GPGPU 和 CUDA 统一内存研究现状综述

庞文豪[1]，王嘉伦[2]，翁楚良[1*]

（1.华东师范大学数据科学与工程学院，上海 200062；2.之江实验室交叉创新研究院，浙江 杭州 310000）

摘 要： 在大数据背景下，随着科学计算、人工智能等领域的快速发展，各领域对硬件的算力要求越来越高。图形处理器（GPU）特殊的硬件架构，使其适合进行高并行度的计算，并且近年来 GPU 与人工智能、科学计算等领域互相发展促进，使 GPU 功能细化，逐渐发展出了成熟的通用图形处理器（GPGPU），目前 GPGPU 已成为中央处理器（CPU）最重要的协处理器之一。然而，GPU 硬件配置在出厂后不容易更改且显存容量有限，在处理大数据集时显存容量不足的缺点对计算性能造成较大的影响。统一计算设备架构（CUDA）6.0 推出了统一内存，使 GPGPU 和 CPU 可以共享虚拟内存空间，以此来简化异构编程和扩展 GPGPU 可访问的内存空间。统一内存为 GPGPU 处理大数据集提供了一项可行的解决方案，在一定程度上缓解了 GPU 显存容量较小的问题，但是统一内存的使用也带来了一些性能问题，如何在统一内存中做好内存管理成为性能提升的关键。本研究对 CUDA 统一内存的发展和应用进行综述，包括 CUDA 统一内存的特性、发展、优势和局限性以及在人工智能、大数据处理系统等领域的应用和未来的发展前景，为未来使用和优化 CUDA 统一内存的研究工作提供有价值的参考。

关键词： 通用图形处理器；统一内存；显存超额订阅；数据管理；异构系统

中图分类号： TP316 **文献标志码：** A **DOI：** 10.19678/j.issn.1000-3428.0068694

Survey on GPGPU and CUDA Unified Memory Research Status

PANG Wenhao[1], WANG Jialun[2], WENG Chuliang[1*]

(1. School of Data Science and Engineering, East China Normal University, Shanghai 200062, China;
2. Research Institute of Interdisciplinary Innovation, Zhejiang Laboratory, Hangzhou 310000, Zhejiang, China)

【Abstract】 In the context of big data, the rapid advancement of fields such as scientific computing and artificial intelligence, there is an increasing demand for high computational power across various domains. The unique hardware architecture of the Graphics Processing Unit (GPU) makes it suitable for parallel computing. In recent years, the concurrent development of GPUs and fields such as artificial intelligence and scientific computing has enhanced GPU capabilities, leading to the emergence of mature General-Purpose Graphics Processing Units (GPGPUs). Currently, GPGPUs are one of the most important co-processors for Central Processing Units (CPUs). However, the fixed hardware configuration of the GPU after delivery and its limited memory capacity can significantly hinder its performance, particularly when dealing with large datasets. To address this issue, Compute Unified Device Architecture (CUDA) 6.0 introduces unified memory, allowing GPGPU and CPU to share a virtual memory space, thereby simplifying heterogeneous programming and expanding the GPGPU-accessible memory space. Unified memory offers a solution for processing large datasets on GPGPUs and alleviates the constraints of limited GPGPU memory capacity. However, the use of unified memory introduces performance issues. Effective data management within unified memory is the key to enhancing performance. This article provides an overview of the development and application of CUDA unified memory. It covers topics such as the features and evolution of unified memory, its advantages and limitations, its applications in artificial intelligence and big data processing systems, and its prospects. This article provides a valuable reference for future work on applying and optimizing CUDA unified memory.

【Key words】 General-Purpose Graphics Processing Unit (GPGPU); unified memory; memory oversubscription; data management; heterogeneous system

0 引言

近年来，随着人工智能、科学计算等领域的快速发展。图形处理器（GPU）变得越来越重要[1]。GPU 最初是一种专门用于加速图像渲染的处理器。与传统的中央处理器（CPU）相比，GPU 拥有数以千计的处理核心，能够同时执行大量的计算任务。这种并行计算的特性使得 GPU 在处理大规模数据集

收稿日期： 2023-10-25 **修回日期：** 2024-01-24

基金项目： 国家自然科学基金（62272171）；浙江省“尖兵”“领雁”研发攻关计划（2022C04006）。

通信作者 E-mail： * clweng@dase.ecnu.edu.cn

和计算密集型任务时表现出色。深度学习等人工智能相关技术通常涉及大量的矩阵运算和张量操作，这些操作可高度并行化，因此 GPU 的用途被扩展到了更广泛的计算领域。

通用图形处理器(GPGPU)是 GPU 的一种，相比于 GPU 应用于图像处理，GPGPU 更多应用于通用计算任务。由于 GPU 强大的并行计算能力和高带宽内存的特性，人们开始探索将 GPU 应用于其他领域的通用计算。GPGPU 的核心思想是利用 GPU 的大规模并行能力来加速可并行化的计算密集型任务。

目前，GPGPU 已应用在包括科学计算、数据分析、人工智能、密码学、图像处理等在内的多个领域。GPU 强大的并行能力可以加速这些应用负载中繁重的计算任务，提高运算性能和运算效率。

相比于 CPU 的架构[2]，GPU 的架构设计将更多的物理空间用于计算单元而非控制单元。以下是 CPU 和 GPU 的对比介绍：

1)设计目标。GPU 是为图形处理等可高度并行的计算任务而设计的，其处理核心多，支持大规模并行计算任务。CPU 的设计目标则注重单线程性能和通用任务处理能力。

2)处理核心。GPU 拥有大量的处理核心，这些核心设计是为了执行相同的指令，处理不同的数据，从而实现并行加速。CPU 的处理核心数量较少，通常以多个物理或逻辑核心的形式存在，每个核心都能独立处理执行不同的指令。

3)内存层次结构。GPU 具有高带宽的全局内存和较大的片上共享内存，可以满足高性能并行计算的需求，同时 GPU 还具有高速的专用缓存，用于对指令和数据进行存储和一定程度的共享。CPU 通常具有较小但较快速的缓存以及较小的内存。

4)内存访问特性。GPU 高带宽的全局内存可以满足数据集的快速访问需求。CPU 由于需要更大程度地顾及通用性且遵守各类接口规范，无法激进地升级内存和 I/O 等，并且片上空间有限，因此通常具有较小的高速缓存和较低的内存带宽。

综上所述，相较于 CPU，GPU 具有不同的设计架构和更好的处理并行计算的能力，这样的特性在科学计算、人工智能等应用场景中具有更好的性能。但是，GPU 作为一个独立硬件，显存容量通常是固定的，并且无法像 CPU 的内存一样进行灵活的扩展。在上层应用所需的数据集越来越大的今天，GPU 强大的并行计算能力和有限的显存容量之间的矛盾更加突出。为了缓解这个矛盾，NVIDIA 公司推出了统一内存[3-4]，用于简化统一计算设备架构(CUDA)编程和更好的拓展 GPU 的显存容量以让其更好地处理更大规模的数据。

与 CUDA 的统一内存类似，开放计算语言(OpenCL)推出共享虚拟内存(SVM)[5]和统一共享内存(USM)[6]等技术来缓解 GPU 显存容量低和异构编程复杂等问题。因为 CUDA 应用更加广泛且关于 CUDA 统一内存的相关研究更多，本文主要对 CUDA 统一内存和在统一内存中的研究工作进行整理介绍。

本文将对 CUDA 统一内存的发展进行梳理，同时整理统一内存的原理和特性，分析其优缺点，并对在 CUDA 统一内存上的研究工作进行总结，最后分析其未来发展研究可能的趋势。

1 GPU 国内外主要厂商

本节将对国内外主要的 GPU 厂商做简要介绍，包括国际领先的 GPU 公司、国内的主要厂家和部分初创公司。

1.1 国外 GPU 厂商

GPU 芯片的全球供应链被美国、欧盟、英国、日韩等占据，由于起步较早，它们位于全球产业链的核心环节，主导着全球 GPU 供应。国际上知名的芯片设计公司主要包括 NVIDIA、AMD、Intel、ARM、高通等公司。目前，NVIDIA 占据着独立 GPU 市场绝大部分的市场份额[7]。Intel 凭借其在 CPU 市场的优势，在集成 GPU 领域的市场占有率优势明显[7]。

NVIDIA 公司是全球领先的半导体公司，于 1993 年成立。其主要业务是设计和制造高性能的 GPU 以及围绕其产品的相关技术。NVIDIA 于 1999 年发布图形芯片，提出 GPU 的概念，并在 2000 年收购 3dfx，后者 1995 年的产品 Voodoo 是真正意义上的第一款消费级 3D 显卡。NVIDIA 在 2007 年推出了 CUDA，这是一种针对 NVIDIA 公司的 GPU 研发的软硬件整合技术。通过使用 CUDA，用户可以简单直接地使用 NVIDIA GPU 来进行通用计算。凭借着 CUDA 不断改进升级和近些年来人工智能、大数据等领域的快速发展，NVIDIA 在独立 GPU 设计和研发上持续发力，成为目前全球最成功的 GPU 公司之一，占据了高端 GPU 的绝大部分的市场份额。

图 1 展示了 NVIDIA 近年来 GPU 的架构发展。每一次的架构升级都是针对市场算力的发展需

要做出的针对性的硬件设计和改进。如 2017 年发布的 Volta 架构，拥有更好的高性能计算（HPC）的性能，Volta 架构也是第一个支持 Tensor Core 的架构，专为人工智能运算和高性能计算所设计。

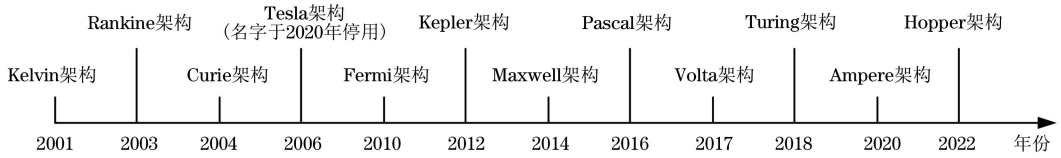


图 1 NVIDIA GPU 架构发展

Fig. 1 Development of NVIDIA GPU architecture

AMD 公司的 GPU 产品占据着较多的全球 GPGPU 市场份额[7]。2006 年 AMD 收购了 ATI，也开启了 AMD 独立芯片研发的新阶段。ATI 是成立于 1985 年的 3D 图形及多媒体技术公司。相比于 NVIDIA 数据中心的高端 GPU，AMD 显卡的性价比更高。此外，在软件生态上 AMD 也具有更好的兼容性和性能。

