计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2021.20200110
58(4):862-887,2021

图计算加速架构综述

严明玉[1,2,3] 李 涵[1,2] 邓 磊[3] 胡 杏[3] 叶笑春[1] 张志敏[1] 范东睿[1,2] 谢 源[3]
[1](计算机体系结构国家重点实验室(中国科学院计算技术研究所) 北京 100190)
[2](中国科学院大学 北京 100049)
[3](美国加州大学圣塔芭芭拉分校 美国加利福利亚州圣塔芭芭拉 93106)
(yanmingyu@ict.ac.cn)

A Survey on Graph Processing Accelerators

Yan Mingyu[1,2,3], Li Han[1,2], Deng Lei[3], Hu Xing[3], Ye Xiaochun[1], Zhang Zhimin[1], Fan Dongrui[1,2], and Xie Yuan[3]
[1](*State Key Laboratory of Computer Architecture* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)
[2](*University of Chinese Academy of Sciences*, *Beijing* 100049)
[3](*University of California at Santa Barbara*, *Santa Barbara*, *California*, *USA* 93106)

Abstract In the big data era, graphs are used as effective representations of data with the complex relationship in many scenarios. Graph processing applications are widely used in various fields to dig out the potential value of graph data. The irregular execution pattern of graph processing applications introduces irregular workload, intensive read-modify-write updates, irregular memory accesses, and irregular communications. Existing general architectures cannot effectively handle the above challenges. In order to overcome these challenges, a large number of graph processing accelerator designs have been proposed. They tailor the computation pipeline, memory subsystem, storage subsystem, and communication subsystem to the graph processing application. Thanks to these hardware customizations, graph processing accelerators have achieved significant improvements in performance and energy efficiency compared with the state-of-the-art software frameworks running on general architectures. In order to allow the related researchers to have a comprehensive understanding of the graph processing accelerator, this paper first classifies and summarizes customized designs of existing work based on the computer's pyramid organization structure from top to bottom. This article then discusses the accelerator design of the emerging graph processing application (i.e., graph neural network) with specific graph neural network accelerator cases. In the end, this article discusses the future design trend of the graph processing accelerator.

Key words graph processing; graph neural network; accelerator; irregular memory access; data locality; dynamic data access scheduling; workload balance

收稿日期:2020-02-26;**修回日期**:2020-05-13

基金项目:国家重点研发计划项目(2018YFB1003501);国家自然科学基金项目(61732018,61872335,61802367,61672499);中国科学院战略性先导科技专项(C类)(XDC05000000);数学工程与先进计算国家重点实验室开放基金(2019A07)
This work was supported by the National Key Research and Development Plan of China (2018YFB1003501), the National Natural Science Foundation of China (61732018, 61872335, 61802367, 61672499), the Strategic Priority Research Program of Chinese Academy of Sciences (XDC05000000), and the Open Project Program of the State Key Laboratory of Mathematical Engineering and Advanced Computing (2019A07).

通信作者:叶笑春(yexiaochun@ict.ac.cn)

摘 要 在大数据时代,图被用于各种领域表示具有复杂联系的数据.图计算应用被广泛用于各种领域,以挖掘图数据中潜在的价值.图计算应用特有的不规则执行行为,引发了不规则负载、密集读改写更新操作、不规则访存和不规则通信等挑战.现有通用架构无法有效地应对上述挑战.为了克服加速图计算应用面临的挑战,大量的图计算硬件加速架构设计被提出.它们为图计算应用定制了专用的计算流水线、访存子系统、存储子系统和通信子系统.得益于这些定制的硬件设计,图计算加速架构相比于传统的通用处理器架构,在性能和能效上均取得了显著的提升.为了让相关的研究学者深入了解图计算硬件加速架构,首先基于计算机的金字塔组织结构,从上到下对现有工作进行分类和总结,并以多个完整架构实例分析应用于不同层次的优化技术之间的关系.接着以图神经网络加速架构的具体案例讨论新兴图计算应用的加速架构设计.最后对该领域的前沿研究方向进行了总结,并放眼于未来探讨图计算加速架构的发展趋势.

关键词 图计算;图神经网络;加速架构;不规则访存;数据局部性;动态访存调度;负载均衡

中图法分类号 TP303

大数据时代,越来越多的数据采用图结构进行表示.图是一种能够表达对象之间复杂关系的数据存储方式,被广泛用于表示人际关系、分子拓扑结构、大脑神经元链接等.图数据中蕴含着丰富的信息,图计算应用是一种挖掘图数据中隐含价值的重要应用.为了快速处理图数据和应对不断增长的图数据,图计算应用被广泛部署于各大数据中心,成为数据中心的典型应用.

源于图的无结构特性,图计算应用在现有通用架构上无法被高效执行.现实生活中的图没有固定的结构,节点的出边分布极度不均匀,节点与节点之间的连接极为随机.由于图计算应用的执行行为依赖于图数据,图数据的以上特性导致图计算应用的执行行为非常不规则.这种不规则的执行行为导致现有的通用架构在计算、访存和通信 3 个方面都面临巨大挑战.在计算方面,计算单元面临负载不均衡、密集读改写更新等挑战,导致基于 CPU 和 GPU 的图计算软件框架的性能严重不足.在访存方面,不规则的细粒度访存导致 CPU 的 L2 和 L3 Cache 的命中率极低[1],Cacheline 利用效率低下,同时也导致了 GPU 的 SIMT(single instruction multiple threads)执行模型遇到了大量的访存歧义(memory divergence).在多节点计算、存储系统方面,不规则的细粒度通信,导致了大量无效通信和通信带宽浪费.

为了应对图计算应用带来的挑战,为图计算定制专用的加速架构是一种高效的解决方案.它能够为数据中心带来数百倍的性能提升和数千倍的能耗提升.图计算加速器的设计理念是根据图计算应用的操作特性改造硬件数据通路,量身定制计算流水线、内存子系统、存储子系统和通信子系统,从而为图计算应用的操作进行固化的硬件表达.近年来,大量的图计算加速架构设计被提出,从不同的角度采取多样的方法解决图计算应用的各项挑战.

为了让相关的研究人员对图计算加速架构的研究现状和发展方向有深入的了解,本文从现有工作出发探讨图计算加速架构设计面临的关键问题和主要解决方法.值得关注的是,本文还着重探讨了一种新兴的图计算应用,即图神经网络.该新兴图计算应用同时具有传统图计算应用和传统神经网络应用的执行特征,并且还具有与传统应用不同的计算和访存特征.例如,由于节点的属性是高维数据,所以节点的属性访问是粗粒度的不规则访问,与传统图计算的细粒度不规则访问不同.除此之外,本文还对图计算加速架构的前沿研究问题进行了归纳和总结,并放眼于未来探讨其发展趋势.本文的工作具有一定的指导作用,读者能够快速明白传统图计算应用加速架构和新兴图神经网络加速架构的设计要点、关键问题及对应的解决方案,了解目前图计算加速架构设计的趋势和机遇,并且将相应的概念和技术应用到未来的图计算加速架构的设计上.

现有调研文献[2]是基于硬件平台对现有的图计算加速工作进行分类,涵盖了现场可编程门阵列(field programmable gate array, FPGA)、3D-stacking、特定应用集成电路(application specific integrated circuit, ASIC)、GPU,目的是对每个工作的设计思想进行介绍.文献[3]基于图计算加速的主要技术(预处理、并行图计算和运行时调度)对现有工作分类.本文的分类方法与前人不同.本文从图计算加速架构的设计角度出发,基于计算机的金字塔组织结构[4],从上到下,根据图计算应用带来的挑战、衍生

的问题和解决方案对现有工作进行分类和总结,并为前人的调研工作补充了许多新的先进设计,以及加入了新兴的图神经网络加速架构的研究工作.除此之外,本文也从图计算加速架构的测试评估与全栈设计角度出发,对未来的研究方向进行了展望.

本文的主要贡献包括 3 个方面:

1) 以加速图计算应用遇到的关键挑战为导向,以解决方案为核心,基于计算机金字塔组织结构,从上到下,逐层对图计算加速架构的研究现状进行了系统的归纳和总结,并以具体例子分析了不同层次优化技术之间的关系.

2) 以具体图神经网络加速架构设计作为例子,着重介绍和总结了新兴图计算应用(图神经网络)与其特定的加速器设计.继神经网络加速架构之后,图神经网络加速器必将掀起新的研究和产业化热潮.

3) 从图计算加速架构评估与设计的角度对图计算加速架构进行了展望,指出了全栈式设计方案是实现产业化应用的关键,并阐述了基于 RISC-V 生态环境,有助于快速且低成本地实现图计算加速架构的设计方案.

1 图计算应用背景

在本节中,我们主要描述了图的特征和图的存储方式、主要的图计算编程模型、典型的传统图计算算法和新兴图神经网络模型.

1.1 图的表示与特征

图(graph)是一种用于表示对象之间联系的抽象数据结构,通常被定义为 $G=(V,E)$.每个对象在图中被定义为节点,V 表示节点的集合.每个对象间的联系在图中被定义为边,E 表示边的集合.集合 E 中每一条边 $e(v_i,v_j)$ 表示节点 v_i 与节点 v_j 的链接.根据边是否有方向和权值,图又可细分为有向/无向图和有权/无权图.对于有向图,边 $e(v_i,v_j)$ 表示节点 v_i 可以从自身(源节点(source vertex))出发到达 v_j 目的节点(destination vertex).节点的出度表示从该节点出发单跳能达到的节点数目,对应的边被称为出边.节点的入度表示从其他节点出发单跳能到达该节点的节点数目,对应的边被称为入边.这些直接链接的节点互相被称为 1-hop 邻居节点.

现实生活中的图大部分结构不固定、极为稀疏且节点度数服从 scale-free(power-law)分布,但局部却具有强集合结构(strong community structure)[5].由于现实场景的不确定性,对象间的联系具有极高的随机性,因此每个对象的链接数目和被链接的对象极为不固定.同时由于绝大部分对象之间是不存在联系的,因此现实生活中的图又是极度稀疏的.除此之外,图数据还存在相对少量的节点与大量其他节点相连的分布特征[6].例如大家熟知的名人效应,名人的被关注人数是非常多的,而大部分关注名人的人本身却只被少量的人所关注.同时,现实生活中的图,局部还存在各类群体等所体现出的强集合结构.

为了提高图数据的存储效率,非常多的数据结构被提出用于去除稀疏,压缩有效图数据.CSR (compressed sparse row)和 CSC(compressed sparse column)是其中使用最广泛的 2 种.这 2 种结构通常包含 3 种数组:偏移数组、边数组和节点属性数组.偏移数组中的元素指的是用于索引每个节点的边表(1-hop 邻居节点编号集合)在边数组中的偏移.边数组保存的是 1-hop 邻居节点编号,编号在每个节点的边表中是顺序的.CSR 格式的数组用于保存出边,而 CSC 格式用于保存入边.节点属性数组用于保存每个节点的属性.如图 1 所示,是 CSR 的数据格式.

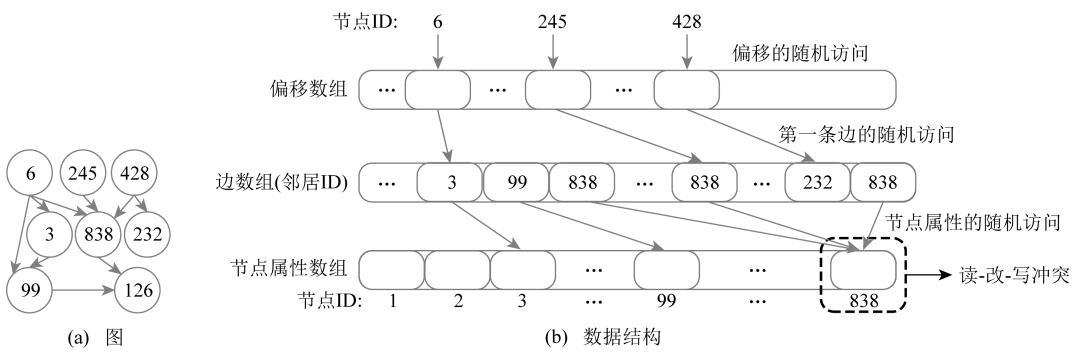


Fig.1 CSR representation

图 1 CSR 格式

1.2 图计算的编程模型

为了增强软件框架或硬件的表达能力,以及挖掘图计算应用的并行性,图计算应用专用的编程模型应运而生.主流的编程模型有 Vertex-centric 编程模型和 Edge-centric 编程模型,这 2 种模型都可以被 *GAS*(*Gather*,*Apply*,*Scatter*)[7] 模型统一表示.*Gather* 函数主要是根据每个节点的入边收集邻居的信息.*Apply* 函数主要是对收集到的信息进行加工以更新节点的属性信息.*Scatter* 函数是将更新之后的信息根据出边扩散到所有邻居节点.这 3 个函数会持续迭代执行,直到迭代次数达到指定次数或者计算结果完成收敛.

Vertex-centric 模型在文献[8]中被提出之后,广泛用于各种图计算加速架构中.如算法 1 所示,该模型的核心思想是"think like a vertex",即从每个源节点或者激活节点出发进行扩散操作,并将信息传递到邻居节点.每个激活节点的计算,以及属于每个激活节点的每一条边的计算和信息传递都是独立的,可挖掘的并行度非常高.Edge-centric 模型在文献[9]中被提出.如算法 2 所示,该模型的核心思想是基于边执行节点的计算.从每一条边出发,为该边的目的节点收集源节点的信息,并用该信息更新目的节点.由于所有的边处理可以同时执行,并行度也非常高.上述模型在实现中可具体表示为 *Process_edge*,*Reduce*,*Apply* 这 3 个依据算法可配的自定义函数.在遍历边的过程中,*Process_edge* 函数用于处理被遍历的边,获得边处理结果.例如在最短路径算法中,利用源节点的属性值和边上的权重计算 2 点的距离.然后边处理结果被送入 *Reduce* 函数中,与相应节点的临时节点属性进行比较.例如在最短路径中,完成前后 2 段距离的比较.最后,待所有的边遍历结束之后,*Apply* 函数利用节点临时属性和一些常量节点属性完成节点属性的正式更新.

Vertex-centric 和 Edge-centric 编程模型的主要区别在于访存行为和计算效率 2 方面.访存行为方面,Vertex-centric 编程模型在每一次的迭代中,从激活节点出发,利用激活节点的编号访问偏移数据,接着利用偏移在边数组中对边表进行访问,然后根据边表内的邻居节点编号对所有邻居节点的节点属性进行访问.由于需要利用激活节点编号对偏移数据进行索引并且需要利用偏移数据对边表进行索引,所以对于每个激活节点,其对偏移数据及边表第 1 条边的访问是不规则的,而对其剩余的其他出边的访问则是顺序的.需要注意是,对于全节点遍历算法,由于所有节点的邻居都会被访问,所以不存在对偏移数据和边数据的不规则访问.此外,利用邻居节点编号索引邻居节点的属性数据,则导致了间接且不规则的细粒度访存,图 1 展示了对节点属性数据的不规则访问.Edge-centric 编程模型则在每一次的迭代中遍历所有的边,执行过程能够以流(stream)方式连续访问所有的边数据.因此,Edge-centric 编程模型不产生对偏移数据和边数据的不规则访存,但对目的节点的访问仍然是不规则的细粒度访存.计算效率方面,Vertex-centric 编程模型在每一次迭代中只遍历上一次迭代被激活(被修改)节点的出边,而 Edge-centric 编程模型在每次迭代都需要遍历所有边,会造成许多无效计算和访存.

