小型微型计算机系统
Journal of Chinese Computer Systems
DOI:10.20009/j.cnki.21-1106/TP.2024-0617
2026年2月 第2期
Vol.46 No. 2026

基于 Cache 功能模拟的 GPU 内存系统建模

袁福焱,郝晓宇,曹振伟,张 森,陈俊仕,安 虹

(中国科学技术大学 计算机科学与技术学院,合肥 230000)
E-mail:fuyanyuan@mail.ustc.edu.cn

摘 要: 重用距离分析是一种常用的基于 Trace 的 Cache 性能分析方法.然而,随着现代 GPU 微架构的持续演进,现有基于重用距离理论的 GPU 内存分析模型由于简化了过多硬件特性,导致了显著的失真.为此,本文提出一种基于 Trace 和 Cache 功能模拟的 GPU 内存系统建模框架,针对现代 GPU 的关键内存特性进行了精确建模,包括 Sector Cache、自适应 L1 缓存分配机制以及写直达与写回策略等.通过在 Volta 架构及多个基准测试套件上的实验验证,论文模型相较现有最先进模型 PPT-GPU-Mem 在多个关键指标上显著提升了预测精度:L2 命中率误差从 43.39% 降至 15.86%,显存读写事务次数误差从 42% 降至 16.85%.

关 键 词: GPU;内存模型;重用距离;功能模拟;NVIDIA NVBit
中图分类号: TP311 **文献标识码:** A **文 章 编 号:** 1000-1220(2026)02-0477-10

Modeling GPU Memory Systems Based on Cache Functional Simulation

YUAN Fuyan, HAO Xiaoyu, CAO Zhenwei, ZHANG Sen, CHEN Junshi, AN Hong
(School of Computer Science and Technology, University of Science and Technology of China, Hefei 230000, China)

Abstract: Reuse distance analysis is a common trace-based method for cache performance analysis. However, with the continuous evolution of modern GPU microarchitectures, existing GPU memory analysis models based on reuse distance theory suffer from significant inaccuracies due to oversimplified hardware characteristics. This paper proposes a GPU memory system modeling framework based on trace and cache functionality simulation, focusing on modeling key memory features of modern GPUs, including sector cache, adaptive L1 cache allocation mechanisms, and write-through versus write-back policies. Experimental validation on the Volta architecture and several benchmark suites demonstrates that the proposed model significantly improves prediction accuracy in several key metrics compared to the state-of-the-art model PPT-GPU-Mem: the L2 hit rate error is reduced from 43.39% to 15.86%, and the memory read/write transaction error decreases from 42% to 16.85%.

Keywords: GPU; memory model; reuse distance; functional simulation; NVIDIA NVBit

0 引 言

图形处理单元(Graphic Processing Unit, GPU)因其在科学计算和人工智能等领域的广泛应用而备受关注.随着 GPU 内存架构从传统的单级共享存储(Scratchpad)演进至现代复杂的多级缓存层次结构,内存系统已成为提升 GPU 整体性能的关键.因此,许多 GPU 算法的优化策略都集中在如何高效利用内存系统[1].在计算机体系结构研究中,模拟是评估新架构设计的重要方法.然而,传统的时钟周期级模拟器(如 GPGPU-Sim[2,3] 和 Multi2Sim[4])尽管能够提供高精度的结果,但因其模拟速度极慢,限制了在快速设计空间探索中的实际应用.相比之下,分析模型[5-7]能够快速预测不同配置下的性能,成为一种更加高效的替代方案.不过 GPU 内存系统的复杂性及其高并发线程的访问模式使得内存建模充满挑战,成为近年来的研究热点.

目前,针对 GPU 内存系统的分析模型已经有诸多研究[8-12],这些研究均采用了一种基于重用距离理论[13](Reuse Distance Theory)的方法,该方法最早被广泛用于 CPU 缓存建模.然而,由于 GPU 的多线程并行特性导致缓存访问模式与 CPU 显著不同,将其用于 GPU 建模需要对其进行扩展.PPT-GPU-Mem[8,15]是目前最先进的 GPU 内存分析模型框架,采用了 Brehob 等人[14]提出的基于栈距离的 Cache 模型(Stack Distance-based Cache Model, SDCM),并模拟了 GPU 的并发执行特性,对 GPU L1 和 L2 缓存进行了建模,在速度和准确性上都优于现有模型.

但是基于 SDCM 模型的研究方法存在以下问题:1)Cache 访存粒度和硬件不匹配:现代 GPU 缓存普遍采用 Sector Cache 技术,将一个 128 字节的 Cache line 划分为 4 个子扇区,Cache 缺失时按照扇区粒度请求下一级内存.而 SDCM 无法建模这一行为,会引入误差;2)多级缓存建模不准确:

收稿日期:2024-12-31 收修改稿日期:2025-03-25 基金项目:中国科学院战略性先导科技专项项目(XDB0500102)资助. 作者简介:**袁福焱**,男,1999 年生,硕士研究生,CCF 会员,研究方向为计算机系统结构、高性能计算、GPU 性能建模;**郝晓宇**,男,1996 年生,博士,研究方向为计算机系统结构、高性能计算、加速指令自动探索;**曹振伟**,男,1999 年生,硕士研究生,CCF 会员,研究方向为计算机系统结构、高性能计算、近存计算;**张 森**,男,2001 年生,硕士研究生,CCF 学生会员,研究方向为计算机系统结构、处理器性能建模、编译优化;**陈俊仕**,男,1990 年生,博士,特任副研究员,CCF 会员,研究方向为计算机系统结构、高性能计算、并行程序设计、并行编程模型和运行时系统;**安 虹**,女,1963 年生,博士,教授,CCF 会员,研究方向为超大规模并行计算机系统结构、片上多处理器体系结构、并行程序设计环境与工具.

SDCM 是概率模型,输出 Cache 整体的命中率,但是不输出单次访存是否命中. 在建模多级缓存时,无法正确过滤上一级的命中请求,导致 Trace 构造不准确,进而导致建模失真;3)对读写事务数的建模不准确:SDCM 对 Cache 的读写操作未作区分,无法建模不同的 Cache 写策略,这导致显存读写事务数估计显著失真. 由于以上问题,基于 SDCM 的 PPT-GPU-Mem 在本文的测试中 L2 和显存相关的数据上有显著失真,如 L2 命中率误差高达 43%,显存事务数误差高达 42%.

要解决以上问题,就需要对缺失的 Cache 特性进行建模. 传统的分析模型无法实现更精细化的建模,而使用复杂的时钟精度模拟又会导致模拟速度过慢. 因此,在建模复杂度与精度之间找到合理的平衡显得尤为重要. Cache 功能模拟[16]是一种高层次模拟技术,专注于建模缓存操作的功能性行为,而不涉及操作时序的细节,速度和准确度位于分析模型和时钟周期模拟之间. 因此,本文基于 Cache 功能模拟,扩展了 PPT-GPU-Mem 框架,补充了缺失的现代 GPU Cache 关键特性,使其能在保持模拟效率的同时显著提高模拟精度,为 GPU 内存性能建模提供了新的方法.

本文贡献如下:1)构建了一个基于 Trace 和 Cache 功能模拟的现代 GPU 内存建模框架:该框架基于 Cache 功能模拟对 SDCM 存在的问题进行了修正,另外还对 GPU 的自适应 L1 缓存分配机制进行了建模. 与目前最先进的内存分析模型 PPT-GPU-Mem 相比,显著提高了 L2 及以下层次建模的准确度;2)扩展了对 GPU 各级存储性能数据的建模能力:该框架不仅能够预测 L1 和 L2 缓存的命中率,还扩展了对 GPU 各级存储性能数据(如显存读取和写入事务数)的建模能力. 这些数据经过硬件实验验证,可用于构建更精确的 GPU 性能模型[15];3)针对 Volta 架构的全面验证:本文对 Volta 架构 GPU 进行了建模,从 Rodinia、Polybench、Pannotia 和 Tango 4 个侧重不同领域的测试集套件中选取了 29 个应用程序共 1736 个 Kernel 对框架的准确性进行了验证. 最终 L1 命中率误差从 14.82% 降低为 11.06%,L2 命中率误差从 43.39% 降低到 15.86%,显存总事务数误差从 42.70% 降低到 16.85%.

