计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2020.20200113
57(6):1191-1207,2020

通用图形处理器缓存子系统性能优化方法综述

张 军[1,2] 谢竟成[2] 沈凡凡[5] 谭 海[3] 汪吕蒙[4] 何炎祥[4]
[1](东华理工大学江西省放射性地学大数据技术工程实验室 南昌 330013)
[2](东华理工大学信息工程学院 南昌 330013)
[3](东华理工大学创新创业学院 南昌 330013)
[4](武汉大学计算机学院 武汉 430072)
[5](南京审计大学 南京 211815)
(zhangjun_whu@whu.edu.cn)

Performance Optimization of Cache Subsystem in General Purpose Graphics Processing Units: A Survey

Zhang Jun[1,2], Xie Jingcheng[2], Shen Fanfan[5], Tan Hai[3], Wang Lümeng[4], and He Yanxiang[4]
[1](*Jiangxi Engineering Laboratory on Radioactive Geoscience and Big Data Technology, Eastern China University of Technology, Nanchang* 330013)
[2](*College of Information Engineering, Eastern China University of Technology, Nanchang* 330013)
[3](*School of Innovation and Entrepreneurship, Eastern China University of Technology, Nanchang* 330013)
[4](*School of Computer Science, Wuhan University, Wuhan* 430072)
[5](*Nanjing Audit University, Nanjing* 211815)

Abstract With the development of process technology and the improvement of architecture, the parallel computing performance of GPGPU(general purpose graphics processing units) is updated a lot, which makes GPGPU applied more and more widely in the fields of high performance and high throughput. GPGPU can obtain high parallel computing performance, as it can hide the long latency incurred by the memory accesses via supporting thousands of concurrent threads. Due to the existance of irregular computation and memory access in some applications, the performance of the memory subsystem is affected a lot, especially the contention of the on-chip cache can become serious, and the performance of GPGPU can not be up to the maximum. Alleviating the contention and optimizing the performance of the on-chip cache have become one of the main solutions to the optimization of GPGPU. At present, the studies of the performance optimization of the on-chip cache focus on five aspects, including TLP(thread level parallelism) throttling, memory access reordering, data flux enhancement, LLC(last level cache) optimization, and new architecture design based on NVM(non-volatile memory). This paper mainly discusses the performance optimization research methods of the on-chip cache from these aspects. In the end, some interesting research fields of the on-chip cache

收稿日期:2020-02-27;**修回日期**:2020-04-13
基金项目:国家自然科学基金项目(61662002,61972293,61902189);江西省放射性地学大数据技术工程实验室项目(JELRGBDT201905);江苏省基础研究计划(自然科学基金)项目(BK20180821)
This work was supported by the National Natural Science Foundation of China (61662002, 61972293, 61902189), the Project of Jiangxi Engineering Laboratory on Radioactive Geoscience and Big Data Technology (JELRGBDT201905), the Natural Science Foundation of Jiangsu Province(BK20180821).

optimization in future are discussed. The contents of this paper have important significance on the research of the cache subsystem in GPGPU.

Key words general purpose graphics processing units (GPGPU); cache subsystem; performance optimization; latency hiding; cache contention

摘 要 随着工艺和制程技术的不断发展以及体系架构的日趋完善,通用图形处理器(general purpose graphics processing units, GPGPU)的并行计算能力得到了很大的提升,其在高性能、高吞吐量等通用计算应用场景的使用越来越广泛.GPGPU 通过支持大量线程的并发执行,可以较好地隐藏长延时访存操作,从而获得高并行计算能力.然而,GPGPU 在处理计算和访存不规则的应用时,其存储子系统的效率受到很大影响,尤其是片上缓存的争用情况尤为突出,难以及时提供计算操作所需的数据,使得 GPGPU 的高并行计算能力不能得到充分发挥.解决片上缓存的争用问题、优化缓存子系统的性能,是优化 GPGPU 性能的主要解决方案之一,也是目前研究 GPGPU 性能优化的主要热点之一.目前,针对 GPGPU 缓存子系统的性能优化研究主要集中在线程级并行度(thread level parallelism, TLP)调节、访存顺序调节、数据通量增强、最后一级缓存(last level cache, LLC)优化和基于非易失性存储(non-volatile memory, NVM)的 GPGPU 缓存新架构设计等 5 个方面.也从这 5 个方面重点分析讨论了目前主要的 GPGPU 缓存子系统性能优化方法,并在最后指出了未来 GPGPU 缓存子系统优化需要进一步探讨的问题,对 GPGPU 缓存子系统性能优化的研究有重要意义.

关键词 通用图形处理器;缓存子系统;性能优化;延迟隐藏;缓存争用

中图法分类号 TP303.1

图形处理器(graphics processing unit, GPU)最初是一种专门用于图像处理的微处理器,随着图像处理需求的不断提升,其图像处理能力也得到迅速提升.2006 年 12 月 4 日英伟达(NVIDIA)公司发布新一代显卡 Geforce 8800 时,首次在公开发布的技术参数中使用了流多处理器(streaming multi-processor, SM)的概念.自此统一的渲染部件颠覆了传统可编程部件组织的像素渲染管线(pixel pipelines)和顶点着色单元(vertex pipelines),形成了当今通用图形处理器(general purpose graphics processing units, GPGPU)[1]的基本雏形.随后,流式图形处理器为 GPU 在通用计算领域的发展奠定了基础.随着 CUDA 和 OpenCL 这些编程模型的出现,在 GPU 中设计并行程序进行通用计算已经不再困难.

在如今的高性能、高通量通用计算领域,GPGPU 已经得到极其广泛的应用,并正在日益扩大其应用范围.强大的并行计算能力和高能效比亦使 GPGPU 成为构建高性能计算系统的首选.2019 年 11 月公布的超算 Top500 中,位于榜首的 Summit[2] 由美国 IBM 公司制造,它使用近 28 000 块 NVIDIA Volta GPU,为其提供了 95%的算力.榜单的第 2 名 Sierra 也使用了 17 280 个 NVIDIA Tesla V100 GPU 构建异构计算平台.在 GPGPU 强大算力的帮助下,它们的计算能力均超越了我国排在榜单第 3 位的神威·太湖之光,尤其是 Summit 在理论计算能力上超越了神威·太湖之光计算能力的 60%①.

现代 GPGPU 支持单指令多数据流(single instruction multiple data, SIMD)[3-4]的执行模式.目前主流的 GPGPU 内部内置了大量的寄存器,其大小通常达到数 KB,能够很好地支持数以万计的并发线程的同时执行,因此,其执行模式又可以称为单指令多线程(single instruction multiple thread, SIMT)[5-6]执行模式.该执行模式下,执行同样任务的线程被组织在一起,以获得很高的线程级并行(thread level parallelism, TLP).

与 CPU 相比,GPGPU 内部的流水结构可以有效实现 3 个并行级别[7-8],即指令级并行(instruction level parallelism, ILP)、数据级并行(data level parallelism, DLP)与 TLP,从而很好地满足高性能、高吞吐量应用场景的计算需求.

通过设立大量的寄存器和片上缓存,GPGPU

① ISC19 世界超算大会发布第 53 届超算 TOP500 榜单:美国超算 Summit 蝉联世界超算冠军,https://www.top500.org/lists/2019/11/

支持成千上万高度并发线程的执行.如有线程出现长延时操作,则立即切换至其他活跃的并发线程执行,从而实现对长延时访存操作的隐藏.然而,如此多的并发线程可能会同时发出大量的访存请求,容易出现对片上缓存资源的争用现象.同时,不规则访存模式的存在使得并发线程的访存变得离散,并导致产生更多的访存请求,使得片上缓存资源的争用现象会进一步加剧,严重时甚至会产生访存"抖动"现象.这使得 GPGPU 缓存中的数据局部性受到破坏,影响整个存储子系统的访问效率.另外,片上的访存队列资源也因此会迅速用尽,造成部分线程后续的访存操作无法得到及时的服务而被阻塞.当 SM 产生空闲时,整个 GPGPU 的性能将会下降.因此,优化缓存子系统的性能,对保证 GPGPU 稳定发挥高计算性能具有重大意义.

采用更合理的线程调度策略、存储访问策略、更优秀的缓存子系统架构设计,可有效改善 GPGPU 缓存子系统的访问性能.有学者早在十年前就展开了能增强 GPGPU 缓存子系统性能的调度方法优化研究[9-10]和体系结构优化研究[11].本文从优化 TLP 调节、优化访存顺序、数据通量增强、LLC 优化、基于 NVM 的新型缓存架构设计等方面,重点分析并讨论近几年来国内外关于提升 GPGPU 缓存子系统性能的优化方法.

1　GPGPU 结构及相关概念

目前,主流 GPGPU 生产厂商主要有 NVIDIA、超微半导体(AMD)和英特尔(Intel),它们的产品有着相似的宏观结构.为统一论述,本文以 NVIDIA 公司发布的 GPGPU 体系结构为基础,并使用其术语进行问题的描述和分析.

1.1　GPGPU 线程的组织和调度

为了高效地管理和调度 GPGPU 中线程的执行,GPGPU 并发执行的线程被组织为线程网格、线程块、线程组 3 个层次结构[12-13],如图 1[13]所示.

GPGPU 中 32 个相邻的、执行同一条指令的多个线程组成一个线程组,称之为 warp,它是 GPGPU 调度任务并行执行的基本单位.在同一个 warp 中,所有线程均按照锁步方式执行.当某个 warp 中不同的线程执行不同的分支指令,需要该 warp 中所有的线程均执行完各自分支路径上的所有指令后,该 warp 才能继续向后执行.

