多GPU系统非一致存储访问优化:研究进展与展望

李 晨,刘 畅*,葛一漩,郭 阳
(国防科技大学计算机学院,湖南长沙 410073)

摘 要: 随着晶体管缩小速度的减缓,单GPU(Graphics Processing Units)的性能提升已经变得越来越具有挑战性,因此,多GPU系统成为了提高GPU系统性能的主要手段. 然而,由于片外物理设计的制约,多GPU系统中处理器间的带宽不均衡导致了非一致存储访问(Non-Uniform Memory Access,NUMA)问题,严重影响多GPU系统的性能. 为了减少非一致存储访问所导致的性能损失,本文首先分析了非一致存储访问出现的原因,并对现有的非一致存储访问解决方案进行了对比. 针对不同维度的非一致存储访问,本文从减少远程访问流量和提升远程访问性能两个方向出发,对非一致存储访问的优化方案进行了总结. 最后,结合这些方案的优缺点,提出了未来多GPU系统非一致存储访问优化的发展方向.

关键词: 多GPU系统;非一致存储访问;GPU访存
基金项目: 国家自然科学基金(No.62202478);国防科技大学自主创新科学基金(No.23-ZZCX-JDZ-12)
中图分类号: TP393 **文献标识码:** A **文章编号:** 0372-2112(2024)05-1783-18
电子学报URL:http://www.ejournal.org.cn **DOI:**10.12263/DZXB.20231198

Non-Uniform Memory Access Optimization on Multi-GPU Systems: Research Progress and Prospect

LI Chen, LIU Chang*, GE Yi-xuan, GUO Yang
(*College of Computer, National University of Defense Technology, Changsha, Hunan* 410073, *China*)

Abstract: Due to the slowdown of transistor scaling, it has become increasingly difficult to enhance the performance of a single GPU (Graphics Processing Units). Therefore, multi-GPU systems have become the main means to improve the performance of GPU systems. However, due to the constraints of off-chip physical design, the bandwidth imbalance between processors in multi-GPU systems leads to non-uniform memory access (NUMA) problems, which seriously affects the performance of multi-GPU systems. In order to reduce the performance loss caused by non-uniform memory access, this paper first analyzes the causes of non-uniform memory access and compares existing solutions for non-uniform memory access. For non-uniform memory access with different dimensions, this paper summarizes optimization solutions for non-uniform memory access from two directions: reducing remote access traffic and improving remote access performance. Finally, combining the advantages and disadvantages of these solutions, this paper proposes the future development direction of non-uniform memory access optimization for multi-GPU systems.

Key words: multi-GPU system; non-uniform memory access; GPU memory access
Foundation Item(s): National Natural Science Foundation of China (No.62202478); NUDT Innovation Science Foundation (No.23-ZZCX-JDZ-12)

1 引言

近年来,随着高性能计算和智能计算等领域的迅猛发展[1~4],数据规模也呈现出爆炸性增长,为实现高吞吐率和高能效比的海量数据处理和计算,通用图形处理器(General-Purpose Graphics Processing Unit,GPGPU)以其强大的并行计算能力,已成为人工智能和科学计算领域主流的协处理器[5~15]. 随着应用需求不断增长,特别是随着人工智能大模型的广泛应用,对GPGPU的算力提出了更高的要求.

然而,受限于光刻和制造成本,晶体管缩放速度越

收稿日期:2023-12-25;修回日期:2024-04-03;责任编辑:宋甲英
*通讯作者:刘畅

来越慢，当前的GPGPU已经接近芯片尺寸最大的限制，集成更多的晶体管到单个GPU芯片上变得极其困难．单纯依靠扩大单个处理器的规模以提升GPU的性能，已难以满足当前高性能计算的算力增长需求[16,17]．为进一步延续摩尔定律，工业界和学术界都将目光投向了多GPU系统．多GPU系统可以提供远大于单GPU系统的运算资源和存储资源，可实现运算规模和运算能力的突破性增长．

NVIDIA的DGX和DGX-2系统在每个节点集成8~16个GPU，主要面向深度神经网络模型的训练，最高能提供2 PFLOPS（2 Peta FLoating-point Operations Per Second）的运算性能．2022年底，NVIDIA更是推出了基于H100的DGX H100多GPU系统以面向高性能AI计算．AMD Instinct GPU支持通过AMD Infinity Fabric互连组成多GPU系统，针对高性能计算和人工智能负载实现超高速度和智能化大规模数据吞吐量．面向超算和数据中心的GPU系统，则提供了更大规模的多GPU互连，以实现E级的计算能力．从单GPU系统到多GPU系统，虽然计算资源和存储资源大大增加，但其协同工作时，多GPU处理器之间共享存储访问和互连通信量也显著增加．受限于多GPU系统不同层次的带宽和延迟不均衡性，多GPU系统的性能难以随着计算资源提升而线性提升，系统性能受到非一致性内存访问（Non-Uniform Memory Access）带宽瓶颈的严重制约而带来一系列挑战．研究证明，GPU应用程序的执行性能，尤其是非规则应用程序，很大程度取决于访存操作的效率[18~21]，因此如何克服非一致性内存访问带来的一系列挑战成为多GPU系统研究的主要研究方向．

本文面向多GPU系统，总结和分析非一致性内存访问发生的原因及其场景，归纳多GPU系统中针对非一致性存储访问瓶颈的相关研究，主要有三个方面的贡献．

（1）将多GPU系统的不均衡存储访问划分为三部分：（a）GPU内的访存；（b）多GPU之间的访存；（c）CPU与GPU之间的访存．并围绕这三个层次进行非一致性存储访问的分析．

（2）将现有研究划分为内存布局[22~40]、线程调度[40~48]、互连架构[5,16,49~56]等多个方向进行综述，并对这些解决方案的特色进行深入分析．

（3）最后，基于综述和分析成果，提出多GPU系统非一致性内存访问研究的未来发展方向．

2 多GPU系统及其非一致存储访问

2.1 GPU系统简介

GPU由于其强大的并行计算能力，目前已广泛应用于图形处理、大规模仿真、大模型，以及深度神经网络等并行计算任务中．GPU采用单指令多线程模型（Single Instruction Multiple Threads，SIMT），每个流式多处理器中均包含一组处理单元，可以并行执行多个线程．通过多线程访存交错和并行计算，GPU实现了延迟隐藏，提高硬件利用率．然而，随着大模型、大数据处理等应用需求的不断扩张，单纯依靠单GPU的运算性能已经越来越难以满足计算任务的需求．因此，多GPU系统成为一种广泛应用于高性能计算和智能计算场景下的计算系统，采用多个GPU处理器互连，集成到单个计算机系统中．图1是一个典型的基于PCIe（Peripheral Component Interconnect express）互连的多GPU系统．在这个系统中，CPU作为主机系统，GPU内存可以共享．多GPU系统通常采用的是传统高性能单卡GPU处理器，即芯片架构上与单卡GPU完全一致．为了实现强大的计算能力和并行处理能力，多GPU系统主要在互连和软件系统上进行优化．

与传统的GPU系统类似，多GPU系统制造商主要还是NVIDIA和AMD．其中，NVIDIA的DGX系列多GPU系统和AMD Instinct系列多GPU系统是较为广泛应用的多GPU系统．NVIDIA与AMD均采用自家私有协议的高速互连技术，以实现多GPU间或CPU与GPU间的高速互连通信，如NVIDIA NVLink技术和AMD Infinity Fabric技术．此外，各数据中心服务商也通过PCIe、CXL协议等互连方式，建立GPU系统服务器集群（Google Cloud、AWS EC2、Aliyun等）．最新一代的NVIDIA DGX H100系统，通过8个NVIDIA H100 GPU的互连，能够在FP8下达到32 PFLOPS的运算能力，满足自然语言处理、推荐系统、数据分析等多方面的应用需求．

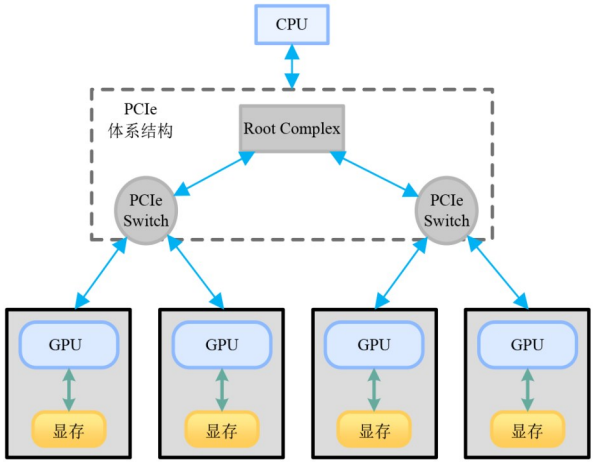

图1 多GPU系统架构

2.2 多GPU系统非一致存储访问特征

非一致存储访问主要指存储访问在相同存储层次下不同距离节点间以及不同存储层次之间的带宽和延

迟不均衡性. 这种不均衡现象往往会形成性能瓶颈,导致不公平的存储访问,特别是在某GPU主机频繁访问其他GPU存储空间时,相比于访问本地存储空间,其延迟显著增加. 因此,非一致性存储访问成为制约其性能提升的关键瓶颈.

在多GPU系统中,如图2与表1所示,非一致性存储层次带宽不均衡主要面向三个维度:GPU片内与多GPU片间的带宽不均衡、CPU与GPU的数据传输带宽不均衡以及GPU片内缓存与片内DRAM(Dynamic Random Access Memory)之间的带宽不均衡. 这三个维度的带宽不均衡继而导致多GPU系统出现三种非一致存储访问现象.

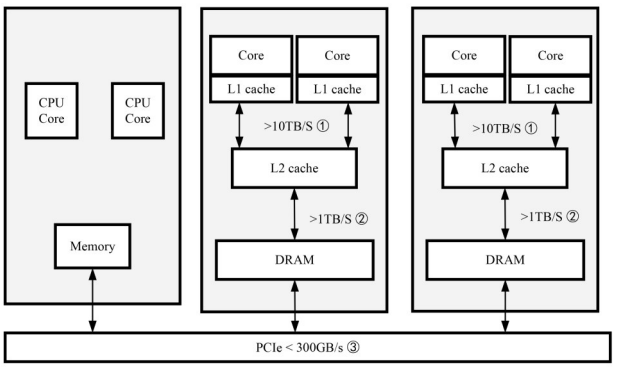

图2　多GPU系统存储层次

表1　带宽对比

内存/互连	技术	带宽
内存	HBM2e[57,58]	2 TB/s
互连	NVLink3.0[54,58]	600 GB/s
互连	PCIe 6.0[58,59]	300 GB/s

(1)多GPU之间的非一致存储访问现象. 在多GPU系统中,各GPU处理器之间并不完全独立,部分数据页会被多个GPU处理器所共享,这会引起大量的GPU间数据访存. 在图1所示的多GPU系统中,即使采用PCIe 6.0协议[59],GPU间互连可提供300 GB/s的带宽,NVIDIA独家的NVLink 3.0[54]最高可提供600 GB/s的带宽,而目前GPU内共享存储[57]的访存带宽高达2 TB/s,因此片内带宽和片外带宽存在巨大鸿沟. Ang等人[60]的研究表明,多GPU系统很难充分利用GPU间互连的外部带宽,这大大增加了数据访问带来的延迟,导致了非一致存储访问现象的产生.