1 相关工作

Tang 等[17]提出了第一个 GPU Cache 分析模型,将重用距离理论引入了 GPU 缓存建模. 他们建模了单个线程块和多个线程块两种情况,并使用 GPGPU-Sim 进行验证. 但是他们工作仅对 L1 进行了建模. Nugteren 等[12]扩展重用距离理论对 GPU 缓存进行了更详细建模,考虑了线程调度、内存延迟、缓存相连度、MSHR 和 Warp 发散等因素. 他们使用 GPU 仿真器采集 Trace. 但是同样只对 L1 进行了建模. Kiani 等[10]基于重用距离分析提出了两种粒度的模型. 粗粒度的模型采集的 Trace 是架构无关的,支持不同参数下的模拟. 细粒度的模型则考虑了缓存块驻留失效和 MSHR 寄存器特性,精度更高. 但他们仅在 Maxwell 和 Kepler 等较旧的 GPU 架构上进行验证,无法评估其在现代 GPU 架构上的适用性. Arafa 等提出的 PPT-GPU-Mem[8,15]是目前最先进的 GPU 内存模型. 该模型基于 SDCM 模型并对 L1 和 L2 进行建模. 该模型利用最新的 NVIDIA NVBit 工具动态采集访存 Trace,相对于编译器插桩和手动源码插桩方式,有更好的速度和适用性. 该模型支持单个线程块和多个活跃线程块的两种粒度的模拟,从而在性能和精度之间提供不同的权衡,但是缺乏对 Sector Cache 等特性进行建模. Lee 等人[5]在他们 GPU 性能模型中集成了一个内存模型,提及了对 Sector Cache 的模拟. 然而,该研究重点关注基于区间分析方法对 GPU 性能进行整体建模,没有对内存模型单独验证,故无法评估其内存建模的准确性.

综上所述,现有 GPU 内存分析模型或缺乏对 L2 缓存的建模,或未能充分考虑现代 GPU 架构的关键缓存特性,如 Sector Cache. 本文提出的方法旨在弥补这些不足.

2 研究背景

2.1 GPU 存储层次

现代 NVIDIA GPU 的微架构如图 1 所示,主要由流式多处理器(Streaming Multiprocessor,SM)、片上互联、共享的 L2 缓存和片外显存(Device Memory)组成. SM 是 GPU 的核心组件,每个 SM 内包含 4 个子核(或称子块),每个子核配备寄存器、不同计算单元和 Warp 调度器.

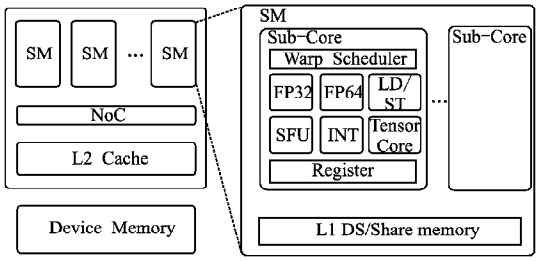


图 1 现代 GPU 架构图

Fig. 1 Modern GPU microarchitecture

GPU 主要包含 4 级存储层次结构:1)寄存器,位于 SM 子核内,读写速度最快;2)L1 数据缓存与共享内存:位于 SM 内,L1 缓存由硬件自动管理,共享内存需程序员显式管理. Volta 架构后两者共享相同片上存储;3)L2 数据缓存:所有 SM 共享的片上缓存,用于缓解显存带宽瓶颈;4)显存:也称为全局内存或设备内存,位于芯片外部,通常采用高带宽显存(如 GDDR 和 HBM)以满足带宽需求. 早期的 GPU 片上存储仅包含固定大小的共享内存,从 Femi 架构[18]开始引入了 L1 和 L2 缓存并延续至今. 如今 GPU 的片上缓存变得越来越大也越来越复杂,成为了 GPU 领域的研究热点.

2.2 CUDA 线程层次与调度

使用 CUDA 编写的 GPU 程序在 GPU 上执行的函数被称为 Kernel,成百上千个线程以并发方式执行相同的 Kernel. CUDA 编程模型将线程组织成 3 个层次:网格(Grid)、线程块(Thread Block)和线程[1],每个层级对应一层任务划分. 另外,GPU 执行时,以 32 个线程为一组调度和执行,这组线程被称为 Warp,以单指令多数据(Single Instruction Multiple Data,SIMD)方式同步执行相同指令,从而提升硬件效率.

Kernel 执行时,首先将线程块调度到不同 SM 上,每个 SM 可并发执行多个线程块. 接着 SM 上所有线程块的 Warp

[1] https://docs.nvidia.com/cuda/cuda-c-programming-guide/

被调度到子核上并发执行,最大并发 Warp 数由架构决定,如 Volta 架构为 16 个. 最后,子核内 Warp 调度器每周期发射执行一条 Warp 指令,当 Warp 不满足条件(如数据依赖)无法执行下一条指令时,Warp 调度器切换到另一个 Warp 执行.

合并访问(Coalesced Access)是 GPU 内存访问优化的一个重要概念. 当一个 Warp 中的多个线程访问连续的内存地址时,这些访问会被合并成一个或几个更大的内存事务,从而提高内存访问效率. 合并按照 Sector 粒度进行.

2.3　SDCM 模型

SDCM 模型[14]能够对任意配置的组相联 Cache 进行建模. SDCM 计算 Cache 命中率分为两步:第 1 步计算每个访存引用的重用距离(Reuse Distance,RD),并统计距离分布,从而构造复用配置(Reuse Profile,RP);第 2 步根据复用配置和 Cache 参数,分析 Cache 命中率.

RD 是相同地址的两次访问间不重复访问地址个数. 首次访问时,RD 为无穷大. 计算 RD 时的地址需要使用 Cache line 地址. 表 1 展示了一个简单的计算示例,其中假设了 Cache line 大小为 2.

表 1　重用距离计算例子

Table 1　Reuse distance caculation exmaple

访问时间	0	1	2	3	4	5
访问地址	2	4	6	2	3	6
Cache line 地址	1	2	3	1	1	3
重用距离	∞	∞	∞	2	0	1

SDCM 模型假设了 Cache 使用 LRU 替换策略,且每个地址引用均匀映射到 Cache line. 假设 Cache 相连度为 A,Cache line 数目为 B(等于 Cache 容量除以 Cache line 大小). 如果一条访存 RD 等于 D,那么单次访存命中率如公式(1)所示. 整个程序的平均命中率可以使用公式(2)计算得到[14].

$$P(h|D)=\sum_{a=0}^{A-1}\binom{D}{a}\left(\frac{A}{B}\right)^{a}\left(\frac{B-A}{B}\right)^{(D-a)} \tag{1}$$

$$P_{Hit}=\sum_{i=0}^{N}P(D_i)\times P(h|D_i) \tag{2}$$

2.4　PPT-GPU-Mem 框架

PPT-GPU-Mem 是目前最先进的 GPU 内存建模框架,主要由 Trace 采集、L1/L2 Trace 构造以及 SDCM 预测 3 个阶段构成. 为了实现比已有模型更高的准确度和更高的模拟速度,PPT-GPU-Mem 在以下方面做出了优化.

首先,在 Trace 采集阶段,PPT-GPU-Mem 使用使了 NVIDIA 最新的 NVBit 工具[19]来采集程序的访存 Trace,相比于手动源码插桩,有更好的适用性,相比于功能模拟采集,能反映硬件真实的访存序列,有更高的准确度.