算法 1. Vertex-centric 编程模型.

① while 仍未结束
② *Scatter* 阶段:
③ for 每个节点 v do
④ if v 具有更新 u then
⑤ 通过 v 的出边发送 u 给邻居节点;
⑥ end if
⑦ end for
⑧ *Gather* 阶段:
⑨ for 每个更新 u do
⑩ if 更新条件满足 then
⑪ 更新节点 $u.dest$;
⑫ end if
⑬ end for
⑭ end while

算法 2. Edge-centric 编程模型.

① while 仍未结束
② *Scatter* 阶段:
③ for 每条边 e do
④ if 节点 $e.src$ 具有更新 u then
⑤ 发送 u 给节点 $e.dest$;
⑥ end if
⑦ end for
⑧ *Gather* 阶段:
⑨ for 每个更新 u do
⑩ if 更新条件满足 then
⑪ 更新节点 $u.dest$;
⑫ end if
⑬ end for
⑭ end while

除了 Vertex-centric 和 Edge-centric 这 2 种主流

的编程模型之外,还有一些其他的编程模型.例如,结合 Vertex-centric 和 Edge-centric 编程模型优点的混合编程模型[10]、以数据为中心的 Data-centric 编程模型[11]等.混合模型的核心是基于激活节点的数目,在每一个迭代动态选择 Vertex-centric 或者 Edge-centric 编程模型,以实现顺序访存和无效计算的 tradeoff.Data-centric 编程模型主要是将点和边都视为数据,并提出"图处理实际上是对数据修改"的概念,该模型为点和边数据的修改设计了基于 *Advance*,*Filter*,*Compute* 这 3 种函数的统一上层接口,用户可以直接定义具体的数据加工函数.Data-centric 模型不仅在 GPU 取得一定的性能提升,还提供了灵活的可编程性.

1.3 图计算应用

图计算应用主要包含了传统的图计算应用和新兴的图计算应用(图神经网络).

传统的图计算应用根据每个迭代是否遍历所有节点可以分成全节点遍历(stationary)算法和激活节点遍历(non-stationary)算法[12].Stationary 的典型算法是 PR(PageRank)、半径估计(diameter estimation)、弱连通分量(weakly connected components)等.由于节点在每个迭代都会更新,也即所有边都会被遍历,所以通常使用 Edge-centric 编程模型能够获得更好的性能.Non-stationary 的典型算法是宽度优先搜索(breadth first search, BFS)、深度优先搜索(depth first search, DFS)、单源最短路径(single source shortest path, SSSP)、单源最宽路径(single source widest path, SSWP)和连通分量(connected components, CC)等.每个迭代过程都从激活节点出发,为了减少冗余计算,一般使用 Vertex-centric 编程模型能够获得更好的性能.对典型算法的具体介绍为:

BFS 是最基本的图搜索算法之一.如图 2(a)所示,它从选定的起始节点(根节点)出发,沿着图的宽度方向遍历图中的所有节点.

DFS 也是最基本的图搜索算法之一.如图 2(b)所示,它从选定的起始节点(根节点)出发,沿着图的深度方向遍历图中的所有节点.

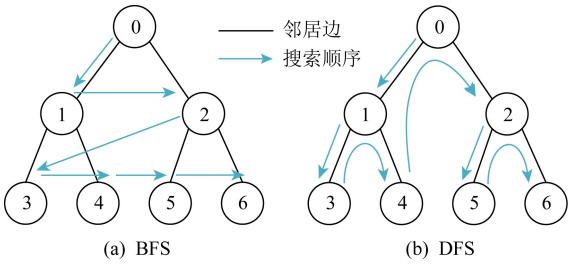


Fig.2 Traditional graph processing examples

图 2 传统图计算应用例子

SSSP 算法是最基本的寻找最短路径的算法之一.它从选定的根节点出发,在权重图中寻找到达图中所有节点的最短路径.主要的实现方式有 Dijkstra 算法和 Bellman-Ford 算法.

SSWP 算法是从选定的根节点出发,在权重图中寻找到达图中所有节点的路径,并使路径中的最小权重边缘最大化.最宽路径问题在网络路由问题、数字合成和投票理论等领域有着广泛的应用.例如,在 2 个节点之间寻找具有最大传输速度的路由.

PR 算法由 Google 公司提出,用于对搜索引擎搜索结果中的网页进行排名.其中,图中的节点表征网页,而图中的边表征超链接.

CC 算法是在图中寻找子图,且需保证该子图内任意 2 个节点是联通的.

表 1 是以上部分算法在 Vertex-centric 模型上的具体实现[13].对于 PR 算法,其中 res 和 $v.deg$ 分别表示边处理的结果和节点的常量属性,α 和 β 分别表示常量系数且 $\alpha+\beta=1$.

Table 1 Algorithm Implement in Vertex-centric Model

表 1 算法在 Vertex-centric 编程模型上的具体实现

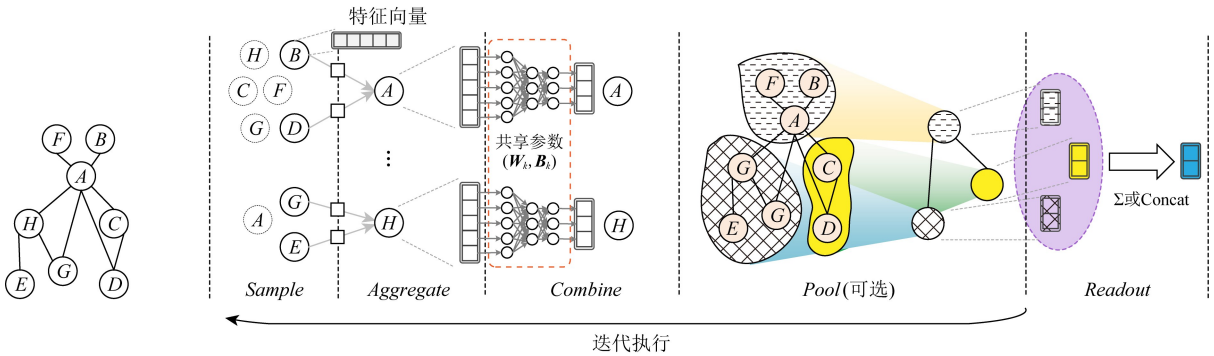
算法	*Process_edge*	*Reduce*	*Apply*
BFS	$u.prop+1$	$\min(v.tProp,res)$	$\min(v.prop,v.tProp)$
SSSP	$u.prop+e.weight$	$\min(v.tProp,res)$	$\min(v.prop,v.tProp)$
CC	$u.prop$	$\min(v.tProp,res)$	$\min(v.prop,v.tProp)$
SSWP	$\min(u.prop,e.weight)$	$\max(v.tProp,res)$	$\max(v.prop,v.tProp)$
PR	$u.prop$	$v.tProp+res$	$(\alpha+\beta\cdot v.tProp)/v.deg$

注:边 $e=(u,v)$中,u 代表源节点,v 代表目的节点; res 和 $v.deg$ 分别代表 $edgeProResult$ 和 $v.cProp$;$v.prop$ 和 $v.tProp$ 分别代表节点的属性和临时属性;α 和 β 是常量.

新兴的图计算应用主要是各种图神经网络模型(graph neural network, GNN).由于图数据的爆发和深度学习的广泛应用,各种图神经网络模型被提出用于分析图数据.例如用于节点和图分类的GCN[14],GraphSage[15],GINConv[16]等模型.图数据中固有的不规则链接给传统的深度学习算法(如卷积神经网络CNN等)带来了巨大的挑战,掀起了图神经网络的研究热潮.图神经网络的输入数据来自非欧几里德(non-Euclidean)领域并通常用能够表示对象间复杂联系的图来表示.因此,图神经网络能够应对许多关键且从前无法高效处理的问题,例如洞察大脑神经元链接[17]和发现新材料[18]等问题.然而,传统的神经网络通常以固定大小且规整的图片或者文本序列等作为输入,无法处理任意大小且节点无序的图数据.

图神经网络首先将图数据转变为低维空间的数据并同时保留图的结构以及原始的节点属性信息;接着通过神经网络进行后续的训练和推断.图神经网络主要包含2个核心阶段Aggregation和Combination阶段.在Aggregation阶段中,每一个节点都遍历各自的邻居节点并执行*Aggregate*函数聚合邻居节点的属性信息.在Combination阶段中,每一个节点的聚合结果都会被*Combine*函数利用神经网络进行变换,以获得新的属性数据.

图3是图神经网络中的一个典型分支图卷积神经网络的图解.*Sample*函数用于对1-hop邻居节点进行采样,目的是减少算法模型的计算量.*Aggregate*函数则是收集邻居节点的信息,产生中间计算结果.*Combine*函数负责利用神经网络对中间结果进行变换和降维,并获得新的节点属性向量.*Pool*函数是可选函数,完成与传统卷积神经网络中*Pooling*函数相同的功能,区别在于*Pool*函数处理的对象是图.*Readout*函数执行合并操作,将所有节点的属性向量进行element-wise的累加操作.它的作用是产生单个属性向量作为其他神经网络模型的输入,以用于图的分类.



(a) 输入图 (b) 卷积层

Fig.3 Illustration of graph convolution neural network (GCN) model[19]

图3 图卷积网络(GCN)模型的图解[19]

如算法3所示,*Aggregate*函数与GAS模型中的*Gather*函数的作用是一致的,根据入边收集邻居节点的信息.而*Combine*函数也和*Apply*函数的作用是一致的,对节点的属性(属性向量)进行更新.所以新兴的图神经网络算法也能够用Vertex-centric模型来表示.典型的图神经网络模型为:

GCN[14]是最为成功的图神经网络模型之一,它在基于频谱的图卷积和基于空间的图卷积之间架起了桥梁,让基于频谱的图卷积理论应用于基于空间的图卷积研究中.

GraphSage[15]进一步采用均匀近邻采样方法减少接收域的扩展(receptive field expansion),提供了权衡预测精度和执行时间的方案.

GINConv[16]是一种极为简单的图神经网络结构,其判别能力等同于Weisfeler-Lehman图同构检验(Weisfeiler-Lehman graph isomorphism test).GINConv学习的顶点特征可以直接用于节点分类和链路预测等任务.

DiffPool[20]提供了一个通用的模型来实现图的*Pool*函数.它可以插入到任意图神经网络中,目的是将原始图转换为更小的图.实际上,DiffPool是使用2个额外的图卷积神经网络来实现图的*Pool*函数.

算法3. GNNs编程模型.

① 初始化 $SampleNum$;

② 初始化***SampleIndexArray***;

③ for 每个节点 $v \in V$ do

④ ***agg_res***←*init*();
⑤ ***sample_idxs***←***SampleIndexArray***[*v.nid*];
⑥ for 每个 *sample_idx* in ***sample_idxs*** do
⑦ *e*(*u*,*v*)←***EdgeArray***[*sample_idx*];
⑧ ***agg_res***←*Aggregate*(***agg_res***, ***u.feature***);
⑨ end for
⑩ ***v.feature***←*Combine*(***agg_res***, ***weights***, ***biases***);
⑪ end for

传统图计算应用和图神经网络应用的相同点在于它们的每一次迭代或者层的执行都包含了遍历邻居和更新节点属性 2 个阶段:1)无论是传统图计算应用还是新兴的图神经网络都通过遍历边收集 1-hop 或者多 hop 邻居的信息作为中间结果;2)中间结果会被用于更新节点属性.

传统图计算应用和图神经网络的区别有 3 点:

1) 如表 2 所示,在传统的图计算应用中,节点属性一般只有单个元素,且在计算过程中只有值在变而元素数量不变.然而如表 3 所示,在图神经网络中,节点属性一般是一个向量或者是更高维的张量数据,元素数量在数千个以上.除此之外,不同图数据集的维度和元素个数不尽相同,并且在执行的过程中,由于神经网络的变换,不仅节点的属性值在变,元素个数和维度也在变化.

2) 传统图计算收集的邻居信息是单个元素,并对每个元素进行归约操作.而图神经网络收集的则是向量或者更高维的数据,对每个元素进行 element-wise 的归约操作.

3) 传统的图计算应用的更新操作较为简单,一般是累加或者比较等操作,所以计算访存比低.然而,在图神经网络中,每个节点的属性更新操作是一个神经网络,并且所有的节点共享同一个神经网络,具有非常高的计算访存比.

Table 2 The Popular Datasets for Traditional Graph Processing

表 2 传统图计算应用的典型数据集

数据集	节点个数	特征长度	边数
Flickr	0.82×10^6	1	9.84×10^6
Pokec	1.63×10^6	1	30.62×10^6
LiveJournal	4.84×10^6	1	68.99×10^6
Hollywood	1.14×10^6	1	113.90×10^6
Indochina-04	7.41×10^6	1	194.11×10^6
Orkut	3.07×10^6	1	234.37×10^6

Table 3 The Popular Datasets for Graph Neural Network

表 3 图神经网络的典型数据集

数据集	节点个数	特征长度	边数
IMDB-BIN	2 647	136	28 624
Cora	2 708	1 433	10 556
Citeseer	3 327	3 703	9 104
COLLAB	12 087	492	1 446 010
Pubmed	19 717	500	88 648
Reddit	232 965	602	114 615 892

2 图计算应用加速的挑战

在本节中,我们主要从计算、访存和多节点通信 3 个方面介绍图计算应用加速面临的挑战.

2.1 对计算的挑战

在计算方面,传统图计算加速主要面临的挑战是不规则负载和密集的读改写更新操作,而图神经网络加速主要面临的挑战是如何高效挖掘大量的并行性和大量的可共享数据.

不规则负载源于图的无结构特性和 scale-free 特性[12].每个激活节点连接的邻居节点数目不固定且分布非常不均匀,小部分的节点具有非常多的邻居节点需要处理,而大部分的节点则只具有少量的邻居节点需要处理.工作量少的节点需要等待工作量大的节点,巨大的工作量差异导致计算部件的利用率大大下降.例如,文献[6]指出了在 GPU 上执行传统图计算应用时,GPU 的利用率仅有 25.3%~39.4%.

密集的读改写更新操作源于大量的处理单元同时更新同一个节点的属性数据.1 个目的节点可能被上千个甚至上万个激活节点连接,该目的节点的属性数据可能会被多个执行单元或者线程同时更新.为了维护处理结果的正确性,必须保证更新操作的原子性.因此需要将并行的更新操作顺序执行,导致计算流水线产生了大量的空泡.文献[21]指出原子操作导致了 96%以上的性能下降.

