中国科学 信息科学
SCIENTIA SINICA Informationis

面向图计算的内存系统优化技术综述

王靖, 张路, 王鹏宇, 徐嘉鸿, 李超, 朱浩瑾, 钱学海 and 过敏意

Citation: 中国科学: 信息科学 **49**, 295 (2019); doi: 10.1360/N112018-00281

View online: http://engine.scichina.com/doi/10.1360/N112018-00281

View Table of Contents: http://engine.scichina.com/publisher/scp/journal/SSI/49/3

Published by the 《中国科学》杂志社

中国科学: 信息科学 2019 年 第 49 卷 第 3 期: 295–313
SCIENTIA SINICA Informationis
《中国科学》杂志社
SCIENCE CHINA PRESS
面向智能应用的定制计算加速器技术专题 • 评述

面向图计算的内存系统优化技术综述

王靖, 张路, 王鹏宇, 徐嘉鸿, 李超*, 朱浩瑾, 钱学海, 过敏意

上海交通大学计算机科学与工程系, 上海 200240
* 通信作者. E-mail: lichao@cs.sjtu.edu.cn

收稿日期: 2018–10–18; 接受日期: 2018–12–12; 网络出版日期: 2019–03–18

国家重点研发计划 (批准号: 2018YFB1003500) 资助项目

摘要 图 (graph) 是一种以顶点和边构成的包含多种信息的复杂数据结构. 图计算 (graph processing) 要求我们将现实条件中的关系属性抽象为图数据结构并进行复杂计算. 由于 CPU 性能提升遇到瓶颈, 人们尝试了多种协处理器或专用加速器, 致力于提高运行速度并节省能耗. 由于图计算具有数据依赖性强、访存 – 计算比高的特点, 提高图计算访存效率是改善系统性能的关键. 尤其是随着图数据规模的扩大, 高效的内存管理优化对异构图计算性能的提高显得尤为重要. 本文将介绍异构架构图计算中内存系统的管理及优化方法, 归纳目前能够提高访存效率的图数据格式; 分析图计算专用加速器 GPU, FPGA, ASIC, PIM 等的架构特点与内存方面的优化工作; 概括国内相关研究进展; 同时总结图计算在内存方面的机遇与挑战.

关键词 图计算, 专用加速器, 内存管理, 内存系统架构, 访存优化

1 引言

图 (graph) 不同于图形 (graphic), 是一种包含了一组顶点以及连接他们的边的数据结构. 在现实世界中, 很多数据可以用图结构表示, 并且可以运用到很多的应用中去, 比如数据科学、社交网络、机器学习、基因组学等. 图数据具有灵活性以及抽象性, 可以很好地表述这些实体之间的关联关系.

图计算 (graph processing 或 graph computing) 是研究客观世界当中事物与事物之间的关系并对其进行完整地刻画、计算和分析的一门处理图的技术, 是大数据关联属性的最佳表达方式. 随着图数据规模的爆炸式增长和复杂计算需求的不断增加, 图计算系统、大规模图处理算法和图计算优化等技术日益受到关注. 当前大数据、云计算等技术正处于快速发展时期, 利用大数据的机器学习和深度学习算法, 都依赖于图计算. 但是图数据本身的关联性、幂律性、迭代运算等特点使得图计算的性能一直存在瓶颈, 加速图计算的运算效率是一个有趣但富有挑战的工作.

引用格式: 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: 295–313, doi: 10.1360/N112018-00281
Wang J, Zhang L, Wang P Y, et al. Memory system optimization for graph processing: a survey (in Chinese). Sci Sin Inform, 2019, 49: 295–313, doi: 10.1360/N112018-00281

图计算本身是一个访存计算比很高的应用, 在图计算的计算过程中, 访存开销占据了很大的比例. 为了提高图计算的性能, 好的内存架构和管理策略必不可少. 目前人们已经从不同的层面来改善图计算过程中的访存模式来提高图计算的性能, 包括对于图数据结构的处理, 设计新的计算模式, 采用新型内存硬件架构, 设计相关加速器等方面来加速图计算的访存效率, 从而优化图计算的性能并尝试降低能耗.

现有的工作针对不同的架构和平台进行了多方面尝试, 主要的架构平台包括高性能 GPU 架构、基于 PIM (processing in memory) 的架构以及专用图计算加速器 (ASIC 和 FPGA). GPU 的高并行度与高带宽对图计算的运算加速起到了至关重要的作用; 为了减少内存的访问带来的开销, 人们设计了各种数据格式和图算法以提高访存效率. PIM 技术让许多新兴硬件应用到图计算中, 如 HMC (hybrid memory cube), ReRAM (resistive RAM) 等, 实现访存一体化, 减少内存数据的移动传输带来的开销. 突破通用处理器的限制, 图计算专用加速器包括基于 FPGA 的设计以及专用芯片 (ASIC) 的设计, 尝试自主设计运算流水线、访存模式等优化策略来加速图计算运算效率以及减少能耗.

本文针对图计算加速环境下不同架构平台下对于内存管理所作出的努力以及其他方面的优化方案进行综述.

2 图计算基本介绍

本节首先介绍图计算的基本概念与图算法的特点分类, 然后总结图计算的计算特征和常见的编程模型.

2.1 图的基本定义

图是一种由顶点 (vertices) 和连接顶点的边 (edges) 构成的数据结构, $G = (V, E)$, 其中 V 表示顶点集合, E 表示边的集合. $e = (v_i, v_j)$ 表示从顶点 v_i 到 v_j 的一条有向边. 同时, 每一个顶点与每一条边都有属于自己的属性值. 不同的领域属性值可代表不同的含义. 如社交网络中, 顶点的属性值为个人的热度, 而边的属性值则表示为有关联的两人之间的紧密程度. 现实中的自然图存在以下的特性.

(1) 稀疏性. 顶点的平均度数很小, 数据分布比较分散, 图数据的稀疏性会导致较差的局部性数据访问和大量的随机访问.

(2) 幂律性. 幂律性是自然图的一个很常见的特征, 图数据中一小部分顶点关联着绝大部分的边. 而大部分的顶点只关联着少量的边. 图数据的幂律性会导致很严重的负载不均衡.

(3) 小世界性. 图中的任意两个顶点都可以通过很少的有限个中间节点相互访问. 这种小世界性给大图的分割与并行操作带来了很大的挑战.

2.2 图计算相关算法

对于常用的图算法进行总结与归纳, 可以将图算法分为以下两类, 如图 1 所示.

(1) 遍历为中心的图算法. 该类算法通常需要以特定方式从特定顶点遍历图, 它们的计算内存访问率相对较低, 存在大量随机访问, 这大大浪费了存储器带宽. 比较经典的算法如广度优先遍历 (breadth-first traversal, BFS)、单源最短路径 (single source shortest path, SSSP)、中介中心度 (betweenness centrality, BC) 等都属于这种算法.

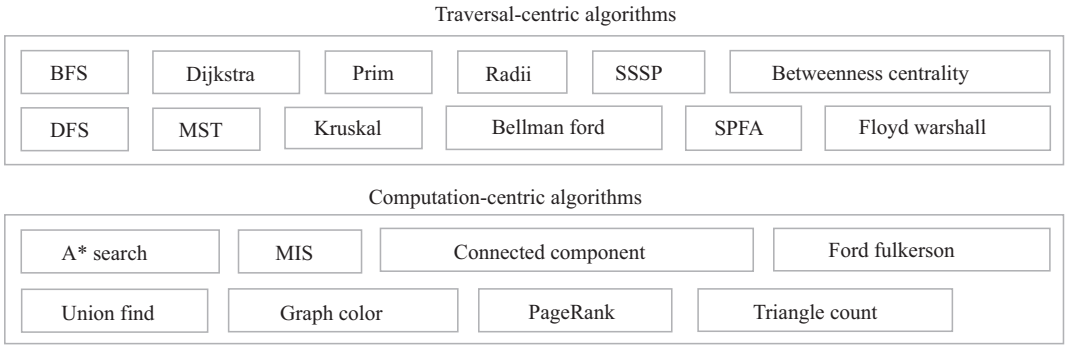

图 1 遍历或计算为中心的图算法归类

Figure 1 Traversal-centric and computation-centric graph algorithms

(2) 计算为中心的图算法. 该类算法在一个迭代周期中有大量的运算进行, 在每一次迭代中所有的顶点都会参与. 相比于遍历为中心的图算法, 这类算法的数据局部性较好, 但是在运算过程中会出现大量的浮点运算, 而且每轮迭代所有顶点的参与会增加算法运算过程中对内存的访问频率. 经典的网页排序 (pagerank)、连通分量 (connected component, CC)、三角形计数 (triangle count, TC) 等都属于这种算法.

2.3 图计算的特点

本小节介绍图计算相较与其他数据运算的特点, 从而分析内存优化和管理在图计算中占据着的重要地位.

(1) 数据访问密集. 前面提到图计算是一种访存 – 计算比相对较高的应用, 在图计算运算过程中的大部分操作都与数据的访问有关, 访存的性能直接影响着图计算的整体性能.

(2) 数据局部性差. 图数据中的某个顶点与其他顶点的相连是随机的, 这导致在图计算过程中对数据访问的随机性, 从而造成糟糕的数据访问局部性.