接着,在 L1/L2 Trace 构造阶段,PPT-GPU-Mem 忠实地模拟了 GPU 线程块到 SM 的调度,以提高 Trace 构造的准确性,实现了轮转法(Round Robin)和随机映射等多种调度算法.

最后,PPT-GPU-Mem 支持模拟部分线程块来提升模型的扩展性. Wang 等[17]的工作表明 SM 上仅模拟单个线程块便足够分析 L1 缓存的命中率. PPT-GPU-Mem 支持仅模拟最大活跃线程块,以实现准确度和速度的平衡. 最大活跃块数目和 Kernel 使用的资源多少(如寄存器数、共享内存大小)以及 GPU 硬件架构资源有关,公式(3) ~ 公式(6)表示了这些关系.

$$\mathrm{MaxActBlk}=\min(\mathrm{BlkLimit}_{\mathrm{Warp}},\mathrm{BlkLimit}_{\mathrm{reg}},\mathrm{BlkLimit}_{\mathrm{shmem}}) \tag{3}$$

$$\mathrm{BlkLimit}_{\mathrm{Warp}}=\frac{\mathrm{MaxThrPerSM}}{\mathrm{BlkSize}} \tag{4}$$

$$\mathrm{BlkLimit}_{\mathrm{reg}}=\frac{\mathrm{MaxRegPerSM}}{\mathrm{RegPerBlk}} \tag{5}$$

$$\mathrm{BlkLimit}_{\mathrm{shmem}}=\frac{\mathrm{MaxShmemPerSM}}{\mathrm{ShmemPerBlk}} \tag{6}$$

3　PPT-GPU-Mem 的不足

虽然 PPT-GPU-Mem 是目前最先进的 GPU 内存建模框架,但仍然存在一些不足. 具包含在 Cache 访存粒度不匹配、L2 trace 构造不准确和显存读写事务数建模不准确 3 个方面.

3.1　Cache 访存粒度

现代 GPU 使用 Sector Cache 来降低 Cache 缺失时不必要内存访问事务的数量. 图 2 展示了非 Sector Cache 和 Sector Cache 不同的访存行为. 假设一个 Warp 中 4 个线程按照 64B 的跨度访问 8B 的数据,Cache line 大小为 128B,扇区大小为 32B,访问的地址空间如图 2(a)所示. 在非 Sector Cache 情况下,L1 缺失时需要从 L2 加载两个 Cache line 共 256B 的数据,如图 2(b)所示. 而在 Sector Cache 下,需要从 L2 加载 4 个扇区共 128B 数据,如图 2(c)所示.

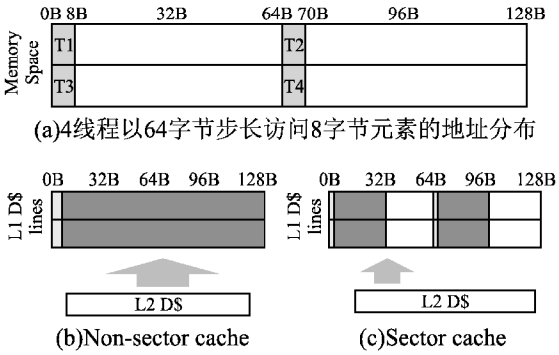


图 2　扇区缓存和非扇区缓存访存粒度对比

Fig. 2　Comparison of access granularity between non-sector cache and sector cache

SDCM 模型没有建模 Sector Cache 行为,经实验,如果按照 128B 的 Cache line 大小进行预测,PPT-GPU-Mem 预测的 L1/L2 命中率会产生显著偏差,并且统计的请求事务数和实际硬件最多会存在 4 倍的关系,存在显著失真. PPT-GPU-Mem 论文设置 Cache line 大小为 32 字节作为 Sector Cache 的近似. 但是 5.3.1 节的实验表明,这样虽然对于 L1 的效果不错,对于 L2 仍然存在较大误差.

3.2　L2 Trace 构造

图 3 展示了 PPT-GPU-Mem 构造 L1 和 L2 Trace 的一个例子. 假设 GPU 有 2 个 SM,程序 Kernel 有 4 个线程块,每个线程块有两条 Warp 访存指令. 经过轮转调度后,线程块 0 和 2 被调度到 SM0 上,1 和 3 被调度到 SM1 上. 接着,SM0 内线

程块的所有 Warp 通过交叉混合得到 SM0 L1 的 Trace,用于 SDCM 模拟,SM1 类似. 最后,SM0 和 SM1 的 L1 Trace 再经过交叉混合得到 L2 的 Trace,用于 L2 SDCM 模拟.

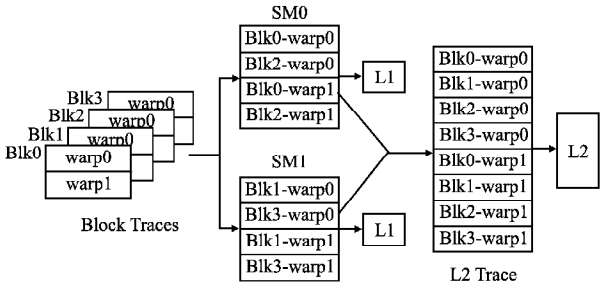


图 3 PPT-GPU-Mem L1 和 L2 Trace 构造过程

Fig. 3 L1/L2 Trace construction of PPT-GPU-Mem

由此可知,PPT-GPU-Mem 没有过滤 L1 中命中的访存指令,这会导致构造的 L2 Trace 有较大偏差. 而 SDCM 模型是概率模型,不会输出单次访问是否命中,因此使用 SDCM 建模多级缓存时,Trace 的构造存在不准确. 之后的实验表明,不过滤 L1 的命中会对 L2 造成较大误差.

3.3 显存读写事务建模

SDCM 模型用于评估 Cache 命中率,而读写事务不仅和命中率有关,还和 Cache 使用的写策略有关. 对于一个采用写回写分配的Cache,状态机如图4所示. 当发生读缺失时,需要先从内存读取一个 Sector,然后分配一个 Cache line,并标记 Cache line 中该 Sector 有效. 当发生写缺失时,同样需要先从内存读取一个 Sector,然后部分更新其中需要写入的数据(比如 4 字节),最后写入 Cache line. 当无法分配空闲 Cache line 时,则需要根据替换策略替换一个 Cache line. 替换的 Cache line 中如果包含标记为脏的 Sector,则需要先写回到内存中. 因此可以发现,在采用写回 Cache 的情况下,读内存也不是仅发生在读缺失时,也可能发生在写缺失时,而写内存的情况并不是发生在写缺失时,而是发生在替换脏 Cache line 以及清空 Cache 写回内存时. 因此,仅通过缓存命中率无法正确建模读写事务.

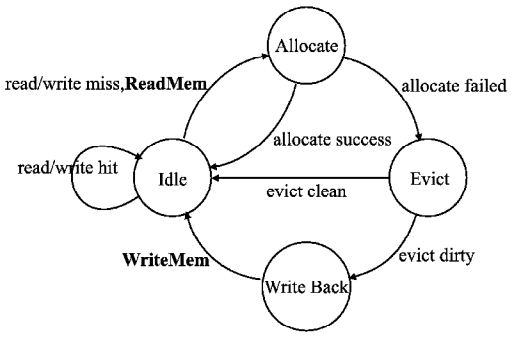


图 4 Cache 功能模拟器写回状态机

Fig. 4 Write-back state machine for Cache functionality simulato

对于现代 GPU,L1 通常采用写直达策略,L2 通常采用写回策略. 并且,Khairy 等[20] 发现 Volta 架构 L2 采用了一种叫做写验证的写分配策略,实现了 1 字节粒度的写操作. 通过每个字节维护一个有效掩码,写缺失时可以仅更新 Sector 中部分字节,从而避免读取内存,减少内存的访问. 不过由于实现这部分建模较为复杂,本文对于 L2 缓存仍然采用一般的写回策略建模.