(2)CPU与GPU之间的非一致存储访问现象. 在多GPU系统中,GPU主要作为加速器处理高度并行的计算任务,其在系统中往往作为CPU的从设备[61~65]. 传统的CPU与GPU的存储器是物理分离的,CPU端的内存被称为主机内存(Host Memory),GPU端的内存被称为设备内存(Device Memory). 在应用程序的运行过程中,待处理的数据需要从主机内存传输到设备内存中,然后待数据处理完成后把计算结果传输回主机内存. 在这种情况下,数据传输难以与任务计算并行. 后续出现的统一内存寻址等技术[66~70]允许CPU与GPU共享内存空间,但即使采用带宽最高的NVLink也只能达到600 GB/s的峰值带宽,与本地内存访问的带宽相差甚远. 因此,出现CPU与GPU之间的非一致性内存访问现象.

(3)GPU内部的非一致存储访问现象. 如图2所示,在GPU片内,通常采用两级缓存方案,其中一级缓存的访问带宽往往可以达到10 TB/s以上. 然而,由于一级缓存的容量一般只有32 KB,无法完全满足应用程序的内存占用(memory footprint)需求,因此大量的数据访存需要通过一级缓存的失效送到下一级存储进行服务[71~74]. 由于不同级存储带宽的差距,导致访存延迟的不均衡,从而出现GPU内部的非一致存储访问现象.

此前,非一致存储访问现象在CPU已经出现过. 在CPU系统中,主要通过软件内存分配、迁移策略以及线程调度等技术[36,75~89]来减少非一致存储访问带来的性能损失. 然而,这些解决方案大多是被动式的,在处理器运行时通过检测访问内存位置和拥塞情况来进行数据迁移等操作. 相比之下,多GPU系统中的应用程序大多并行程度较高且数据量大,数据迁移可能会导致频繁的流水线刷新,产生高额的开销;数据复制可能会增大内存的压力,导致缓存抖动等现象;并且GPU运行的大量并行线程也使GPU中的线程调度比CPU更加复杂. Milic等人发现,直接采用CPU的NUMA调度和内存布局分配策略已经难以满足多GPU系统的性能和可扩展性的需要[90]. 如何有效地进行多GPU系统的非一致存储访问优化是进一步提升GPU性能所亟待解决的主要问题,本文主要归纳总结面向多GPU系统的非一致存储访问问题的解决方案.

3　非一致存储访问解决方案

在多GPU系统中,非一致存储访问为多GPU系统性能提升带来了诸多挑战,主要体现在多GPU系统协同工作时大量的数据通信和远程访问的高开销. 为了减少由于非一致存储访问问题带来的性能损失,GPU设计需要在存储系统管理、计算资源分配和互连等方面实现非一致存储访问感知,如图3所示,主要的解决方案可以分为减少远程访问数据流量和提升远程访问性能两种,其中主要研究方向包括内存布局、线程调度和互连架构等. 这三方面在多GPU系统非一致存储访问的三个层次上各有侧重,且均有体现.

内存布局是指在多GPU系统中,如何有效地分配和管理内存资源. 在多GPU系统中,由于每个GPU都

有自己的本地内存,非一致存储访问问题使得这些内存之间的访问速度存在显著差异,因此如何合理地分配数据和任务到不同的GPU内存中,成为了一个重要的问题. 内存布局的目标是在保证数据访问效率的同时,尽量减少数据在不同GPU之间的传输开销. 合理的内存布局可以提高数据局部性,降低数据传输延迟,从而提升系统的整体性能.

线程调度是指在多GPU系统中,如何有效地分配和调度计算任务. 在多GPU系统中,由于各个GPU的计算能力和负载情况可能不同,因此需要一种高效的线程调度策略. 一方面平衡各个GPU的负载,充分利用系统资源;另一方面,在保证任务执行效率的同时,尽量避免资源冲突和竞争,紧密耦合内存布局和线程调度两个策略,降低通信开销,进一步提升系统的整体性能.

互连架构是指多GPU系统中各个GPU之间的通信和连接方式. 在多GPU系统中,各个GPU之间存在大量数据交换和同步,因此互连架构的设计对于系统的性能至关重要. 互连架构的目标是在保证数据传输效率的同时,尽量减少通信延迟和冲突. 合理的互连架构可以提高系统的可扩展性和可靠性,从而进一步提升系统的整体性能.

上述三个方面的研究方向,在多GPU系统非一致存储访问的优化上,基本上是围绕在减少远程访问数据流量和提升远程访问性能两个解决思路上.

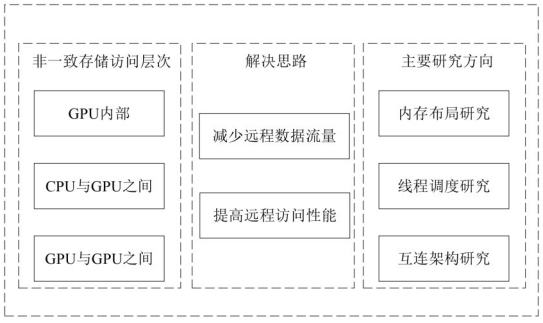

图3 非一致存储访问解决方案

如何减少远程数据流量? 在多GPU系统中,实现线程和数据的紧耦合能够有效地减少远程数据流量,这主要通过对线程和数据进行合理的布局实现. 对数据的操作可以分为数据布局和缓存保护两种方式[5,54,60]. 其中,数据布局是将线程使用的数据进行静态分配或动态调度到线程所在的处理器中,但是在多GPU系统中,为了保证数据的一致性,对数据的动态调度需要考虑流水线刷新等操作产生的高额开销;而缓存保护是保护缓存中的具有较高局部性且失效开销较大的数据,防止其被其他数据替换,从而改善数据的时间局部性. 通过合理的线程调度实现数据局部性的充分利用,从而实现计算和数据的耦合;同时线程调度可以将访问同一内存区域的线程分配到一起,实现内存访问请求的合并,以利用数据局部性提高系统性能.

如何提高远程访问的性能? 提高远程访问的性能主要是通过更高效利用外部带宽,降低和隐藏远程访问延迟等方式来实现的. 由于外部带宽有限和多GPU系统处理器内核之间的依赖性,数据通信不可避免. 因此,如何高效地利用有限的外部带宽是提升多GPU系统性能的有效方法;并且,由于多GPU系统采用物理链路连接和集中式虚拟内存管理单元,因此可以通过更合理的封装架构和降低数据失效时页表查询开销来降低远程访问的时间,例如,设计NVLink等新型连接方式以及远程数据的MSHR表;同时,多GPU系统可以通过有效的线程调度策略和线程切换隐藏远程访问延迟,提升系统性能.

针对多GPU系统中存在的非一致存储访问瓶颈,本节从三个维度出发,综合考虑计算和数据的耦合、数据的局部性、提高互连网络的带宽利用率以及隐藏访存请求延迟等方案,研究归纳多GPU系统的非一致存储访问的优化方案.

3.1 GPU内非一致存储访问解决方案

在CPU端进行数据分配之后,大量数据将被拷贝至GPU端的共享内存中. GPU虽然可以通过切换线程隐藏访存延迟,但在部分数据密集型应用程序的执行过程中,由于第一级缓存空间有限,Cache命中率较低,大量线程需要访问共享内存甚至更低一层存储器获取数据. 访存线程过多,等待访存请求返回而造成堵塞,延迟无法被完全隐藏,导致系统性能下降. 为了降低非一致存储访问的影响,需要充分地利用缓存中数据的局部性,减少共享内存的访问次数,同时降低数据传输延迟. 现有研究主要从内存布局、线程调度以及互连架构等三个方面解决其非一致存储访问问题.

3.1.1 内存布局

循环和数组结构通常具有计算密集和数据并行的特征,但在某些应用中,由于数据依赖和相关控制的限制,它们在GPU上的高效运行受到挑战. 为了在GPU上实现高效运行,我们需要对计算和数据进行重构,通过循环合并与拆分的计算重构来增加应用程序的可并行性,以尽可能消除操作间的依赖关系,提高计算密集性. 同时,对线程内和线程间的数据访问进行重构,以减少GPU计算核心的存储访问次数,从而提高系统性能. Piccoli等人[27]提出了一种自动分析代码技术,通过编译器分析程序中的循环和数组变量以确认被复用的数据页,从而实现高效数据分配改善数据局部性. 然而,这种方法存在较大的局限性,它只支持LLVM(Low Level Virtual Machine)编译器,并且通过分析循环和数

组变量进行数据分配,缺乏一定的灵活性,容易造成无用数据的传递,使适用应用数据规模受限.

通过静态分配数据无法分析GPU在运行过程中产生的信息,例如数据迁移和计算等,导致方案缺乏灵活性,无法完全实现计算和数据的耦合. 在GPU片内,内存数据动态迁移主要是通过数据预取实现的,而预取需要考虑数据一致性的问题. 因此,如何通过动态数据迁移减少维护数据一致性的开销,成为内存数据分配的重要挑战. 同时,由于L1缓存的容量有限,已经被分配到L1缓存中的数据,可能会被未来访问的数据来回替换,导致较高的访问延迟. 这种现象称为缓存抖动(Thrashing). 缓存抖动会造成重复访问或周期性访问的数据无法在本地缓存中获取,只能发送到更低级缓存获取,增加访问数据的延迟. 因此,需要采取合适的策略改善数据的局部性,提高整体性能.

传统的L1缓存是每个GPU内核所私有的,这在单核数据处理条件下有利于提升系统的访存性能. 然而,对于多核读写共享数据的情况,需要确保数据的一致性,如果读共享数据的局部性较低,则会进一步提升数据一致性开销. 为了减少远程访问的次数以及节省内存资源以更好地存储具有较高局部性的数据,如图4所示,Ibrahim等人提出了一种共享L1缓存的组织方式[31]. 与传统的私有L1缓存对比,每个处理器内核中的L1缓存只缓存特定地址范围的数据,从而确保缓存中数据的零重复性. 同时,Ibrahim等人还提出了三种优化的解决方案:首先,降低发送内存请求的粒度,以减少互连网络带宽浪费;其次,平衡发送到其他内核和发送到L2缓存的内存请求数量,以减少阻塞;最后,通过起始的数据采样来决定L1缓存的组织方式,从而避免对私有数据友好的应用程序造成性能损失.

为了更好地维护数据的时间局部性,本文将数据局部性分为四种类型:(1)流数据:将不具有局部性的数据称为流数据,由于流数据不具有局部性,不适合更新到本地缓存,反而会替换具有局部性的数据,造成性能损失;(2)warp内部局部性:由warp加载来的数据在同一个warp内被多次重用,称之为warp内部局部性(intra-warp locality);(3)warp间局部性:由warp加载,但被不同的warp重用的数据,称之为warp间局部性(intra-warp locality);(4)有些数据同时具有warp间和warp内部局部性. 因此防止流数据更新到本地的缓存中替换具有局部性的数据是一种主要的解决缓存抖动的思路. 这种方法主要从以下两个角度实现.

