软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2023,34(4):1977−1996 [doi: 10.13328/j.cnki.jos.006732]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

ChattyGraph: 面向异构多协处理器的高可扩展图计算系统*

蒋筱斌[1,2], 熊轶翔[1,2], 张 珩[1], 武延军[1], 赵 琛[1]

[1](中国科学院 软件研究所, 北京 100190)

[2](中国科学院大学, 北京 100049)

通信作者: 张珩, E-mail: zhangheng17@iscas.ac.cn

摘 要: 现阶段, 随着数据规模扩大化和结构多样化的趋势日益凸现, 如何利用现代链路内链的异构多协处理器为大规模数据处理提供实时、可靠的并行运行时环境, 已经成为高性能以及数据库领域的研究热点. 利用多协处理器(GPU)设备的现代服务器(multi-GPU server)硬件架构环境, 已经成为分析大规模、非规则性图数据的首选高性能平台. 现有研究工作基于 Multi-GPU 服务器架构设计的图计算系统和算法(如广度优先遍历和最短路径算法),整体性能已显著优于多核 CPU 计算环境. 然而, 这类图计算系统中, 多 GPU 协处理器间的图分块数据传输性能受限于 PCI-E 总线带宽和局部延迟, 导致通过增加 GPU 设备数量无法达到整体系统性能的类线性增长趋势, 甚至会出现严重的时延抖动, 进而已无法满足大规模图并行计算系统的高可扩展性要求. 经过一系列基准实验验证发现,现有系统存在如下两类缺陷: (1) 现代 GPU 设备间数据通路的硬件架构发展日益更新(如 NVLink-V1, NVLink-V2), 其链路带宽和延迟得到大幅改进, 然而现有系统受限于 PCI-E 总线进行数据分块通信, 无法充分利用现代 GPU 链路资源(包括链路拓扑、连通性和路由); (2) 在应对不规则图数据集时, 这类系统常采用过于单一的设备间数据组织和移动策略, 带来大量不必要 GPU 设备间经 PCI-E 总线的数据同步开销, 导致本地性计算同步等待时延开销过大.因此, 充分地利用各类现代Multi-GPU服务器通信链路架构来设计可扩展性强的图数据高性能计算系统亟待解决.为了达到 Multi-GPU 下图计算系统的高可扩展性, 提出一种基于混合感知的细粒度通信来增强 Multi-GPU 图计算系统的可伸缩性, 即采用架构链路预感知技术对图结构化数据采用模块化数据链路和通信策略, 为大规模图数据(结构型数据、应用型数据)最优化选择数据交换方法. 综合上述优化策略, 提出并设计了一种面向 Multi-GPU 图并行计算系统 ChattyGraph. 通过对 GPU 图数据缓冲区优化, 基于 OPENMP 与 NCCL 优化多核 GPU 协同计算, ChattyGraph 能在 Multi-GPU HPC 平台上自适应、高效地支持各类图并行计算应用和算法. 在 8-GPU NVIDIA DGX 服务器上, 对各种真实世界图数据的若干实验评估表明: ChattyGraph 显著实现了图计算效率和可扩展性的提升, 并优于其他最先进的竞争对手性能, 计算效率平均提升了 1.2×−1.5×, 加速比平均提升了 2×−3×, 包括 WS-VR 和 Groute.

关键词: 大规模; 图计算; 多协处理器; 总线; 通信

中图法分类号: TP301

中文引用格式: 蒋筱斌, 熊轶翔, 张珩, 武延军, 赵琛. ChattyGraph: 面向异构多协处理器的高可扩展图计算系统. 软件学报, 2023, 34(4): 1977−1996. http://www.jos.org.cn/1000-9825/6732.htm

英文引用格式: Jiang XB, Xiong YX, Zhang H, Wu YJ, Zhao C. ChattyGraph: Highly Scalable Graph Computing System for Heterogeneous Multi Accelerators. Ruan Jian Xue Bao/Journal of Software, 2023, 34(4): 1977−1996 (in Chinese). http:// www. jos. org. cn/1000-9825/6732.htm

* 基金项目: 国家自然科学基金(62002350)

收稿时间: 2021-09-12; 修改时间: 2022-04-20; 采用时间: 2022-06-23; jos 在线出版时间: 2022-07-22

ChattyGraph: Highly Scalable Graph Computing System for Heterogeneous Multi Accelerators

JIANG Xiao-Bin[1,2], XIONG Yi-Xiang[1,2], ZHANG Heng[1], WU Yan-Jun[1], ZHAO Chen[1]

[1](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)

[2](University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: Recently, with the increasing trend of data scale expansion and structure diversification, how to use the heterogeneous multi accelerators in modern link to provide a real-time and reliable parallel runtime environment for large-scale data processing has become a research hotspot in the field of high performance and database. Modern servers equipped with multi accelerators (GPU) has become the preferred high-performance platform for analyzing large-scale and irregular graph data. The overall performance of existing research designing graph computing systems and algorithms based on multi-GPU server architecture (such as breadth first traversal and shortest path algorithm) has been significantly better than that of multi-core CPU computing environment. However, the data transmission performance between multi-GPU of existing graph computing system is limited by PCI-E bandwidth and local delay, leading to being unable to achieve a linear growth trend of performance by increasing the number of GPU devices, and even serious delay jitter which cannot satisfy the high scalability requirements of large-scale graph parallel computing systems. After a series of benchmark experiments, it is found that the existing system has the following two types of defects. (1) The hardware architecture of the data link between modern GPU devices is rapidly updated (such as NVLink-V1 and NVLink-V2), and its link bandwidth and delay have been greatly improved. However, the existing systems are still limited by PCI-E for data communication, and cannot make full use of modern GPU link resources (including link topology, connectivity, and routing); (2) When dealing with irregular graph data, such systems often adopt single data movement strategy between devices, bringing a lot of unnecessary data synchronization overhead between GPU devices via PCI-E bus, resulting in excessive time-wait overhead of local computing. Therefore, it is urgent to make full use of various communication links between modern multi-GPU to design a highly scalable graph computing system. In order to achieve the high scalability of the multi-GPU graph computing system, a fine-grained communication based on hybrid perception is proposed to enhance the scalability of the multi-GPU graph computing system. It pre-awares the architecture link, uses the modular data link and communication strategy for different graph structured data, and finally selects the optimal data exchange method for large-scale graph data (structural data and application data). Based on above optimization strategies, this study proposes and designs a graph oriented parallel computing system via multi-GPU named ChattyGraph. By optimizing data buffer and multi-GPU collaborative computing with OpenMP and NCCL, ChattyGraph can adaptively and efficiently support various graph parallel computing applications and algorithms on multi-GPU HPC platform. Several experiments of various real-world graph data on 8-GPU NVIDIA DGX server show that ChattyGraph significantly improves graph computing efficiency and scalability, and outperforms other advanced competitors. The average computing efficiency is increased by 1.2×−1.5× and the average acceleration ratio is increased by 2×–3×, including WS-VR and Groute.

Key words: large scale; graph computing; multi coprocessor; bus; communication

随着硬件不断发展的趋势, 搭载多块通用图形处理单元(general graphic procssor units, GPU)的服务器已成为现代高性能计算(high performance computing, HPC)的主流运算平台[1−4]. 单块 GPU 处理单元提供了万级以上线程数的高并行计算能力和高于 100GB/s 访存带宽, 大幅提高了高通量工作负载的性能. 例如: NVIDIA DGX 工作站包含 8 块 NVIDIA V100 GPU 卡, 单 GPU 提供了多达 5 120 流处理器(SMX)、32 GB DDR5 全局访存. 随着智能产业和物联网的蓬勃发展, 各类事物间的关联数据规模、复杂性都呈现井喷式增长[5,6]. 这类复杂网络通常以图结构(点,边)形式的数据集进行表征, 然而, 多核 CPU 和搭载单块 GPU 协处理器的服务器并行计算平台面对规模不断扩大的复杂图数据, 这类平台的存储和计算能力已经远无法满足大规模图数据的处理需求. 面向搭载多块设备间紧密互联 GPU 的服务器环境, 如何架构和实现大规模图数据并行处理应用和算法,越来越为研究者关注; 同时, 如何应用 Multi-GPU 服务器为解决各类新兴领域的大规模图数据应用(如社交网络分析[6]、异常检测[7]、推荐系统和网络拓扑分析[8])提供高质量、低时延的服务, 也开始得到广泛关注.

在当前 GPU 高性能图计算系统研究工作中, Multi-GPU 图数据并行系统设计逻辑围绕以顶点(vertex)为中心的思想进行图数据切分、图分块迭代计算和整体同步[1−4,9−12]. 具体地, 主处理模块(CPU 端)通过子图切块策略, 将原始图数据均衡划分为分区子图, 以本地性并行计算的方式加载各子图至 GPU 设备, 并以点为中心同步不同设备计算完成的子图内更新顶点值(updated vertex value); 进而经过若干迭代轮加载计算和整体同步、主控逻辑判断后, 直到全局达到特定收敛条件[13−16]. 通常情况下, 基于以顶点为中心并行模型的 GPU 并行处

理有两种设备间任务调度模型.

1) 同步模型(synchronization model): 遵循整体同步并行计算模型, 即在每个迭代计算轮加载和处理本地子图分块之后, 所有设备之间均需全局通信同步[12,17−19];

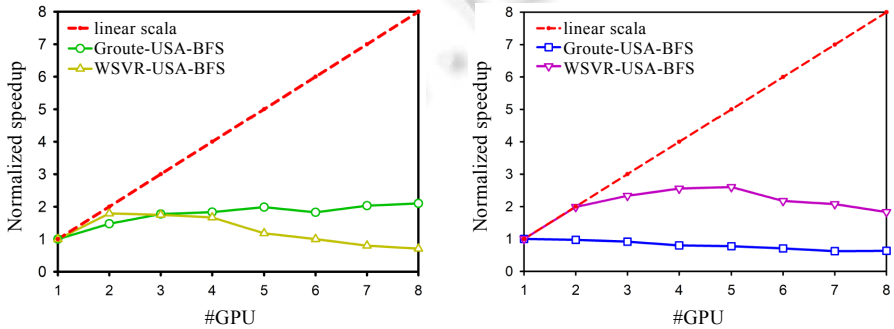
2) 异步模型(asynchronization model): 基于流水线工作列表(worklist)的任务队列实现设备间计算和异步通信, 这使得自定义代码中不需要制定显式同步条件, 只设置细粒度(非批量)同步点来更新本地子图数据[1,17,18].