由于 SDCM 模型无法建模不同的 Cache 写策略,基于 SDCM 的 PPT-GPU-Mem 仅通过 L2 命中率估计显存的读写事务. 本文实验表明,PPT-GPU-Mem 显存相关数据存在显著偏差,具体分析在 5.2.3 节.

4 改进的内存模型

为了解决 SDCM 方法存在的问题,本文实现了一个支持 Sector Cache 和不同写策略的 Cache 功能模拟器,用于替代原本的 SDCM 模型. 图 5 展示了本工作实现的内存建模框架. 框架分为 Trace 采集和内存模拟两部分. Trace 采集部分和 PPT-GPU-Mem 一致. 在内存模拟部分,采用 Cache 功能模拟器对 L1 和 L2 建模. 功能模拟器除了输出命中率,还会输出对下一级内存的读写请求. 这既保证了构造 L2 Trace 时可以过滤 L1 命中的请求,也使得模拟器可以对 L2 的写策略进行建模,产生正确的显存事务请求. 框架还对一些其他 GPU 特性进行了建模,比如适应 L1 缓存(即 L1 缓存大小会根据 Kernel 使用共享内存的多少而自适应调整),这提高了建模的准确度.

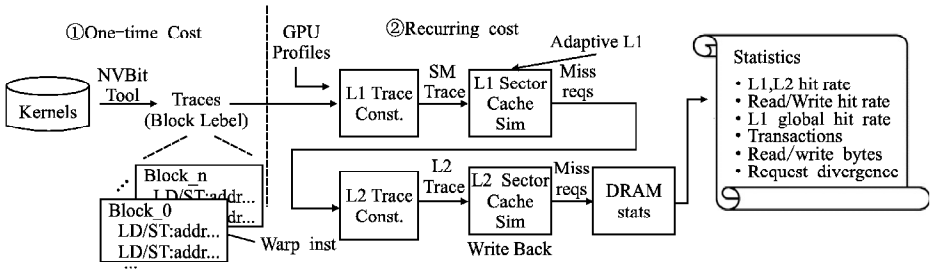


图 5 基于 Cache 功能模拟的 GPU 内存建模框架

Fig. 5 GPU memory modeling framework based on Cache functional simulation

框架输入信息为 GPU 硬件配置,包括 GPU 架构版本、SM 数目、L1/L2 的 Cache 配置参数等. 对于输出信息,不同于以往大部分工作[8,10-12] 仅输出 L1、L2 缓存的命中率,本文框架还输出各级存储的读写事务数、请求发散度(单个内存请求的平均事务数)等. 这些数据在对 GPU 整体性能进行建模时,可以用于构建更准确的内存性能模型[15]. PPT-GPU-Mem 在其后续的 GPU 性能建模工具 PPT-GPU[15] 中支持输出更多数据,但仅针对L1和L2缓存命中率进行了硬件验证. 本工

作对其他输出数据进行了全面的硬件对比验证,弥补了 PPT-GPU 的不足. 本文实现的代码开源于 GitHub[2] 上.

4.1 Cache 功能模拟

本文实现了一个支持 Sector Cache 和多种写策略的,LRU 多路组相连 Cache 功能模拟器. 图 6 显示了 Cache 功能模拟器的数据结构和原理. 模拟器采用双向链表来维护每一个 Cache Set,除了首部节点,每个链表节点对应一个 Cache line. 双向链表用于维护 Cache 的访问顺序,当访问某个 Cache line 后,需要将该链表节点移动到首部. 因此采用 LRU 替换策略时,链表尾部节点对应需要替换的节点. 另外,模拟器使用一个哈希字典记录地址到 Cache line 链表节点的映射,用于快速判断 Cache 访问是否命中,以及命中后操作链表节点. 因此,Cache 模拟器所有操作包括读取、写入、替换、更新 LRU 信息的时间复杂度均为 $O(1)$,这保证模拟器有着足够快的模拟速度.

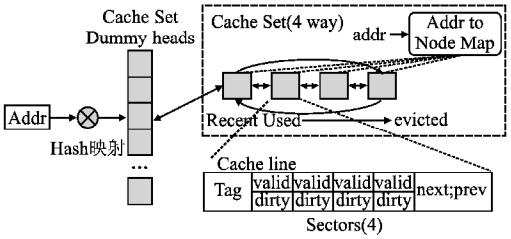


图 6　Cache 功能模拟

Fig. 6　Cache functional simulation

链表节点数据结构记录了 Cache line 信息. 为了实现 Sector Cache,链表节点需要记录 Cache line 中每个 Sector 的信息. 最终每个链表节点包含:1)地址标签(tag),用于区分不同的内存访问地址,等于地址除以 cache line 大小;2)next,prev 指针,用于实现双向链表;3)每个 sector 的有效位(valid)信息. 初始设置为 0,当 Cache 缺失填充后,根据地址将对应的 Sector 有效位置为 1. 当读取时,即使 Cache line 命中,如果 Sector 有效位为 0,仍然判定为缺失;4)每个 Sector 的脏位(dirty)信息,用于实现写回策略,当 Sector 被写入时设置为脏,当 Cache line 被替换时,将所有设置为脏的 Sector 写回内存.

另外,图 6 展示了本文采用哈希映射的方法将地址映射到 Cache Set,而非简单的模运算线性映射. Nugteren 等人[12]发现 Fermi 架构的 Cache 将地址经过哈希函数的结果作为 Cache Set 的索引. 本文在实验时,发现 PolyBench 测试集中的许多程序每次循环按照 Cache 容量倍数的跨度访问数组,理论上会导致每次访问地址映射到相同的 Cache set,导致严重的冲突缺失. 但是硬件采集的性能数据却没有发生该现象,因此推断 Volta 等新的硬件均采用了哈希映射. 通过实现哈希映射,保证了每个 Cache 访问被均匀地划分到不同的 Cache Set,降低了冲突缺失率.

模拟器实现了写回与写直达、写分配与写不分配,两两组合共 4 种写策略. 根据不同的写策略,调整对下一级产生的读写事务. 更多细节可以参考本文开源代码,这里就不再赘述.

[2]https://github.com/TheRainstorm/ppt-gpu-subcore

[3]https://github.com/TheRainstorm/GPGPUs-Workloads

4.2 自适应 L1 Cache

本文对 GPU 自适应 L1 缓存进行了建模. 从 Volta 架构开始,SM 中 L1 数据缓存和共享内存共享同一块片上存储. L1 的大小会根据 Kernel 使用的共享内存大小而自适应设置[20]. 以 Volta 架构为例,L1 缓存和共享内存共享 128KB 的存储空间. 共享内存大小可以在运行时被 GPU 驱动设置为 0,8,16,32,64,96KB 中的一个,选择的目标是在避免共享内存成为 SM 利用率瓶颈的情况下尽可能小,腾出空间分配给 L1 缓存. 例如,对于不使用共享存储的程序,全部 128KB 空间被分配给 L1 缓存,从而可以较大提高程序性能. 公式(7)表示了共享内存大小的计算方法. 其中 MaxActBlk(x)把公式(3)看作是关于共享内存大小 x 的函数. MaxShmemPerSM 表示共享内存的最大大小(Volta 中为 96KB). 最后,L1 缓存大小通过 SM 片上存储总大小减去共享内存大小得到. 5.3.3 节表明实现该特性能提升 L1 预测的准确度.

$$\begin{aligned}\text{AdaptiveShmem} = \min_{x}\{x \in \text{ShmemCarveout} \mid \text{MaxActBlk}(x) \\ = \text{MaxActBlk}(\text{MaxShmemPerSM})\}\end{aligned} \tag{7}$$