(3) 数据依赖性高. 数据依赖性是由图中顶点连接属性关联的性质引起的. 严重的数据依赖使得图在运算过程中并行执行困难, 当多个顶点试图同时更改同一顶点数据时, 会出现严重的数据冲突.

2.4 图计算常见编程模型

图在顶点之间有复杂的数据依赖性. 编程模型可以通过探索围绕顶点或边的计算模式尽可能地解耦这些相关的依赖关系. 现有的用于图计算编程模型基本上可以分为两个类别: 以顶点为中心的模型和以边为中心的模型. 一些图计算加速器还尝试了二者的融合.

以顶点为中心的模型: 使用此模型的图算法通过 “Think as a vertice [1]” 的方式处理图. 它独立处理每个顶点的任务, 并通过边对临接顶点进行计算和数据传输. 由于每个顶点都被单独处理, 因此可以通过同时调度保证高并行度. 以顶点为中心的计算模型可以很方便地表示各种图算法并且可以实现顶点的高并行, 所以这种编程模型已被广泛用于许多图加速器 [2~5]. 然而, 在以顶点为中心的模型中会存在大量的随机访问, 从而导致潜在且昂贵的内存访问开销.

以边为中心的模型: X-Stream [6] 是第一个使用以边为中心的模型即 GAS 模型来处理每条边, 主要通过 3 个步骤迭代执行: (1) 收集其源顶点的信息 (gather); (2) 更新 (apply); (3) 将更新后的值发送到各目标顶点 (scatter). 该编程模型可以实现对边的顺序访存. 现有的图加速器通常使用以边为中心

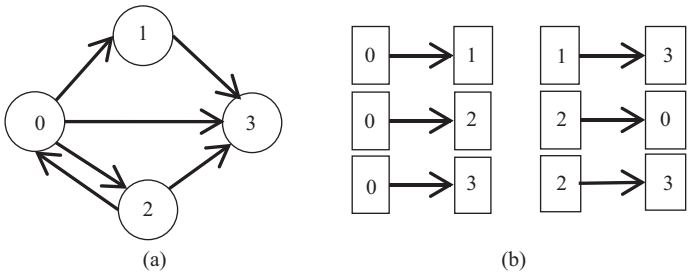

图 2 图拓扑与 Edge Array 表示

Figure 2 Graph topology and edge array representation. (a) Graph topology; (b) edge array

的模型来提高其有限内存带宽的利用率 [7,8]. 但与以顶点为中心的编程模型相比, 以边为中心的模型缺乏灵活的调度能力, 同时该模型还可能导致对顶点的大量随机访问. 因此, 通常需要额外的优化, 例如细粒度分区和定制的顶点更新策略 [7,9].

混合编程模型: 混合编程模型是指在以顶点和边为中心的编程模型之间进行切换, 利用两种模型各自的优点来加速图计算的运行 [10]. 当活跃顶点比率相对较高时, 要求顶点为中心的模型负责. 相反, 活跃顶点比率相对较低时, 使用边为中心的模型应对. 模型切换的决定是根据当前系统激活顶点的比例来确定的, 而这个阈值通常和内存带宽有关系. 针对不同类型的算法还可以灵活地设计新的编程模型, 如 PCPM [11] 以分割为中心处理特定 PageRank, 综合考虑了以边为中心和以顶点为中心的优势以及算法特点, 与 GAS 编程模型相结合, 显著减少了通信和随机 DRAM 访问的开销.

本节主要介绍了图计算中数据、算法以及计算的特征及访存的挑战. 面对这些特征, 接下来将从以下几个方面来探讨图计算中的内存管理与优化技术: 第 3 节介绍常见的图数据表示方法和分割方式, 从数据的组织形式上提高访存效率; 第 4~6 节从 3 种不同的架构 (GPU 架构、基于 PIM 的计算架构以及图专用加速器 FPGA、ASIC 架构) 来探讨提高图计算性能的内存管理方案和优化方法.

3 基于访存优化的图数据格式

自然图数据的稀疏性、幂律性以及小世界性给图计算的访存、图数据分割, 以及计算负载均衡带来很大的挑战, 因而在进行图计算运行任务之前, 我们常对自然图数据进行预处理. 设计相应的存储格式、重排序机制以及分块策略可以提高图计算访存效率.

3.1 图数据表示方法

结合不同计算平台的存储特性、访存特性以及数据局部性, 设计适当的图数据表示方式, 能够细粒度地帮助图计算性能提升. 本小节主要介绍常见图数据结构的访存特点. 图数据的基本表示主要为两种形式: Edge Array 和 Adjacent List.

Edge Array 是图数据表示的默认并且最简单原始的方式, 并且被很多的系统使用. 图数据存储为一个数组, 其中包含与每条边的源顶点和目标顶点相对应的整数对. 扩展版的 Edge Array 通过两个数组来存储图数据, 一个数组存储顶点的属性值, 另一个数组用来存储边以及边的属性, 即 Edge List 用于存储边相关的数据. 图 2 展示了一个图结构的实例以及其对应的 Edge Array 的表示方法. 另外, Edge Array 是图数据的一种基本的表示方法, 不同的系统根据系统的需求, 对 Edge Array 进行分块、排序等操作以适应平台的需求. 很多系统 [2,6,7,9,10,12,13] 都是采用的 Edge Array 作为图的最基本表

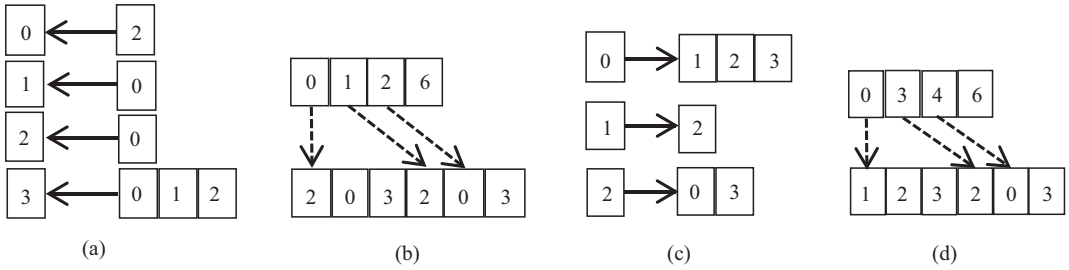

图 3 图的邻接表表示方法

Figure 3 Adjacent list representation. (a) Adjacent list (incoming edge); (b) compressed sparse column; (c) adjacent list (outgoing edge); (d) compressed sparse row

示方法, 然后根据不同的系统对 Edge Array 进行重新排序、分割等构造出适应系统和存储特性的存储模式. 采用 Edge Array 这种表示方法可以很方便地从内存中顺序读取边, 对于边的读取有很好的访存局部性, 在以边为中心的计算模型中有很好的性能, 如 X-Stream [6] 采用以边为中心的计算模型, 在运算过程对边进行 Scatter-Gather 操作, 通过 Edge Array 的表示方法, 可以顺序读取边数据内存, 提高图数据的访存效率.

Adjacent List 存储每个顶点对应的边的数组. 每个顶点指向一个数组, 该数组包含其入边的所有源顶点 (图 3(a)), 或者是所有出边的目标顶点 (图 3(c)). 在图计算系统中, 对于图数据的邻接表表示一般构造为图 3(b) 与 (d) 所对应的压缩稀疏列 (compressed sparse column, CSC) 与压缩稀疏行 (compressed sparse row, CSR). 这种表述方法一般需要包含 3 个数组来表示图数据: 顶点属性值、图数据中边的源点 (CSC) 或者是目标顶点 (CSR), 以及每一个顶点的所有边数组的起始索引. 这种数据布局方式的优点为, 每一个顶点的所有边顺序存储在内存中, 在进行图计算的过程中可以线性顺序访问, 减少了随机访问从而提高内存访问速度. 因此在以顶点为中心的图计算模型中, 邻接表的形式可以增加顺序访存的几率, 提高访存与运算性能. 大多数图计算系统以及加速器 [3, 14~18] 都采用邻接表的格式进行图数据存储.

上文提到图数据的基本存储格式, 不同的图计算系统根据系统的特性以及存储的特性对基础的图数据描述方式进行优化, 以满足图计算系统的访存、计算特性. 下面介绍一些在设计图数据表示中使用的常用优化.

对图数据进行融合: 基本的 CSR 表示方法虽然可以线性顺序地对顶点进行访问, 但是对于顶点属性的访问还是一个随机访存的过程. 有一些图计算系统将一些图数据的信息进行组合 (如顶点索引、属性值等), 可以极大地提高数据的局部性, 降低对内存的随机访问. GraphOps [17] 增加了一个数据结构, 按照 CSR 中顶点的存放顺序来存储顶点的属性值. CyGraph [18] 修改原始的 CSR 的表示方法, 将顶点状态和相邻顶点的信息存储在数组的一个元素中. 该方法显著提高了存储器访问效率.

对图数据进行压缩: 由于加速器的片上内存空间有限, 为了更多地在片上内存中存储图数据, 提高数据访问速度, 一些图计算加速器设计对图数据进行压缩以缩小存储空间. GraphH [19] 在顶点的个数小于最大的顶点索引时, 对图数据的索引进行重定义, 从而消除空白索引来缩小一定的数据存储空间. Graphicionado [2] 和 ForeGraph [7] 采用粗粒度多级索引的方式来缩小图数据的存储规模.