然而, 经过调研和验证, 现有 GPU 服务器图并行计算系统在扩展到 Multi-GPU 服务器环境时, 其可扩展性和性能仍存在问题和挑战[1,3,6,20]: 首先, 考虑到 Multi-GPU 服务器搭载带宽不同和空间各异的层级内存, 包括主存、GPU 设备的共享内存和全局内存, 而各设备间的数据通过各种不同总线链路形式(PCI-E, NVLink-V1, NVLink-V2)进行相互通信, 这类体系结构上的异构性, 为 Multi-GPU 平台大规模图数据计算系统的扩展性和性能带来设计挑战; 其次, 在应对稀疏和不规则图数据时, 各子图分区的同步内核、计算内核之间均具有内在结果依赖性. 现有的 GPU 图计算系统通过两种典型的总线和 GPU 设备间互联通道(即 PCI-E 和 NVLink)来优化 GPU 服务器间数据通信链路. 从现有系统来看, GPU 间通信与 CPU-GPU 多采用以共享 PCI-E 总线带宽方式来进行数据交换, 然而, PCI-E 总线的带宽和时延局限性限制了整体 Multi-GPU 计算系统的可扩展性, 例如: PCI-E 的带宽约为 2.5GB/s, 延迟达到 0.7ms. 近年来, 随着 NVLink、对等(P2P)通信和基于环的通信技术的发展, 各类 GPU 服务器内设备间的数据交换性能得到了极大提升. 经过调研, 目前最先进的 Multi-GPU 设备间的数据交换方式主要包含如下 3 种通信库实现: 1) 显式数据交换; 2) 统一虚拟地址(unified virtual address, UVA); 3) NCCL(NVIDIA 集合通信库). 这 3 类通信方式都在一定程度上提升了设备间数据交换性能, 为面向 Multi-GPU 高性能服务器进行图并行计算系统设计提供了扩展性和性能上更为弹性的选择[1,11,14,19,21].

进一步, 在 NVIDIA DGX 服务器上, 对当前最先进的 GPU 图并行计算系统 WSVR[12]和 Groute[1]进行广度优先遍历(BFS)和单源最短路径(SSSP)基准测试, 本文对其性能评估发现, 这类系统在多 GPU 设备上的可扩展性仍存在缺陷. 图 1 统计了在扩展到多块 GPU 设备时, WSVR 和 Groute 的性能加速比情况(SSSP 和 BFS 基准测试性能表明, 两个系统的可扩展性仍存在缺陷). 这两类系统仍无法可以达到类线性增长趋势, 甚至在扩展到 4 块 GPU 以上时, 计算性能会有大幅下降. 进而, 本文对 WSVR 和 Groute 的多 GPU 设备间数据通信策略分析发现: 这类系统在设备间协调策略上尚存在缺陷, 无法优化利用多块 GPU 设备进行高效协同计算. 其主要原因包括:

1) 仍依赖于主处理器总线 PCI-E 通信模型. 这类系统忽略了利用现代 GPU 设备间链路和机制进行协同优化, 包括 NVLink 技术和集合通信模型等;

2) 缺乏有效的策略以应对各类不同数据粒度间的非规则性图数据集的数据交换, 从而导致了大量的设备间同步开销, 使这类系统难以权衡 GPU 设备本地性计算和设备间远程通信的开销平衡.



(a) USA-road 运行 SSSP 算法性能对比 (b) USA-road 运行 BFS 算法性能对比

图 1 Groute 和 WS-VR 的扩展性实验

本文对 Multi-GPU 图数据计算系统(WSVR, Groute)进行了深度并行机理性分析, 深入研究了 P2P 和 UVA

两种通信模式(见第 1.2 节). 本文研究发现: 对于不同大小的图计算数据集, 并非所有情况均适用于选择同一类模式. 因此, 构建一套自适应的通信方法会有效提升通信和资源利用率. 该策略不仅能为不同的图应用基准提供高效的并行计算性能, 而且会在图算法的不同执行阶段设置最优化的缓存通信性能. 例如: GPU 的 UVA(unified virtual addressing space)模式提供了易于实现的设备访存读写方式, 更适用于小粒度数据集读写; 而 NCCL 方法可以将设备内的消息细粒度化传输, 降低对等通信开销, 更适用于大粒度的数据移动算法(例如 PageRank). 此外, NCCL 方法在遍历算法中具有良好的通信性能, 但同时也会在多轮迭代过程中引入大量初始化操作开销, 反而会影响图迭代算法的性能. 基于这些观测结果, 本文设计了一种高效、可扩展的大规模图数据并行计算方法. 重点研究了最优通信模式选择.

本文的主要贡献总结如下:

(1) 本文量化了 NVLink, PCI-E 等通信链路技术, 在不同的图基准、数据集、执行阶段和 GPU 规模上, 对当前先进 Multi-GPU 服务器的数据链路特性进行了综合分析;

(2) 基于硬件通信链路特性, 对比分析了 3 类通信模式的通信效率, 设计并实现了一种基于混合感知的细粒度通信模型, 优化了多 GPU 间图数据通信效率, 针对不同的图数据结构采用协同通信模式. 结果表明, 混合图数据感知的通信模式降低了 GPU 间的通信时间 1.2×–1.5×;

(3) 实现了新的高可扩展 Multi-GPU 图并行计算系统 ChattyGraph, 优化了多 GPU 设备内图数据缓冲区以及实现了基于 OPENMP 与 NCCL 的多核 CPU 和多 GPU 设备协同计算, 并提出了 3 类用户友好的 API 接口, 实现了各类标准图并行计算算法(广度优先遍历、PageRank、最短路径遍历等);

(4) 实验结果表明: ChattyGraph 与 GPU 图数据并行计算系统(WS-VR, Groute)相比, 达到了 1.5×–3×的性能提升; 同时, 在 8GPU 下, 平均加速比普遍能达到 3×–4×左右, 比 WS-VR 和 Groute 性能加速比提升 2 倍. 另外, 实验也从多角度验证了所提方案的性能优势, 在图数据量和 GPU 数量不断扩大的情况下, ChattyGraph 的图数据处理效率提升越发明显.

本文第 1 节介绍 GPU 高性能图计算系统的相关工作背景, 并给出现代 GPU 设备间的通信链路技术和数据移动策略. 第 2 节给出 Multi-GPU 下高扩展图并行计算系统的架构动机, 并分析高可扩展性的图计算瓶颈和应对策略. 进一步地, 第 3 节详细介绍高扩展图计算系统 ChattyGraph 新的细粒度通信策略. 第 4 节具体描述高扩展图计算系统 ChattyGraph 的实现细节, 包括 GPU 设备内图数据表征及缓冲区优化、在多核 CPU 和多块 GPU 设备间图数据协同计算优化. 第 5 节描述具体的实验平台、实验设计以及实验结果和分析. 最后是本文的总结和未来展望.

1 背 景

利用 GPU 协处理器的高并行计算能力构建高性能、可扩展性图计算应用和算法优化的工作一直是高性能、数据库以及系统领域的研究热点. 随着图数据量的规模不断增大、应用需求的日益增多, 各类基于 GPU 的图计算系统层出不穷, 例如, Medusa[11], Gunrock[19], WSVR[12], Groute[1]等. 这类面向 GPU 服务器的图并行计算系统常围绕以顶点为中心的并行架构设计思想, 将各类图数据应用建模为以顶点为中心的并行计算范式. 为了应对更大规模的图数据应用需求, 研究工作逐渐转向利用搭载多块 GPU 处理器的 Multi-GPU 服务器平台. 这类研究通过对子图分块后持续加载数据至各协处理器设备中进行计算, 实现了充分利用多个 GPU 的平台上协同处理大规模图数据的可行解决方案[1,3,20].

在本节中, 本文首先简要讨论 Multi-GPU 图数据并行处理系统的相关研究工作; 然后, 从软件系统的角度深入分析当前 3 种现代 GPU 设备间数据交换链路架构和通信方式; 最后, 本文具体分析了当前 Multi-GPU 图数据并行计算系统的若干缺陷.

1.1 面向Multi-GPU服务器的图数据并行计算系统

分布式或共享内存的高性能图数据计算研究一直是高性能、数据库以及系统领域的研究热点, 并且催生了诸多面向 GPU 处理器的图计算系统[10−12]和高性能算法设计[9,14,22−24]. 利用 GPU 的大规模并发线程不仅可以

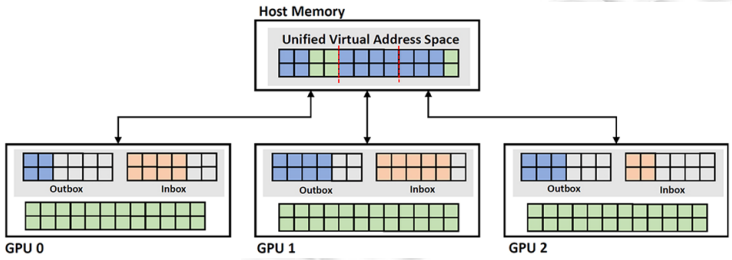
提高常规数据处理的性能, 例如张量计算、矩阵分解等, 而且大幅提升了大规模数据吞吐, 为各处理单元(如thread, warp, block)的数据加载和读写提供了高带宽低时延的并发计算环境.

随着数据规模的不断增加, 近年来, 在搭载多块互通互联 GPU 设备的 Multi-GPU 服务器上开展高性能大规模图数据处理已开始受到各类研究的关注[6,20]. 研究工作主要集中于利用多块 GPU 设备来实现更大规模图数据处理. 然而, 当前 Multi-GPU 图计算系统通过 PCI-E 互联总线统一扩展策略来实现图数据块间的通信和同步, 其整体的性价比低, 即加入更多 GPU 设备所带来的性能提升并不明显, 甚至会显著下降. 本文对当前 Multi-GPU 图数据计算系统的并行范式进行分析, 这类处理技术大多采用随机图划分策略和单一总线通信策略来实现可伸缩性.

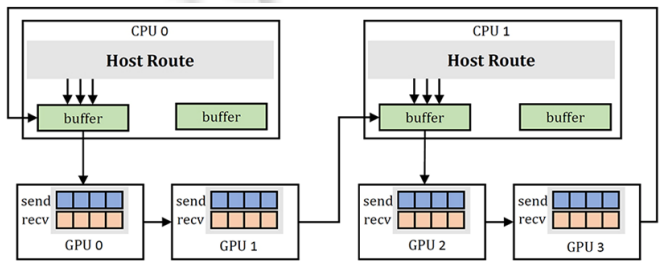
具体地, 以下分别介绍两类当前最先进的 Multi-GPU 图数据并行计算系统架构, WSVR 和 Groute.

WSVR[12]: 如图 2(a)所示, WSVR 采用点切分方案, 将切分完的边集均匀地分配给多台 GPU 设备. 通信过程采用顶点细化技术来指定边界顶点, 以最小化数据量通过 PCI-E 总线实现 GPU 之间的通信. WSVR 的并行计算过程利用离线(offline)和在线(online)两个阶段计算各个设备间需要同步更新的工作负载: 1) 离线阶段在预处理过程中识别并标记边界顶点; 2) 在线阶段利用并行二进制前缀来动态地从边界顶点中细化更新的顶点, 即, 在通信中只传输在上一轮计算中更新且其他设备需要的顶点. 在多个 GPU 设备数据交换阶段, WSVR 采用主机端缓存策略, 通过在 CPU 主存端开辟一块统一虚拟内存(UVM)空间, 各 GPU 设备访存区与该 UVM 缓存空间交互同步, 实现上传和下载更新数据的操作, 最终同步各 GPU 设备间更新的数据. WSVR 采用的上述数据交换策略虽然减少了冗余边界的同步开销,但使用 UVM 作为主机端缓存的策略, 导致各 GPU 设备间的数据交换均以 PCI-E 总线链路为交换通道, 尤其在多块 GPU 同时数据同步时会带来 PCI-E 总线拥塞, 产生巨大的同步开销;

Groute[1]: 如图 2(b) 所示, Groute 研究了 Multi-GPU 服务器下图数据处理的异步执行模型, 提出了一种多 GPU 设备间的计算任务调度和通信策略. 通过利用 GPU 间的数据链路, Groute 构建环形拓扑来改进 GPU 间的数据通信性能, 采用 P2P(peer-to-peer)和路由(route)选择相结合的方案, 即: 在节点内 GPU 间采用 P2P 直接通信; 节点间由主机端路由选择策略选择合适的路径进行数据交换. 尽管 Groute 通过屏蔽全局同步屏障来优化了设备间异步数据交换, 然而 Groute 的异步编程模型需要用户自定义图计算应用数据的路由 Route、链路 Link 等操作, 系统的局限性使其无法适配更多常见图数据应用.



