计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202440073
61(6): 1401−1428, 2024

面向多核 CPU 与 GPU 平台的图处理系统关键技术综述

张 园[1,2] 曹华伟[1,3] 张 婕[1] 申 玥[1,3] 孙一鸣[1,2] 敦 明[1] 安学军[1,2] 叶笑春[1]
[1](处理器芯片全国重点实验室(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](中国科学院大学南京学院 南京 211135)
(zhangyuan-ams@ict.ac.cn)

Survey on Key Technologies of Graph Processing Systems Based on Multi-core CPU and GPU Platforms

Zhang Yuan[1,2], Cao Huawei[1,3], Zhang Jie[1], Shen Yue[1,3], Sun Yiming[1,2], Dun Ming[1], An Xuejun[1,2], and Ye Xiaochun[1]
[1](*State Key Lab of Processors* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190)
[2](*University of Chinese Academy of Sciences*, *Beijing* 100049)
[3](*University of Chinese Academy of Sciences*, *Nanjing* 211135)

Abstract As a key technique for analyzing and mining relationships, graph computing has been widely used in smart healthcare, social network analysis, financial anti-fraud, road navigation, computational sciences, and others. In recent years, with the development of parallel structures, memory access methods, interconnection structures, and synchronization mechanisms of general-purpose CPU and GPU architecture, multi-core CPU and GPU have become common platforms for accelerating graph processing. However, there are significant challenges to graph processing in terms of improving performance and achieving high scalability, such as the large scale and irregular distribution of data, complex data dependence, high communication-to-computation ratio, and dynamic changes of vertices and edges. To address the above challenges, a large number of graph processing systems based on multi-core CPU and GPU platforms are proposed and achieve good results. To provide readers with a comprehensive understanding of graph processing optimizations on multi-core CPU and GPU platforms, we first present the basic concepts, including the graph data structures, the typical graph algorithms, and the characteristics of graph application. Then the challenges of graph processing are clarified. After that, we present the existing graph processing systems based on multi-core CPU and GPU platforms. From the perspective of accelerated graph processing design, we summarize the key technology optimizations in detail and systematically, including pre-processing, memory access optimization, computing acceleration, and overhead reduction of data communication. Finally, we analyze the performance and scalability of state-of-art graph processing and conclude the future development trend of graph processing based on different perspectives, which expects to bring certain inspiration to relevant researchers to explore high-performance graph processing system.

Key words multi-core CPU and GPU platform; graph processing systems; graph data representation; load balancing; irregular memory access; dynamic graph processing

收稿日期: 2024-02-01; **修回日期**: 2024-03-06

基金项目: 国家重点研发计划项目(2023YFB4502305); 北京市自然科学基金项目(4232036); CAS 青年创新促进会项目
This work was supported by the National Key Research and Development Program of China (2023YFB4502305), the Beijing Natural Science Foundation (4232036), and the CAS Project for Youth Innovation Promotion Association.

通信作者: 曹华伟(caohuawei@ict.ac.cn)

摘 要 图计算作为分析与挖掘关联关系的一种关键技术，已在智慧医疗、社交网络分析、金融反欺诈、地图道路规划、计算科学等领域广泛应用. 当前，通用 CPU 与 GPU 架构的并行结构、访存结构、互连结构及同步机制的不断发展，使得多核 CPU 与 GPU 成为图处理加速的常用平台. 但由于图处理具有处理数据规模大、数据依赖复杂、访存计算比高等特性，加之现实应用场景下的图数据分布不规则且图中的顶点与边呈现动态变化，给图处理的性能提升和高可扩展性带来严峻挑战. 为应对上述挑战，大量基于多核 CPU 与 GPU 平台的图处理系统被提出，并在该领域取得显著成果. 为了让读者了解多核 CPU 与 GPU 平台上图处理优化相关技术的演化，首先剖析了图数据、图算法、图应用特性，并阐明图处理所面临的挑战. 然后分类梳理了当前已有的基于多核 CPU 与 GPU 平台的图处理系统，并从加速图处理设计的角度，详细、系统地总结了关键优化技术，包括图数据预处理、访存优化、计算加速和数据通信优化等. 最后对已有先进图处理系统的性能、可扩展性等进行分析，并从不同角度对图处理未来发展趋势进行展望，希望对从事图处理系统研究的学者有一定的启发.

关键词 多核 CPU 与 GPU 平台；图处理系统；图数据表示；负载均衡；不规则访存；动态图处理

中图法分类号 TP301

图作为表达实体及实体间联系的一种基本数据结构，已在互联网、金融、医疗及脑科学等领域有着广泛的应用. 图计算作为解决关联数据分析的一种重要工具，已成为下一代人工智能的核心技术，并得到学术界和工业界的广泛关注. 随着大数据时代的快速发展，图数据呈爆炸式增长，且图中的顶点与边信息实时变化，这给图数据处理带来严峻挑战. 为实现图数据的高效分析与挖掘，越来越多的工作致力于研究面向多核 CPU 和专用 GPU 加速器的图处理方法，旨在通过充分利用硬件资源的计算、存储、互连及同步机制等特性提高图处理的性能.

自 2010 年谷歌提出大规模分布式图处理系统 Pregel[1] 以来，图处理加速研究在通用多核 CPU 与 GPU 平台上呈现出井喷式发展. 学术界先后提出了 Ligra[2]，Galois[3]，GraphIt[4]，GraphChi[5]，GridGraph[6]，Gemini[7]，FLASH[8]，Gunrock[9-10]，Tigr[11]，SEP-graph[12]，Subway[13]，HyTGraph[14]，Groute[15-16]，GUM[17]，EvoGraph[18]，GraphBolt[19]，GraphOne[20]，TEGRA[21] 等图处理系统. 在工业界，为了高效挖掘应用数据背后的深层信息，现有工作不断致力于研发高性能图处理系统，如 Giraph[22]，GraphX[23]，GRAPE[24]，Plato[7,25-26]，GraphScope[27]，TuGraph Analytics[28] 等，并在此基础上围绕图计算的典型应用场景，如社交网络分析、金融风控、商品推荐、道路规划等开展落地工作. 可见，图处理系统的研发具有极高的研究与应用价值.

与传统的应用任务不同，图处理任务具有访存密集且不规则、数据高度依赖、通信计算比高等特性. 因此，图处理系统需考虑数据的高效存储和快速计算 2 部分. 在存储方面，由于图数据高度稀疏且分布不均匀，通常基于压缩存储格式进行设计. 此外，对于动态变化的图数据，需要综合考虑存储、查询、更新效率等，可使用基于组合格式的方法进行存储. 在计算方面，图算法主要包括计算、更新和同步 3 个阶段. 其中，计算阶段根据具体的算法执行基本操作；更新阶段根据计算结果修改相关状态和属性信息；同步阶段则将更新后的数据以通信的方式在不同的子任务间进行同步，以保证算法执行的正确. 对于图处理系统，数据的表示与存储、算法的并行策略、编程模型的选择以及通信机制的设计与使用等都至关重要.

多核 CPU 与 GPU 硬件的快速更新给图处理的加速带来了机遇与挑战. 硬件设备计算与存储资源的升级，使得图处理性能显著提升. 但由于图计算任务是典型的访存不规则任务，且现实图数据规模在急剧增加并动态变化，图处理系统仍面临着资源利用率低、性能难以满足实际应用需求、系统可扩展性差等诸多问题. 因此，设计并研发高性能的图处理系统，需要深入分析图算法的执行模式和系统运行时的特性，并充分利用新型硬件的存储与计算资源等.

数字经济时代的到来，使得如谷歌搜索引擎、百度地图、亚马逊购物平台等关键应用背后的数据越来越复杂，亟需利用图处理相关技术来挖掘其背后的潜在商用价值，这些加快了图计算在学术界的成果产出，并推动了产业界相关技术与平台的落地应用. 为了使读者和相关领域研究人员了解图处理系统领域的发展过程，本文首先对图处理的基础概念，包括分层框架、数据表示、典型算法、应用场景、面临的挑战和已有的图处理编程模型及典型图处理系

统等进行详细介绍. 然后以设计高性能、可扩展、易编程的图处理系统为出发点, 从硬件平台选型、整体框架设计、统一编程接口实现等方面, 针对具体图数据预处理、访存优化、计算加速、降低通信开销等关键技术进行详细而系统地分析与阐述. 最后对已有先进图处理系统的性能、可扩展性等进行分析, 并从不同角度对图处理系统的未来发展趋势进行展望, 期望能为该领域的研究人员带来一定的启发.

当前已有的图处理应用领域的综述论文从不同角度对图处理算法及系统框架进行总结与分析. 文献 [29] 对 GPU 平台上的图处理系统的框架、编程模型和优化关键技术进行介绍, 并实验验证了对于不同系统使用相关优化所带来的性能提升. 文献 [30] 针对单机多核系统下的核外(out-of-core)大规模图处理系统进行阐述, 梳理了该领域优化所使用的关键技术并阐明其未来发展趋势. 文献 [31] 基于异构架构, 对图计算内存系统的管理及优化方法进行梳理与归纳, 并总结了图计算在内存方面的机遇与挑战. 文献 [32] 基于加速器平台, 阐述了图计算加速设计的 3 个关键技术, 即预处理、并行图计算和运行时调度. 文献 [33] 通过梳理使用高性能和并行计算技术来优化图计算过程的相关工作, 以阐明图计算体系结构和系统软件关键技术. 在实际应用中, 图数据往往呈现动态变化, 从而使得动态图处理研究深受产业界关注, 也带动了学术界相关成果的产出. 文献 [34] 针对动态图处理系统的概念与分类、模型和并行策略进行归纳, 阐明了动态图处理所面临的挑战. 文献 [35] 以图在实际中如何使用为出发点, 通过不同方式, 如邮件、Slack 通道、Twitter 平台等收集信息并展开相关调查, 以了解并分析图数据的类型、图计算的执行策略、用户所使用的图计算软件类型, 以及图处理过程中用户所面临的挑战, 来启发并指导图处理领域相关研究. 与前述工作侧重点不同的是, 本文针对多核 CPU 与 GPU 平台上高性能、易扩展的静态图和动态图处理系统, 分析其涉及到的关键技术并进行系统性分析与总结, 希望能为该领域的研究人员带来一定的启发.

1 基本概念

本节首先对图处理系统分层框架进行简要介绍, 然后对图数据存储结构、典型图算法和图计算应用等基础概念进行描述.

1.1 图处理系统分层框架

图处理系统被广泛应用在各大关键应用中, 其对应的分层框架如图 1 所示. 最底层是硬件平台层, 对应的典型硬件平台包括中央处理器(central processing unit, CPU)、图形处理器(graphics processing unit, GPU)、现场可编程门阵列(field programmable gate array, FPGA)和专用集成电路(application specific integrated circuit, ASIC)等. 此外, 还存在存内计算(processing in memory, PIM)和粗粒度可重构架构(coarse-grained reconfigurable architecture, CGRA)等平台. 平台层的核心部件主要为存储单元、计算单元、内部互连结构等. 上一层对应的是关键技术层, 其核心技术包括图数据的预处理与表示、数据的访存、负载划分及数据通信等. 再上 2 层对应的是图处理编程模型和对外提供的应用编程接口. 当前, 图处理编程模型主要包括以点为中心、以边为中心、以子图为中心等编程模型. 对于动态图, 其在并行计算时, 通常采用点中心或边中心的编程模型并结合增量计算的思想来处理, 简称“增量计算模型”. 而最上面

Fig. 1 The layer framework of graph processing system

图 1 图处理系统分层框架

2层,对应的是图处理系统的典型算法和应用领域.典型的图算法包括宽度优先搜索(breadth first search, BFS)、单源最短路径(single source shortest path, SSSP)、网页排名(pagerank, PageRank/PR)、连通分量(connected components)、介数中心性(betweenness centrality, BC)、标签传播(label propagation, LP)、信念传播(belief propagation, BP)、三角形计数(triangle counting, TC)、k核(k-core)子图等.图处理系统应用前景广阔,已在金融、医疗、交通、互联网等领域得到广泛应用.