对图数据进行排序: 对图数据进行排序是为了更改图数据在内存中的存储顺序, 使得数据具有更好的访问局部性, 提高访存效率. 基于排序的优化一般是针对 Edge Array 的布局格式进行的优化方案. 将图数据中的 Edge Array 按照源点或者是目标节点的顺序进行排序, 可以提高内存访问的局部性, 因

为邻近节点的数据可能已经被预取缓存, 这将极大地提高访存效率. 在图计算过程中, 读取源点的属性, 并相应地更新目标顶点属性. 如果源点经过排序访问, 则可以减少读取开销. 类似地, 如果目标节点经过排序, 则写入过程可以更高效. 当然也有一种混合的方式, 它通过同时对边的源点和目标节点进行排序, 构成一种 Grid 的形式, 读和写都相对高效 [2, 4, 7].

3.2 大规模图数据分割中的内存管理

对大规模图数据进行分割主要基于以下两种情况. 一种情况为存储限制. 系统快速内存 (加速器中的片上内存以及服务器中的内存) 虽然速度很快但大小有限, 不能一次性加载全部的图数据, 因此需要二级存储 (加速器中的内存以及服务器中的磁盘等) 来存储图数据, 此时需要将图数据分割成可以加载到快速存储中的子图, 计算单元可以通过多次加载子图的方式实现对大规模图数据的处理. 另一种需要考虑的情况是计算并行. 系统有很大的带宽并且存在多个计算单元, 将图数据分割成多个子图分别放到不同的计算单元中进行异步计算, 通过合理利用带宽和计算资源来提高系统的运算速度. 值得一提的是, 研究人员在进行图数据分割时, 通常都会综合考虑两种情况以得到更优的效果.

单机子图分割: GraphChi [20] 是第一个提出使用单机进行大规模图计算的系统, 由于内存空间有限, GraphChi 采用了图数据分割的方法, 将大规模图数据分割成多个大小可以载入内存的子图, 称为 Shard. 以边的目标节点为操作标准, 该系统将图数据分割成多个不相交的片段, 对所有目标节点位于该片段的边按照其源点进行排序, 从而形成不同的 Shard. 配合 GraphChi 设计的并行滑动窗口 (parallel sliding windows, PSW) 算法, 该分割方式可以减少对二级存储的非线性访问, 而大大增加了数据的访问速度和运算速度.

面向目标节点的图分割: 很多系统 [2, 4, 19, 21] 采用面向目标节点的分割方法, 分区方法通常在每个分区中具有不相交的目标顶点. 所有传入边都与分区的目标顶点相关联, 构成子图参与运算. 由于边的目标节点是不相交的, 因此每一个分块对于顶点的更新都是相互独立的, 这样就可以避免数据冲突到来的开销. Graphicionado [2] 采用这种分区方法来确保每个分区都可以连接到暂存器内存, 充分利用其低延迟高带宽特性. GraphP [21] 也采用该种方式, 但 GraphP 的设计是为了减少不同加速器上的分区之间的通信, 从而可以改善 HMC 立方体之间的通信, 加速图计算的性能.

面向源节点的图分割: 有些系统 [10, 22] 会根据实际情况采用面向源点的分割方法, 分区方法通常在每个分区中具有不相交的源顶点. 所有传出边都与分区的源顶点相关联. 目标顶点将包含在相应的分区中. 由于在每个分区中的源顶点索引通常是连续的, 这就可以确保顺序的内存访问. 使用面向源的分区, 可以方便地在图计算过程中确定需要更新的顶点属性的分区.

面向网格的图分割: GridGraph [23] 为了降低 GraphChi 对源点排序的预处理开销以及较多的磁盘写入操作而设计了一种 Grid 的分割模式. 将图数据的顶点分割成为 P 个不相交的子块, 然后将边的源点所在的子块作为行, 边的目标顶点所在的子块作为列, 将图数据中的所有的边放到 $P \times P$ 的网格 (grid) 中. 该种分割方式可以很高效地减少磁盘的写入次数. 当系统进行运算时, 读取源节点 – 计算 – 更新目标节点. 由于网格分块中的每一列图数据边的目标节点都是相同的, 因此目标节点已经被预取到缓存中, 所以采用列的方向从上到下进行计算, 可以利用缓存特性极大地减少磁盘的写入操作, 从而提高运算效率. Fore-Graph [7] 使用这种方法来充分利用 FPGA 有限的片上存储器. GraphR [9] 也采用这种分割方式, 利用 ReRAM 的存储特性来设计高性能低功耗的图计算加速器.

4 GPU 异构环境下的图计算内存管理

4.1 GPU 架构特点

GPU 采用单指令多数据 (single instruction multiple data, SIMD) 架构, 通过大规模并行获得高性能. 在 GPU 中, 大多数芯片区域由算术逻辑单元使用, 而一小部分区域用于控制单元和高速缓存. 尽管 GPU 可以通过提供高度并行性来提高程序运行的性能, 但是针对图计算却存在一定的挑战. 图计算通常表现为不规则的数据访问模式, 因此程序可能无法使用 GPU 的并行运算提供的峰值性能.

GPU 虽然有很高的内存访问带宽, 但是其内存空间大小非常有限. 并且在 CPU 架构下, 可以使用二级存储来协助大规模图数据的计算, 但是在 GPU 中, GPU 内存与主机 DRAM 之间通常使用 PCIe 相连, 其数据传输速度较慢. GPU 通常配备高速但容量很小的片上共享内存, 可用于缓存频繁访问的数据, 以减少访问设备上全局内存的需要. 但是, 如果多个线程同时访问共享内存中的不同数据, 则可能会导致内存 Bank 的冲突, 从而限制了程序运行的并行度.

设计高效内存管理策略也是 GPU 用于图计算加速需要考虑的关键问题. GPU 中的线程运行是以 Wrap 为单位, 因此程序在运行过程中通常以块来进行内存访问. 如果由 Warp 发出的对全局存储器的访问在一个或几个存储器访问单元内合并并且对齐, 则可以显著提高存储器带宽的利用率.

数据表示和分割方式也是解决大规模图计算加速的重要方面. 当 GPU 的全局内存不能适应大规模的图数据时, 使用 Out-of-Core 的形式进行图处理是另一个 GPU 图加速需要考虑的地方. 采用前面提到的图分割方式, 将大规模图数据分割成多个较小的子图, 以及使用合适的数据表示方法, 在需要时将数据交换进 GPU 内存, 是解决此问题的潜在解决方案. 因此在 GPU 中也需要考虑合适的数据表示方法和分割方式来优化内存的使用.

4.2 GPU 上的数据布局优化

合并 Warp 线程访问: 在传统的基于 CPU 的图形处理算法和系统中, 人们常设计数据布局以实现连续的存储器访问以增强 TLB 和高速缓存命中率. 但是在 GPU 中, 所有 GPU 处理器都共享一个全局内存, 对于每一次的内存访问, 向多个线程同时提供数据可以提高访存的性能. GPU 中的线程以组 (CUDA 中的 Wrap) 的形式执行. 如果连续线程正在访问连续的内存地址, 则 Warp 线程对全局内存的访问将合并为单次内存访问. 这样就可以极大地提高内存访问效率.

串联窗口对应线程块: 由于图计算的不规则性, 它的不对称的计算过程与 GPU 对称的硬件架构不匹配, 在 GPU 上直接应用传统的图计算方法就会遇到严重的 GPU 利用率不足的问题. 一些常见的 GPU 图加速系统试图通过设计更加紧密和更加规则的图数据布局来优化内存访问. 大部分的 GPU 图加速系统如 Tigr [24], Medusa [25], MapGraph [26], Frog [27] 等都是前面提到的 CSR 的布局方式, 这种存储模式可以压缩图数据占据的内存空间大小. 但是使用 CSR 会在运算过程中对顶点属性的访问产生很大的随机性, 造成了大量的随机访问. 受 GraphChi [20] 的启发, CuSha [28] 实现了 G-Shard. G-Shard 的构造方法与 Graphchi [20] 中的 Shard 一样. 同时 G-Shard 提出串联窗口 (concatenated window), 即在一次运算中将与当前窗口相关的边都存储下来, 这样每一个线程就可以通过这个串联窗口的数据来访问每一个顶点. 在 CuSha 中, 每个 G-Shard 对应一个线程块, 这样可以保证更好的数据局部性. 另外, G-Shards 彼此不相交, 不同 G-Shard 上的计算可以异步执行, 这契合了 GPU 高效并行的特点.

使用 Page 避免 Warp 分歧: G-Shard 大小的不确定性会到导致 Wrap 分歧的问题. 与高速缓存概念相似, 我们使用同样大小的 Page 来存储, 从而仅需要关注对 Page 的管理. 图计算系统 GTS [29],

GStream [30] 使用 Slotted Page 的形式来存储图, 将图分割成一系列的 Slotted Pages. 对于小型的邻接表, 可以直接放入 Slotted Pages, 或者是一个 Slotted Page 可以存放多个顶点的临接表. 对于高维度的顶点需要多个 Page 来存储该邻接表, 其中有一个 Page 用来存放这些 Page 的信息. 由于 Page 的大小都是固定的, 所以可以避免随机大小的 G-shard 带来的 Wrap 分歧等问题.