多个相互协作的 warp 被组织成一个 CTA

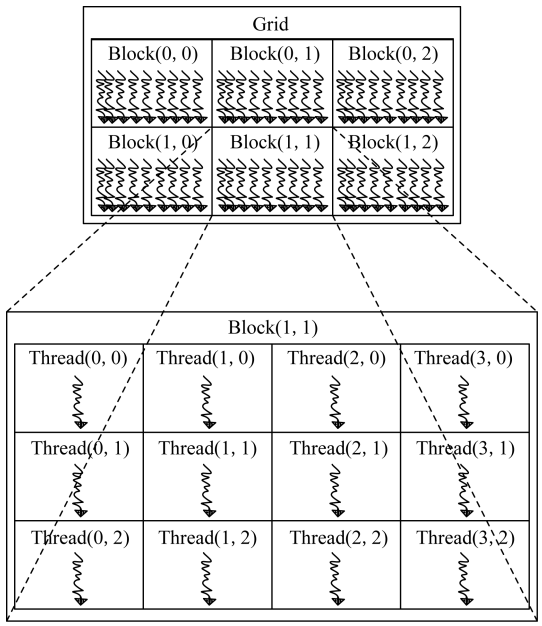

Fig. 1　Organization of threads in GPGPUs

图 1　GPGPU 中线程的组织

(collaborate thread array).同一个 CTA 中的线程之间可以相互通信.同时,CTA 是被分发调度的任务单位.通常情况下,CTA 通过轮转的方式被依次分配到各个 SM 中执行,每个 SM 独立对 CTA 中的数个 warp 调度执行.CTA 的大小通常由程序员指定,其规模一旦确定就不再变化.为了更好地组织、管理和调度线程,多个 CTA 又被组织成线程网格(thread grid, TG).

关于 GPGPU 的宏观和微观架构,已有诸多文献[14-19]详细描述,在此不再赘述.

1.2　缓存子系统的层次结构

通常,GPGPU 的缓存子系统主要由片上一级缓存(L1 cache)、二级缓存(L2 cache,由于主流的 GPGPU 中只设计了两级的片上缓存,因此又可以称为 LLC)和片上互联网络(network-on-chip, NoC)组成.如图 2 所示,每个 SM 都具有 4 种不同的片上一级存储器,包括可以存放 CTA 内共享数据的共享存储器(shared memory)、为本地访存服务的私有数据缓存(private data cache)、存放纹理数据的纹理缓存(texture cache)以及存放常量和参数的常量缓存(constant cache).其中,前两者是可读写的,后两者是只读的(在现有的 GPGPU 性能优化研究中,后者基本未被分析考虑).

正常情况下,GPGPU 上的数据缓存每周期至少能处理一个访问请求.当访问请求未命中时,通过有限规模的访问失效状态保留寄存器(missing

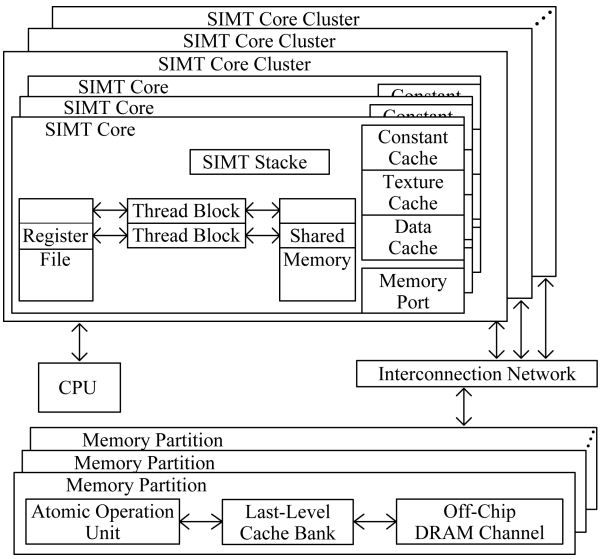

Fig. 2 The basic logical structure of GPGPUs

图 2 GPGPU 的基本逻辑结构

status hold register, MSHR)记录和合并对同一缓存行的失效请求,并据此向低一级的缓存进行访问.MSHR 被设计为一个包含多个表项的全相联寄存器,其中的每个表项服务一个缓存行的访问失效.

从图 2 可以看出,GPGPU 中的数个 SM 组成一个簇,并通过 NoC 连接到 LLC.NoC 只传递 4 种类型的数据:1)从 SM 发送到 LLC 的读请求和;2)写请求;3)从 LLC 发送到 SM 的读应答和;4)写确认.通过 NoC,可以实现片上一级缓存和二级缓存之间的数据访问.

2 缓存子系统性能优化方法

GPGPU 中的运算部件所需要的数据如果无法从寄存器中得到,则会直接从片上缓存子系统中申请访问.因此,缓存子系统的性能高低对 GPGPU 的性能影响很大.

GPGPU 缓存子系统性能的下降通常是由缓存资源竞争和访存行为不规则这 2 个主要原因引起.虽然 GPGPU 片上的缓存容量较大,但由于 GPGPU 上同时执行的并发线程数量过于庞大,使得 GPGPU 的缓存资源显得非常紧张.如果执行访存密集型应用程序,则很容易引起 GPGPU 片上资源的争用.另外,目前有很多应用在 GPGPU 上执行的过程中,即使是在同一个 warp 内部,也存在不同线程访问的数据属于不同的数据块,这种离散的访存行为进一步加剧了片上缓存资源的争用.

目前解决上述问题的常用方法主要包括:1)通过 TLP 调节技术,使得片上可以执行合适数量的并发线程;2)在 TLP 一定的条件下,通过调整线程的执行顺序,改变线程间的访存顺序,从而有效提升 GPGPU 缓存子系统隐藏访存长延迟的能力;3)通过数据通量增强技术,提高 NoC 的数据访问服务能力,从而提高服务密集访存行为的能力;4)提高 LLC 对片上一级缓存数据访问服务的能力;5)利用 NVM 新型存储材料存储密度高、读性能好的优势,优化 GPGPU 缓存子系统结构设计,可以大大提高 GPGPU 的片上缓存容量,从而更好地提升缓存子系统的数据访问效率.因此,本节将从上述 5 个方面展开论述.

2.1 TLP 调节技术

在一个 SM 中,只要有一个 warp 可以继续执行计算任务,其他线程的访存延迟就可以被有效隐藏.并行线程数量的上升将有利于提升 GPGPU 隐藏访存延迟的能力,并且更多的并发线程也可以更好地保持资源的高利用率.因此,GPGPU 应尽量维持高的 TLP.但是,当出现片上缓存资源的争用加剧时,应适当降低 TLP.

在现有针对 TLP 调节技术的研究中,更多的是倾向于基于不同类型任务的特点,充分利用 GPGPU 隐藏长延时访存操作的能力,既可以有效提升系统计算资源的利用率,又能使有限的缓存资源满足尽可能多并发线程的访存需求.

根据不同任务的计算/访存占比,大致可以将任务分为计算密集型任务和访存密集型任务 2 类[20].其中,计算密集型任务的计算操作次数相对访存操作次数更多,访存密集型任务则与计算密集型任务相反.对于访存密集型任务,如遇到缓存命中率低的情况,则会频繁地更新缓存(尤其是一级缓存)中的数据.

为了很好地分析说明 TLP 对 GPGPU 性能的影响,我们使用目前流行的 GPGPU 性能分析模拟器 GPGPU-Sim[19]开展了相关实验.实验中采用的基准体系架构基于 NVIDIA 的 Fermi[15,21]架构,基准测试程序包括 BFS,MUM,NN.图 3 描述了在每个 SM 上逐步增加可并发执行的 CTA 数量时这 3 个应用程序性能的变化情况.此处的性能用每周期指令数(instruction per cycle, IPC)表示.从图 3 可以看出,BFS 的性能随 TLP 的变化不大;MUM 的性能则是当 SM 中并发的 CTA 数量为 2 时最大,随着 CTA 数量的继续增加,其性能呈现不断下降的

趋势;NN 的性能则随着 TLP 的增大而不断提升.通过分析发现,MUM 的访存次数明显高于其他 2 个应用,TLP 的增加会加剧片上缓存资源的争用.

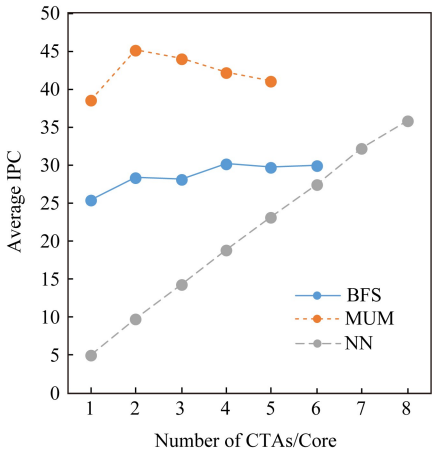

Fig. 3　Changes in average IPC with CTA number

图 3　平均 IPC 随 CTA 数目的变化

对于多应用并发执行的情形,应针对不同应用的特点,为不同的应用分配不同的资源.对于计算密集型任务,可以适当调高 TLP,让其能更好地利用计算资源.而对于访存密集型任务,应适当调低其 TLP,从而在保持高资源利用率的情况下,防止片上缓存资源被过度地争用.因此,此类问题的研究中,通常都倾向组合不同特性的任务并发执行,这样的技术被称为 CKE(concurrent kernel execution)技术.该技术也是目前 TLP 调节技术的研究热点.

目前,针对 GPGPU 缓存子系统性能提升的 TLP 调节技术主要从 CTA 和 warp 两个不同的粒度层次进行研究.这些研究主要集中在 3 个方面:1)增大 TLP;2)通过 cache 绕行技术,在不降低 TLP 的前提下避免稀缺资源拥塞;3)限制各并行任务的 TLP,以避免过度的资源争用.