(a) WS-VR 统一虚拟地址通信(UVA)机制

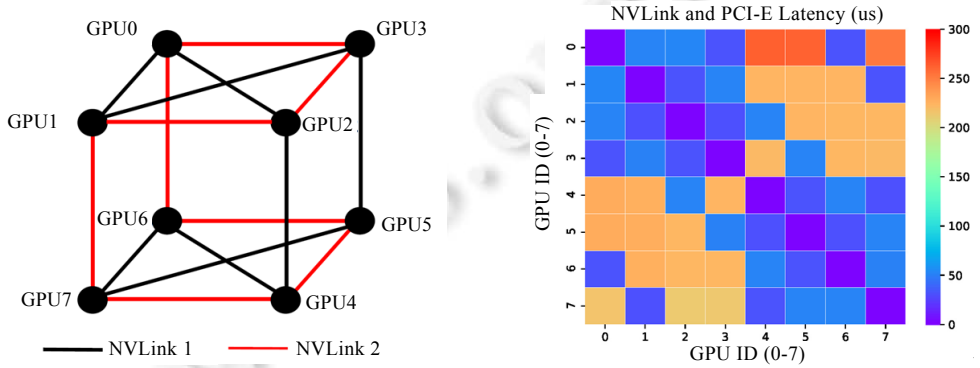


(b) Groute 缓冲区环形链路通信机制

图 2 两种先进的多 GPU 图数据处理系统

1.2 现代GPU间通信方式

近年来, GPU 通常被作为协处理器来弥补 CPU 在大规模数据并行计算上的缺陷, 而 CPU 处理器主要负责数据的预处理和后处理、内存分配、进程管理以及调度和管理 GPU 核操作和 GPU 间数据通信. CPU 与 GPU 之间通过前端总线(FSB, 例如 QPI 和 HyperTransport)相互连接. 前端总线与北桥连接, 并由北桥通过 PCI-E, NVLink-V1 和 NVLink-V2 链路连接到 GPU 显卡. 图 3 展示了本文实验提供的 GPU 服务器平台, 搭载 8 块互联互通 NVIDIA V100 GPU 卡的 NVIDIA DGX 服务器, 每个 GPU 包含 5 120 个 SMX、32GB 全局内存, 并通过 PCI-E, NVLink-V1 和 NVLink-V2 技术互联互通连接. 通过量化 GPU 间链路的性能特性, 表 1 给出了这 3 种通信链路在多个 GPU 之间跃点时的通信带宽(MB/s)和延迟(ms), 证实了 PCI-E 总线在通信性能上的瓶颈.



(a) NVIDIA DGX 的拓扑结构　　(b) GPU 间通信带宽热力图

图 3　具有 8 个 NVIDIA V100 GPU 的 NVIDIA DGX 服务器的内部链接拓扑

表 1　GPU 间链路互联特性

设备链路	带宽(MB/s)	0-hop (ms)	1-hop (ms)	2-hop (ms)
PCI-E	2 872.111	0.348 176	0.686 045	0.732 432
NVLink-V1	17 196.9	0.058 15	0.117 411	0.176 563
NVLink-V2	27 228.67	0.036 726	0.078 07	0.113 05

在本节中, 下面以 PCI-E, NVLink-V1 和 NVLink-V2 为例, 重点讨论 GPU 设备间数据通信的互联技术.

PCI-E: 高速串行计算机扩展总线标准, 即外围组件互联快速总线. 在高性能 GPU 服务器内, 设备间的数据链路通过 PCI-E 总线构架. 一个或多个 GPU 设备通过 PCI-E 总线连接到 CPU 主机端. 与 CPU 和 DRAM 之间数据带宽速度相比, 由于大量待处理数据需要从源设备(磁盘或源 GPU 设备访存)加载至 DRAM 之后, 再经 PCI-E 加载至目的 GPU 访存空间, 局限的 PCI-E 总线带宽限制了多块 GPU 协处理器和 CPU 之间的数据传输效率. 因此, 局限的 PCI-E 总线带宽和延迟使其成为 GPU 高性能算法设计的主要性能瓶颈[25−28], 无论是从主机内存加载到 GPU 的数据通信, 还是启用 P2P 通信时 GPU 设备间的数据移动. 从表 1 可见: 相对比 NVIDIA NVLink 技术, PCI-E 总线的数据通信带宽明显偏低,并且在 *n*-hop 通信延迟上开销远高于其他两类(平均 7×, 9.5×). 此外, PCI-E 由于数据一致性必须在软件层面进行统一管理, 导致了在执行 CUDA 接口之后需要刷新缓存, 同时, 其系统级别的原子性无法保障;

NVLink-V1: 基于高速信号互联(NVHS)总线[20], 其控制器由 3 层组成, 即物理层、数据链路层以及交换层, 提供比 PCI-更快的设备间对等通信, 可以同时支持 CPU-GPU 和 GPU-GPU 之间的点到点互联通信. NVLink 通过插槽连接, 作为双向链路, 每个 NVLink 都包含上行链路和下行链路两个子链路.此外, NVLink 和 PCI-E 的效率特性均取决于数据包的大小(如后文图 5(a)所示). 从表 1 可见: 通过 NVLink-V1 连接的多块 GPU 在发生数据跨设备点通信时, NVLink-V1 的通信带宽和延迟明显优于 PCI-E 总线,数据传输带宽提升达到 6.7×, 并且通信延迟降低 7×;

NVLink-V2: 基于 NVLink-V1 优化的桥接器. 相对比 NVLink-V1, NVLink-V2 使用硬件地址转换, 提供了

Unified Memory 机制支持 GPU 设备直接访问 CPU 地址, 大幅提高了设备间的链路带宽, 并为 NVIDIA Tesla GPU 设备提供了更多的链路插槽[20]. 当添加更多的 NVLink-V2 链路连接时, 可以实现更高的设备间通信带宽. 与 NVLink-V1 相比, NVLink-V2 将每条链路的带宽优化了约 50%, 如表 1 所示. 此外, 本文通过不同数据集大小的通信方式, 进一步评估了 NVLink-V1 和 NVLink-V2 的通信带宽对比.

图 3(a)以 NVIDIA DGX 高性能服务器为例, 具体地展示了现代 Multi-GPU 服务器内各 GPU 之间的互通互联情况. 在 NVIDIA DGX 的拓扑结构中, 每个 GPU 设备占据立方体的一个黑点, 16 个边是 NVLink 连接, 由 8 个 NVLink-V1 和 8 个 NVLink-V2 组成, 剩余两点之间的连接由 PCI-E 组成. 数据通过 PCI-E 通信时, 需要跨越 CPU 内存端进行两段式通信(显存-主存-显存), 因此会导致低效效果. 而 NVLink 和 PCI-E 的 GPU 之间的两个终端无法自路由, 需要由用户指定路由的源 GPU 和目标 GPU 在内核函数中实现显式路由操作.

基于底层互联链路, 在多 GPU 集群中设计了多种通信方式, 包括点对点通信和集合通信. 点对点通信直接依附于通信链路, 通过 CUDA 中的通信 API: cudaMemcpy 和 cudaMemcpyPeer 实现 CPU-GPU 和 GPU-GPU 之间的通信. 集合通信通常会涉及多个发送者和接收者, 其操作包括广播(broadcast)、分散(scatter)、聚集(gather)等. 有效第实现集合通信, 要综合考虑集群中的所有设备状态, 因此需要从底层的硬件拓扑结构出发, 选择合适的通信路径, 以解决通信过程中的冗余、同步和死锁问题. 在本节中, 下面介绍 UVA 和 NCCL 两种通信方式, 优化传统通信过程中的通信延迟.

UVA: 统一虚拟寻址(unified virtual addressing)是 CUDA v4.0 版本起支持的新特性. UVA 本质上没有缓解 PCI-E 的低带宽和高延迟, 其通过零拷贝(zero-copy)内存为所有内存提供一个虚拟地址空间, 允许 GPU 中的访存指针直接内存访问, 而不需要显式的数据拷贝; 通过固定内存(pinned memory)锁定主存存储页, 避免因分页引起的页面缺失. Dipanjan Sengupta 等人[10,29]在 GraphReduce 中发现: 顺序内存访问时, UVA 具有更好访存性能. 因此, GraphReduce 采用 UVA 方式分配内存空间, 并配合数据预取将数据传输和 GPU 计算过程重叠, 以此提高图计算效率. 统一内存(unified memory, UM)基于 UVA 实现, 创建一个 CPU 和 GPU 的统一托管内存池, 内存池中已经分配的空间可以由 CPU 或 GPU 中相同的指针直接访问. UM 在图计算领域常被用于主存与显存的交互方式, Pengyu Wang 等人[30]评估了不同 UM 配置下图计算的工作负载, 设计了 GRUS 解决超额调配 GPU 内存场景, 针对不同图数据大小和数据结构优先级, 采用不同的内存访问模式, 优化了图计算的内存访问效率. 然而, 由于多 GPU 之间的数据竞争, 该方法无法很好地扩展到多 GPU 环境中;

NCCL: NVIDIA 开发的 GPU 集合通信库 NCCL (NVIDIA collective communication library)高效地实现了多设备间的高性能通信, 现已集成到多个深度学习平台上. 目前, NCCL 库有两个版本: NCCL-V1(开源)和 NCCL-V2(闭源). 为了最大化传输带宽, NCCL 通信库可以自动识别节点内的 NVLink, PCI-E 和 QPI 链路, 在硬件拓扑结构中构建环状通信路径. 将大数据集细粒度切分成小块, 沿环状网络以流水线形式传输. 据分析: 当数据切分份数远大于设备数时, 集合通信所需要的时间趋于点对点通信时间, 即集合通信时间不会随着设备数量的增加而增加, 为集合通信的扩展性能提供了可能.

如表 2 所示, 以 3 种最为常见的点对点通信方式为例: Zero-Copy 是采用 UVA 特性的传输形式, MemCopy 是传统显式的数据拷贝形式, MemCpyAsync 是异步的显式数据拷贝形式.

表 2 点对点通信在传输不同大小数据时的通信延迟

设备间数据传输大小 Size (KB)	设备间零拷贝 Zero-Copy (ms)	同步式设备间内存访问 MemCopy (ms)	异步式设备间内存拷贝 MemCpyAsync (ms)
4	0.017	0.040	0.025
16	0.037	0.061	0.050
64	0.118	0.136	0.124
256	0.621	0.405	0.372
1 024	3.317	1.497	1.419
4 096	25.272	5.391	5.307
16 384	174.579	27.144	28.302

从表中可以看出: 对于小数据传输时, 选择 Zero-Copy 的通信延迟都要优于其他两种; 而在传输大数据, Zero-Copy 性能不佳. 结合图 4 对比 UVA 和 NCCL 在传输不同数据大小时的通信带宽, 同样验证了 UVA 在传

输小数据时的通信带宽要略优于 NCCL; 而 NCCL 在广播大数据时, 通信带宽要远优于 UVA 和其他点对点通信方式.