1.2 国内 GPU 厂商

相较于国际上的 GPU 的发展，国内 GPU 起步较晚。近些年来，随着人工智能等领域的飞速发展，市场对算力的需求越来越高，国内涌现出了多个国产 GPU 厂商。

沐曦集成公司[8]致力于设计具有完全自主知识产权、针对异构计算等各类应用的高性能 GPU 芯片。旗下的产品覆盖了人工智能、云计算、数据中心等高性能异构计算领域。其创始团队成员主要来自 AMD 等国际公司，有丰富的高性能 GPU 产品的研发经验。目前公司推出了用于人工智能推理、训练与通用计算等能够满足数据中心高能效和高通用性算力需求的 GPU 产品。

摩尔线程公司[9]致力于通用图形计算和高性能计算，旨在研发全功能 GPU 芯片和相关产品，着力构建中国视觉计算和人工智能领域的计算平台。设计的全功能 GPU 支持智能加速计算、物理仿真与科学计算等多种组合工作负载，同时能够兼顾算效与算力。

壁仞科技公司[10]致力于开发原创通用的计算生态，旨在建立一个高效率的软硬件平台，提供智能计算领域的整体性解决方案。其主要业务是云端通用计算 GPU，包括人工智能训练和推理、图形渲染和高性能计算等领域。壁仞科技公司发布的首款 GPGPU 芯片 BR100[11]创造了当时的全球算力记录，优异的性能表现标志着国内 GPGPU 芯片进入了“每秒千万亿次计算”的时代。

天数智芯半导体公司[12]致力于打造完全自主可控、国际一流的高性能云端计算芯片 GPGPU，旨在从芯片端解决算力问题。公司于 2018 年启动 GPGPU 芯片设计，是一家提供 GPGPU 高端芯片和超级算力的企业。其主要面向的场景是人工智能训练、高性能计算等，可以为自动驾驶、医疗、互联网、教育和金融等各行业提供服务。

寒武纪[13]自 2016 年成立以来，专注于人工智能芯片的设计研发和技术创新，公司提供智能加速卡、智能加速系统、智能边缘计算模组和软件开发平台等产品和技术，其中思元 370 系列智能加速卡已经成功应用在业界的人工智能训练等任务中。

海光深度计算处理器（DCU）[14]是基于 GPGPU 的架构而设计的科学计算和人工智能加速处理器，适合计算密集型和运算加速领域，能够为云计算、数据库、人工智能和大数据分析等领域提供强大的算力。

景嘉微公司[15]是国产 GPU 的主要厂商，具有完全自主的知识产权，也是国内第一家自主开发 GPU 并已经大规模商用的 GPU 企业。产品主要应用在图形处理领域。公司分别在 2018 年和 2019 年推出自主研发的 JM7200 和 JM7201，进一步推动了国产 GPU 的技术发展。

芯瞳半导体公司[16]旨在开发国产自主的 GPU 和人工智能芯片，致力于自主设计研发 GPU 芯片，为云端、终端客户提供可持续发展的国产 GPU 解决方案。公司的主要业务包括 GPU 芯片设计、异构计算平台方案等。

燧原科技公司[17]成立于 2018 年 3 月，公司专注于人工智能领域云端和边缘算力平台，产品由训练、推理、软件平台组成，旨在提供高性能的 GPGPU 芯片。

此外，登临科技公司[18]专注于高性能通用计算平台的芯片研发与技术创新，通过自主架构创新的 GPU+，解决了效率和通用性的双重难题。芯动科技公司[19]是国内一站式 IP 和芯片定制服务及 GPU 的领军企业，目前正在推出数据中心高性能服务器 GPU。兆芯公司[20]能够同时掌握 CPU、GPU、芯片组三大核心技术，主要致力于集成高性能显卡。中船重工七一六研究所[21]自主研发的国

产 GPU JARI G12 是 2018 年性能最强的国产通用 GPU，并支持 OpenCL 2.0，满足并行计算和云计算的使用需求。中微电科技公司[22]自研处理器拥有 ICubeCC 编译器和完全自主知识产权的多线程虚拟管线指令集架构(MVP ISA)两大核心技术，自主设计的“南风一号”是高性能桌面 GPU 芯片并已成功点亮，同时还在研发用于游戏和人工智能的 GPU。

随着人工智能领域的不断发展，尤其是大模型的兴起和国外高端 GPU 产品对国内市场的限售等影响，国产 GPU 的发展会受到越来越多的重视。

2 GPU 主流编程模型

由于异构计算的特殊性，为了更好地发挥 GPU 的性能优势，除了不断升级 GPU 硬件以外，还需要更好的软件设计生态以充分利用 GPU 的并行计算能力，因此需要有针对 CPU-GPU 异构计算的编程模型。

GPGPU 的编程模型需要充分发挥 GPU 的并行计算能力。与传统的图形渲染相比，编程模型需要根据 GPU 的特性，加速人工智能、科学计算、数据分析等通用计算任务。基于对 GPU 硬件资源更好和更便捷的应用，GPGPU 的软件体系应运而生。

目前最主流的 GPGPU 编程模型是 NVIDIA 公司推出的 CUDA[23]和跨平台的开放式编程框架 OpenCL[24]。它们是用于 GPGPU 编程的异构编程模型。这类编程模型提供了一种将 GPU 作为并行计算设备使用的方法。除了上述两个编程模型以外，还有 AMD 的 Radeon 开放计算平台(ROCm)[25]等，也支持在 GPU 上进行通用计算。这些框架提供了统一的编程接口和工具。本节主要介绍 CUDA 和 OpenCL 编程模型。

2.1 CUDA

CUDA 是 NVIDIA 公司在 2007 年推出的 GPGPU 的编程模型，目前 CUDA 仅应用在 NVIDIA 的 GPU 上。通过使用 CUDA，开发者可以充分利用 NVIDIA GPU 强大的并行计算能力，加速包括科学计算、数据分析、人工智能在内的各种计算密集型的场景任务。在 CUDA 的 CPU 和 GPU 的异构编程中[26]，host 指的是 CPU 及其存储器，device 指的是 GPU 及其存储器。CUDA 是 C/C++编程语言的一种拓展，CPU 端的代码就是 C/C++的编程语言，GPU 端的代码与 CPU 端几乎一样，其中需要特定的关键字来界定 host 端和 device 端的代码。为使编写的代码能够在 CPU-GPU 异构系统上执行，NVIDIA 基于低层虚拟机(LLVM)[27]设计了 CUDA 编译器 NVCC。

传统的 CUDA 编程操作序列如下[26]：

1)声明并分配 host 端内存和 device 端显存；

2)初始化 host 端的数据；

3)将数据从 host 端传输到 device 端；

4)执行 GPU 的核函数操作；

5)将 GPU 的计算结果传输到 host 端。

伴随着 CUDA 统一内存的推出，从开发者的角度来看，编程模型发生了很大的改变。此外，CUDA 为开发人员提供了丰富的编程库。如 CUDA 中实现基本线性代数子程序(BLAS)的 cuBLAS 库[28]提供的 API 支持向量和矩阵代数运算，快速傅里叶变换(FFT)的 CUDA 库 cuFFT[29]提供的 API 可以大大简化开发者的操作，另外还包括 cuRAND[30]和 NVIDIA 性能元件(NPP)[31]等。

2.2 OpenCL

OpenCL[24]是异构平台并行编程的开放标准框架，适用于云服务器、超级计算机、个人计算机(PC)、移动设备和嵌入式平台等各种加速器的跨平台并行编程。OpenCL 最初由 Apple 公司研发，目前由 Khronos 团队维护，从诞生至今，由 AMD、NVIDIA、IBM 和 Intel 等公司不断完善，成为目前除了 CUDA 以外最受欢迎的 GPGPU 编程语言。Khronos 团队在 2009 年发布了 OpenCL 1.0，并在 2020 年发布了 OpenCL 3.0。与 CUDA 只能在 NVIDIA 的 GPU 上运行不同的是，只要异构设备支持 OpenCL，基于 OpenCL 的代码就能在异构系统上运行，设备可以是 GPU、现场可编程门阵列(FPGA)和数字信号处理器(DSP)等。此外，OpenCL 核函数在运行时才进行编译，这给了 OpenCL 编程带来了更高的灵活性和可扩展性。

3 CUDA 统一内存

由于 CPU-GPU 异构架构的特殊性，相比于直接在 CPU 上运行的程序，在异构编程模型上进行数据管理变得更加复杂。本节主要介绍 CUDA 统一内存和 CUDA 中几个重要的内存管理技术。

3.1 需要统一内存的原因

1)GPU 显存容量有限且不易拓展。

由于 GPU 和 CPU 的架构设计和优势任务不同，GPU 处理数据需要更高的位宽和更快的传输速度，因此 GPU 使用的显存颗粒不同于内存颗粒，目前主流的单颗显存的位宽是 32 bit，通常只有 1 GB 和 2 GB 的容量大小。GPU 显存的容量也与 GPU

显存位宽有关，如 128 bit 位宽的 GPU 只能支持 4 个单颗显存。相比于 CPU，考虑到 GPU 功能的局限性，加之散热、功耗、整机物理空间等因素的影响，GPU 的体积有限。此外，独立 GPU 一般是厂商封装好的独立硬件，所以 GPU 的显存容量有限且不易拓展。

2）异构编程的特殊性。

在 CPU-GPU 异构编程的传统编程模式 CKC 操作流程[32-33]中，处理器只能对自己内存中的数据进行操作。应用程序开发人员需要在主机端和设备端分配空间，并手动在 GPU 本地和 CPU 系统之间显式地移动数据，并且还要保证其正确性和效率。

