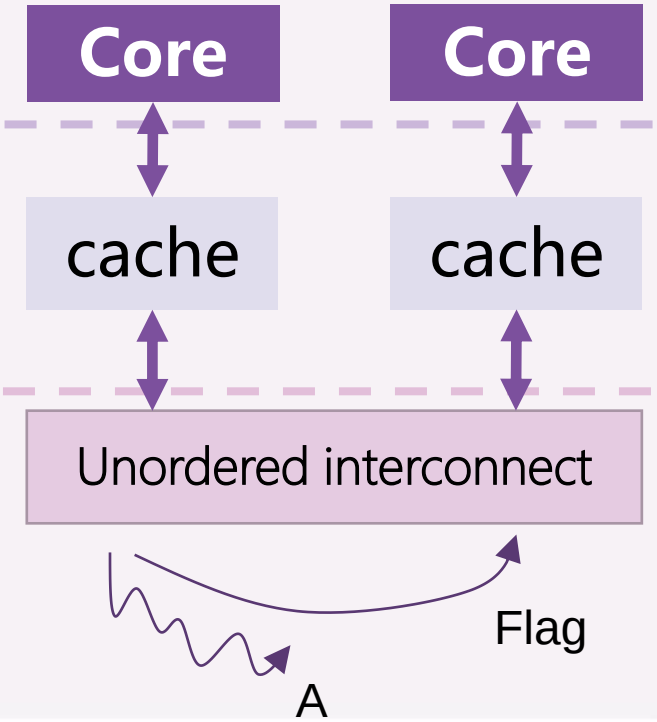
- 直觉上，A和B都会打印出1?
- 实际中，线程0数据移动到线程1的过程做不出这种保证。

存储连贯性模型：乱序多核系统之基石

Memory Consistency Model

```
//HW thread 0//HW thread 1
A = 1;          while(Flag==0){}//自旋
B = 1;          printf("A=%d\n",A);
Flag = 1;       printf("B=%d\n",B);
```

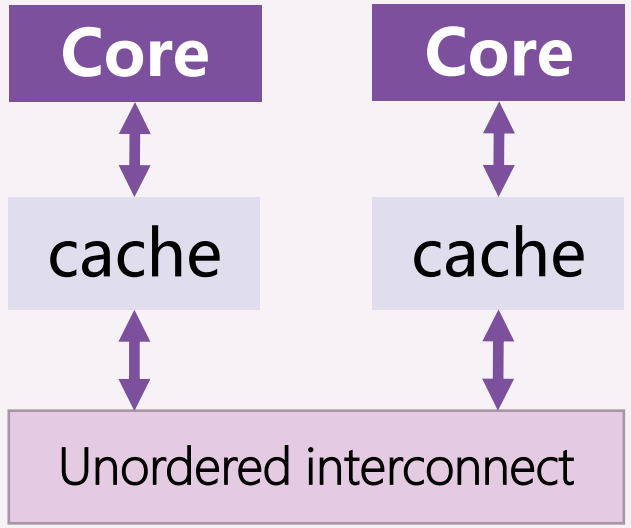
- 编译器指令重排序(reordering)
- 处理器流水线乱序执行(OoO)
- 非阻塞缓存 (non-blocking)
- 写缓冲器
- 无序总线



存储连贯性模型：乱序多核系统之基石

Memory Consistency Model

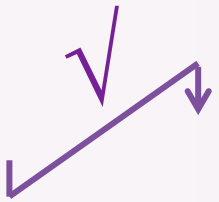
- 遍布整个计算系统的乱序通过借助ILP和MLP，极大地成就了今天的单核IPC表现。
- 在单核→多核挖掘TLP的演进中，宽松存储连贯性模型为每个工程师圆“鱼和熊掌兼得”梦。
- “既要**乱序**，也要**多核**”



产生约束 && 释放自由，宽松连贯性如何两者兼得?