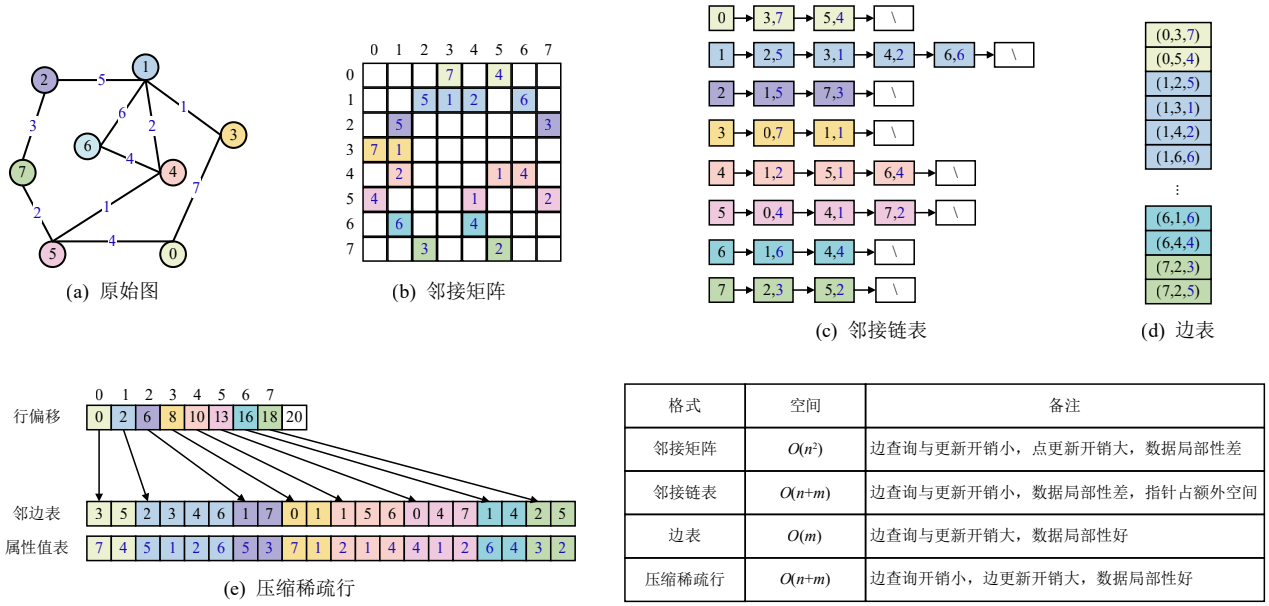
1.2 图数据的表示

图作为表示实体与实体间关联关系的一种抽象数据结构,由顶点和边组成,通常表示为$G=(V,E)$,其中$V=\{v_1,v_2,\cdots\}$代表图中顶点的集合,$E=\{e_1,e_2,\cdots\}$代表图中边的集合.对于一个无向图,如果顶点v_i到顶点v_j存在一条边,则其边表示为$e(v_i,v_j)$.如果图中边存在权重属性,则图可表示为$G=(V,E,W)$,其$W=\{w_{e_1},w_{e_2},\cdots\}$为边的权重集合,$w_{e_i}$表示图中边$e_i$对应的权重值.真实应用背后抽象的图数据往往表现为稀疏、幂律分布、小世界等特性[36-38].为了更好地存储和表达图数据,图处理系统通常使用邻接矩阵、压缩稀疏行/列、邻接链表、边表等存储格式,典型举例如图2所示.图2及本节中,n和m分别表示图中的顶点数和边数.

邻接矩阵(adjacency matrix, AM),其采用$n\times n$的矩阵$\boldsymbol{M}$表示,当顶点i与顶点j之间存在一条边,则$M(i,j)=1$,否则$M(i,j)=0$.采用此种存储格式,其对应的空间复杂度为$O(n^2)$.由于图数据往往表现为稀疏性,使得矩阵中有效信息占比少,从而导致存储空间效率差.虽然向邻接矩阵插入和删除边可在$O(1)$时间完成,但当图中顶点增加时,会导致存储结构的整体改变,带来图数据更新的高开销,因此在图处理系统中,该存储格式应用较少.但随着图神经网络的兴起,越来越多的工作集中在研究稀疏矩阵向量乘与稀疏矩阵矩阵乘等的加速[39-40],使得该结构在图神经网络相关研究中得到较多应用.

压缩稀疏行(compressed sparse row, CSR)是一种高效的图数据存储格式,已被广泛应用在图处理系统中[2,3,7,9,10,14,41-42].CSR主要由行偏移(row offset)、邻边表(adjacency list)和属性值表(value list)组成.其中,行偏移存储每个顶点的第1个邻居在邻边表中的偏移位置;邻边表依次存储图中每个顶点的所有邻居信息;属性值表,则对应存储图中边的属性信息.使用CSR格式存储图数据,其所占用内存空间为$O(n+m)$,具有存储高效且数据访问局部性好等优点,因此被广泛应用在图处理系统中.但该结构的灵活性较差,在向图数据插入或删除顶点与边时,需要移动相关数据,更新速度慢.类似的结构还有压缩稀疏列(compressed sparse column, CSC)和坐标(coordinate, COO)表示.

邻接链表(adjacency list, AL)是图存储的一种常见格式,它使用一个数组存储图中所有的顶点信息.而对于每个顶点的邻边信息,则使用链表进行表示,它需要$O(n+m)$的存储空间,具有灵活性好的优点,



格式	空间	备注
邻接矩阵	$O(n^2)$	边查询与更新开销小,点更新开销大,数据局部性差
邻接链表	$O(n+m)$	边查询与更新开销小,数据局部性差,指针占额外空间
边表	$O(m)$	边查询与更新开销大,数据局部性好
压缩稀疏行	$O(n+m)$	边查询开销小,边更新开销大,数据局部性好

Fig. 2 Illustration of graph data representations

图2 图数据表示示意图

可以在$O(1)$时间内完成边的插入操作，在$O(d)$时间内完成边的删除操作，其中d为该顶点的边数. 但邻接链表局部性差，难于并行，且数据检索耗时. 此外，存储指针数据信息会带来额外存储空间开销，故较少应用在静态图处理系统中，在动态图处理中，通常会结合其它存储结构共同使用.

边表（edge list, EL）使用连续存储空间依次记录图中所有边的信息，包括边的源顶点、目的顶点和属性值信息，它需要$O(m)$的存储空间. 使用边表存储图信息，具有一定的灵活性，更新简单、并行性好，但是局部性差，数据查询效率低. 少量的核外图处理和动态图处理系统使用该存储格式.

当前，静态图处理系统中，图数据的存储通常采用 CSR 存储格式，少数图处理系统使用邻接矩阵、边表等格式存储. 对于动态图处理系统，除了考虑图数据的高效存储，还要兼顾图数据更新速度，故一些研究集中在优化图数据的表示与存储[20,43-46]，从而提高动态图处理的性能.

1.3 图算法

在图处理系统中，图算法提供了关联数据分析与挖掘的基本方法，该方法描述了如何处理图数据，以发现一些定性或者定量的结论. 图处理系统的优化与图算法息息相关. 当前，图处理领域，依据算法特性的不同，重点关注三大类核心算法，分别为路径搜索类、中心性计算类和社区发现类. 路径搜索类算法以搜索为基础，通过遍历来探索顶点间的最优路径，对应的典型算法为 BFS, SSSP 等. 中心性计算类算法通过计算来识别图中重要的顶点及其对图拓扑结构的影响等，对应的典型算法包括 PageRank, BC 等. 社区发现类算法通过评估顶点间的关系来发现图中的群体行为或偏好等，常见的算法为 TC, k-core, LP, BP 等.

BFS 是一种基本的图遍历算法，它是 Graph500 基准的核心函数之一[47]. 该算法的基本思想是：随机选取图中的一个顶点作为起始顶点，然后逐层迭代遍历所有可达的邻居顶点，以形成宽度优先搜索树. 该算法的遍历方式包括“自上而下（top-down, TD）”和“自下而上（bottom-up, BU）”和“混合（hybrid）”3 种. 当前，由于混合方式综合了 TD 和 BU 方式的优点，因而得到了广泛的研究与使用[2,12,42,48-49].

SSSP 是另一种基本的图遍历算法，该算法也是 Graph500 基准的核心函数之一[47]. 该算法的基本思想是：计算从一个顶点出发到图中其它所有顶点的最短路径距离值. 基本的单源最短路径算法包括迪杰斯特拉（Dijkstra）算法[50]和贝尔曼-福特（Bellman-Ford）算法[51]. Dijkstra 算法工作效率高但并行性差，Bellman-Ford 算法并行性好但工作效率差. 为权衡算法的工作效率和并行性，德尔塔步进（Δ-stepping）算法[52]被提出，并得到广泛研究[53-56].

PageRank 是最流行的图处理算法之一，被广泛地应用在搜索引擎领域，用来分析网页的相关性和重要性. 该算法的基本原理是：在图结构中，如果 1 个顶点与其他多个顶点存在边相连，则说明该顶点较为重要，对应的排名也较高. 算法的执行过程为：循环迭代计算图中每个顶点的网页排名值，直到算法收敛或达到限制的迭代次数. PageRank 经常需要大量的访存带宽，同时要求硬件具有浮点计算能力. 为提高 PageRank 性能，相关研究提出传播块[57]和划分中心的处理[58]等优化方法.

TC 用于统计图中三角形的数目. 对于该算法，一个顶点不但要访问它自己的所有邻居顶点，还需要访问它的邻居顶点的所有邻居顶点，即需要访问当前活跃顶点的每个二阶邻居顶点，故该算法在计算过程中，存储访问量、网络消息量等较直接邻居图算法明显增长，使得其高效实现面临着更大的挑战. 该算法在金融反欺诈、垃圾邮件检测等方面得到广泛的使用. 当前图处理领域，部分工作致力于基于 CPU 或 GPU 平台来加速该算法的处理[59-60].

1.4 图计算应用

在现实世界中，很多关键应用背后的信息都可以抽象成图数据，并进一步使用图处理系统进行分析和挖掘潜在信息. 图计算在社交网络分析、金融反欺诈、道路导航、电商推荐、链接预测、生物制药等方面存在广泛应用.

在社交网络应用中，用户可以抽象成顶点，用户之间的关系可以抽象成边，如关注与被关注、好友关系等. 近年来，社交网络规模越来越大，通过将社交网络抽象成图数据，可以将图处理算法应用其上，如使用 BFS 算法寻找 n 跳好友，使用 PageRank 算法寻找社交网络中的“名人”等. 文献 [61] 介绍了社交网络分析的基本概念、方法和工具，以及如何用图表示社交网络中的顶点和关系. 文献 [62] 证实了社交网络图的幂律性、无标度性和小世界性等典型特征.

在金融反欺诈领域，交易账户可以抽象成顶点，交易可以抽象成边，如转账、汇款等. 基于抽象出的图数据，图处理算法可以挖掘个体之间隐藏的关联关系，如企业之间的股权关系、不同客户之间的交易链路等，从而应用于反洗钱、交易风险预测、团伙欺

诈等领域. 文献 [63] 设计并实现了基于 GPU 平台的 LP 算法框架，用于处理企业的大规模 LP 工作负载，以进行实时欺诈检测. 文献 [64] 引入了增量欺诈检测框架 Spade，用于解决实际应用中基于动态图的欺诈检测开销大的问题.

在道路规划领域，地点可以抽象成顶点，地点之间的路线信息可以抽象成边，如距离、时间、耗费等. 基于抽象出的图数据，可以使用 SSSP 算法计算出两地之间的最短路径、耗时最短路径、花费最短路径等. 文献 [65] 给出了使用 Dijkstra 算法寻找道路网络中最短路径的例子. 文献 [66] 使用改进的 PageRank 算法来获取监控摄像头的影响权重，并提出一种动态图算法来从交通视频中寻找同行车辆.

在电商推荐领域，用户及商品等信息可抽象为顶点，用户与用户间、用户与商品间的关系可抽象成边，如用户好友关系、用户收藏商品、用户购买商品等. 基于抽象出的图数据，通过分析以往交易中的用户和购买商品关系，可以将用户购买的商品推荐给具有相似购买关系的其他用户. 此外，也可根据商品和商品之间的关系进行相关推荐. 文献 [67] 提出一种图模型表示用户-商品信息，将用户或商品表示为顶点，将交易或相似关系表示为边，并支持 3 种推荐方法. 文献 [68] 通过商品之间连接的边判断商品之间是否存在互补或替代关系，之后根据边的方向判断相关产品品质的优劣，从而构建商品的替代和互补关系网络，实现互补和替代品的推荐. 文献 [69] 介绍了淘宝如何使用基于图的方法来构建大规模的商品图，以提高电商推荐中的准确性、多样性和效率.

2 图处理系统面临的挑战

现实应用背后抽象的图数据，往往具有高度稀疏、分布不均匀、数据规模大、数据动态变化等特性[36-38,61]，同时，图处理是典型的访存密集且不规则任务，这使得图处理加速面临着诸多困难. 此外，当前通用的多核 CPU 与 GPU 平台在执行图处理任务的过程中表现出了各自的优势和不足. 对此，本节将围绕通用硬件平台 CPU 和 GPU，从图数据存储与访存、图算法执行、图处理系统可扩展性等方面展开分析，进而阐述图处理加速设计中面临的难点与挑战.

2.1 图数据存储与访存

当前，图数据规模正呈爆炸式增长，据 2022 年 9 月脸书(Facebook)公司最新发布的数据显示，该公司社交网络图数据规模已包含超过 29.34 亿个顶点以及千亿条边. 针对超大规模的图数据，单机多核 CPU 和 GPU 系统由于内部存储空间有限，往往需要依赖外部存储资源. 但由于不同级别的存储系统在访存延迟和访存带宽上存在明显区别，使得图处理系统中数据的存储与访问成为了加速图处理的难点之一. 此外，在实际应用的业务场景中，图中的边与顶点信息动态变化，使得静态图处理系统中的存储机制往往不适用于动态图处理系统. 动态图数据的存储要考虑数据查询与更新等多种因素，设计较难[70-71].

图处理过程中的数据大体可以分为 2 类：一类是图的原始数据；另一类是图处理过程中频繁访问的数据，如 BFS 算法中存储父顶点信息的数组、SSSP 算法中存放距离信息的数组等. 对于原始的图数据，静态图处理系统大多采用 CSR 格式进行高效存储，但在不进行优化的情况下，对处理过程中频繁访问的数据，往往存在访存分歧，导致算法执行过程中出现缓存(cache)缺失率高、系统访存效率低等问题. 此外，图数据的幂律分布特性[36]，即少数顶点连接大多数的边，而多数顶点只连接少数的边，往往会导致有限的缓存不能很好地处理度数高的顶点，从而使得高速存储设备缓存等不能得到充分利用. 因此，在图处理系统中，如何减少冗余存储、提高图数据的空间与时间局部性等对系统的性能提升至关重要.