大量待挖掘的细粒度并行性存在于图神经网络应用中,如何对此进行高效挖掘是一项极具挑战性的任务.细粒度的并行性源于同一节点的 *Aggregate* 函数和 *Combine* 函数可以流水执行,并且不同节点的处理可以同时执行.通常来说,在 Aggregation 阶段,所有的节点可以同时根据边表数据执行 *Aggregate* 函数,收集邻居节点的属性向量,并且 *Aggregate* 函数对每个属性向量的元素单独操作,所以不仅具有

edge-level parallelism 还具有 intra-vertex(element-wise)parallelism.在 Combination 阶段,所有的节点可以同时进行基于神经网络的变换操作,不仅具有 inter-vertex parallelism 还具有 matrix-vector multiplication parallelism.除此之外,每个节点的 *Aggregate* 函数和 *Combine* 函数可以进行流水执行,也就是 Aggregation 阶段和 Combination 阶段可以实现融合执行(inter-phase fusion[19]).

传统的 CPU 是 latency-oriented 架构,倾向于利用复杂的逻辑结构减少计算任务的运行时间.CPU 的并发线程数和处理核数有限,对并行性的挖掘能力有限,因此无法挖掘如此富裕的并行性.GPU 是 throughput-oriented 架构,倾向于提高整体的吞吐量,同时并发上万条线程.GPU 架构通常对粗粒度并行性的挖掘支持甚好.例如 NVIDIA 基于它们的硬件架构提出了 cuBLAS 库,用于挖掘矩阵乘等粗粒度操作的并行性.因此,利用这些硬件优化的矩阵乘操作能够有效地执行 Combination 阶段.但是,Combination 阶段和 Aggregation 阶段需要单独执行,无法实现 2 阶段的融合执行,即 2 阶段无法流水执行,也造成了大量对中间结果数据的冗余访问.如果要在 GPU 实现 2 阶段的融合执行,则需要细粒度的同步机制.然而,在无法高效支持细粒度同步的 GPU 上,挖掘细粒度并行性的同步开销和通信开销极大.此外,在 Aggregation 阶段中,由于每个节点的邻居数目不均匀,因此每个节点的计算图通常是不一样的[19],导致了动态且不规则的计算.然而,专注于挖掘规则计算的 GPU,并不能高效支持这种动态且不规则的并行计算.

大量可共享的数据存在于图神经网络的 Combination 阶段中,如何利用该机会提高执行效率也是一项极具挑战性的任务.由于所有节点都基于同一个神经网络完成节点属性向量的变换,因此神经网络的权重参数被共享于每一个节点的 *Combine* 函数中.如果这些可共享数据能够被高效的数据流支持,在计算单元之间被高效地复用,从而减少对片上存储的频繁且冗余的访问,则能大大提高计算效率.

然而,由于权重矩阵会被所有节点共享,因此需要将权重数据共享于多个处理核或者线程中,导致了大量的数据拷贝操作,造成了极大的通信开销.例如在 Intel Xeon CPU 上造成了 36%的额外执行时间[19].即使是在同一个处理核内,处理核内的计算单元之间也无法像 Google TPU[22]那样以 weight-stationary 方式复用权重数据.

2.2 对访存的挑战

访存的挑战,主要在于传统图计算应用的间接且不规则的细粒度访存和图神经网络的间接且不规则的粗粒度访存.

间接且不规则的细粒度访存来源于图的无结构特性和细粒度的数据更新.传统图计算应用中,由于每个节点的邻居节点和数目均不规则,邻居节点属性数据存储的位置不规则,以及传统的图计算应用更新的数据一般仅在 4~8 B,因此在遍历邻居节点的过程中产生了大量的间接且不规则的细粒度访存,导致传统内存子系统变得非常低效.传统的数据预取器因为缺少访存的可预测性,难以实现高效预取.传统 Cache 替换策略无法挖掘数据的时间局部性,1 个 32 B 或以上的 Cacheline 平均只有 6 B 左右的数据被使用后就会被替换出 Cache,导致 Cacheline 的利用效率非常低[23-24].另外,片外访存的效率也极低,原因在于 DRAM 数据请求的粒度也为 32 B 或以上且 DRAM 的 Row Buffer 大小在 4 KB 以上[23-24].

间接且不规则的粗粒度访存来源于图的无结构特性和粗粒度的数据更新,但与传统图计算应用的不规则访存具有不一样的特征.在图神经网络中,节点的属性是 1 个具有上百个元素的特征向量,大小达到上千个字节以上,需要多个 Cacheline 才能缓存下,具有一定的空间局部性,所以 Cacheline 的利用率非常高[25].然而也正由于特征向量过大,在同等容量的 Cache 中,只能缓存少量的节点属性向量,导致数据复用距离(reuse distance)严重增加.因此,在邻居节点信息收集的过程中,特征向量被频繁地替换,Cache 的缺失率非常高.虽然片外访存的粒度较大,能够有效地挖掘 DRAM Row Buffer 的数据局部性,但是同样因为频繁的替换导致了大量的片外访存,需要更多的带宽[25].

2.3 对多节点通信的挑战

多节点通信方面,主要挑战来自传统图计算应用的不规则细粒度通信和来自图神经网络的不规则粗粒度通信.

不规则的通信源于节点属性数据分布在不同的子系统中,不规则的遍历导致了不规则的通信.无论在传统图计算应用中还是在图神经网络中,由于单节点系统无法存储甚至是无法处理大规模的图数据,所以需要多个乃至上百个节点子系统一起完成存储和处理任务.因为节点之间的链接是不均匀的,所以不存在一种算法能够把图数据很好地划分到不同的子系统中并实现无通信执行.在执行过程中,

被遍历的节点是无法被提前预测的.它们的节点属性被分布在未知的子系统中,只有在获取边数据那刻才知道数据被存放到哪个子系统中.这种随机性导致了大量的不规则通信,严重影响了多节点系统的性能和执行效率[7,26-27].除此之外,传统图计算的数据更新粒度较细,而现有的通信系统都是为传输粗粒度数据而设计的,导致了现有的通信系统通信效率非常低[28].

3 基于通用架构的图计算框架简介

为了让读者更容易理解图计算加速架构的设计,我们先对基于传统通用架构的图计算框架进行简单的介绍.图计算应用具有与常见应用不一样的执行行为模式,对强调通用性的通用架构十分不友好,所以图计算系统需要为图计算应用做各种特定的软件优化.这些软件优化主要是图数据划分方案和预处理等优化技术.

基于 CPU 通用架构研发面向传统图计算应用的框架,主要是利用 CPU 中大容量的 Cache 和大容量的内存(DRAM)等优势,以处理更大规模的图并减少对存储(storage)的访问.主要的代表性工作有 Pregel[8], PowerGraph[7], GraphMat[29], GraphChi[30] 等.Pregel 框架提出了 Vertex-centric 的编程模型和相应的软件框架,以挖掘细粒度的并行性和提高编程效率.PowerGraph 框架提出了基于节点度数感知的图数据划分方法以解决幂律(power-law)图对计算、存储和通信的压力,PowerGraph 还提出了 GAS 模型和 Delta-Caching 机制以高效支持更多图计算算法;GraphMat 框架提供前端利用在线预处理技术将图计算的处理转变为基于代数运算的操作,以挖掘 CPU 对代数运算的硬件优化;GraphChi 利用离线预处理将图数据分成 Interval 和 Shard,然后在运行时利用 Parallel Sliding Windows 在线预处理技术复用图数据,以提高内存容量和存储带宽的利用率,从而以单节点实现比分布式系统更高的性能.

基于 GPU 通用架构研发面向传统图计算的框架,主要是为了利用 GPU 的高带宽片外存储挖掘图计算应用的内存级别的并行性,以及利用 GPU 的万线程挖掘图计算的计算并行性和掩盖访存延迟.代表性工作主要有 Tigr[6], Gunrock[11], CuSha[31] 等.Tigr 致力于利用离线预处理技术将结构不规则的图转变为结构规则的图,以均衡线程之间的负载.Gunrock 提出了 Data-centric 编程模型,并给出基于 *Advance*, *Filter*, *Compute* 这 3 种函数的抽象.通过该抽象可以让 Gunrock 实施各种在线预处理优化方案以均衡负载和去掉冗余计算等.Cusha 则提出 G-Shards 和 Concatenated Windows 这 2 种新的图数据表示方式,并利用预处理构造上述 2 种数据结构.G-Shards 和 Concatenated Windows 这 2 种结构有助于 GPU 实现合并的访存操作,以减少访存歧义.

基于传统架构设计的图神经网络框架大部分都是基于 GPU 设计的,目的是利用 GPU 中富裕的计算资源满足图神经网络对计算资源的需求.主要的代表性工作有 NeuGraph[32], PyTorch Geometry[33], DGL(deep graph library)[34] 等.NeuGraph 是第 1 个既高效又具有高拓展性的图神经网络框架.NeuGraph 弥合传统图计算系统与传统神经网络系统的鸿沟,实现了高效的图神经网络框架.NeuGraph 通过细粒度图分块方案将图数据分成若干个子图,并在子图的处理之间构建数据流,以高效支持多 GPU 训练和挖掘并行性,从而大大增强了框架的可拓展性.DGL 利用消息通信机制将传统的神经网络系统进行重新封装,实现了图神经网络框架.PyTorch Geometry 则利用基于 GPU 硬件优化的 *Scatter* 函数执行 Aggregation 阶段和矩阵乘法执行 Combination 阶段,并提供通用的消息编程模型以支持更多的新模型.然而 DGL 和 PyTorch Geometry 无法高效支持多 GPU 训练,并不具备较强的拓展性.

以上的图计算系统能够高效挖掘现有通用架构的优势.然而,为了支持图计算特有的不规则执行行为,预处理的开销非常大.例如基于 CPU 的 GraphMat 则需要昂贵的转换开销将图操作转换为代数运算.基于 GPU 的传统图计算框架 Tigr 的离线预处理时间达到了处理时间的 1 倍以上.基于 GPU 的传统图计算框架 Gunrock 的在线预处理时间则达到了处理时间的 2 倍以上.基于 GPU 的图神经网络框架面临由不规则访存导致的昂贵原子操作和片上与片外之间的频繁数据替换[25].此外,由于通用架构的计算流水线、内存子系统、存储子系统和通信子系统着眼于通用性,无法对图计算应用的执行语义做出直接且专一的优化,例如无法控制 Cache 实现感知节点度数(degree-aware)的替换策略等.而且许多为了支持通用性的硬件设计,例如 CPU 的乱序执行,导致面积和能耗开销极大,无法满足数据中心对面积和能耗的需求.因此为了达到高性能和高能效,面向图计算加速的专用架构应运而生.

4 图计算加速架构设计理念与分类方法

本节首先介绍图计算加速架构的设计理念,接着介绍本文的分类方法.

4.1 图计算加速架构的设计理念

为了解决第 3 节所述的挑战,学术界和产业界均开展了大量面向图计算应用的专用架构研究工作.图计算加速架构的出现具有必然性,其主要的 3 个原因是:1)图计算被广泛应用;2)图数据爆发;3)市场需要性能更高、能耗更低的解决方案.

图计算在众多领域得到广泛应用,并运行于各大数据中心[35-37].随着图数据规模的不断扩增,基于通用性设计理念的传统架构无法满足图计算应用的需求,不能高效解决图计算应用所面临的诸多挑战.因此需要一种能够降低数据中心能耗开销和散热成本,并提高数个量级性能的解决方案,即面向图计算应用的专用加速架构.

图计算加速器的设计理念是根据图计算应用的操作改造硬件数据通路,量身定制计算流水线、内存子系统、存储子系统和通信子系统,从而为图计算应用的操作进行固化的硬件表达.1)对流水线进行定制优化,根据图计算应用的执行行为直接用固定的硬件逻辑处理程序的控制流和数据流.2)通过修改内存子系统或者存储子系统对访存操作进行直接表达.3)逐渐拉近计算单元与数据存储单元之间的距离,从而获得更高的带宽和更小的访存延迟.4)通过定制的多节点系统拓展机制和通信接口实现多节点图计算加速系统.

4.2 分类方法

本文以加速图计算应用遇到的关键挑战为导向,以关键问题的解决方案为核心对现有工作进行系统归纳.具体而言,从计算机金字塔组织结构[4]出发,从上到下,根据每一层遇到的挑战归纳和介绍现有工作.

1) 传统图计算应用加速技术分类.如图 4 所示,从金字塔的顶端出发,首先在计算单元层次上遇到的挑战是不规则负载和密集的读改写更新操作,导致了传统架构出现了负载不均衡和昂贵的原子开销等问题.现有加速架构的解决方案的核心思路是将负载重新映射到不同的处理单元或者处理逻辑上,以及减少读改写更新操作和原子操作开销.

在片上存储层次上遇到的挑战是间接且不规则的细粒度访存,导致了传统处理器产生 Cache 命中率低和 Cacheline 利用率低的问题.现有解决方案的核心思路是通过大容量片上存储保存更多被细粒度访问的数据并提供细粒度的访存支持,或是为 Cache 定制访存调度单元动态调度访存,以挖掘数据局部性.

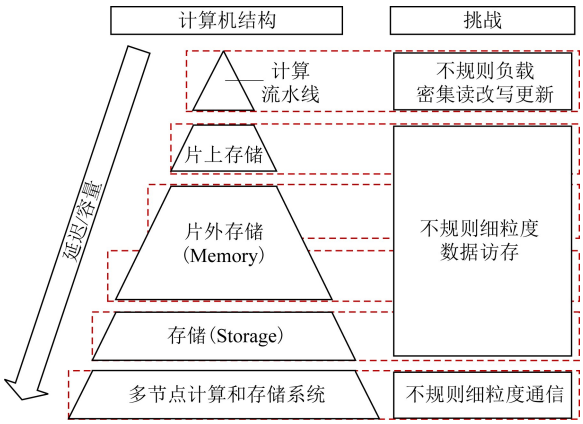


Fig.4 Computer organization pyramid and the challenge imposed by traditional graph processing

图 4 计算机金字塔组织结构与传统图计算面临的挑战

在片外存储层次上遇到的挑战同样是间接且不规则的细粒度访存,导致了传统处理器产生了片外带宽利用率低和利用效率低的问题.现有的主要解决方案是基于访存依赖定制专用预取器,或是基于图计算应用的访存模式修改 DRAM 的设计.

在近内存层次中,需要应对的挑战是图计算应用的受限于内存带宽(memory bandwidth bound)和受限于内存延迟(memory latency bound).现有解决方案的核心思路是通过将计算单元集成于 DRAM 中或将 ReRAM 作为计算单元,从而获取大量的 DRAM 带宽和缩短数据传输延迟.

在存储(storage)层次上,遇到的挑战同样是间接且不规则的细粒度访存.细粒度的访存导致了只能执行粗粒度访存的 Flash 存储设备效率极低.现有工作的主要解决方案是将图计算应用的关键操作卸载(offload)到 Flash 内部或者 Flash 控制器上执行,并对访存进行合并,以提高访存效率.

在多节点存储层次上,遇到的挑战是不规则的细粒度通信.不规则的细粒度通信导致了多节点系统产生了非常多的无效通信而且大大降低了现有粗粒度通信系统的通信效率.现有解决工作的核心思路是基于数据划分方法将计算局部化或者通过排序合并细粒度通信.

根据上述分类方法,现有应用于传统图计算应用的加速技术可如表 4 进行归纳分类.