2.1.1　提升 TLP 技术

正如图 3 所示,对于有的应用程序来说,增大计算任务的 TLP 可以带来更强的延迟隐藏能力,可以容忍更长的访存延迟和更密集的访存请求,这意味着可以增强对不规则访存行为的处理能力.

近几年,不少研究人员发现提升 TLP 可以有效提升 GPGPU 的性能,此类研究主要集中于 CKE 技术.

在 Fermi 架构之前,单一时刻一个 GPGPU 上只支持单 kernel 的运行,片上资源的利用率普遍较低,甚至可能造成某些硬件资源剩余.在推出 Fermi 架构后,可以利用片上的剩余资源运行其他的 kernel.较早的 CKE 技术[22-24]主要关注多 kernel 的并行性和公平性,但未注重片上缓存子系统的状态,因此 GPGPU 的整体性能难以达到最大.

为了使不同类型的 kernel 互补性地使用 SM 的片上资源,Xu 等人[25]和 Wang 等人[26]分别提出了 Warped-Slicer 及 SMK 算法,实现了对不同类型 kernel 的组合优化执行.在 SM 资源分配接近耗尽时,SMK 会将大量占用最稀缺资源的 CTA 换出,换入占用稀缺资源较少的、规模更大的 CTA,进一步提高 TLP.同时,SMK 使用的抢占技术需要在 GPGPU 中进行上下文切换,产生了大量的缓存数据交换[27],造成性能损失.Park 等人[28]提出精确控制开销的协同抢占方法 Chimera,采用了 CTA 的 flush、上下文切换和 drain 等 3 种抢占方式,有效降低了由于抢占带来的性能开销.Li 等人[29]提出提前保存状态的方法 PEP,也较好地减少了上下文切换的开销.Park 等人[30]则在 SMK 的基础上,与空间多 kernel 并发处理框架[22]相结合,使资源分区更合理,超过了 SMK 性能的 13.9%.此外,Liang 等人[31]提出了将不能充分利用的资源分配给其他已经正在运行的 kernel,提升了整体的运行速度.

CKE 技术通常会受到固定的资源分配方案和任务组织方案的限制,灵活调整这些规则,有利于提升片上资源利用率.分立结构的片上存储器各自固定不同层次的缓存容量,使用可调节的统一片上存储器[32]替代,可以根据任务需求调节不同类型存储器的容量,有利于将 SM 上的 CTA 数量分配到极限.但 SM 上并发的 CTA 数量是受到严格限制的,若所有并发 CTA 所使用的缓存资源都较少时,片上的缓存等资源将得不到充分利用,造成片上资源的浪费.Yoon 等人[33]提出了一种虚拟线程(virtual thread, VT)体系结构,将每个 SM 上并发执行的 CTA 数量分配至硬件容量的极限,并将这些并发的 CTA 置为活动和非活动状态,活动并发执行的 CTA 数量仍然符合硬件物理限制.当处于活动状态的某个 CTA 中的所有 warp 到达一个较长的访存延迟时,该 CTA 的状态变为不活跃,下一个就绪的处于不活跃状态的 CTA 将取代它,从而有效地保持缓存资源的利用率.相比 SMK,VT 避免了保存和恢复块状态以及交换大量缓存数据所产生的较大开销.此外,与 VT 相似的 VTB 技术[34]将 2 个 CTA 合为一个大的 CTA,考虑到 shared memory 资源访问的压力,只有在 CTA 中的 warp 访问 shared memory 时才予以分配 shared memory,一旦使用完毕,则立即释放对 shared memory 的使用.

在提升 TLP 水平的同时,还要使并发执行的线程保持高度活跃的状态,才能保持片上资源的高利用率.Kim 等人[35]提出 warp 预执行,对处在长延迟访存操作的 warp 继续对后续指令进行获取和解码,识别并预先执行不依赖长延迟访存操作的指令,能够更好地保持处于活跃状态的线程数量.Xiang 等人[36]指出资源的分配和回收若总是采用 CTA 级,它所占用的全部资源则需等到最后的 warp 之行结束才能回收,其占用的缓存在第一时间未得到及时释放.因此,他们提出了 warp 级的资源分配回收机制,使得更多的 warp 得到提前执行的机会,可以更大限度地提升片上资源的利用率.

表 1 对比了 5 种提升 TLP 技术的主要特征.从表 1 可以看出,Park 等人提出的 Maestro 多任务调度框架从空间和时间上同时考虑多任务调度资源分配的合理性,使得其资源的利用率和性能收益在这 5 个方法中最好.Warped-Slicer 方法主要是使用了更小的任务调度单位,使得资源利用率得到了较好的提升.但是,由于需要 CTA 的换入换出,其性能收益也不高.SMK 由于需要在上下文之间切换,开销很大,其性能收益相对最低.

Table 1 Features Comparison of the Main Strategies with TLP Enhancing

表 1 提升 TLP 的主要策略的特点对比

Strategy	Resource Utilization	Realizing Complexity	Performance Gains
Warped-Slicer[25]	High	Low	Low
SMK[26]	General	Higher	Lower
GPU Maestro[30]	High	High	High
VT[33]	General	Low	Low
VTB[34]	General	Low	Low

特此说明,各表中程度由低到高依次表示为:Lower<Low<General<High<Higher.

2.1.2 保持 TLP 技术

提升 TLP 可以有效地保持甚至提高片上资源的利用率.但是,TLP 的提升也可能带来更激烈的缓存争用,甚至会造成无法迅速解除的访存阻塞.cache 绕行技术可在缓存出现访问失效时,将访问下一级存储器所获取的数据直接传送给寄存器.因此,在出现片上缓存资源激烈争用时,采用 cache 绕行技术既可以有效保持 SM 上的 TLP 及 NoC 等片上其他资源的利用率,又能有效避免片上缓存资源争用的加剧.

由于采用 cache 绕行访问得到的数据保存在寄存器中,其获得的数据可重用性很低.因此,对低重用的数据访问更适合采取 cache 绕行技术.及时分析掌握运行时的访存行为特征,可以动态地找出绕行缓存的合适时机.Jia 等人[37]提出内存请求优先级缓冲(MRPB)策略,通过区分请求的来源分析预测为其提供服务是否会导致缓存抖动或阻塞,并判断相应的访存请求是否绕行 cache.但是,MRPB 并未真正分析访问请求的数据是否被重用,也即并未根据请求访问数据的特征来选择被绕行的对象.Koo 等人[38]提出访问模式感知的缓存调度架构 APCM,则依据 warp load 指令的特征识别低重用的数据,并选择需要绕行 cache 的访存请求.Lee 等人[39]提出 Ctrl-C cache 绕行框架,通过访存指令感知算法更细致地识别每条指令的缓存重用行为,并做出 cache 绕行决策.Li 等人[40]和 Chen 等人[41]提出的 cache 绕行策略则主要考虑了访存数据的重用距离特征.不同于预测重用,Dai 等人[42]提出通过预测是否能命中缓存来决定是否绕行,其实验结果表明,该方法优于 Chen 等人[41]提出的 cache 绕行策略.与直接识别数据的重用特征不同,Liang 等人[43]在并行的多任务中,以 CTA 为单位采取绕行,通过对增加绕行块带来的效果进行采样分析,逐步找到最合适的绕行方案.

也有学者通过编译技术静态分析访存的行为特征,可以提前识别是否适合采取 cache 绕行的访存操作.Liang 和 Xie 等人[44-48]分别提出了一种编译时框架和一种编译运行时相结合的缓存绕行框架.后者在编译时识别局部性好和局部性差的访存指令,并标识需要 cache 绕行的 load 指令,其余的访存指令则在运行时分析处理.

有一类应用程序,它们在运行过程中使用的运算数据绝大部分都只被访问一次,其访问的数据通常被称为流式数据.流式数据的局部性非常差,因此对这种类型的数据访问非常适合采用 cache 绕行技术.Choi 等人[49]提出了一种针对于读取操作的 cache 绕行方法,避免没有重用特征的流式数据进入 LLC.Tian 等人[50]提出的自适应 cache 绕行策略,有效预测并避免流式数据进入一级缓存.

此外,有一些综合性的缓存管理方案[51-52]也有效结合了绕行技术.

表 2 对比了 6 种主要的使用 cache 绕行技术的优化策略的特点.从表 2 中可以发现,利用数据的重用特征选择合适的绕行 cache 对象是提高 cache 绕

行效率的重要方式.其中,MRPB 和 TLP modulation and cache bypassing 未采用重用特征提取机制,其性能收益相对一般.APCM 精细地利用了每个 load 指令的局部性特征,获得了较高的性能收益.Liang 和 Xie 等人利用编译时预测和运行时感知的重用特征提取机制实现的 Coordinated static and dynamic cache bypassing 缓存绕行策略获得了不错的性能收益.

Table 2　Features Comparison of the Main Strategies with Cache Bypassing

表 2　使用 cache 绕行的主要策略的特点对比

Strategy	Reordering Requests	Extracting Reuse Features	Realizing Complexity	Performance Gains
MRPB[37]	Needed	No Need	Low	General
CBWT[41]	No Need	Run-time Forecast	Higher	High
Dynamic GPU Cache Bypassing [50]	No Need	Run-time forecast	General	Low
Coordinated Static and Dynamic Cache Bypassing[45]	No Need	Compile-time Forecast and Run-time Awareness	High	Higher
APCM[38]	No Need	Run-time Awareness	Higher	Higher
TLP Modulation and Cache Bypassing[43]	No Need	No Need	High	General

2.1.3　TLP 限制策略

随着 SM 上的 TLP 逐渐升高,缓存资源也逐渐变得紧张.若在硬件资源的限制范围内继续提升 TLP,缓存争用将加剧,甚至出现缓存访问“抖动”,直至出现访存阻塞.此时缓存子系统性能出现严重损失,而 TLP 的升高带来的隐藏访存延迟的好处不足以弥补这一损失,GPGPU 总体性能将会受到下降的影响.因此,在缓存资源竞争加剧的情况下适当限制 TLP,可以有效防止 GPGPU 的性能下降.