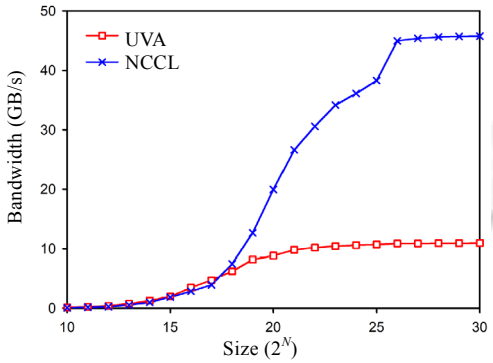


图 4 UVA 和 NCCL 在传输不同大小数据时的通信带宽对比

- 现有系统扩展到 Multi-GPU 链路技术时存在的可扩展性缺陷

综上所述, 当前最先进的面向 Multi-GPU 服务器的大规模图数据并行计算系统[1,3,20]尽管为解决大规模图数据应用问题提供了均衡图切分和异步同步等各类策略以适用于大规模图并行计算, 然而这类 Multi-GPU 图数据并行计算系统仍然在系统可扩展性上无法表现优化的性能, 甚至在 4 块以上 GPU 上的数据处理性能反而下降明显. 经过上述系统性分析, 本文发现当前 Multi-GPU 图计算系统存在两类重要缺陷: 1) 部分忽略了系统内部连接(包括链路拓扑、连通性和路由)的设计; 2) 通常采用过于单一的设备间数据组织和移动策略, 如 UVA 等, 这种选择单一的 GPU 显式数据传输技术在应对不规则图数据集计算时, 会带来大量设备计算同步开销, 并且无法充分利用高性能的 NVLink 链路带宽资源. 根据上述实验评估结论, 本文进一步给出 Multi-GPU 高可扩展图计算系统的设计动机.

2 Multi-GPU 下高扩展图并行计算系统的设计动机

随着 GPU 设备间链路通信技术的持续更新, 设备间数据移动的性能得到显著改进. 然而, 现有相关研究工作在设计 Multi-GPU 图并行计算系统时, 在一定程度上忽视了对这类先进设备间链路技术优势的使用, GPU 设备间高效的链路带宽资源并未得到充分利用.

在本节中, 基于对服务器可伸缩性评估的研究, 本文总结了不同节点间互连技术在不同粒度图数据、通信模式以及负载均衡的影响(带宽、延迟等). 之后, 给出了在扩展到多 GPU 平台上架构和设计大规模图并行计算系统所面临挑战和应对策略选择.

2.1 扩展到Multi-GPU平台图并行计算的优化策略选择

- 体系结构感知的连通性预判

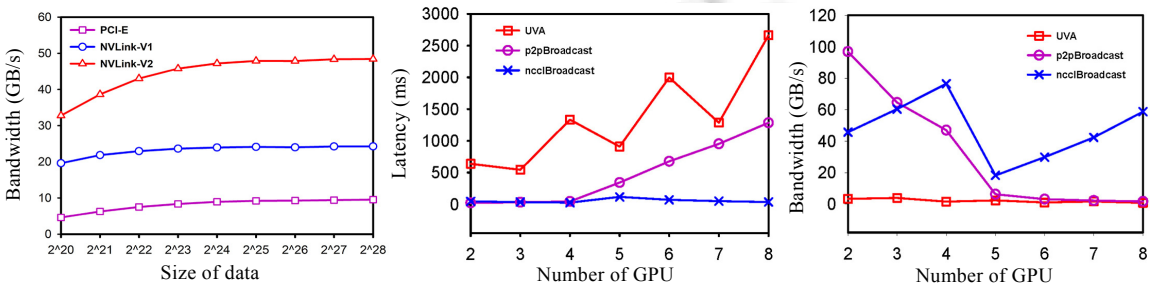
在 Multi-GPU 服务器中, 各 GPU 设备均配置若干链路桥接器(图 3), 点对点(P2P)链路的可选路径也往往各不相同. 这种异构连接性, 使得当前 Multi-GPU 图计算系统的单一通信方式难以实现最优的通信链路选择. 传统 P2P 通信需要对传输到不同设备的顶点进行分类别打包, 否则将会有通信数据冗余. 此外, P2P 通信无法充分利用硬件网络拓扑中的 NVLink 链路, 造成链路资源利用率低下. 从图 5(b)的延迟和性能在多块 GPU 设备的评估表明: 当从 4 块 GPU 扩展到 5 块 GPU 时, 系统性能会急剧下降. 进一步, 根据图 3(a)中各 GPU 设备的互联情况可以看出: 上层的 4 个互联设备[GPU0–GPU3]和下层 4 个设备[GPU4–GPU7]不在同一平面连接, 这会导致可扩展性能不佳. 通常情况下, 同一平面互联互通的多块 GPU 设备间的通信延时要远小于跨平面 GPU 之间的通信.

因此, 在进行大规模大粒度数据移动操作之前, 通过预先检测服务器体系结构下多 GPU 设备间的连通性,

从而预判数据路由流的最优选择, 避免设备之间单一的数据传输路径.

- 设备间同步数据的粒度选择

从图 5 给出的数据可见, 利用不同链路技术的数据传输带宽随数据包大小而变化. 具体地, 在传输 1M–256M 数据包大小时, GPU 设备间的数据链路带宽随着数据包大小的增加, 也同样呈现出上升趋势. 同时, 从图 5(b)的带宽和延迟分析发现: 当 GPU 设备数量大于 4 时, 设备间的数据移动开销增大, 导致点对点传输(UVA, P2PBroadcast)操作的整体性能急剧下滑. 同时, 不同数据包的延迟与数据集的大小成类线性关系. 对比来看, WSVR 通过利用统一内存访问(UVA)在 CPU 内存中缓存顶点值的方法, 在处理 GPU 计算同步时, 由于每轮迭代都需对顶点进行大量更新和同步, 势必带来 CPU-GPU 间 I/O 操作, 引入过大的延迟和同步开销.



(a) 传输 1M–256M 数据时的通信带宽 (b) 3 种广播通信模式的通信延迟 (c) 3 种广播通信模式的通信带宽

图 5 不同大小的数据包在各链路技术之间通信性能的情况

因此, 在应对不规则的、稀疏的图数据集时, Multi-GPU 图并行计算系统的可扩展性不仅需要考虑根据数据粒度来选择最优的设备间链路, 更为重要的是, 决策传输数据包的大小和选择更为合理的数据传输操作(P2P/UVA/显式/集合通信等), 以达到最优传输性能.

2.2 本文的方法

为了使当前基于 GPU 的图数据并行计算系统能够更好地扩展到 Multi-GPU 平台上, 本文拟构建优化的通信模型来识别各 GPU 设备的连通性和工作负载调度情况. 第 3 节将介绍我们的 Multi-GPU 图计算系统的全新基于混合感知的细粒度图计算通信模型. 为了解决不同结构数据通信模式选择和计算与通信平衡性的权衡挑战, 本文设计并实现了一个通用的 GPU 图并行计算框架, 并对所提策略(GPU 缓存优化、多核 CPU 和多 GPU 设备协同)进行集成化实现, 形成面向 Multi-GPU 图计算并行系统 ChattyGraph, 第 4 节将详细描述其实现部分.

3 面向 Multi-GPU 服务器基于混合感知的细粒度图计算通信模型

通过数据感知通信[3,4]和 GPU 间连接性[20]和本地计算与远程通信的权衡分析[25]的动机启发, 我们提出了新的多 GPU 图数据细粒度通信模型, 以解决第 2 节所提挑战. 当前的 Multi-GPU 图数据并行计算系统无法充分利用 GPU 设备间的数据链路, 导致系统可扩展性和性能无法达到最优化, 而本文所提出的全新微调可伸缩图数据并行处理策略用于选择最佳通道的结构感知数据通信运行时. 为了解决 Multi-GPU 图计算系统的可扩展性问题, 本文通过以架构拓扑中高带宽链路的资源利用为中心, 来构建高效设备间数据交换机制.

3.1 混合图感知内部通信运行时

如第 2 节讨论所述, 现有 Multi-GPU 图数据并行计算系统在迭代计算执行过程中选择了单一特定的通信模型. WSVR 图数据计算框架采用统一内存管理技术(UVA)在主机内存端进行数据同步, Groute 图数据计算框架也采用主机内存端进行数据同步, 并配合传输路由, 优化了 WSVR 中完全的 UVA 形式, 属于不完全的主机内存端同步.

相比之下, ChattyGraph 使用细粒度和粗粒度的 GPU 间通信操作, 这是一种混合的数据结构感知的内部通

信运行时. 这是因为在不规则图数据处理的并行迭代过程中, 不同的处理数据集有不同大小的值需要同步, 并要求不同类型的并行. ChattyGraph 旨在为不同粒度的结构数据和应用数据、状态数据提供不同的 GPU 间通信操作以及不同的 GPU 内内存访问. 为了降低 PCI-E 的带宽、延迟等资源局限性, ChattyGraph 充分利用 NVLink 构建 GPU 之间的路径, 大大排除了 CPU 参与的计算. 如下所示, ChattyGraph 在我们的混合运行时中主要选择了两个优化的通信操作.

- 通过集合操作的结构数据路径

统一虚拟寻址主要用于存储在 CPU 主机缓冲区中的整个结构图数据集, 在初始化阶段, 经由 CSR 图数据压缩后分割和分配给多 GPU, 各 GPU 访存内设置切分后, 部分图数据大小的空间与该缓冲区对等, 包括顶点偏移量和边, 底层 GPU 设备通过寄存器操作以保障缓存的一致性. 然而, 底层有限的 PCI-E 带宽使得 CUDAMemcpyAsync 操作无法扩展到从多个设备访问海量图数据结构, 成为扩展到多 GPU 系统时的主要瓶颈. 与传统的结构数据同步机制不同, ChattyGraph 通过预先分配全局顶点值数组的缓冲区来部署广播操作, 因为所有互连 GPU 设备中都存在大量对更新值的访问. 更具体地, ChattyGraph 消除了 CPU 内存缓冲区作为路由器的参与. 整个同步阶段完全在 GPU 之间执行, 如图 6 蓝色区域所示: ChattyGraph 将原本位于 CPU 内存端的全局顶点值数组移入各 GPU 设备显存中, 在每个 GPU 设备中维护一个全局顶点值数组, 更新的顶点通过 NCCL 进行链路广播, 以此最大限度地减少 CPU 的参与. 由于全局顶点值数组保存在设备显存中的原因, 顶点值的更新过程可以直接在 GPU 设备中完成而无需 PCI-E 的参与. ChattyGraph 充分利用 GPU 多线程的特性 (如图 7 所示), 单线程控制数组对应位置并对对应位置的值进行更新, 图中 recvBuffer 的 V1 位对应全局顶点值数组的 V1 位.