4.3 GPU 中 Out-of-Core 图计算访存优化

边排序与内存合并: 为了处理无法加载到内存中的大规模图数据, GraphReduce [31] 将图数据分割大小近似的子图, 并根据源的顶点对子图中的边进行排序, 以匹配 GPU 中的内存访问模式. 为了利用 GPU 内存合并方法将未访问的数据预取到内存中进行顺序访问, GraphReduce 采用统一虚拟寻址 (unified virtual address, UVA) 的方式来分配内存空间, 并使用直接内存访问 (directed memory access) 技术通过 PCIe 直接对内存进行操作. 这些方法使访存顺序执行, 并且可以通过预取的方式将 GPU 计算和数据传输进行重叠, 提高运算效率.

虚拟分割: 图数据的不规则内存访问会导致 GPU 并行计算的不均衡性, 从而限制了 GPU 大规模并行带来的优势. 图数据的幂律性是导致访问不规则的主要原因, 为了提高内存的规则访问, Tigr [24] 采用了一种虚拟节点的方式对幂律图数据进行转换. 它定义维度的界限值, 当顶点的维度高于界限值就将该顶点进行分割, 分割成为多个维度均衡的虚拟顶点参与运算. 通过这种虚拟分割的方式, 既可以保证程序运行的正确性, 也可以提供图数据的规则访问, 充分发挥 GPU 并行计算的性能.

异步数据传输: GTS [29] 只使用 GPU 处理图, 没有 CPU 的参与. 该系统使用 CUDA 流方法, 将未访问的图数据传到 GPU 硬件内存上而将已经访问的数据直接换到磁盘上. 数据分为两种, 被标记为属性类数据 (顶点属性和边的权重, 在计算中不断更新的数据) 和拓扑类数据 (基本结构数据), 后将属性类数据复制存储在 GPU 内存中, 而对拓扑类数据使用流方法从主机内存中拷到 GPU 硬件内存中. GTS 采用异步 GPU 流的方式来执行数据的异步传输. 这种方式重叠了主存到 GPU 内存的访问等待时间, 减少了计算等待时间. 通过重叠数据传输的方式来提高 GPU 利用率.

4.4 GPU 进行图计算其他方面的努力

为了可以更好地在 GPU 上进行图计算的加速, 除了内存管理方面做的努力外, 研究者也在其他的方面进行了大量的工作. 这里做一个简单的总结.

动态调度与双相分解: 图数据的不规则性会严重的影响 GPU 并行运算效率. 除了对数据本身进行预处理之外, 在图计算进行过程中, GPU 也会采取一定的调度策略来提高性能. MapGraph [26] 中使用动态调度 (基于协作线程数组 CTA、基于扫描、基于 Warp) 和双相分解策略来均衡数据的处理, 从而提供较高的运算并行度, 提高运算性能.

粗细粒度调度的结合: GunRock [32] 使用了两种调度策略, 分别是针对单个线程的细粒度调度和针对 Warp 和 CTA 的粗粒度调度. 两种策略在不同的图数据上有不同的效果: 针对单线程的细粒度调度策略在较为均衡的数据分布以及直径较高的图数据上效果比较好, 这种策略可以很好地均衡一个 CTA 中的线程运行; 针对 Wrap 和 CTA 的粗粒度调度策略在幂律图上效果较好.

边为中心与点为中心的切换: GraphReduce [31] 在编程模型上使用 GAS 编程模型, 为了降低图数据之间的通信开销. 该系统在 gather 和 scatter 时使用以边为中心的编程模式, 在 apply 阶段采用以顶点为中心的编程模式.

易用性编程接口: Medusa [25] 和 MapGraph [26] 提供了一系列的 GPU 上进行图计算的编程接口, 可以方便用户进行图计算程序的编写, 提高开发效率. Gunrock [32] 还提供了易用性编程接口. 实现了数据为中心的抽象.

5 基于 PIM 架构下图计算内存管理技术

5.1 硬件特点与构造

PIM 在 20 年前就被提出, 然而当时没有足够大的内存来支持这种复杂的设计, 所以并没有普及. 随着在复杂计算中内存瓶颈日益凸显, PIM 又被重新提上日程. 它的硬件基础即为普通存储器, 如 DRAM, ReRAM, HMC 等, 当然由于存储器类型不同框架设计会有所改变, 性能也会有所不同. 集成电路元件作为 CPU 的协从加速器, 它的主要短板在于随机存储器访问所带来的时延和内存带宽的浪费, 以及存储资源的分散造成的频繁数据移动与通信消耗. 而 PIM 的提出与初步实现给这些资源和能耗浪费问题带来了福音. 由于 3D 堆叠技术的进步如高效 HMC 的娴熟使用, 人们更青睐于使用 PIM+HMC 来优化内存上处理的图计算.

HMC 中的 PIM 操作基本上执行 3 个步骤 [33], 从 DRAM 读取数据, 对逻辑管芯中的数据执行计算, 然后将结果写回到相同的 DRAM 位置. 根据 HMC 2.0 版本, PIM 单元在 HMC 包内以原子方式执行读、修改和写操作. 在 RMW 操作期间相应的 DRAM 组被锁定, 不能为同一组的任何其他存储器请求提供服务. 所有 PIM 操作仅包含一个内存操作数, 操作在立即值和内存操作数上执行.

5.2 存储访问与并行优化

PIM 与 HMC 结合优化: GraphPIM [33] 是一种用于全栈邻近数据计算 NDP (near-data processing) 的图计算框架 [34]. HMC 这一新兴技术使得人们能够在指令级别的存储器内进行 PIM 的分流操作, 克服图计算的性能瓶颈. GraphPIM 对现代图计算工作负载进行分析, 提供硬件和软件机制, 以有效地利用 PIM 和 HMC. 主处理器实现 PIM 分流单元, 它们可以确定当前存储器指令的数据路径. 由于 PIM 区域是不可缓存的, 访问 PIM 内存区的指令请求都将绕过缓存层次结构并直接装载到 HMC 上. 之后他们团队又提出 GraphBIG [35], 一个全面的图计算基准测试套件, 帮助用户观察图计算在硬件架构上的行为, 比如可以看到内存吞吐量等想要关心的问题.

内存带宽优化: 为了最大限度地减少内存带宽浪费, ReBFS [36] 利用内存处理 (PIM) 与 ReRAM 相结合的模式, 提高了计算和 I/O 性能. 它的好处在于图数据可以在 ReRAM 中持久存储和处理, 并可以达到低数据移动开销和高 bank 级并行计算的要求. Tesseract [37] 充分利用可用存储器带宽的新硬件架构 HMC, 解决了随机访问模式导致的本地带宽降级问题, 以及由全局数据访问的局部性导致的计算效果不可预测等问题. GraphH [19] 集成了基于 SRAM 的片上顶点缓冲器, 以消除本地带宽衰减, 并且引入了可重构双网连接以提供高全局带宽.

内存相关的通信优化: Tesseract [37] 提出了不同存储器分区之间的有效通信方法, 设计了独特硬件设计的编程接口. 它还包括两个专门用于图处理的存储器访问模式的硬件预取程序. 尽管与基于 DRAM 的系统相比, PIM 具有数量级的加速, 但 Tesseract 通过 SerDes 链路产生过多的交叉立方通信, 其带宽远小于 HMC 的聚合本地带宽, 这是因为顶点编程模型所需的数据组织受限. GraphP [21] 认为基于 PIM 的图处理系统应该将数据组织作为一阶设计考虑因素. 这是一种新型的基于 HMC 的软件/硬件协同设计的图计算系统, 主要减少了通信和能耗. 它通过 (1) "源切割" 分区, 从根本上改变跨

多维数据集通信, 从每个跨立方体边缘的一个更新远程放置到每个副本; (2) “两阶段顶点程序”, 一种为 “源切割” 分区设计的编程模型, 具有两个操作: GenUpdate 和 ApplyUpdate; (3) 分层通信和重叠, 通过提出的分区和编程模型提供的独特机会进一步提高了性能.

5.3 能耗效率优化

热感知与及时冷却: GraphBig [35] 提出了一种热感知源限制机制 CoolPIM, 可控制运行时 PIM 卸载的强度. 该技术使用基于软件的技术将 HMC 的存储器管芯保持在正常操作温度内. 研究观察到 HMC 的工作温度远高于传统的 DRAM, 甚至可以通过被动冷却解决方案引起热关断. 但是, 即使使用服务器冷却解决方案, 当内存处理被高度利用时, HMC 也无法将内存芯片的温度维持在正常工作范围内, 从而导致更高的能耗和性能开销.

分区和调度算法: 如何分配数据和调度处理流程以避免冲突和平衡工作负载也是重要的降低能耗的方式. GraphH [19] 引入了分区和调度算法, 如 Index Mapping Interval-Block 和 Round Interval Pair, 从而平衡了工作负载, 避免了冲突. 进一步分别进行优化, 以减少同步开销和重用片上数据.