Relaxed Consistency(RC)

```
/*A, B, Flag初值均为0*/
//HW thread 0                    //HW thread 1
A = 1;                           while(Flag==0){}//acquire
B = 1;                           printf("A=%d\n",A);
Flag = 1 ;//release              printf("B=%d\n",B);
```

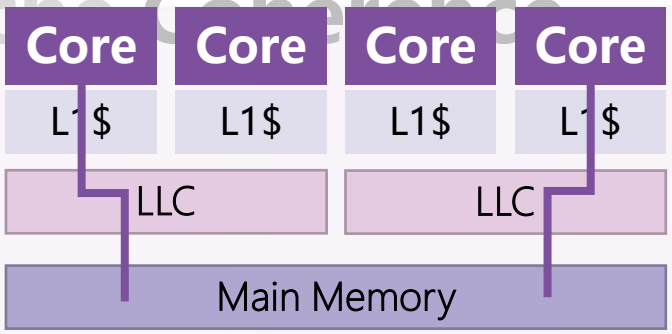


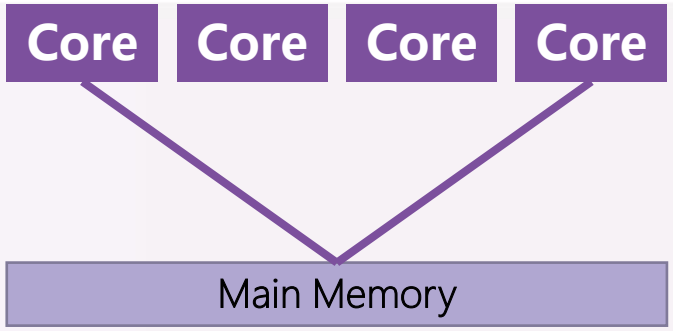
- 仍以该程序段为例，在关键区（critical section）附近通过显式指令（load.acquire、store.release）来约束，在非关键区释放乱序潜力。
  - acquire：**之后**的访存操作不会比该操作更**早**地发生**。
  - release：**之前**的访存操作不会比该操作更**晚**地发生**。

**发生：为便于理解提供了直觉解释。实际上是不准确的。准确表达为"被其他hart或观测者观测到"，即被其他hart的load到该写入值。

缓存一致性

Cache Coherence





- 为**存在多个间接访存者**的系统提供"**每个当事人直接访问共享内存**"的假象。
- 这种假象成为了CPU多核协同工作的基础。并且一般由**硬件**维护，对程序员/编译器几乎不可见。
- 在目前主流CPU设计范式中，连贯性机制和一致性协议正交。但连贯性往往也基于上述假象（比如定义全局内存顺序）展开论述。

GPU的缓存一致性

Cache Coherence of GPU

- 早期的GPU发展经历了和CPU相对独立的发展过程，GPU曾经的架构更像面向图形业务的DSA，其上层应用并不寻求频繁的线程间同步。
- 所以早期GPU放弃对硬件缓存一致性的支持。当应用确需同步时，使用开销偏大的软件同步手段来实现。但通用并行编程的应用场景逐渐开拓之后，高性能缓存一致性方案的重要性开始凸显[1]。
- 静态、规则的拓扑驱动型应用→动态、不规则的数据驱动型应用[2]

  比如　　图形渲染　　→　　图型、树形数据结构

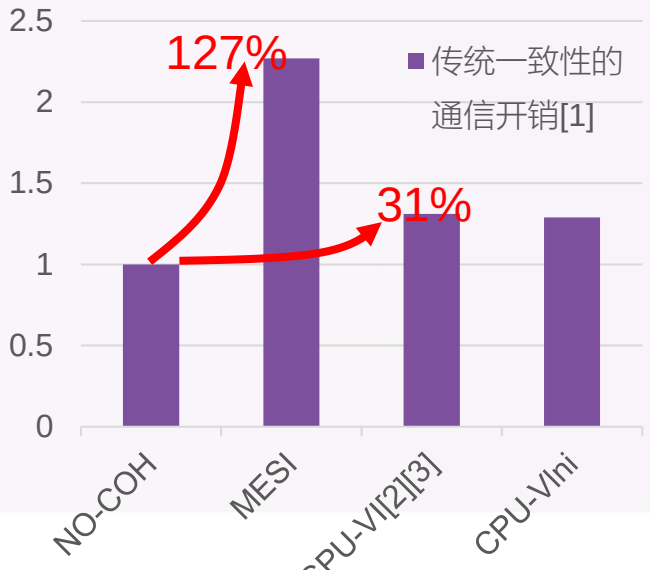
  - Barnes-Hut N体仿真问题
  - 最小生成树问题

Ref
[1] HPCA20 HMG: Extending Cache Coherence Protocols Across Modern Hierarchical Multi-GPU Systems, UBC, Nvidia
[2] MICRO14 Accelerating Irregular Algorithms on GPGPUs Using Fine-Grain Hardware Worklists, Cornell