首先可以通过减少流数据更新到L1缓存中,这主要通过旁路防止流数据的写入来实现的. Tian等人[28]提出了自适应缓存旁路(ABYP),通过监视被逐出的高速缓存行,并在设计的预加载表中收集此信息,如果加载分配的缓存行在没有重复使用的情况下被多次逐出,则未来的加载请求不会在L1缓存中写入. 然而,由于L1的容量有限,很容易把一些具有强局部性的数据也误判为流数据,从而降低了局部性;为了解决这一问题,Koo等人[29]提出了访问模式感知的缓存管理(APCM),该方法通过监视采样性的warp的访问来动态检测每条加载指令的局部性类型,并根据检测到的局部性类型来选择是否进行缓存旁路和缓存保护. 这两种方法有效地减少了流数据更新到本地缓存中,降低了访问数据的失效概率.

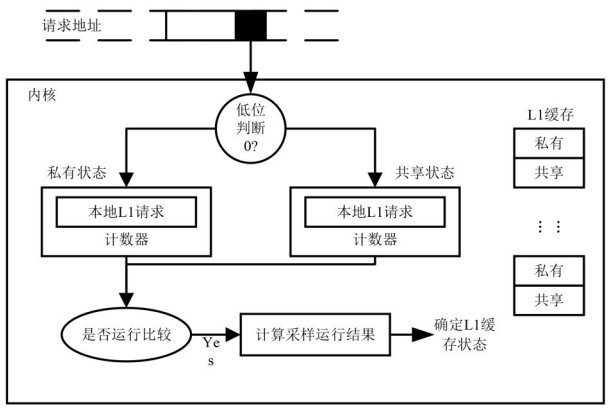

图4　共享L1缓存方案

对多GPU间共享数据的缓存行进行保护,从而避免流数据的更新替换是实现缓存保护的另一种重要思路. Duong等人[30]提出了一种基于访问次数的缓存保护方法. 首先通过预测获得每个缓存行可能被访问的次数,然后在这个次数用尽之前对该缓存行进行保护,即不允许其他数据替换该缓存行. 然而,这种方法的准确率依赖于计数预测的准确程度,如果预测准确率低,就会导致有限的缓存资源被局部性不高的数据占用,从而使得具有更高复用性的数据无法被缓存,进而造成性能下降. 为了解决这一问题,Koo等人[29]提出了一种根据数据相关性来预测缓存中的有效数据寿命的方法,通过分析数据相关性来设置保护位,一旦保护位被设置为1,该缓存行就不能被其他数据替换,这种方法可以更加有效地预测缓存行的寿命,从而实现更高的性能.

3.1.2　线程调度

GPU片内的线程调度可以隐藏处理核的数据访问延迟,并提高数据复用和避免缓存争用,进而提升系统性能. 在GPU片内,一种常见的策略是尽量将使用相同数据的线程调度到同一个内核程序中,以便充分利用L1缓存的局部性. Li等人[46]对线程局部性进行了分类,并探讨如何利用线程间的局部性. 他们将线程间的局部性分为算法相关、缓存行相关、数据相关、写相关和无相关性的线程. 他们证明了线程间局部性在GPU

应用程序中的重要性,并提出了线程集群的概念以及线程聚类的方法.这些方法将具有潜在重用的线程调度分组到同一个内核中.这种方法主要实现了线程间的局部性,将具有局部性的线程聚集在一起,从而实现数据的复用.同时,为了减少远程访问次数并充分实现数据的时间局部性,线程调度可以与内存布局相结合.时间局部性指的是线程使用的数据可能被之后的线程继续使用,但由于缓存抖动现象的存在,顺序的调度线程可能会导致这种局部性消失.为了利用好时间局部性并减少线程调度的次数,可以采用乱序调度线程的方法.例如,在上一个周期内调度的线程将数据取回L1缓存之后,访问相同数据的线程先进行调度,防止复用的数据替换,从而充分利用数据的时间局部性,减少了缓存抖动.

此外,随着3D存储器成为当前主流的GPU存储器[91~95],线程调度需要与3D存储器相结合.3D堆叠存储器包含一个基本逻辑层,容纳GPU核心.这种体系结构下包含两种基本特点:(1)主要进行计算的主GPU内核,称为GPUPIC(GPU Processing In Core),其类似于现代GPU核心;(2)较小并且功能较弱的GPU核心,称为GPU-PIM(GPU-Processing In Memory),其布局在主存储器下的逻辑层中,用于辅助执行运算.3D存储器可以将指令链卸载到存在数据的存储器附近的计算单元之中,这种方法称之为PIM(Processing In Memory).由于计算单元与存储器紧密集成,因此数据可以以更高的带宽和更低的延迟从存储器中获取数据.但是由于PIM的计算核心容量较小,如何对线程进行合理的分配从而能最大程度地利用资源成为需要解决的主要问题之一.Pattnaik等人[47]介绍了一种基于回归亲和力的预测模型和机制,这种方式通过识别哪些内核能在PIM中获益来选择性地进行线程分配,并且还通过并发内核管理机制来并发调度内核.然而这种线程调度策略会造成缓存不一致的问题,尤其是对于采用写回方式的设备,可能造成主存储器与L1缓存数据不一致,需要大量的维护开销从而降低系统性能.

3.1.3 互连架构

在GPU系统中,一旦L1缓存发生失效,必须通过互连网络将请求发送到内存子系统中进行访问.然而,互连网络的带宽是有限的,如果大量内存请求在短时间内被发送到互连网络,就会造成互连网络的阻塞.为了更好地利用互连网络资源,必须对其进行合理的优化.在传统的GPU系统中,互连网络对物理通道进行了虚拟化,通过轮询的方式来依次占用通道,从而提高互连网络利用率.此外,在多GPU系统中,还可以通过线程调度和内存数据布局来减少互连网络中的阻塞.同时,由于L1缓存的容量较小,导致缓存抖动的问题,这需要采取一定的策略来解决.Wang等人[56]提出了用于高性能GPGPU的集成且均衡的片上存储器(IBOM)架构,从根本上扩大了L1缓存的大小.它通过轻量级指令集、编译器和微体系结构支持以利用空闲的寄存器文件空间,通过聚合资源的均衡技术提高资源的利用率,从而实现更高的性能和能效.

同时,失效的内存请求发送到存储控制器上的延迟与处理器和存储控制器的布局紧密相关.在传统的2D-Mesh网络中,存储控制器的布局影响了内存请求在互连网络中的跳数,跳数决定了数据传输的效率.因此,合理地对存储控制器进行布局,可以减少总的内存请求跳数,从而减少访问内存子系统的延迟.Jang等人[55]提出了一种合理分配存储控制器的方法,通过计算所有的计算核心到存储控制器的最小距离来布局存储控制器,这种方法通过最小化所有内核访问存储控制器所需要的跳数,从而减少互连网络中的延迟.

3.2 CPU与GPU互连的非一致存储访问解决方案

随着多GPU技术的不断发展,Muller等人[96]提出了CUDA系统架构框架,该框架对CUDA线程和内存的层次结构进行了扩展,使得类似CUDA的代码能够在多节点、多GPU系统上运行.在现有的CPU-GPU系统中,CPU主要负责复杂逻辑和事务处理等串行操作,而GPU则承担大规模并行运算的任务,从而充分发挥计算系统的处理能力.

在一个典型的CUDA程序中[97,98],需要把数据从CPU内存拷贝到GPU内存,然后调用核函数对数据进行操作,最后再将数据传送回CPU内存.因此,CPU-GPU系统中的非一致存储访问对系统的整体性能产生影响,这需要将内存数据、线程调度和体系结构方案结合起来,实现数据与计算的紧密耦合.同时,需要寻求能更好地利用外部带宽和减少数据传输延迟的互连机制.

3.2.1 内存布局

对于CPU-GPU系统,非一致存储访问的内存布局方案主要是考虑数据的初始化分解问题.在应用程序初始阶段,由于数据量相对较小,GPU的显存容量远大于数据量,数据甚至可以被全部分配到GPU显存之中.在这一阶段,通常采用程序员手动分配的方法,但这要求程序员有丰富的经验和专业基础.Jungwon等人[22]提出了一种OpenCL(Open Computing Language)框架,允许将单GPU程序迁移到多GPU上,从而减少了程序员的工作负担;同时,它还允许通过采样运行来分析预测GPU处理器计算核心的访问内存区域.此外,该框架还根据记录采样运行过程中传输的数据量,为程序员的手动内存布局提供依据.这种方法虽然比盲目的数据

分配更加地可靠有效,但采样运行需要耗费大量的时间,且这种方法在粗粒度上对数组访问进行简单的分析,容易造成大量无用数据的传递.随着应用程序数据量的进一步增加,无用数据的传递会导致内存资源的争用,从而使系统性能下降.

虽然采样运行可以为应用程序开发人员进行程序分区提供依据,但是这种方法仍然需要程序员手动参与程序的分配.随着应用程序数据量的增加,由程序员来进行数据分配不仅耗费精力,还很容易出现数据分配不平衡、不合理的问题.因此,人们更倾向于用软硬件协同的方法进行有效的数据分解,从而减少程序员的工作量.Luk等人[41]提出对GPU代码的数据依赖进行分析,实现了数据的自动分解.类似的Ji等人也提出了自己的观点[50],这些研究自动化了数据分解,但迫使应用程序开发人员使用独特的编程模型和应用程序编程接口重写代码,这增加了编程的工作,因为需要修改代码以实现以下目标:(1)在CPU和GPU之间传输数据,(2)调用GPU代码,以及(3)访问GPU代码中的变量.为了对应用程序开发人员隐藏数据分解,Lee等人[43,49]提出了一种可以在多GPU系统上运行单个GPU代码的编程框架,他们的框架分析内存访问行并对整个线程进行检查,因此可以不受约束地应用于各种应用程序,但是内存访问行为分析会产生比Kim的方法更长时间的等待延迟.同时,Thejas等人[23]提出了AMGE(Automatic Multi-GPU Execution)框架,它在内核启动时在更细粒度上对计算网络和数据进行分解,并通过试运行来找出最小字节移动量的组合,以此来进行数据和程序分配.这种方法更加综合地考虑了列共享和行共享,在更细粒度上进行内存数据分解从而减少了无用数据的传递量,但是AMGE只针对一维的数组数据,并且对某些数据规模较大的应用程序不仅不会提升,反而会造成一定的性能下降,同时分析的开销较大,无法找出数据相关和线程块本地访问的数组变量.因此,如何减少启动和执行的延迟,是大规模应用程序自动化数据分解必须解决的问题.

上述的工作主要集中于一维或二维存储器上数组的分解,而随着三维存储器的发展,AMGE只能分解单维的数据,并且假定数据量小于GPU存储器的总容量.然而,随着应用程序的进一步扩展,这个假设逐渐失效,因此需要设计高效的多维数据分配策略,为了更好地对数据进行自动分解,Ryotaro等人[24]提出了一种将一维存储器地址映射为三维存储器地址的方法.同时,通过选定的线程运行数据分解的方式,以减少不必要的数据传递.这种方式在一维数组数据分配策略的基础上,将内存数据分配方法扩展到了三维存储器上.同时,利用更细粒度的分解来减少数据不必要的传递,从而适用于更大数据规模的应用程序,有效提升了系统性能.