通过前面对图 3 的分析可以知道,有些任务运行时因大量消耗其他硬件资源只能产生有限的 TLP,TLP 即使达到物理上限,也不足以使缓存产生抖动;有些任务虽可继续增加 TLP,但由于其占用的片上资源过多,继续提升 TLP,反而导致系统性能下降.Bakhoda 等人[19]观察到一些任务通过降低 TLP 减少了缓存争用,并提高了系统性能.

Kayıran 等人[20]提出了一种动态调节各个 SM 上并发 CTA 数量的方法.当执行部件因缓存阻塞空闲时,通过暂停一些 CTA 中 warp 执行减少对缓存的争用.与他们不同的是,Rogers 等人[53]从 warp 细粒度的角度提出 CCWS TLP 调节方案.通过对 warp 局部性丢失情况的动态分析,主动限制并发 warp 的数量,以减少对 L1 缓存行的访问争用.不同于 CCWS 通过检测缓存失效被动调节 TLP 的方法,Rogers 等人通过动态分析片上缓存占用是否超过缓存资源容量上限,进一步提出了限制 TLP 的方法 DAWS[54],有效避免缓存访问的抖动.与 CCWS 的思想类似,Oh 等人[55]和 Wang 等人[56]通过限制片上并发执行的 CTA 数量,减少了片上缓存争用.Chen 等人[41]则通过运行时检测争用和访存拥塞状态,提出了一种融合限制 TLP 和 cache 绕行技术的 GPGPU 缓存管理策略,动态地控制活跃的 warp 数量.

另外,Narasiman 等人[57]提出的两级 warp 调度策略将 SM 上所有活跃的 warp 分为 2 组.同一时刻只有一组 warp 执行,当活跃 warp 组中所有的 warp 被阻塞时,另一组 warp 才被调度执行.两级调度策略使得活跃的并发线程数减少了一半,因此片上缓存的争用将得到有效的降低.与他们稍有不同的是,Jog 等人[58]提出的线程调度策略 OWL 以 CTA 为粒度进行两级调度.

当前,多任务并发执行已成为 GPGPU 性能优化研究的热点.多任务并发执行无疑会大大增加线程对片上缓存资源的争用.动态分析不同任务的执行特点,实时调节不同任务对不同资源的使用度,可以实现不同执行任务对各种硬件资源进行互补性地使用.尤其是对片上资源占用率高的执行任务,应动态限制其并发执行的线程数量,并通过利用限制 TLP 调节技术,有效减少片上资源的访问争用情形.Dublish 等人[59]针对多任务情形下的缓存争用,提出了线程调度策略 Poise,利用了机器学习的方法,分析出不同特征任务运行时的最佳 TLP,取得了较好的性能效果.

另外,一些综合性的缓存管理方案[51-52]和线程调度方法[60]也有效结合了 TLP 限制策略.表 3 归纳了 4 种 TLP 限制策略的主要特征.其中,CCWS 和

Poise 都是在运行时对数据的重用特征进行被动感知,获得的性能提升较为有限.DAWS 和 CBWT 则是在运行时主动预测重用特征,其准确性更佳,因此获得了相对较好的性能提升.

Table 3 Features Comparison of the Main TLP Limitation Strategies

表 3 几种 TLP 限制策略的特点对比

Strategy	Performance Gains	Difficulty	Hotspot Data Protection	Extracting Reuse Features
CCWS[53]	Low	Low	Adopted	Run-time Awareness
DAWS[54]	High	General	Adopted	Run-time Forecast
CBWT[41]	High	High	Adopted	Run-time Forecast
Poise[59]	Low	Higher	Adopted	Compile-time Classification and Run-time Awareness

2.2 访存顺序调节技术

对 GPGPU 缓存子系统来说,增强访存延迟的隐藏能力是改善访存效率的一种重要方式.除了使用 TLP 调节技术,还可以通过访存顺序调节技术增强隐藏访存延迟的能力,而预取和访存请求重排序正是 2 种主要的访存顺序调节技术.

2.2.1 预取

预取技术利用当前 warp 的访存操作,一并将该 warp 或其他 warp 后续访存的数据提前读取到片上缓存中,其本质上是改变了后续访存操作的顺序.通过预取,一方面可以有效降低 cache 的强制失效率,避免过多的计算停顿;另一方面,由于一次访存操作可以满足当前和后续多次访存请求的需要,整体上也降低了对片外访存的开销.因此,预取技术在某种程度上可以有效提升存储子系统的访问效率.

预取面临准确性和及时性两大主要问题,这两大问题可以通过准确计算预取地址和选择合适的预取时机来解决.

如果预取地址计算错误,该失效的预取操作会带来缓存资源争用的压力.有些应用的访存请求体现出了明显的规律,例如大量使用线程 ID 引用内存地址[9]、运行相同代码的线程具备相似的访存特征[61]等.这些规律方便对访存预取的步长进行分析,从而可以准确计算预取数据的地址.

对于计算比较规则的应用来说,预取策略可以实现非常高的准确度.Lee 等人[9]提出了多线程感知预取的策略,利用应用程序中大量使用线程 ID 访问内存地址的特点,为其他线程预取所需数据,并自适应地限制预取,以避免预取带来过大的缓存压力.Oh 等人[61]提出了 WASP 的预取机制,在了解了相关 warp 之间的访存差异后,选择执行相同访存指令且比当前 warp 执行速度慢的 warp 进行数据预取.Koo 等人[62]提出的 CAPS 预取策略,则使用了每个 CTA 的基地址来计算后续 warp 的预取偏移量,其预取准确率超过了 97%,并通过定期发出访存请求的方式,获取了更多合并访存请求的机会,可为大部分任务带来平均 8%的性能提升.

然而,预取到缓存中的数据不一定能立即被传送到寄存器参与运算,通常需要等待任务执行到相应的指令时才能被访问,这样则有可能被后续的访存数据置换出去.因此,预取的时机选取很重要.预取过早,对应需求的任务还未被调度执行,可能导致预取访问的数据还未被使用就被置换掉.预取过晚,任务发出访存请求时,数据未被及时预取至缓存中,导致相应任务访问 cache 失效.

为连续 warp 执行预取时,由于片外访存的延时相对较大,预取的时机往往过晚.Jog 等人[63]提出了预取感知的 warp 调度策略,在两级 warp 调度的基础上,为不连续的 warp 选择合适的预取时机,使得系统平均性能提高了 7%.Caragea 等人[10]提出的数据预取机制 RAP,可根据资源使用情况动态调整预取距离,使资源紧张状态下的预取更为有效.与他们不同的是,Oh 等人[64]提出的 warp 调度框架 APRES 将访存特征相似的 warp 成组调度,若组中的第 1 个 warp 访问 cache 失效,则由该 warp 为全组其他的 warp 进行数据预取.

此外,Jog 等人[58]提出的数据预取机制中重点考虑如何提高片上二级缓存的命中率.Sethia 等人[65]提出的自适应预取技术则重点考虑了如何提高 GPGPU 的能效.

表 4 对主要的缓存预取策略进行了对比.其中,CAPS,WASP 和 MT-prefetching 预取准确度较高,因此获得了较好的性能收益.RAP 虽然实现了较为复杂的软件预取,但只考虑了预取步长因素,其预取准确度较低,由此获得的性能收益相对较低.

Table 4　Features Comparison of the Main Strategies with Prefetching

表 4　主要的应用数据预取策略的特征对比

Strategy	Difficulty	Performance Gains	Data Target	Accuracy	Dependence
MT-prefetching[9]	High	High	Other Threads	High	Stride, Prefetch Degree and Hardware Status
PA[63]	Lower	Low	Other Threads	Low	Strideand Prefetch Degree
APRES[64]	Low	Low	Self	General	Stride and Prefetch Degree
WASP[61]	General	High	Self	High	Base Address, Stride and Warp Status
CAPS[62]	General	High	Other Threads	High	Base Address and Stride
RAP[10]	General	Lower	Self	Low	Stride

2.2.2　访存请求重排序

当计算所需的数据无法命中片上 cache 时,将产生片外访存请求.由于片外访存的延时大大超过缓存提供数据服务的时间,因此应尽可能减少片外访存次数.对此,GPGPU 实现了两级访存合并机制,即 warp 内的访存合并和 warp 间的访存合并.

当片上缓存访问失效增多时,warp 的执行性能必然受到影响.同时,片外访存请求会相应增加.根据每个片外访存的特点,适当调整发出片外访存的顺序,及时满足需求量大的访存请求或局部性好的访存请求,可以有效提升及时服务后续计算任务的能力,访存延迟也可以更有效地隐藏在计算过程之中.

通常情况下,CTA 中所有的 warp 运行结束后,才能给 SM 调度分配新的 CTA 执行.但是,CTA 中有些 warp 的执行速度比较慢,会拖累整个 CTA 的执行,可以把它们称为 CTA 中的关键 warp.因此,应优先服务 CTA 中关键 warp 的访存请求.Lee 等人[66]提出的线程调度 CAWS,就是优先服务 CTA 中的关键 warp,使其利用缓存的机会大大增加,加速了关键 warp 的执行,从而也加速了整个 CTA 的执行.为了提升 CAWS 的适应性,Lee 等人又联合 Arunkumar[67]提出了 CAWA 线程调度策略,将线程调度与缓存使用的优先级进行了结合.