5 实验结果与分析

5.1 实验设置

为了评估模型改进的效果,本文在 4 个不同基准测试套件上进行了测试. 表 2 列出了使用的程序描述、程序包含的 Kernel 数以及访存 Trace 的大小. 本文从使用广泛的 Rodinia 测试集[21]中选择了 13 个应用. Rodinia 测试集应用差异度较高,涵盖了图、机器学习、线性代数、生物医学、分子动力学等各领域应用. 本文从 Polybench 测试集[22]选择了 9 个程序,Polybench 主要包含各种数值计算程序,如矩阵、卷积等,对内存系统的压力很大. Polybench 不需要数据输入文件,在编译时可以指定每个程序运行测试的规模,本文使用默认的标准规模. 本文从 Pannotia 测试集[23]中选择了 4 个程序,Pannotia 以不规则的图应用为主,有着不规则的数据结构和访问模式,因此对模型的考验很大. 为了测试本文框架对机器学习程序的适用性,本文还从 Tango[24]中选取了包含 AlexNet 和 LSTM 在内的 3 个机器学习应用,涵盖了卷积神经网络和循环神经网络两种主要的深度学习算法. 由于本文的模型是逐 Kernel 模拟,在选取程序时,本文避免选择那些对单个小 Kernel 重复调用成百上千次的程序. 过多的小 Kernel 会导致 Trace 采集和模拟时间过长. 一些工作[25]考虑从所有 Kernel 中选取代表 Kernel 或者使用采样方式来降低模拟时间,将这种方法和本工作结合可以作为未来工作.

最后,本文最后一共测试了 37 个应用实例,总计 1736 个 kernel,采集的 Trace 总大小为 196 GB. 其中部分应用使用不同的输入文件和参数运行了多次,Pannotia 应用还包含采用不同算法的编译版本. 表 2 中程序名称圆括号内数字表示应用运行实例数目. 在计算结果时,本文将每个应用实例内所有 Kernel 指标取平均,作为该应用实例的结果. 本文测试中使用的所有运行参数和数据均可以在本文维护的开源的基准测试集仓库[3]中找到.

本文对 Volta 架构的 TITANV GPU 进行了详细的建模，表 3 展示了模型使用的硬件参数. 由于 NVIDIA 没有公开所有参数，因此部分参数需要推测设置，这部分未公开参数使用星号标出. 本文参考 Khairy 等人工作[2]将 L1、L2 组相联数分别设置为 4 和 32，该值和 PPT-GPU-Mem 中采用的 64 和 16 不同. 本文 L1 缓存采用写直达的策略，L2 缓存采用了写回写分配的策略，这个和大部分工作一致[2,15,20]. 出于实现的简单性，本文 L1、L2 缓存均采用 LRU 替换策略.

表 2 测试集

Table 2 Benchmarks

程序名称	描述	Kernel 数	Trace 大小	程序名称	描述	Kernel 数	Trace 大小
b+tree[20]	B+树搜索与范围查找	2	719MB	lavaMD	分子动力学模拟	1	475MB
backprop	反向传播神经网络	2	160MB	nn	K 最近邻算法	1	2MB
bfs	图的广度优先搜索	16	4MB	nw	DNA 序列对比	255	120MB
cfd	流体动力学数值模拟	24	729MB	pathfinder	网格最短路径求解	5	161MB
dwt2d	二维离散小波变换	10	173MB	srad	去斑点各向异性扩散	300	5GB
gaussian(2)	高斯消去法求解方程	156	5MB	streamcluster	在线流数据聚类	224	7GB
hotspot	芯片热分布物理模拟	1	62MB				
2mm[21]	两次矩阵乘	2	80GB	gesummv	标量、向量和矩阵乘	1	2GB
3mm	三次矩阵乘	3	15GB	syrk	对称秩-2k 操作	1	40GB
bicg	双共轭梯度求解器	2	1GB	convolution2D	二维卷积	1	2GB
gemm	通用矩阵乘法	1	5GB	jacobi2D	二维 Jacobi 模板计算	40	2GB
gemver	向量乘和矩阵加	3	3GB				
AlexNet-23]	卷积神经网络	22	18GB	LSTM	循环神经网络	1	557KB
GRU	循环神经网络	2	1MB				
color(4)[22]	图顶点着色算法	92	6GB	mis(2)	最大独立集算法	38	2GB
fw(3)	图最短路径算法	447	2GB	pagerank(2)	网页排名概率分布算法	83	4GB

表 3 TITANV 配置参数

Table 3 TITANV GPU parameters

架构	volta/7.0	#SM	80
共享内存(KB)	0,8,16,32,64,96	替换策略*	LRU
L1 容量	32-128KB	L2 容量	4.5MB
L1 缓存行	128B	L2 缓存行	128B
L1 相连度*	4	L2 相连度*	32
L1 扇区大小	32B	L2 扇区大小	32B
L1 写策略*	写直达-写分配	L2 写策略*	写回-写分配

本文使用 NVIDIA Nsight Compute 性能分析工具[4] 收集硬件访存相关性能数据. Nsight Compute 是 NVIDIA 官方推出的用于取代传统 nvprof 工具的新一代性能分析平台，支持设置重放模式和缓存刷新策略. 本文采用默认的 Kernel 级重放，并启用缓存刷新. 启用缓存刷新策略会在每次每次重新运行 Kernel 时刷新缓存，从而保证缓存相关性能数据的一致性. 本文也通过多次采集数据取平均值，来减少采集误差.

作为本文对比的基准，PPT-GPU-Mem 的源代码包含在其后续工作 PPT-GPU[5] 中，本文提取其中内存模型部分用于对比. 对比时，由于 PPT-GPU-Mem 没有实现 Sector Cache，使用 128 字节 Cache line 误差会很大. 因此为了公平的比较，本文将其 L1 和 L2 Cache line 大小设置为 32 字节，以粗略近似 Sector Cache 的情况，其余硬件参数保持相同.

5.2 结果分析

表 4 展示了本文模型和 PPT-GPU-Mem 在建模 Volta 架构时，对 L1、L2 命中率和各级读写事务数结果对比. 本文采用平均绝对百分比误差(MAPE)和皮尔逊相关系数(Pearson Correlation Coefficient)来评价模型的效果. MAPE为所有运行实例误差绝对值的平均值，越接近 0 越好. 相关系数用来衡量预测值和硬件实际值的线性相关性，越接近 1 越说明预测值和真实值趋势一致. 由于部分预测指标在某些运行实例中可能为 0，本文在计算每个指标 MAPE 时排除了硬件值为 0 的应用实例.

表 4 建模 NVIDIA TITANV 架构，PPT-GPU-Mem 和本文模型的平均绝对误差和相关系数对比

Table 4 Average absolute error and correlation rate of PPT-GPU-Mem versus our model when modeling NVIDIA TITANV

Metrics	MAPE		Correlation	
	PPT-GPU	Ours	PPT-GPU	Ours
L1 Hit Rate	14.70%	11.06%	0.99	0.99
L1 Hit Rate(Global)	14.82%	11.17%	0.99	0.99
L1 Hit Rate(Global Load)	10.22%	7.26%	0.99	1.00
L2 Reads	23.66%	11.54%	0.95	0.95
L2 Writes	17.24%	**11.80%**	1.00	1.00
L2 Total Reqs	21.57%	**9.11%**	1.00	1.00
L2 Hit Rate	43.39%	**15.86%**	0.76	**0.94**
L2 Read Hit Rate	158.42%	**38.98%**	0.75	**0.90**
DRAM Reads	53.06%	**13.33%**	0.87	**1.00**
DRAM Writes	896.21%	1383.14%	0.50	**1.00**
DRAM Total Reqs	42.70%	**16.85%**	0.94	1.00

从表 4 中可知：1)PPT-GPU-Mem 和本文模型在 L1 上预测的准确度都比较高，误差分别为 14.50% 和 11.17%；2)本文模型在 L2 和显存相关内存指标上准确度提升显著. L2 命中率误差从 43.03% 降低为 15.73%，显存事务次数误差从 42.70% 降低到 16.85%；3)L2 和显存相关内存指标的相关系