这种执行模型会带来两个问题：首先，数据的迁移和核函数执行串行化执行，总执行时间更长，使用复杂的异步执行来重叠数据迁移和核函数执行可以缓解这个问题；其次，随着 GPGPU 应用程序工作集大小的增加[34-36]，有限的容量将更容易成为性能瓶颈[36-38]，当处理的数据大于 GPU 显存的时候，程序员必须重新定义数据结构来传递数据。

近年来，应用程序数据集的规模不断增长，然而 GPU 的显存容量并未实现显著增加，应用程序数据集远远超过了 GPU 的显存容量。为了克服这个问题，开发者需要严格地手动重新组织计算，这增加了工作量，也增加了编程错误的风险。

为了应对以上的挑战，NVIDIA CUDA[39]和 AMD[40-41]公司引入了软件运行时，提供了 CPU-GPU 的统一虚拟内存，并提供了系统中任何处理器访问单个虚拟地址空间的编程模型。

3.2 CUDA 统一内存的基本概念

CUDA 统一内存[42]是 CUDA 6.0 中推出的一种新型的内存编程模型，旨在简化 GPU 编程模型中的内存管理、数据传输和超额订阅 GPU 显存。在典型的 PC 和集群节点中，CPU 和 GPU 的内存是分开进行管理的，并通过高速连接通道连接。在 CUDA 6.0 之前，CPU 和 GPU 之间共享的数据必须在两个内存中分配，并由程序在它们之间显式复制，这增加了编程的复杂性。

CUDA 统一内存创建了一个在 CPU 和 GPU 之间共享的托管内存池，CPU 和 GPU 可以使用单个指针访问分配的统一内存。在数据使用的过程中，系统会隐式地在主机（CPU 端）和设备（GPU 端）之间申请的统一内存中分配数据。这个过程是对用户透明的。

图 2 展示了 CUDA 统一内存的架构示意图，逻辑上主机内存和 GPU 设备显存在一个虚拟内存地址空间中，在物理上仍使用高速连接通道（如 PCIe 总线）连接。

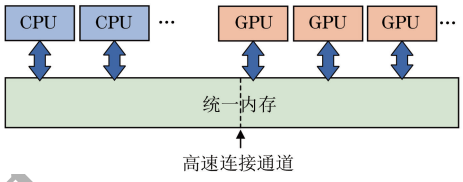


图 2 CUDA 统一内存

Fig. 2 Unified memory in the CUDA

3.3 CUDA 统一内存的发展过程

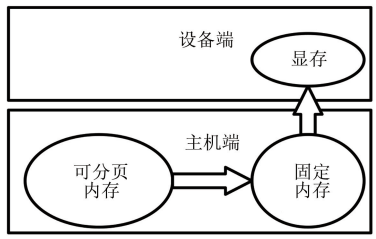
在推出统一内存之前，NVIDIA 推出了多个内存管理技术。

1）可分页内存和固定内存。

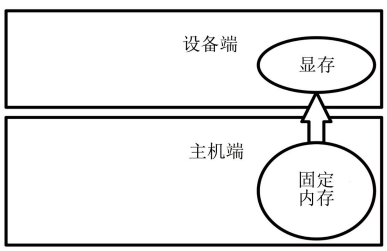
在默认情况下，主机端 CPU 的数据分配在可分页内存[2]中。GPU 无法直接从可分页主机内存中访问数据。因此，当 GPU 要使用 CPU 内存中数据的时候，CUDA 驱动程序必须首先分配一个固定或称为临时页锁定的主机内存，将需要用到的主机数据复制到这部分内存中，然后传输到 GPU 端的设备显存中[2]。

固定内存是 GPU 可直接访问的主存区域。在申请内存的时候也可以使用 cudaHostAlloc() 直接在主机内存中分配固定内存，避免在可分页主机内存和固定主机内存之间进行数据传输的开销。直接申请的固定内存使用 cudaFreeHost() 释放。使用主机固定内存和使用可分页内存都通过 CUDA 的内存拷贝函数[如 cudaMemcpy()]进行数据传输。

图 3 展示了主机端不同类型内存上的数据在 CPU 和 GPU 之间的传输过程。



(a)可分页内存中的数据传输



(b)固定内存中的数据传输

图 3 可分页内存和固定内存中的数据传输

Fig. 3 Data transfer in pageable memory and pinned memory

2)零拷贝内存。

零拷贝内存是 CUDA 2.2 中加入的一种内存管理技术[43],是一种特殊的内存映射,它允许 GPU 直接访问 CPU 主机内存,允许将主机端的内存地址映射到 GPU 设备的内存空间,不需要显式地将数据从主机内存复制到 GPU 的显存中。零拷贝内存可以理解为 GPU 共享 CPU 的内存。在集成 GPU 中,CPU 和 GPU 的物理内存是相同的,这避免了多余的内存拷贝。在目前主流的独立 GPU 中,零拷贝内存中的数据在使用前需要传输到 GPU,并且数据不会被缓存在 GPU 显存上,数据在被处理计算后,也不会驻留于物理显存,映射的固定内存只被读取或写入一次。零拷贝内存的缺点也很明显,即 GPU 每次使用数据都需要通过 CPU 和 GPU 之间的连接通道传输,而连接通道的带宽有限,在这个过程中 GPU 物理显存高带宽的优势没有被体现出来。

3)统一虚拟寻址(UVA)[44-45]。

UVA 是在 CUDA 4.0[46]中引入的一项技术。在没有引入这项技术之前,CPU 内存和 GPU 显存的地址空间是各自独立的,两者拥有独立的物理地址和逻辑地址,当需要数据交换时,数据传输需要地址转换。当使用 UVA 时,所有已安装的设备内存和主机内存共享一个虚拟地址空间,整个系统的内存空间被统一管理,无需指定该地址属于主存或显存,这简化了 CPU-GPU 之间的数据管理。当不使用 UVA 时,GPU 访问位于 CPU 端内存中数据的时候需要先获取本设备的指针,再进行访存操作。UVA 技术是统一内存的基础技术之一。

4)统一内存。

基于 UVA 技术,统一内存不再需要两套指针来管理 CPU 主存和 GPU 的设备显存,统一内存上的数据也不需要进行主存和显存的区分。数据在主存和显存之间由驱动程序来隐式地进行数据迁移。CUDA 统一内存的架构示意图如图 4 所示。统一内存可以从系统的任何处理器中进行访问。统一内存创建了一个在 CPU 和 GPU 之间共享的托管内存空间。在该空间中,所有处理器都使用单个指针访问统一内存。

5)CUDA 内存管理技术的特点和关系。

CUDA 异构系统的数据传输需要使用固定内存。相对于固定内存,可分页内存是操作系统中的内存管理机制,用于虚拟内存管理。在 CUDA 中,CPU 和 GPU 之间的数据拷贝是使用 DMA 实现

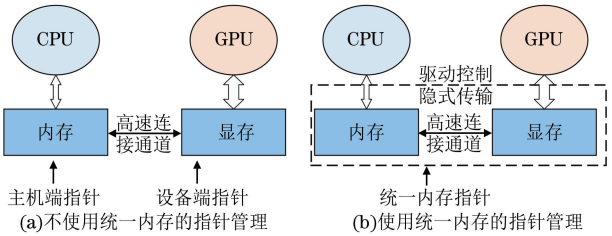


图 4 统一内存的指针示意图

Fig.4 Pointer illustration of unified memory

的,如果使用可分页内存,则会增加虚拟地址转换的开销。固定内存不会被操作系统虚拟内存分页管理,也不会被交换到磁盘上,这保证了这块内存始终驻留在物理内存上,不会被破坏或者被重新定位,对于其访问也不需要地址转换。因此,GPU 通过 DMA 能够直接对固定内存中的数据进行访问,进而加快数据访问的速度。固定内存是零拷贝内存和统一内存的基础,固定内存的引入便于 CPU 和 GPU 之间互传数据。

在需要申请的数据占用内存空间较大并且 GPU 核函数对这部分数据使用频次较低的情况下,可以申请零拷贝内存。在这种场景下数据不会占用 GPU 显存,当 GPU 核函数需要用到这些数据时,会通过 CPU 和 GPU 之间的连接通道直接从零拷贝内存中将数据取到 GPU 中计算处理,整个过程不会占用 GPU 显存。零拷贝内存通过 cudaHostAlloc()申请,最初主机和 GPU 设备管理两个指针,指向同一片内存地址。当引入 UVA 之后,通过 cudaHostAlloc()申请的零拷贝内存指针可以同时被 CPU 和 GPU 使用。

UVA 允许主机和设备代码在同一个虚拟地址空间中引用数据,且只需要一个指针。基于 UVA,统一内存[32]定义了一个托管内存空间,其中任何处理器都可以看到具有公共地址空间的单个一致内存映像。统一内存提供了一个“单指针数据”类型,其概念类似于零拷贝内存,但是零拷贝内存中 CPU 端的数据不会被复制到 GPU 的显存中,GPU 核函数会直接使用这部分数据,而统一内存会将数据传输到对应的主机或设备内存中。

4 统一内存的工作原理、优缺点和优化策略

统一内存实现了 GPU 显存的超额订阅,本节主要介绍统一内存的工作原理,使用统一内存的优缺点和性能提升优化策略。

4.1 统一内存的工作原理

统一内存隐式地将数据在 CPU 和 GPU 之间通过高速连接通道进行迁移,当 CPU 程序或 GPU

程序运行的代码需要访问数据且数据不在需要访问数据的处理器内存上时，CUDA 驱动就会负责将内存页面从统一内存的另一侧迁移到发起访问的处理器一侧，这个过程通过缺页中断的方式触发，然后将数据以页迁移的方式迁移到目标物理内存上。这个过程是被动迁移的，也称为按需迁移。NVIDIA 推出的 Pascal 架构的 GPU 是第一个通过其页迁移引擎为虚拟内存缺页中断和页迁移提供支持的 GPU 架构。