2.2 图算法执行

图算法的执行，往往采用层级迭代处理的方式. 在算法执行过程中，基于不同的顶点运行相同的算法所依赖的关系往往是不可预测的，这会导致算法执行过程中存在大量冗余的计算和不规则访存，从而使得算法执行效率低. 图数据的依赖复杂特性，使得数据划分后难以达到任务均衡且通信开销小的目的. 同时，图数据的幂律分布特性，易导致算法执行过程中出现负载不均衡、计算资源利用率低等问题. 图算法在并行处理过程中存在条件竞争，为了确保算法执行的正确性，往往使用锁或原子操作来解决这一问题，但这些操作开销大，影响图处理系统性能的提升. 此外，图算法执行过程中通常采用同步或异步方式执行. 同步方式执行过程中，存在明显的同步屏障开销和“木桶”效应. 异步方式执行会存在数据竞争，且需频繁的数据通信使得通信开销大、硬件资源利用率低，造成图算法处理性能低下.

此外，不同的图算法在执行过程中表现出不同的特性，相同的算法在不同的层处理过程中也表现出不同的特性，故算法在执行过程中对系统资源的需求存在差异，这使得算法优化设计需要兼顾算法

特性和系统运行时等信息. 但 CPU 与 GPU 平台运行时信息难以收集, 往往需要依靠代价模型进行估算, 这不但引入了额外开销, 而且精确性难以保障. 因此, 设计高效的图算法执行策略是图处理系统的难点之一.

2.3 图处理可扩展性

近年来, 图数据规模不断增长, 为了满足实际应用的需求, 图处理系统亟需存储容量大、计算资源充足的硬件平台来加速. 但随着存储带宽和计算资源的升级, 图处理系统却呈现了较差的可扩展性. 单机存内图处理系统难以处理超大规模的图数据, 且很难扩展成单机核外及分布式图处理系统. 受限于多级存储资源的访存延迟和带宽差别, 以及单机系统有限的计算资源, 单机核外图处理系统难以处理超大规模图数据. 而多节点分布式图处理系统面临着数据通信开销大、系统负载不均衡等问题, 使得随着节点数目的增多, 性能提升较少.

此外, 静态图处理系统在设计与实现中, 未考虑图数据的动态变化特性, 使得其不适用于变化的实际应用场景. 相对于静态图数据的处理, 动态图数据的处理往往需要结合静态图和图变化的特性, 在存储和计算方面进行重新设计, 以实现高效的动态图处理. 在可编程方面, 现有的开源图处理系统, 往往需要使用者掌握一定的编程能力才能进行修改, 其可扩展差、可开发难度大.

3 图处理编程模型

图处理基本的编程模型包括: 以点为中心[1,2,5,8,11,13,72-75]、以边为中心[76-77]、以数据为中心[9-10]、以子图为中心的编程模型[78], 和基于矩阵向量乘[39-40,79]、基于增量计算的模型[19,21,80-83] 等. 在此基础上, 形成了一些典型的图处理编程模型, 包括整体同步并行(bulk synchronous parallel, BSP)计算编程模型[1] 和收集—应用—扩散(gather-apply-scatter, GAS)编程模型[72]. 此外, GPU 平台出现了初始—计算—更新(initialize-compute-update, ICU)编程模型[73]、推进—过滤—计算(advance-filter-compute, AFC)编程模型[9-10]、活跃—计算—联合(active-compute-combine, ACC)编程模型[42] 等. 本节将对图处理基本编程模型和 BSP 及 GAS 典型编程模型进行具体介绍, 并总结其在 CPU 与 GPU 平台静态和动态图处理方面的应用情况及各自优缺点等.

3.1 基本编程模型

3.1.1 6 种基本编程模型

1)点中心编程模型

以点为中心(vertex-centric)的编程模型的核心思想是“像点一样思考(think-like-a-vertex)”, 其最早在 Pregel[1] 系统中应用. 图中顶点可以修改自己的状态、接收来自邻居顶点的信息、向邻居顶点发送信息等. 在这种编程模型中, 图算法被分为多个迭代, 在每轮迭代中, 多个活跃顶点的数据计算和传递是相互独立的, 因此可以并行处理, 并行度较高. 同时, 由于每轮迭代中活跃顶点的信息只能传播给邻居顶点, 算法需要多轮迭代才能达到收敛状态. 点中心编程模型具有很强的表达能力和高并行特性, 故在静态图处理和动态图处理系统中拥有广泛的应用. 典型的点中心图处理系统包括: Ligra[2], GraphChi[5], FLASH[8], CuSha[73], Tigr[11], GraphBolt[19] 等. 但点中心编程模型存在大量随机访存, 从而导致存储带宽利用率低.

2)边中心编程模型

以边为中心(edge-centric)的编程模型由 X-Stream[76] 提出, 它对边进行流(stream)式访问, 执行基于边迭代的扩散(scatter)和收集(gather)操作. 对于边数远大于顶点数的图, 图处理开销由对边的访问和更新开销决定, 因此与随机访问边相比, 流式访问边可使系统性能更优. 同时, 使用流式分区降低了对顶点的随机访问开销, 这种方法对核外图处理较适用.X-Stream[76], WolfGraph[84], ShenTu[77], 以及一些基于 FPGA 的图处理系统应用此编程模型较多.

3)子图中心编程模型

以子图为中心(subgraph-centric)的编程模型由 Giraph++[78] 提出, 其核心思想是“像图一样地思考(think-like-a-graph)”. 在分布式图处理系统中, 使用以点为中心的编程模型隐藏了分区信息, 无法实现对许多特定算法的优化. 以子图为中心的编程模型专注于子图, 旨在充分利用分区内的通信, 从而避免消息传递或调度机制带来的高开销, 因此收敛速度较快. Subway[13] 中异步执行模式下, 采用了以子图为中心的思想对图数据进行处理, 减少了 CPU 与 GPU 间数据传输开销, 并加快了算法收敛速度, 使得图处理性能优于同步执行模式下的性能.

4)数据中心编程模型

以数据为中心(data-centric)的编程模型由 Wang 等人[9-10] 提出, 主要针对当前活跃集合进行操作, 包括要参与计算的活跃顶点集合或活跃边集合. 以数据为中心的编程模式支持活跃顶点集合和活跃边集合的切换, 例如从现有的活跃顶点集合生成一个新的活跃邻边集合. 为处理活跃集合, Wang 等人设计了推进(advance)、过滤(filter)、计算(compute)这 3

个步骤, 分别用于从当前活跃集合生成新的活跃集合、基于当前活跃集合按照一定过滤策略生成新的活跃集合、定义对当前活跃集合中元素(顶点或边)的操作并并行执行.

5)矩阵向量乘编程模型

基于稀疏矩阵向量乘(sparse matrix-vector multiplication, SpMV)的编程模型将图作为稀疏矩阵进行处理, 可以利用高性能计算领域的稀疏线性技术进行优化. 稀疏线性代数技术在图分析处理领域和高性能计算领域之间建立了连接.PEGASUS[79] 基于映射-规约(map-reduce)策略提出广义迭代矩阵向量乘(generalized iterated matrix-vector multiplication)运算, 来统一表示多种不同的图算法并进行并行化. GraphMat[39] 基于以点为中心的编程模型, 将多种典型图处理算法映射到广义稀疏矩阵向量乘运算. 此外, GraphBLAST[40] 是 GPU 平台上基于矩阵向量乘的高性能图处理框架. 基于 GraphBLAST 框架的 BFS 算法将未访问的顶点抽象为列掩码向量$\boldsymbol{v}$, 将当前层活跃顶点抽象为列向量$\boldsymbol{f}$, 图的原始邻接矩阵为$\boldsymbol{A}$, 则对应 BFS 算法迭代执行一次的运算可抽象为$\neg \boldsymbol{v}.*(\boldsymbol{A}^{\mathrm{T}}\times \boldsymbol{f})$.

6)增量计算模型

动态图数据存在多个时刻的图快照, 若每次都采用全量计算的方式进行处理, 则面临计算开销大、存储效率低等问题. 对此, 越来越多的动态图处理工作[19-21,80,83] 集中于使用增量计算模型来处理动态变化的图数据. 在图处理过程中, 考虑到连续快照间动态变化数据占整个图的比例较少, 增量计算模型基于先前快照的部分结果进行计算, 并使用记忆化的迭代计算状态, 可明显减少冗余计算. 图处理系统中, 冗余计算是指对图中某一顶点的属性值, 需要执行多次相同的计算, 才能使该属性值达到最终结果. 增量计算模型包括: 基于修正的增量计算模型和基于重计算的增量计算模型. 基于修正的增量计算模型通过记录顶点之间的依赖关系来识别更新后受影响的顶点, 并在错误的历史计算结果上进行校正得到正确值. 对应的典型工作为: KickStarter[80], GraphBolt[19], GraPU[85], RisGraph[86] 等. 基于重计算的增量计算模型会识别受更新影响的顶点, 并对这些顶点进行重计算, 而其他不受影响的顶点则直接复用历史值, 不参与计算. 典型的基于重计算的增量计算模型为 Kineograph[87], GraphIn[83], EvoGraph[18], iGraph[88] 和 TEGRA[21]. 此外部分工作[19,89] 在动态处理过程中使用混合的全量与增量计算策略.

3.1.2 总结与讨论

以点为中心的编程模型在 CPU 和 GPU 平台的静态与动态图处理系统中应用最为广泛, 它具有编程思想简单、易于并行等优点. 但由于大多数图数据的顶点度遵循幂律分布且顶点间关系依赖复杂等, 这使得基于点中心编程模型的图处理系统在运行时易存在负载不均衡、访存不规则等问题, 导致性能不理想. 以边为中心的编程模型, 通过在边表上执行图算法的流式迭代计算, 可避免随机读写图数据造成的高开销, 从而实现基于图数据分块的单机大规模图数据高效处理. 尽管如此, 以边为中心的图处理系统易存在计算资源利用率不高、算法收敛慢等问题. 为了加快图算法的收敛, 并减少其执行中的访存与通信开销, 以子图为中心的图处理系统被提出, 并被应用在 CPU 与 GPU 平台. 考虑到图数据分布不规则且顶点间关系依赖复杂, 这使得均衡的划分子图并最小化子图间的依赖难以实现, 从而以子图为中心的图处理系统性能易受初始划分影响. 以数据为中心的编程模型应用较少, 在 GPU、FPGA 和加速器平台存在少量应用. 基于矩阵向量乘的编程模型早期在 CPU 平台使用广泛, 后来扩展到 GPU 平台. 考虑到 GPU 平台张量核(tensor core)的推出, 利用张量核来加速基于矩阵向量乘的图处理系统具有一定意义. 由于现实中的图数据往往呈现动态变化, 这使得基于增量计算模型的动态图处理系统在 CPU 平台得到广泛研究. 但由于增量计算会增加存储开销, 且实现逻辑较复杂, 故 GPU 平台上相关工作较少.

3.2 典型编程模型

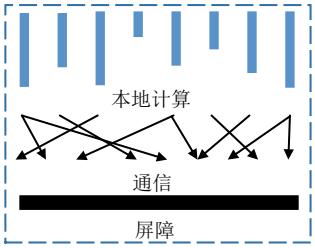
3.2.1 2 种典型编程模型

1)BSP 编程模型

BSP 模型是典型的以点为中心的编程模型[1], 其对应的处理过程和使用举例如图 3(a)和图 3(b)所示. 该模型的一次计算过程包含一系列的全局超步(super-step), 每个超步由本地并发计算、全局通信和同步 3 个阶段组成. 该编程模型思想较为简单, 通过将计算划分为多个超步, 能够有效避免死锁, 同时将计算与通信分开处理, 可简化通信模式等. BSP 编程模型思想广泛应用在静态图处理和动态图处理等领域[2,5,19,90]. 但 BSP 编程模型的缺点是同步开销大, 加之图的幂律特性导致其在处理过程中极易出现负载不均衡, 从而影响图处理系统的整体性能.

2)GAS 编程模型

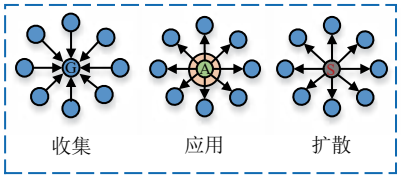
GAS 模型是一种异步编程模型, 它消除了显式同步, 每个节点依据已有资源的情况调度自己负责



(a) BSP编程模型

以SSSP算法为例，使用BSP编程模型：
1）选取初始顶点，初始化所有顶点的距离为INF.
2）超步0：初始顶点的距离更新为0，并设置为活跃顶点.
3）循环执行接下来的超步1…n：
P：每个处理器基于本地内存中的活跃顶点并行执行距离计算；
B：机器节点之间通信顶点最新距离消息；
S：同步等待直到所有节点没有计算和通信；
终止条件为一次超步执行后不存在距离更新的顶点.

(b) 基于BSP编程模型的SSSP算法



(c) GAS编程模型

以SSSP算法为例，使用GAS编程模型：
Gather(D_u, $w(u, v)$, D_v): return D_v+$w(u,v)$; /* 收集所有邻居信息*/
sum(a, b): return min(a, b); /* 基于sum函数计算最短距离*/
Apply(D_u, *new_dist*): D_u = *new_dist*; /* 应用最新最短距离*/
Scatter(D_u, $w(u, v)$, D_v): /* 扩算最新距离给所有邻居顶点*/
if (*changed*(D_u)) *Activate*(v);
if (*increased*(D_u)) return *NULL*; else return $D_u + w(u, v)$.