6 图计算专用加速器中的内存管理技术

6.1 硬件特点与构造

ASIC 与 FPGA 都是集成电路, 片上计算单元的集中和高并行度提高了资源利用率, 使得它们运行速度很快, 直接电路的功能减少了通信消耗, 使得功耗很小, 这也是它们迅速火起来的主要原因. 然而伴随的最大问题是片上存储资源少, 这也是通用协同处理器的通病. 面对大规模图计算的需求, 我们不仅需要解决提高片上内存利用率的问题, 同时更要考虑片上与主机 CPU 的内存通信与切换.

由于 ASIC 是根据功能将电路直接焊制到晶片上, 所以功能一经焊制无法改变. FPGA 的出现让我们可以反复配置数字逻辑, 也称可编程性, 虽然相比于 ASIC 运行速度降低但可编程的设计让更多人使用, 执行效率也远比 CPU 高效. FPGA 内部不同类型的可编程资源, 除可编程逻辑单元 LE 外还有静态随机存取存储器 SRAM (static random access memory)、闪存和块 RAM (block RAM) 等. FPGA 通过使用流水线多指令单数据模型 MISD 构建这些资源, 可以提供更高的并行性.

6.2 片上与片下内存访问优化

对于图计算加速器, 提高内存效率主要是通过设计高效的存储器子系统 [2, 3]. 为了构建高效的存储器子系统, 人们在片上 BRAM 和片下存储器的高效带宽利用方面进行了大量研究与实验.

片上内存访问优化: FPGP [4], ForGraph [7], FPGA-HMC 平台 [38] 使用 BRAM 为随机访问的顶点提供高带宽和低内存延迟. 为了改善 BRAM 上顶点的局部性, 研究者们采用粗粒度划分和专用数据暂存策略来提高 BRAM 上顶点的重用率. Zhou 等 [8] 提出了一种数据布局, 可优化片外存储器性能, 并实现高效的存储器激活或关闭功能, 以降低片上存储器功耗. 他们提出片上存储器由许多 BRAM 模块组成, 每个模块存储相同数量的顶点数据, 并设计了一个 “启用” 端口来选择性地激活和停用 BRAM 模块. 使用这种方法, 当访问的数据存储在 BRAM 模块中时, BRAM 模块被激活, 否则它被停用以保存 BRAM 功率. 每个模块都有 p 个读端口和 p 个写端口 (表示为 pR/pW), p 个处理流水线可以同时读写片上存储器. 为了隐藏从 DRAM 访问顶点数据的延迟, 还采用了双缓冲技术: 将片上存储器均匀

地分成两个块, 一个块存储正在处理的分区的顶点集, 另一个块中存储控制器预取的下一个分区的顶点集.

片外存储器带宽优化: FPGA 和 CPU 在通过高速缓存一致性互连连接的情况下, 也采用异构架构. FPGA 可以在不中断 CPU 的情况下访问主机存储器. Zhou 等在另一篇文章 [10] 中提出, 这两个处理器可以轻松地相互协作, 以处理比单个 FPGA 板更高并行度的大型图. 对于内存不足的 FPGA 上的大型图处理过程, 还有一些研究专注于寻找更直接的数据交换方式. 在 FPGP [4] 和 ForeGraph [7] 中, 数据可以直接从磁盘或闪存流动传输到 FPGA 板上的处理单元. Song 等 [9] 提出的 GraphR 采用 ReRAM 作为存储介质, 基于 ReRAM 总线的图处理引擎可以实现高效的稀疏矩阵处理.

内存层次结构优化: 设计并使用新的内存架构可以提高内存访问速率. 很多图加速器通常采用暂存器来代替传统的缓存. 暂存器以存储器充当内容, 可寻址高速缓存, 并可以人为控制. Graphicionado [2] 使用 eDRAM 作为 ASIC 的暂存器来存储需要频繁随机访问的图数据, 例如目标顶点. 根据访问特征, Ozdal 等 [3] 在 RTL 级设计了多个不同种类图数据的专用缓存. 由于这些存储器资源可以以有效的方式紧密地连接到处理单元, 因此基于 ASIC 的图加速器可以在芯片上实现高吞吐量.

6.3 并行内存访问与执行

流水线模式: 流水线模式能够提供更高的并行性. 基于 ASIC 或 FPGA 的图计算加速器中的处理单元通常以流水线的形式组织 [4,7,38]. 图算法的指令是流水线的. GraphOps [17] 生成的加速器由一系列硬件模块组成, 图数据和计算的元数据以流的形式在模块间传输.

点对点的存储交互: 点对点的存储交互也能够提升并行度. Graphlet [39] 构建了可重构的硬件加速框架. 连接并行的图处理单元 (graph processing elements, GPE)、存储交互网络和运行时间管理单元. 在存储交互网络中, 每个 GPE 有着对应的 FIFO, 并由时间管理单元管理执行操作. 这个框架在并行数据处理后有点对点的存储交互, 从而整体 GPE 和存储的带宽被提高.

多个 Bank 和 I/O 端口: 内存级并行性 (memory level parallelism, MLP) 指能够支持的同时内存请求的数量. 较高的 MLP 可以减少数据密集型应用程序的总内存访问时间, 如图处理通常需要内存设备来支持足够的并发内存请求. 传统的增强内存级并行性的方法主要有两种. 一种方法是使用多组 (Bank) 模式. DRAM 由许多独立的 Bank 组成并且具有可以利用的并行性, 可以同时连接到处理单元并且可以同时访问 [39]. 增加 I/O 端口也可以增加内存级并行性. Graphicionado [2] 可以在暂存器上手动设计端口数量, 当端口数等于处理单元数时, 可以获得最高 MLP. FPGA 上的 BRAM 也可以人为控制以实现这一目标 [8]. 这些 BRAM 通常组合在一起形成具有多个 I/O 端口的存储器块.

新型内存架构 HMC: HMC 这两年被提上日程, 用于提高访存效率. 邻近数据计算 NDP [34] 通过将工作负载加载到集成的 HMC 来协同地增强 FPGA 的图计算能力. 这也能明显提高访存带宽和并行性 [15,40,41]. Zhang 等 [15] 提出, 数据包是串行到达 HMC 的, 当多个数据包到达时, 我们可以通过一些处理使这些数据包到达 HMC 中不同的层以提高并行度. 他们团队中 Khoram 等 [40] 提出了另一种方法, 利用类似于 MapReduce 的过程将大图分为小图, 再分割为子小图. 每一个子小图为一个聚类, 在分别处理之后的合并过程中, HMC 作为合并周期的主要载体.

共享内存: 共享内存模型通常可以避免消息传递模型中图数据的冗余副本和额外的存储空间. 它也易于实现和设计. 但是如果某些顶点由许多相邻顶点更新, 则在同一存储器位置上可能存在许多数据冲突, 这也是我们之后需要考虑的方向. ForeGraph [7] 使用分布式共享内存. FPGP [4] 采用基于 FPGA 的共享内存模型. 它为多个 FPGA 板维护一个全局共享顶点存储器, 每个板为多个处理单元保

留顶点缓存. 需要在迭代之间进行同步以保持内存一致性. 受有限带宽的限制, 全局共享顶点存储器可能会限制 FPGA 的可扩展性.

6.4 内存带宽优化

带宽利用率指每次传输的有效值比率. 图处理中的随机访问通常访存效率大幅下降, 并浪费大量带宽, 提高内存带宽利用率可以减少内存访问的总数. 两种提高带宽利用率的方式包括合并请求方式和顺序访问边操作.

合并请求方式: 意味着将多个小项目的转移组合成较少的大项目. 该方法在图加速器 [5,8,15] 中被广泛采用. 例如, 如果内存请求在顶点或边列表中相邻, 这些请求可以作为一个块的请求合并, 否则可能存在几个导致带宽浪费的随机访问 [5].

顺序访问边操作: 意味着从存储器到交换器顺序访问第一条边 [8], 可以减少对边的随机访问. 在以顶点为中心的模型中, 顶点的边可以顺序传输到芯片上 [2]. 该方法可以充分利用以边为中心的模型中的带宽. 但是, 边可能需要重新排序, 以便以更有效的方式运行 [10,15].

6.5 内存层次结构优化

缓存层次结构的重塑是提高访存的重要方法. 图处理的局部性差使得当前的缓存层次结构缺乏效率. 很高的缓存未命中率将加大内存访问延迟, 导致计算资源的利用不足. 重塑缓存层次结构意味着为图计算设计新的缓存架构和机制.

Scratchpad 内存: 用作明确控制的可寻址缓存. 暂存器内存对图处理过程是封闭的, 它可以为数据访问提供高性能 [42,43]. Graphicionado [2] 使用暂存器内存来存储临时顶点属性数组和边设置以优化随机数据访问. 类似地, Ozdal 等 [3] 还根据访问行为为顶点、边和其他图信息设计了不同类型的缓存.

局部感知缓冲区: 是图数据的专用缓存, 具有较好局部性, 例如高层次顶点. 幂律图中的高层次顶点总是可能会被多次访问. Tunao [44] 提出可以缓存这些高层次且被多次访问的顶点以提高性能. FPGP [4] 和 ForeGraph [7] 使用类似网格的分区方法改进了顶点的局部性, 并为顶点子集设计了特殊的片上缓冲区, 以便在重用时快速访问.