Table 4 Accelerating Technique Classification for Traditional Graph Processing

表 4 现有应用于传统图计算应用的加速技术分类

系统	层次	挑战	相关工作
单节点系统	计算流水线	不规则负载	Graphicionado[24], AsynACC[38], GraphDynS[13], GraphH[39]
		密集读改写操作	Graphicionado[24], GraphDynS[13], AccuGraph[40], AsynACC[38], GraphPIM[41], GraphH[39], FPGP[42], ForeGraph[43]
	片上存储	不规则的细粒度访存	ForeGraph[43], HATS[44], PHI[45], GRASP[46], Graphicionado[24], GraphH[39], GraphDynS[13], Yan et al.[47]
	片外存储	不规则的细粒度访存	DROPLET[48], Tesseract[49], GraphDynS[13], GraphiDe[50], HyVE[51]
	近内存	不规则的细粒度访存	Tesseract[49], GraphP[26], GraphQ[27], GraphH[39], GraphR[52], GraphSAR[53], GRAM[54], RaGRa[55], RPBFS[56], GraphPIM[41], PEI[57]
	存储(storage)	不规则的细粒度访存	ExtraV[58], GraFBoost[59], GraphSSD[60], FlashGraph[61], Elyasi et al.[62]
多节点系统	通信接口	不规则的细粒度通信	Tesseract[49], GraphP[26], GraphQ[27], GraphH[39]

2) 图神经网络加速技术分类.如图 5 所示,从金字塔的顶端出发,首先在计算层次遇到的挑战是如何高效挖掘大量并行性.受限于计算资源或是线程同步开销,传统架构无法提供轻便且高效的并行执行方式.现有解决方案的核心思路是利用脉动阵列(systolic array)减少权重数据的共享开销.

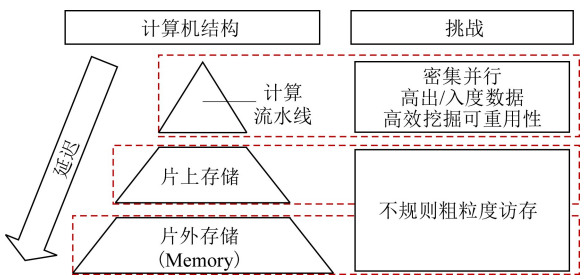


Fig.5 Computer organization pyramid and the challenge imposed by graph neural network

图 5 计算机金字塔组织结构与图神经网络面临的挑战

在片上存储层次遇到的挑战是不规则的粗粒度访存,导致了片上存储命中率低.目前主要的解决思路是利用大量片上存储和批量处理技术挖掘粗粒度访存数据的空间局部性和时间局部性,减少频繁的数据替换.

在片外访存的层次,遇到的挑战仍然是不规则的粗粒度访存,导致了片外预取效率低和带宽利用效率低等问题.现有解决方案的核心思路是基于特定的图数据划分方法提高预取效率,通过消除稀疏等技术提高带宽的利用效率.

5 传统图计算应用的加速方案

本节基于计算机金字塔组织结构,从上到下,根据每一层遇到的挑战归纳和介绍现有工作.

5.1 计算流水线优化

计算流水线的主流优化方向是均衡负载和降低读改写更新开销.

1) 负载均衡.负载均衡方案可以分为直接映射方式和感知节点度数映射方式 2 类.

① 直接映射方式.这种方式直接根据节点的编号进行简单的 Hash 映射,然后根据 Hash 映射的结果将负载分配到对应的处理逻辑上.这类方案简单且易实现,开销也非常小.例如,Graphicionado 直接利用节点的编号对流水线的总数进行取余操作,根据余数将节点的处理任务映射到相应的流水线.虽然每条流水线的平均负载在整体上是大致相等的,但是在局部时间内是不均衡的.原因在于被遍历节点的编号非常不规则,无法保证在局部时间内每条流水线都能被分配到工作.除此之外,Scatter 阶段的更新执行方式导致 Graphicionado 流水线的前端和后端之间也出现了负载不均衡.原因在于在 Scatter 阶段,从每个源节点出发遍历出边,被访问更新的目的节点数目通常为十几个以上,甚至高达上万个.因此,后端处理的目的节点数目是前端处理的源节点数目的十几倍以上.

直接映射的方法直接绑定了节点和流水线的对应关系.虽然这种一对一的固定负载分配方式实现简单,但是不能全面解决负载不均衡问题,无法动态适应在迭代过程中不断变化的负载.

② 感知节点度数映射.为实现动态负载的调度,大部分的设计都从导致负载不规则的起因出发,即不均匀的节点度数分布.如图 6 所示,GraphDynS 通过定制负载派遣单元(dispatching element, DE)来根据节点的出度进行动态任务派遣,消除了处理逻辑和负载的固定绑定关系.并同时根据源节点和目的节点数量的比值,对流水线的后端处理单元

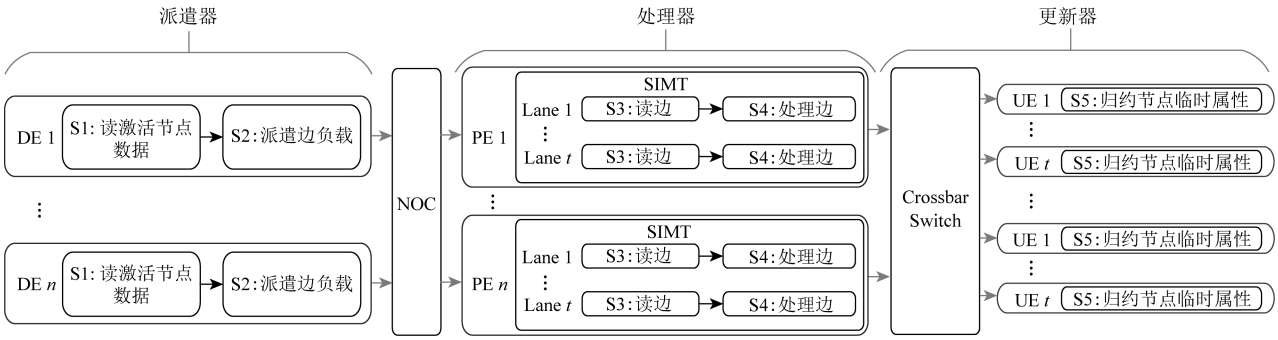


Fig. 6 The degree-aware workload balance design of GraphDynS[13]

图 6 GraphDynS[13]的感知节点度数的负载均衡设计

(processing element, PE)进行 SIMT 的拓展.每个 SIMT 槽完成 1 个目的节点的更新操作,以此平衡同一流水线前端和后端的负载.

2) 降低读改写更新开销.主要的研究方向有消除读改写冲突、防止读改写冲突和无阻塞原子操作.由于遍历的不规则,节点的更新无法预测.为防止出现读改写冲突,数据的更新需通过原子操作进行保护.原子操作的核心是顺序执行冲突的操作,因此会导致并行性缺失.

① 消除冲突主要是通过在线或者离线预处理对访问目的节点属性数据的更新操作进行排序和分块以消除冲突.GraphH, FPGP, ForeGraph 将节点重新划分成多个不相交的块.因为每个块之间是不相交的,所以块之间可以无冲突并行执行,但是块内仍然需要顺序执行.这类方法会导致非常昂贵的预处理开销,开销甚至比实际的执行时间、数据访问量和计算量更多.原因在于,该类方法需要反复访问整个图数据以及高开销的排序操作.

② 防止冲突主要是通过监测流水线的执行,防止对相同节点的更新操作同时进行.Graphicionado 和 AsynACC 利用 CAM(content addressable memory)定制查询电路,通过流水级之间的消息确认和阻塞机制,防止多个处理单元同时更新相同的节点.该类方法能够保证节点数据更新的正确性,但是限制了并行性的挖掘,且需要高功耗的 CAM 以及用于缓存节点更新数据的待发射队列.

③ 降低原子开销主要通过定制的原子操作部件完成原子操作,同时降低流水线的阻塞时间.GraphDynS 通过定制原子执行流水线来减少控制开销并实现无阻塞的原子更新.如图 7 所示,GraphDynS 通过比较流水级中操作数和对应操作数的地址,根据流水级中地址的比较结果选择最新的操作数送入到 ALU 中,从而实现无阻塞的原子更新.AccuGraph 则利用图计算算法中属性数据更新的单调性,设计了并行执行的累加器,实现了大量原子操作的并发执行.上述方法虽然能够减少流水线阻塞,并保留程序

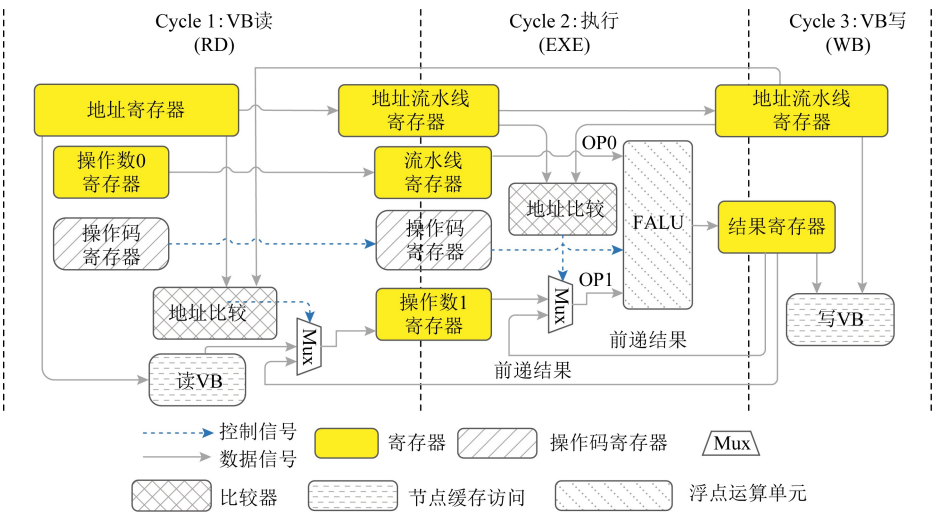


Fig. 7 Zero-stall atomic execution pipeline[13]

图 7 零延迟原子执行流水线[13]

的执行并行度,但是需要较大的片上缓存保证快速获得操作数,否则流水线仍然会因为等待数据产生大量阻塞.GraphPIM 将原子操作卸载(offload)到 HMC(hybrid memory cube)DRAM 存储上,并利用 HMC 对原子操作的支持在数据所在地进行 in-place 的原子操作,从而减少数据传输开销和缩短原子操作路径.然而,这种方法会增加额外的片外内存请求,并且不规则的细粒度内存请求会导致带宽浪费和极低的访存效率.

5.2 片上存储优化

片上访存优化的目的是挖掘节点属性数据的局部性,减少高延迟的片外访存.主要方法有 2 种:1)大容量片上存储,并辅以相应的图数据分块方法;2)动态探测局部性并调度访存.

1) 大容量片上存储.Graphicionado, GraphH, GraphDynS 等工作首先对数据进行分块,然后将数据分批读入到大容量的 eDRAM 进行处理.访问 eDRAM 的延迟比访问片外 DRAM 少 2 个量级以上,并且 eDRAM 支持细粒度的内存访问,因此不规则访存的数据能够被快速且高效地获取.这类方法需要批量切换数据分块,切换过程需要较长的数据等待时间.虽然可以使用 ping-pong buffer 机制来掩盖数据分块切换的延迟,但是片上存储的开销会变得非常大,例如 Graphicionado 的 64 MB eDRAM 占用的片上存储面积就高达 44 mm^2(32 nm 工艺)[47,63],导致的能耗开销就达到了整个加速设计的 90%.

2) 动态调度访存.该类方法主要面向传统架构的 Cache,核心是调度访存的顺序或定制 Cacheline 替换策略.

如图 8 和图 9 所示,HATS 利用图的强集合结构特性设计了基于深度优先算法的访存调度引擎来挖掘节点数据的时间局部性.PHI 为传统架构的 Cache 增加额外计算部件来缓存和处理图计算中的 *Scatter* 函数,从而减少无数据局部性的数据在处理核心与 Cache 之间的双向流动,大大节省了 Cache 到处理核心的带宽.GRASP 首先利用预处理对节点按照度数排序,目的是将高出度节点的数据集中在连续的地址空间,然后通过地址空间识别缓存高出度节点的属性数据的 Cacheline,从而阻止高复用的节点属性数据因 Cache thrashing 被切换出 Cache.

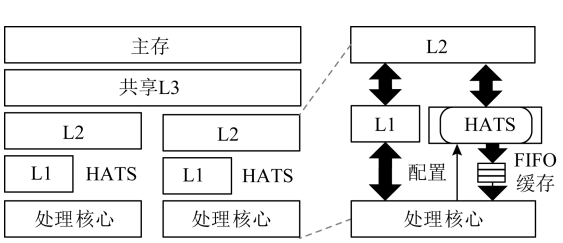


Fig. 8 HATS engine in each core

图 8 每个处理器核中的 HATS 引擎

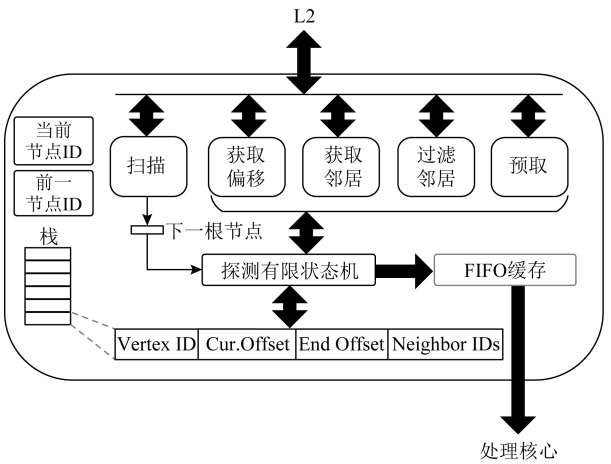


Fig. 9 Micro architecture of HATS[44]

图 9 HATS[44] 的微架构说明图

上述方法需要对现有通用处理器的 Cache 结构进行修改.从实际应用的角度考虑,不仅难以获得 Intel, ARM 等公司的授权进行修改,而且由于现有通用处理器结构的复杂性,修改 Cache 结构并定制辅助指令的工作量不亚于重新定制独立的加速架构,且部分硬件设计还需要昂贵的在线预处理支持.

除上述工作之外,也有工作是对现有加速架构的片上储存进行优化.文献[47]通过对 Vertex-centric 编程模型进行优化,利用激活节点和偏移的一一对应关系,通过未利用的带宽提前获取偏移数据,为下一次迭代的执行消除了对偏移数据的不规则访存.该方法与 Graphicionado 相比,片上存储容量减少一半以上,且同时能够获得大致相同的性能.

5.3 片外存储优化

片外存储的优化目标是提升片外带宽的利用率和片外访存的效率.

片外带宽利用率的优化方法主要是根据图数据的访存依赖设计专用的数据预取器[13,48-49].