GPU的缓存一致性

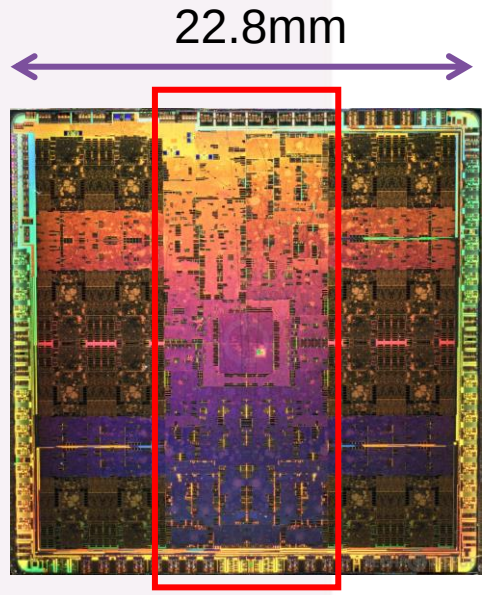
Cache Coherence of GPU

- CPU中广泛应用的基于Directory的MESI协议不适合于GPU众核场景。
  - MESI的总线通信开销——通信带宽已经是GPU的典型性能瓶颈
  - Directory的面积开销——直接和成本挂钩
- 并且随核数Scale Up的增长曲线都很差。



粗略估算：经典集中式Directory有N个entry，每个entry由n个bit组成。
其中，N和L1$的数量成线性关系，系数≥1
n也与L1$的数量成线性关系，系数≈1
所以Directory总容量**与L1$数量成平方关系**。