在统一内存编程模型中，在逻辑上 CPU 和 GPU 共用一个地址空间，但是在物理上，内存和显存是分开的。以一个 GPU 应用程序为例，当其访问的页面位于 GPU 端的显存上时，程序可以直接使用这部分数据；当这部分数据位于 CPU 端的内存上时，首先驱动程序会触发一个缺页中断，然后将这部分数据从 CPU 端的内存迁移到 GPU 端的显存中，之后 GPU 程序才可以使用这部分数据；当显存容量被占满时，并且 GPU 程序需要访问新的数据的时候，位于 GPU 中的数据会被驱逐出一部分，再将新的数据迁移到显存中。

图 5 展示了当发生缺页中断时驱动程序对页面处理的示意图。

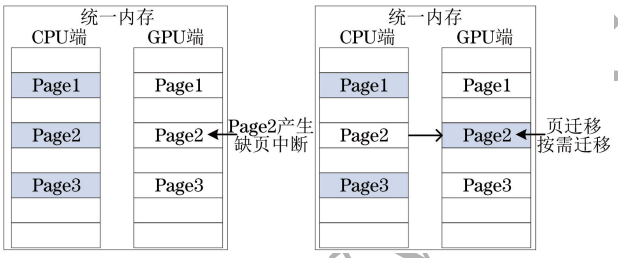


图 5 统一内存缺页中断处理

Fig. 5 Page fault handling in unified memory

4.2 统一内存的优缺点

统一内存的引入为 CUDA 编程提供了更多的可能性。其简化了 CPU 和 GPU 之间的数据传输，简化了 CPU-GPU 异构程序设计，使程序员能够专注程序设计本身，而不必在 CPU 和 GPU 之间的数据移动上进行手动管理，这样可以减少代码的复杂性和错误，提高编程效率。同时，统一内存也提高了 GPU 处理大数据集负载的能力。相比于 GPU 强大的算力，GPU 的显存容量有限，尤其是面对大规模数据的时候，不管是对于数据库大容量数据的操作，还是在深度学习使用 GPU 进行加速的训练中，GPU 的容量会成为限制应用程序的一个瓶颈。

统一内存有两大好处。首先，单一内存空间可以实现不同计算设备之间的无缝数据共享。程序员可以假设一个他们已经熟悉的虚拟内存系统。不同设备之间数据传输的操作从程序员转移到了系统软件和硬件上。其次，统一内存使 GPU 程序能够处理大于 GPU 显存容量的数据集，可以实现 GPU 显存的超额订阅，即使用比 GPU 显存容量更大的内存空间，对于此类工作负载，CPU 内存充当备份存储，数据以页粒度按需隐式迁移和驱逐。这从一定程度上缓解了 GPU 显存容量不足的问题。

统一内存虽然带来了好处，但是也存在一定的问题。首先是性能问题，CUDA 为用户提供了异构系统中逻辑上统一的内存地址，但是 CPU 和 GPU 物理内存空间是分开的，虽然不需要显式地移动数据，但还是需要驱动程序隐式地进行数据传输。这个过程（尤其是缺页中断的过程）会使程序处理时间变长，进而影响程序性能。此外，调入和调出 GPU 显存需要通过 CPU 和 GPU 之间的连接通道，如 PCIe[47]，互连和中断处理程序在 CPU 和 GPU 之间进行代价比较高的通信。特别是当应用程序处理的数据局部性差且需要频繁地换入换出的时候，性能影响会更大。

因此，针对 CUDA 统一内存的使用，需要结合预取和预驱逐策略来避免过多的缺页中断，进而提高程序性能。

4.3 数据预取和预驱逐

在统一内存的使用过程中会通过缺页中断来进行数据页面的迁移，也可以通过数据预取和预驱逐来降低缺页中断的发生，进而提高系统性能。

NVIDIA 在驱动层提供了基本的统一内存管理模块[48]来自动检测数据访问模式，能够在一定程度上优化页面的放置，减少缺页中断的发生。考虑到应用程序的复杂性和工作负载的多样化，当应用程序数据访问模式不明显时，开发人员可以检测数据访问模式，并对数据放置进行手动优化，进而根据应用负载特点对应用程序做针对性的数据预取和驱逐的优化。

针对缺页中断和数据访问模式的检测，NVIDIA 提供了性能分析工具 nvprof[49]，在支持计算能力 8.0 及更高的 GPU 设备上，可以使用 NVIDIA Nsight System[50]对 CPU 和 GPU 进行跟踪，也可以使用 NVIDIA Nsight Compute[51]对 GPU 内核进行性能分析及可视化。在此基础上，NVIDIA 提供了特定的 API 来实现数据预取和预驱逐。如在 CUDA 8.0[52]中引入的 cudaMemPrefetchAsync()可用于数据显式预取、cudaMemAdvise()可用于提供内存使用提示，通过

这些 API 可以指定特定内存区域的数据以冗余副本或显存驻留等方式降低缺页发生率。

数据预驱逐主要是发生在显存容量用尽的情况下,需要选择换出显存的页面,常见的利用最近最少使用算法对页面进行换出。此外,因为 NVIDIA 驱动程序会跟踪 GPU 端的可用空间,应用程序可以从 NVIDIA 驱动程序中获取可用空间信息,从 ID 关联表中获取下一个执行的核函数信息[53]。基于这些信息,可以对当前执行的核函数和将执行的下 N 个核函数预计不会访问的数据进行驱逐换出。

5 统一内存的领域应用

随着对 CUDA 统一内存研究不断积累和近几年统一内存在支持的功能上的不断完善,统一内存已经成功应用在多个领域。本节将梳理统一内存在大数据处理、深度学习和高性能计算等领域的研究工作。

在传统的 GPU 数据的内存管理中,GELADO 等[54]提出了一种简化和优化 GPU 数据管理并需要更改应用程序的编程模型。JABLIN 等[55]提出的一种 CPU-GPU 之间数据传输管理工具 CGCM 提供编译器和运行时支持,使 CUDA 程序中 GPU 显存管理自动化,此外还优化了迭代核函数调用的 CPU-GPU 通信。JABLIN 等[56]还提出了一个支持递归数据结构的自动 CPU-GPU 内存管理的运行时系统 DyManD。PAI 等[57]引入软件一致性机制来减少 CPU 和 GPU 之间的冗余数据传输。ALSABER 等[58]提出的 SemCache 利用类似缓存一致性的协议来优化 CPU 和 GPU 之间的通信。WANG 等[59]提出的 SuperNeurons 利用张量级的数据访问模式来提高深度神经网络训练的 GPU 显存利用率。这些在传统编程模式下的 GPU 数据管理策略也给了统一内存中的数据管理一些启发。

5.1 统一内存在大数据系统领域的应用

相比于 CPU,GPU 除了在算力方面的优势以外,更高的内存带宽也使其在加速大数据处理系统[60-61]中更有优势,如在 MapReduce[62]的基础上,HE 等[63]设计并实现了使用 GPU 进行加速计算的 Mars 框架。相较于大数据系统处理的数据规模,GPU 有限的显存容量容易成为系统的瓶颈,CUDA 统一内存的推出为大数据处理系统的设计和优化提供了更多的方案。

统一内存的引入拓展了 GPU 的显存。驱动主导的按需迁移的特性减少了数据处理系统中 CPU 和 GPU 之间的数据传输的总量,但是 NVIDIA 驱动程序隐式处理的缺页中断带来了性能的损失,如何减少缺页中断成为优化大数据处理的一个新的方向。

5.1.1 页面预取、预驱逐和复用优化

统一内存超额订阅造成的性能开销可以通过程序员的编程工作来减少[42,64-67],例如进行手动预取和数据驱逐,或者将预取请求和驱逐请求重叠。但是,通过手动操作需要程序员明确区分读取和写入数据,并且了解数千个并发线程中的数据局部性,从而将页面显式地映射到 CPU 内存或 GPU 显存。这对手动数据管理提出了很高的要求。

CUDA 8.0[52]引入了专有的基于树的硬件预取器,与其他预取器和按需分页相比,这种预取器可以在一次传输中迁移虚拟地址空间中连续多个 64 KB 大小的块。该预取器的作用是提前对位于当前缺页中断页面 2 MB 范围内的所有页面进行预取操作。硬件预取器是一种硬件机制,其根据访问模式预测数据的可能需求,并在数据被请求之前提前对其进行预取,从而减少缺页中断。基于树的硬件预取器根据数据访问的局部性进行预测和预取,预取连续 64 KB 的块可以提高内存传输效率和带宽利用率。

当 GPU 显存满载且工作负载仍需页迁移以进行接下来的工作的时候,需要从 GPU 显存进行页面驱逐。默认使用的最近最少使用(LRU)策略的 4 KB 页面的驱逐方法并不能提高性能,因为在 LRU 列表中的页面在虚拟地址中可能间隔较远,这破坏了局部感知预取器的语义,在解决缺页中断的时候开销会更大,且激进的预取可能从显存中换出频繁使用的页面,带来更大的性能开销。影响系统性能的关键是缺页中断的发生率,其中针对 GPU 程序负载提前预取数据以提高二级缓存命中率已经在 OWL 等工作中被广泛研究[68-70]。ZHENG 等[38]在统一内存背景下提出了用户控制的预取器,这项工作表明预取更大的内存块可以提高 PCIe 的利用率并减少传输延迟。基于此,GANGULY 等[71-73]结合硬件预取器进行了多项研究。文献[71]是第一篇分析硬件预取器语义的工作,然后进一步分析了硬件预取器和 CPU-GPU 统一内存中页面驱逐策略之间的相互作用。基于硬件预取器,该文中提出了两种不需要程序员干预、局部感知的预驱逐策略,分别是顺序局部和基于树的邻域驱逐方案,其中基于树的邻域驱逐方案是自适应的,其驱逐大小在 4 KB~2 MB 之间。通过将重新设计的预驱逐策略和硬件预取器相结合,程序负载可提供一个数量级的性能加速。