(d) 基于GAS编程模型的SSSP算法

Fig. 3 The typical programming models

图 3 典型编程模型

的任务，执行性能更好[72]，对应的处理过程和举例如图 3(c)和图 3(d)所示. GAS 编程模型通常是以边为中心的编程模型，执行顶点切割，将度数大的顶点分割成多个镜像以分配到不同的计算节点上. 算法执行过程由收集、应用和扩散 3 个阶段组成. 在收集阶段，图顶点的所有副本同时聚合相邻顶点的数据并将聚合结果发送给主副本；在扩散阶段，每个副本将更新后的值按需发送给相邻顶点. 该模型通过将度数极大的点的计算并行化，从而改善了系统的负载均衡. GraphReduce[91]使用了基于点中心和边中心的混合 GAS 编程模型. 此外，在动态图处理领域，GraphIn[83]基于 GAS 编程模型，设计并实现了增量版的 GAS 编程模型（incremental gather-apply-scatter, I-GAS）. 在图神经网络（graph neural network, GNN）领域，SAGA-NN[92]，NGra[93]等工作将 GAS 编程模型思想扩展到 GNN 系统中.

3.2.2 总结与讨论

图处理系统中，BSP 和 GAS 编程模型作为经典的图处理编程模型，在图计算发展初期被提出，其编程模型与思想被广泛应用在 CPU 与 GPU 平台的静态图处理与动态图处理系统中. BSP 编程模型思想简单，但处理过程中存在全局同步、同步屏障开销大，且算法并行执行容易出现“木桶效应”. GAS 编程模型可实现细粒度的并发，并通过缓存中间结果来减少计算量和网络通信开销，该编程模型对遵循幂律分布特性的图的并行处理非常有利. BSP 和 GAS 编程模型各有利弊，在使用时，需综合考虑图数据的分布属性、平台的计算与存储资源和图处理任务的特性等进行合理选择.

4 图处理系统与分类

自图计算兴起至今，学术界已在该领域产出了多篇图处理系统相关工作. 这些系统的差别体现在图数据是否动态变化、图数据规模与使用的硬件资源、所依赖的硬件平台等. 基于以上，本文梳理的图处理系统的划分具体如图 4 所示.

为了更好地区分图处理系统的差异，依据处理的图数据是否动态变化，可将图处理系统分为静态图处理和动态图处理系统. 静态图处理，其在处理过程中，图数据中顶点与边信息不会发生变化，其处理过程中需重点解决访存不规则、负载不均衡、通信计算比高等挑战. 典型的静态图处理系统包括：Ligra[2]，Galios[3]，GridGraph[6]，Gemini[7]，Tigr[11]，Gunrock[9-10]，Subway[13]，GUM[17]等. 动态图处理，其图数据在处理过程中会随着增、删的顶点与边信息而发生改变，故高效地存储与更新图数据显得至关重要. 同时，动态图处理过程中，由于相邻时间间隔内变化的图数据通常在图数据中占比较小，若采用全量计算会存在大量的冗余访存与计算，采用高效的增量计算可显著提高图处理与查询性能. 典型的动态图处理系统相关工作包括：KickStarter[80]，GraphBolt[19]，GPMA[46]，DZiG[94]，InGress[95]，Layph[96]，ACGraph[97]，CommonGraph[70]等. 早期，图处理系统主要围绕静态图展开，但随着互联网+的快速发展，越来越多的应用背后抽象的关联数据呈现实时动态变化，故近几年动态图

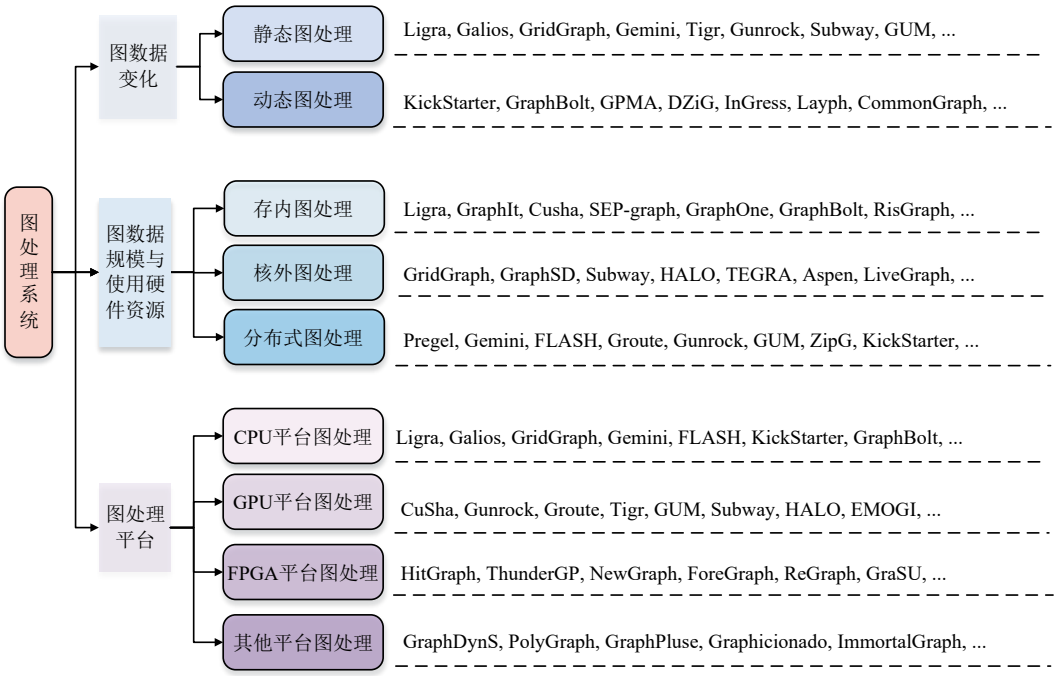


Fig. 4 Classification of graph processing system

图 4 图处理系统分类

处理系统得到快速发展.

依据图数据处理的规模和硬件存储资源大小，可将图处理系统分为存内图处理系统、核外图处理系统和分布式图处理系统. 存内图处理系统，其往往处理数据规模小的图，处理过程中的图数据和算法执行的属性及状态信息等可完全存储在平台的内存中，对应相关优化主要解决访存不规则、负载不均衡等难题. 当前，典型存内图处理系统包括：Ligra[2]，GraphIt[4]，CuSha[73]，SEP-graph[12]，GraphOne[20]，GraphBolt[19]，RisGraph[86] 等. 核外图处理系统，往往处理中等规模的图数据，该图数据在处理过程中不能完全加载到硬件平台内存中，需要借助二级存储设备(多核 CPU 平台下的固态硬盘和磁盘、GPU 平台下的 CPU 内存等)存储部分图数据. 由于使用不同级别的存储设备，不同的访存延迟和带宽是核外图处理系统在设计与优化中考虑的重点. 目前已有的典型核外图处理系统包括：GridGraph[6]，GraphSD[98]，Subway[13]，HALO[99]，TEGRA[21]，Aspen[100]，LiveGraph[101]，ACGraph[97] 等. 分布式图处理通常处理大规模图数据，其利用多节点的分布式存储与计算资源来处理图数据，处理过程中首先会将图数据按照计算节点数划分为不同的子图，然后依次加载子图进行并行计算，计算过程中数据同步与通信是制约该系统性能和扩展性的关键因素. 典型的分布式图处理系统包括：Gemini[7]，FLASH[8]，Groute[15-16]，Gunrock[9-10]，GUM[17]，ZipG[102]，KickStarter[80] 等. 大数据的快速发展使得表达关联关系的图数据规模呈现爆炸式增长，进而带动了核外图处理系统和分布式图处理系统的相关研究与发展.

此外，根据图处理系统所依赖的硬件平台的不同，可将图处理系统分为基于多核 CPU 平台的图处理、基于 GPU 平台的图处理、基于 FPGA 平台的图处理和基于其他硬件平台(如 AISC、PIM、CGRA 等)的图处理. 多核 CPU 平台是最常用的图处理平台，它具有内存和缓存容量大的特性，适用于处理复杂逻辑任务，但计算资源有限等特性. 基于多核 CPU 平台的图处理系统包括：Ligra[2]，Galios[3]，GridGraph[6]，Gemini[7]，FLASH[8]，KickStarter[80]，GraphBolt[19] 等.GPU 平台是加速高性能和并行计算的主流加速平台，该平台具有计算资源丰富、访存带宽高等优势. 但 GPU 平台的全局内容容量有限，使得其难于处理数据规模很大的图数据. 基于 GPU 平台的图处理系统包括：Cusha[73]，Gunrock[9-10]，Groute[15-16]，Tigr[11]，GUM[17]，Subway[13]，HALO[99]，EMOGI[103] 等. 此外，FPGA 与 ASIC 等平台也具有各自的优劣，在此不做过多介绍，只列举其对应典型工作. 基于 FPGA 平台图处理系统包括：HitGraph[104]，ThunderGP[105]，NewGraph[106]，ForeGraph[107]，ReGraph[108]，GraSU[109] 等. 此外还存在一些基于其它硬件平台的图处理系统，如 Graphicionado[110]，GraphDynS[111]，PolyGraph[112]，GraphABCD[113]，Graph-

Pulse[114], JetStream[115], DepGraph[116] 等. 由于 GPU 平台拥有大量的计算资源和高存储带宽，加之 NVIDA 对外提供通用并行计算架构 CUDA，使得其成为高性能计算领域的主流加速器，并在加速图处理领域备受青睐.

考虑到本文主要探讨 CPU 与 GPU 平台上的图处理系统，接下来，将先围绕静态图，对 CPU 与 GPU 平台的存内图处理、核外图处理和分布式图处理系统进行具体阐述. 然后对动态图数据，从图数据的存储设计和动态图数据计算 2 个方面进行总结.

4.1 静态图处理

静态图处理系统自 2010 年起，得到了广泛的研究，并在学术和工业领域取得了一定突破. 为了更好地展示现有图处理相关工作，表 1 汇总了多核 CPU 与 GPU 平台上典型的图处理系统所使用的平台、数据表示格式、编程模型、主要优化技术与实现算法等. 本节将对这些典型的静态图处理系统进行系统性阐述.

1)存内图处理

存内图处理系统对应处理的图数据规模偏小，通常仅需使用硬件平台的内存就能完成图处理. 早期的图处理系统大多集中于存内图处理系统的研究.CPU 平台上对应典型工作包括 Ligra[2], Galois[3], GraphMat[39], Polymer[117], GraphIt[4] 等，GPU 平台上对应的工作主要为 CuSha[73], Gunrock[9-10], SEP-graph[12], SIMD-X[42], GSWITCH[126], G2[137] 等.

Ligra 是麻省理工学院相关团队实现的基于多核 CPU 平台的存内图处理系统，其采用数据驱动的计算模式. 在迭代处理过程中、计算稀疏情况下，以推(Push)方式执行，即从当前层活跃顶点出发，寻找其对应出边，并根据出边信息更新对应的目的顶点属性；而在计算稠密情况下，采用拉(Pull)方式执行，对于每个顶点，寻找其所有入边，并依据入边的源顶点是否是当前层活跃顶点来决定是否更新当前访问顶点的属性信息. Galois 是异步的存内图处理系统. GraphMat 是基于 SpMV 来执行图处理的框架. Polymer 是基于非统一内存访问(non uniform memory access, NUMA)架构的存内图计算系统，它改善了数据访问的亲和性，从而提高了 NUMA 架构下图计算系统的性能及可扩展性. GraphIt 是基于多核 CPU 平台的图计算领域编程框架，其后端集成多种图处理优化策略，并对外提供简单易用的编程接口，从而简化了图处理系统的编程.

基于多核 CPU 平台的存内图计算系统，其对应的主要优化集中在提升访存效率和充分利用多核计算资源等. 充分利用 CPU 平台缓存资源、提高算法执行中的负载均衡、减少原子操作和冗余计算等都多核 CPU 平台存内图处理常用的优化策略.

CuSha 是基于 GPU 平台的图处理系统，它针对将 CSR 存储格式应用于 GPU 平台面临的访存分歧、GPU 利用率低等问题，提出 2 种新颖的图表示形式：面向 GPU 的分片(G-shard)和级联窗口(concatenated windows)，从而实现合并访存，提高了系统 GPU 资源的利用率. Gunrock 针对 GPU 平台上不规则访存与计算应用编程复杂、性能难以保障的问题，提出了以数据为中心的新型高层抽象，将图计算过程定义为推进、过滤、计算三阶段原语. 该三阶段原语可自然地映射到 GPU 平台，并与 GPU 平台的特定优化策略相结合，使得图处理系统编程更灵活、性能更好. 目前，Gunrock 在提高线程间负载均衡方面，使用了细粒度线程调度策略，并基于提出的编程原语进一步与 Pusht 和 Pull 混合执行模式相结合. SIMD-X 提出了运行时的任务管理和调度，从而实现了高性能的图处理. SEP-graph 针对不同输入数据与算法，在图处理的同步与异步、Push 与 Pull 执行方式和算法的数据驱动与拓扑驱动特性之间选择最优图处理策略. GSWITCH 利用预先训练的机器学习模型来自动选择最优的执行策略以提高图处理的性能. G2 在基于 CPU 平台的 GraphIt 后端集成 GPU 平台图处理优化核心技术，从而实现既支持 CPU 又支持 GPU 硬件的高层图处理语言.

不同于 CPU 平台，GPU 平台的单指令多线程(single instruction multiple threads, SIMT)执行模式，使得 GPU 平台存内图处理面临更严峻的访存不规则、负载不均衡挑战. 对此，已有相关工作提出 GPU 友好的图数据表示、合并访存、静态与动态负载均衡等策略来提高 GPU 平台存内图处理性能.