通过对访存请求重排序,可以增加访存请求的合并机会,并减少对下一级存储器的访问,从而降低 GPGPU 缓存子系统的访存压力.Jia 等人[37]提出了一种内存请求优先级缓冲(MRPB)的硬件结构,采用数个队列对访存请求重新排序,将来自同一 warp 的访存请求分组,很好地开发了 warp 内的数据局部性.Kloosterman 等人[68]提出 WarpPool 合并来自多个 warp 的缓存请求,开发了 warp 间的数据局部性,也有效减少了多次发出同一访存请求的概率.与他们的思想不同的是,Sethia 等人[69]提出了基于访存感知的线程调度策略 Mascar,当片上访存排队资源处于饱和状态时,通过打破严格按照队列次序进行访存服务的限制,优先服务缓存可以满足数据访问请求的 warp,提升了缓存中数据的重用能力.

另外,每一种线程调度方式都有自己的局限性和优势,例如对于具有 warp 间局部性的工作负载,公平的轮转调度器性能要比贪婪的 warp 调度器更优秀.为此,Lee 等人[70]提出了 iPAWS,根据 warp 的指令发射模式在 2 种调度器之间进行动态适配,实质上也是对 warp 间的访存顺序进行动态调整.此外,对各个 SM 轮转分配 CTA 的调度方法在均衡硬件负载的同时,也可能破坏了连续 block 之间存在的局部性.为此,Lee 等人[71]提出的线程调度策略 BCS,尽量将 2 个连续的 CTA 分配给同一个 SM,使连续 warp 的访存数据可以更好地重用,开发了 CTA 间的数据局部性.

表 5 对比了主要的访存请求重排序策略的特征.其中,BC 的优化粒度是线程块级的,获得访存请求合并的机会最少,因此性能收益相对最低.iPAWS 通过对访存指令 Load 模式的分析,动态调正 warp 调度策略,获得了较高的性能收益.MRPB 通过对访存请求进行重排序,开发了 warp 内的数据局部性,并结合缓存绕行机制,也获得了较高的性能收益.

Table 5　Features Comparison of the Main Strategies with Memory Accesses Reordering

表 5　主要的访存请求重排序策略的特征对比

Strategy	Difficulty	Performance Gains	Latency Hiding	Grain
MRPB[37]	General	Higher	Reorder Requests	Warp
Mascar[69]	Lower	General	Reorder Requests	Warp
CAWS[66]	General	General	Reorder Requests	Warp
CAWA[67]	High	High	Reorder Requests	Warp
iPAWS[70]	Low	Higher	Reorder Requests	Warp
WarpPool[68]	Lower	General	Merge Requests	Warp
BC[71]	Lower	Low	Merge Requests	Block

2.3 数据通量增强技术

随着 GPGPU 的计算性能逐渐增强,SM 可以支持更高的 TLP,运算所需的数据量大大增加,这要求缓存子系统能够承载更高的数据通量.然而对 GPGPU 缓存子系统而言,简单增加其容量,会使硬件规模急剧上升.为此,研究者们提出了数据通量增强的技术,利用更优秀的结构改进缓存子系统,实现了更高的访存效率.

NoC 在缓存子系统中担负着传递数据的任务,增加单位时间内数据传输的通量有助于应对更密集的访存请求.

通常情况下,数据的请求和应答都在同一网络上传输.Kim 等人[72]提出 DA2mesh 网络架构,将访存请求和访存应答在不同的片上网络上传输,使得系统整体性能提升了 36%,同时片上网络的能耗降低了 15%.在实际的片上网络中,访存请求的数据量明显低于访存数据回传的数据量.Jang 等人[73]采用不对称片上网络的设计,较好地提升了片上缓存子系统的数据访问效率,使得系统性能提高了 25%.另外,Cheng 等人[74]试图在不增加规模的情况下最大化不对称 NoC 设计的性能.Ziabari 等人[75]对几种不对称的 NoC 设计进行了比较.

来自各个 SM 的访存请求中,可能存在访问同一行 LLC 的、可以被进一步合并的访存请求.Zhao 等人[76]提出簇内合并访存请求的策略,降低了 NoC 的通信压力,提升了缓存子系统承载高数据通量的能力.Zhao 等人还提出了 2 种二级路由设计[77-78],缩减了 NoC 的总体规模.

表 6 对比了 7 种主要的 NoC 优化方案的特点.其中,Zhao 等人[77]的方法还涉及对 LLC 的优化.一些为进一步降低 NoC 成本的方案[11,72]试图在资源分配更少的情况下最小化性能丢失,而 Zhao 等人[79]试图在减小功耗的情况下保持性能.

从表 6 可以看出,将访存请求和访存应答在不同的片上网络上传输,对提升 NoC 的性能较为有利,这是因为可以针对 2 个网络所传数据的不同特点进行优化,DA2mesh NoC 和 Packet Pump NoC 都因此获得了相对较高的性能收益.而采用二级路由的 NoC 设计方案利于带来较好的能效收益,Adaptive LLC 和 CD-Xbar 都因此受益.另外,Adaptive LLC 可以动态地绕行靠近 LLC 的路由结构,可以获得比 CD-Xbar 更高效的访问速度.

Table 6 Features of Seven NoC Optimization Strategies

表 6 7 种 NoC 优化策略的特征

Strategy	Major Improvement	Performance Gains	Energy Efficiency Gains
DA2mesh[72]	Performance and Energy Efficiency	High	Low
VC Monopolizing and Partitioning[73]	Performance	General	Normal
cfNoC[79]	Energy Efficiency	Normal	General
Packet Pump NoC[74]	Performance	High	General
ICC[76]	Performance and Energy Efficiency	Low	Lower
Adaptive LLC [77]	Performance and Energy Efficiency	General	High
CD-Xbar[78]	Performance and Energy Efficiency	Low	Higher

2.4 针对 LLC 的优化

与目前主流的 CPU 不同的是,目前主流的 GPGPU 通常只包含二级片上缓存.因此,GPGPU 中的 L2 缓存也即 LLC.

由于数据在 LLC 中只存在唯一的一份,LLC 一次不能服务访问对同一数据的多个请求,因此,这些访存请求只能串行执行.Zhao 等人[77]提出的方法可以动态改变 LLC 的状态,支持将集中访问的数据复制多份,并放在不同的位置同时进行访存,从而实现了对这些数据串行访问的并行化,提高了对 LLC 的访问效率.

保护 LLC 中数据的局部性同样可以提高访存效率.Choi 等人[49]提出了一种读取绕行的策略,可以有效避免局部性差的流数据进入 LLC.Mu 等人[80]提出的缓存管理方案也考虑提高 LLC 重用数据的驻留机会.Dublish[81]提出将更多的 LLC 带宽提供给那些片上缓存争用不激烈的 SM.为了减小 LLC 的带宽压力,Dublish 等人[82]又提出将一级缓存通过轻量级环形网络连接以支持数据共享,减少了 29% 的 LLC 流量,这些带宽可以用于服务其他的访存请求.Candel 等人[83]提出在 LLC 中加入类缓存的缓冲结构(FRC, Fetch and Replacement Cache-like structure),延缓 LLC 中被置换数据的换出,提高了

这部分数据的访问命中率.此外,为了打破有关数据局部性利用方法的层次局限,Vijaykumar 等人[84]协调 NUMA 与 CTA 调度,将访问同一批数据的 CTA 集中在同一个 SM 运行.

另外,一些研究者们还谋求将少量计算单元集成在较低层次的存储器件(如主存[85])中,避免移动低计算需求的数据集,减少了 SM 对 LLC 的访存数量.类似地,Pattnaik 等人[86]提出的 NDP 解决方案将少量计算单元集成在 LLC 中,也减少 SM 与 LLC 之间的数据传输.与基准 GPU 相比,该策略减少了 44%的片上数据移动,提供了平均 31%的性能提升和 16%的能效改进.

表 7 比较了 5 种 LLC 性能改进策略的特点.其中,Mu 等人基于对重用数据的保护提高 LLC 性能,由于 LLC 中的数据重用特征会受到数据频繁替换的影响,因此获得的性能提升相对有限.而 FRC 利用添加类似于缓存的结构,减小了数据频繁替换的影响.虽然增加了硬件开销,但是 FRC 换取了良好的重用特征提取能力,因此获得了相对更好的性能提升.

Table 7　Five Features of LLC Performance Improvement Strategies

表 7　5 种 LLC 性能改进策略的特点

Strategy	Key Technologies	Hotspot Data Protection	Complexity	Performance Gains
Adaptive LLC[77]	Access Parallelization	No	High	General
Selective Cache Management Scheme[49]	Read Bypassing	Yes	General	High
Priority-based Cache Management[80]	Reuse Data Protection	Yes	General	Lower
Slack-aware DRAM Scheduling[81]	Bandwidth Throttle	No	Low	Low
FRC[83]	Fetch and Replacement Cache-like Structure	Yes	General	Higher

2.5　NVM 在 GPGPU 缓存子系统中的应用

采用传统 SRAM 设计的缓存子系统规模和功耗都很大,且容量小.随着 NVM(non-volatile memory)非易失性存储器件的广泛应用,为 GPGPU 上的缓存系统架构设计注入了新的活力.NVM 具有更高存储密度和良好的读数据性能,但是其存在着写功耗大、写速度慢、寿命相对有限的缺陷.