在社交网络分析[74]、网络搜索[75]、流行病学中的图形应用[76]和数据挖掘算法等场景中需要对大型不规则的树和图等数据结构进行操作,并且高度依赖输入。这些负载情况下的数据内存访问不规则,数据空间局部性很低。在使用 GPU 进行操作的时候,需要频繁地换入换出,增加了性能开销。预取器[38,67]根据访问数据的时空局部性提前预取数据。在对常规数据及密集、顺序的内存访问的场景下,预取器的使用减少了缺页中断的数量,能够更好地利用 PCIe 的带宽,进而降低负载开销,提升性能。但是在对访问不规则的数据密集型的应用进行传统的数据预取时,使用 LRU 替换大量的页面而不区分冷热数据时,性能下降会更加严重。

针对不规则应用程序中稀疏且很少访问的数据,CUDA 8.0[52]和 OpenCL[77-78]尽量使用主机端固定的“零拷贝”内存缓冲区。这样不使用统一内存的超额订阅,避免迁移数据到 GPU,稀疏访问也可以受益于低延迟的直接访问。GANGULY 等[72]对流行的 GPU 工作负载的内存访问模式进行了广泛的分析,将数据访问类型分为常规数据和不规则数据的访问。通过对不规则应用的工作集进一步的研究,将数据分为冷数据结构和热数据结构,其中,冷数据结构的分配访问很少且稀疏,热数据结构具有密集的顺序访问。在此基础上,文献[72]提出了一种针对不规则通用应用程序的动态页面放置策略,利用 Pascal 或 Volta 的 GPU 架构[64]硬件中的访问计数器[67]来识别稀疏和密集内存访问并区分热分配数据和冷分配数据,根据内存访问模式和访问频率,使用启发式方法自适应地在页迁移和零拷贝内存之间进行切换。文献[72]所提出的方案实现了将热数据页固定在设备显存中,同时远程访问使用零拷贝内存的冷数据的方法,因此根据数据使用的特性,在低延迟远程访问和高带宽设备本地访问之间取得了平衡,减少了内存页抖动,最终提高了不规则的数据密集型应用的性能。

在使用统一内存进行超额订阅的场景下,性能开销的一部分来自缓慢的 CPU-GPU 互联上内存页面的抖动。GANGULY 等[73]提出了一种应用程序感知的自适应框架,该框架可以拓展软件运行时管理,利用缺页中断和页迁移信息来检测 CPU-GPU 互连中的底层模式。该框架根据负载应用的特点,自适应地调整最佳的内存管理策略来降低因使用超额订阅而引起的内存页抖动,进而提升应用程序的性能。

针对超额订阅带来的性能开销,LI 等[79]提出了一种称为驱逐-节流-压缩(ETC)的内存管理框架,该框架能重叠 GPU 驱逐页面的延迟、降低抖动成本并增加有效的内存容量,从而提高 GPU 应用程序的性能。文献[79]指出:当内存超额订阅时,常规的应用程序和使用不规则数据集的应用程序会表现出不同类型的行为,常规应用程序受页面驱逐延迟的影响较大,而不规则的应用程序则容易出现内存页抖动。ETC 将应用程序分为常规和不规则的应用程序,并使用下面步骤优化应用程序:首先,通过主动驱逐腾出 GPU 显存空间,进而隐藏驱逐延迟;其次,通过内存感知节流来改善抖动成本,当缺页中断频率变高时,该节流会动态降低 GPU 的并行度;最后,压缩容量,可在不增加物理容量的情况下启用更大的工作集。

但是,在处理过程中,对于很多大规模、不规则的应用,ETC 框架的效果不是很好。主动驱逐在很大程度上依赖于预测正确的时机,以尽量避免过早或过晚的驱逐,但是不规则的应用程序在短时间内访问大量页面,很难做到预测正确。因此,主动驱逐对不规则的数据应用程序来说是无效的,不正确的驱逐甚至会损害系统性能。

针对在 GPU 显存容量不足的情况下的页面驱逐策略,广泛使用的 LRU 策略以及高级替换策略 CLOCK-Pro[80]和 RRIP[81]在出现页面抖动时的访问效率较低。分层页面驱逐(HPE)[82]采用了分层页面驱逐的策略,这是一种针对统一内存的新的驱逐替换策略。在典型的 GPU 应用程序[83-85]的访问模式中,具有连续地址的虚拟页面一般具有良好的空间局部性,且虚拟页面在缺页中断后通常在短时间内处于热状态,即页面在迁移到 GPU 显存后不久会被重新引用,驱逐新迁移的页面会导致抖动,文献[82]将这种情况称为“即时抖动”。HPE 通过维护一个软件管理的页面集链,并根据页面集的最近使用时间和频率来选择驱逐的候选对象。HPE 旨在解决 LRU 在出现页面抖动时数据访问的低效率问题,同时保留 LRU 的优势,通过对不同负载的应用选择合适的驱逐策略,并在多种驱逐策略间动态切换,进而提高应用的性能。另外,作者在 GPU 端添加了一个非常小的组关联缓存来记录页面遍历命中信息,此信息会定期传输到 GPU 驱动程序以更新页面集链。

在 HPE 的基础上,协调页面预取和驱逐(CPPE)[86]针对页面预取和预驱逐相结合时效率低下的问题进行研究。为了允许通过页面预取进行有效的页面驱逐,CPPE 使用 HPE 的修改版本

(MHPE)来处理内存页抖动时的数据访问模式。为了通过页面驱逐来促进页面预取,CPPE 引入了访问模式感知预取器,该预取器根据 MHPE 选择的驱逐候选中的访问模式来预取页面。CPPE 以细粒度的方式合并了修改的页面驱逐策略、MHPE 和访问模式感知预取器。CPPE 是使用细粒度机制协调页面预取和驱逐来管理统一内存的超额订阅策略。

KIM 等[87]对现代 GPU 中使用的缺页中断处理机制中出现的主要低效问题进行了全面分析。为了分摊缺页中断的成本,GPU 运行时会同时处理大量的 GPU 缺页中断,这种缺页中断的批量处理引入了大规模的序列化,大大降低了 GPU 的执行吞吐量。在此基础上,该文中提出了一种 GPU 运行时软件和硬件解决方案,通过增加处理的缺页中断的数量,既增加批量大小,又增加并行量来分摊 GPU 运行时缺页中断的处理时间。此外,通过使用 CPU-GPU 的数据页迁移和数据驱逐重叠执行的方式降低开销。该工作支持的场景是统一虚拟内存模型中图计算工作等大规模不规则应用程序的执行。在处理细节上,通过使用线程超额(TO)订阅来增加单次处理的缺页中断数量,进而分摊错误的处理时间。线程超额订阅是一种类似于 CPU 线程块上下文切换的技术。另外,页迁移占总执行时间的比例较大,通过向每个 GPU 核心分配更多的线程块来降低开销。与先进的页面预取机制[38]和 ETC[79]机制相比,使用该方法后的性能有明显提升。

在使用 GPU 的工作负载中,当 GPU 引用 CPU 端的内存时,页面的重复替换会降低性能开销。PARK 等[88]分析发现了循环页抖动的行为,它是一种重复替换相同页面的现象。为了加速出现循环页抖动情况的应用程序,作者在驱动程序中设计并实现了一组新函数,用于检测循环页抖动并通过在 GPU 页面中保留页面来减轻抖动,这样可以避免循环使用的页面被驱逐出 GPU 显存,进而实现页的复用。

5.1.2 从编译角度的优化

除了从页面的预取、预驱逐和热数据的页面复用的角度来降低缺页中断和内存页抖动来优化统一内存中的数据管理以外,也有相关工作从编译[89-90]的角度来对统一内存的使用进行优化。LI 等[89]通过编译器分析以及运行时优化统一内存管理。该文将统一内存驱动程序/操作系统发起的按需数据传输称为隐式数据传输,将批量数据移动调用产生的数据传输称为显式数据传输,无论是由应用程序开发人员还是编译器等产生的数据移动都属于显式数据迁移。该文针对大型应用程序数据集的需求,提出一组混合隐式/显式数据传输方案,该方案使用基于目标重用距离的方法能够利用细粒度的数据局部性,总体上具有较好的性能和数据移动效率,同时采用了编译器运行时协作的方法,在 LLVM 基础架构中实现了可靠、高效的 GPU 统一内存数据管理。在系统运行过程中不需要开发人员干涉,其中编译器负责分析高级数据访问行为,运行时负责处理动态执行状态和编译器提供的静态信息以此来优化数据映射和移动,自适应地选择隐式和显式的数据传输,并减少数据内存页抖动。该方案与更多的依赖运行时或操作系统进行内存管理建议的方法[57-58, 91-92]不同,其大部分分析是在编译时执行的,执行开销会明显降低。

针对大型的不规则内存访问的应用程序可能出现的内存页面抖动问题,CHANG 等[90]实现了 DynaMap,从 LLVM 的编译路径的角度来进行优化。DynaMap 通过识别、预测和利用页面的空间局部性来进行动态页迁移,将页面在被驱逐之前的空间利用率作为一个指标,结合编译路径在运行系统中使用日志来估计利用率,并基于缺页中断率设置自适应阈值来确定统一内存中的页面是否迁移。

5.1.3 数据传输优化

当使用 CUDA 统一内存超额订阅 GPU 显存时,CPU 端可分页内存和 CPU 端统一内存之间的数据传输也会有很大的开销[93]。WANG 等[94]设计了 D-Cubicle,这是一个运行时模块,通过自适应、多线程的方式加速 CPU 端管理的可分页内存和统一内存之间的数据传输。该文结合大规模分析型查询中的数据特征,根据处理的数据块的大小对数据进行分组,对数据组进行传输优化。在此过程中,首先记录数据传输信息,然后自适应调节参与数据传输的线程数,在不同的资源竞争情况下动态维持较高的传输速度,进而实现了系统的性能优化。