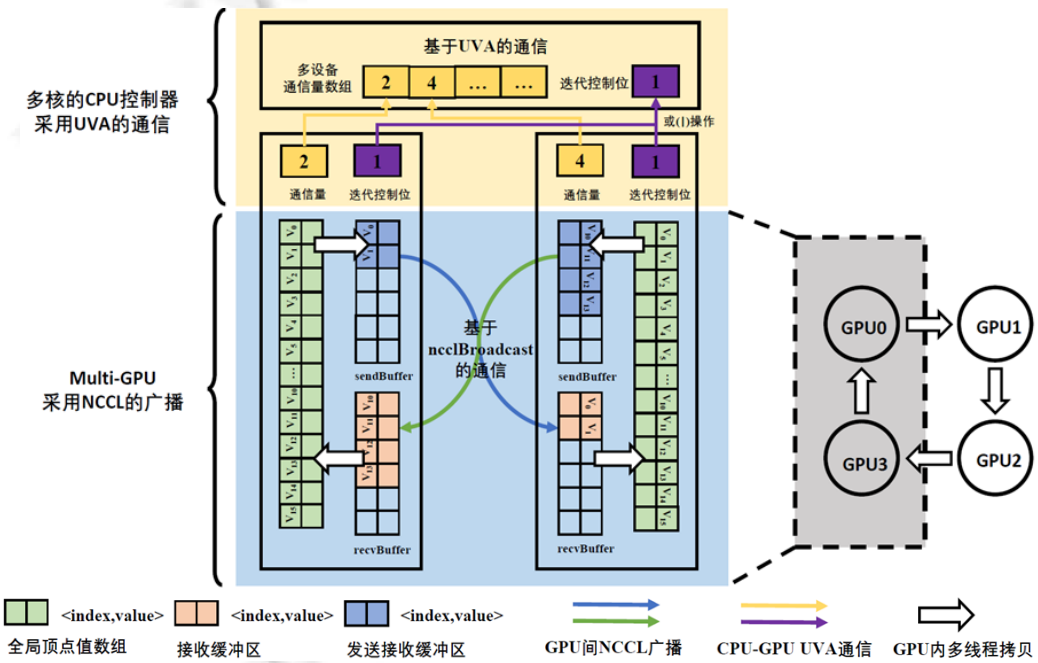


图 6 混合通信模型

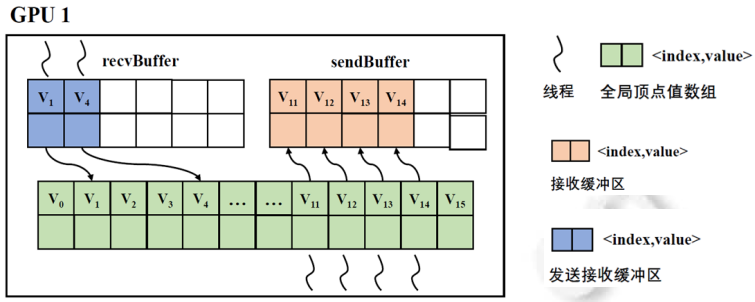


图 7 多线程更新

由算法 1 的 ChattyGraph 主控流程伪代码所示: 当设备完成本轮计算任务(第 10 行)后, 需要通信的顶点值会保存在发送缓冲区, 并由 NCCL 的广播函数进行广播(第 15 行), 广播的目的地址是每个设备的接收缓冲区. 在下一轮迭代开始时, 所有设备都需从上轮迭代的接收缓冲区中更新顶点值(第 6 行), 通过多线程管理接收缓冲区数组的每一位, 更新对应关键词的 *value* 值. 通过采用集合操作的 GPU 内弹性缓冲区空间策略. 值得注意的是: 各 GPU 设备间的结构数据以 NCCL 的广播策略感知 GPU 间链路通信效率, 构建 GPU 集群硬件环状拓扑, 经由 NVLink 高速通路进行数据移动. 这一策略显著地降低了对统一虚拟寻址的操作开销, 不仅一定程度上摆脱了 PCI-E 的带宽资源依赖, 而且有效避免了 GPU 的缓存一致性保障机制所带来的开销.

- 通过 P2P 操作的状态数据路径

图计算迭代过程中的状态数据主要包括 GPU 设备状态、需要传输的定量数据计数和迭代间控制的标志位等. 状态数据集是迭代式图计算应用所必需的. 通过度量, 状态数据量的粒度相对结构数据量较小, 然而对在 Multi-GPU 处理中的系统扩展性能具有不可忽视的影响, 因为 GPU 设备的状态控制需要在每一轮迭代后根据不断更新的状态值判断是否需要同步, 进而影响迭代轮收敛结果. 这类状态数据如图 6 黄色区域所示. 具体地, 算法 1 给出了 GPU 的状态数据的执行流程. 状态数据的通信量会影响 CPU 端显式数据拷贝指令的发布, 即第 12 行需要将通信量拷贝回主存, 以便第 15 行广播指令确定通信数据数量. 因此, 当设备完成本轮计算任务(第 10 行)后, 会首先将该设备计算得出的通信量值(*count* 值)通过 UVA 方式拷贝到内存, 以设备 ID 作为关键词检索(*devID.count* 值)(第 12 行); 整个系统受迭代控制位控制, 每轮迭代结束前, 都需判断系统是否停止迭代(第 19 行), 因此, 每个设备将计算得到的迭代控制位通过 UVA 方式拷贝到内存中, 并由 CPU 管理是否需要结束系统迭代. 然而, 大量同步操作使设备的并行性普遍陷入串行模式执行. 为了减少缺陷, 我们采用多核控制多 GPU 设备(第 3 行), 即通过多线程的线程号来管理 GPU 设备号, 以对这些数据实现异步 I/O 策略可能性. 与传统图数据计算系统采用单核控制多 GPU 设备相比, 单核控制多 GPU 设备虽然在不同 GPU 设备计算和通信中实现并行, 但是由于单核的原因, CPU 对控制指令的发射过程依旧是串行的, 如后文图 9(a)所示. 传统图计算系统大都采用 for 循环管理 GPU 设备号, CPU 对多 GPU 设备号依次串行发射控制指令, 如若指令间没有阻塞点, CPU 无需等待 GPU 运行结果, 实现多 GPU 并行计算. 而当计算和传输中间存在阻塞点时, 例如需要等待计算结果完成才能进行正确的数据传输(第 11 行), 阻塞点(如 streamSynchronize)会中断 for 循环中的控制指令流, CPU 对其余 GPU 设备的同步和计算指令无法发出, 阻塞其余 GPU 设备正常运行, 不利于多 GPU 设备的并行性能提升. 因此, ChattyGraph 采用多核并行控制, 通过多线程的线程号管理多个 GPU 设备号, 多线程的控制指令并行发出, 如后文图 9(b)部分所示, 即使单个设备的计算过程中存在阻塞点, 也不会影响其余设备的正常运行.

算法 1. ChattyGraph 主控流程伪代码.

方法:

(1) *initialization*;

(2) **WHILE** Iteration not finish **do**

```
(3)    devID=threadID;
(4)    FOR targetDevID in 0 to numGPUs do
(5)      IF targetDevID not equal to devID THEN
(6)        download(devID.values,targetDevID.values,targetDevID.count);
(7)      END IF
(8)    END FOR
(9)
(10)   compute(·);
(11)   streamSynchronize(·);
(12)   cudaMemcpy(devID.count,count);
(13)
(14)   FOR targetDevID in 0 to numGPUs do
(15)     ncclBroadcast(devID.sendBuffer,targetDevID.recvBuffer,devID.count,devID);
(16)   END FOR
(17)
(18)   /* Does iteration finish? */
(19)   IF all devices finished THEN
(20)     Iteration finish
(21)   END IF
(22)   ELSE THEN
(23)     Iteration not finish
(24)   END ELSE
(25) END WHILE
```

其中, 设备信息通过设备 ID(包括源设备 *devID* 和目标设备 *targetDevID*)进行检索, 包括设备中存储的全局顶点值数组(*devID.values*)、设备发送缓冲区(*devID.sendBuffer*)、设备接收缓冲区(*devID.recvBuffer*)和设备通信量(*devID.count*).

download 函数用于在 GPU 设备中将接收缓冲区中的顶点值更新到全局顶点值数组中; *streamSynchronize* 函数为 *CUDA API*, 用于阻塞 CPU 线程等待 GPU 运行结束; *cudaMemcpy* 函数为 *CUDA API*, 用于 CPU 与 GPU 间显式的数据通信; *ncclBroadcast* 为 *NCCL API*, 用于在多 GPU 之间实现广播通信; *compute* 函数提供的图算法实现的 API 接口如下.

- *InitVertex*(*vertex*): 用于初始化本地共享内存中的顶点值, 其中, 参数 *vertex* 表示通信同步后的顶点;
- *ComputeScatter*(*vertex*,*edge*,*shared_vertex*): 通过全局内存中顶点和边值, 每个线程计算对应共享内存顶点, 其中, *vertex* 和 *edge* 表示全局内存中的顶点和边, *shared_vertex* 表示每个线程对应的共享内存存储顶点;
- *ComputeReduce*(*shared_vertex*,*nbr_shared_vertex*): 用于更新邻居顶点, 其中, *shared_vertex* 表示共享内存中顶点, *nbr_shared_vertex* 表示共享内存中顶点的邻居顶点;
- *UpdateStatus*(*computed_vertex*,*previous_vertex*): 用于判断顶点在本轮是否有更新, 其中, 参数 *computed_vertex* 表示顶点本轮计算的结果, *previous_vertex* 表示顶点上轮计算结果.

4 ChattyGraph 系统实现与优化

本节对上述的细粒度图数据通信模型进行了集成实现, 设计了面向 Multi-GPU 高可扩展图并行计算系统 ChattyGraph. 图 8 给出了 ChattyGraph 的整体系统架构, 主要包含两大模块: 通信引擎和计算引擎. 其中: 通

信引擎用于 GPU 设备间数据传输, 包括多 GPU 间的环状广播和单 GPU 内数据更新; 计算引擎用于 CPU 对 GPU 的核函数的执行和控制逻辑, 各迭代轮计算结果保存在相应缓冲区以便下轮通信. ChattyGraph 通过标准 C++和 CUDA 实现, 为 Multi-GPU 下图并行计算提供了统一运行引擎, 通过提供用户友好的以顶点为中心的编程 API 接口, 以实现多 GPU 设备图数据应用.

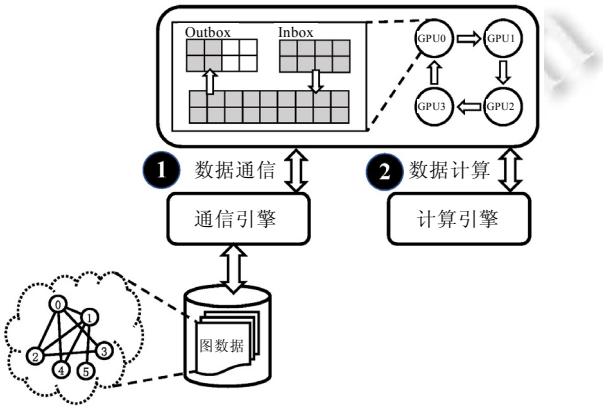


图 8 ChattyGraph 系统架构图

4.1 GPU图数据表征存储及设备缓冲区优化

通信引擎主要负责 CPU-GPU 与 GPU-GPU 间的数据读写和同步, 对顶点数据在全局 GPU 间同步采用 ncclBroadcast 广播, 而对于状态数据采用在主存中开辟 UVA 空间, 以显式 cudaMemcpyAsync 提供传输策略. 在通信模块中, 由于各 GPU 全局访存空间有限, 缓冲区的预分配和初始化对设备间数据传输和同步、GPU 缓存空间的优化尤为重要.