6.6 能耗效率优化

图计算加速器的性能可以用每秒遍历的边距 (traversed edges per second, TEPS) 来计量. 能量效率可进一步定义为 TEPS 每瓦特 (TEPS/W). 现有的图计算加速器常通过具有固有低能耗的专用电路提供高性能, 代表性的就是 FPGA 和 ASIC. 但是, 大多数图计算应用内存访问计算比率较高, 访存消耗不能忽视. 邻近数据计算框架 [34] 的结果表明, PageRank 在内存上消耗的能耗超过 60%. 这也说明对内存能耗的优化可以进一步提高能源效率.

新兴存储器技术利用新兴存储器技术将计算逻辑集成在存储器内是一个很好的降低能耗的方式. 例如, 前所述的 HMC [19,21,33,37] 和 ReRAM [9,45], 这种架构改造可以沿着数据进行现场计算. 它自然避免了频繁的数据移动以节省能源. 在这一点上, 我们可以通过使用这些新兴的存储设备轻松取代传统的 DRAM.

电源门控技术是一种广泛使用的技术. 它可以在逻辑电路空闲时关闭它以节省能量. 该方案适用于可认为控制的存储器 [10,45]. 例如, Zhou 等 [10] 将这个方法应用于 FPGA 上的 BRAM, 通过启用的

表 1 基于不同内存优化的图计算系统总结

Table 1 Summary of graph processing systems on memory optimization

Optimization scheme	Graph processing systems [system, architecture, year]
Data layout	[GraphOps [17], FPGA, 2016], [CyGraph [18], FPGA, 2014], [GraphH [19], PIM, 2018], [Graphicionado [2], ASIC, 2016], [ForeGraph [7], FPGA, 2017], [FPGP [4], FPGA, 2017], [Tigr [24], GPU, 2018], [X-Stream [6], CPU, 2013], [Mosaic [14], CPU, 2017], etc.
Graph partition	[GraphChi [20], CPU, 2012], [Cusha [28], GPU, 2014], [Graphicionado [2], ASIC, 2016], [GraphH[19], PIM, 2018], [FPGP [4], FPGA, 2017], [GraphP [21], PIM, 2018], [GStream [30], GPU, 2015], [GTS [29], GPU, 2016], [GridGraph [23], CPU, 2015], [ForeGraph [7], FPGA, 2017], [Gemini [16], CPU, 2016], [Frog [27], GPU, 2015], [Page [46], CPU, 2015], [GraphReduce [31], GPU, 2015], etc.
Storage media	[GraphPIM [33], PIM, 2017], [Graphicionado [2], ASIC, 2016], [FPGP [4], FPGA, 2017], [ForeGraph [7], FPGA, 2017], [FPGA-HMC [15], FPGA, 2018], [Ozdal [3], ASIC, 2016], [GraphH [19], PIM, 2018], [GraphR [9], PIM, 2018], [GraphP [21], PIM, 2018], [Tesseract [37], PIM, 2015], etc.
Memory access	[REBFS [36], PIM, 2018], [Tesseract [37], PIM, 2015], [GraphH[19], PIM, 2018], [Graphicionado[2], ASIC, 2016], [GraphGen [5], FPGA, 2014], [FPGA-HMC [38], FPGA, 2018], [GTS [29], GPU, 2016], [Frog [27], GPU, 2015], etc.
Communication	[Tesseract [37], PIM, 2015], [GraphP [21], PIM, 2018], [GraphH [19], PIM, 2018], [Groute [47], GPU, 2017], [FPGP [4], FPGA, 2017], [ForeGraph [7], FPGA, 2017], [Frog [27], GPU, 2015], [Tornado [48], CPU, 2016], [GTS [29], GPU, 2016], etc.
Power	[GraphBIG[35], PIM, 2015], [GraphH[19], PIM, 2018], [Graphicionado [2], ASIC, 2016], [GraphP[21], PIM, 2018], [GraphR [9], PIM, 2018], [GraphPIM [33], PIM, 2017], [HyVE [45], CPU, 2018], [Tesseract [37], PIM, 2015], [Ozdal [3], CPU, 2016], [Congra [49], CPU, 2017], [Tunao [44], ASIC, 2017], etc.

端口选择性地激活和停用 BRAM, 这是降低整体 FPGA 能耗的关键. BRAM 模块仅在存储所需数据时激活. 该策略亦可用于 ReRAM [45], 通过控制 ReRAM bank 的激活来节省边沿访问的能量.

第 3~6 节分别从 GPU、PIM、图计算专用加速器 FPGA 和 ASIC 这几个方面详细分析了异构架构下的图计算内存系统优化, 我们对以上提到的常见框架或系统进行了总结, 如表 1 所示.

7 国内近期研究进展

针对图计算的研究, 国内的学者也都展现了极大的积极性, 虽然国内在图计算的研究过程中起步比较晚, 但是国内各大机构的研究人员也都在图计算的研究方面取得了不错的成绩.

清华大学设计并实现了基于 Grid 数据切分的 out-of-core 图数据处理系统, 采用二级分割模式, 将大规模图数据分割成 Grid 的数据格式, 充分利用内存的访问局部性, 提高数据访问和处理的性能. 基于多种优化策略的高性能分布式图处理系统 Gemini [16], 采用基于分块 (chunk) 的图数据分割模式, 尽力保证图数据的局部性, 便于数据的访问和计算. 考虑到 NUMA 架构下, 远程内存访问开销大的问题, 对数据进行二次分割, 分到不同的套接字 (socket) 上, 极大地降低了内存访问的延迟; 同时采用压缩系数矩阵的方式降低了边处理的内存访问需求, 对于异构架构的图计算, 近两年他们也设计了基于 FPGA 加速的图计算系统 ForeGraph [7] 和 FPGP [4]. ForeGraph 关注了各 FPGA 之间内存的通信问题, FPGP 关注了板上资源如内存的限制, 将顶点集存储在共享片内 RAM 中, 将外存上的边集分割以流式处理. 在 PIM 内存计算领域, 提出了基于 PIM 的图计算专用加速器 GraphH [19], 将大图分割为

多个紧密的数据结构并行计算, 并取得了很好的实验效果.

华中科技大学设计了基于数据流模型的 FPGA 图专用加速器 [50], 提出通过数据流的方式突破底层处理器效率低下问题, 并且会改善大图计算的计算性能和访存效率. 基于数据流的图处理最大的问题是当前的活跃数据很多, 片上内存不够存储, 于是使用调度表, 在片上内存中维护相关数据, 利用访存过程中的时间局部性, 提高内存访问的性能. 为了克服图数据并行处理过程中的数据冲突问题, 研究人员提出了高效的冲突管理方法 AccuGraph [51], 原子顶点通过累加器的方式进行高并行度的更新, 高吞吐量的片上内存被分为独立的部分以处理多路访问, 在保证正确性的情况下提升并行度. 除了在 FPGA 加速图计算上作出自己的贡献, 华中科技大学针对 GPU 架构下异步图计算容易产生数据冲突和加锁开销的问题, 提出了 GPU 图计算系统 Frog [27], 它使用一种着色模型的调度算法来减少数据冲突, 同时这种方式解决了大图无法放入 GPU 内存的问题, 基于着色算法的分割后的数据集互相之间的不冲突, 是能够达到高并行度的主要原因.

北京大学的研究人员提出图划分感知的图计算引擎 PAGE [46], 通过联机监控执行过程中的系统状态调整计算引擎的并行策略, 将图数据划分后存储在各个 worker 的主存中, 使用新的消息处理器和动态并发控制模型, 以处理不断增加的本地消息处理工作负载. 北京大学设计的 Seraph [52] 是一个支持高效的作业级并行的图计算系统, 它采用数据与计算模型分离的方式和写时拷贝机制优化面向同一图数据的多个并发计算任务, 并采用延后镜像协议为在不同时间提交的备份请求生成一致的内存图镜像降低容错开销. 这个模型允许多个并发作业共享内存中的图数据, 每个作业只需要维护少量特定于作业的图数据而不是维护所需要的全部数据, 大大节省了内存占用.

上海交通大学提出了基于同步计算调度和异步数据传输的复合计算引擎 PowerSwitch [53], 认为分布式图计算中同步与异步模式各有好处, 并提出一种动态的同步异步切换策略, 提高计算的性能. 在图处理过程中, 图数据被分区到多个节点, 在计算中通过顶点的复制为共享内存访问提供本地缓存. 混合图划分算法的计算引擎 PowerLyra [13] 提出, 对所有顶点进行统一处理容易引起负载不平衡、通信和内存消耗成本高等问题, PowerLyra 能够动态地自适应地应用不同次数顶点的各种计算和分割策略, 对低次顶点和高次顶点分别进行微分计算和划分, 还设计了数据布局优化, 以提高通信过程中图计算访问的缓存局域性, 在同样大小的内存中可以处理规模更大的图. 基于 NUMA 架构下高性能图计算系统 Polymer [54] 指出, 随机或交错分配的图数据都将严重阻碍数据的局部性和并行性, 顺序的节点间内存访问 (远程内存访问) 具有比节点内和节点间随机访问高得多的带宽, 通过优化图数据布局和访问策略来最小化随机和远程内存访问, 并要求访问线程在其本地内存节点中分配内存, 从而消除远程访问. 考虑多图运行对于服务器内存带宽以及 CPU 的竞争情况, 研究人员设计出基于共享内存的并发图计算调度方案 Congraph [49], 主要关注原子操作成本的增加与使用更多 CPU 内核的可用内存带宽的增加之间的权衡, 以最大限度地利用已有内存带宽.