5.2 统一内存在深度学习领域的应用

近年来,人工智能尤其是深度学习领域得到了飞速的发展,随着模型的不断变大,数据集规模不断增长,对算力也提出了更高的要求,本节将介绍统一内存在深度学习领域的应用。

在深度神经网络(DNN)变得更深和更广的背景下,如何高效地进行 CPU-GPU 异构系统的数据管理变得更为重要。现有的 DNN 数据集规模已经

远远超过了目前最先进的 GPU 的显存容量。在这种情况下,为了使用 GPU 加速 DNN 的训练,研究人员从数据压缩[95-98]、数据重计算[59, 99-100]和 CPU 和 GPU 之间高效的内存交换[36, 53, 59, 95, 101-107]等角度来优化 DNN 的训练。部分工作[53, 102]对内存交换的研究将统一内存作为研究重点。

DeepUM[53]设计了一个框架,使用高效内存交换的方式来优化深度学习的训练,允许使用 CUDA 统一内存为 DNN 超额订阅 GPU 的显存,将 CPU 主存作为后备存储。该文中作者针对 NVIDIA 设备驱动层和 CUDA 的 Runtime 层做了一层封装,使用 CUDA 统一内存优化了在 DNN 训练过程中显存容量不足的问题。针对统一内存带来的性能损失和 DNN 负载的特点,作者做了预取和预驱逐的优化,在 DeepUM 框架下,在 DNN 训练过程中,在 GPU 中的训练数据能够预取到设备端,减少了缺页中断的发生。同时,在 GPU 显存容量不足的情况下,预驱逐的方式能将 GPU 端的数据率先驱逐出 GPU,这样既提高了 PCIe 带宽的利用率,又为接下来要用到的数据提前腾出了 GPU 的显存空间,减少了缺页中断发生时数据驱逐的时间和 PCIe 的占用,进而减少缺页中断的发生,提高训练性能。

DNN 训练主要分为两个阶段:前向传播和后向传播,两个过程都涉及数据移动。如果没有高效的内存管理方案,大型 DNN 在 GPU 训练中容易受到显存容量的限制。AWAN 等[102]设计并实现了一种新颖的核外 DNN 训练框架 OC-DNN,它利用 CUDA 统一内存和 Pascal 与 Volta GPU[64]中新的硬件机制,以及统一内存的通信原语和预取[cudaMemPrefetchAsync()]与内存建议[cudaMemAdvise()][108]等统一内存的新的 API 接口,进行核外 DNN 训练。具体是通过在每个 DNN 操作前手动地添加页面预取指令来减少 GPU 程序的缺页中断,为超大型 DNN 实现高效的核外训练。

5.3 统一内存在高性能计算领域的应用

异构计算中的加速器(GPU、FPGA 等)在高性能计算中变得越来越重要,近些年随着数据集越来越大,应用程序数据集的扩展已经超出了加速器的容量。DRAGON 框架[91]从利用 CUDA 统一内存来拓展 GPU 显存的角度来利用 GPU 处理大的数据集。DRAGON 框架[91]的提出解决了 GPU 应用程序在比主机内存更大的数据集上操作时的核外数据访问问题。使用 CUDA 统一内存和非易失性内存(NVM)作为显存的后备存储器的目标是使一般程序能够使用 NVM 来拓展 GPU 显存和计算机内存以处理大规模数据。它需要修改用户代码和修改驱动设备。使用统一内存和 NVM、利用缺页中断的功能允许将 NVM 存储直接映射到 GPU 作为其可寻址空间,从而为现有程序提供更大的内存支持。将可寻址全局内存空间透明地扩展到 NVM 改变了传统的 GPGPU 编程范例来应对大数据集工作负载,使更多类别的 GPGPU 应用程序能够透明地对驻留在 NVM 中的 TB 级别的数据集进行操作,同时保留数据缓冲区的完整性。DRAGON 利用 NVIDIA GPU 的缺页中断机制扩展 CUDA 统一内存的功能,通过将 GPU 设备代码可寻址的内存空间透明地直接映射到 NVM 设备上,大大拓展了设备显存以及主机内存的容量,进而使高性能计算系统能够处理 TB 级别的数据。

5.4 统一内存在不同领域的应用差异

在不同应用领域,针对不同负载特征可以对统一内存进行针对性使用和优化。

在大数据系统领域,主要应对的是数据密集型任务[72, 94, 109],数据的 I/O 更容易成为性能瓶颈。在将数据传输到 GPU 计算之前,可以使用 CPU 对数据进行预处理,通过数据压缩或 CPU 预计算等预处理操作,减少传输的数据量,进而降低数据 I/O 的负载。例如,针对 GPU 设计的数据库,主存端的数据预处理和传输也需要进行针对性的优化,在 D-Cubicle[94]中将 CPU 的资源用于数据的处理,并加速主存和统一内存之间的数据传输,进而提升系统整体的性能。

在深度学习领域和高性能计算中,面临更多的是计算密集型任务[53],需要借助 GPU 的高并行能力和高显存带宽进行更快的计算。在使用统一内存的过程中,需要向 GPU 显存中迁移更多的待处理数据[102]。此外,深度学习中迭代次数多,数据重复利用的规律较明显,因此可以记录数据的利用情况,指导后面的训练做更好的数据预取。深度学习大多是在深度学习框架上运行,在对数据进行检测和预取处理的时候也需要结合数据类型的特点(如 Tensor 数据类型)进行优化。在高性能计算中[91],统一内存的使用和优化应更注重数据访问效率和对多设备协同计算的支持能力。

6 统一内存的未来研究趋势

随着大数据时代的来临,不管是人工智能领域,还是 GPU 数据库领域,基于 GPU 的异构系统处理的数据量将会越来越大,而独立 GPU 显存容量低的特性在短时间内也很难改变,因此 CUDA 推出的

统一内存依然有很大的研究空间。自 CUDA 统一内存推出以来,后续的 CUDA 版本不断对统一内存进行改进,以便其更好地适应计算需求。同时,NVIDIA GPU 架构的不断更新迭代,也在更好地支持统一内存的功能。

统一内存为了提供更好的可编程性和拓展 GPU 的显存容量,增加了很多的隐式数据处理,包括缺页中断、数据迁移等,通过 NVIDIA 驱动程序对其进行处理。2022 年 5 月,NVIDIA 开源了其内核驱动程序,这为优化统一内存使用过程中的性能提供了更多的可操作空间。结合统一内存的原理,研究人员可以从减少缺页中断、主动数据迁移、数据预驱逐、热数据的复用等方面进行优化。此外,针对负载的特性,也可以从虚拟地址转换,新硬件存储等角度来优化统一内存负载。在深度学习中,在训练过程中会出现大量控制逻辑简单、计算可并行度高的迭代过程,针对训练模型和数据集的特性,做好数据预取和预驱逐能够有效减少训练过程中缺页中断的发生,能够有针对性地对负载进行优化。

随着各行业对 GPGPU 的依赖越来越高,以及自动驾驶等新领域的崛起,结合 GPU 强大算力、高带宽访存和低容量显存之间相矛盾的特点,在未来对 GPGPU 编程的研究中如何让 GPU 处理更大规模的数据量的研究依然有很大的发展空间。

7 结束语

本文主要对 CUDA 的统一内存做了研究总结。对 GPGPU 的发展和 GPU 的硬件发展以及国内外 GPU 厂商等方面做了介绍,分析了 CUDA 统一内存的发展历程、原理和目前在热门领域的研究现状。

目前,CUDA 的统一内存已经成功应用于人工智能训练、大数据处理系统和高性能计算等领域,自统一内存技术推出以来,随着 CUDA 对统一内存更多新功能的支持以及 NVIDIA GPU 架构的不断更新,统一内存更加高效和易用。除了带来更便捷的编程性和超额订阅的优势以外,如何减少使用统一内存所带来的性能损失是优化统一内存的关键,并且结合负载特征,减少缺页中断所带来的性能损失,将能更好地利用 GPGPU 的资源,提升系统的性能。

参考文献

[1] 中关村云计算产业联盟. 2022 年中国云计算生态蓝皮书[EB/OL]. [2023-10-16]. https://baike.baidu.com/reference/61785033/533aYdO6cr3_z3kATKaPnqr0N33HMN-kvuXXUbpzzqIP0XOpSo_sUIEz6NYwsPVmHQ_e_pttbZkGyeGuB0pC7f4WdOg8QrUrnXX4UTfKzb_wuI9zl4MV-tEW.
Zhongguancun Cloud Computing Industry Alliance. Cloud computing ecosystem report[EB/OL]. [2023-10-16]. https://baike.baidu.com/reference/61785033/533aYdO6cr3_z3kATKaPnqr0N33HMN-kvuXXUbpzzqIP0XOpSo_sUIEz6NYwsPVmHQ_e_pttbZkGyeGuB0pC7f4WdOg8QrUrnXX4UTfKzb_wuI9zl4MV-tEW. (in Chinese)

[2] NVIDIA Corporation. How to optimize data transfers in CUDA C/C++[EB/OL]. [2023-10-16]. https://developer.nvidia.com/blog/how-optimize-data-transfers-cuda-cc/.

[3] NVIDIA Corporation. NVIDIA Tesla P100[EB/OL]. [2023-10-16]. https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf.

[4] NVIDIA Corporation. NVIDIA TESLA V100 GPU architecture[EB/OL]. [2023-10-16]. https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf.

[5] ARM Corporation. ARM Mali GPU OpenCL developer guide[EB/OL]. [2023-10-16]. https://developer.arm.com/documentation/101574/0502/OpenCL-2-0/Shared-virtual-memory.

[6] ASHBAUGH B. Cl Intel unified shared memory[EB/OL]. [2023-10-16]. https://registry.khronos.org/OpenCL/extensions/intel/cl_intel_unified_shared_memory.html.

[7] Jon Peddie Research. Q1'22 saw a decline in GPU and PC shipments quarter-to-quarter[EB/OL]. [2023-10-16]. https://www.jonpeddie.com/news/q122-saw-a-decline-in-gpu-and-pc-shipments-quarter-to-quarter/l.

[8] METAX-TECH Corporation. METAX-TECH[EB/OL]. [2023-10-16]. https://www.metax-tech.com/.

[9] MTHREADS Corporation. MTHREADS[EB/OL]. [2023-10-16]. https://www.mthreads.com/.

[10] BIREN Technology Corporation. BIRENTECH[EB/OL]. [2023-10-16]. https://www.birentech.com/.