在 Multi-GPU 的数据初始化阶段, ChattyGraph 对各 GPU 设备一个发送缓冲区和两个接收缓冲区. 此外, 考虑到顶点值数据的持续读写更新, ChattyGraph 采用全局顶点值数组, 用以记录每一轮迭代顶点的计算值. 在每一轮 GPU 计算迭代轮开始之前, 各 GPU 设备都会从接收缓冲区中将上一轮迭代从其他设备中接收到的顶点值更新进全局顶点值数组, 并从全局顶点值数组中取出所需要的顶点进行计算过程. 最终迭代轮的计算结果仍保存于全局顶点值数组中, 同时将更新后边界顶点填充进发送缓冲区中, 等待同步通信.

为了实现多任务的并行执行, ChattyGraph 通过设置发送和接收缓冲区可以提升 GPU 设备间的数据移动性能. 双接收缓冲区保证 GPU 在更新时不会因读写冲突而写入脏值. 因为多设备异步计算的原因, 多设备间执行步骤相同, 但执行速度不同. 假设当设备 *A* 正在从本设备的接收缓冲区更新上轮数据到全局顶点值数组中, 但此时设备 *B* 已计算完成并开始向设备 *A* 的接收缓冲区发送本轮计算的更新值, 此时便形成了读写冲突. 因此, 我们采用双接收缓冲区, 并利用奇偶迭代数交替使用两个接收缓冲区.

4.2 基于OPENMP与NCCL的多核GPU协同计算

在CPU和GPU协同计算方面, 现有Multi-GPU图并行计算系统常采用单核控制多设备的编程模式. 然而, 我们发现: 现有并行计算系统中常会出现阻塞点, 限制多 GPU 高性能并行. 以指令顺序为基础, 在顶点的计算指令中, 不仅要得到每个顶点的计算更新值, 还需要统计通信量大小. 而不管是显式的 P2P 传输 CUDAMemcpyPeerAsync 还是 NCCL 通信库中的 ncclBroadcast, 其 API 接口 *count* 参数都为整数型数据类型. 因为 CPU 与 GPU 之间的异步特性, CPU 在发送完成计算核函数指令(第 10 行)后不会等待 GPU 计算, 随即发送通信指令(第 12 行), 而此时通信量还未计算得出, 即发送了错误的通信数量. 为了获得正确的通信值,现有 GPU 图并行计算系统不得不使用同步机制 cudaStreamSynchronize(第 11 行)来等待设备计算的完成, 这将导致大量 straggler 同步开销, 影响计算任务的并行度.

为了解决阻塞点问题, ChattyGraph 摈弃了传统单核控制多设备的编程模式, 提出了多核控制多设备机制.

具体地, ChattyGraph 采用 OpenMP 多核编程框架以控制多线程计算, 每一个线程控制对应 GPU 的核函数调度(第 3 行), 最终实现多 GPU 与 CPU 多核处理器间的协同并行处理(如图 9(b)部分). 采用多核控制多设备的机制下, ChattyGraph 允许单设备执行其相应计算指令而无需等待其他设备执行情况, 因而不会影响整体系统的并行度和性能. 据对现有 GPU 图计算系统调研可知, 采用 OpenMP 和 NCCL 的协作计算模型尚未有相关研究工作进行. 通过实验验证, 以多线程加载核函数的机制大幅提升了 Multi-GPU 服务器的图计算性能, 并得到更好的系统可扩展性.

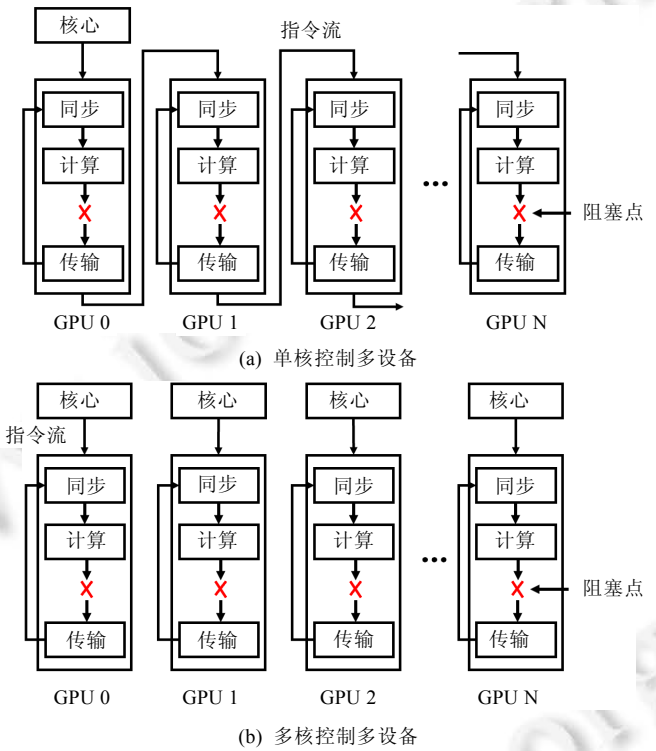


图 9 单核与多核控制多设备时 CPU 指令流顺序对比

4.3 细粒度化顶点更新值同步

计算引擎主要负责顶点在 GPU 中的计算过程. 在计算过程中, 为了尽可能地减少 GPU 间通信量, 采用仅通信更新的边界顶点. 如何确定系统中更新的边界顶点分为以下两步.

- 在图划分过程中, 首先标记边界顶点, 边界顶点的定义是: 一条边的两端顶点不在同一个分区中, 则两点属于边界顶点. 遍历所有边集, 判断具体边的源顶点和目的顶点是否跨 GPU 设备, 即边的源顶点和目的顶点不在同一个分区(GPU 设备), 则标记为边界顶点;
- 在计算过程中, 对更新顶点进行标记. 以 CUDA Warp-Level 原语支持的集体操作为基础, 通过线程来计算每一个顶点的值, 32 个线程为一个 warp, 并以一个 warp 作为一个单元. ChattyGraph 中涉及的原语如下:
  - __any_sync: 当 warp 返回值中任意一个非零, 则返回非零;
  - __ballot_sync: 当 warp 中第 n 个线程处于活跃状态, 则返回非零;
  - __popc: 统计 warp 中活跃线程数量.

首先, __any_sync 确定 32 个线程中更新了哪几个线程, __ballot_sync 将其转换为 32 位 01 序列. 在序列化的支持下, __popc 可以统计一个 warp 中更新顶点的数量并记录其位置, 最终将标记了更新位和边界位的顶点

的位置与其值填充进输出缓冲区中.

5 评估实验和结果分析

5.1 实验环境及数据

- 实验平台

在配置 8 块互联互通的 NVIDIA V100 GPU 协处理器的 NVIDIA DGX 服务器上进行验证实验. 每个 GPU 有 5 120 个流式多处理器、32 GB 全局内存和 768 KB 二级缓存. 该系统还包括两个 64 核 Intel(R) Xeon(R) CPU Platinum 8163(2.5 Hz)和 256 GB DDR4 主内存, 运行 Ubuntu 16.04(内核 4.15.0)和 CUDA 10.0. 整体实验平台 NVIDIA DGX 服务器的连接性如图 3(a)所示. 每个 GPU 支持 4 个 NVLink 链路插槽, 其中, 两个链路通过 NVLink-V2 连接, 另外两个链路通过 NVLink-V1 连接.

- 图数据集

表 3 给出了标准的开放图数据集合, 以用于本文系统性能和可扩展性验证. 这类图数据均采自真实世界的应用场景, 本文通过这类斯坦福大学大规模网络数据集的集合, 以展现各种大小和功能的图基准验证. 我们选择了 4 个公开可用的真实世界图数据, 这些图数据也广泛应用于各类 GPU 图数据并行计算系统的性能和可伸缩性评估标准.

表 3 数据集特征

数据集	顶点数	边数	大小(GB)
soc-LiveJournal1[31]	4 847 584	68 993 773	1
USA-road[32]	23 947 360	58 333 344	1.2
osm-eur[33]	173 789 216	347 997 111	7.08
Twitter[34]	61 578 432	1 468 365 182	24.37

- 基准测试集

为了进行更为全面和公平的比较, 本文选择了 3 种标准的图计算和遍历算法进行系统评估, 包括单源最短路径(SSSP)、广度优先搜索(BFS)和 PageRank(PR). 我们在性能、可扩展性等指标中展示了这 3 种算法的运行时间(每类算法运行 10 次取平均值).

- 对比系统

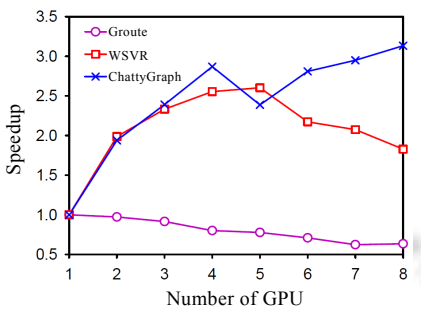
本文的对比系统选用当前最先进的 GPU 图数据并行计算系统(WS-VR, Groute)来进行评估, 并分别报告各自系统的整体执行性能、分阶段执行性能、系统可扩展性、运行时间、冗余工作负载减少以及延迟度量. 上述两种系统与 ChattyGraph 都使用相同的平台(NVIDIA DGX)进行评估, 并提供对比数据.

5.2 与先进图数据并行计算系统性能对比

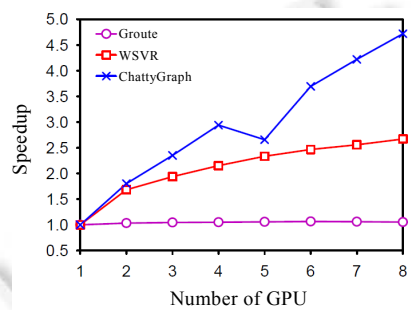
首先, 我们对 ChattyGraph 和当前 Multi-GPU 图数据并行计算系统(Groute 和 WS-VR)进行性能比较. 通过对比遍历和计算应用(SSSP 和 PageRank)的性能结果, 从表 4 的整体执行效率可见, ChattyGraph 在所有应用程序中大部分显著优于其他系统. 其中, 对于 USA-road 图中的 SSSP 应用, ChattyGraph 在 8GPU 下执行的整体时间(4 045.44 s)显著低于另外两个系统 WS-VR 和 Groute 至 3.5× (14 711.8 s)和 3.9× (15 825.89 s). 进一步, 从图 10 中, 我们评估对比了 ChattyGraph 和 Groute, WS-VR 在多块 GPU 引入时的计算性能的可扩展性. 整体而言,从使用 1GPU 到 8GPU, ChattyGraph 可以比这两个系统性能更高、更具可扩展性; 与单 GPU 相比, ChattyGraph 在 8 个 GPU 上实现了 3×的加速. 与 Groute 相比, ChattyGraph 在 1GPU 和 8GPU 配置下执行效率高于 3×. 这是因为 ChattyGraph 用了更为高效的通信运行时, 通过对不同粒度的数据集采用性能各异的链路和操作, 显著地优化了多 GPU 之间协同的数据通信; 相对而言, Groute 则着重关注异步通信驱动的多 GPU 协作, 忽略了当前多 GPU 之间的高速互联通道. 与 WS-VR 相比, ChattyGraph 也实现了更好的性能, 仅在 5GPU 配置性能稍弱. 这是由于当前硬件拓扑结构无法形成环状拓扑, 有额外的通信开销.