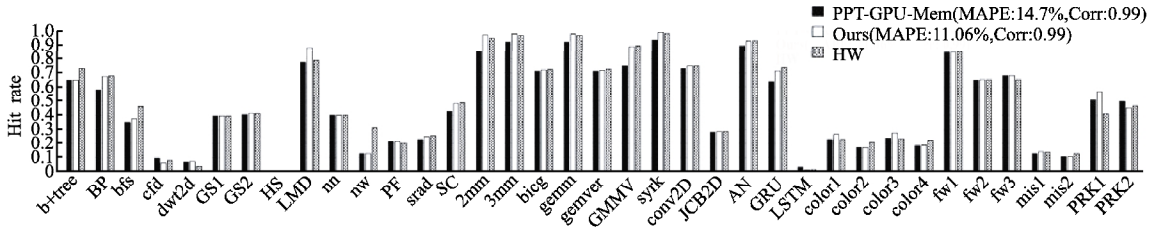
[4]https://docs.nvidia.com/nsight-compute/NsightComputeCli/

[5]https://github.com/NMSU-PEARL/PPT-GPU

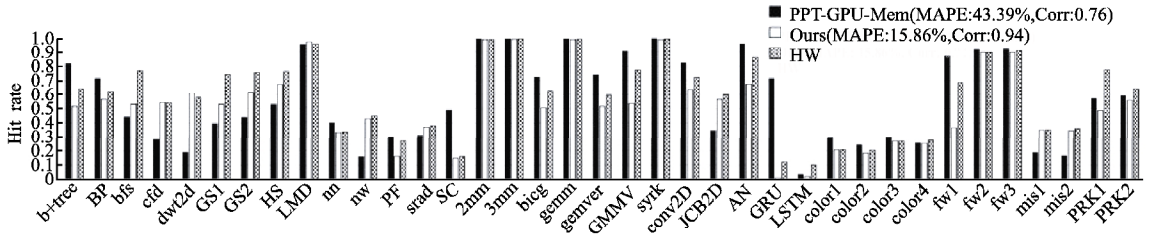
数得到大幅提升,相关系数均达到 0.9 以上. 接下本文来按照内存层次依次分析对比结果.

5.2.1　L1 缓存结果分析

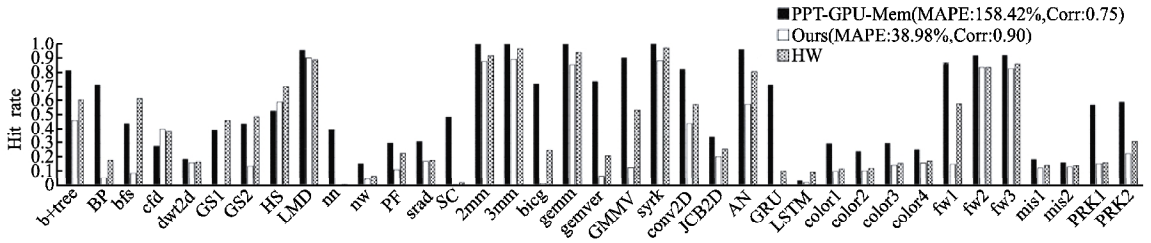
图 7(a)展示了两个模型在不同应用中的 L1 缓存命中率与硬件真实值的对比. 图中数据条依次表示 PPT-GPU-Mem、本文模型和硬件真实值,应用顺序与表 2 保持一致. 结果表明,在 L1 缓存建模方面,PPT-GPU-Mem 和本文模型均表现出较高的准确性,平均误差分别为 14.50% 和 11.17%. 值得一提的是,本文模型在 L1 全局读取命中率上的误差仅为 7.26%,低于 10%. 此外,两个模型在构造每个 SM 的 L1 访存



(a)L1命中率



(b)L2命中率



(c)L2读取命中率

图 7　本文模型和 PPT-GPU-Mem 对 L1 和 L2 命中率的预测值对比

Fig. 7　L1 and L2 hit rate prediction comparation between PPT-GPU-Mem and our model

序列时,仅模拟了最大活跃线程块,这表明只模拟部分线程块已足以准确建模 GPU 的 L1 缓存行为. 另一方面,PPT-GPU-Mem 模型的低误差也说明,对于 L1 缓存,采用 32 字节的 Cache line 近似模拟 Sector Cache 不会显著影响精度. 因此对于对速度要求较高的性能建模任务,采用这种简化是可行的.

本文在 L1 缓存建模准确度上的提升,主要得益于引入了自适应缓存机制. L1 缓存大小会根据 Kernel 使用共享内存的情况动态调整. 对于那些未针对共享内存进行专门优化的程序,更大的 L1 缓存容量能够显著提升性能. 5.3.3 节敏感性实验将进一步说明自适应缓存行为的建模对提高 L1 模拟准确度的重要性. 此外,表 4 还展示了 L2 缓存的读写请求数结果. 在 L1 缓存准确度显著提升的情况下,本文模型将 L2 缓存总请求数的误差从 22.5% 降低至 10.4%.

5.2.2　L2 缓存结果分析

表 4 展示了本文模型在 L2 缓存命中率上的显著提升. 相比 PPT-GPU-Mem 模型,本文模型将 L2 命中率误差从 42.0% 降低至 15.7%,同时将相关系数从 0.77 提高至 0.95. 在 L2 读取命中率方面,改进尤为明显,误差从 252% 大幅降低至 35.7%.

图 7(b)和图 7(c)分别显示了两种模型在不同应用程序上的 L2 总体命中率和 L2 读取命中率的表现. 通过分析图 7(c)可以发现,PPT-GPU-Mem 模型的 L2 读取命中率在许多应用上明显高于硬件实际值. 其主要原因在于 PPT-GPU-Mem 未能有效过滤 L1 缓存命中的访存请求,导致其构造的 L2 访存序列仍然保留较高的局部性,从而使得预测的 L2 命中率偏高. 在 nn 和 SC 等应用中,这一问题表现得尤为明显. 这些应用的实际 L2 读取命中率几乎为 0,而 PPT-GPU-Mem 模型的预测值却接近 L1 读取命中率,造成严重失真,最终导致其 L2 平均读取命中率误差高达 252%. 针对这一现象,5.3.2节将进一步探讨是否过滤 L1 命中的访存请求对结果局部性的影响.

5.2.3　显存结果分析

如表 4 所示,显存读取和总事务数预测误差大幅降低,其中显存总事务数预测误差从 42.70% 降低为 16.85%. 对于显存写入事务数,研究发现测试中存在许多异常点. 比如 rodinia 的 nn 程序,该程序 L2 写入的地址无重复,写入数据量为 171 KB,理论上这些指令应该全部写入显存,但是 Nsight 实际采集到的显存写入事务数却非常接近 0,显存写数据量仅为 416B. 这种结果可能是由于 Nsight 分析工具的限制,或者硬件中存在某种优化机制避免了显存写入导致的. 这些异常点

使得显存写入事务数 MAPE 异常高. 如果采用对异常点更不敏感的正规化均方根误差(Normalized Root Mean Square Error)指标进行评估,则本文模型的表现更为优越,其误差从 2.16降至 0.36.

图 8 显示了本文模型与 PPT-GPU-Mem 显存事务预测值和硬件真实值在所有 Kernel 上的相关性对比. 横坐标表示硬件真实值,纵坐标表示模型预测值,三角为 PPT-GPU-Mem 结果,圆圈为本文模型结果. 结果可以清晰地看到,本文模型的预测值更符合真实硬件值,数据点基本分布在斜率为 1 的直线上,而 PPT-GPU-Mem 则偏离直线较远.