2)核外图处理

随着图数据规模的不断增长，越来越多的图处理工作集中在单机核外图处理系统. 对于基于 CPU 与 GPU 平台的单机核外图处理系统，由于硬件平台片上存储(如 CPU 平台内存或 GPU 平台显存)容量有限，在处理过程中往往利用二级存储设备来存储图的边数据.CPU 平台典型的核外图处理系统包括 GraphChi[5], X-Stream[76], GridGraph[6], PathGraph[138], VENUS[121], NXGraph[139], HUS-Graph[140], MultiLogVC[123], GraphSD[98], CLIP[122], Wonderland[141], LUMOS[120] 等. GPU 平台上典型异构图处理系统包括 Totem[128-129],

Table 1 Typical Static Graph Processing Systems

表 1 典型静态图处理系统

平台及类别		典型系统	数据表示	编程模型	核心优化	实现算法
CPU	存内图处理	Ligra[2]	CSR	点中心同步	混合 Push 和 Pull 模式间动态切换	BFS，BC，Radii，CC，PR，SSSP
		Galois[3]	CSR	点中心异步	拓扑感知任务调度和优先级调度	BFS，CC，PR，SSSP，BC
		GraphMat[39]	DCSC	点中心同步	基于 SpMV 实现图处理	PR，BFS，TC，CF，SSSP
		Polymer[117]	CSR	点中心同步	面向 NUMA，减少远程内存访问	PR，SpMV，BP，BFS，CC，SSSP
		GraphIt[4]	CSR，CSC	点中心同步	图计算领域编程语言，混合优化	BFS，CC，CF，PR-D
		此外，Gagra[118]，Ligra+[119] 等.				
	核外图处理	GraphChi[5]	EL	点中心异步	并行滑动窗口，优化对硬盘的访问	PR，SpMV，CC，TC，CF
		X-Stream[76]	EL	边中心同步	以边为中心计算，以流的方式处理边	CC，SSSP，SpMV，PR，BP，ALS
		GridGraph[6]	EL	边中心同步	两级图划分，双滑动窗口，减少 I/O	BFS，WCC，SpMV，PR
		LUMOS[120]	EL	点中心异步	提出依赖驱动的核外图处理	PR，CoEM，DP，BP，WPR，LP
		GraphSD[98]	CSR	点中心异步	状态和依赖感知的更新策略	PR，PR-D，CC，SSSP
		此外，VENUS[121]，CLIP[122]，MultilogVC[123] 等.				
	分布式图处理	Pregel[1]	CSR	点中心 BSP 同步	高效、易用、可扩展性强	PR，SSSP，CC，LP
		PowerGraph[72]	CSR	点中心 GAS 同步	点中心划分实现负载均衡，减少通信	SSSP，ALS，PR
		Gemini[7]	CSR+CSC	点中心同步	高效内存管理，通信技术及计算模型	PR，CC，SSSP，BFS，BC
		FLASH[8]	CSR	点中心同步	简单、高效地分布式图分析编程模型	BFS，KC，CC，BC，MM，TC，等（70 多种算法）
		此外，PowerLyra[124] 等.				
GPU	存内图处理	Medusa[125]	CSR	点中心 同步	使用 CSR 表示和消息日志减少读开销	BFS，SSSP
		Cusha[73]	CSR	点中心 ICU 同步	提供 G-shard 和 CW 机制，合并访存	BFS，SSSP，CC，SSWP，NN，…
		Gunrock[9-10]	CSR+SOA	数据中心同步	AFC 编程模型和高性能原语，混合调度	BFS，SSSP，BC，PR，CC
		SEP-graph[12]	CSR+CSC	点中心混合	Push/Pull, Sysn/Async, TD/DD 混合处理	PR，SSSP，BFS，BC
		SIMD-X[42]	CSR	点中心 ACC 同步	运行时任务管理	BFS，PR，SSSP，k-Core
		GSWITCH[126]	CSR+CSC	点中心同步	机器学习模型自动调优组合图模式	BFS，CC，PR，SSSP，BC
		G2[127]	CSR+CSC	点中心同步	图计算领域原语，混合多种优化	PR，CC，BC，SSSP，BFS
		此外，IrGL [127] 等.				
	核外图处理	Totem[128-129]	CSR	点中心同步	低度数据 GPU 处理高度数据 CPU 处理	BFS，PR，BC，SSSP
		Graphie[130]	EL	边中心异步	异步边流输传输较少开销，重命名技术	BFS，SSSP，CC
		Subway[13]	CSR	点中心异步	子图生成，减少数据传输（CPU-GPU）	SSSP，SSWP，BFS，CC，PR，…
		EMOGI[103]	CSR	点中心同步	基于零拷贝，优化图算法提高数据传输	SSSP，BFS，CC，PR
		HyTGraph[14]	CSR	点中心异步	混合数据转换管理，代价感知异步调度	PR，SSSP，CC，BFS
		此外，GTS[131]，GraphReduce[91]，Garaph[132]，HALO[99]，Grus[133] 等.				
	分布式图处理	LUX[134]	CSR	点中心同步	数据重划分以实现 GPU 间负载均衡	PR，CC，SSSP，BC，CF
		Gluon[135]	CSR	点中心同步	混合划分优化通信	BFS，CC，PR，SSSP
		Groute[15-16]	CSR	点中心异步	借鉴路由思想通信，实现异步图处理	BFS，SSSP，PR，CC
		GUM[17]	CSR	点中心同步	负载迁移的高效多 GPU 图分析系统	BFS，SSSP，PR，WCC
		此外，MultiGraph[41]，SWARMGRAPH[136] 等.				

GraphReduce[91]，GTS[131]，Graphie[130]，Garaph[132]，Subway[13]，Scaph[142]，Ascetic[143]，HALO[99]，Grus[133]，EMOGI[103] 和 HyTGraph[14] 等图处理系统.

GraphChi 是点中心的核外图处理系统，使用了区间分片结构存储图数据，图中顶点被划分为不相交的区间和一个碎片结构，并提出使用并行滑动窗口策略来减少随机的读取与写入数据到磁盘. X-Stream 是基于边中心的核外图处理系统，它以流的方式来处理边，来减少随机访存. GridGraph 使用了两层划分机制和双滑动窗口模型来避免中间数据写入

磁盘，并使用选择调度策略来避免非必要块数据的读取，提高了核外图处理性能. PathGraph 基于路径中心的编程思想，使用一系列基于树的方式存储图数据，同时并行迭代处理划分的路径以充分利用访存的局部性. VENUS 提出了一种新颖的计算架构——基于点中心的“流式”处理来实现图数据的顺序加载和更新功能的并行执行. 同时，为减少随机读写磁盘数据，VENUS 充分利用系统的内存来缓存大量顶点数据. NXGraph 针对不同的内存预算采用 3 个自适应的更新策略，来确保图数据访存的局部性. 文献 [144] 在每次迭代的每个区间中通过创建动态的分片（当前迭代层所用的活跃边）来减少无用边的磁盘读写. LUMOS 提出了依赖驱动的核外图处理技术，通过正确识别未来依赖项实现数据的跨迭代层传播来减少磁盘数据的读写，从而提高整体处理速度. HUS-Graph 提出了混合的更新策略，针对活跃顶点的数据可自适应地选择数据读写和计算模型. MultiLogVC 通过使用 CSR 表示和消息日志来减少读取非活跃的顶点和边带来的开销. GraphSD 通过在计算过程中持续收集图数据的状态和依赖信息来优化磁盘数据读写效率.

基于多核 CPU 平台的核外图处理系统，其主要的优化集中在减少跨层级存储设备间数据读写的开销、提高数据并行处理的负载均衡等方面. 随着存储设备的不断更新，跨存储层数据的访问延迟和带宽进一步得到改善，这将会提高单机多核 CPU 平台图处理的性能.

GPU 平台上的核外图处理，往往利用 CPU 内存存储图数据的边信息，而图数据的顶点及其属性值或状态信息存储在 GPU 平台. 在图处理过程中，由于部分图数据未能存储在 GPU 平台上，CPU 与 GPU 间数据传输是核心优化之一. 现有 GPU 平台核外图处理系统中，GraphReduce, Graphie, GTS 基于 cudaMemcpy 函数来实现 CPU-GPU 的数据传输，在处理过程中，它们先将图数据划分成多个子图，每次传输具有活跃顶点的子图（活跃子图）到 GPU 平台进行处理. 该方法的优势为 CPU 处理开销小、CPU-GPU 传输过程中高速串行计算扩展总线（pcie）传输的带宽利用率高，但由于活跃子图中存在非活跃的顶点与边信息，从而存在冗余的数据传输. 该方法适用于子图存在大量活跃边的情况. 为了减少无效子图数据的传输，Subway, Scaph, Ascetic 采用不同策略来减少 CPU 与 GPU 间传输数据量，以提高图处理性能. 其中，Subway 每次迭代根据当前层的活跃顶点提取其邻边，重新组成新的子图数据并进行传输，大量减少了数据传输量. 但该方案中，CPU 平台数据重组压缩开销较大，适用于子图活跃边比例小和平均度较低的图数据. HALO, Grus 使用统一内存（unified memory, UM）的方式进行数据传输，大大简化了编程难度. 但该方案需要每次触发页错误，以页面粒度 4 KB 进行数据传输，存在冗余的数据传输和传输开销大问题. 基于 UM 的数据传输更适用于能在 GPU 内存中放得下的图数据. 此外，EMOGI 深入分析了基于零拷贝方式的隐式 CPU 与 GPU 传输方式，该方法基于缓存行大小粒度 32~128 B 执行数据传输，提出数据合并对齐处理的方案来提高传输效率，以解决带宽利用率不稳定问题，从而提高核外图处理性能. 该方法适用于活跃边占比小，同时平均度较高的子图. HyTGraph 综合上面几种数据传输方式，提出了混合的 CPU 与 GPU 数据管理方式来实现高效的数据传输，从而提高 GPU 平台核外图处理性能. 此外，一些工作[75,128-129,132]从计算加速出发，利用 CPU 与 GPU 计算资源来加速核外图处理.

当前，GPU 平台核外图处理系统的性能瓶颈主要集中在 CPU 与 GPU 间的高数据传输开销. 但随着 GPU 存储设备容量的提升和 CPU 与 GPU 间互连技术的快速发展，CPU 与 GPU 间的访存延迟将会降低，同时访存带宽将进一步提升，这有利于提升 GPU 平台核外图处理的性能. 此外，已有相关工作中，多数工作忽略了 CPU 计算资源的使用，因此，可以设计相关策略，卸载部分图计算任务到 CPU 平台，以充分利用 CPU 与 GPU 的计算与存储资源，进一步加速 GPU 平台核外图处理.

3）分布式图处理

对于数据规模非常大的图数据，单机核外图处理通常不能满足实时要求. 对此，分布式图处理系统自 2010 年起得到广泛关注，并衍生出基于多核 CPU 与 GPU 平台的分布式图处理相关优化工作.CPU 平台典型的分布式图处理系统有 Pergel[1], PowerGraph[72], Gemini[7], PowerLya[124], FLASH[8] 等.GPU 平台上也设计并实现了 Medusa[125], LUX[134], Gluon[135], Groute[15-16], GUM[17] 等分布式图处理系统.

Pergel 是谷歌（Google）公司首先提出的分布式图处理系统，其采用 BSP 计算模型，该编程模型思想简单，但存在不同节点间消息同步开销大、负载不均衡的问题. 针对 BSP 编程模型存在的问题，PowerGraph 提出了异步的 GAS 的编程模型，但该图处理框架的划分方法带来大量的冗余副本，增加了内存开销.

Gemini 是国内清华大学研发的高性能分布式图处理系统，它采用 Push 和 Pull 的混合计算模式，同时采用细粒度负载均衡及数据压缩等优化方法，腾讯公司的 Plato 系统就是基于此系统研发的. 此外，阿里巴巴公司的 TuGraph Analytics 系统在设计上也参考了 Gemini[145]. FLASH 是国内 GraphScope 团队研发的分布式图分析算法编程框架，该框架是一个具有强大表达能力的高效图处理系统. FLASH 设计了全新的点中心编程模型，实现了分布式图分析表达中所需的“灵活控制流”“对点集的操作”“跨邻域通信”3 个关键要求，使得图处理框架所支持的图算法达到 70 多种，同时大大减少了算法实现所需的代码量.

基于多核 CPU 平台的核外图处理系统，其主要优化围绕图数据划分、负载均衡、数据通信等展开. 图数据规模的增大使得顶点间关联关系更趋于复杂，这增加了均等划分及最小化划分后通信的难度. 同时，分布式图数据处理中存在大量的数据通信，加之网络通信带宽明显低于 CPU 内存访问带宽，这使得大规模图数据集下分布式图处理系统在性能和可扩展性方面还有待进一步优化.