DROPLET 提出了感知图数据结构的预取器.该预取器被集成在 DRAM 控制器上,用于识别数据结构之间的访存依赖并直接在 DRAM 控制器上执行具有这类依赖的访存,减少依赖访存导致的处理核心与 DRAM 之间的回路延迟.该方法能够掩盖的延迟有限,一般仅占 20%[64] 左右.Tesseract 设计了 2 种预取器:列表预取器和消息触发预取器.列表预取器专用于访存具有空间局部性的边表数据,能够

比较好地挖掘边表数据的空间局部性.消息触发预取器根据远程的更新消息直接触发本地节点属性数据的预取,在 Tesseract 图计算加速系统中用于掩盖远程通信延迟和访存延迟.该方法主要面向 HMC 集群系统,并不具有非常强的通用性.GraphDynS 基于软硬件协同设计理念设计了图数据的精确预取器.GraphDynS 对 Vertex-centric 编程模型进行了优化,在执行过程中动态获取访存的起始地址和数据长度,并设计了对应的精确预取器用于边和节点数据的精确预取,掩盖了访存延迟并减少了冗余访问.该方法需要依赖 Vertex-centric 编程模型,因此并不适用于基于 Edge-centric 编程模型的加速架构.

片外访存效率优化的方法主要是设计窄通道(narrow channel)DRAM 或者根据访存模式定制 DRAM 减少访存开销.

根据文献[23]分析,传统图计算应用运行在传统架构上时,每一行 Cacheline(64 B)只有 4 B 左右的数据被使用,为此 Emu Chick[65] 和 Intel PUMA Graph Processor[66] 设计窄通道 DRAM 和对应的 DRAM 控制器.该设计通过修改 DRAM 内部 chip 的数据传输逻辑,提供了细粒度(4 B)的 DRAM 访存和 16 倍的访问速率,大大提高了 DRAM 访存效率.

HyVE 则利用 ReRAM,DRAM,SRAM 设计了一个混合的内存结构,目的在于减少访存开销.HyVE 根据图数据中不同数据结构访存的特性,将数据置于不同种类的存储中.ReRAM 用于存储只读且连续访问的边数据,片上 SRAM 用于缓存部分细粒度访问的节点数据,DRAM 用于存储所有可读可写的节点数据等.该方法不仅能利用 ReRAM 的非易失性和 power-gating 技术减少静态功耗,还能去除 ReRAM 在写操作方面的劣势,即写延迟长以及可写次数有限.

5.4 近内存优化

为了缩短数据与计算单元的距离以获得更高的带宽和更低的延迟,计算部件被集成到内存内部.主要研究现状包含:1)基于 HMC 集群的图计算加速系统[26-27,41,49];2)基于具有计算能力和存储特性的 ReRAM 的加速系统[53-56].

基于 HMC 集群的图计算加速系统的设计核心是在 HMC 的逻辑层为图计算应用添加特定的计算单元和计算调度硬件.根据 HMC 2.1 规格说明,单个 HMC 内部提供的带宽最高可达 320 GBps,外部可提供的带宽达到 480 GBps[67].HMC 逻辑层允许的最大功耗密度为 94 mW/mm^2[68].传统图计算应用是访存密集型应用,具有极低的计算访存比,对带宽需求大,计算部件需求少.因此,在功耗密度范围内,架构师可以在逻辑层添加逻辑电路,利用充足的带宽来满足图计算应用的带宽需求.这类图计算加速系统需要修改 HMC 的逻辑层,所需的工艺比较难,产品化周期长.

基于 ReRAM 的图计算加速系统的核心是将配置为计算的 ReRAM 与配置为存储的 ReRAM 集成一体,既缩短数据与执行单元的距离也获得充足的带宽和计算能力.如图 10 所示,ReRAM 不仅可以组织为存储图数据的 DRAM(ReRAM memory),同时也可以组织为 Crossbar 进行矩阵乘法计算(graph engine).同时使用这 2 种组织方式不仅能够获得充足的带宽,也能获得高并行的计算能力.由于图计算应用的输入图都极为稀疏,因此目前主要的研究问题是如何高效地执行稀疏矩阵乘(SPMV),即去除无效计算和访存.GraphR,RaGRa,GraphSAR 的核心设计都是对邻接矩阵进行分块,其他工作围绕 ReRAM 实现图计算加速系统的执行模型开展.GRAM 基于 ReRAM 实现了 Vertex-centric 的执行

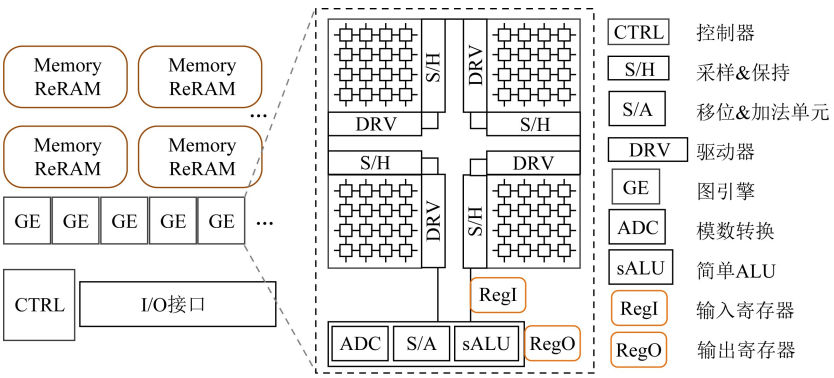


Fig. 10 The design of ReRAM-based graph processing accelerator (GraphR[52])

图 10 基于 ReRAM-based 图计算加速设计(GraphR[52])

模型从而减少数据的移动,RaGRa 为 3D ReRAM 实现了专用的行列混合执行模型,减少冗余的数据移动与计算.

然而 ReRAM 的可计算性存在局限,仅能完成乘加操作,因此能够支持的算法较少,仍然需要额外的传统电路完成比较等操作.

5.5 近存储优化

图数据的规模日益剧增,导致内存无法容纳如此多的数据.为了高效支持更大规模的图数据处理,在 Flash 存储端增加加速单元是一种可行的方案.Flash 存储提供的带宽非常小,一般在 3.6 GBps 左右[59],而访存粒度却非常粗,一般为页大小,8192 B 左右[59].由于传统图计算的访存粒度小(4 B 或者 8 B),主要的优化方向是提高 Flash 的带宽利用效率.

ExtraV 利用 Flash 端的加速装置降低主机访存的开销,但不具备进行图数据更新的能力.ExtraV 简化了主机 CPU 对存储的访问,减少了 I/O 访存延迟和开销,并提出了 expand-and-filter 的方法对图数据进行解压缩,提取有效数据送给主机 CPU,从而提高加速单元与主机 CPU 之间的带宽效率.GraFBoost 则将大部分的计算任务从 CPU 中卸载到 Flash 上的加速单元,利用排序、归一加速单元序列化和合并细粒度访存.加速单元首先记录随机的点数据更新请求,然后利用插值归一的方式进行排序,从而合并不规则的细粒度访存,减少 I/O 访存并提高 I/O 访存效率.GraphSSD 提出了感知图语义的 SSD,目的是从图数据的 I/O 访存出发,将 SSD 的访存方式更改为更适合图更新(点或边更改)的表达方式.

5.6 多节点优化

除了 5.1~5.5 节所述的单节点优化之外,还有面向多节点图计算加速系统的优化,主要包含横向扩展方案、降低通信开销和同步开销.

AsynACC 利用片上 Crossbar Switch 进行多节点扩展,但是多端口的片上 Switch 非常昂贵,限制了该设计的进一步拓展.在 40 nm 的工艺下,64 端口 32 b 传输宽度的 Switch 的功耗和面积分别达到了 2.75 W 和 4 mm^2[69].Emu Trick 系统则利用传统的片上网络(RapidIO[70])实现计算节点之间的互联,有效降低了节点拓展的开销.除此之外,Emu Trick 还利用线程迁移技术将负载移动到数据所在的计算节点中,从而减少节点数据广播和同步的开销.Fore-Graph 则利用总线(例如 PCI-e)将多个 FPGA 链接在一起,从而获得更多的片上存储 BRAM(block RAMs),减少不规则的片外数据访问.

如图 11 所示,文献[26-27,41,49]利用 HMC 提供的 SerDes Link 互联接口构建多节点系统.与 HBM(high bandwidth memory)相比较,HMC 标准提供 cube 之间高速互联的能力,带宽达到 480 GBps[67],所以大量工作利用 HMC 组建多节点图计算加速系统.Tesseract 提出的消息通信机制为基于 HMC 的多节点系统提供了图计算应用专属的多节点通信系统.GraphP 在 Tesseract 基础上提出了基于源节点的划分方法,并配合 2 阶段节点更新编程模型和层次通信方案,显著减少了节点间的通信和同步开销.GraphH 提出可重配置的双 Mesh 网络,动态为 cube 之间的通信分配带宽,从而最大化通信带宽的利用率.GraphQ 对节点数据的处理进行重排序,从而对 cube 之间的多个通信消息进行打包并进行批量传输,不仅提高了通信带宽利用效率,还有效地掩盖了 cube 之间的通信延迟.虽然基于 HMC 集群的方案能够获得非常好的性能和节点传输带宽,但由于封装技术和 HMC 存储容量(8 GB)的限制,无法实现更大规模的多节点系统,因此整个加速系统一般只有 16 个左右的 cube.

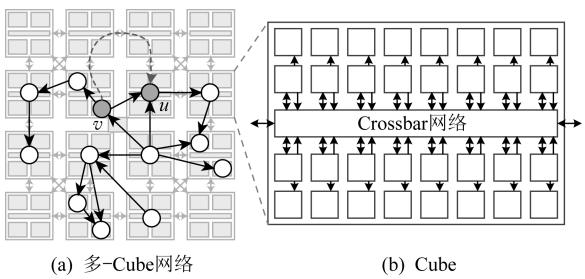


Fig. 11 The design of HMC-based multiple-node graph processing accelerator (Tesseract[49])

图 11 基于 HMC 多节点图计算加速系统设计(Tesseract[49])

5.7 不同层次优化技术的关系分析

图计算加速架构一般来说是一个比较完整的系统设计.在最初的工作中,通常在系统的各个层次都会有相应的优化技术.而后续的工作一般是在前人的基础上继续解决未解决的问题或者新问题,也就是说后续的工作都具有各自的优化核心或者重点优化的层次,不过层次之间也存在着协同优化的关系.因此,在本节我们从最初的设计出发,重点描述不同层次之间的协同优化.

在基于 ASIC 平台的第 1 款图计算加速架构 Graphicionado 的设计中,包含了片外存储、片上存储和计算流水线 3 个层次的优化.Graphicionado 利用

预取器提高片外带宽的利用率,供给充足的数据给片上存储.片上存储则提供细粒度的访存以消除不规则细粒度访存对性能和能效的影响.在片上存储快速为计算流水线提供数据的基础上,Graphicionado 设计了图计算专用的计算流水线以进行流水作业并保证计算的正确性.后续的 GraphDynS 设计也包含了片外存储、片上存储和计算流水线 3 个层次的优化.精确预取提高了片外带宽的利用效率且大容量片上存储解决了不规则细粒度片外访存,因此才能保证源源不断地给计算流水线供给数据.究其根本,吞吐量得到有效提升后,计算流水线层次的问题才会凸显出来.因此,GraphDynS 继而设计了无阻塞原子操作微架构流水线保证计算流水线的无阻塞执行,以及提出了感知节点度数的负载映射方式平衡流水线内或者流水线间的负载.在后续的其他工作中,HyVE 主要是在片上存储和片外存储层次上利用各种存储介质构建混合且层次化的存储子系统,以提高存储子系统的能效.AccuGraph 主要是在计算流水线层次对原子操作进行优化.文献[47]主要是在片上存储层次对片上存储的利用效率进行优化.

在最初的多节点 Tesseract 设计中,它的优化技术主要集中于多节点存储层次、近内存层次和片外存储层次.Tesseract 基于多节点 Memory-centric 的架构实现了节点间的通信机制,并在 HMC 的逻辑层添加计算逻辑和定制的数据预取部件以挖掘富裕的带宽以及掩盖访存和通信延迟.在多节点系统实现之后,不规则通信的问题被凸显出来,所以 GraphP 在 Tesseract 的多节点存储系统上提出了 source-cut 的数据分块方案、双阶段编程模型和层次通信机制用于优化多节点存储层次遇到的不规则通信.GraphQ 则在 GraphP 基础上对计算流水线内的节点数据处理进行重排序,从而对子节点之间的多个通信消息进行打包并进行批量传输,以进一步解决不规则通信.

综上所述,不同层次需要解决的问题不一样,各个层次的优化相辅相成.不同层次的优化手段通常都能够被有机地组合起来,成为完整图计算加速系统设计的一部分,以获得更高的整体性能和整体能效.

5.8 不同平台典型设计对比

本节从不同的平台出发,基于典型的设计对不同平台的传统图计算加速架构进行对比,分析不同平台的优缺点.传统图计算主流的加速平台包含 FPGA,ASIC,NDP(near DRAM processing),NSP(near storage processing),PIM(processing in memory).

基于 FPGA 平台的设计具有可配置性强、部署快的优点.FPGA 平台为硬件编程人员提供了直接为图计算应用定制加速架构的硬件编程接口,但对编程人员的素质要求也非常高.基于 FPGA 平台设计的图计算加速架构主要是为加速图计算应用提供具有一定通用性的框架,减少重复劳动和编程负担.例如,FPGP[42]工作基于 Interval-shard 的图数据分块方法在 FPGA 上实现了 streamlined Vertex-centric 编程模型,构建了大规模图计算硬件加速框架.FPGP 具有一定的灵活性,能支持主流的传统图计算算法,并大大减轻了用户的编程负担.但是文献[43]指出 FPGP 可能无法达到具有大容量片上存储的通用处理器执行传统图计算应用的性能.本质原因是片上存储资源少,需要精细设计片上数据替换策略以更高效地使用片上存储资源.

基于 ASIC 平台的设计能够获得比 FPGA 更高的性能,因为基于 ASIC 的图计算加速架构能够运行在更高的频率下,并为加速设计添加更多的计算资源和片上存储资源.ASIC 设计主要是利用大容量的片上储存缓存被不规则访问的数据,同时细化访存的数据通路.例如,Graphicionado[24]基于 Vertex-centric 的编程模型定制了图计算专用执行流水线以支持大部分的主流算法.除此之外,Graphicionado 用 64 MB 的 eDRAM 储存被不规则细粒度访问的节点属性数据.假如每个节点属性是 4 B,该片上存储一共可以存储 16×10^6 个节点属性,但是片上存储是非常耗费面积和功耗的.同时随着图规模的增大,分块数目不断增多,图数据分布更加稀疏,导致性能下降会非常严重.例如,在 Graphicionado 的工作中,当分块数目从 1 变成 8 时,性能下降 50%.根据统计报告[71]所述,Facebook 的用户(社交网络的节点)高达 20 亿,并且每年以 17%的速度增长,那么利用大容量片上存储和数据划分的 ASIC 解决方案会面临急剧的性能下降.