上述方法主要采用在系统运行初期进行静态调度来实现计算和数据的耦合.静态调度可以更好地分配数据,而动态调度通常需要考虑调度哪些数据以及何时进行调度.在传统的内存数据布局策略中,通常采用具有一定阈值的数据迁移策略来决定哪些数据会被调度,即当计算单元请求某个数据页到达一定次数之后再进行数据页的迁移.随着GPU上运行的应用程序数据量增加,设置数据迁移的阈值不仅不会提高性能,反而会导致多GPU系统外部带宽的利用率降低.因此,Neha等人[25]提出了低阈值的内存数据迁移策略,同时提出通过提高取数据的粒度的来减少远程访问次数.这种方式能够更好地利用总线的带宽,但是可能会造成无用数据在互连网络中的传递,继而带来缓存抖动,降低数据的局部性.

由于低阈值数据迁移可能会使大量数据短时间内被发送到总线上,引发严重的总线争用,导致GPU总体请求数据的时间过长,进而影响GPU系统的性能.针对如何确定动态调度时机,Tianhao等人[26]提出了一种解决方案,如图5所示.他们通过建立远程数据的MSHR(Miss-Status Handling Registers)表,将访问远程数据的操作合并,减少了访问远程数据的次数,避免了访问远程数据中断的现象.此外,采用预取部件提高取数据的粒度,可避免多次远程数据访问操作.这种方法有效提升了访问远程数据的性能,但仍有可能导致无用数据的传输.在数据规模不断扩大的情况下,需要进一步考虑如何在更细粒度上进行操作.

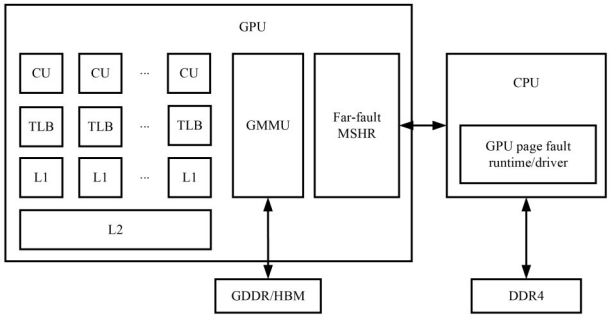

图5　远程数据MSHR示意图

3.2.2　线程调度

在多GPU系统中,线程作为数据的使用者,其分配和执行决定了数据的使用效率.为了更好地利用内存资源,不仅需要对内存数据进行布局,还需要对线程进行合理的调度.与传统的CPU线程不同,GPU中的线程大多数具有依赖性,这种依赖性对线程分配有较大的影响.提高CPU和GPU利用率的一种有效途径是在CPU和GPU上异步执行数据并行的内核,当内核之间

的依赖性无法消除时,如何合理地进行内核的划分就成为了一项有挑战性的工作.在过去的方法中,通常需要程序员确保CPU和GPU内核之间的数据依赖关系,但这种方法很显然不适用于数据量愈发膨胀的应用程序,目前线程调度主要存在三方面的问题:(1)对于不规则存储器访问模式的数据并行的内核很难在多个设备中进行分区;(2)当GPU数量增多时,分区的决策变得更为复杂,需要考虑更多的数据传输成本;(3)多GPU系统的性能与GPU的数据不是固定不变的,这种变化会影响分区决策.

与内存布局相同的是,线程的初始分配也由应用程序开发人员手动进行.为减轻程序员在线程调度方面的负担,Luk等人提出了一种新型API(Application Programming Interface),能支持将线程自动划分到CPU和GPU上[41],然而,此方法只能支持两个设备(一个CPU和一个GPU),且适用于数据并行内核的API限制较多,同时需要耗费大量的时间和功耗对所有线程的访问位置进行动态分析.Kim等人也提出了一种静态线程分配法[42],但此方法规定了线程数量是GPU数量的整数倍,且每个线程的访问位置必须是规则的.这种方法虽能实现线程分配且无需程序员介入,但由于内核和硬件的严格限制,大部分情况下多GPU系统无法从中获益.随着应用程序复杂性提升,设计一种对程序员透明、适用大部分应用程序的线程分配策略变得越来越重要.

随后,Lee等人提出了单核多设备SKMD(Single Kernel Multi-Device)系统[43],这是一种能够透明地协调跨多个非对称GPU和CPU的单个数据并行内核协同执行的体系结构.SKMD系统包含内核转换器、缓冲区管理器以及分区程序.其中,内核转换器负责将执行内核划分为分区就绪内核和数据合并的内核;在内核转换后,缓冲区管理器对内核进行静态分析,旨在明确每个工作集的内存访问模式.根据静态分析结果,为每个计算任务的内存访问范围进行最优化的内核分区,以减少内核数据传输,若无法分析出内核访问的内存区域,则将整个输入传递到每个设备,并且合并所有设备的输出.为了合并不同设备不规则位置的输出,内核转换器会生成合并内核,并在CPU设备上启动合并内核.此方法可根据数据和线程的关系来有效地分配线程,但由于在采样分析过程中可能会将整个输入传递到设备中,从而引发严重的传输压力.

为了减少内核启动开销和排队等待时间,Tang等人提出了SPAWN[44],用于控制动态生成的内核,并允许调度程序更好地混合使用动态生成的内核与原始的内核,有效地隐藏其余开销,提高GPU资源的利用率.同时,Park等人提出GPU Maestro来执行动态资源管理[45],有效地利用多任务GPU.GPU Maestro利用空间多任务和SMK(Simultaneous Multi-Kernel)发现最佳的GPU资源分区.

随着统一内存网络(Unified Memory Network,UMN)的提出,CPU-GPU中的内存数据分配和线程调度能够互补,实现数据的高效利用,减少数据的远程流量,隐藏访问存储延迟.

3.2.3 互连架构

多GPU系统中GPU与CPU的互连方式通常有三种[48].

CPU内存网络(CPU Memory Network,CMN):在CMN中,CPU的内存不仅用于构建自己的内存网络,还利用该内存网络与GPU相连接,这种方法取代了PCIe总线,并且降低了CPU与GPU互连的通信开销,但是,每个GPU仍直接连接到本地内存.因此,访问远程GPU的内存仍然需要较高的延迟,但是这种方式通过内存网络消除了PCIe的带宽瓶颈.

GPU内存网络(GPU Memory Network,GMN):与CMN相比,GMN将所有GPU的本地内存互连在一起.CPU与GPU的接口与传统的多GPU系统保持相同,但GPU的内存被用于构建内存网络.因此,访问远程GPU的内存与CPU内存网络组织之间存在很大差异.在CMN中,远程GPU的内存访问都需要由远程GPU发送请求.对于传统的多GPU系统,请求通过PCIe通道发送到远程GPU进行访问,而GMN使GPU能够直接通过内存网络访问其他GPU的内存.根据远程内存的位置,请求在到达目标内存模块之前可能会经过一个或多个中间路由器(或HMC(Hybrid Memory Cube)).

统一内存网络(Unified Memory Network,UMN):CPU和GPU内存网络可以组合在一起创建单个统一内存网络(UMN).在这种网络组织中,CPU的内存和GPU的内存互连在一起,形成一个单一的内存网络.因此,可以通过遍历这个内存网络来访问系统中的任何一个存储器模块.在执行内核之前,需要将数据从CPU内存显式地传输到GPU内存.对于GMN(以及基线),数据通过PCIe传输到GPU,然后传输到GPU的内存.对于CMN,片上网络为CPU和GPU之间的数据传输提供更高的带宽,但仍然需要将数据复制到每个GPU的DRAM.然而,利用CPU和GPU之间的统一的存储器网络,CPU存储器和GPU存储器之间不一定需要数据传输,因为在CPU和GPU之间共享相同的存储器网络和存储器模块,并显式地在不同的内存模块对CPU和GPU数据进行分区,例如,CPU和GPU分别独占使用其本地存储器.然而,考虑到内存网络和虚拟寻址的可用性,所有物理内存都可以由CPU和GPU共享.

在CPU与多GPU的系统中,高效的数据共享和内存管理成为了一项新的挑战.当多个GPU协同执行同一应用程序时,设置合理的内存策略以实现GPU之间应用程序所需要的数据共享是提升性能的常见方法.Ziabari等人[99]提出了一种策略,将多个GPU单元视为一个单元,并将每个GPU的内存视作一个总体缓存单元,这种方式使得存在依赖性的内核能够互相访问数据,从而减少了CPU与GPU之间的数据传递次数,提高系统性能.

同时,为了解决GPU数据传输的序列化执行开销,Bhatotia等人[51]提出使用流水线的双缓冲技术来重叠通信与计算时间.他们在GPU内存中设置两个缓冲区分别用于数据传输和分块计算.同时,为了确保数据传输的异步性、减少分配开销以及避免缓存抖动,CPU缓冲区使用了环形的锁页内存.对于数据密集型应用程序,为了避免高并发导致的访存冲突,Shredder使用了线程合作机制从全局内存中将数据取到共享内存中.之后,Mokhtari等人[52]提出一种处理大数据集的CPU-GPU数据传输优化方案,他们使用四级流水线的数据预取技术解决了CPU-GPU数据传输的问题.这四级流水线分别为预取、地址生成、数据组装、数据传输和内核计算,并且在CPU端设置地址缓冲区和预取缓冲区.在预取地址生成阶段,GPU端内核将原来的读操作的访问地址转换为预取地址.按照GPU线程的访问地址按序存入CPU端的地址缓冲区中,这样可以实现连续的内存访问.同时,为了减少发送给CPU的地址信息,还采用了一个模式识别组件,将多个地址转换为一个"基地址+距离"的模式,减少发送数量,并且使GPU内核在计算当前数据时,保证了下一个流水线中的数据正在被传输,从而重叠了计算与传输.

随着应用程序数据量的增加,Sabne等人[53]提出了自动化流水线技术以重叠计算与数据传输的时间,并且提出了一种自动化计算划分技术(COmputation SPlitting,COSP)来解决数据量大于GPU内存大小的问题.COSP将一个大的计算划分为一些小的子计算,每个子计算所需的数据是整个数据的一部分,结合数据预取技术和流水线技术,一个子计算内核的计算和下一个子计算任务的数据传输是重叠的,从而隐藏数据请求的延迟,提升系统性能.对于单节点配多GPU的结构,Mohammed Sourouri[100]提出了一种优化节点内数据传输的方案,提出了目前处理节点内通信使用单MPI(Message Passing Interface)/OpenMP(Open Multi-Processing)线程或者双CUDA(Compute Unified Device Architecture)流的不足,并提出新的方法:结合使用多CUDA流和多OpenMP线程使得数据可以同时发送和接收.每个GPU采用多OpenMP线程是为了降低内核启动开销以及计算与通信的空闲等待时间.每一对GPU间采用多个CUDA流,GPU可以同时启动多个CUDA流分别对应不同的操作(发送边界数据,接收数据,计算内部数据),这样在同一个处理阶段中可以进行双向数据传输,同时内部数据的计算操作可以与数据传输同时进行.

3.3　GPU与GPU互连的非一致存储访问解决方案

多GPU互连时的非一致存储访问现象相对于CPU与GPU互连以及GPU片内的非一致存储访问更为复杂.在传统的GPU中,通常通过将远程数据页拷贝到本地以实现本地存储访问,从而避免远程存储访问开销.在多GPU系统中,当数据页同时被多个GPU共享时,这种数据页搬运技术非常低效,并且需要大量的报文以维护数据的一致性.另外,多GPU系统采用集中式虚拟内存管理单元,通过CPU端的IOMMU进行管理,因此GPU内部的TLB缺失必须要发送到CPU中进行查询,而这种远程页表查询的链路延迟可以达到2 μs以上,严重影响多GPU系统的性能.为了更好地降低多GPU系统的非一致存储访问的影响,需要通过软硬件协同设计和优化,从封装架构、内存布局和线程调度等角度出发,克服多GPU系统在存储架构上的性能瓶颈,提升多GPU系统的性能.