Medusa 是基于 GPU 平台的分布式图处理系统，对外提供了应用程序编程接口，方便用户在 GPU 上进行图计算编程. 为了减少 CPU 与 GPU 间的数据通信开销，Medusa 在图划分与存储上采用多副本机制，但这增加了数据存储空间的使用. 此外，Medusa 没有尝试解决 GPU 上图计算任务并行度低的问题. LUX 提出了以 GPU 处理为核心的分布式图计算系统，采用了同步的点中心编程模型，并使用动态的图数据重划分策略来达到 GPU 间数据任务的负载均衡. Groute 提出基于多 GPU 的异步图计算算法，该算法借鉴路由网络思想，将细粒度计算任务抽象为消息包，并在 CPU 与 GPU 组成的互连网络结构中流动，且仅当任务流动到对应的 GPU 时才会被其处理. 相比于 Gunrock，Groute 不仅具有更高的性能，同时系统的可扩展性也具有一定的提升. 2023 年，阿里巴巴公司 GraphScope 团队设计并实现了基于多 GPU 的高性能图处理框架 GUM，该框架针对多 GPU 下图处理面临的资源利用率低、数据通信开销高、算法执行收敛慢等问题，利用负载迁移(work stealing)技术来解决图计算任务在多 GPU 环境下的动态负载不均衡和算法执行存在长尾效应问题，从而进一步提升了多 GPU 下图处理的性能.

与多核 CPU 平台相同，基于 GPU 平台的分布式图处理也面临着数据划分难、数据通信开销大、节点间负载难以平衡等问题. 不同的是，GPU 平台分布式图处理还需考虑 GPU 的存储容量有限、计算核数目多和 SIMT 的调度执行模式等，以充分利用 GPU 的高并行和高存储带宽等特性.

4.2 动态图处理

由于现实中抽象的图数据呈现动态变化，这使得动态图处理应用更加广泛. 与静态图处理系统相比，动态图处理系统在数据存储结构设计和不同时刻图数据的查询、计算及更新上更复杂，需要充分考虑图数据顶点和边的增加及删除等. 本节围绕已有动态图处理框架，分别从动态图数据结构设计和动态图计算加速 2 个方面对现有图处理系统进行系统阐述.

1)动态图数据结构设计

针对动态图数据中顶点和边信息不断变化的特性，动态图数据结构的设计需要充分考虑数据更新、计算及存储开销等特性. 当前，多核 CPU 平台上典型动态图数据结构设计工作包括：STINGER[146]，LLAMA[147]，RisGraph[86]，Kineograph[87]，Aspen[100]，Teseo[148]，GraphOne[20]，SAGA-Bench[149]，VCSR[150]，XPGraph[151]，DGAP[82] 等. GPU 平台上的应典型工作包括：cuSTINGER[43]，GPMA[46]，faimGraph[45]，Hornet[44] 等.

STINGER 为流图设计了高性能的数据结构，其充分考虑了系统性能、可扩展性，及图处理的时间与空间特性，设计使用链表存储动态图中顶点的邻接表，但其存在页表缓存(translation lookaside buffer)缺失率高、随机访问开销高等问题. LLAMA 支持多版本图快照的外存表示和存储共享，并基于静态图处理中典型的 CSR 结构，分别使用顶点表和边表来存储顶点和边数据，可基于旧快照增量地生成新快照. 针对邻接链表的缺点，即删除边的过程需要耗时遍历边数组，RisGraph 为高度数顶点添加了哈希索引，从而可快速确定被删除的边在数组中的位置，来达到快速更新图数据的目的，但索引结构带来了额外的存储开销. Aspen 是卡内基梅隆大学相关团队开发的高吞吐流式图处理系统，该系统利用压缩纯功能搜索树(compressed purely-functional search tree)来存储图中顶点的边信息，以提高图更新的性能. GraphOne 结合了边表和邻接链表存储表示的优点，设计了统一的图数据存储来实现动态图批处理和流式处理的高效查询与分析. 此外，一些研究工作[152-154] 致力于提高动态图数据的存储和查询效率，取得了较好的成果.

基于多核 CPU 平台的动态图数据结构的设计，

需要充分考虑图数据变化情况下的高效存储、更新及查询等. 目前, 已有的动态图数据结构, 围绕 CSR 结构的变体、树结构、链表结构等展开, 取得了较好的性能. 但针对动态图数据的变化, 多版本图数据的高效存储和计算是难点, 具体包括高效的存储并更新多版本数据、维护多版本图计算的正确, 以及加快多版本图数据的计算等.

GPU 平台存在内存资源容量有限及内存管理较 CPU 复杂的问题, 因此动态图数据结构设计显得更加复杂.cuSTINGER 是第一个 GPU 平台上的动态图数据存储结构, 它源于 STINGER 思想, 支持图数据的快速更新, 并考虑了数据的局部性, 采用结构体数组(structure of array, SOA)方式进行存储. faimGraph 针对 GPU 平台难以处理大规模动态图数据问题, 提出了全动态 GPU 框架, 并利用不同的结构存储图数据的边、顶点等信息来提高图数据的高效存储、查询与管理. GPMA[46] 基于现有的数据结构, 通过预留存储空间来快速执行插入更新. 该数据结构设计能够限制图重构的范围, 降低了重构开销, 其在 GPU 平台上能够取得较好性能. Hornet 通过预先分配一系列块数组作为图数据顶点的存储池, 从而减轻了 CPU 与 GPU 间数据传输时 GPU 内存分配开销. 同时, 它设计使用块数组来存储顶点的边, 使用顶点数组来存储顶点的边数及其邻居表指针, 从而提高了动态图数据的存储效率和可扩展性.

考虑到 GPU 平台显存容量有限、内存管理复杂, 且编程难度大等问题, 其动态图数据的高效存储与更新较多核 CPU 平台更难. 目前已有的 GPU 平台图数据结构的设计也主要集中于基于数组、链表、树等结构展开, 相关工作较少. 但考虑到 GPU 平台的高并行和高存储带宽, 在 GPU 平台上设计高效的动态图数据结构和增量图计算引擎, 可进一步提升动态图处理性能.

2)动态图计算加速

当前动态图处理系统, 基于是否支持历史数据的存储与查询, 可分为单快照动态图处理和多快照动态图处理. 单快照动态图处理系统会维持一个不断更新的图快照用于计算, 并支持在线更新; 而多快照图处理系统则存在对应不同时间窗口的多个快照, 部分系统支持在线更新. 动态图处理的加速主要基于增量计算模型的实现. 增量图计算模型充分考虑图数据的更新所带来的变化在已有迭代结果中增量的计算, 可大大减少冗余计算开销, 并得到了学术界的广泛关注. 基于以上分析, 本节将介绍基于单快照的增量图处理系统和基于多快照的增量图处理系统.

基于单快照的增量图处理系统包括: KickStarter[80], GraphBolt[19], DZiG[94], Tripoline[81], ACGraph[97], GraphIn[83], EvoGraph [18] 等. KickStarter 是针对单调类图算法设计的异步增量计算图处理系统, 其通过维护一棵依赖关系树, 对历史计算结果进行修正, 并将修正后的顶点值作为在新图上计算的初始值, 以达到加速收敛的目的. GraphBolt 是基于 BSP 编程模型提出的增量流式图处理系统, 其存储每轮迭代的历史计算结果, 根据更新信息和顶点之间的依赖关系识别受更新影响的顶点, 并对受影响顶点的历史聚合值进行修正, 而其他不受更新影响的顶点则直接复用历史计算结果. DZiG 针对 GraphBolt 工作对稀疏计算模式不友好问题做了稀疏感知的进一步优化. Tripoline 通过挖掘三角形不等关系来减少动态图处理过程中的冗余操作, 以加快处理速度. ACGraph 通过维护依赖层树来调整受影响顶点的执行顺序, 以合并处理顶点的多次更新, 并减少冗余计算, 从而提高动态图处理性能. GraphIn 和 EvoGraph 是基于 GPU 平台的高效动态图处理系统, 其针对动态图的更新和计算进行优化, 并实现了 I-GAS 编程模型.

基于多快照的增量图处理系统包括: TEGRA[21], CommonGraph[70] 等. TEGRA 动态图处理系统采用了重计算的增量思想, 与 GraphBolt 相似, 它也需要存储历史迭代计算结果并根据更新信息和依赖关系识别受更新影响的顶点, 但不同的是, 这些受影响的顶点通过重新计算的方式得到正确的顶点值. CommonGraph 针对不同时间窗口的多个快照, 通过提取多个快照的公共子图和差异子图, 并将边删除转换为边增加的处理策略来提高动态图处理的性能.

目前, 基于单快照与多快照的动态图处理难点在于如何充分地利用已有数据信息来高效地进行增量图计算. 增量图处理系统中, 挖掘图数据依赖信息、减少冗余计算和额外存储开销, 对提升动态图处理系统的性能至关重要. 目前多核 CPU 平台下, 基于动态图的增量计算工作已取得一定成果, 而 GPU 平台的相关工作较少, 若能将多核 CPU 平台相关增量计算优化灵活地应用到 GPU 平台, 利用 GPU 的高并行与高存储带宽加速动态图处理, 将会进一步提升动态图处理的性能.

5 图处理优化关键技术

自图计算兴起至今, 基于 CPU 与 GPU 平台的图

处理系统在存储和计算等方面取得了较好的成果. 基于此, 本节将从多核 CPU 和 GPU 平台常用的图处理优化关键技术, 具体包括图数据预处理、数据访存、计算优化和数据通信 4 个方面对现有图处理系统优化的关键技术进行系统介绍.

5.1 数据预处理

图数据预处理旨在图算法执行前, 对图中数据进行去除冗余、压缩及调整数据分布等来加快图算法的执行. 当前, 图处理领域的数据预处理方法包括: 去除冗余数据、设计图数据的表示方式、对数据进行压缩及数据重排等.

5.1.1 数据预处理方法

1)去除冗余数据

现实应用中抽象的图数据, 往往存在一定比例度为零的顶点(孤立顶点)和重复的边, 若不对其进行处理, 会带来额外的存储开销, 同时图算法执行过程中, 孤立顶点的存在会带来无效的数据加载, 重复的边会增加冗余的计算. 因此, 在加载图数据时, 有必要去除图数据中冗余的数据. 文献 [155] 阐明了将生成图中的孤立顶点和重复边去除可带来一定的性能提升.

2)设计图数据的表示方式

对于静态图处理系统, 最常用的图数据表示格式为 CSR, CSC, COO 及 EL 等. 由于图处理是典型的访存不规则应用, 为了减少图算法执行过程中访存分歧导致的性能低问题, 需要针对不同的执行特性, 选择使用不同的存储格式. Ligra[2] 图处理系统使用了 CSR 和 CSC 组合的存储格式. Gemini[7] 则充分考虑分布式图算法的执行模式, 使用位图辅助的 CSR 和双压缩机制的 CSC 存储格式.

3)数据压缩

为了减少图数据的存储与传输的开销, 已有相关工作[119,156-157] 研究使用无损压缩算法来减少数据存储开销. 尽管数据压缩能够减少数据的存储空间, 但是在图处理过程中压缩与解压缩时间开销较大. 文献 [156] 为了能够使 GPU 平台处理超过显存大小的图数据, 设计并实现了 GPU 平台上基于压缩图数据的图遍历, 并取得了较好的图处理性能. 文献 [157] 提出了基于规则的数据压缩, 并实现了基于压缩后的数据进行图分析的方法, 从而消除了解压缩时间开销, 提高了图处理性能.

4)数据重排

数据重排即从提高数据的时间局部性和空间局部性角度出发, 通过对图数据的分布情况进行重映射来改善数据的局部性, 提高数据的复用和高速缓存的命中率, 从而加快算法的执行, 提高系统整体性能. 当前常用的数据重排方法, 包括基于图数据顶点度的重排和基于拓扑的数据重排. 基于图数据顶点度的重排优化已在图处理系统中得到广泛应用[118,158-161], 它们通过对频繁访问的顶点(高度数顶点)进行一定策略的分组重排, 来改善算法执行的时间局部性. 基于拓扑的数据重排依据顶点的拓扑进行重映射, 如依据 BFS 或 DFS 的顺序[138,161] 对连接的顶点进行分组重排, 从而改善算法执行的空间局部性[162].

5.1.2 总结与讨论

图数据的预处理方法中, 去除冗余数据、设计图数据的表示方式和数据重排优化, 通常在图算法执行前对图数据进行一次处理即可, 它通常利用多核 CPU 平台来执行. 依据 Graph500 性能计算标准, 预处理时间不记录在算法执行时间内. 但相关算法优化工作在进行性能评价时, 会将不同预处理的时间开销进行对比, 并将这部分时间进行分摊处理. 而图数据的表示使用适用于并行计算的数据结构来存储图数据, 但与原始的数据结构相比, 会存在一些额外的存储空间占用. 数据压缩技术虽然能减少图处理过程中存储资源的占用, 但压缩与解压缩会带来额外的时间开销.

5.2 访存优化

图算法在执行过程中的依赖复杂且难以预测依赖关系, 使得算法执行过程中存在严重的低效访存. 为提高访存效率, 相关工作提出了合并访存的优化技术. 其主要通过数据重排、数据重组、使用高速缓存等手段来提高数据局部性, 减少访存延迟和访存分歧等.

5.2.1 访存优化方法

1)数据重组

图数据的不规则特性, 使得图数据局部性较差, 从而导致图数据在处理过程中存在较高的缓存缺失率和分支缺失率[73], 进而带来 CPU 平台流水线效率低等问题. 此外, GPU 平台的单指令、多线程的执行模式, 使得基于其的图处理面临着更严重的低效访存. 对此, 图处理系统设计并使用局部性好的数据表示方式、数据重组方式来提高图数据的访存效率. 文献 [11] 通过对边数组进行重排, 实现了基于边数组的合并访存. 文献 [48,90] 设计了面向 GPU 的 CSR 结构, 通过使用位图自适应的 CSR 结构和线程束对齐(warp-aligned)的邻边表, 来提高 BFS 算法执行的访存效率.