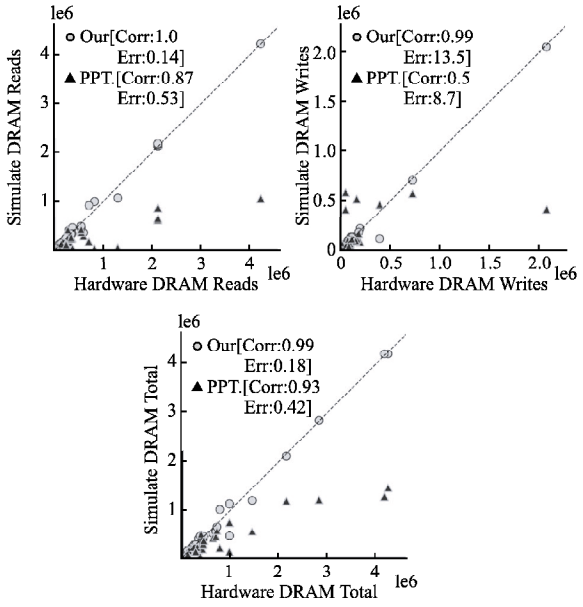


图 8 本文模型和 PPT-GPU-Mem 显存事务数硬件相关性对比

Fig. 8 Hardware correlation of memory transaction counts between the proposed model and PPT-GPU-Mem

前面已经分析,SDCM 模型无法对 Cache 写策略进行建模. 为了评估写策略对显存读写事务的影响,本文使用 Nsight 统计的硬件真实数据,使用 L2 读取请求数和 L2 读取缺失

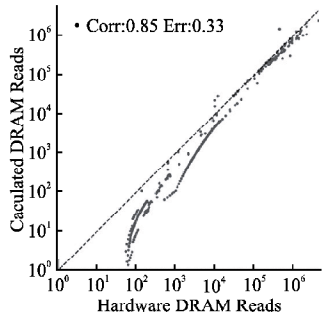


图 9 基于 L2 命中率的 DRAM 读取事务数硬件相关性

Fig. 9 Hardware correlation of DRAM read transactions based on L2 hit rate

率,来计算显存读取事务数. 将所有 Kernel 计算结果和硬件真实显存读取事务数进行对比,得到相关性如图 9 所示,图像使用对数坐标轴. 可以发现,大部分计算值都小于硬件真实值,平均误差达到了 33%. 这说明仅通过 Cache 命中率,没法准确对读写事务进行准确建模. 本文模型对写策略进行了建模,显存读取事务数误差达到 13.33%,这验证了建模的有效性.

5.3 敏感性测试

本节单独测试模型做出的各种改进对结果的影响,以验证改进的有效性. 表 5 展示了禁用不同改进后,L1、L2 命中率和显存请求总数与硬件的误差对比. 在测试某一项改进时,保证其他参数完全一致.

表 5 禁用不同改进后的最终模型硬件误差对比

Table 5 Hardware error comparison of the final model with different improvements disabled

	Final	L1NonSec	L2NonSec	NoL1Filter	NoAdaptL1
L1 Hit Rate	11.07%	10.53%	11.07%	11.07%	19.33%
L2 Hit Rate	15.47%	16.45%	17.16%	37.75%	14.60%
DRAM Total Reqs	17.83%	17.12%	28.14%	24.80%	23.03%

5.3.1 Sector Cache 影响

为了测试 Sector Cache 对模型效果的影响,本文使用最终模型作为基准,单独测试了 L1 和 L2 缓存在启用和关闭 Sector Cache 时的表现. 关闭 Sector Cache 时,将 Cache line 大小设置为 32 字节,作为对 Sector Cache 的粗略近似. 结果如表 5 所示. 其中,Final、L1NonSec 和 L2NonSec 分别表示最终模型、L1 禁用 Sector Cache、L2 禁用 Sector Cache 的结果.

可以看出,L1 缓存的命中率在启用与关闭 Sector Cache 时差别不大,这表明在 GPU L1 缓存的建模中,Sector Cache 的影响较小. 可以使用更小的 Cache line 来粗略近似,以往的工作也都采用了这种方法. 然而,在 L2 缓存中,关闭 Sector Cache 导致 L2 缓存命中率的误差稍微上升,并且显存事务数的误差明显增加. 这表明,对于 L2 缓存,建模 Sector Cache 是必要的,能够提升显存建模的准确性.

5.3.2 过滤 L1 命中

PPT-GPU-Mem 在构建 L2 访存序列时,并未过滤掉 L1 命中的访存请求. 为了研究这一点对 L2 命中率的影响,本文使用最终模型作为基准,测试了在关闭 L1 过滤时的结果,如表 5 中的 NoL1Filter 所示. 与最终模型对比,结果表明关闭 L1 过滤后,L2 命中率误差显著上升,从 12.57% 增加至 37.75%. 统计结果显示,在关闭 L1 过滤的情况下,86% 的应用实例中 L2 命中率均高于硬件实际值,这表明在未过滤 L1 命中的情况下,L2 命中率偏高.

为进一步分析原因,本文使用了 SDCM 模型中累计栈距离分布图(Cumulative Stack Distance Distribution,CSDD)[14],来可视化过滤前后 L2 访存序列的局部性变化. 图 10 显示了若干应用程序在启用和关闭 L1 过滤情况下 L2 的 CSDD 图. 图横坐标表示重用距离,纵坐标表示小于横坐标重用距离的访存所占比例. 程序局部性越好,较小重用距离的访存占比越高,曲线就越靠近左上角. 由于强制性缺失(Compulsory Miss)的存在,纵坐标可能无法达到 1. 图中除了 hotspot 程序,过滤前和过滤后局部性基本一致,其它程序,过滤后的局部性都要明显弱于过滤前. 其中 convolution2D 程序在过滤前 L2 命中率为 90%,启用过滤后仅为 44%,这和实际硬件的 57%

更接近. 综上所述,建模 L2 缓存时,不过滤 L1 命中会对 L2

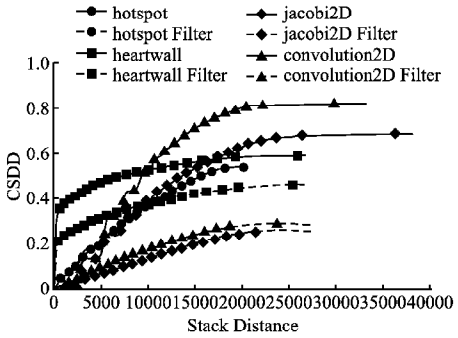


图 10　是否过滤 L1 命中情况下,不同应用 L2 累计栈距离分布对比

Fig. 10　Comparison of L2 cumulative stack distance distribution under different applications with and without filtering L1 hits

的访存局部性造成较大影响.

5.3.3　自适应 L1 缓存

为了研究自适应 L1 缓存机制造成的影响,本文测试了在启用和关闭改机制情况下模型的误差. 表 5 中 NoAdaptL1 表示了关闭自适应 L1 缓存下的的结果. 可以看到,关闭情况下,L1 命中率误差增大显著,从 11.07% 增加为 19.33%.

为了研究为什么会造成这么大的误差,本文统计了所有 Kernel 开启自适应缓存情况下,L1 缓存的大小的分布,结果表明,128KB 的 L1 占比最高,为 66%,且 64KB 和 112KB 各占 15% 和 13%,即大部分 Kernel L1 大小比默认的 32KB 要大的多,因此未建模自适应 L1 缓存情况下误差较高. 结果同样说明测试集中大部分程序都没有利用共享内存进行充分优化,而是依赖于 Cache 自动利用局部性,因此硬件的自适应性 L1 缓存特性可以加速这些程序性能.

5.4　Ampere 适用性测试

为了测试本文模型在其他 GPU 架构上的效果,本文使用表 2 相同测试集,测试了模型在 NVIDIA A100 上的效果. A100 的硬件参数 SM 数量为 108,L1 大小最大为 196 KB,其余 Cache 参数和表 3 的 TITANV 架构类似.