3.3.1　内存布局

在多GPU系统中,有效地减少无用数据占用有限的内存在多GPU系统中是实现高效计算的关键.为了达到这个目的,Ramashekar等人[32]通过在运行时寻找数据的交叉关系,并通过编译器辅助来进行识别、分配和管理应用程序的相交的数据.这种方法能够按照每个GPU上运行的计算任务所需要的数据量来分配数据,同时可以有效地跟踪缓存区的分配,从而最大化跨块的数据重用和多GPU机器上存储器的利用率.但是这种方法通过分析数组边界来分配数据,容易导致大量无用数据的传递,随后,Dashti等人[36]提出了对读共享数据进行复制操作来减少远程访问流量,Akhil等人[38]提出的首次访问数据页布局对于流数据访问效率具有较高效率.Kim等人[34]提出了一种识别独占访问数据的机制,并将该数据与访问该数据的线程块放在同一个GPU上,从而减少了GPU间的数据通信,同时通过访问数据块的位置来调度线程分配,从而保证数据页对齐,并通过应用轮询页面和子页面交错来提高访问的局部性.这种方法能够有效地提高整体性能,但只能应用于线程内部的局部性,没有考虑行局部性和列局部性等因素,并且需要额外的硬件子页面开销.Akhil等人[38]提出了L1.5缓存策略,专为远程数据缓存设计,这部分缓存是L2缓存中划分出来的,无需额外的硬件开销.通过缓存远程数据,这种方法降低了远程数据访

问的开销. 然而,随着数据量的扩大,L2缓存容量有限,如何平衡缓存与数据量的大小成为一个核心问题. 随着数据量的扩大,L2缓存的大小并不能完全缓存远程数据,需要对远程数据进行合理的选择,同时对于读写共享的数据页并没有很好地减少远程流量的方法. 由于读写共享的数据页的数据量远远超过了L2缓存的容量,它所造成的非一致存储访问影响要大得多,并且无法采用L1.5缓存等方法进行解决. 为了更好地实现数据的时间局部性,解决读写共享数据页所导致的非一致存储访问问题,Young等人发现在更细粒度对共享数据进行操作[39],可以将共享数据的规模缩小. 因此Young等人提出了CARVE,即把DRAM的部分存储区域划分为远程数据缓存,在缓存行粒度上对远程共享数据进行缓存. 同时,提出使用GPU-VI协议来保持读写共享数据的一致性,每次有数据写入共享数据页时,就会对该数据广播失效. 同时,由于读写共享数据页的属性在某些写操作之后可能转换为私有或者读共享,而私有数据和读共享数据不需要维持一致性的开销,因此,为了降低维护一致性的开销,Young等人还设置了一定概率对数据页进行随机转换. 这种方法通过在更细粒度上实现远程数据缓存,有效减少对远程数据的访问,提升了系统性能.

虽然CARVE可以将远程数据缓存在本地以减少远程数据流量,但是对于在多GPU系统上运行的不规则应用程序,如果完全依赖本地缓存来减少片外流量,可能会导致线程严重依赖L2缓存,从而引发严重的缓存抖动现象. 因此,如何提高这类不规则应用程序的性能是需要解决的主要问题之一. 为了提高不规则应用程序的性能,通常需要将保护数据的局部性与CARVE等方法结合起来,Khairy等人提出了cache remote的智能缓存管理技术[40],以改善这些不规则的工作负载. 该技术将访问分为RONCE(cache Remote ONCE)和RTWICE(cache Remote TWICE)两种模式. 在RTWICE中,远程数据请求会首先检查L2缓存,如果未命中,则请求会被重定向到数据所属的GPU之中,在这种情况下,远程读取请求被缓存处理两次,一次在存放数据的GPU的L2缓存,一次在发出请求的L2缓存中. 当数据被多个SM使用时,即具有warp间局部性(inter-warp locality)时,这种方法能获得一定的性能提升. 然而,由于GPU中流处理访问(Streaming Pattern)机制非常常见,针对每一组数据只在较短的一段时间内访问,且不重复访问,如果将这组数据搬运到本地存储器中,不仅会占用有限的内存资源,还会导致互连网络阻塞,造成性能下降. 因此,Khairy等人针对这种数据提出了RONCE,并提出编译器辅助的远程请求旁路. 在这种状况下,只需要具有数据的本地GPU进行一次缓存就可以完成该远程数据请求操作. 这种方法利用对不具有局部性的远程数据请求进行旁路,不进行本地数据缓存,从而减少了内存的占用.

CPU上提出过的动态页面迁移技术在GPU中很难取得很好的效果. 在GPU中进行动态页面分析和迁移的成本可能非常高,这是因为GPU执行的线程大多是并行的,且具有数据依赖性. 为确保数据的一致性,进行动态的页面迁移时必须刷新流水线,这会产生高昂的成本. 针对多GPU系统的非一致存储访问问题,目前主要通过内存数据静态分配策略进行优化,而很少通过数据动态迁移来实现. 然而,内存数据静态分配无法充分利用应用程序运行过程中产生的动态信息,因此具有一定的局限性,为更好地提升多GPU系统性能,必须降低动态迁移的成本. 一种最简单的动态迁移是数据预取. 与单GPU类似,数据预取并不会消耗过高的成本,但可能会造成内存资源的浪费. Dashti等人[36]提出在数据量不够多的情况下,对于读共享的数据采取复制到所有使用该数据页的GPU节点上,从而减少对读共享数据的移动,这种方法可能对于运行过程中可能改变的数据状态无法有效地应对,同时对于读写共享的数据无法进行有效地解决.

过去的GPU系统中,静态分配的主流方法是MCM(Multi-Chip Modules)[38]提出的首次访问数据页分配,即数据页在首次访问时由GPU分配到对应的GPU存储器. 然而,这种方法缺乏灵活性,且无法有效提升多GPU应用程序在数据共享访问下的性能. 同时,布局后的数据页将固定在该GPU之中,将来从其他GPU的访问将以缓存行粒度来进行. 这种方法存在以下问题:(1)首次访问的数据页在GPU应用程序开始运行时局部性较高,但随着应用程序的运行,局部性可能快速下降;(2)首次访问数据页布局可能会导致GPU内存负载严重不均衡,使某些GPU上缓存过多数据页,而其他GPU的数据页稀少;(3)FCFS(First Come First Serve)策略可能会导致多次远程数据迁移,引发刷新流水线和TLB shutdown的代价;(4)GPU流水线刷新的高成本阻碍了数据页动态迁移优化.

为了解决上述挑战,Baruah等人提出了几种解决方案[37]:(1)为了解决局部性的变化问题,提出了动态页面分类(Dynamic Page Classification,DPC),通过计算将页面分为五类,对不同页面采取不同的策略;(2)针对负载不均衡问题,提出了延迟的数据页布局,检测第一次接触数据页所在的内核的页面占用率,并根据这个占用率高低来决定是否进行迁移;(3)针对多次数据页迁移,提出了合并数据页迁移,等待页面请求达到一定数量再统一进行服务;(4)针对流水线刷新等的高成本,采取更细粒度的刷新策略,在迁移时,观察是否需

要彻底刷新计算单元和排空流水线,如果需要迁移的数据与流水线中正在执行的程序无关,就直接发出排空消息,只刷新对应地址的TLB(Translation Lookaside Buffer). 这几种方法通过考虑负载均衡来优化数据静态分配,通过降低GPU片内刷新流水线的成本来实现更好的数据迁移,降低互连通信的流量,提升了系统的性能.

同时,为了减少远程访问时间,最大化利用有限的外部带宽,Agarwal等人还提出了一种带宽感知数据页布局的方法[33]. 它能够根据存储器访问的次数、数据需求量以及系统中可用的总内存带宽来平衡合理的布局数据页,从而最大化利用多GPU的内外部内存带宽,提升系统性能.

Kumar等人[101]提出了一种硬件支持的将单GPU程序代码并行分配到多GPU系统上执行的方法. 其核心目标是对用户透明地实现最小化远程数据访问. 他们提出了一种数据位置感知的线程块调度器,用于调度GPU上大部分输入数据都在的线程块. 调度器利用GPU工作负载大概率迭代多次启动内核来处理大量数据这一特征,通过数据位置感知调度器利用这种可预测性来跟踪每个线程块对特定GPU卡的内存访问亲和性,并存储这些信息,以便为未来的迭代做出调度决策. 为了进一步减少远程访问的数量,他们还提出了一种混合机制,可以根据其访问行为在多个GPU的内存之间迁移或复制数据页. 实验表明,该方法使得大多数内存访问都是对本地GPU内存的访问.

3.3.2　线程调度

在多GPU系统中,CPU向GPU分配的计算任务与数据紧密关联,因此在任务分配前需要充分考虑线程和数据的关系,将线程调度机制和内存布局分配结合起来,使线程尽可能分配到数据对应的数据处理器中,以实现数据的局部性并减少GPU处理器间的通信量. 在多GPU系统中,传统的方案是采用两级轮转机制,即在多个GPU间轮转分配线程块,然后在各GPU内部的SM核之间进行轮转线程块分配. 这种方案打乱了线程块存在的数据局部性,导致系统性能下降. 为了更好地利用数据局部性,Kim等人提出了SKE(Scalable Kernel Execution)[48],这种方法通过提供一组GPU作为一个虚拟的GPU来提高可编程性. 在传统的GPU互连系统中,内核需要进行分区然后再在多GPU上进行,而SKE允许内核跨多个GPU执行,而不需要对内核进行分区,同时,建立了全局调度器和本地调度器,将线程块索引展开为一维网络,进行线性分配,从而提高相邻线程之间的局部性.

另外,由于GPU存储系统的设计目标是维持较高的吞吐量,因此应尽可能保持工作组具有较大的全局存储器访问粒度,即工作组中的工作项对全局存储器中数据的连续访问. 在多GPU的线程调度中,需要深入分析应用程序的数据和计算特征,以便更好地挖掘线程块与相应计算数据的关系,实现计算和数据的紧密耦合. Khairy等人通过将数据分为行共享、列共享和线程内部共享的数据[40],并建立表单记录每个内核访问的地址范围,同时将连续线程放在一起以提高局部性,并将访问同一地址的内核流出到同一个节点中. 这样可使线程和数据处在同一个节点,从而降低了访问远程数据的延迟.

3.3.3　互连架构

当前工业界的多GPU系统产品主要是将多个GPU直接通过总线互连在一起,缺少针对性的优化和管理设计,因此易用性和可执行性较差. 此外,非一致存储访问对应用程序在多GPU系统上执行的影响日益突出. 为了提高GPU的规模以提升系统性能,需要从体系结构出发,从通信的角度来对多GPU系统进行更有效的封装.