[1]中指出经典Directory在某商用GPU架构中的面积开销**比L2$的总面积大28%**。

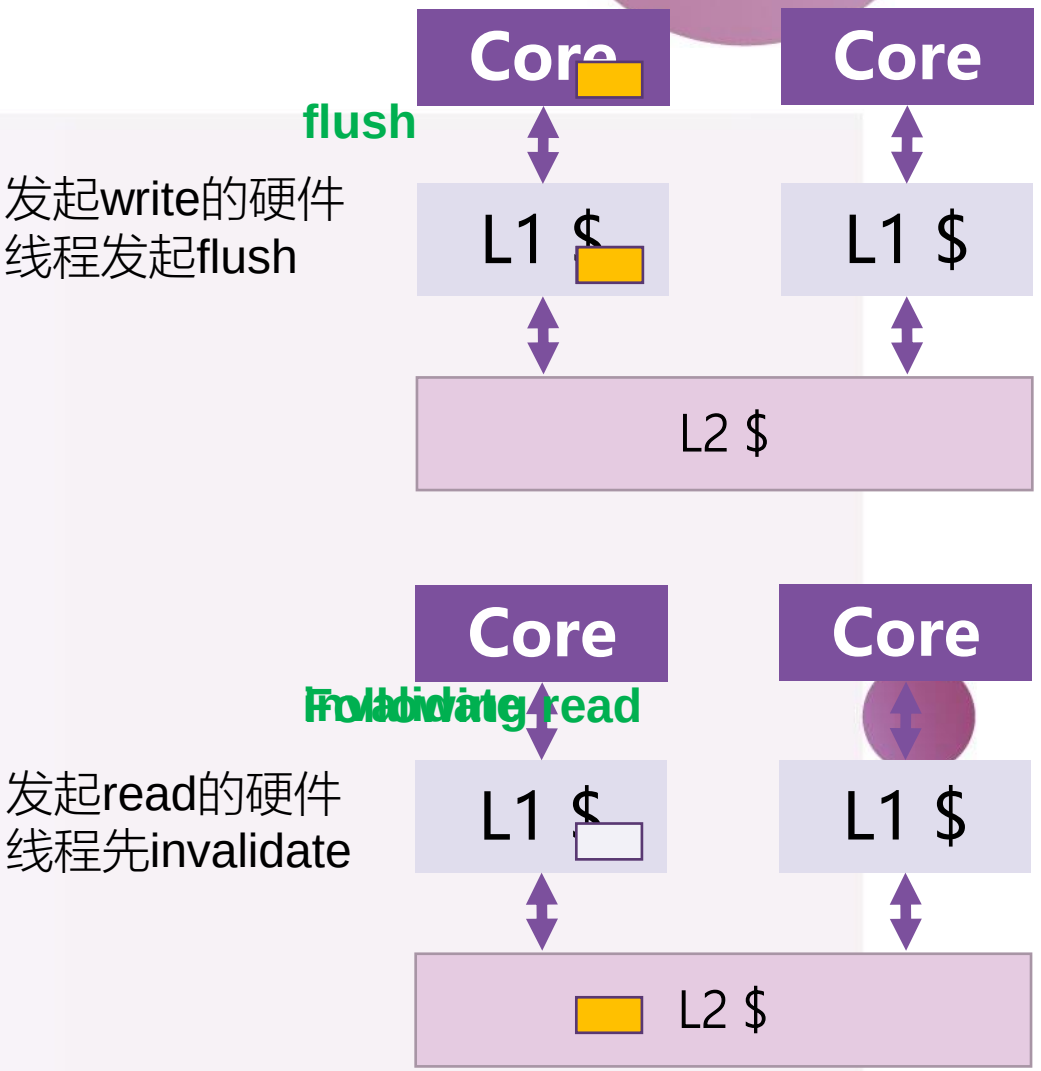


Ref
[1] HPCA13 Cache Coherence for GPU Architectures, UBC, AMD
[2] IEEE Micro 2011, A 32-Way Multithreaded Sparc Processor.
[3] OpenSPARC T2 Core MicroArch Spec, Sun

软件一致性

Software-based Coherence

- 古老的概念[1][2]。但在现代CPU架构中依旧不可缺席[3]。
- 一种做法是在编译时向程序插入“额外信息”：
  - 属性标记（volatile、non-cacheable）[2]
  - invalidate、flush[1]
- **宽松连贯性中acquire, release语义所标记之处与invalidate, flush需求重合。**



Ref
[1] ISCA82 NYU Ultracomputer
[2] 1985 IBM RP3(Research Parallel Processor Prototype)
[3] Arm® Architecture Reference Manual for A-profile architecture

连贯性导向的缓存一致性[1]

Consistency-directed Coherence

- 宽松连贯性中的acquire和release语义包含了单向禁止乱序要求。
- 在此基础上，做出以下规定（假设写回/写分配cache）：
  - release发生时，L1所有dirty cache line写回L2（即flush*），然后release对应的写入再写回L2；
  - 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）。
- 在这种机制里，**不需要额外定义独立的invalidate或flush指令**。因为consistency所要求的acquire/release语义和这两种cache行为充分耦合。

Ref
[1] Synthesis Lectures on Computer Architecture: A Primer on Memory Consistency and Cache Coherence 2nd
*selective flush

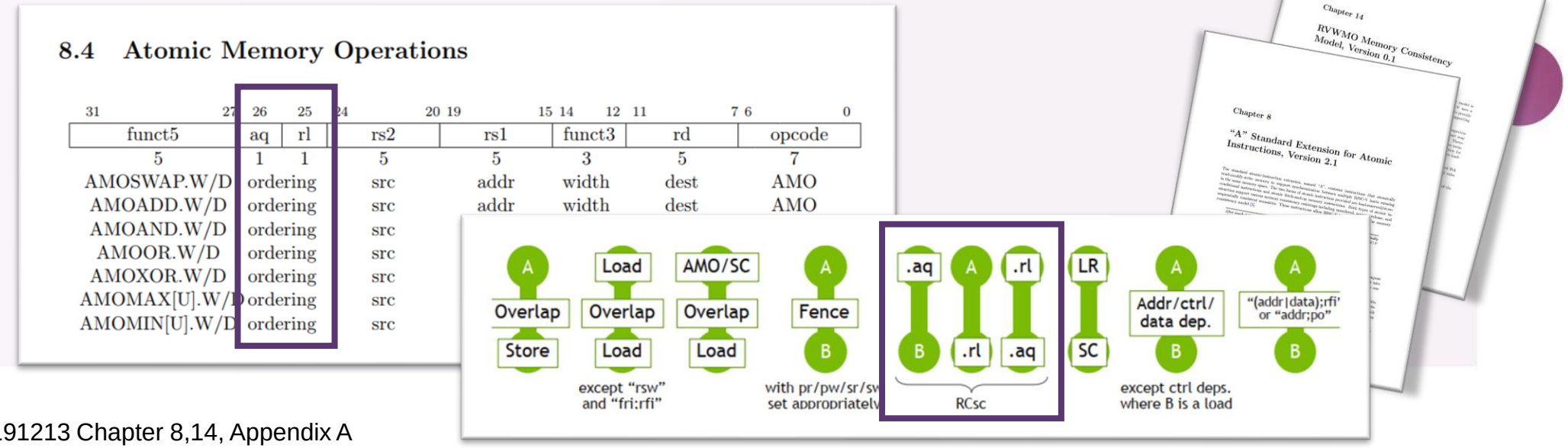
RVWMO，RISC-V弱存储排序[1]