[11] BIREN Technology Corporation. BR100[EB/OL]. [2023-10-16]. https://www.birentech.com/News_details/16125806.html.

[12] ILUVATAR Corporation. ILUVATAR[EB/OL]. [2023-10-16]. https://www.iluvatar.com/.

[13] CAMBRICON Corporation. CAMBRICON[EB/OL]. [2023-10-16]. https://www.cambricon.com/.

[14] HYGON Corporation. HYGON[EB/OL]. [2023-10-16]. https://www.hygon.cn/product/accelerator.

[15] JINGJIA MICRO Corporation. JINGJIA MICRO[EB/OL]. [2023-10-16]. https://www.jingjiamicro.com/.

[16] SIETIUM Corporation. SIETIUM[EB/OL]. [2023-10-16]. https://www.sietium.com/.

[17] ENFLAME-TECH Corporation. ENFLAME-TECH[EB/OL]. [2023-10-16]. https://www.enflame-tech.com/.

[18] DENGLINAI Corporation. DENGLINAI[EB/OL]. [2023-10-16]. https://denglinai.com/.

[19] INNOSILICON Corporation. INNOSILICON[EB/OL]. [2023-10-16]. https://www.innosilicon.cn/.

[20] ZHAOXIN Corporation. ZHAOXIN[EB/OL]. [2023-10-16]. https://www.zhaoxin.com/.

[21] Shanghai Marine Diesel Engine Research Institute. CSIC-711[EB/OL]. [2023-10-16]. http://www.csic-711.com/ch/main.asp.

[22] ICUBECORP Corporation. ICUBECORP[EB/OL]. [2023-10-16]. http://www.icubecorp.cn/.

[23] NVIDIA Corporation. CUDA toolkit, develop, optimize and deploy GPU-accelerated apps[EB/OL]. [2023-10-16]. https://developer.nvidia.com/cuda-toolkit.

[24] OpenCL Corporation. OPEN standard for parallel programming of heterogeneous systems[EB/OL]. [2023-10-16]. https://www.khronos.org/opencl/.

[25] AMD Corporation. AMD ROCm™ documentation[EB/OL]. [2023-10-16]. https://rocm.docs.amd.com/en/latest/.

[26] NVIDIA Corporation. An easy introduction to CUDA C and C++[EB/OL]. [2023-10-16]. https://developer.nvidia.

com/blog/easy-introduction-cuda-c-and-c/.
[27] LLVM. The LLVM compiler infrastructure[EB/OL]. [2023-10-16]. https://llvm.org/.
[28] NVIDIA Corporation. cuBLAS[EB/OL]. [2023-10-16]. https://docs.nvidia.com/cuda/cublas/.
[29] NVIDIA Corporation. cuFFT API reference[EB/OL]. [2023-10-16]. https://docs.nvidia.com/cuda/cufft/index.html.
[30] NVIDIA Corporation. cuRAND[EB/OL]. [2023-10-16]. https://docs.nvidia.com/cuda/curand/index.html.
[31] NVIDIA Corporation. NVIDIA 2D image and signal Performance Primitives (NPP)[EB/OL]. [2023-10-16]. https://docs.nvidia.com/cuda/npp/index.html.
[32] NVIDIA Corporation. NVIDIA CUDA-C-programming[EB/OL]. [2023-10-16]. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html.
[33] LI Z F, PENG B C, WENG C L. XeFlow: streamlining inter-processor pipeline execution for the discrete CPU-GPU platform[J]. IEEE Transactions on Computers, 2020, 69(6): 819-831.
[34] KWON Y, RHU M. A case for memory-centric HPC system architecture for training deep neural networks[J]. IEEE Computer Architecture Letters, 2018, 17(2): 134-138.
[35] MENG C, SUN M, YANG J, et al. Training deeper models by GPU memory optimization on TensorFlow[EB/OL]. [2023-10-16]. https://www.semanticscholar.org/paper/Training-Deeper-Models-by-GPU-Memory-Optimization-Meng-Sun/497663d343870304b5ed1a2ebb997aaf09c4b529#:~:text=In%20this%20paper,%20we%20propose%20a%20general%20dataflow-graph%20based%20GPU.
[36] RHU M, GIMELSHEIN N, CLEMONS J, et al. vDNN: virtualized deep neural networks for scalable, memory-efficient neural network design[C]//Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). Washington D. C., USA: IEEE Press, 2016: 1-13.
[37] RHU M, O'CONNOR M, CHATTERJEE N, et al. Compressing DMA engine: leveraging activation sparsity for training deep neural networks[C]//Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA). Washington D. C., USA: IEEE Press, 2018: 78-91.
[38] ZHENG T H, NELLANS D, ZULFIQAR A, et al. Towards high performance paged memory for GPUs[C]//Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA). Washington D. C., USA: IEEE Press, 2016: 345-357.
[39] NVIDIA Corporation. NVIDIA pascal architecture[EB/OL]. [2023-10-16]. https://www.nvidia.com/en-us/data-center/pascal-gpu-architecture/.
[40] AGARWAL N, NELLANS D, STEPHENSON M, et al. Page placement strategies for GPUs within heterogeneous memory systems[C]//Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM Press, 2015: 607-618.
[41] AMD Corporation. AMD Graphics Cores Next (GCN) architecture[EB/OL]. [2023-10-16]. https://www.techpowerup.com/gpu-specs/docs/amd-gcn1-architecture.pdf.
[42] HARRIS M. Unified memory in CUDA 6.0[EB/OL]. [2023-10-16]. https://developer.nvidia.com/blog/unified-memory-in-cuda-6/.
[43] NVIDIA Corporation. CUDA C++ best practices guide, zero copy[EB/OL]. [2023-10-16]. https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#zero-copy.
[44] NVIDIA Corporation. Peer-to-peer unified virtual addressing[EB/OL]. [2023-10-16]. https://developer.download.nvidia.com/CUDA/training/cuda_webinars_GPUDirect_uva.pdf.
[45] NVIDIA Corporation. CUDA C++ best practices guide, unified virtual addressing[EB/OL]. [2023-10-16]. https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html#unified-virtual-addressing.
[46] NVIDIA Corporation. CUDA toolkit 4.0[EB/OL]. [2023-10-16]. https://developer.nvidia.com/cuda-toolkit-40.
[47] SPECIFICATION B. PCI express® base specification revision 3.0[EB/OL]. [2023-10-16]. https://pcisig.com/pci-express-base-specification-revision-30#:~:text=This%20specification%20describes%20the%20PCI%20Express%C2%AE%20architecture.
[48] NVIDIA Corporation. NVIDIA CUDA-C-programming-performance tuning[EB/OL]. [2023-10-16]. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#performance-tuning.
[49] NVIDIA Corporation. Profiler user's guide[EB/OL]. [2023-10-16]. https://docs.nvidia.com/cuda/profiler-users-guide/index.html.
[50] NVIDIA Corporation. NVIDIA Nsight Systems[EB/OL]. [2023-10-16]. https://developer.nvidia.com/nsight-systems.
[51] NVIDIA Corporation. NVIDIA Nsight Compute[EB/OL]. [2023-10-16]. https://developer.nvidia.com/nsight-compute.
[52] NVIDIA Corporation. CUDA runtime API[EB/OL]. [2023-10-16]. https://docs.nvidia.com/cuda/cuda-runtime-api/.
[53] JUNG J, KIM J, LEE J. DeepUM: tensor migration and prefetching in unified memory[C]//Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM Press, 2023: 207-221.
[54] GELADO I, STONE J E, CABEZAS J, et al. An asymmetric distributed shared memory model for heterogeneous parallel systems[C]//Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM Press, 2010: 347-358.
[55] JABLIN T B, PRABHU P, JABLIN J A, et al. Automatic CPU-GPU communication management and optimization[C]//Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, USA: ACM Press, 2011: 142-151.
[56] JABLIN T B, JABLIN J A, PRABHU P, et al. Dynamically managed data for CPU-GPU architectures[C]//Proceedings of the 10th International Symposium on Code Generation and Optimization. New York, USA: ACM Press, 2012: 165-174.
[57] PAI S, GOVINDARAJAN R, THAZHUTHAVEETIL M J. Fast and efficient automatic memory management for GPUs using compiler-assisted runtime coherence scheme[C]//Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques (PACT). Washington D. C., USA: IEEE Press, 2012: 33-42.
[58] ALSABER N, KULKARNI M. SemCache: semantics-aware caching for efficient GPU offloading[C]//Proceedings of the 27th International ACM Conference on Supercomputing. New York, USA: ACM Press, 2013: 421-432.
[59] WANG L N, YE J M, ZHAO Y Y, et al. SuperNeurons: dynamic GPU memory management for training deep neural networks[C]//Proceedings of the 23rd ACM SIGPLAN