基于 NDP 的平台设计能取得更高的带宽和更小的延迟,因为计算单元和储存单元的距离更近.例如,Tesseract[49]基于 HMC 集群构建了图计算加速架构.Tesseract 在 HMC 的逻辑层添加了 ARM core 用于处理传统图计算中简单的计算,以拉近与 DRAM 的距离.并通过 HMC 默认提供的 SerDes 通信接口链接多个 cube,以实现多节点的图计算加速系统.虽然 Tesseract 图计算加速架构能够获得很高的带宽,但是由于传统图计算的访问粒度比较细,导致了

带宽的利用效率非常低.由于 cube 之间的负载不均衡导致了整个系统的同步开销非常大.除此之外,3D-stack 的工艺仍然不成熟,而且由于 HBM 的广泛应用,美光公司基本已经放弃 HMC[72],无法继续提供持久的技术或者工艺支持.

基于 NSP 平台的设计主要是为了解决规模日益增长的图数据对存储的需求以及降低对数据通路的压力.巨大规模的图数据导致了数据无法完全被保存到 DRAM 上,需要频繁地访问 storage.由于 storage 的访存带宽非常小且访存延迟非常大,频繁的数据替换成为了大规模图计算的瓶颈.为了应对这个瓶颈,GraFBoost[59] 为 SSD(solid state drive)增加额外的访存处理单元用于合并细粒度的访存,提高了带宽的利用率,并减少在 storage 和片上计算资源之间的数据传输回路.

基于 PIM 平台的设计主要是利用 ReRAM 既可以作为存储又可以作为计算单元的特性,进一步缩短了数据与存储介质(device)之间的距离.现有的 ReRAM 只支持加法和乘法操作,无法支持 SSSP 这类基于比较操作的算法,能够高效支持的应用有限.除此之外,现有的 ReRAM 工艺最多只能保障 16 b 的整形乘法计算,无法为高精度计算的图计算应用提供保障.例如,GraphR[52] 利用 ReRAM Crossbar 执行矩阵乘法,工作核心实际上是在 ReRAM Crossbar 上高效执行稀疏矩阵乘.为了利用 ReRAM 的计算特性,GraphR 牺牲了部分通用性,能被 GraphR 高效支持的图计算种类较少.

不同平台间的比较如表 5 所示,基于 FPGA 平台的设计具有配置灵活和部署快的优点,但是受限于有限的硬件资源,无法提供高性能的解决方案.基于 ASIC 平台的设计性能高,但因为依赖大容量片上存储和分块方法,伸缩性差.基于 NDP 平台的设计具有一定的伸缩性,性能能够随着 DRAM 存储容量的增大而提升,然而,NDP 平台的设计所需要的工艺仍然不成熟.基于 NSP 平台的设计能够处理规模更大的图数据,但带宽低且带宽利用效率也低.基于 PIM 平台的设计能取得非常好的性能和能效,但局限于 ReRAM 的特性,通用性较差.

Table 5 Comparison of Different Platforms

表 5 不同平台优缺点对比

平台	设计原则	优点	缺点
FPGA 平台	通用性图计算框架	可配置性强、部署快	片上硬件资源少、频率低
ASIC 平台	为不规则的细粒度访存定制存储子系统	性能高	可扩展性低
NDP 平台	使计算单元靠近 DRAM	带宽高、DRAM 访存延迟低	工艺不成熟
NSP 平台	合并访存,减少数据传输回路	可扩展性高	带宽利用效率低
PIM 平台	存内计算	性能高、能效高	通用性低

6 图神经网络加速架构案例与讨论

图神经网络是新兴的图计算应用,具有与传统图计算应用不同的执行特征,解决方案也有所不同,且现有研究工作较少,因此进行单独介绍.本节将以现有的第 1 款图神经网络加速器 HyGCN[19] 作为案例,从计算、片上存储、片外访存这 3 个方面讨论图神经网络加速架构.

6.1 计算流水线优化

每个节点的 *Aggregate* 函数的计算图是不规则的[73],而 *Combine* 函数是规则的.由于每个节点的邻居节点数目不一样,所以执行不同节点的 *Aggregate* 函数会有不同的计算图.而由于每个节点在执行 *Combine* 函数时共享同一个神经网络并且每层的每一个神经元具有相同的连接,因此所有节点的计算图都是相同的.

为了高效支持不规则计算图和利用规则计算图提高执行效率,如图 12 所示,HyGCN 分别设计了 2 个引擎:Aggregation 引擎和 Combination 引擎.

Aggregation 引擎由若干个 SIMD(single instruction multiple data)core 和边调度器组成.边调度器根据边表中的邻居节点将每一个邻居节点向量的 element-wise 归约操作分配到不同 SIMD core 的不同 lane 上,以挖掘 edge-level parallelism 和 intra-vertex parallelism,并将不规则的计算平均分配到不同 lane 上,实现负载均衡.

Combination 引擎主要由多个小的脉动阵列和点调度器组成.该设计用于挖掘 inter-vertex parallelism 和 matrix-vector multiplication parallelism,

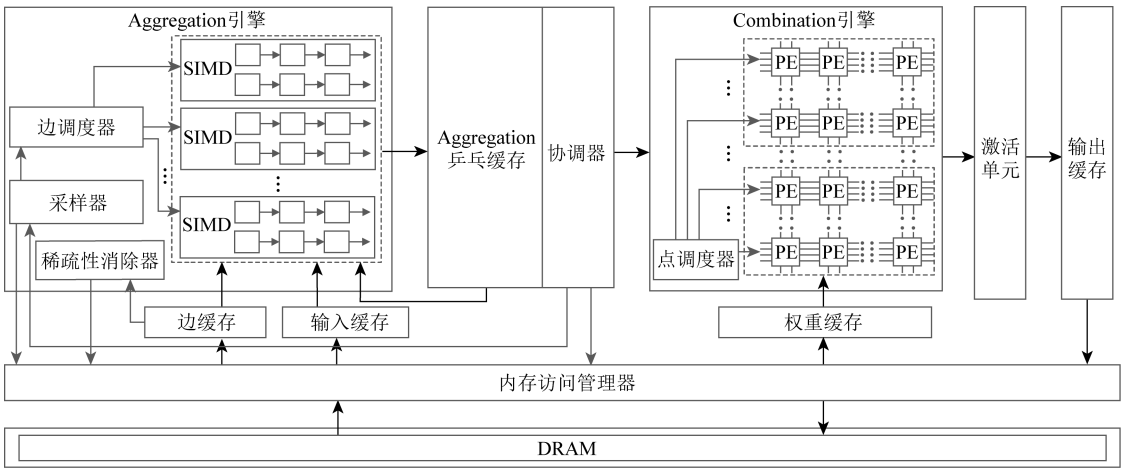


Fig. 12 A graph convolutional neural network accelerator with hybrid architecture (HyGCN[19])

图 12 基于混合架构的图神经网络加速架构(HyGCN[19])

以及复用权重数据.点调度器将若干个节点组合成一个小节点集合,然后将该集合送入其中一个小脉动阵列中.小脉动阵列的每一行负责集合中一个节点的向量变换,小脉动阵列的所有节点共享权重.该执行方式被称为独立执行模式,既能利用规则计算图提高数据复用率和并行性,同时也能减少每个节点的平均执行延迟.点调度器还能将更多的节点组合成一个大节点集合,一起送入所有的小脉动阵列中,实现所有小脉动阵列中的所有节点共享权重.该方式被称为联合执行模式,能够利用规则的计算图挖掘更多的数据复用,从而降低功耗.但相对于独立执行模式,联合执行模式需要集合更多的节点一起执行,所以节点的平均执行延迟有所增加.

6.2 片上访存优化

为了挖掘各种数据的局部性,HyGCN 为具有不同访存模式的数据设计了不同的缓存.每个节点的边表大小不固定,所以对于边数据的访存,HyGCN 基于 eDRAM 提供了一种支持细粒度访存(32 B)的边缓存.每个节点的输入特征向量一般都在 128 个元素以上,HyGCN 基于 eDRAM 和多个 bank 进行设计,提供了一种支持粗粒度访存(512 B)和高吞吐量的输入缓存.权重矩阵的数据是按时钟顺序被访问的,且每列的权重元素一般多于 128 个.但是脉动阵列无法在 1 个 cycle 消耗如此多的数据,所以将权重缓存的访存粒度设置为与脉动阵列的 FIFO 相同的大小.由于每个节点的输出向量由脉动阵列产生,访存粒度细,因此 HyGCN 基于 eDRAM 和多个 bank 进行设计,提供了一种支持细粒度访存(4 B)和高吞吐量的输出缓存.在此基础上,HyGCN 为上述每个缓存增加 1 倍的存储空间,实现双缓冲机制,以掩盖访存延迟.除此之外,HyGCN 还在 Aggregation 引擎和 Combination 引擎之间增加了 ping-pong buffer,一方面用于缓存 Aggregation 引擎的中间结果,另一方面用于流水 Aggregation 引擎和 Combination 引擎的执行.

6.3 片外访存优化

片外访存优化主要包含了节点向量数据分块机制、基于窗口滑动收缩动态消除稀疏优化和基于优先级的动态访存调度.

为了提高片外带宽的利用率,HyGCN 将节点平均分成若干个区间,然后根据区间将邻接矩阵分成若干块.在进行计算时,每个块内所有源节点的特征向量被读入到输入缓存中.由于访存非常连续且规则,所以带宽的利用率非常高.然而稀疏特性导致大量的读入特征向量数据均非所需,所以 HyGCN 提出了基于窗口滑动收缩的方法来消除稀疏.该方法根据 ping-pong buffer 的大小和节点区间大小分别设定窗口的高和宽.窗口从邻接矩阵的上方出发滑落,直到窗口的第 1 行遇到不为 0(源节点和目的节点之间存在连接)的值.接着窗口从下往上进行收缩,直到窗口的最后 1 行遇到不为 0 的值.这样就能够消除大量由稀疏导致的无效访存.该方法对图神经网络有效的原因在于,图神经网络中的特征向量的元素数目是传统图计算节点属性个数的上百倍以上.因此,即使窗口滑动收缩方法在传统图计算上开销大,但是在图神经网络中的收益是十分可观的.

为了协调 2 个引擎之间的片外访存,HyGCN 设计基于优先级的动态内存调度.Aggregation 引擎

从片外主要读取的数据为边数据和输入特征向量;Combination 引擎从片外主要读取的数据为权重,写入片外的数据为输出向量.这些数据的片外访存请求会同时进行,访存地址的不连续,导致 DRAM 的 row buffer 命中率低,访存延迟增加 50%以上[74].为了提高 row buffer 的命中率,HyGCN 首先对这些访存请求进行处理,合并具有连续地址的访存请求.然后按照处理节点计算的数据请求顺序,执行片外的数据请求.最后,HyGCN 利用低地址段寻址 bank 和 channel,从而挖掘 DRAM 的 bank 和 channel 级别访存并行性.

除了 HyGCN 外,GraphACT[75]也是该领域已发表的工作.该设计基于 CPU-FPGA 的异构实现了图神经网络的加速.GraphACT[75]在 CPU 上基于轻量的预处理优化 Aggregation 函数的执行.然后利用由 CPU 和 FPGA 组成的异构系统加速和流水 *Aggregate* 函数和 *Combine* 函数的执行.其中在 CPU 上的优化主要是识别节点间的共享邻居节点集合,辅助 FPGA 去除共享邻居节点的冗余归约操作,复用中间计算结果和减少 BRAM 的访存.

7 不同架构设计的应用场景比较

不同的图计算加速架构的应用场景因优化技术的特点而有所不同,因此我们以具体的架构设计从支持的算法、支持的图规模和需要的图特性 3 个方面比较它们的应用场景.

基于 FPGA 平台的传统图计算加速架构通常会提供模板化的 FPGA 硬件模块,具有非常强的灵活性,适合需要快速部署的应用场景.例如 FPGA 开发者可以基于 ForeGraph 提供的模板库进行模块组装和添加新的硬件逻辑完成新算法在 FPGA 上的重配置,以实现新算法硬件加速架构的快速部署.然而,也由于有限的片上存储资源和片外带宽,ForeGraph 支持的图规模无法与基于 ASIC 平台设计的加速架构相比.ForeGraph 架构中的所有优化技术不需要图数据具有特定的特点,例如图数据的强集合结构特性.

基于 ASIC 平台的传统图计算加速架构通常利用大容量的片上存储系统定制内存子系统,从而优化不规则的细粒度片外访存.例如 Graphicionado 的核心设计就是利用大容量片上存储被不规则访问的数据,能够高效加速 BFS,SSSP,PR 和协同滤波等算法,因此该架构可以高效支持导航、网页搜索和推荐系统等应用.限制于片上存储的容量和单节点系统的设计,Graphicionado 支持的图规模无法与基于 NDP 平台设计的加速架构相比.Graphicionado 的所有优化技术并不需要图数据具有特定的属性.为传统 CPU 设计的图计算加速单元 HATS 则基于传统 Cache 结构对访存进行调度,挖掘图数据局部性.由于被集成到传统通用架构上,HATS 能够支持非常多的算法,但图的规模仍然受限于内存容量,并且需要图数据具有强集合结构特性.

基于 NDP 平台的传统图计算框架能够实现与内存存储容量成正比的性能提升.例如,支持 PR 和 SSSP 等算法的 Tesseract.然而,受限于每平方毫米散热的限制,Tesseract 难以支持计算密集型的算法.Tesseract 支持的图规模受限于多节点系统内的 cube 数目和每个 cube 的容量.由于封装在同一 Package 上的 HMC cube 的数目是有限的,并且每个 cube 的存储容量也无法达到传统内存 DDR4 的规模,因此支持的图规模无法与传统分布式系统相提并论.Tesseract,GraphP,GraphR,GraphH 等工作都不需要图数据具有特定的属性.

基于 NSP 平台的传统图计算框架则非常适合用于处理极大规模的图数据.因为图数据规模过大,无法完全都存储到内存上,这时候瓶颈通常出现在 storage 的数据访问上,所以对于 storage 的优化尤为重要.GraFBoost 通过合并细粒度访问和减少 CPU 与 storage 之间的数据传输,有效地利用了 storage 的有限带宽,实现了高效的极大规模图计算加速架构.GraFBoost 支持的算法有 PR 和 BFS 等归约形式的算法,不需要图数据具有特定的属性.

基于 PIM 平台的传统图计算框架计算效率非常高,比较适合能用矩阵乘法表示的图计算算法.基于 ReRAM 设计的 DRAM 容量能够达到 TB 级以上,非常适合存储超大规模的图数据.例如 GraphR 能利用 ReRAM 这个大存储特性处理大规模图数据,利用 ReRAM 的计算特性高效执行 PR 等算法.GraphR 以及后续的 GraphSAR 和 GRAM 都不需要图数据具有特定的属性.

面向传统图计算应用的加速架构一般不适合用于图神经网络的加速,而面向图神经网络的加速架构一般也不适合用于传统图计算应用的加速.本质的原因是两者具有不一样的计算和访存行为,甚至是不一样的通信行为.具体的原因有 3 个:1)传统图计算应用的计算较为简单,而图神经网络的计算是密集的矩阵向量乘法或者是矩阵乘法;2)虽然传统

图计算应用和图神经网络的访存都是不规则,但前者是不规则的细粒度访存而后者是不规则的粗粒度访存;3)虽然传统图计算应用和图神经网络的通信都是不规则,但前者是不规则的细粒度通信而后者是不规则的粗粒度通信.因此,HyGCN 以及 GraphACT 等面向图神经网络的加速架构无法高效加速传统图计算应用,而传统图计算加速架构 Graphicionado 等也无法高效加速图神经网络.