为了更好地提高远程访问的性能,减少GPU处理器间的通信,Akhil等人提出了一种封装级集成的多GPU模块技术MCM-GPU(Multi-Chip Modules GPU)[38]. 如图6所示,将单GPU划分为多个GPU模块,并将多个GPU模块集成在封装里,通过高带宽和高能效的信号传输技术(Ground Reference Signaling)互连,从而实现GPU扩展. MCM-GPU使多个GPU处理器共享同一个内存子系统,降低了数据一致性的开销. 为了提高MCM系统的性能,Akhil还提出了首次访问数据页布局的方法,将数据页布局在首次访问的处理器中,从而减少数据页的移动;并将L2缓存划分一部分作为L1.5缓存来缓存远程数据,L1.5缓存由GPM(General Purpose Modular)内的所有SM共享,本地访问会跳过L1.5缓存;同时提出了GPM线程调度,将连续的线程调度到同一个GPM中,从而提高数据局部性. 这几种方法可以更好地对GPU进行扩展,充分利用结构资源共享,同时不需要额外的编程工作,提升了系统的性能. 但是首次访问数据页的布局可能会导致负载不均衡以及数据局部性降低,L1.5缓存随着应用程序数据量的扩大而容量不足,这需要进一步的软硬件优化;同时这种方法没有考虑行局部性和列局部性,同时如果首次访问数据页并没有被处理器复用,而是被其他处理器复用,这种方案没有考虑运行过程中的动态信息,会产生额外的错误开销,并且线程调度通过轮询来实现,并未考虑数据页替换,无法充分利用数据局部性,从而导致MCM系统的程序运行一开始性能较高,但后续性能逐渐下降.

与MCM不同,Milic等人提出了一种NUMA(Non-

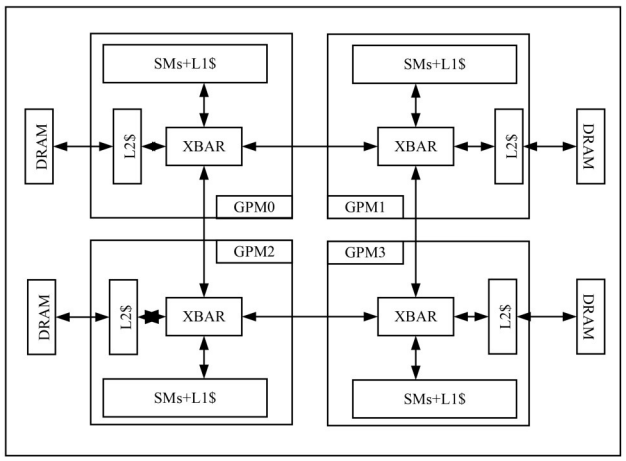

图6 GPM示意图

Uniform Memory Access)感知的GPU互连的方式[90],GPU与高带宽的交换机相连再与CPU互连,并通过交换机来合理地对带宽进行分配,从而提高系统的性能.这种方法延续了L1.5和首次访问数据页布局的思想,同时提出了一种缓存带宽分区策略:首先为本地数据和远程数据访问各分配一半的带宽,然后每隔一定的时间检测带宽使用状况,当外部带宽被占用饱和而内部带宽未饱和时,交换机进行带宽的重分配从而充分利用带宽资源.这种方法虽然能够更加充分利用GPU的带宽资源,但是对于不规则应用程序,这种方法的性能还需要进一步优化.同时,这种方法考虑线程内部的行局部性和列局部性,将内核网络和每个数据结构划分为了与GPU数量相同的N个连续的块,然后分配给各个GPU上,但是没有考虑行共享和线程内部的局部性,从而导致无法充分地利用数据局部性来更好地提升系统性能.

尽管业界对多GPU间通信机制进行了持续的研究,但在多GPU系统中进一步提升性能仍然是一项重大挑战[102].由于通过DMA这些大型传输模式进行GPU间通信会暂停GPU内核流水线,影响GPU的关键执行路径.而在内核执行期间通过细粒度的远程内存访问会导致高延迟,超过GPU通过多线程隐藏这些操作的能力.为了解决这些问题,NVIDIA提出了PROACT[103],通过联合编译时和运行时信息,利用点对点传输的灵活性和DMA(Direct Memory Access)传输的高效性,实现数据传输的优化.PROACT为开发人员提供了一个易于使用的P2P存储编程模型,利用配置文件和GPU运行时支持来跟踪数据生成,执行传输合并,并通过动态传输以实现高利用率.PROACT通过在数据生成后立即主动执行传输,有效重叠数据传输和计算.

4 总结与展望

为了应对高性能计算应用的需求,单GPU逐渐向多GPU发展以提升性能.多GPU系统相对于单GPU系统的性能并不是随着GPU数目的增长而线性提升,由于片内和片间数据带宽的巨大鸿沟,GPU处理器间的大量通信导致的非一致存储访问成为多GPU系统性能增长的天然屏障.随着大数据和深度学习的不断发展,应用程序的规模不断增长,同时应用程序的复杂度显著提高,多GPU系统处理器之间的数据交换也呈现不断增长的趋势,非一致存储访问带来的负面效应正日益凸显.大规模程序任务的运行特征决定了多GPU系统的数据交换不可避免.因此,如何提高数据交换的性能是多GPU系统下一步亟需解决的问题.

4.1 总结

由于GPU系统中,访问本地数据和远程数据存在非一致存储访问问题,而直接采用CPU的NUMA调度和内存布局策略已经难以满足GPU系统的性能和可扩展性要求.因此,为了减少非一致存储访问带来的性能损失并提升系统性能,GPU设计主要从减少远程访问流量和提高远程访问性能两个角度出发.

减少远程访问流量的关键在于提高数据的空间局部性和线程的并行性.通过实现数据的空间局部性,可以减少线程远程访问数据的次数.数据的空间局部性可以通过数据布局实现,即按照数据的使用线程将其存放在附近的位置.数据布局可以分为静态分配和动态调度两种方式.静态分配在应用程序运行前实现,可以解决数据的初始化分配问题,但缺乏灵活性,无法感知应用程序在运行过程中线程对数据需求的动态变化.动态调度可以感知应用程序对数据需求的动态变化,但需要考虑数据迁移带来的流水线停滞等问题.因此,降低动态调度开销并同时结合静态分配和动态调度两种方法是减少远程访问流量的有效途径.

增加远程访问性能通常通过减少远程访问的时间和隐藏远程访问延迟实现.减少远程访问的时间需要高效的互连架构和有效的多GPU封装形式.这主要集中在将线程和数据集中在一起,同时提高互连的通信带宽.在多GPU系统中,通常通过切换线程的方式隐藏数据访问延迟,从而减少远程访问所带来的性能损失.

4.2 展望

非一致存储访问主要源自于多GPU系统的存储管理问题.当前主要的解决方案是通过设计高效的内存布局机制、线程调度机制以及互连机制,减少数据通信流量并降低数据通信开销,从而缓解非一致存储访问.然而,这些方法没有过多地考虑多GPU系统的数据交换的存储管理机制,并且对多GPU系统的框架缺少针对性的优化和管理设计.此外,目前的解决方案还缺少

将软件和硬件相结合的优化方案. 使用软件解决方案具有较高的可配置性和灵活性,但缺乏对应用程序的特征感知,难以减少系统通信的数量或优化数据与计算的相对位置. 而单纯使用硬件解决方案,虽然可以高效并安全地收集数据访存特征和应用程序的行为特征,并根据行为特征对其进行优化,但在策略选择和实现方面相对固定,灵活性相对不足,因此如何分别利用好软件和硬件解决方案的优势,并将它们融合起来使用,变得尤其重要. 因此,在目前已有的解决方案的基础上,可以从以下几个方面进行设计优化.

如何设计高效的动态数据迁移机制来减少多GPU处理期间的数据交换? 动态访存感知的数据分类传输机制可以结合大规模程序任务的运行特征以调整数据访问和传输机制,有效地提升多GPU系统处理器间数据交换的效率. 针对动态数据迁移面临成本较高的问题,如何最小化动态迁移的成本是设计高效的动态数据迁移机制需要面临的主要问题.

如何降低远程页表查询开销? 当前多GPU系统中,当发生GPU内部的TLB缺失时,需要将请求发送到CPU端进行页表查询. 然而,由于多个租户可能共享GPU中的资源,在查找TLB时可能会造成严重的干扰,引发频繁的TLB缺失. 一次未命中可能导致数百个线程的长时间暂停,致使GPU无法调度足够的线程来隐藏访问延迟,从而降低系统吞吐量. 因此,如何通过高效的虚拟内存管理机制来降低远程页表查询的开销,避免多应用程序页表查询的争用,从而缓解非一致存储访问成为多GPU系统内存布局机制面临的挑战.

如何克服数据一致性的开销问题? 对于多GPU互连的系统,数据一致性是继续解决的关键问题. 在多GPU系统中,同一份数据可能在多个GPU内含有拷贝. 在应用程序运行过程中,如何确保数据一致性,同时最小化实现数据一致性的开销,是解决访问远程数据延迟的主要问题.

如何设计高效的封装架构来提升多GPU系统性能? 为了更好地提高GPU系统的规模以提升系统性能,需要设计更加高效的封装架构来减少数据通信量和通信成本. 提高多GPU性能的一种方法是通过板级和系统级集成提高应用程序性能,将多个最大尺寸的单片GPU封装到一个多GPU系统,实现多芯粒封装. 但是传统的GPU间互连方式导致的数据移动开销和能量消耗显著影响了多GPU系统的整体性能和能源效率. 因此,如何通过构建更加合理可靠高效的封装架构来提升多GPU系统性能是下一步主要的研究方向之一.

如何更好地提高外部带宽利用率? 在过去的工作中,远程数据访问包括直接数据行访问(以缓存行为单位,512 Byte)和按需取数据页(以数据页为单位,4 kB)两种方式. 两种方式的带宽利用率、取数据频率和延迟不同,不同的数据访问传输模式对系统性能有着不同的影响. 因此,如何通过高效的数据传输来最大化地利用有限的外部带宽成为多GPU系统存储管理设计的挑战.

如何通过特征感知进行非一致存储访问优化? 特征感知的核心思想是体系结构的优化应该基于在线收集应用程序行为特征和数据流特征,针对不同的应用程序特征进行针对性的优化. 这种方法通过感知应用程序运行的特征来进行数据分配和线程调度. 因此,如何针对应用程序的不同特征以及运行过程中的不同特征来进行针对性优化是多GPU系统性能提升的主要研究方向之一.

虽然上述挑战带来了诸多困难,但它们也为我们指引了多GPU系统的发展方向,当前和未来的研究除了进一步提高互连的性能之外,主要研究目标就是解决这些挑战问题. 结合系统中的封装架构问题和数据一致性的问题,未来主要有两个大的发展方向:一方面是结合软硬件的方法来提高系统访问远程数据的性能,降低非一致存储访问的影响;另一方面是支持多芯片数据一致性的方案,减少维持数据一致性的开销. 综上所述,提升多GPU系统的整体性能,关键在于如何降低非一致存储访问的开销. 这需要我们在未来的研究中不断探索和创新,以应对多GPU系统面临的挑战和机遇.

参考文献

[1] CHEN M, MAO S W, LIU Y H. Big data: A survey[J]. Mobile Networks and Applications, 2014, 19(2): 171-209.

[2] CHEN G, LI G, PEI S, et al. High performance computing via a GPU[C]//2009 First International Conference on Information Science and Engineering. Piscataway: IEEE, 2009: 238-241.