表 4 先进图计算系统对比

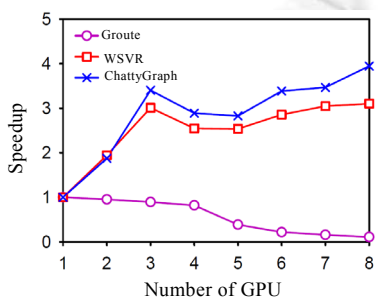
数据集	算法	GPU 数量	WSVR	Groute	ChattyG
USA-road	SSSP	1	10 415.3	33 261.37	10 406.8
		2	5 814.39	22 565.26	5 478.73
		3	5 952.88	18 749.16	4 359.85
		4	6 234.02	18 158.48	3 820.02
		5	8 829.48	16 751.32	9 012.92
		6	10 406.9	18 212.56	4 566.57
		7	13 029.6	16 384.66	4 418.74
		8	14 711.8	15 825.89	4 045.44
USA-road	PageRank	1	39.693	428.875	39.684
		2	20.016	214.366	24.08
		3	20.515	138.486	23.662
		4	23.178	106.844	23.872
		5	34.765	86.882	27.226
		6	32.348	78.635	27.135
		7	35.576	72.392	28.691
		8	42.68	70.535	31.554
Twitter	PageRank	1	19 751.5	13 762.296	18 051.7
		2	18 804.2	9 772.269	17 507.7
		3	8 632.92	8 104.444	8 615.4
		4	8 596.6	7 970.716	8 488.03
		5	7 864.87	8 954.205	7 766.54
		6	7 800.82	8 021.378	7 488.87
		7	7 830.87	6 999.526	7 669.34
		8	6 470.85	7 705.487	5 954.21
LiveJournal	PageRank	1	781.653	230.665	781.328
		2	402.4	242.332	417.014
		3	259.677	258.378	229.693
		4	306.863	281.903	270.467
		5	308.234	598.843	276.369
		6	273.666	1 068.281	230.799
		7	256.483	1 478.873	225.335
		8	251.955	2 144.472	198.01



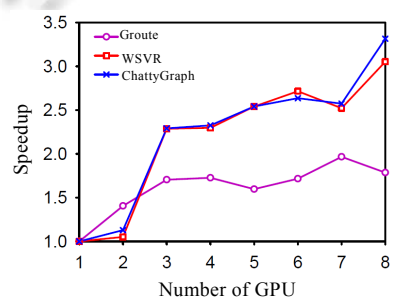
(a) 在 USA-road 运行 BFS



(b) 在 osm-eur 运行 SSSP



(c) LiveJournal 运行 PageRank



(d) Twitter 运行 PageRank

图 10 ChattyGraph 与先进图计算系统扩展性对比

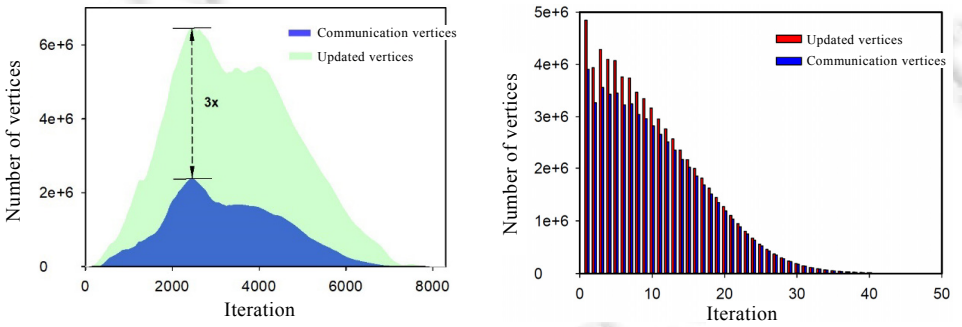
此外, 需要额外注意的是: 随着 GPU 设备数量的增加, ChattyGraph 实现了比其他两个系统更高的加速比(在 8GPU 下平均 3.8×). 从 SSSP 应用程序上的这些结果可以看出: ChattyGraph 在该图遍历算法中实现了更好的性能和更高的可扩展性, 执行加速比分别对比 Groute 和 WSVR 达到了 4.48×和 1.8×的提升.

从对这 3 个数据集的 PageRank 标准测试来看, ChattyGraph 也表现出良好的性能和可扩展性. 尽管 ChattyGraph 对 Twitter 图数据的效率不如 Groute 和 WS-VR, 但在扩展到多个 GPU 设备时, ChattyGraph 仍然实现了更好的可扩展性. 例如, 图 10(c)、图 10(d)展示了 ChattyGraph, Groute 和 WS-VR 在 LiveJournal 和 Twitter 图上执行 PageRank 的可伸缩性比较. ChattyGraph 在 8GPU 上的平均加速比普遍能达到 3×–4×左右. 这是因为 PageRank 应用程序在 GPU 设备之间的通信具有更大的工作负载. 而我们的 ChattyGraph: 1) 改进了不规则结构感知通信; 2) 通过减少冗余工作负载, 最大限度地减少通信开销.

为了进一步评估 ChattyGraph 的可伸缩性, 我们评估了其他算法和数据集上的可伸缩性, 即 BFS 在 USA-road 图和 SSSP 在 osm-eur 图(如图 10(a)、图 10(b)所示)上的运行结果. 评估结果表明: 当 GPU 数量达到 4 个以上时, 其他两个系统(Groute 和 WS-VR)的性能会有明显的下降; 只有 ChattyGraph 在扩展到 6GPU–8GPU 时实现了加速, 相对比 4GPU 配置下具有 1.1×和 1.6×的性能提升.

5.3 通信量优化

进一步, 考虑到具体的策略针对 Multi-GPU 下的设备间通信展开, 本文对 ChattyGraph 的通信量优化进行了深入的分析. 从通信量优化的实验结果(图 11)可见: 在每一轮迭代过程中, ChattyGraph 中的通信开销显著减少, 达到 3×的下降. 通信量对于在多 GPU 平台上的可扩展性至关重要: 通信量越小, 表示 ChattyGraph 系统能够进一步降低设备间数据通信的依赖性, 进而提升整体系统在 Multi-GPU 服务器上的可扩展性.



(a) 在 USA-road 上运行 SSSP (b) 在 USA-road 上运行 PageRank

图 11 更新顶点与通信顶点数量对比

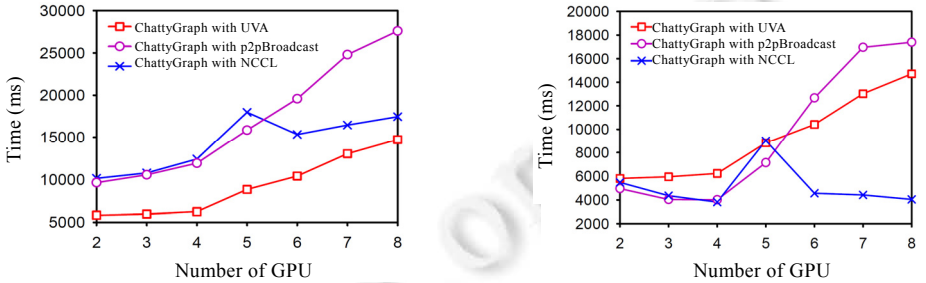
从图 11(a)中的结果可以看出: 与更新顶点总数相比, ChattyGraph 只传递了三分之一的顶点值, 其中: 在 USA-road 图数据运行 SSSP 基准测试中, 也不乏有不需要 GPU 间通信的时刻. 通过引入更新的边界顶点划分传输, 每一轮迭代, ChattyGraph 都显著减少了顶点值的通信量. 同样, 以 USA-road 为例, 运行 PageRank 基准测试进行评估(如图 11(b)所示). 结合这两张图我们可以观察到: 针对不同算法, 在最高值需要通信的顶点数目不同. PageRank 中, 通信顶点占更新顶点总数的百分比(80.6%)远远大于 SSSP 算法(37.4%). 我们可以得出结论:对于遍历算法而言, 采用 ChattyGraph 对数据通信时间上的实质性改进更为显著.

5.4 OpenMP对可扩展性的影响对比

为了比较单核与多核控制多设备对系统可扩展性的影响, 我们分别比较了 Peer-to-Peer Broadcast, UVA Broadcast(WS-VR)和 NCCL Broadcast 在有无 OpenMP 下的图计算时间, 以下实验都以 USA-road 为例运行 SSSP 基准测试.

图 12(a)是采用单核控制多 GPU 编程模式, 由于统计通信量的原因, 需要在计算核函数后进行流同步等待, 导致了多 GPU 在单核下的串行指令顺序, 破坏了多 GPU 的并行性. 而图 12(b)采用 OpenMP 多核控制多

GPU 编程模式, 多 GPU 在不同的流中并行启动执行核函数, Synchronize 过程不会影响其他 GPU 的运行. 综合两张图我们可以观察到: 在未使用 OpenMP 的情况下, P2P 和 NCCL 的性能下降 50%, 同时, 因为指令类串行化的原因, 随着 GPU 设备数量的增加, 图计算时间也会随之增加; 而对于采用 OpenMP 模式的 3 种通信方式, 性能都得到了显著的提升, 尤其是在 2GPU–4GPU 与 5GPU–8GPU 时, NCCL 可以呈现出非常不错的延展性. 4GPU–5GPU 下 NCCL 处理性能下降, 是因为 GPU 跨层的影响.



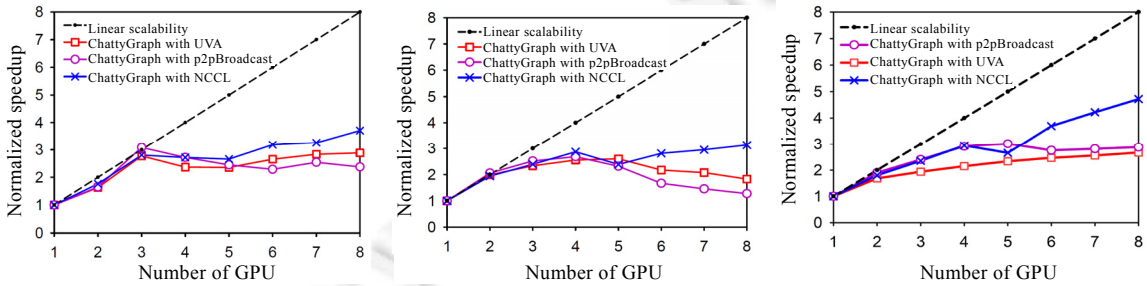
(a) 单核控制多 GPU 在 3 种通信模式下的处理时间 (b) 多核控制多 GPU 在 3 种通信模式下的处理时间

图 12 OpenMP 对扩展性分析

5.5 GPU间多种通信方式对可扩展性影响对比

为了观察将系统 ChattyGraph 的图数据处理过程从一个 GPU 扩展到多个 GPU 的效果, 通过比较 UVA, NCCL 和 p2pBroadcast 这 3 种通信技术, 对 USA-road, eur-osm 和 LiveJournal 这 3 种图数据在 ChattyGraph 中进行评估, 并说明从 1GPU 扩展到 8GPU 时的性能.