8 展　　望

本节将以图计算加速架构系统设计为主线,从测试评估与全栈系统设计这 2 个方面对图计算加速架构研究方向进行展望.

8.1 测试评估

测试评估的未来主要研究方向是准备具有代表性的图数据集合和算法模型测试集合,作为评估图计算加速架构性能和执行效率的标准.由于图计算应用解决问题的多样性,被用来测试的图数据应能表征图计算应用的多样性.同时,算法模型测试集也应能表征市场和应用场景的需求.基于上述标准设计的图计算加速架构,才能够面向市场和面向需求.

1) 构建具有代表性的图数据测试集合.为了表征问题的多样性,图数据集合应该具有 3 种特性:多样性、多维度和多模.多样性指的是图数据集合应具备表征主流应用领域数据的能力,例如人际关系和大脑神经元链接等.多维度指的是每个节点的属性应包含多个维度的信息.例如对于单个药物而言,它具有价格、保质期和使用方式等独立信息.多模指的是图数据中的节点和链接的类型是可以不统一的.例如图 13 所示,在包含人际关系和药物联系的混合图数据中,人与人之间的联系可以是患者与医生的治疗关系,而人与药物之间的联系可以是腰椎病患者与腰椎病药品的治疗方式.

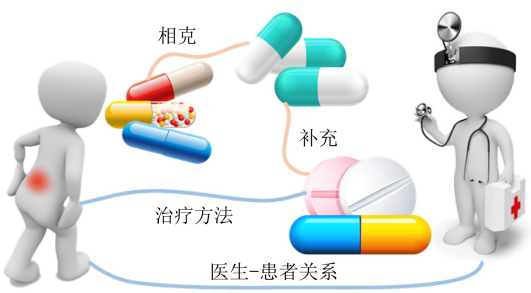


Fig. 13　The example of multimode graph data

图 13　多模图数据例子

目前已有众多大规模开源图数据集,可应用于传统的图计算应用中,例如 SNAP 实验室[76]、TAMU 大学[77]和 GraphVis[78]发布的数据集合.然而,由于图神经网络的图数据需要专家标注,因此应用于该新兴图计算应用的现有开源图数据集合类型少、规模小,且一般只涉及单模甚至单个维度.其中开源且被标注的图数据集合有德国多特蒙德大学[79]提供的数据集库和 GraphVis[78]数据库,但规模仍较小.图神经网络架构的蓬勃发展亟需大型开源图数据集合.

2) 统一图数据存储方式.作为图计算应用的输入及核心处理数据,若开发者能够方便地获取具有统一存储形式的开源图数据集或者企业用户能够按照统一标准提供待处理的图数据,则可避免昂贵的数据预处理过程,并削弱图计算应用对通用处理器的依赖.因此,建立统一且高效的图数据存储标准,并受后续研发的图计算加速架构支持,能够有效降低图数据存储差异性导致的开发成本与数据中心的配置成本,是促进图计算加速架构产业化发展的有力途径之一,也是一个非常值得深入研究的方向.

3) 构建代表性算法测试集.图计算适用于众多现实应用场景,为不同领域完成图数据处理的工作,因此算法测试集合应具备既能全面表征其应用场景,又能体现发展方向的能力.虽然传统图计算应用的基准测试集非常多,但是目前应用于图神经网络的基准测试集仍非常少.因此为了全面评估图神经网络加速架构,准备具有代表性且能体现一定周期内图神经网络发展趋势的算法测试集合是一项极具应用价值的工作.

4) 分析算法的执行特征.专用加速器的设计核心在于针对应用的特定行为进行优化,因此在架构设计过程中,对应用进行深入分析,找到瓶颈所在是极为重要的环节.目前分析图神经网络的执行特征的工作仅有文献[19,25].在计算机架构研究领域,产业界和学术界鲜有对不同图神经网络模型的系统性分析,无法形成对新架构设计的指导.因此,对图神经网络算法测试集执行特征的分析必将成为辅助图计算加速器架构设计的研究方向之一.

8.2 系统设计

要完成图计算加速架构的产业化,必然需要一套完整的全栈系统设计方案,既能向上提供服务,又能向下挖掘极限性能和效率.具体而言,该方案须从软件接口和硬件设计这 2 方面应对如下需求和开展研究工作.

1) 友好的接口和高效的功能转换(易用性).为提高上层用户(算法开发和编程人员)的体验,面向用户的编程接口应该易用且友好.因此,开发易入门的编程框架是非常重要的.研究方向可以是基于或者借鉴 PyTorch[80] 或 TensorFlow[81] 等易入门且被广泛使用的编程框架设计图计算加速架构专用的编程框架.该编程框架应该能抽象各平台或者各类图计算加速架构的通信和控制接口,实现硬件透明和统一加速.因此,统一高效的编程接口也是图计算加速架构未来的研究方向之一.

研发高效的功能转换机制,根据算法设计人员的初始功能代码进行面向图计算加速硬件的优化,也能够保证软件的易用性.编译中间件是构建算法和架构间桥梁的关键.构建类似 TVM 的编译中间件,为已有图计算应用加速架构提取所支持加速的操作或者计算图,能够延长现有加速架构的使用寿命.因此构建编译中间件也是图计算加速架构未来的研究方向之一.

2) 通用图计算加速架构(通用性).为了解决更多图相关的问题和支持新种类的图数据,图计算加速架构应具有一定的通用性.图计算应用在日益更新,比如在新兴的图神经网络中,算法模型就在不断变化.除此之外,商品推荐图、社交网络等图都在不停地增加或者减少节点和链接.然而现有的工作普遍不支持动态图计算.传统的图计算应用、动态图计算和图神经网络应用的执行行为既有相同之处也有不同之处,设计能够同时高效支持这些应用的通用图计算加速架构,是非常具有吸引力和应用前景的研究方向.

3) 主客机或多节点互联高速通信接口(连通性).由于图数据的数据量极大且增速快,因此亟需一种高效的数据传输方案或者高速的通信接口,用于连接主机和加速架构,以及连接多节点加速架构的各个子系统.图计算应用加速架构无法成为一个完全独立的系统,并且为了兼容现有数据中心的需求,加速架构通常作为客机存在,也即作为协处理器,与主机相联.现有的通信接口,例如 PCIE 和 NVLINK 等目前无法满足大规模图计算对传输带宽的需求,导致传输成为了瓶颈.由于单节点系统内存在 DRAM 存储资源有限、片上储存资源有限和计算资源有限的问题,实现多个乃至上百个子系统互联,以建立更大型的图计算加速系统,提供更多的硬件资源是非常有必要的.因此,为大规模图计算加速架构设计主客机或多节点系统的高速通信接口也是未来的热门研究方向之一.

4) 多节点加速系统的拓展机制(可拓展性).多节点加速架构应该具有可拓展性,既能集合少量的子系统处理轻量的图计算任务,也能集合大量的子系统处理大型任务,以满足所提供的不同服务的需求.在节点的拓展过程中,如何实现线性的性能增长也将是未来的热门研究方向之一.

5) 大规模图计算加速系统的容错机制(容错性).容错能力对于处理大型图计算任务的系统来说,是非常重要的.图规模不断扩大,导致程序的执行时间也将不断增长.为了应对处理过程中出现的机器故障或者其他问题,整个系统必须具有容错的能力,从而能在故障发生之后快速恢复图计算应用的执行.因此,面向大规模图计算加速系统的容错机制也将是图计算加速架构未来的研究方向.

6) 大规模图计算加速系统的安全机制(安全性).性能和能效得到提升后,系统的安全性成为了新的需求.在图数据爆发和万物互联的时代,随着图计算应用领域不断扩张,越来越多的隐私数据被加速处理,因此对数据的安全性保护也变得愈加重要.没有用户希望自己的隐私被暴露在公共场合中,所以加速系统应该具有足够的安全保障,确保用户的数据不被泄露.因此,如何保障图计算加速系统中的数据安全也将是重要的研究方向.

7) 图计算加速架构专用模拟器(探索设计).为了对图计算加速架构的设计空间进行探索以及进行前期架构的验证,实现架构的模拟是行之有效且尤为重要的途径.图计算应用中的不规则执行行为会导致模拟器的执行非常慢,其次图数据规模的不断增大和加速器架构模拟细节的增加,对于加速架构的模拟更是雪上加霜.因此,为图计算加速架构定制模拟器的同时,加速模拟也尤为重要.除此之外,目前不存在能够完美支持所有图计算应用的加速架构.所以图计算加速架构的模拟器还应该具有一定的可配性,能够基于需求被配置用于探索各种可能性的图计算加速架构.具有一定可配性的图计算加速架构用模拟器必将大大促进整个图计算加速架构研究社区的发展.因此,结合图计算应用的执行行为和图数据的特征提升图计算加速架构模拟器的模拟速度,以及设计可配置的图计算加速架构模拟器也将是未来热门的研究方向.

8) 基于 RISC-V 的图计算加速系统研究(加速定制).图计算加速系统的设计可以基于现有 RISC-V 联盟的开源资源,减少开发周期和开发成本.RISC-V

联盟可以为定制图计算加速系统提供生态系统和各种开源工具,其具体优势有:①RISC-V 指令集的模块性强,并提供可修改性.其中 P 指令和 V 指令可被定制为特定应用需要的功能.例如,可用于定制感知图数据结构的访存指令,从而优化预取效率;或者用于定制粗粒度原子操作指令,提高并行度和减低原子操作开销.②RISC-V 联盟提供了编译工具到模拟器、到 IC 综合工具的全套工具链.基于开源的工具链,开发者能够在其上为图计算加速系统的定制优化做修改,从而大大提高产业化速度,降低研发成本,并为可持续开发和优化提供可能性.除此之外,各种开源已验证的 IP 均可用于芯片的设计,例如利用 OmniXtend 互联接口接连多个子系统或内存子系统.③由于主机和加速器之间的指令集统一且 RISC-V 指令集无需授权即可修改,所以可以完全消除主机与图计算加速器之间的数据传输,控制直接相连并共享存储,同时还能实现主机与加速器之间的无缝任务迁移.

综上所述,RISC-V 构建的生态系统为构建图计算加速全栈系统提供了非常多有利的工具和资源,为系统的实现和优化增加了更多的可能性.因此,基于 RISC-V 的图计算加速系统也将成为未来的研究方向之一.

9 总　　结

本文以图计算应用加速遇到的关键挑战为导向,以解决方案为核心,基于计算机金字塔组织结构,从上到下,逐层对图计算加速架构的研究现状进行了系统的归纳和总结.除此之外,本文以第 1 款图神经网络加速器 HyGCN 作为案例,对新兴图计算应用加速架构进行着重介绍.最后,本文从未来图计算加速架构的测试评估与全栈系统设计的角度,对图计算加速架构未来研究方向进行了展望.我们相信本文对传统图计算加速架构和图神经网络加速架构的系统性总结和前沿研究方向的展望将有助于读者了解图计算加速架构的研究现状和图计算加速架构的前沿研究方向.

参　考　文　献

[1] Ye Nan, Hao Ziyu, Zheng Fang, et al. Adaptability of BFS algorithm and many-core processor [J]. Journal of Computer Research and Development, 2015, 52(5): 1187-1197 (in Chinese)
(叶楠, 郝子宇, 郑方, 等. BFS 算法与众核处理器的适应性研究[J]. 计算机研究与发展, 2015, 52(5): 1187-1197)

[2] Horawalavithana Y S. On the design of an efficient hardware accelerator for large scale graph analytics [R]. Tempa, FL: University of South Florida

[3] Gui Chuangyi, Zheng Long, He Bingsheng, et al. A survey on graph processing accelerators: Challenges and opportunities [J]. Journal of Computer Science and Technology, 2019, 34(2): 339-371

[4] Toy W N, Zee B. Computer Hardware-Software Architecture [M]. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference, 1986

[5] Leskovec J, Lang K J, Dasgupta A, et al. Statistical properties of community structure in large social and information networks [C] //Proc of the 17th Int Conf on World Wide Web. New York: ACM, 2008: 695-704

[6] Nodehi S A H, Qiu Junqiao, Zhao Zhijia. Tigr: Transforming irregular graphs for GPU-friendly graph processing [J]. ACM SIGPLAN Notices, 2018, 53(2): 622-636

[7] Gonzalez J E, Low Y, Gu Haijie, et al. PowerGraph: Distributed graph-parallel computation on natural graphs [C] // Proc of the 10th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2012: 17-30

[8] Malewicz G, Austern M H, Bik A J C, et al. Pregel: A system for large-scale graph processing [C] //Proc of the ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2010: 135-146

[9] Roy A, Mihailovic I, Zwaenepoel W. X-stream: Edge-centric graph processing using streaming partitions [C] //Proc of the 24th ACM Symp on Operating Systems Principles. New York: ACM, 2013: 472-488

[10] Zhou Shijie, Prasanna V K. Accelerating graph analytics on CPU-FPGA heterogeneous platform [C] //Proc of the 29th Int Symp on Computer Architecture and High Performance Computing (SBAC-PAD). Piscataway, NJ: IEEE, 2017: 137-144

[11] Wang Yangzihao, Davidson A, Pan Yuechao, et al. Gunrock: A high-performance graph processing library on the GPU [C] //Proc of the 21st ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2016: 11:1-11:12

[12] Khayyat Z, Awara K, Alonazi A, et al. Mizan: A system for dynamic load balancing in large-scale graph processing [C] // Proc of the 8th ACM European Conf on Computer Systems. New York: ACM, 2013: 169-182

[13] Yan Mingyu, Hu Xing, Li Shuangchen, et al. Alleviating irregularity in graph analytics acceleration: A hardware/software co-design approach [C] //Proc of the 52nd Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2019: 615-628

[14] Kipf T N, Welling M. Semi-supervised classification with graph convolutional networks [J]. arXiv preprint, arXiv: 1609.02907, 2016

[15] Hamilton W, Ying Zhitao, Leskovec J. Inductive representation learning on large graphs [C] //Proc of the Advances in Neural Information Processing Systems. Cambridge, MA: MIT Press, 2017: 1024-1034

[16] Xu Keyulu, Hu Weihua, Leskovec J, et al. How powerful are graph neural networks? [J]. arXiv preprint, arXiv:1810.00826, 2018

[17] Diez I, Sepulcre J. Neurogenetic profiles delineate large-scale connectivity dynamics of the human brain [J]. Nature Communications, 2018, 9(1): 1-10

[18] Xie Tian, France-Lanord A, Wang Yanming, et al. Graph dynamical networks for unsupervised learning of atomic scale dynamics in materials [J]. Nature Communications, 2019, 10(1): 1-9

[19] Yan Mingyu, Deng Lei, Hu Xing, et al. HyGCN: A GCN accelerator with hybrid architecture [C] //Proc of the 26th IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2020: 15-29

[20] Ying Zhitao, You Jiaxuan, Morris C, et al. Hierarchical graph representation learning with differentiable pooling [C] //Proc of the Advances in Neural Information Processing Systems. Cambridge, MA: MIT Press, 2018: 4800-4810