[3] RONG C M, LIU L, CHEN G L. Big data and smart computing: Methodology and practice[J]. Concurrency and Computation: Practice & Experience, 2016, 28(11): 3077-3078.

[4] KARNAGEL T, BEN-NUN T, WERNER M, et al. Big data causing big (TLB) problems: Taming random memory accesses on the GPU[C]//Proceedings of the 13th International Workshop on Data Management on New Hardware. New York: ACM, 2017: 1-10.

[5] KARNAGEL T, BEN-NUN T, WERNER M, et al. Big data causing big (TLB) problems: Taming random memory accesses on the GPU[C]//Proceedings of the 13th Interna-

tional Workshop on Data Management on New Hardware (DAMON'17). New York: ACM, 2017: 6.

[6] CHRISTOPHE E, MICHEL J, INGLADA J. Remote sensing processing: From multicore to GPU[J]. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2011, 4(3): 643-652.

[7] MIELIKAINEN J, HUANG B, WANG J, et al. Compute unified device architecture (CUDA)-based parallelization of WRF Kessler cloud microphysics scheme[J]. Computers & Geosciences, 2013, 52: 292-299.

[8] KERR A, DAN C, RICHARDS M. QR decomposition on GPUs[C]//Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units (GPGPU-2). New York: ACM, 2009: 71-78.

[9] STONE S S, HALDAR J P, TSAO S C, et al. Accelerating advanced MRI reconstructions on GPUs[C]//Proceedings of the 5th Conference on Computing frontiers. New York: ACM, 2008: 261-272.

[10] 张舒. 模式识别并行算法与GPU高速实现研究[D]. 成都: 电子科技大学, 2009.
ZHANG S. Research on Parallel Algorithm of Pattern Recognition and High-speed Implementation of GPU[D]. Chengdu: University of Electronic Science and Technology of China, 2009. (in Chinese)

[11] LAI L Y, TSAI K H, LI H W. GPGPU-based ATPG system: Myth or reality?[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(1): 239-247.

[12] MARSHALL S, VANHOY G, AKOGLU A, et al. GPGPU based parallel implementation of spectral correlation density function[J]. Journal of Signal Processing Systems, 2020, 92(1): 71-93.

[13] BURTSCHER M, NASRE R, PINGALI K. A quantitative study of irregular programs on GPUs[C]//2012 IEEE International Symposium on Workload Characterization (IISWC). Piscataway: IEEE, 2012: 141-151.

[14] GARCIA V, DEBREUVE E, BARLAUD M. Fast k nearest neighbor search using GPU[C]//2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. Piscataway: IEEE, 2008: 1-6.

[15] GARCIA V, DEBREUVE E, NIELSEN F, et al. K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching[C]//2010 IEEE International Conference on Image Processing. Piscataway: IEEE, 2010: 3757-3760.

[16] SHALF J. HPC interconnects at the end of Moore's law [C]//Optical Fiber Communication Conference (OFC) 2019. Washington: OSA, 2019: 1-9.

[17] KNOCHEL U. 3D integration: Opportunities, design challenges and approaches[C]//2012 IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS). Piscataway: IEEE, 2012: 4.

[18] ZHOU C, ZHANG T. High performance graph data imputation on multiple GPUs[J]. Future Internet, 2021, 13 (2): 36.

[19] WANG K B, DING X N, LEE R B, et al. GDM: Device memory management for gpgpu computing[C]//The 2014 ACM International Conference on Measurement and Modeling of Computer Systems. New York: ACM, 2014: 533-545.

[20] LASHGAR A, SALEHI E, BANIASADI A. A case study in reverse engineering GPGPUs: Outstanding memory handling resources[EB/OL]. (2015-09-04) [2023-06-13]. https://doi.org/10.1145/2927964.2927968.

[21] O´NEIL M A, BURTSCHER M. Microarchitectural performance characterization of irregular GPU kernels[C]//2014 IEEE International Symposium on Workload Characterization (IISWC). Piscataway: IEEE, 2014: 130-139.

[22] KIM J, KIM H, LEE J H, et al. Achieving a single compute device image in OpenCL for multiple GPUs[J]. ACM SIGPLAN Notices, 2011, 46(8): 277-288.

[23] CABEZAS J, VILANOVA L, GELADO I, et al. Automatic parallelization of kernels in shared-memory multi-GPU nodes[C]//Proceedings of the 29th ACM on International Conference on Supercomputing. New York: ACM, 2015: 3-13.

[24] SAKAI R, INO F, HAGIHARA K. Towards automating multi-dimensional data decomposition for executing a single-GPU code on a multi-GPU system[C]//2016 Fourth International Symposium on Computing and Networking (CANDAR). Piscataway: IEEE, 2016: 408-414.

[25] AGARWAL N, NELLANS D, O'CONNOR M, et al. Unlocking bandwidth for GPUs in CC-NUMA systems[C]//2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA). Piscataway: IEEE, 2015: 354-365.

[26] ZHENG T H, NELLANS D, ZULFIQAR A, et al. Towards high performance paged memory for GPUs[C]//2016 IEEE International Symposium on High Performance Computer Architecture (HPCA). Piscataway: IEEE, 2016: 345-357.

[27] PICCOLI G, SANTOS H N, RODRIGUES R E, et al.

Compiler support for selective page migration in NUMA architectures[C]//Proceedings of the 23rd International Conference on Parallel Architectures and Compilation. New York: ACM, 2014: 369-380.

[28] TIAN Y Y, PUTHOOR S, GREATHOUSE J L, et al. Adaptive GPU cache bypassing[C]//Proceedings of the 8th Workshop on General Purpose Processing Using GPUs. New York: ACM, 2015: 25-35.

[29] KOO G, OH Y, RO W W, et al. Access pattern-aware cache management for improving data utilization in GPU [C]//Proceedings of the 44th Annual International Symposium on Computer Architecture. New York: ACM, 2017: 307-319.

[30] DUONG N, ZHAO D L, KIM T, et al. Improving cache management policies using dynamic reuse distances[C]// 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture. Piscataway: IEEE, 2012: 389-400.

[31] IBRAHIM M A, KAYIRAN O, ECKERT Y, et al. Analyzing and leveraging shared L1 caches in GPUs[C]//Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques. New York: ACM, 2020: 161-173.

[32] RAMASHEKAR T, BONDHUGULA U. Automatic data allocation and buffer management for multi-GPU machines[J]. ACM Transactions on Architecture and Code Optimization, 10(4): 60.

[33] AGARWAL N, NELLANS D, STEPHENSON M, et al. Page placement strategies for GPUs within heterogeneous memory systems[J]. ACM SIGPLAN Notices, 2015, 50 (4): 607-618.

[34] KIM H, HADIDI R, NAI L F, et al. CODA: Enabling colocation of computation and data for multiple GPU systems[J]. ACM Transactions on Architecture and Code Optimization, 15(3): 32.

[35] BEN-NUN T, LEVY E, BARAK A, et al. Memory access patterns: The missing piece of the multi-GPU puzzle [C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New York: ACM, 2015: 1-12.

[36] DASHTI M, FEDOROVA A, FUNSTON J, et al. Traffic management: A holistic approach to memory placement on NUMA systems[C]//Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2013: 381-394.

[37] BARUAH T, SUN Y F, DINCER A T, et al. Griffin: hardware-software support for efficient page migration in multi-GPU systems[C]//2020 IEEE International Symposium on High Performance Computer Architecture (HPCA). Piscataway: IEEE, 2020: 596-609.

[38] ARUNKUMAR A, BOLOTIN E, CHO B, et al. MCM-GPU: Multi-chip-module GPUs for continued performance scalability[C]//Proceedings of the 44th Annual International Symposium on Computer Architecture. New York: ACM, 2017: 320-332.

[39] YOUNG V, JALEEL A, BOLOTIN E, et al. Combining HW/SW mechanisms to improve NUMA performance of multi-GPU systems[C]//Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture. New York: ACM, 2018: 339-351.

[40] KHAIRY M, NIKIFOROV V, NELLANS D, et al. Locality-centric data and threadblock management for massive GPUs[C]//2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). Piscataway: IEEE, 2020: 1022-1036.

[41] LUK C K, HONG S, KIM H. Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping[C]//Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture. New York: ACM, 2009: 45-55.

[42] LEE Y W, Y J HEO, CHO C S, et al. Open-CL based multi GPU acceleration for video object detection[C]// 2021 IEEE International Conference on Consumer Electronics (ICCE). Piscataway: IEEE, 2021: 1-3.

[43] VEGA A, BUYUKTOSUNOGLU A, BOSE P. Transparent CPU-GPU collaboration for data-parallel kernels on heterogeneous systems[C]//Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques. Piscataway: IEEE, 2013: 245-255.

[44] TANG X L, PATTNAIK A, JIANG H P, et al. Controlled kernel launch for dynamic parallelism in GPUs[C]//2017 IEEE International Symposium on High Performance Computer Architecture (HPCA). Piscataway: IEEE, 2017: 649-660.

[45] PARK J J K, PARK Y, MAHLKE S. Dynamic resource management for efficient utilization of multitasking GPUs [C]//Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2017: 1-5.

[46] LI A, SONG S L, LIU W F, et al. Locality-aware CTA clustering for modern GPUs[C]//Proceedings of the

Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2017: 297-311.

[47] PATTNAIK A, TANG X L, JOG A, et al. Scheduling techniques for GPU architectures with processing-in-memory capabilities[C]//Proceedings of the 2016 International Conference on Parallel Architectures and Compilation. New York: ACM, 2016: 31-44.

[48] KIM G, LEE M, JEONG J, et al. Multi-GPU system design with memory networks[C]//Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture. New York: ACM, 2014: 484-495.

[49] LEE J, SAMADI M, MAHLKE S. VAST: The illusion of a large memory space for GPUs[C]//2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT). Piscataway: IEEE, 2014: 443-454.

[50] JI F, LIN H, MA X. RSVM: A region-based software virtual memory for GPU[C]//Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques. Piscataway: IEEE, 2013: 269-278.

[51] BHATOTIA P, RODRIGUES R, VERMA A. Shredder: GPU-accelerated incremental storage and computation [C]//Proceedings of the 10th USENIX conference on File and Storage Technologies. New York: ACM, 2012: 14.

[52] MOKHTARI R, STUMM M. BigKernel: High performance CPU-GPU communication pipelining for big data-style applications[C]//2014 IEEE 28th International Parallel and Distributed Processing Symposium. Piscataway: IEEE, 2014: 819-828.

[53] SABNE A, SAKDHNAGOOL P, EIGENMANN R. Scaling large-data computations on multi-GPU accelerators [C]//Proceedings of the 27th International ACM Conference on Supercomputing. New York: ACM, 2013: 443-454.

[54] KANG S, FENDER A, EATON J, et al. Computing PageRank scores of web crawl data using DGX A100 clusters [C]//2020 IEEE High Performance Extreme Computing Conference (HPEC). Piscataway: IEEE, 2020: 1-4.

[55] JANG H, KIM J, GRATZ P, et al. Bandwidth-efficient on-chip interconnect designs for GPGPUs[C]//Proceedings of the 52nd Annual Design Automation Conference. New York: ACM, 2015: 1-6.