2)使用高速缓存

高速缓存具有更低的访存延迟，合理使用缓存可以提高图处理系统的性能[118,162]. 由于系统中高速缓存容量有限，为了充分利用其特性，往往使用缓存块的技术优化相关处理. 文献 [118] 提出使用面向通量的缓存块机制来改善访存效率. 文献 [162] 提出了缓存感知的(cache-aware)的合并访存，通过缓存图处理过程中的状态信息来达到合并访存.

3)减少访存开销

图算法执行是典型的访存密集型任务，故减少访存开销可提升图处理的性能. 为了减少访存开销，文献 [163] 提出计算和磁盘读写的双并行，同时设计计算和磁盘数据读写处理的重叠来加快多核 CPU 平台图处理性能. 文献 [144] 通过使用动态划分和动态数据表示来减少无效边的读取和访问，从而提高图处理性能.

5.2.2 总结与讨论

图处理系统是典型的访存密集且不规则应用，多核 CPU 平台和 GPU 平台都需重点针对图处理系统中存在的访存效率低问题进行相关优化. GPU 平台的数据访存优化，可通过数据重组优化来达到数据的合并访存. 由于多核 CPU 平台存在多级缓存，且缓存容量较 GPU 平台大，故通常使用缓存来提高数据时间和空间局部性、改善数据预取效率、提高缓存命中率、从而提高图处理性能. 在多核 CPU 与 GPU 平台核外图处理系统中都需重点考虑访存优化，通过减少数据读写量实现计算和数据访问的重叠等来加速图处理.

5.3 计算加速

图数据的不规则、依赖复杂等特性给图处理加速在计算层次的设计与优化等方面带来巨大挑战. 从关键技术优化角度看，图处理系统在计算层次的主要优化目标是充分挖掘算法执行模式和系统硬件特性来加快算法收敛、提高计算资源的利用率. 常见的计算优化方法包括：负载均衡、减少冗余计算、优化算法执行模式等，下面将依次进行介绍.

5.3.1 计算加速方法

1)负载均衡

由于图数据的顶点度服从幂律分布，即每个顶点的邻居顶点个数存在显著差异，使得图数据迭代处理过程中，每个顶点的计算任务量差异较大，进而导致严重的负载不均衡. Enterprise[49] 基于 GPU 平台不同粒度的并行机制和顶点度的分布情况，提出了基于线程、线程束、线程块、网格等不同粒度的负载均衡策略来提高 GPU 资源的利用率. 该负载均衡调度思想被用在 Gunrock[9-10]、SIMD-X[42] 等图处理系统中. 此外，针对图数据分布的不规则带来的负载不均衡问题，Tigr[11] 提出虚图概念，对原始不规则的图数据使用虚图转换策略，使得转换后的顶点度数控制在一定的范围内，来保证每个顶点的负载近似相等，进而直接使用顶点粒度的负载均衡.

负载均衡策略分为基于点的负载均衡和基于边的负载均衡. 基于点的负载均衡策略应用在点中心的编程模型，而基于边的负载均衡在边中心的编程模型中应用较多. 在基于点的负载均衡策略中，由于图中顶点的度分布不均衡，必要时可再进一步基于边进行负载划分，以达到负载均衡. 此外，近年来，细粒度的工作窃取和核函数融合技术也应用于典型图处理系统，如 Gemini[7], X-Stream[76], SIMD-X[42], GraphIt[4] 等. 工作窃取技术是指让已完成任务的线程“窃取”其它线程未完成的任务来执行，以充分提高系统的资源利用率. 为了减少频繁开启线程带来的开销，核函数融合技术通过将多个执行函数组合成一个，可减少系统中线程创建和销毁的开销.

2)减少冗余计算

图算法执行过程中对图中某一顶点的计算，在并行处理中会重复出现多次，且随着图数据规模的增大，这种冗余计算带来的额外开销对图处理性能的影响会明显增加. 为减少图处理系统中的冗余计算，Gunrock[9] 在“过滤”阶段会对去除活跃队列中相同和非必要的活跃顶点. 文献 [56] 为了减少 SSSP 算法执行过程中的冗余计算，提出属性感知的数据重排优化，从而提高了算法执行的效率. 此外，动态图处理系统中，在图数据规模变化相对小的情况下，若采用全量计算会带来大量的冗余顶点的计算. 对此，GraphBolt[19], TEGRA[21] 等动态图处理系统采用增量计算模型. 但基于增量计算的图处理系统，处理过程中不恰当的顶点执行顺序会导致冗余计算. 对此，ACGraph[97] 提出了高效的运行时方法来进一步减少冗余计算，提高算法并行执行效率.

3)优化算法执行模式

由于图数据往往遵循幂律分布，使得图算法迭代执行过程中激活的顶点的数目难以预测. 因此针对活跃顶点的稀疏和稠密情况，存在 Push 和 Pull 这 2 种执行方式. 对于图数据处理过程中的更新是否立即可见，又分为同步执行和异步执行. 对于算法本身特性，又可以分为拓扑驱动的算法和数据驱动的算法. 对于动态图处理，基于增量的计算与全量计算的

执行模式也是考虑的重点. 在算法执行过程中, 为了使算法性能更好, 文献 [12] 针对不同的执行模式, 提出混合图处理框架. 文献 [137] 也充分分析了算法运行时的特性, 针对不同执行模式提出了相关的优化, 从而加快了算法执行的特性.

5.3.2 总结与讨论

由于图数据的度通常遵循幂律特性, 且图算法执行具有依赖复杂等特性, 使得图处理并行计算在多核 CPU 与 GPU 平台, 易出现负载不均衡、冗余计算开销大、算法收敛慢等问题. 对此, 多核 CPU 与 GPU 平台都需优化负载均衡策略、提升图算法执行的效率、选择最优执行模式等来加速图处理. 其中, 在负载均衡优化方面, 由于多核 CPU 与 GPU 平台计算资源存在差异, 所以这 2 个平台负载均衡策略在细节上存在不同. 算法执行模式的选择方面, 也需要针对于这 2 个平台各自的特性, 进行特定的优化设计, 以保证在各平台上优化后取得较好性能.

5.4 通信优化

对于存内图处理系统, 数据间通信主要来自线程间的通信, 由于其可依靠全局信息来实现共享, 故对其优化较少. 但单机存内图处理系统中, 对于 NUMA 架构, 存在相关研究工作[7,17,124] 致力于其在数据划分方面进行优化, 来尽可能做到减少跨 CPU 间的数据访存.

核外图处理系统中, 对应的数据通信优化技术主要解决不同级别存储设备间数据的传输. 而对于分布式图处理系统, 数据通信主要来自不同计算节点间的通信. 对于 CPU 平台来说, 系统访问内存的速度与系统访问二级存储设备(如固态硬盘、磁盘等)的速度存在明显差异. 对于 GPU 平台, GPU 与 CPU 间的数据传输往往需要利用受限的 PCIe 带宽资源. 为了减少通信开销, 核外图处理系统往往在数据划分、减少冗余数据传输、提高通信速度等角度进行优化.

5.4.1 通信优化方法

1)数据划分

数据划分技术将数据规模大的图数据划分为不同的子图, 然后基于不同的子图进行并行处理. 该技术是大规模图处理系统优化的重点之一, 其对图处理任务负载的划分和数据通信有重要影响. 良好的数据划分策略有助于实现负载均衡和减少通信数据量. 当前数据划分主要包括基于点的划分、基于边的划分以及点和边的混合的划分等.

文献 [6] 采用两级的层划分来减少核外图处理的磁盘访问并最大化利用 CPU 的缓存, 以提高核外图处理的性能. 文献 [164] 针对分布式图处理中的数据划分问题, 提出了两阶段的数据划分机制来实现图数据在顶点和边 2 个维度的数据均衡, 同时减少边切割以最小化通信开销. 文献 [165] 针对优先级迭代图计算模式算法, 提出了基于热点数据的均衡划分策略来提高图算法处理的性能.

2)减少冗余数据传输

在分布式和核外图处理系统中, 往往存在较大的数据通信开销. 为了减少数据通信开销, 相关研究从减少通信数据量角度出发, 分析算法执行过程中的数据重用特性来减少冗余数据的开销. 典型的如文献 [142-143] 等, 通过进行优化设计来提高算法执行过程中数据的复用, 从而减少算法执行过程中 CPU 与 GPU 间数据的传输. 此外, 为了减少数据传输, 一些相关研究[7] 在数据划分后存储顶点的副本信息来减少数据传输量, 但这在一定程度上会增加存储空间的占用.

3)加速通信速度

随着互联网的快速发展, 新型硬件设备也在快速发展并持续更新中. 如在计算资源方面, CPU 与 GPU 趋向于更多核、更强算力发展, 而在硬件互连方面, 统一虚拟内存(unified virtual memory)技术[166]、GPU 与 CPU 高速互连(NvLink)技术[167]、PCIe 技术[168] 也在持续更新. 硬件的更新在一定程度上加快了数据通信的速度. 目前, 部分工作[13,17,103,133] 集中于利用先进的互连硬件来加速特定的图处理, 并取得了较好的性能. 此外, 近年来随着远程直接内存访问(remote direct memory access)技术的快速发展, 一些工作[169-172] 致力于利用该技术来减少图处理系统中数据通信的延迟开销, 并通过负载迁移策略来改善系统处理过程中的负载不均, 从而提高处理性能.

5.4.2 总结与讨论

在多核 CPU 与 GPU 平台上, 图算法执行过程中都会产生大量的消息, 这明显增加了算法的通信开销. 合理的数据划分可减少划分后不同计算节点间的通信开销. 此外, 对于核外图处理系统, 减少跨不同存储层的数据传输量、提升设备互连技术、使用具有低延迟和高带宽的通信链路, 也可明显减少多核 CPU 与 GPU 平台的数据通信开销, 从而提高图处理性能.

6 图处理系统分析

6.1 性能分析

自 2010 年谷歌公司提出分布式图处理框架

Pregel[1] 以来，随着计算机硬件设备存储和计算资源的提升，对于不同规模的图数据，图处理系统基于不同平台在性能、能效等方面取得了很大提升. 为了更好地评估图处理系统的性能，在此选择 Graph500[47] 标准中使用的性能评价指标：每秒遍历的边数（traversed edges per second, *TEPS*）、每秒遍历的百万边数（million-traversed per second, *MTEPS*）、每秒遍历的十亿边数（giga-traversed per second, *GTEPS*）来衡量图处理的性能，其对应的计算如式（1）~（3）所示. 存在 Green Graph500[47] 能效榜单，其对应的评价指标为图处理性能与功耗的比值. 对应 MTEPS/W 是指每消耗一瓦特电功率在单位时间内（1 s）所能遍历的百万边数.

$$TEPS = \frac{\text{边数}}{\text{图算法(BFS/SSSP)执行时间/s}}, \tag{1}$$

$$MTEPS = \frac{TEPS}{10^6}, \tag{2}$$

$$GTEPS = \frac{TEPS}{10^9}. \tag{3}$$

2023 年 11 月的 Graph500[47] 性能榜单中，图 BFS 算法的性能可达 138 867 GTEPS，其与 2010 年的最高性能 7.086 7 GTEPS 相比，图算法处理能力呈现出多个数量级的提升. 对于 SSSP 算法，其处理性能也从 2017 年的 12.88 GTEPS 提升到 2023 年的 15 335.9 GTEPS. 与此同时，根据 Green Graph500 能效榜单，对于大规模图数据，其能效值由 5.41 MTEPS/W 提升到了 22 094.87 MTEPS/W，对于小规模图数据，则由 64.12 MTEPS/W 提升到了 16 238.05 MTEPS/W. 虽然 Graph500 和 Green Graph500 这 2 个榜单不能完全代表当前图处理系统的性能，但其从一定程度上反映出了图处理系统在性能和能效方面的提升，同时也体现了国内外相关领域图计算系统的发展.

为了更好地展示图处理系统在 CPU 与 GPU 平台上的性能，本文系统地梳理了典型的图处理系统在 BFS 算法、SSSP 算法或其它典型图算法上的性能，对应结果如表 2 所示. 表 2 列举了当前 CPU 与 GPU 平台上，部分存内、核外和分布式图处理代表性工作 Ligra[2]，GraphSD[98]，FLASH[8]，Tigr[11]，HyTGraph[14]，GUM[17] 的执行性能.

Ligra 是多核 CPU 平台上广泛使用、性能较好的存内图处理系统. 其在不同的数据集上的 BFS 和 SSSP 算法的性能都较好. GraphSD 是当前多核 CPU 平台下先进的核外图处理系统，由于其处理的数据规模很大，需要依赖磁盘进行处理，故其在 PageRank 和 SSSP 算法上性能较差. FLASH 是多核 CPU 平台上先进的分布式图处理框架，其集成了多种图算法，与其他图处理系统相比其可扩展性更好. 但 FLASH 图处理框架受限于计算资源和通信的影响，其对应的性能相对较差. Tigr 是 GPU 平台先进的存内图处理系统，由于其 GPU 显存容量有限，所以只能处理 GPU 显存放的下的图数据，但由于 GPU 的并行计算资源丰富，且访存带宽高，故其上的 BFS 和 SSSP 算法性能较好. HyTGraph 是 GPU 平台先进的核外图处理系统，其使用了混合的 CPU 与 GPU 间输出传输策略，从而提高了大规模图处理系统. 它在不同的大规模图数据集上的 BFS 和 SSSP 算法上获得较好性能. GUM 是当前 GPU 平台下先进的分布式图处理系统，

Table 2 Performance of Classic Static Graph Processing Systems

表 2 典型静态图处理系统性能

系统	平台	数据集	BFS 性能/GTEPS	SSSP 性能/GTEPS
Ligra	40 核 Intel CPU E7-8870	Twitter[173]	4.579	0.553
		rMat27[36]	5.012	0.526
GraphSD	2×8 核 Intel CPU E5-2620	Uk-union[174]	0.010（PageRank）	0.001
		rMat30[36]	0.009（PageRank）	0.001
FLASH	20 节点，单节点 30 核 Intel CPU E5-2682 v4	Uk-2002[175]	0.132	0.043（TC）
		soc-orkut[176]	0.334	0.035（TC）
Tigr	8 GB，P4000 GPU	soc-orkut[176]	2.117	0.760
		Pokec[177]	3.010	1.464
HyTGraph	11 GB，GTX 2080Ti GPU	Twitter[174]	2.306	0.938
		UK-2007[174]	3.761	1.191
GUM	32 GB V100×8 NvLink	Web2001[178-179]	25.147	2.587（PageRank）
		road-USA[175]	0.082	0.784（PageRank）

注：TC（triangle counting）.