[21] Ma Lingxiao, Yang Zhi, Chen Han, et al. Garaph: Efficient GPU-accelerated graph processing on a single machine with balanced replication [C] //Proc of the USENIX Annual Technical Conf (USENIX ATC 17). Berkeley, CA: USENIX Association, 2017: 195-207

[22] Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit [C] //Proc of the 44th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2017: 1-12

[23] Eyerman S, Heirman W, Du Bois K, et al. Many-core graph workload analysis [C] //Proc of the Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2018: 282-292

[24] Ham T J, Wu L, Sundaram N, et al. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics [C] //Proc of the 49th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2016: 56:1-56:13

[25] Yan Mingyu, Chen Zhaodong, Deng Lei, et al. Characterizing and understanding GCNs on GPU [J]. IEEE Computer Architecture Letters, 2020, 19(1): 22-25

[26] Zhang Mingxing, Zhuo Youwei, Wang Chao, et al. GraphP: Reducing communication for PIM-based graph processing with efficient data partition [C] //Proc of the 24th IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2018: 544-557

[27] Zhuo Youwei, Wang Chao, Zhang Mingxing, et al. GraphQ: Scalable PIM-based graph processing [C] //Proc of the 52nd Annual IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2019: 712-725

[28] Tumeo A, Feo J. Irregular applications: From architectures to algorithms [J]. Computer, 2015, 48(8): 14-16

[29] Sundaram N, Satish N, Patwary M M A, et al. GraphMat: High performance graph analytics made productive [J]. arXiv preprint, arXiv:1503.07241, 2015

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

[31] Khorasani F, Vora K, Gupta R, et al. CuSha: Vertex-centric graph processing on GPUs [C] //Proc of the 23rd Int Symp on High-performance Parallel and Distributed Computing. New York: ACM, 2014: 239-252

[32] Ma Lingxiao, Yang Zhi, Miao Youshan, et al. NeuGraph: Parallel deep neural network computation on large graphs [C] //Proc of the USENIX Annual Technical Conf (USENIX ATC 2019). Berkeley, CA: USENIX Association, 2019: 443-458

[33] Fey M, Lenssen J E. Fast graph representation learning with PyTorch Geometric [J]. arXiv preprint, arXiv:1903.02428, 2019

[34] Wang Minjie, Yu Lingfan, Zheng Da, et al. Deep graph library: Towards efficient and scalable deep learning on graphs [J]. arXiv preprint, arXiv:1909.01315, 2019

[35] Page L, Brin S, Motwani R, et al. The PageRank citation ranking: Bringing order to the Web, 1999-66 [R]. Stanford, CA: Stanford InfoLab, 1999

[36] Lerer A, Wu L, Shen Jiajun, et al. PyTorch-BigGraph: A large-scale graph embedding system [J]. arXiv preprint, arXiv:1903.12287, 2019

[37] Alibaba. Alibaba/Euler: A distributed graph deep learning framework [DB/OL]. [2020-02-25]. https://github.com/alibaba/euler

[38] Ozdal M M, Yesil S, Kim T, et al. Energy efficient architecture for graph analytics accelerators [J]. ACM SIGARCH Computer Architecture News, 2016, 44(3): 166-177

[39] Dai Guohao, Huang Tianhao, Chi Yuze, et al. GraphH: A processing-in-memory architecture for large-scale graph processing [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018, 38(4): 640-653

[40] Yao Pengcheng, Zheng Long, Liao Xiaofei, et al. An efficient graph accelerator with parallel data conflict management [C] //Proc of the 27th Int Conf on Parallel Architectures and Compilation Techniques. Piscataway, NJ: IEEE, 2018: 8:1-8:12

[41] Nai Lifeng, Hadidi R, Sim J, et al. GraphPIM: Enabling instruction-level PIM offloading in graph computing frameworks [C] //Proc of the 23rd IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2017: 457-468

[42] Dai Guohao, Chi Yuze, Wang Yu, et al. FPGP: Graph processing framework on FPGA a case study of breadth-first search [C] //Proc of the ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2016: 105-110

[43] Dai Guohao, Huang Tianhao, Chi Yuze, et al. ForeGraph: Exploring large-scale graph processing on multi-FPGA architecture [C] //Proc of the ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2017: 217-226

[44] Mukkara A, Beckmann N, Abeydeera M, et al. Exploiting locality in graph analytics through hardware-accelerated traversal scheduling [C] //Proc of the 51st Annual IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2018: 1-14

[45] Mukkara A, Beckmann N, Sanchez D. PHI: Architectural support for synchronization-and bandwidth-efficient commutative scatter updates [C] //Proc of the 52nd Annual IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2019: 1009-1022

[46] Faldu P, Diamond J, Grot B. Domain-specialized cache management for graph analytics [C] //Proc of the 26th IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2020: 234-248

[47] Yan Mingyu, Hu Xing, Li Shuangchen, et al. Balancing memory accesses for energy-efficient graph analytics accelerators [C] //Proc of the IEEE/ACM Int Symp on Low Power Electronics and Design (ISLPED). Piscataway, NJ: IEEE, 2019: 1-6

[48] Basak A, Li Shuangchen, Hu Xing, et al. Analysis and optimization of the memory hierarchy for graph processing workloads [C] //Proc of the 25th IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2019: 373-386

[49] Ahn J, Hong S, Yoo S, et al. A scalable processing-in-memory accelerator for parallel graph processing [C] //Proc of the 42nd Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2015: 105-117

[50] Angizi S, Fan Deliang. GraphiDe: A graph processing accelerator leveraging In-DRAM-Computing [C] //Proc of the Great Lakes Symp on VLSI. New York: ACM, 2019: 45-50

[51] Huang Tianhao, Dai Guohao, Wang Yu, et al. HyVE: Hybrid vertex-edge memory hierarchy for energy-efficient graph processing [C] //Proc of the Design, Automation & Test in Europe Conf & Exhibition (DATE). Piscataway, NJ: IEEE, 2018: 973-978

[52] Song Linghao, Zhuo Youwei, Qian Xuehai, et al. GraphR: Accelerating graph processing using ReRAM [C] //Proc of the 24th IEEE Int Symp on High Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2018: 531-543

[53] Dai Guohao, Huang Tianhao, Wang Yu, et al. GraphSAR: A sparsity-aware processing-in-memory architecture for large-scale graph processing on ReRAMs [C] //Proc of the 24th Asia and South Pacific Design Automation Conf. New York: ACM, 2019: 120-126

[54] Zhou Minxuan, Imani M, Gupta S, et al. GRAM: Graph processing in a ReRAM-based computational memory [C] //Proc of the 24th Asia and South Pacific Design Automation Conf. New York: ACM, 2019: 591-596

[55] Huang Yu, Zheng Long, Liao Xiaofei, et al. RaGRa: Leveraging monolithic 3D ReRAM for massively-parallel graph processing [C] //Proc of the Design, Automation & Test in Europe Conf & Exhibition (DATE). Piscataway, NJ: IEEE, 2019: 1273-1276

[56] Han Lei, Shen Zhaoyan, Shao Zili, et al. A novel ReRAM-based processing-in-memory architecture for graph computing [C] //Proc of the 6th IEEE Non-Volatile Memory Systems and Applications Symp (NVMSA). Piscataway, NJ: IEEE, 2017: 1-6

[57] Ahn J, Yoo S, Mutlu O, et al. PIM-enabled instructions: A low-overhead, locality-aware processing-in-memory architecture [C] //Proc of the 42nd Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2015: 336-348

[58] Lee J, Kim H, Yoo S, et al. ExtraV: Boosting graph processing near storage with a coherent accelerator [J]. Proceedings of the VLDB Endowment, 2017, 10(12): 1706-1717

[59] Jun S W, Wright A, Zhang Sizhuo, et al. GraFBoost: Using accelerated flash storage for external graph analytics [C] //Proc of the 45th ACM/IEEE Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2018: 411-424

[60] Matam K K, Koo G, Zha Haipeng, et al. GraphSSD: Graph semantics aware SSD [C] //Proc of the 46th Int Symp on Computer Architecture. New York: ACM, 2019: 116-128

[61] Zheng Da, Mhembere D, Burns R, et al. FlashGraph: Processing billion-node graphs on an array of commodity SSDs [C] //Proc of the 13th USENIX Conf on File and Storage Technologies (FAST 2015). Berkeley, CA: USENIX Association, 2015: 45-58

[62] Elyasi N, Choi C, Sivasubramaniam A. Large-scale graph processing on emerging storage devices [C] //Proc of the 17th USENIX Conf on File and Storage Technologies (FAST 2019). Berkeley, CA: USENIX Association, 2019: 309-316

[63] Wilton S J E, Jouppi N P. CACTI: An enhanced cache access and cycle time model [J]. IEEE Journal of Solid-State Circuits, 1996, 31(5): 677-688

[64] Hashemi M, Ebrahimi E, Mutlu O, et al. Accelerating dependent cache misses with an enhanced memory controller [C] //Proc of the 43rd ACM/IEEE Annual Int Symp on Computer Architecture (ISCA). Piscataway, NJ: IEEE, 2016: 444-455

[65] Dysart T, Kogge P, Deneroff M, et al. Highly scalable near memory processing with migrating threads on the Emu system architecture [C] //Proc of the 6th Workshop on Irregular Applications: Architecture and Algorithms (IA3). Piscataway, NJ: IEEE, 2016: 2-9

[66] WikiChip Fuse. DAPRA ERI: HIVE and Intel PUMA graph processor [EB/OL]. [2020-02-25]. https://fuse.wikichip.org/news/2611/darpa-eri-hive-and-intel-puma-graph-processor/

[67] Hybrid Memory Cube Consortium. Hybrid memory cube specification 2.1[EB/OL]. [2020-02-25]. https://www.nuvation.com/sites/default/files/Nuvation-Engineering-Images/Articles/FPGAs-and-HMC/HMC-30G-VSR_HMCC_Specification.pdf

[68] Shevgoor M, Kim J S, Chatterjee N, et al. Quantifying the relationship between the power delivery network and architectural policies in a 3D-stacked memory device [C] //Proc of the 46th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2013: 198-209

[69] Cakir C, Ho R, Lexau J, et al. Modeling and design of high-radix on-chip crossbar switches [C] //Proc of the 9th Int Symp on Networks-on-Chip. New York: ACM, 2015: 20:1-20:8

[70] RapidIO. RapidIO Trade Association [EB/OL]. [2020-02-25]. http://www.rapidio.org

[71] Statista. Global social media ranking statistic [EB/OL]. [2020-02-25]. https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/

[72] Schlapka A. Micronannounces shift in high-performance memory roadmap strategy [EB/OL]. [2020-02-25]. https://www.micron.com/about/blog/2018/august/micron-announces-shift-in-high-performance-memory-roadmap-strategy

[73] Kulkarni M, Burtscher M, Inkulu R, et al. How much parallelism is there in irregular applications? [J]. ACMSIGPLAN Notices, 2009, 44(4): 3-14

[74] Zhang Zhao, Zhu Zhichun, Zhang Xiaodong. A permutation-based page interleaving scheme to reduce row-buffer conflicts and exploit data locality [C] //Proc of the 33rd Annual ACM/IEEE Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2000: 32-41

[75] Zeng Hanqing, Prasanna V. GraphACT: Accelerating GCN training on CPU-FPGA heterogeneous platforms [J]. arXiv preprint, arXiv:2001.02498, 2019

[76] Jure L, Andrej K. SNAP Datasets: Stanford large network dataset collection [DB/OL]. [2020-02-25]. http://snap.stanford.edu/data

[77] Texas A&M University. SuiteSparse matrix collection [DB/OL]. [2020-02-25]. https://sparse.tamu.edu

[78] Ryan R, Nesreen A. GraphVis: Interactive visual graph mining and machine learning [DB/OL]. [2019-11-10]. http://networkrepository.com

[79] Kristian K, Nils M, Petra M, et al. Benchmark data sets for graph kernels [DB/OL]. [2020-02-25]. http://graphkernels.cs.tu-dortmund.de

[80] Paszke A, Gross S, Massa F, et al. PyTorch: An imperative style, high-performance deep learning library [C] //Proc of the Advances in Neural Information Processing Systems. Cambridge, MA: MIT Press, 2019: 8024-8035

[81] Abadi M, Barham P, Chen Jianmin, et al. TensorFlow: A system for large-scale machine learning [C] //Proc of the 12th USENIX Symp on Operating Systems Design and Implementation (OSDI 2016). Berkeley, CA: USENIX Association, 2016: 265-283

Yan Mingyu, born in 1990. PhD. Student member of CCF. His main research interests include graph-based hardware accelerator and dataflow architecture.

严明玉,1990 年生.博士,CCF 学生会员.主要研究方向为基于图的硬件加速器和数据流架构.

Li Han, born in 1994. PhD candidate. Student member of CCF. Her main research interests include computer architecture and graph-based hardware accelerator. (lihan-ams@ict.ac.cn)

李 涵,1994 年生.博士研究生,CCF 学生会员.主要研究方向为计算机体系结构和基于图的硬件加速器.

Deng Lei, born in 1990. Postdoctoral fellow at University of California, Santa Barbara, CA, USA. His main research interests include brain-inspired computing, machine learning, neuromorphic chip, computer architecture, tensor analysis, and complex networks. (leideng@ucsb.edu)

邓 磊,1990 年生.美国加州大学圣塔芭芭拉分校博士后.主要研究方向为类脑计算、机器学习、神经形态芯片、计算机体系结构、张量分析和复杂网络.

Hu Xing, born in 1988. Postdoctoral fellow at University of California, Santa Barbara, CA, USA. Her main research interests include domain-specific architectures and deep learning system security. (xinghu@ucsb.edu)

胡　杏,1988 年生.美国加州大学圣塔芭芭拉分校博士后.主要研究方向为特定领域架构和深度学习系统安全.

Ye Xiaochun, born in 1981. PhD, associate professor. Member of CCF. His main research interests include software simulation, algorithm paralleling and optimizing, and architecture for high-performance computer.

叶笑春,1981 年生.博士,副研究员,CCF 会员.主要研究方向为软件仿真、算法并行和优化、高性能计算机架构.

Zhang Zhimin, born in 1963. Professor and PhD supervisor. Senior member of CCF. His main research interests include SoC design and computer architecture. (zzm@ict.ac.cn)

张志敏,1963 年生.研究员,博士生导师,CCF 高级会员.主要研究方向为 SoC 设计和计算机体系结构.

Fan Dongrui, born in 1979. PhD, professor, PhD supervisor. Distinguished member of CCF. His main research interests include many-core processor design, high throughput processor design and low power micro-architecture. (fandr@ict.ac.cn)

范东睿,1979 年生.博士,研究员,博士生导师,CCF 杰出会员.主要研究方向为众核处理器设计、高通量处理器设计和低功耗微架构.

Xie Yuan, born in 1974. Received his PhD degree from Electrical Engineering Department, Princeton University in 2002. Professor. Fellow of ACM and IEEE. His main research interests include VLSI design, electronic design automation, computer architecture, and embedded systems design. (yuanxie@ucsb.edu)

谢　源,1974 年生.于 2002 年在普林斯顿大学电气工程学院取得博士学位.教授,ACM 和 IEEE 会士.主要研究方向为 VLSI 设计、电子设计自动化、计算机体系结构和嵌入式系统设计.