RISC-V Weak Memory Ordering

- Relaxed Consistency变体。正式商用、业界定义的严谨模型，非仅学术用途。

4条公理，13条具体规则

- 特殊排序约束主要通过“A”扩展指令和“I”基础指令集的fence指令实现。“A”扩展指令定义了.aq和.rl下标用于实现acquire和release语义。



8.4 Atomic Memory Operations

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				
AMOOR.W/D	ordering		src				
AMOXOR.W/D	ordering		src				
AMOMAX[U].W/D	ordering		src				
AMOMIN[U].W/D	ordering		src				

Chapter 14
RVWMO Memory Consistency Model, Version 0.1

Chapter 8
“A” Standard Extension for Atomic Instructions, Version 2.1

Ref
[1] RV spec V20191213 Chapter 8,14, Appendix A

RVWMO，RISC-V弱存储排序

RISC-V Weak Memory Ordering

- 体系结构一直是一个“工程-理论”螺旋前进的领域。 RVWMO是目前业界**最晚、最新**的商用CPU存储连贯性模型。并且理论分析（Formalism）**完成度很高**。
  - 模型严谨性对高性能细粒度同步语义[1]的正确运行至关重要。
  - Nvidia（CUDA、PTX）在存储连贯性问题上也曾犯过错误[2][3]。
- RVWMO采用“公理”+“操作模型”结合的方式完成定义。说明性材料的**可读性佳**。

Hart 0		Hart 1	
	li t1, 1		li t1, 1
(a)	sw t1,0(s0)	(e)	sw t1,0(s1)
(b)	lw a0,0(s0)	(f)	lw a2,0(s1)
(c)	fence r,r	(g)	fence r,r
(d)	lw a1,0(s1)	(h)	lw a3,0(s0)

Outcome: a0=1, a1=0, a2=1, a3=0

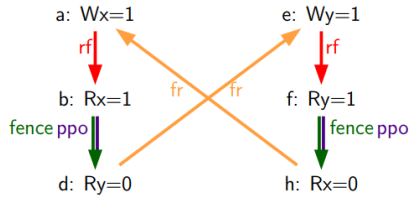


Figure A.2: A store buffer forwarding litmus test (outcome permitted)

Ref
[1]IISWC17 HeteroSync: A Benchmark Suite for Fine-Grained Synchronization on Tightly Coupled GPUs
[2]ASPLOS15 GPU Concurrency: Weak Behaviours and Programming Assumptions, UCL
[3]ASPLOS19 A Formal Analysis of the NVIDIA PTX Memory Consistency Model, Nvidia

展望RVWMO和GPU的结合

RVWMO+GPU

- 主要目标：业界应用价值，非仅学术用途。
- 从理论角度，作为CPU模型的RVWMO，其优化目标有别于GPU。
  - 最显著的区别：latency-oriented OR throughput-oriented[1]
  - CPU对乱序的追求远比GPU激进，因此RVWMO的某些"宽松性"对于GPU而言是可以不需要的。
    - 比如公理1就呈现出此问题：Load Value Axiom——store buffer。
- 从工程角度，无效化的范围如何记录？
  - 换言之，如何用硬件机制辅助软件一致性？

Ref
[1] ISCA13 Exploring Memory Consistency for Massively-Threaded Throughput-Oriented Processors

总结

Conclusion

GPU需要**宽松连贯性**以支持**细粒度同步**语义

RV定义了一套宽松连贯性模型RVWMO

我们寻求在“承影”GPGPU中适配RVWMO以实现这种结合的**业界价值**

感谢聆听！

欢迎各位嘉宾提问指正！