采集 Trace 是模型最耗时的一步之一,不过由于模型采集 Trace 时按照线程块粒度划分保证了硬件无关性,模型支持使用一个 GPU 采集的 Trace 模拟另一个 GPU 的性能结果. 因此,本文复用了 TITANV 上采集的 Trace,并用它来预测 A100 GPU 的性能.

表 6　建模 NVIDIA A100 架构,PPT-GPU-Mem 和本文模型结果对比

Table 6　Comparison of PPT-GPU-Mem and our model results when modeling NVIDIA A100

Metrics	MAPE		Correlation		NRMSE	
	Old-	New	Old	New	Old	New
L1 Hit Rate	17.06%	15.12%	0.98	0.98	0.12	0.13
L1 Hit Rate (Global Load)	9.67%	6.72%	1.00	1.00	0.07	0.06
L2 Hit Rate	35.60%	**16.38%**	0.73	0.85	0.35	0.27
L2 Read Hit Rate	229.38%	**36.86%**	0.7	0.86	0.78	0.48
DRAM Total Reqs	47.56%	59.61%	0.86	**0.99**	1.53	**0.35**

最终结果如表 6 所示. 对于 L1 缓存,本文模型 L1 命中率误差为 15%,L1 全局加载命中率误差仅为 6.7%. 对于 L2 缓存,本文模型将 L2 命中率误差从 PPT-GPU-Mem 的 35.6% 降低至 16.38%,L2 读取命中率从 229.38% 降低为 36.86%. 对于显存事务数,在对异常点更不敏感的 NRMSE 评价指标上,本文模型将误差从 1.53 显著降低为 0.35,并且相关系数也提升到 0.99,展现了不错的硬件一致性.

综上,本文模型在 A100 同样有较好效果,且同样显著改善了原本 PPT-GPU-Mem 在 L2 和显存上的结果. 这充分说明本模型适用于其他 GPU 架构,有良好的适用性.

6　总结与展望

本文改进了目前最先进的 GPU 内存分析模型 PPT-GPU-Mem,基于 Cache 功能模拟,对 GPU 的 Sector Cache、自适应 L1 缓存、Cache 写策略等详细的微架构特性进行了建模,在 L2 缓存和显存事务数上显著提高了准确性. 本文使用改进的模型,对使用 Volta 架构的 TITANV GPU 进行了建模,在 37 个应用实例共 1736 个 Kernel 上对模型进行了充分验证. L1 命中率误差从 14.82% 降低为 11.06%,L2 命中率误差从 43.39% 降低到 15.86%,显存总事务数误差从 42.70% 降低到 16.85%,显存事务数的相关性大幅提高,均接近 1. 本文对做出的每一个改进进行了单独对比测试,验证了每一步改进的有效性. 最后,本文还验证了模型在 Ampere 架构上的效果,说明了模型的适用性. 本文的研究成果可用于 GPU Cache 参数的设计空间搜索,辅助 GPU 硬件设计. 同时,由于内存系统的建模对 GPU 性能模型的准确度至关重要,本文的工作可集成到现有 GPU 性能模型[15]中,提高 GPU 整体性能建模的准确性,这也是本文下一步的工作.

References:

[1] Dao T, Fu D Y, Ermon S, F, et al. FlashAttention: fast and memory-efficient exact attention with io-awareness [J]. arXiv. 2205.14135v2, 2022.

[2] Khairy M, Shen Z, Aamodt T M, et al. Accel-sim: an extensible simulation framework for validated GPU modeling [C]//ACM/IEEE 47th Annual International Symposium on Computer Architecture, 2020: 473-486.

[3] Lew J, Shah D A, Pati S, et al. Analyzing machine learning workloads using a detailed GPU simulator [C]//IEEE International Symposium on Performance Analysis of Systems and Software, 2019: 151-152.

[4] Gong X, Ubal R, Kaeli D. Multi2sim kepler: a detailed architectural GPU simulator [C]//IEEE International Symposium on Performance Analysis of Systems and Software, 2017: 269-278.

[5] Lee J, Ha Y, Lee S, et al. GCoM: a detailed GPU core model for accurate analytical modeling of modern GPUs [C]//Proceedings of the 49th Annual International Symposium on Computer Architecture, 2022: 424-436.

[6] Wang L, Jahre M, Adileho A, et al. MDM: the GPU memory divergence model [C]//53rd Annual IEEE/ACM International Symposium on Microarchitecture, 2020: 1009-1021.

[7] Huang J C, Lee J H, Kim H, et al. GPUMech: GPU performance modeling technique based on interval analysis [C]//47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014:

268-279.

[8] Arafa Y, Badawy A H, Chennupati G, et al. Fast, accurate, and scalable memory modeling of GPGPUs using reuse profiles[C]//Proceedings of the 34th ACM International Conference on Supercomputing, 2020: 1-12.

[9] Arafa Y, Chennupati G, Barai A, et al. GPUs cache performance estimation using reuse distance analysis[C]//IEEE 38th International Performance Computing and Communications Conference (IPCCC), 2019: 1-8.

[10] Kiani M, Rajabzadeh A. Efficient cache performance modeling in GPUs using reuse distance analysis[J]. ACM Transactions on Architecture and Code Optimization, 2018, 15(4): 1-24.

[11] Wang D, Xiao W. A reuse distance based performance analysis on GPU L1 data cache[C]//IEEE 35th International Performance Computing and Communications Conference (IPCCC), 2016: 1-8.

[12] Nugteren C, Braak G J van den, Corporaal H, et al. A detailed GPU cache model based on reuse distance theory[C]//IEEE 20th International Symposium on High Performance Computer Architecture (HPCA), 2014: 37-48.

[13] Agarwal A, Hennessy J, Horowitz M. An analytical cache model [J]. ACM Transactions on Computer Systems, 1989, 7(2): 184-215.

[14] Brehob M, Enbody R. An analytical model of locality and caching [M]. Michigan State University, Department of Computer Science and Engineering, 1999.

[15] Arafa Y, Badawy A H, Elwazir A, et al. Hybrid, scalable, trace-driven performance modeling of GPGPUs[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2021: 1-15.

[16] Brais H, Kalayappan R, Panda P R. A survey of cache simulators [J]. ACM Computing Surveys, 2020, 53(1): 1-32.

[17] Tang T, Yang X, Lin Y. Cache miss analysis for GPU programs based on stack distance profile[C]//31st International Conference on Distributed Computing Systems, 2011: 623-634.

[18] Villa O, Stephenson M, Nellans D, et al. NVBit: a dynamic binary instrumentation framework for NVIDIA GPUs[C]//52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019: 372-383.

[19] Khairy M, Akshay J, Aamodt T, et al. Exploring modern GPU memory system design challenges through accurate modeling [C]//ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA), 2020: 473-486.

[20] Che S, Boyer M, Meng J, et al. Rodinia: a benchmark suite for heterogeneous computing [C]//IEEE International Symposium on Workload Characterization (IISWC), 2009: 44-54.

[21] Grauer Gray S, Xu L, Searles R, et al. Auto-tuning a high-level language targeted to GPU codes[C]//Innovative Parallel Computing (InPar), 2012: 1-10.

[22] Che S, Beckmann B M, Reinhardt S K, et al. Pannotia: understanding irregular GPGPU graph applications[C]//IEEE International Symposium on Workload Characterization (IISWC), 2013: 185-195.

[23] Karki A, Keshava C P, Shivakumar S M, et al. Detailed characterization of deep neural networks on GPUs and FPGAs[C]//Proceedings of the 12th Workshop on General Purpose Processing Using GPUs, 2019: 12-21.

[24] Avalos Baddouh C, Khairy M, Green R N, et al. Principal kernel analysis: a tractable methodology to simulate scaled GPU workloads [C]//54th Annual IEEE/ACM International Symposium on Microarchitecture, 2021: 724-737.

[25] Naderan Tahan M, Seyyedaghaei H, Eeckhout L. Sieve: stratified GPU-compute workload sampling[C]//IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), 2023: 224-234.