Symposium on Principles and Practice of Parallel Programming. New York, USA: ACM Press, 2018: 41-53.
[60] 裴威, 李战怀, 潘巍. GPU 数据库核心技术综述[J]. 软件学报, 2021, 32(3): 859-885.
PEI W, LI Z H, PAN W. Survey of key technologies in GPU database system[J]. Journal of Software, 2021, 32(3): 859-885. (in Chinese)
[61] 李志方. 异构体系结构上的数据处理加速[D]. 上海: 华东师范大学, 2021.
LI Z F. Accelerating data processing on the heterogeneous architecture[D]. Shanghai: East China Normal University, 2021. (in Chinese)
[62] DEAN J, GHEMAWAT S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.
[63] HE B S, FANG W B, LUO Q, et al. Mars: a MapReduce framework on graphics processors[C]//Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT). Washington D. C., USA: IEEE Press, 2008: 260-269.
[64] SAKHARNYKH N. Unified memory on pascal and volta[EB/OL]. [2023-10-16]. https://on-demand.gputechconf.com/gtc/2017/presentation/s7285-nikolay-sakharnykh-unified-memory-on-pascal-and-volta.pdf.
[65] SAKHARNYKH N. Beyond GPU memory limits with unified memory on pascal[EB/OL]. [2023-10-16]. https://developer.nvidia.com/blog/beyond-gpu-memory-limits-unified-memory-pascal/.
[66] NVIDIA Corporation. Maximizing unified memory performance in CUDA[EB/OL]. [2023-10-16]. https://developer.nvidia.com/blog/maximizing-unified-memory-performance-cuda/.
[67] SAKHARNYKH N. Everything you need to know about unified memory[EB/OL]. [2023-10-16]. https://on-demand.gputechconf.com/gtc/2018/presentation/s8430-everything-you-need-to-know-about-unified-memory.pdf.
[68] JOG A, KAYIRAN O, NACHIAPPAN N C, et al. OWL: cooperative thread array aware scheduling techniques for improving GPGPU performance[J]. ACM SIGPLAN Notices, 2013, 48(4): 395-406.
[69] JOHNSON T L, MERTEN M C, HWU W W. Run-time spatial locality detection and optimization[C]//Proceedings of 30th Annual International Symposium on Microarchitecture. Washington D. C., USA: IEEE Press, 1997: 57-64.
[70] JOG A, KAYIRAN O, MISHRA A K, et al. Orchestrated scheduling and prefetching for GPGPUs[C]//Proceedings of the 40th Annual International Symposium on Computer Architecture. New York, USA: ACM Press, 2013: 332-343.
[71] 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 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA). Washington D. C., USA: IEEE Press, 2019: 224-235.
[72] GANGULY D, ZHANG Z Y, YANG J, et al. Adaptive page migration for irregular data-intensive applications under GPU memory oversubscription[C]//Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS). Washington D. C., USA: IEEE Press, 2020: 451-461.
[73] GANGULY D, MELHEM R, YANG J. An adaptive framework for oversubscription management in CPU-GPU unified memory[C]//Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE). Washington D. C., USA: IEEE Press, 2021: 1212-1217.
[74] HILDRUM K, YU P S. Focused community discovery[C]//Proceedings of the 5th IEEE International Conference on Data Mining. Washington D. C., USA: IEEE Press, 2005: 4.
[75] REN B, AGRAWAL G, LARUS J R, et al. SIMD parallelization of applications that traverse irregular data structures[C]//Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). Washington D. C., USA: IEEE Press, 2013: 1-10.
[76] COMER T. Accelerating Geographic Information Systems (GIS) data science with RAPIDS cuSpatial and GPUs[EB/OL]. [2023-10-16]. https://medium.com/rapids-ai/acclerating-gis-data-science-with-rapids-cuspatial-and-gpus-fd012b27af0a.
[77] AMD Corporation. AMD APP SDK OpenCL optimization guide[EB/OL]. [2023-10-16]. https://www.amd.com/system/files/TechDocs/AMD_OpenCL_Programming_Optimization_Guide2.pdf.
[78] AMD Corporation. ARM Mali GPU OpenCL developer guide[EB/OL]. [2023-10-16]. https://documentation-service.arm.com/static/633fe2dbda191e7fe057f2ac.
[79] LI C, AUSAVARUNGNIRUN R, ROSSBACH C J, et al. A framework for memory oversubscription management in graphics processing units[C]//Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM Press, 2019: 49-63.
[80] JIANG S, CHEN F, ZHANG X. CLOCK-Pro: an effective improvement of the clock replacement[C]//Proceedings of the USENIX Annual Technical Conference. Berlin, Germany: Springer, 2005: 323-336.
[81] JALEEL A, THEOBALD K B, STEELY S C, et al. High performance cache replacement using Re-Reference Interval Prediction (RRIP)[J]. ACM SIGARCH Computer Architecture News, 2010, 38(3): 60-71.
[82] YU Q, CHILDERS B, HUANG L B, et al. HPE: hierarchical page eviction policy for unified memory in GPUs[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(10): 2461-2474.
[83] CHE S, BOYER M, MENG J Y, et al. Rodinia: a benchmark suite for heterogeneous computing[C]//Proceedings of the IEEE International Symposium on Workload Characterization (IISWC). Washington D. C., USA: IEEE Press, 2009: 44-54.
[84] STRATTON J A, RODRIGUES C, SUNG I J, et al. Parboil: a revised benchmark suite for scientific and commercial throughput computing[EB/OL]. [2023-10-16]. https://www.semanticscholar.org/paper/Parboil%3A-A-Revised-Benchmark-Suite-for-Scientific-Stratton-Rodrigues/5f3cce1bc739ebfc03e003010d3438bb318efc14?p2df.
[85] GRAUER-GRAY S, XU L F, SEARLES R, et al. Auto-tuning a high-level language targeted to GPU codes[C]//Proceedings of 2012 Innovative Parallel Computing (InPar). Washington D. C., USA: IEEE Press, 2012: 1-10.
[86] YU Q, CHILDERS B, HUANG L B, et al. Coordinated page prefetch and eviction for memory oversubscription management in GPUs[C]//Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS). Washington D. C., USA: IEEE Press, 2020: 472-482.
[87] KIM H, SIM J, GERA P, et al. Batch-aware unified memory management in GPUs for irregular workloads[C]//Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM Press, 2020: 1357-1370.
[88] PARK D, KIM H, HAN H. Page reuse in cyclic thrashing

of GPU under oversubscription: work-in-progress[C]// Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES). Washington D. C., USA: IEEE Press, 2020: 15-16.
[89] LI L D, CHAPMAN B. Compiler assisted hybrid implicit and explicit GPU memory management under unified address space[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New York, USA: ACM Press, 2019: 1-16.
[90] CHANG C H, KUMAR A, SIVASUBRAMANIAM A. To move or not to move? Page migration for irregular applications in over-subscribed GPU memory systems with DynaMap[C]//Proceedings of the 14th ACM International Conference on Systems and Storage. New York, USA: ACM Press, 2021: 1-12.
[91] MARKTHUB P, BELVIRANLI M E, LEE S Y, et al. DRAGON: breaking GPU memory capacity limits with direct NVM access[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Washington D. C., USA: IEEE Press, 2018: 414-426.
[92] WU K, REN J, LI D. Runtime data management on non-volatile memory-based heterogeneous memory for task-parallel programs[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Washington D. C., USA: IEEE Press, 2018: 401-413.
[93] 王嘉伦. 基于 GPU 的大规模数据分析查询的统一内存管理和系统性能优化[D]. 上海:华东师范大学, 2023.
WANG J L. Unified memory management and system performance optimization for GPU-based large-scale analytical query processing[D]. Shanghai: East China Normal University, 2023. (in Chinese)
[94] WANG J L, PANG W H, WENG C L, et al. D-Cubicle: boosting data transfer dynamically for large-scale analytical queries in single-GPU systems[J]. Frontiers of Computer Science, 2023, 17(4): 174610.
[95] BAE J, LEE J, JIN Y, et al. FlashNeuron: SSD-enabled large-batch training of very deep neural networks[C]// Proceedings of the 19th USENIX Conference on File and Storage Technologies (FAST 21). Philadelphia, USA: ACL Press, 2021: 387-401.
[96] CHOUKSE E, SULLIVAN M B, O'CONNOR M, et al. Buddy compression: enabling larger memory for deep learning and HPC workloads on GPUs[C]//Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). Washington D. C., USA: IEEE Press, 2020: 926-939.
[97] HAN S, POOL J, TRAN J, et al. Learning both weights and connections for efficient neural networks[EB/OL]. [2023-10-16]. http://arxiv.org/abs/1506.02626.
[98] JAIN A, PHANISHAYEE A, MARS J, et al. Gist: efficient data encoding for deep neural network training[C]//Proceedings of the ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA). Washington D. C., USA: IEEE Press, 2018: 776-789.
[99] CHEN T Q, XU B, ZHANG C Y, et al. Training deep nets with sublinear memory cost[EB/OL]. [2023-10-16]. http://arxiv.org/abs/1604.06174.
[100] GRUSLYS A, MUNOS R, DANIHELKA I, et al. Memory-efficient backpropagation through time[EB/OL]. [2023-10-16]. http://arxiv.org/abs/1606.03401.
[101] PENG X, SHI X H, DAI H L, et al. Capuchin: tensor-based GPU memory management for deep learning[C]// Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM Press, 2020: 891-905.
[102] AWAN A A, CHU C H, SUBRAMONI H, et al. OC-DNN: exploiting advanced unified memory capabilities in CUDA 9 and Volta GPUs for out-of-core DNN training[C]// Proceedings of the 25th International Conference on High Performance Computing (HiPC). Washington D. C., USA: IEEE Press, 2018: 143-152.
[103] HILDEBRAND M, KHAN J, TRIKA S, et al. AutoTM: automatic tensor movement in heterogeneous memory systems using integer linear programming[C]//Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM Press, 2020: 875-890.
[104] HUANG C C, JIN G, LI J Y. SwapAdvisor: pushing deep learning beyond the GPU memory limit via smart swapping[C]// Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems. New York, USA: ACM Press, 2020: 1341-1355.
[105] LE T D, IMAI H, NEGISHI Y, et al. TFLMS: large model support in TensorFlow by graph rewriting[EB/OL]. [2023-10-16]. http://arxiv.org/abs/1807.02037.
[106] RASLEY J, RAJBHANDARI S, RUWASE O, et al. DeepSpeed: system optimizations enable training deep learning models with over 100 billion parameters[C]// Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. New York, USA: ACM Press, 2020: 3505-3506.
[107] REN J, LUO J L, WU K, et al. Sentinel: efficient tensor migration and allocation on heterogeneous memory systems for deep learning[C]//Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA). Washington D. C., USA: IEEE Press, 2021: 598-611.
[108] CHIEN S, PENG I, MARKIDIS S. Performance evaluation of advanced features in CUDA unified memory[C]// Proceedings of the IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC). Washington D. C., USA: IEEE Press, 2019: 50-57.
[109] 王鹤澎, 王宏志, 李佳宁, 等. 面向新型处理器的数据密集型计算[J]. 软件学报, 2016, 27(8): 2048-2067.
WANG H P, WANG H Z, LI J N, et al. New processor for data-intensive computing[J]. Journal of Software, 2016, 27(8): 2048-2067. (in Chinese)

编辑 陆燕菲