[56] WANG J F, WANG Q, JIANG L, et al. IBOM: An integrated and balanced on-chip memory for high performance GPGPUs[J]. IEEE Transactions on Parallel and Distributed Systems, 2018, 29(3): 586-599.

[57] CHUN K C, KIM Y K, RYU Y, et al. A 16-GB 640-GB/s HBM2E DRAM with a data-bus window extension technique and a synergetic on-die ECC scheme[J]. IEEE Journal of Solid-State Circuits, 2021, 56(1): 199-211.

[58] LI A, SONG S L, CHEN J Y, et al. Evaluating modern GPU interconnect: PCIe, NVLink, NV-SLI, NVSwitch and GPUDirect[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(1): 94-110.

[59] SHARMA D DAS. PCI express 6.0 specification: A low-latency, high-bandwidth, high-reliability, and cost-effective interconnect with 64.0 GT/s PAM-4 signaling[J]. IEEE Micro, 2021, 41(1): 23-29.

[60] LI A, SONG S L, CHEN J Y, et al. Tartan: Evaluating modern GPU interconnect via a multi-GPU benchmark suite[C]//2018 IEEE International Symposium on Workload Characterization (IISWC). Piscataway: IEEE, 2018: 191-202.

[61] HU Y C, LU L. Design of a simulation model for high performance LINPACK in hybrid CPU-GPU systems[J]. The Journal of Supercomputing, 2021, 77(12): 13739-13756.

[62] NORMI A H, SUHAILA A H, NORMA A. Statistical filtering on 3D cloud data points on the CPU-GPU platform [J]. Journal of Physics: Conference Series, 2021, 1770: 012006.

[63] SOUZA R, FERNANDES A, TEIXEIRA T S F X, et al. Online multimedia retrieval on CPU-GPU platforms with adaptive work partition[J]. Journal of Parallel and Distributed Computing, 2021, 148: 31-45.

[64] SOURAVLAS S, SIFALERAS A, KATSAVOUNIS S. Hybrid CPU-GPU community detection in weighted networks[J]. IEEE Access, 2020, 8: 57527-57551.

[65] YIN F, SHI F. Cluster optimization algorithm based on CPU and GPU hybrid architecture[J]. Cluster Computing, 2022, 25(4): 2601-2611.

[66] AGARWAL N, NELLANS D, EBRAHIMI E, et al. Selective GPU caches to eliminate CPU-GPU HW cache coherence[C]//2016 IEEE International Symposium on High Performance Computer Architecture (HPCA). Piscataway: IEEE, 2016: 494-506.

[67] ZHANG S Q, QIN Z, YANG Y H, et al. Transparent partial page migration between CPU and GPU[J]. Frontiers of Computer Science, 2020, 14(3): 13.

[68] SUN Y F, BARUAH T, MOJUMDER S A, et al. MGPUSim: Enabling multi-GPU performance modeling and op-

timization[C]//Proceedings of the 46th International Symposium on Computer Architecture. New York: ACM, 2019: 197-209.

[69] LI C, AUSAVARUNGNIRUN R, ROSSBACH C J, et al. A framework for memory oversubscription management in graphics processing units[C]//Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2019: 49-63.

[70] GANGULY D, ZHANG Z Y, YANG J, et al. Interplay between hardware prefetcher and page eviction policy in CPU-GPU unified virtual memory[C]//Proceedings of the 46th International Symposium on Computer Architecture. New York: ACM, 2019: 224-235.

[71] LINDHOLM E, NICKOLLS J, OBERMAN S, et al. NVIDIA tesla: A unified graphics and computing architecture[J]. IEEE Micro, 2008, 28(2): 39-55.

[72] BORDAT A, DOBIAS P, KERNEC J L, et al. GPU based implementation for the pre-processing of radar-based human activity recognition[C]//25th Euromicro Conference on Digital System Design (DSD). Piscataway: IEEE, 2022: 593-598.

[73] 田绪红, 陈茂资, 田金梅. DirectX发展及相关GPU通用计算技术综述[J]. 计算机工程与设计, 2009, 30(23): 5432-5436, 5559.
TIAN X H, CHEN M Z, TIAN J M. Survey of development of DirectX and GPGPU[J]. Computer Engineering and Design, 2009, 30(23): 5432-5436, 5559. (in Chinese)

[74] 吴俊杰. 层次存储的访问分析与优化方法研究: 重用性、相似性与亲和性[D]. 长沙: 国防科学技术大学, 2009.
WU J J. Research on Analysis and Optimization of Data Access for Memory Hierarchy[D]. Changsha: National University of Defense Technology, 2009. (in Chinese)

[75] BOLOSKY W, FITZGERALD R, SCOTT M. Simple but effective techniques for NUMA memory management[J]. ACM SIGOPS Operating Systems Review, 1989, 23(5): 19-31.

[76] BLAGODUROV S, ZHURAVLEV S, FEDOROVA A, et al. A case for NUMA-aware contention management on multicore systems[C]//Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques. New York: ACM, 2010: 557-558.

[77] DAS R, AUSAVARUNGNIRUN R, MUTLU O, et al. Application-to-core mapping policies to reduce memory interference in multi-core systems[C]//Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques. New York: ACM, 2012: 455-456.

[78] FALSAFI B, WOOD D A. Reactive NUMA: A design for unifying S-COMA and CC-NUMA[C]//Proceedings of the 24th Annual International Symposium on Computer Architecture. New York: ACM, 1997: 229-240.

[79] HARDAVELLAS N, FERDMAN M, FALSAFI B, et al. Reactive NUCA[J]. ACM SIGARCH Computer Architecture News, 2009, 37(3): 184-195.

[80] LI H, TANDRI S, STUMM M, et al. Locality and loop scheduling on NUMA multiprocessors[C]//1993 International Conference on Parallel Processing - ICPP'93 Vol2. Piscataway: IEEE, 1993: 140-147.

[81] SAULSBURY A, WILKINSON T, CARTER J, et al. An argument for simple COMA[J]. Future Generation Computer Systems, 1995, 11(6): 553-566.

[82] TAM D, AZIMI R, STUMM M. Thread clustering[J]. ACM SIGOPS Operating Systems Review, 2007, 41(3): 47-58.

[83] ROY P, SONG S L, KRISHNAMOORTHY S, et al. NUMA-caffe[J]. ACM Transactions on Architecture and Code Optimization, 2018, 15(2): 1-26.

[84] ZHANG W, JIANG Z H, CHEN Z G, et al. NUMA-aware DGEMM based on 64-bit ARMv8 multicore processors architecture[J]. Electronics, 2021, 10(16): 1984.

[85] VERGHESE B, DEVINE S, GUPTA A, et al. Operating system support for improving data locality on CC-NUMA compute servers[J]. ACM SIGOPS Operating Systems Review, 1996, 30(5): 279-289.

[86] SONG W, JUNG H J, J H AHN, et al. Evaluation of performance unfairness in NUMA system architecture[J]. IEEE Computer Architecture Letters, 2017, 16(1): 26-29.

[87] KAESTLE S, ACHERMANN R, ROSCOE T, et al. Shoal: Smart allocation and replication of memory for parallel programs[C]//Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC'15). New York: USENIX Association, 2015: 263-276.

[88] BAPTISTE L, VIVIEN Q, ALEXANDRA F. Thread and memory placement on NUMA systems: Asymmetry matters [C]//Proceedings of the 2015 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC'15). New York: USENIX Association, 2015: 277-289.

[89] LIU M, LI T. Optimizing virtual machine consolidation performance on NUMA server architecture for cloud

workloads[C]//2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA). Piscataway: IEEE, 2014: 325-336.
[90] MILIC U, VILLA O, BOLOTIN E, et al. Beyond the socket: NUMA-aware GPUs[C]//Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. New York: ACM, 2017: 123-135.
[91] LAMAAZI H, MIZOUNI R, OTROK H, et al. Smart-3DM: Data-driven decision making using smart edge computing in hetero-crowdsensing environment[J]. Future Generation Computer Systems, 2022, 131: 151-165.
[92] WANG G Y, WANG Y. 3DM: Domain-oriented data-driven data mining[J]. Fundamenta Informaticae, 2009, 90(4): 395-426.
[93] TOPOL A W, LA TULIPE D C, SHI L, et al. Three-dimensional integrated circuits[J]. IBM Journal of Research and Development, 2006, 50(4/5): 491-506.
[94] 吴际, 谢冬青. 三维集成技术的现状和发展趋势[J]. 现代电子技术, 2014, 37(6): 104-107.
WU J, XIE D Q. Current status and trends of three-dimensional integrated technology[J]. Modern Electronics Technique, 2014, 37(6): 104-107. (in Chinese)
[95] LOH G H, XIE Y, BLACK B. Processor design in 3D die-stacking technologies[J]. IEEE Micro, 2007, 27(3): 31-48.
[96] STRENGERT M, MÜLLER C, DACHSBACHER C, et al. CUDASA: Compute unified device and systems architecture[C]//Proceedings of the 8th Eurographics Conference on Parallel Graphics and Visualization. New York: ACM, 2008: 49-56.
[97] ZHU Z C, XU S Z, TANG J, et al. GraphVite: A high-performance CPU-GPU hybrid system for node embedding[C]//The World Wide Web Conference. New York: ACM, 2019: 2494-2504.
[98] GONG L, ZHANG C, DUAN L, et al. Nonrigid image registration using spatially region-weighted correlation ratio and GPU-acceleration[J]. IEEE Journal of Biomedical and Health Informatics, 2019, 23(2): 766-778.
[99] ZIABARI A K, SUN Y F, MA Y N, et al. UMH: A hardware-based unified memory hierarchy for systems with multiple discrete GPUs[J]. ACM Transactions on Architecture and Code Optimization, 2016, 13(4): 35.
[100] SOUROURI M, GILLBERG T, BADEN S B, et al. Effective multi-GPU communication using multiple CUDA streams and threads[C]//2014 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS). Piscataway: IEEE, 2014: 981-986.
[101] MATAM K K, MOHAMMAD R A, MURALI A. Efficient automatic parallelization of a single GPU program for a multiple GPU system[J]. Integration, 2019, 66: 35-43.
[102] MUTHUKRISHNAN H, LUSTIG D, VILLA O, et al. FinePack: Transparently improving the efficiency of fine-grained transfers in multi-GPU systems[C]//2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA). Piscataway: IEEE, 2023: 516-529.
[103] MUTHUKRISHNAN H, NELLANS D, LUSTIG D, et al. Efficient multi-GPU shared memory via automatic optimization of fine-grained transfers[C]//2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA). Piscataway: IEEE, 2021: 139-152.

作者简介

李 晨 男,1989年11月出生于江西省九江市. 博士,现为国防科技大学计算机学院助理研究员. 主要研究方向为微处理器设计.
E-mail: lichen@nudt.edu.cn

刘 畅 男,1988年10月出生于内蒙古巴彦淖尔市. 现为国防科技大学计算机学院助理研究员,从事集成电路设计与验证等工作.
E-mail: nymph@uestc.edu.cn

葛一漩 男,1998年12月出生于山东省菏泽市. 现为国防科技大学计算机学院硕士. 主要研究方向为微处理器设计.
E-mail: geyixuan16@nudt.edu.cn

郭 阳 男,1971年出生. 现任国防科技大学计算机学院研究员,博士生导师,主要研究方向为微处理器设计.
E-mail: guoyang@nudt.edu.cn