从图 13 的结果来看: 从 1GPU 到 4GPU 的加速, ChattyGraph 中采用的 3 种通信方法都达到了最佳的类线性结果, 分别达到了 2.72×, 2.87×, 2.94×; 当对执行配置超过 4 个 GPU 设备时, NCCL 方法的效果优于其他两种方法(UVA 和 p2pBroadcast), 平均提供了 1.6×和 1.9×的数据性能提升. 这是因为 4 个 GPU 通过 NVLink 完全连接, 当扩展到 5 个 GPU 时, GPU5 需要从其他所有 4 个 GPU 请求同步相应的更新顶点值, 这类数据的通信请求将会通过 PCI-E 总线进行操作, 导致性能的一定下降. ChattyGraph 通过使用细粒度的数据链路方法减少了通信开销, 因此, 这 3 种方法在扩展到更多 GPU 时, 都实现了良好的可伸缩性. 即使扩展到 3GPU, 集成这 3 种通信方式的 ChattyGraph 的性能也可以实现接近于最佳线性可伸缩性的性能, 相对比采用单块 GPU 达到 3×性能提升.



(a) LiveJournal 上运行 PageRank (b) USA-road 上运行 BFS (c) eur-osm 上运行 SSSP

图 13 多种通信方式对扩展性分析

6 总 结

在这项工作中, 本文提出了 ChattyGraph, 一种新型的面向多协处理器高性能环境的大规模图数据并行计算系统, 用于在当前 HPC 节点上实现与现代总线链路互联互通的 GPU 设备集成的高可扩展图数据计算. 从目前最先进的 Multi-GPU 图并行计算技术来看, 当前的系统仍然存在以下缺陷: (1) 无法充分利用 GPU 的高通

量连接性; (2) 通常采用过于单一的设备间数据组织和移动策略. ChattyGraph 采用混合通信运行时技术, 以结合最小化通信开销, 从而为 Multi-GPU 图计算并行系统提供了高可扩展性支持, 充分利用现代 GPU 链路技术的高带宽和低时延特性. 在大规模真实图数据和标准测试集上的结果表明: ChattyGraph 相比 WSVR 和 Groute 在性能和可扩展性上均实现了显著改进, 并且随着 GPU 数量的增加和图数据规模的扩大, ChattyGraph 可以更为高效地扩展. 未来的工作将围绕分布式多协处理器的高性能环境展开, 拟充分利用现代网络链路技术, 以进一步提升 ChattyGraph 的计算性能和应用范围.

References:

[1] Ben-Nun T, Sutton M, Pai S, *et al*. Groute: An asynchronous multi-GPU programming model for irregular computations. In: Proc. of the 22nd ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. ACM, 2017. 235−248.

[2] Fu ZS, Personick M, Thompson B. Mapgraph: A high level API for fast development of high performance graph analytics on GPUs. In: Proc. of the Workshop on GRAph Data Management Experiences and Systems. ACM, 2014. 1−6.

[3] Pan YC, Wang YZH, Wu YD, *et al*. Multi-GPU graph analytics. arXiv:1504.04804, 2015.

[4] Khorasani F, Vora K, Gupta R, *et al*. CuSha: Vertex-centric graph processing on GPUs. In: Proc. of the 23rd Int'l Symp. on High-Performance Parallel and Distributed Computing. ACM, 2014. 239−252.

[5] Wu YD, Wang YZH, Pan YC, *et al*. 2015. Performance characterization of high-level programming models for GPU graph analytics. In: Proc. of the IEEE Int'l Symp. on Workload Characterization (IISWC). IEEE, 2015. 66−75.

[6] Ben-Nun T, Levy E, Barak A, *et al*. Memory access patterns: The missing piece of the multi-GPU puzzle. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis (SC 2015). ACM, 2015. 19:1−19:12.

[7] Adar E, Re C. Managing uncertainty in social networks. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 2007, 30(2): 15−22.

[8] Kwak H, Lee C, Park H, *et al*. What is Twitter, a social network or a news media? In: Proc. of the 19th Int'l Conf. on World Wide Web (WWW 2010). New York: ACM, 2010. 591−600.

[9] Shirahata K, Sato H, Matsuoka S. Out-of-Core GPU memory management for MapReduce-based large-scale graph processing. In: Proc. of the 2014 IEEE Int'l Conf. on Cluster Computing (CLUSTER). 2014. 221−229. https://doi.org/10.1109/CLUSTER.2014.6968748

[10] Sengupta D, Song SL, Agarwal K, *et al*. GraphReduce: Processing large-scale graphs on accelerator-based systems. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. ACM, 2015. 28.

[11] Zhong JL, He BS. Medusa: Simplified graph processing on GPUs. IEEE Trans. on Parallel and Distributed Systems, 2014, 25(6): 1543−1552.

[12] Khorasani F, Gupta R, Bhuyan LN. Scalable SIMD-efficient graph processing on GPUs. In: Proc. of the 2015 Int'l Conf. on Parallel Architecture and Compilation (PACT). 2015. 39−50. [doi: 10.1109/PACT.2015.15]

[13] Kyrola A, Blelloch G, Guestrin C. GraphChi: Large-scale graph computation on just a PC. In: Proc. of the 10th USENIX Symp. on Operating Systems Design and Implementation (OSDI 2012). 2012. 31−46.

[14] Hong S, Kim SK, Oguntebi T, *et al*. Accelerating CUDA graph algorithms at maximum warp. In: Proc. of the ACM SIGPLAN Notices. Vol.46. ACM, 2011. 267−276.

[15] Naumov M, Vrielink A, Garland M. Parallel depth-first search for directed acyclic graphs. In: Proc. of the 7th Workshop on Irregular Applications: Architectures and Algorithms. 2017. 1−8.

[16] Liu H, Huang HH. Enterprise: Breadth-first graph traversal on GPUs. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. 2015. 1−12.

[17] Wang TT, Rong CT, Lu W, *et al*. Survey on technologies of distributed graph processing systems. Ruan Jian Xue Bao/Journal of Software, 2018, 29(3): 569−586 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5450.htm [doi: 10.13328/j.cnki.jos.005450]

[18] Zhang CB, Li Y, Jia T. Survey of state-of-the-art fault tolerance for distributed graph processing jobs. Ruan Jian Xue Bao/Journal of Software, 2021, 32(7): 2078−2102 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6269.htm [doi: 10.13328/j.cnki.jos.006269]

[19] Wang YZH, Davidson A, Pan YC, *et al*. Gunrock: A high-performance graph processing library on the GPU. In: Proc. of the 21st ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming. ACM, 2016. 11.

[20] Li A, Song SL, Chen JY, *et al*. Tartan: Evaluating modern GPU interconnect via a multi-GPU benchmark suite. In: Proc. of the 2018 IEEE Int'l Symp. on Workload Characterization (IISWC). IEEE, 2018. 191−202.

[21] Tran HN, Kim JJ, He BS. Fast subgraph matching on large graphs using graphics processors. In: Proc. of the Int'l Conf. on Database Systems for Advanced Applications. Springer, 2015. 299−315.

[22] Busato F, Bombieri N. BFS-4K: An efficient implementation of BFS for kepler GPU architectures. IEEE Trans. on Parallel and Distributed Systems, 2015, 26(7): 1826−1838.
[23] Merrill D, Garland M, Grimshaw A. High-performance and scalable GPU graph traversal. ACM Trans. on Parallel Computing, 2015, 1(2): 14.
[24] Djidjev H, Chapuis G, Andonov R, *et al*. All-pairs shortest path algorithms for planar graph for GPU- accelerated clusters. Journal of Parallel and Distributed Computing, 2015, 85: 91−103.
[25] Martinasso M, Kwasniewski G, Alam SR, *et al*. A PCIe congestion-aware performance model for densely populated accelerator servers. In: Proc. of the Int'l Conf. for High Performance Computing, Networking, Storage and Analysis. IEEE, 2016. 63.
[26] Merrill D, Garland M, Grimshaw A. Scalable GPU graph traversal. In: Proc. of the ACM SIGPLAN Notices, Vol.47. ACM, 2012. 117−128.
[27] Narayanan D, Harlap A, Phanishayee A, *et al*. Pipedream: Generalized pipeline parallelism for DNN training. In: Proc. of the ACM Symp. on Operating Systems Principles (SOSP). 2019.
[28] Wang GH, Venkataraman S, Phanishayee A, *et al*. Blink: Fast and generic collectives for distributed ML. In: Proc. of the 3rd MLSys Conf., arXiv:1910.04940, 2020.
[29] Wang J, Zhang L, Wang PY, *et al*. Memory system optimization for graph processing: A survey. Scientia Sinica Informationis, 2019, 49(3): 295−313 (in Chinese with English abstract). [doi: 10.1360/N112018-00281]
[30] Wang P, Wang J, Li C, *et al*. Grus: Toward unified-memory-efficient high-performance graph processing on GPU. ACM Trans. on Architecture and Code Optimization (TACO), 2021, 18(2): 1−25.
[31] Stanford large network dataset collection. https://snap.stanford.edu/data/soc-LiveJournal1.html
[32] 9th DIMACS implementation challenge. http://www.dis.uniroma1.it/challenge9/download.shtml
[33] Karlsruhe Institute of Technology. http://i11www.iti.uni-karlsruhe.de/resources/roadgraphs.php
[34] Cha M, Haddadi H, Benevenuto F, *et al*. Measuring user influence in Twitter: The million follower fallacy. In: Proc. of the 4th Int'l AAAI Conf. on Weblogs and Social Media (ICWSM), 2010. 10−17.

附中文参考文献:

[17] 王童童, 荣垂田, 卢卫, 等. 分布式图处理系统技术综述. 软件学报, 2018, 29(3): 569−586. http://www.jos.org.cn/1000-9825/5450.htm [doi: 10.13328/j.cnki.jos.005450]
[18] 张程博, 李影, 贾统. 面向分布式图计算作业的容错技术研究综述. 软件学报, 2021, 32(7): 2078−2102. http://www.jos.org.cn/1000-9825/6269.htm [doi: 10.13328/j.cnki.jos.006269]
[29] 王靖, 张路, 王鹏宇, 等. 面向图计算的内存系统优化技术综述. 中国科学: 信息科学, 2019, 49(3): 295−313. [doi: 10.1360/N112018-00281]

蒋筱斌(1996－), 男, 硕士, 主要研究领域为操作系统, 云原生, 图计算.

熊轶翔(1996－), 男, 硕士, 主要研究领域为操作系统, 云原生, 高性能计算.

张珩(1990－), 男, 博士, CCF 专业会员, 主要研究领域为分布式与并行计算, 大数据处理, 操作系统.

武延军(1979－), 男, 博士, 研究员, 博士生导师, CCF 杰出会员, 主要研究领域为操作系统.

赵琛(1967－), 男, 博士, 研究员, 博士生导师, CCF 高级会员, 主要研究领域为编程语言, 编译技术.