随着研究人员的不断努力,NVM 的性能缺陷和寿命缺陷有了很大改善,研究人员试图将 NVM 融入到 GPGPU 上的缓存子系统设计架构中,并逐渐形成了 NVM 混合传统缓存材料、完全使用 NVM 材料构建缓存子系统的两大发展方向.表 8 列举了近几年 5 种具有代表性的应用于 GPGPU 缓存的 NVM 材料的特性,并与传统存储材料作了对比.

Table 8　Five Features of NVM Material Used on GPGPU Cache

表 8　5 种使用在 GPGPU 缓存上的 NVM 材料的特性

Characteristics	SRAM	DRAM	STT-RAM	RRAM	MRAM
Cell Size/F^2	120-200	6	6-50	4-10	45
Endurance(read/write times)	10^{16}	10^{15}	10^{15}	10^{8}	10^{15}
Read Latency	Very Fast	Fast	Fast	Fast	Fast
Write Latency	Very Fast	Fast	Low	Fast	Fast
Leakage Power	High	High	Low	Low	Low
Dynamic Power(read/write)	Low	High	Low/High	Low/High	Low
Non-Volatile	No	No	Yes	Yes	Yes
Applicable	Cache	Main Memory	Cache	Main Memory	Cache/Main Memory

多数 NVM 材料在写入速度方面达不到传统材料的水平,但在读取速度方面十分接近传统材料.对此,研究者们考虑在传统缓存中混合使用 NVM,利用 NVM 存储读取频次更高的数据,发挥其读取速度的优势;同时使用传统材料缓存,为写入次数较多的数据提供缓存,平衡 NVM 写入寿命和写入速度的缺陷.设计这样的混合缓存,需要充分考虑使用的 NVM 种类、NVM 缓存的容量比例等因素,满足缓存在整体规模、寿命、容量和功耗等方面的要求.

Goswami 等人[87]提出将 STT-MRAM 作为只读

存储器件融入到SRAM缓存系统中,充分利用了STT-MRAM较好的读性能,显著改善了shared memory的只读数据访问性能.Zhang等人[88]提出的SRAM与STT-MRAM混合的缓存结构,通过在运行时识别数据在读写次数方面的特征,预测和推断需要缓存的数据应该使用哪种材料的缓存.

采用NVM与传统材料混合的解决方案较好地平衡了NVM的优势与缺陷.然而,完全采用NVM构建缓存可以更好地利用其高密度的优势,其进一步扩大的缓存空间带来的不只是更长的数据保持时间和更小的缓存抖动,还有利于支撑更高的TLP.

Satyamoorthy等人[89]提出了一种在固定规模和功率约束下用MRAM代替SRAM shared memory的设计,并通过设计写缓冲区隐藏写入延迟.Zhang等人[90]提出使用电阻随机存取存储器(RRAM)取代基于SRAM的LLC缓存,将片上缓存的空间扩大了30倍,很好地缓解了LLC的抖动问题.Samavatian等人[91]设计了一种基于STT-RAM的LLC,也带来了系统16%的平均性能增幅.

传统SRAM的替代方案之中,DWM是极少数在读写性能上近乎持平于传统存储材料的电子自旋式存储技术.Venkatesan等人[92]提出了一种新的片上缓存架构STAG,首次尝试以DWM组织GPGPU的片上存储.DWM通过访问晶体管共享电路实现了数据存储的高密度,但由于对其访问需要做出类似移动磁带带头的动作,也会带来一定的访问延迟.

另外,NVM技术也可独立用在更高层次的寄存器中.基于STT-RAM的寄存器[93-94]和混合寄存器[95]均已被提出.Mittal等人[96]更是提出了一种基于SOT-RAM的寄存器设计方案,它在保持与SRAM寄存器相同性能的同时,比SRAM和STT-RAM寄存器提供了更高的能效比.Gebhart等人[32]提出的统一片上存储器结构将存储器静态地划分为寄存器和L1数据缓存.Jing等人[97]提出将寄存器和数据缓存融合在一起,通过对寄存器地址转换实现缓存仿真寄存器的功能,具有统一寻址和管理策略,大大提高了访存敏感型任务的性能.

表9对比了应用于GPGPU缓存组织架构的6种主要NVM技术的特征.从表9可以看出,通过传统材料混合NVM的缓存组织方式能获得较佳的性能提升.在STAG中,DWM代替了传统的cache,虽然DWM材料存储密度高,有利于增加片上缓存的容量,但是其访问时需要移动磁带带头,因此其带来的性能提升相对较低.

Table 9 Features Comparison of Several NVM Technologies Used on GPGPU Cache

表9 几种GPGPU缓存上使用的NVM技术的特征对比

Strategy	Applied Memory Technology	Energy Efficiency Gains	Performance Gains
STT-MRAM Read-only Cache[87]	Shared Memory: SRAM+STT-RAM	Lower	High
FUSE[88]	L1D Cache: SRAM+STT-RAM	General	Higher
MRAM Shared Memory[89]	Shared Memory: MRAM	Normal	High
3D Resistive Cache[90]	LLC: RRAM	High	General
Efficient STT-RAM LLC[91]	LLC: STT-RAM	Low	General
STAG[92]	Cache: DWM	Higher	Low

3 结束语

研究者们提出的不同缓存子系统性能优化方法均在一定程度上提升了GPGPU缓存子系统的效率,有利于GPGPU的性能提升.但由于各种方法考虑问题角度的局限性,缓存子系统的性能潜力仍未得到充分开发.

GPGPU的不断发展使得缓存子系统性能优化方法在5方面还存在挑战:

1) 随着单GPU计算能力的不断提升以及GPU集群技术的不断发展,GPU上支持的并发kernel数的增长速度远远超过缓存容量的增长速度,片上缓存资源争用问题愈发突出.对此,需对高并发的多个计算任务进行更有效地调度,并有效结合编译技术,从时间和空间上更好地开发缓存中数据的局部性,有效解决片上缓存争用问题.

2) 由于写性能、功耗以及寿命方面的缺陷,传统缓存的替代方案依然存在不足,部分材料的读性能和寿命虽然能与传统材料相媲美,但大大牺牲了其存储密度和功耗方面的优势.在此方面的研究中,一方面可以开发密度更高、功耗及读写性能与传统

缓存相当甚至超越的新型存储材料,以全面替代现有的传统存储器件;另一方面,针对传统材料和新型材料融合的片上缓存子系统解决方案,需要开发更有效的任务调度算法,并有效结合编译技术,更好地利用传统存储材料和新型存储材料各自的优势,在保证性能的前提下,提高存储密度和使用寿命,同时有效降低片上存储子系统的功耗.

3) 目前,为了更有效地解决 CPU 与 GPGPU 之间的数据传输长延时问题,不少研究提出 CPU 和 GPGPU 融合的体系架构,且目前已有相关的商用处理器产品.在该体系架构下,CPU 和 GPGPU 共用片上存储,片上存储资源的争用问题变得更加突出,对其进行性能优化需要面对更高的复杂性,目前针对此类体系结构中缓存子系统的优化工作并不多.

4) 由于 GPGPU 上并发执行的线程数以万计,片上数据传输的数据量也非常巨大,现有的片上网络 NoC 由于其结构问题,当遇到访存请求过多的情况下,存在传输效率不高、功耗较大、无法有效区分冗余访存请求等问题,使得访存请求的数据传输时延增加,降低了片上缓存子系统的性能.为此,一方面可以对现有的 NoC 结构进行优化设计;另一方面需要更好地对冗余访存进行合并,提高 NoC 的数据传输效率,以更好地提升片上缓存子系统的数据访问效率.

5) 功耗问题始终是制约 GPGPU 发展的重要瓶颈.众多研究表明,片上缓存是 GPGPU 功耗产生的主要来源之一,这很大程度上限制了 GPGPU 片上缓存子系统的发展.结合任务调度,利用 DVFS 以及门控功耗等技术,实现对片上缓存子系统的功耗优化,以有效提升 GPGPU 片上缓存子系统的容量,从而缓解片上缓存的争用问题.

参　考　文　献

[1] Nickolls J, Dally W J. The GPU computing era [J]. IEEE Micro, 2010, 30(2), 56-69

[2] Hines J. Stepping up to summit [J]. Computing in Science & Engineering, 2018, 20(2): 78-82

[3] Luebke D, Humphreys G. How GPUs Work [J]. Computer, 2007, 40(2): 96-100

[4] Steffen M, Zambreno J. Improving SIMT efficiency of global rendering algorithms with architectural support for dynamic micro-kernels [C] //Proc of the 43rd Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2010: 237-248

[5] Keckler S W, Dally W J, Khailany B, et al. GPUs and the future of parallel computing [J]. IEEE Micro, 2011, 31(5): 7-17

[6] Lindholm E, Nickolls J, Oberman S, et al. NVIDIA Tesla: A unified graphics and computing architecture [J]. IEEE Micro, 2008, 28(2): 39-55

[7] Sankaralingam K, Nagarajan R, Liu Haiming, et al. Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture [C] //Proc of the 30th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2003: 422-433

[8] Wills L, Taha T, Baumstark L, et al. Estimating potential parallelism for platform retargeting [C] //Proc of the 9th Working Conf on Reverse Engineering (WCRE 2002). Piscataway, NJ: IEEE, 2002: 55-64

[9] Lee J, Lakshminarayana N B, Kim H, et al. Many-thread aware prefetching mechanisms for GPGPU applications [C] //Proc of the 43rd Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2010: 213-224

[10] Caragea G C, Tzannes A, Keceli F, et al. Resource-aware compiler prefetching for many-cores [C] //Proc of the 9th Int Symp on Parallel and Distributed Computing. Piscataway, NJ: IEEE, 2010: 133-140