8 机遇与挑战

研究工作者们已经设计了很多高性能、低功耗的图计算系统, 也提出了多种内存管理的优化策略. 随着新技术和新硬件的不断涌现, 图计算依旧面临很多挑战, 发展机遇更是不容小觑. 通过上述调查分析, 我们列出了图计算系统设计过程中在以下几个方面的挑战与发展机遇.

- 图计算加速器高效编程环境的设计. 研究人员对于异构加速的图计算系统研究已经非常深入, 但是仍然缺少高效的开发环境. 虽然在 GPU 架构下有 CUDA 作为编程框架, 但是在 PIM 架构以及专用加速器方面的编程很大程度上依赖于使用硬件描述语言的低级编程. 这就要求开发人员必须了解

底层硬件细节, 使得异构图计算系统的设计开发周期非常长. 图算法设计时, 高级编程语言与硬件设计描述之间的有效转换映射仍是问题. 通用高层次综合技术 (high-level synthesis, HLS) 提供了一种可行的解决方案, 但没有充分考虑图特征, 运行效率低下. 在进行异构图计算加速器设计时, 如果图程序遇到并发和性能错误, 程序员必须重建和重新连接硬件电路, 代价昂贵. 因此设计基于图计算特性的 HLS 系统, 针对图计算的编译技术、错误诊断工作以及简洁易用的编程库是加快异构图计算系统设计开发的重要途径.

• 新兴硬件与技术在图计算上的应用. 随着技术的发展, 新的硬件出现也给图计算的加速带来机遇和挑战. 前文提到的 HMC 与 ReRAM 等新兴存储硬件用来进行图计算的存储和加速, 并在性能和能量方面取得了良好的效果. 然而这些新硬件还未充分利用. 例如, GraphR [9] 仅使用一层 ReRAM, 但事实是 ReRAM 经常堆叠使用. GPU 的架构随着研究人员的努力不断地更新换代, 其计算能力与存储性能也在不断地更新完善, 这些都为基于 GPU 架构的异构图计算系统开发提供了新的可能. 利用 FPGA 进行高效低功耗的图计算加速器设计是一个很有前景的课题. 许多 FPGA 供应商如亚马逊[1)]、百度[2)]、腾讯[3)]提供了云 FPGA 开发环境. 云上丰富的资源和集成开发工具为图处理加速器的敏捷开发提供了机会. 这些新兴的硬件为图计算的内存优化和性能提升提出了新的挑战和可能.

• 图拓扑结构复杂性的处理. 随着新的图应用出现, 不同的应用下对于图的描述有不同的要求, 图的顶点和边的属性也是多样化的. 图的这些复杂属性会给图计算的内存访问和计算带来很大的挑战 [55], 现有的图处理技术还未能有效地解决这类问题. 现在的工作主要是针对静态的图数据进行研究和分析, 涉及动态图数据的研究还比较少. 动态图数据随时间改变结构, 对于算法的设计及加速器的设计都带来极大的挑战. 动态图处理是一个热门的研究课题 [56, 57]. 一些基于子图增量变化的方法在小规模增量下已经取得了相对较好的效果 [48], 但是大规模时流式图的有效处理仍然是一个悬而未决的问题. 尤其在异构环境下, 对于流式图的研究更是充满了挑战.

9 总结

本文主要关注现有图计算系统中的内存管理技术与优化方案. 基于不同的架构分析总结了图计算中的内存技术优化, 同时指出了图计算加速在现阶段的挑战以及新兴技术与新兴硬件给图计算加速带来的机遇. 希望本文可以帮助更多的研究人员了解图计算的运算模式与内存系统的优化策略, 从而设计开发出更加高性能的图计算系统.

参考文献

1 Malewicz G, Austern M H, Bik A J, et al. Pregel: a system for largescale graph processing. In: Proceedings of the 2010 International Conference on Management of Data, Indianapolis, 2010. 135–146

2 Ham T J, Wu L, Sundaram N, et al. Graphicionado: a high-performance and energyecient accelerator for graph analytics. In: Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture, Taipei, 2016. 1–13

3 Ozdal M M, Yesil S, Kim T, et al. Energy ecient architecture for graph analyticsaccelerators. In: Proceedings of the 23rd ACM/IEEE Annual International Symposium on Computer Architecture, Seoul, 2016. 166–177

4 Dai G H, Chi Y Z, Wang Y, et al. FPGP: graph processing framework on FPGA a case study of breadth-first search. In: Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, 2016.

1) Amazon EC2 F1 Instances. http://aws.amazon.com/ec2/instance-types/f1/.

2) Baidu FPGA Cloud. http://cloud.baidu.com/product/fpga.html.

3) Tencent FPGA Cloud. http://cloud.tencent.com/product/fpga.

105–110
5 Nurvitadhi E, Weisz G, Wang Y, et al. Graphgen: an FPGA framework for vertex-centric graph computation. In: Proceedings of the 22nd IEEE International Symposium on Field-Programmable Custom Computing Machines, Boston, 2014. 25–28
6 Roy A, Mihailovic I, Zwaenepoel W. X-stream: edge-centric graph processing using streaming partitions.In: Proceedings of the 24th ACM Symposium on Operating Systems Principles, Farmington, 2013. 472–488
7 Dai G, Huang T, Chi Y, et al. Fore-graph: exploring large-scale graph processing on multi-FPGA architecture. In: Proceedings of the 25th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, 2017. 217–226
8 Zhou S J, Prasanna V K. Accelerating graph analytics on CPU-FPGA heterogeneous platform. In: Proceedings of the 29th International Symposium on Computer Architecture and High Performance Computing, Campinas, 2017. 137–144
9 Song L, Zhuo Y, Qian X, et al. GraphR: acceleratinggraph processing using ReRAM. In: Proceedings of the 24th IEEE International Symposium on High-Performance Computer Architecture, Vienna, 2018. 531–543
10 Zhou S, Chelmis C, Prasanna V K. High-throughput andenergy-ecient graph processing on FPGA. In: Proceedings of the 24th International Symposium Field-Programmable Custom ComputingMachines, Washington, 2016. 103–110
11 Lakhotia K, Kannan R, Prasanna V. Accelerating PageRank using partition-centric processing. In: Proceedings of the 28th USENIX Security Symposium on Annual Technical Conference, Santa Clara, 2018. 427–440
12 Gonzalez J E, Low Y, Gu H, et al. Powergraph: distributed graph-parallel computation on natural graphs. In: Proceedings of the 10th Usenix Symposium on Operating Systems Design and Implementation, Hollywood, 2012. 17–30
13 Chen R, Shi J, Chen Y, et al. Powerlyra: differentiated graph computation and partitioning on skewed graphs. In: Proceedings of the 10th European Conference on Computer Systems, Bordeaux, 2015. 1–15
14 Maass S, Min C, Kashyap S, et al. Mosaic: processing a trillion-edge graph on a single machine. In: Proceedings of the 12th European Conference on Computer Systems, Belgrade, 2017. 527–543
15 Zhang J, Khoram S, Li J. Boosting the performance of FPGA-based graph processor using hybrid memory cube: a case for breadth rst search. In: Proceedings ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, 2017. 207–216
16 Zhu X W, Chen W G, Zheng W M, et al. Gemini: a computation-centric distributed graph processing system. In: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, 2016. 301–316
17 Oguntebi T, Olukotun K. GraphOps: a dataflow library for graph analytics acceleration. In: Proceedings of the 24th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, 2016. 111–117
18 Attia O G, Johnson T, Townsend K, et al. CyGraph: a reconfigurable architecture for parallel breadth-first search. In: Proceedings of the 28th International Parallel and Distributed Processing Symposium Workshops, Phoenix, 2014. 228–235
19 Dai G H, Huang T H, Chi Y Z, et al. GraphH: a processing-in-memory architecture for large-scale graph processing. IEEE Trans Comput-Aided Des Integr Circuits Syst, 2018. doi: 10.1109/TCAD.2018.2821565
20 Kyrola A, Blelloch G, Guestrin C. Graphchi: large-scale graph computation on just a pc. In: Proceedings of the 10th Usenix Symposium on Operating Systems Design and Implementation, Hollywood, 2012. 31–46
21 Zhang M X, Zhuo Y W, Wang C, et al. GraphP: reducing communication for PIM-based graph processing with efficient data partition. In: Proceedings of the 24th IEEE International Symposium on High-Performance Computer Architecture, Vienna, 2018. 544–557
22 Umuroglu Y, Morrison D, Jahre M. Hybrid breadth-first search on a single-chip FPGA-CPU heterogeneous platform. In: Proceedings of the 25th International Conference on Field Programmable Logic and Applications, London, 2015. 1–8
23 Zhu X W, Han W T, Chen W G. GridGraph: large-scale graph processing on a single machine using 2-level hierarchical partitioning. In: Proceedings of USENIX Annual Technical Conference, Santa Clara, 2015. 375–386
24 Nodehi S A H, Qiu J Q, Zhao Z J. Tigr: transforming irregular graphs for GPU-friendly graph processing. In: Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and