[11] Bakhoda A, Kim J, Aamodt T M. Throughput-effective on-chip networks for manycore accelerators [C] //Proc of the 43rd Annual IEEE/ACM Int Symp on Microarchitecture. Los Alamitos, CA: IEEE Computer Society, 2010: 421-432

[12] John C. Professional CUDA C Programming [M]. Birmingham, the UK: Wrox Press Ltd. 2014

[13] NVIDIA. Cuda C programming guide [R]. Santa Clara, CA: NVIDIA, 2011

[14] He Yanxiang, Zhang Jun, Shen Fanfan, et al. Thread scheduling optimization of general purpose graphics processing unit: A survey [J]. Chinese Journal of Computers, 2016, 39(9): 1733-1749 (in Chinese)
(何炎祥, 张军, 沈凡凡, 等. 通用图形处理器线程调度优化方法研究综述[J]. 计算机学报, 2016, 39(9): 1733-1749)

[15] NVIDIA. NVIDIA's Next Generation CUDA Compute Architecture: Fermi, V1.1 [R]. Santa Clara, CA: NVIDIA, 2009

[16] NVIDIA. NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110 [R]. Santa Clara, CA: NVIDIA, 2012

[17] NVIDI. NVIDIA Tesla V100 GPU Architecture, WP-08608-001_v01 [R]. Santa Clara, CA: NVIDIA, 2017

[18] Fung W W L, Aamodt T M. Thread block compaction for efficient SIMT control flow [C] //Proc of the 17th Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2011: 25-36

[19] Bakhoda A, Yuan G L, Fung W W L, et al. Analyzing CUDA workloads using a detailed GPU simulator [C] //Proc of the IEEE Int Symp on Performance Analysis of Systems and Software. Piscataway, NJ: IEEE, 2009: 163-174

[20] Kayıran O, Jog A, Kandemir M T, et al. Neither more nor less: Optimizing thread-level parallelism for GPGPUs [C] // Proc of the 22nd Int Conf on Parallel Architectures and Compilation Techniques. Piscataway, NJ: IEEE, 2013: 157-166

[21] Wittenbrink C M, Kilgariff E, Prabhu A. Fermi GF100 GPU architecture [J]. IEEE Micro, 2011, 31(2): 50-59

[22] Adriaens J T, Compton K, Kim N S, et al. The case for GPGPU spatial multitasking [C] //Proc of the 18th Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2012: 1-12

[23] Pai S, Thazhuthaveetil M J, Govindarajan R. Improving GPGPU concurrency with elastic kernels [J]. ACM SIGARCH Computer Architecture News, 2013, 41(1): 407-418

[24] Zhong Jianlong, He Bingsheng. Kernelet: High-throughput GPU kernel executions with dynamic slicing and scheduling [J]. IEEE Transactions on Parallel and Distributed Systems, 2013, 25(6): 1522-1532

[25] Xu Qiumin, Jeon H, Kim K, et al. Warped-slicer: Efficient intra-SM slicing through dynamic resource partitioning for GPU multiprogramming [C] //Proc of the 43rd Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2016: 230-242

[26] Wang Zhenning, Yang Jun, Melhem R, et al. Simultaneous multikernel GPU: Multi-tasking throughput processors via fine-grained sharing [C] //Proc of the 22nd Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2016: 358-369

[27] Tanasic I, Gelado I, Cabezas J, et al. Enabling preemptive multiprogramming on GPUs [C] //Proc of the 41st Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2014: 193-204

[28] Park J J K, Park Y, Mahlke S. Chimera: Collaborative preemption for multitasking on a shared GPU [J]. ACM SIGPLAN Notices, 2015, 50(4): 593-606

[29] Li Chen, Yang Jun, Zigerelli A, et al. PEP: Proactive checkpointing for efficient preemption on GPUs [C] //Proc of the 55th ACM/ESDA/IEEE Design Automation Conf. Piscataway, NJ: IEEE, 2018: 1-6

[30] Park J J K, Park Y, Mahlke S. Dynamic resource management for efficient utilization of multitasking GPUs [J]. ACM SIGOPS Operating Systems Review, 2017, 51(2): 527-540

[31] Liang Yun, Huynh H P, Rupnow K, et al. Efficient GPU spatial-temporal multitasking [J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 26(3): 748-760

[32] Gebhart M, Keckler S W, Khailany B, et al. Unifying primary cache, scratch, and register file memories in a throughput processor [C] //Proc of the 45th Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2012: 96-106

[33] Yoon M K, Kim K, Lee S, et al. Virtual thread: Maximizing thread-level parallelism beyond GPU scheduling limit [C] // Proc of the 43rd Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2016: 609-621

[34] Yang Yi, Xiang Ping, Mantor M, et al. Shared memory multiplexing: A novel way to improve GPGPU throughput [C] //Proc of the 21st Int Conf on Parallel Architectures and Compilation Techniques. Piscataway, NJ: IEEE, 2012: 283-292

[35] Kim K, Lee S, Yoon M K, et al. Warped-preexecution: A GPU pre-execution approach for improving latency hiding [C] //Proc of the 22nd Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2016: 163-175

[36] Xiang Ping, Yang Yi, Zhou Huiyang. Warp-level divergence in GPUs: Characterization, impact, and mitigation [C] // Proc of the 20th Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2014: 284-295

[37] Jia Wenhao, Shaw K A, Martonosi M. MRPB: Memory request prioritization for massively parallel processors [C] // Proc of the 20th Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2014: 272-283

[38] Koo G, Oh Y, Ro W W, et al. Access pattern-aware cache management for improving data utilization in GPU [C] //Proc of the 44th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2017: 307-319

[39] Lee S Y, Wu C J. Ctrl-C: Instruction-aware control loop based adaptive cache bypassing for GPUs [C] //Proc of the 34th Int Conf on Computer Design. Piscataway, NJ: IEEE, 2016: 133-140

[40] Li Chao, Song S L, Dai Hongwen, et al. Locality-driven dynamic GPU cache bypassing [C] //Proc of the 29th ACM Int Conf on Supercomputing. New York: ACM, 2015: 67-77

[41] Chen Xuhao, Chang Liwen, Rodrigues C I, et al. Adaptive cache management for energy-efficient GPU computing [C] //Proc of the 47th Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2014: 343-355

[42] Dai Hongwen, Li Chao, Zhou Huiyang, et al. A model-driven approach to warp/thread-block level GPU cache bypassing [C] //Proc of the 53rd ACM/EDAC/IEEE Design Automation Conf. Piscataway, NJ: IEEE, 2016: 1-6

[43] Liang Yun, Li Xiuhong. Efficient kernel management on GPUs [J]. ACM Transactions on Embedded Computing Systems, 2017, 16(4): 1-24

[44] Xie Xiaolong, Liang Yun, Sun Guangyu, et al. An efficient compiler framework for cache bypassing on GPUs [C] //Proc of the IEEE/ACM Int Conf on Computer-Aided Design. Piscataway, NJ: IEEE, 2013: 516-523

[45] Xie Xiaolong, Liang Yun, Wang Yu, et al. Coordinated static and dynamic cache bypassing for GPUs [C] //Proc of the 21st Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2015: 76-88

[46] Liang Yun, Xie Xiaolong, Sun Guangyu, et al. An efficient compiler framework for cache bypassing on GPUs [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2015, 34(10): 1677-1690

[47] Liang Yun, Xie Xiaolong, Wang Yu, et al. Optimizing cache bypassing and warp scheduling for GPUs [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, 37(8): 1560-1573

[48] Xie Xiaolong, Liang Yun, Li Xiuhong, et al. Enabling coordinated register allocation and thread-level parallelism optimization for GPUs [C] //Proc of the 48th Int Symp on Microarchitecture. New York: ACM, 2015: 395-406

[49] Choi H, Ahn J, Sung W. Reducing off-chip memory traffic by selective cache management scheme in GPGPUs [C] //Proc of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units. New York: ACM, 2012: 110-119

[50] Tian Yingying, Puthoor S, Greathouse J L, et al. Adaptive GPU cache bypassing [C] //Proc of the 8th Workshop on General Purpose Processing Using GPUS. New York: ACM, 2015: 25-35

[51] Khairy M, Zahran M, Wassal A. SACAT: Streaming-aware conflict-avoiding thrashing-resistant GPGPU cache management scheme [J]. IEEE Transactions on Parallel and Distributed Systems, 2016, 28(6): 1740-1753

[52] Kim K Y, Park J, Baek W. Improving the performance and energy efficiency of GPGPU computing through integrated adaptive cache management [J]. IEEE Transactions on Parallel and Distributed Systems, 2018, 30(3): 630-645

[53] Rogers T G, O'Connor M, Aamodt T M. Cache-conscious wavefront scheduling [C] //Proc of the 45th Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2012: 72-83

[54] Rogers T G, O'Connor M, Aamodt T M. Divergence-aware warp scheduling [C] //Proc of the 46th Annual IEEE/ACM Int Symp on Microarchitecture. Piscataway, NJ: IEEE, 2013: 99-110

[55] Oh Y, Koo G, Annavaram M, et al. Linebacker: Preserving victim cache lines in idle register files of GPUs [C] //Proc of the 46th Annual Int Symp on Computer Architecture. New York: ACM, 2019: 183-196

[56] Wang Jianfei, Wang Qin, Jiang Li, et al. Ibom: An integrated and balanced on-chip memory for high performance gpgpus [J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 29(3): 586-599

[57] Narasiman V, Shebanow M, Lee C J, et al. Improving GPU performance via large warps and two-level warp scheduling [C] //Proc of the 44th Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2011: 308-317

[58] Jog A, Kayiran O, Chidambaram Nachiappan N, et al. OWL: Cooperative thread array aware scheduling techniques for improving GPGPU performance [J]. ACM SIGPLAN Notices, 2013, 48(4): 395-406

[59] Dublish S, Nagarajan V, Topham N. Poise: Balancing thread-level parallelism and memory system performance in GPUs using machine learning [C] //Proc of the 25th Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2019: 492-505

[60] Lai B C C, Kuo H K, Jou J Y. A cache hierarchy aware thread mapping methodology for GPGPUs [J]. IEEE Transactions on Computers, 2014, 64(4): 884-898

[61] Oh Y, Yoon M K, Park J H, et al. WASP: Selective data prefetching with monitoring runtime warp progress on GPUs [J]. IEEE Transactions on Computers, 2018, 67(9): 1366-1373

[62] Koo G, Jeon H, Liu Zhenhong, et al. CTA-aware prefetching and scheduling for GPU [C] //Proc of the IEEE Int Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2018: 137-148

[63] Jog A, Kayiran O, Mishra A K, et al. Orchestrated scheduling and prefetching for GPGPUs [C] //Proc of the 40th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2013: 332-343

[64] Oh Y, Kim K, Yoon M K, et al. APRES: Improving cache efficiency by exploiting load characteristics on GPUs [J]. ACM SIGARCH Computer Architecture News, 2016, 44(3): 191-203

[65] Sethia A, Dasika G, Samadi M, et al. APOGEE: Adaptive prefetching on GPUs for energy efficiency [C] //Proc of the 22nd Int Conf on Parallel Architectures and Compilation Techniques. Piscataway, NJ: IEEE, 2013: 73-82

[66] Lee S Y, Wu C J. CAWS: Criticality-aware warp scheduling for GPGPU workloads [C] //Proc of the 23rd Int Conf on Parallel Architectures and Compilation Techniques. New York: ACM, 2014: 175-186

[67] Lee S Y, Arunkumar A, Wu C J. CAWA: Coordinated warp scheduling and cache prioritization for critical warp acceleration of GPGPU workloads [C] //Proc of the ACM/IEEE Int Symp on Computer Architecture. New York: ACM, 2015: 515-527

[68] Kloosterman J, Beaumont J, Wollman M, et al. WarpPool: Sharing requests with inter-warp coalescing for throughput processors [C] //Proc of the 48th Int Symp on Microarchitecture. New York: ACM, 2015: 433-444

[69] Sethia A, Jamshidi D A, Mahlke S. Mascar: Speeding up GPU warps by reducing memory pitstops [C] //Proc of the 21st Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2015: 174-185

[70] Lee M, Kim G, Kim J, et al. iPAWS: Instruction-issue pattern-based adaptive warp scheduling for GPGPUs [C] //Proc of the 22nd Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2016: 370-381

[71] Lee M, Song S, Moon J, et al. Improving GPGPU resource utilization through alternative thread block scheduling [C] //Proc of the 20th Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2014: 260-271

[72] Kim H, Kim J, Seo W, et al. Providing cost-effective on-chip network bandwidth in GPGPUs [C] //Proc of the 30th Int Conf on Computer Design. Piscataway, NJ: IEEE, 2012: 407-412

[73] Jang H, Kim J, Gratz P, et al. Bandwidth-efficient on-chip interconnect designs for GPGPUs [C] //Proc of the 52nd Annual Design Automation Conf. New York: ACM, 2015: 1-6

[74] Cheng Xianwei, Zhao Yang, Zhao Hui, et al. Packet pump: Overcoming network bottleneck in on-chip interconnects for GPGPUs [C] //Proc of the 55th ACM/ESDA/IEEE Design Automation Conf. Piscataway, NJ: IEEE, 2018: 1-6

[75] Ziabari A K, Abellán J L, Ma Y, et al. Asymmetric NoC architectures for GPU systems [C] //Proc of the 9th Int Symp on Networks-on-Chip. New York: ACM, 2015: 1-8

[76] Zhao Xia, Kaeli D, Wang Zhiying, et al. Intra-cluster coalescing and distributed-block scheduling to reduce GPU NoC pressure [J]. IEEE Transactions on Computers, 2019, 68(7): 1064-1076

[77] Zhao Xia, Adileh A, Yu Zhibin, et al. Adaptive memory-side last-level GPU caching [C] //Proc of the 46th Int Symp on Computer Architecture. New York: ACM, 2019: 411-423

[78] Zhao Xia, Ma Sheng, Wang Zhiying, et al. CD-Xbar: A converge-diverge crossbar network for high-performance GPUs [J]. IEEE Transactions on Computers, 2019, 68(9): 1283-1296

[79] Zhao Xia, Ma Sheng, Liu Yuxi, et al. A low-cost conflict-free NoC for GPGPUs [C] //Proc of the 53rd Annual Design Automation Conf. New York: ACM, 2016: 1-6

[80] Mu Shuai, Deng Yandong, Chen Yubei, et al. Orchestrating cache management and memory scheduling for GPGPU applications [J]. IEEE Transactions on Very Large Scale Integration Systems, 2013, 22(8): 1803-1814

[81] Dublish S. Student research poster: Slack-aware shared bandwidth management in GPUs [C] //Proc of the 25th Int Conf on Parallel Architectures and Compilation Techniques. New York: ACM, 2016: 451-452

[82] Dublish S, Nagarajan V, Topham N. Cooperative caching for GPUs [J]. ACM Transactions on Architecture and Code Optimization, 2016, 13(4): 1-25

[83] Candel F, Valero A, Petit S, et al. Efficient management of cache accesses to boost GPGPU memory subsystem performance [J]. IEEE Transactions on Computers, 2019, 68(10): 1442-1454

[84] Vijaykumar N, Ebrahimi E, Hsieh K, et al. The locality descriptor: A holistic cross-layer abstraction to express data locality in GPUs [C] //Proc of the 45th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2018: 829-842

[85] HsiehK, Ebrahimi E, Kim G, et al. Transparent offloading and mapping (TOM) enabling programmer-transparent near-data processing in GPU systems [J]. ACM SIGARCH Computer Architecture News, 2016, 44(3): 204-216

[86] Pattnaik A, Tang Xulong, Kayiran O, et al. Opportunistic computing in GPU architectures [C] //Proc of the 46th Annual Int Symp on Computer Architecture. New York: ACM, 2019: 210-223

[87] Goswami N, Cao Bingyi, Li Tao. Power-performance co-optimization of throughput core architecture using resistive memory [C] //Proc of the 19th Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2013: 342-353

[88] Zhang Jie, Jung M, Kandemir M. FUSE: Fusing STT-MRAM into GPUs to alleviate off-chip memory access overheads [C] //Proc of the 25th Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2019: 426-439

[89] Satyamoorthy P, Parthasarathy S. MRAM for shared memory in GPGPUs [R]. Charlottesville, VA: University of Virginia, 2011

[90] Zhang Jie, Donofrio D, Shalf J, et al. Integrating 3D resistive memory cache into GPGPU for energy-efficient data processing [C] //Proc of the 14th Int Conf on Parallel Architecture and Compilation Techniques. Piscataway, NJ: IEEE, 2015: 496-497

[91] Samavatian M H, Abbasitabar H, Arjomand M, et al. An efficient STT-RAM last level cache architecture for GPUs [C] //Proc of the 51st Annual Design Automation Conf. New York: ACM, 2014: 1-6

[92] Venkatesan R, Ramasubramanian S G, Venkataramani S, et al. Stag: Spintronic-tape architecture for GPGPU cache hierarchies [J]. ACM SIGARCH Computer Architecture News, 2014, 42(3): 253-264

[93] Liu Xiaoxiao, Mao Mengjie, Bi Xiuyuan, et al. An efficient STT-RAM-based register file in GPU architectures [C] //Proc of the 20th Asia and South Pacific Design Automation Conf. Piscataway, NJ: IEEE, 2015: 490-495

[94] Zhang Hang, Chen Xuhao, Xiao Nong, et al. Architecting energy-efficient STT-RAM based register file on GPGPUs via delta compression [C] //Proc of the 53rd Annual Design Automation Conf. New York: ACM, 2016: 1-6

[95] Li Gushu, Chen Xiaoming, Sun Guangyu, et al. A STT-RAM-based low-power hybrid register file for GPGPUs [C] //Proc of the 52nd ACM/EDAC/IEEE Design Automation Conf. Piscataway, NJ: IEEE, 2015: 1-6

[96] Mittal S, Bishnoi R, Oboril F, et al. Architecting SOT-RAM based GPU register file [C] //Proc of the IEEE Computer Society Annual Symp on VLSI. Piscataway, NJ: IEEE, 2017: 38-44

[97] Jing Naifeng, Wang Jianfei, Fan Fengfeng, et al. Cache-emulated register file: An integrated on-chip memory architecture for high performance GPGPUs [C] //Proc of the 49th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2016: 1-12

Zhang Jun, born in 1978. PhD, associate professor, master supervisor. Member of CCF. His main research interests include performance and energy optimization of GPU architecture, and high performance computing.

Xie Jingcheng, born in 1993. Master candidate. Student member of CCF. His main research interests include cache optimization of GPU architecture.

Shen Fanfan, born in 1987. PhD. Member of CCF. His main research interests include performance and energy optimization of non-volatile memory.

Tan Hai, born in 1977. PhD, professor, master supervisor. Member of CCF. His main research interests include many-core architecture and big data.

Wang Lümeng, born in 1988. PhD candidate. His main research interests include energy optimization of GPU architecture.

He Yanxiang, born in 1952. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include trustworthy software engineering, computer architecture, and distribution computing.