Operating Systems, Virginia, 2018. 622–636
25 Zhong J L, He B S. Medusa: simplified graph processing on GPUs. IEEE Trans Parallel Distrib Syst, 2014, 25: 1543–1552
26 Fu Z S, Personick M, Thompson B. MapGraph: a high level API for fast development of high performance graph analytics on GPUs. In: Proceedings of Workshop on GRAph Data management Experiences and Systems, Snowbird, 2014. 1–6
27 Shi X H, Liang J L, Di S, et al. Optimization of asynchronous graph processing on GPU with hybrid coloring model. In: Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Francicso, 2015. 271–272
28 Khorasani F, Vora K, Gupta R, et al. CuSha: vertex-centric graph processing on GPUs. In: Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing, Vancouver, 2014. 239–252
29 Kim M S, An K, Park H, et al. GTS: a fast and scalable graph processing method based on streaming topology to GPUs. In: Proceedings of the 2016 International Conference on Management of Data, San Francisco, 2016. 447–461
30 Seo H, Kim J, Kim M S. Gstream: a graph streaming processing method for large-scale graphs on gpus. In: Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Francicso, 2015. 253–254
31 Sengupta D, Song S L, Agarwal K, et al. GraphReduce: processing large-scale graphs on accelerator-based systems. In: Proceedings of the 27th International Conference for High Performance Computing, Networking, Storage and Analysis, Austin, 2015. 1–12
32 Wang Y Z, Davidson A, Pan Y C, et al. Gunrock: a high-performance graph processing library on the GPU. In: Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Barcelona, 2016. 265–266
33 Nai L F, Hadidi R, Sim J, et al. GraphPIM: enabling instruction-level PIM offloading in graph computing frameworks. In: Proceedings of the 23rd IEEE Symposium on High Performance Computer Architecture, Austin, 2017. 457–468
34 Gao M Y, Ayers G, Kozyrakis C. Practical near-data processing for in-memory analytics frameworks. In: Proceedings of the 24th International Conference on Parallel Architectures and Compilation Techniques, San Francisco, 2015. 113–124
35 Nai L, Xia Y, Tanase I G, et al. GraphBIG: understanding graph computing in the context of industrial solutions. In: Proceedings of the 27th International Conference for High Performance Computing, Networking, Storage and Analysis, Austin, 2015. 1–12
36 Han L, Shen Z, Liu D, et al. A novel ReRAM-based processing-in-memory architecture for graph traversal. ACM Trans Storage, 2018, 14: 1–26
37 Ahn J, Hong S, Yoo S, et al. A scalable processing-in-memory accelerator for parallel graph processing. In: Proceedings of the 42nd International Symposium on Computer Architecture, Portland, 2015. 105–117
38 Zhang J L, Li J. Degree-aware hybrid graph traversal on FPGA-HMC platform. In: Proceedings of the 26th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, 2018. 229–238
39 Betkaoui B, Thomas D B, Luk W, et al. A framework for FPGA acceleration of large graph problems: graphlet counting case study. In: Proceedings of International Conference on Field-Programmable Technology, Delhi, 2011. 1–8
40 Khoram S, Zhang J, Strange M, et al. Accelerating graph analytics by co-optimizing storage and access on an FPGA-HMC platform. In: Proceedings of the 26th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, 2018. 239–248
41 Wang Q B, Jiang W R, Xia Y L, et al. A messagepassing multi-softcore architecture on FPGA for breadth-first search. In: Proceedings of International Conference on Field-Programmable Technology, Beijing, 2010. 70–77
42 Windh S, Budhkar P, Najjar W A. CAMs as synchronizing caches for multithreaded irregular applications on FPGAs. In: Proceedings of the 34th IEEE/ACM International Conference on Computer-Aided Design, Austin, 2015. 331–336
43 Wang L, Yang X J, Dai H D. Scratchpad memory allocation for arrays in permutation graphs. Sci China Inf Sci, 2013, 56: 1–13
44 Zhou J, Liu S, Guo Q, et al. Tunao: a high-performance and energy-efficient reconfigurable accelerator for graph processing. In: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

IEEE, Madrid, 2017. 731–734
45 Huang T H, Dai G H, Wang Y, et al. HyVE: hybrid vertex-edge memory hierarchy for energy-efficient graph processing. In: Proceedings of the 2018 Design, Automation and Test in Europe Conference and Exhibition, Dresden, 2018. 973–978
46 Shao Y, Cui B, Ma L. PAGE: a partition aware engine for parallel graph computation. IEEE Trans Knowl Data Eng, 2015, 27: 518–530
47 Ben-Nun T, Sutton M, Pai S, et al. Groute: an asynchronous multi-GPU programming model for irregular computations, In: Proceedings of the 23th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Austin, 2017. 235–248
48 Shi X, Cui B, Shao Y, et al. Tornado: a system for real-time iterative analysis over evolving data. In: Proceedings of the 2016 International Conference on Management of Data, San Francisco, 2016. 417–430
49 Pan P, Li C. Congra: towards efficient processing of concurrent graph queries on shared-memory machines. In: Proceedings of the 35th IEEE International Conference on Computer Design, Boston, 2017. 217–224
50 Jin H, Yao P C, Liao X F, et al. Towards dataflow-based graph accelerator. In: Proceedings of the 37th International Conference on Distributed Computing Systems, Atlanta, 2017. 1981–1992
51 Yao P C, Zheng L, Liao X F, et al. An efficient graph accelerator with parallel data conflict management. In: Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques, Limassol, 2018. 1–12
52 Xue J, Yang Z, Qu Z, et al. Seraph: an efficient, low-cost system for concurrent graph processing. In: Proceedings of the 23rd International Symposium on High-performance Parallel and Distributed Computing, Vancouver, 2014. 227–238
53 Xie C N, Chen R, Guan H B, et al. Sync or async: time to fuse for distributed graph-parallel computation. In: Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Francisco, 2015. 194–204
54 Zhang K Y, Chen R, Chen H B. NUMA-aware graph-structured analytics. In: Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Francisco, 2015. 183–193
55 Zhang M X, Wu Y W, Chen K, et al. Exploring the hidden dimension in graph processing. In: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, 2016. 285–300
56 Sha M, Li Y, He B, et al. Accelerating dynamic graph analytics on GPUs. Proc VLDB Endow, 2017, 11: 107–120
57 Chen H Y, Sun Z G, Yi F, et al. BufferBank storage: an economic, scalable and universally usable in-network storage model for streaming data applications. Sci China Inf Sci, 2016, 59: 012103

Memory system optimization for graph processing: a survey

Jing WANG, Lu ZHANG, Pengyu WANG, Jiahong XU, Chao LI*, Haojin ZHU, Xuehai QIAN & Minyi GUO

Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240*, China*
* Corresponding author. E-mail: lichao@cs.sjtu.edu.cn

Abstract Containing a variety of information, a graph is a complex data structure comprising vertices and edges. Graph processing or graph computing is the abstraction of the relation and properties of graph structures in real-life situations and performing some complex computations. Owing to the bottlenecks in the performance of the central processing unit, many coprocessors and domain-specific accelerators have been designed to improve running speed and to save energy. Considering the strong data dependence of graphs, improving the efficiency of memory access is a critical issue to improve system performance. In particular, memory management and optimization have become extremely important due to the expansion of graph data scale and the acceleration of various graph processing. This study aims to propose a memory architecture and management methods in graph processing on heterogeneous architecture. We describe the graph data layout that can improve the efficiency of memory access, analyze recent work on memory optimization and features of GPU, FPGA, ASIC, PIM, among others. Furthermore, we summarize relevant research progresses in recent years in China, and conclude the opportunities and challenges of graph processing in memory.

Keywords graph processing, accelerator, memory management, memory system architecture, memory access optimization

Jing WANG was born in 1996. She received her B.S. degree from the Northwestern Polytechnical University. She is currently working toward obtaining a Ph.D. degree in the Shanghai Jiao Tong University. Her research interests include graph processing and programming model.

Chao LI was born in 1986. He received his B.S. degree from the Zhejiang University and his Ph.D. degree from the University of Florida. He is currently a tenure-track assistant professor at the Department of Computer Science and Engineering at Shanghai Jiao Tong University. His research interests include high-performance computer architectures, data center power management, and emerging technologies/applications.

Haojin ZHU is currently a professor at the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. He received his B.S. degree in 2002 from the Wuhan University, China, his M.S. degree in 2005 from Shanghai Jiao Tong University, China, both in computer science, and a Ph.D. degree in electrical and computer engineering from University of Waterloo, Canada, in 2009. His current research interests include network security and data privacy.

Minyi GUO received his B.S. and M.E. degrees in computer science from the Nanjing University, China and a Ph.D. degree in information science from the University of Tsukuba, Japan, in 1982, 1986, and 1998, respectively. He was a visiting professor at the Department of Computer Science, Georgia Institute of Technology. Besides, he was a full time professor at the University of Aizu, Japan, and he is currently the head of the Department of Computer Science and Engineering, Shanghai Jiao Tong University, China. His main research interests include automatic parallelization and data-parallel languages, bioinformatics, compiler optimization, and high-performance computing.