它在 8 个支持 NvLink 的 V100 GPU 下进行测试, 在网页和道路图上都取得了较好的性能.

Ligra 和 HyTGraph 都对 Twitter 数据集进行了测试, 但这 2 个 Twitter 数据集在图规模上存在一部分差距, 且 Ligra 和 HyTGraph 分别使用多核 CPU 与 GPU 平台对其进行处理, 故性能存在一定的区别. 对于不同的工作, 考虑到硬件平台差异性和优化侧重点的不同, 所以对应性能存在部分差距, 但基于大规模图数据的处理, 其性能均待进一步的提升. 在数据集方面, 对于中小规模的图数据集, GPU 平台图处理具有一定的优势, 对于超大规模图数据, 分布式多核 CPU 平台更有优势. 当前图处理领域, 基于单节点多 GPU 的硬件平台, 其 1 s 可以完成千亿规模的图计算, 在图处理规模、图处理性能等方面较先前工作存在的明显提升.

对于动态图处理系统, 由基于多快照的全图重计算演变到现在的增量图计算, 大大减少了处理过程中的冗余计算, 从而提高了处理的性能. 当前, 业界先进的动态图处理, 以 Layph[96] 为例, 在 52 超线程 CPU 平台上, 其针对动态图数据的处理性能可达 1.87 GTEPS.

6.2 可扩展性分析

当前, 在 CPU 与 GPU 平台下, 基于存内的图处理系统未考虑数据划分、数据通信等问题, 难以扩展到核外和分布式图处理系统, 或扩展后其性能较差. 而对于核外图处理系统, 由于其存在不同级存储带宽差问题, 已有的多数相关工作通常局限于设计并实现单节点二级存储(如基于 CPU 的硬盘或固态硬盘的核外图处理系统、基于单节点单 GPU 的核外图处理系统)的核外图处理. 对于分布式图处理系统, 由于大规模图数据的依赖趋于更复杂, 故难以做到数据划分的均衡和最小化割边, 从而带来分布式图处理系统的负载不均衡和数据通信开销大问题. 对基于 PCIe 互连的 GPU 平台, 核外图处理系统和分布式图处理系统受限于 PCIe 带宽资源, 可扩展性也较差. 而在支持 NVLink 互连的 CPU-GPU 异构平台下, 可扩展性较好. 但这样的平台, 最终还会因为存储空间有限、访存带宽差等, 使得大规模图处理的可扩展性差.

对于图算法而言, 已有的图处理系统往往局限于典型的几种图处理算法, 如 BFS, SSSP, PageRank, CC 等, 同时, 尽管一些图处理系统对外提供了应用编程接口, 但考虑到图处理系统通常设计并实现某些特定的优化, 使得其扩展其它图算法时出现扩展难度大、扩展后性能不理想等问题. 此外, 对于不同类型的数据, 如社交网络图和网页搜索图等, 其呈现高度的幂律特性, 而道路图通常具有较长直径特性, 已有图处理系统对于不同特性的图数据性能差异较大、可扩展性相对较差. 最后, 随着图数据规模的不断增长, 已有图处理系统的可扩展性还有待进一步提升.

此外, 考虑到现实应用中的图数据往往呈现动态变化, 但已有的静态图处理系统通常不能处理动态图数据. 与此同时, 现有的动态图处理系统, 由于过多考虑加速动态图的处理, 忽略了对静态图的处理, 这使得静态图处理与动态图处理存在明显的割裂, 故在处理静态图和动态图方面, 表现出了不理想的可扩展性.

6.3 应用平台分析

图处理系统所依赖的平台包括通用的多核 CPU 平台和 GPU 平台, 以及基于 FPGA 和 ASIC 的专用平台等. 通用多核 CPU 平台具有大容量的内存和多级缓存, 适用于处理复杂逻辑的处理任务, 且编程和调试较其它平台相比较简单, 但其计算资源通常有限. 早期图处理系统相关优化集中在多核 CPU 平台, 并在存内、核外和分布式图处理方面取得较好性能. 通用的 GPU 平台拥有大量的计算资源和高存储带宽, 但其显存大小有限, 通常为 16 GB 或 32 GB. 当前 GPU 平台上小规模图数据和中等规模的图数据处理均可取得较好的性能. 但对于大规模图数据, 考虑到多 GPU 资源的成本开销和跨计算节点 GPU 间通信开销大等, 目前相关研究工作较少.FPGA 平台具有可编程性、高并行性和灵活性等, 使其可以定制化设计高能效图处理系统, 但 FPGA 的编程相对复杂, 需要对硬件有一定的掌握, 且存在可重构的成本开销大的问题. ASIC 平台通过定制化的设计可以实现高能效的图处理系统, 但其灵活性差、设计和研发的成本高.

综合考虑计算资源、存储容量、可编程性等, 对于中小规模的图数据, GPU 的显存能够存放处理过程中的所有数据, 或基于 GPU 平台核外图处理系统, 其与别的平台相比使用 GPU 平台在性能和可编程等方面具有相对明显的优势. 对于超大规模的图数据, 考虑到系统的可维护性以及研发成本等, 可优先考虑使用大规模分布式多核 CPU 平台. 从能效特性考虑, 基于 FPGA 平台可实现高性能且低功耗的图处理系统, 且与 ASIC 平台相比, 其研发成本低, 可以作为首先的平台. 但对于定制化研发高性能、低功耗的图

处理系统的需求，基于 ASIC 平台设计软硬协同的图处理架构，并对其进行模拟、仿真、校验与流片等，具有重大的创新意义. 此外，考虑到存算一体平台可打破存储墙瓶颈，故基于此也可设计并实现高性能、低功耗的图处理系统.

7 总结与展望

图计算系统是时下学术界和工业界最热门的研究方向之一，本文首先对图处理系统编程框架分层结构、图数据表示、典型算法及应用领域等进行介绍，然后总结了当前图处理系统所面临的存储与计算挑战，之后系统梳理了当前图处理系统的典型编程模型，并分类总结了现有的典型图处理系统. 之后从数据预处理、访存优化、计算加速和降低通信开销等方面对图处理系统领域的关键技术进行详实而系统地分析. 最后对业界先进图处理系统的性能、可扩展性等进行总结. 由于图处理系统研究尚处于热门的研究阶段，其仍拥有非常大的研究与优化空间，具体包括 5 个方面：

1)超大规模图数据处理

随着大数据时代的飞速发展，图数据规模呈爆炸式增长，图计算应用对存储和计算资源的需求也愈加提高，单机核外图处理系统难以执行超大规模图数据处理，因此设计扩展性良好的多节点分布式图计算系统至关重要，目前学术界和工业界虽已有一些研究成果，但图处理的访存不规则、算法执行依赖复杂、通信计算开销比大等问题仍然需要解决，使得该领域研究充满挑战. 相信在未来，该方向会成为图处理应用的研究热点之一.

2)动态图数据处理

现实应用背后抽象的图数据呈动态实时变化，动态图数据的高效存储与计算将会是未来图处理加速发展的必然趋势. 而基于动态图数据的大规模图神经网络模型设计与训练近两年也得到了初步的发展，相信不久会引起研究的热潮，并对图计算未来应用的发展起到关键作用. 对于大规模动态图数据，通过设计流水线的计算模式来实现硬件存储与计算资源的充分利用，也是现阶段研究的热点领域之一. 动态图数据的处理在存储结构设计、计算加速、资源调度等方面有着广阔的研究前景.

3)异构硬件加速图处理

随着 GPU, FPGA, ASIC 等新型硬件计算单元，存储单元及互连系统的快速迭代更新，越来越多的研究集中在依靠特定领域加速硬件或加速器来设计并研发软硬协同的图处理系统，以充分发挥硬件的计算与存储性能，提高图处理系统的性能. 例如，当前 GPU 平台图处理系统研究取得较好成果，但随着 GPU 平台专门提供用于乘加运算的 Tensor 核单元，相信以稀疏矩阵向量乘的方式加速不规则图处理相关应用的研究也会是未来研究的热点之一.

4)并发图数据处理

当前，针对图处理系统的研究主要集中在对于单任务执行时的性能优化. 但在图处理的实际应用中，存在着并发执行图处理任务的需求. 并发图任务的实际执行场景中，存在着多个不同的图处理任务依赖相同的图数据的情况，如在公路图数据上，所有的任务依赖同一份地图数据. 对于这种需要处理并发任务的业务场景，如何充分利用图数据的空间和时间局部性，尽可能共享系统的存储与计算资源将会是未来研究的趋势和热点. 同时，如何设计多任务之间混合粒度的调度策略，实现并发图处理任务的高性能和高通量执行也是一个亟待开发的研究方向. 在更为普遍的场景中，图数据并不是一成不变的，因此存在多个图处理任务依赖同一动态变化的图数据应用场景，基于动态图研究并发图处理将会是并发图数据处理的发展趋势.

5)图挖掘系统

数字时代的到来，使得现实应用背后抽象的图数据呈现高度复杂，这亟需利用高层的图挖掘相关算法来进一步分析图数据背后的深层关系，以更好地利用潜在的关联信息，从而更好地指导和决策生活. 尽管如此，图挖掘算法中的模式识别、子图匹配等算法是非确定性多项式(non-deterministic polynomial)完全问题，其在处理过程中往往存在大量的冗余计算，且执行流程难以预测，加之数据规模的增长和动态变化等特性，使得其性能难以满足实际需求. 因此，设计并研发高性能、高能效的图挖掘处理系统是当前图计算领域的热点研究之一. 应实际需求，其未来的发展趋势将是设计并研发基于大规模动态图的流式图挖掘系统.

作者贡献声明：张园拟定论文提纲，并完成论文撰写；曹华伟检查、指导、优化论文相关内容；张婕和申玥参与论文中动态图部分内容的修改和讨论；孙一鸣和敦明参与论文关键技术部分内容的修改；安学军和叶笑春指导论文.

参 考 文 献

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

[2] Shun Julian, Blelloch G E. Ligra: A lightweight graph processing framework for shared memory[C]//Proc of the 18th ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2013: 135–146

[3] Nguyen D, Lenharth A, Pingali K. A lightweight infrastructure for graph analytics[C]//Proc of the 24th ACM Symp on Operating Systems Principles. New York: ACM, 2013: 456–471

[4] Zhang Yunming, Yang Mengjiao, Baghdadi R, et al. GraphIt: A high-performance graph DSL[J]. Proceedings of the ACM on Programming Languages, 2018, 2: 1−30

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

[6] Zhu Xiaowei, Han Wentao, Chen Wenguang. GridGraph: Large-scale graph processing on a single machine using 2-level hierarchical partitioning[C]//Proc of the 2015 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2015: 375–386

[7] Zhu Xiaowei, Chen Wenguang, Zheng Weimin, et al. Gemini: A computation-centric distributed graph processing system[C]//Proc of the 12th USENIX Conf on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2016: 301–316

[8] Li Xue, Meng Ke, Qin Lu, et al. FLASH: A framework for programming distributed graph processing algorithms[C]//Proc of the 39th Int Conf on Data Engineering (ICDE). Piscataway, NJ: IEEE, 2023: 232−244

[9] 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: 1−12

[10] Wang Yangzihao, Pan Yuechao, Davidson A, et al. Gunrock: GPU graph analytics[J]. ACM Transactions on Parallel Computing, 2017, 4(1): 1−49

[11] Sabet A H N, Qiu Junqiao, Zhao Zhijia. Tigr: Transforming irregular graphs for GPU-friendly graph processing[C]//Proc of the 23rd Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2018: 622–636

[12] Wang Hao, Geng Liang, Lee R, et al. SEP-graph: Finding shortest execution paths for graph processing under a hybrid framework on GPU[C]//Proc of the 24th Symp on Principles and Practice of Parallel Programming. New York: ACM, 2019: 38–52

[13] Sabet A H N, Zhao Zhijia, Gupta R. Subway: Minimizing data transfer during out-of-GPU-memory graph processing[C]//Proc of the 15th European Conf on Computer Systems. New York: ACM, 2020: 1−16

[14] Wang Qiange, Ai Xin, Zhang Yanfeng, et al. HyTGraph: GPU-accelerated graph processing with hybrid transfer management[C]// Proc of the 39th Int Conf on Data Engineering (ICDE). Piscataway, NJ: IEEE, 2023: 558−571

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

[16] Ben-Nun T, Sutton M, Pai S, et al. Groute: Asynchronous multi-GPU programming model with applications to large-scale graph processing[J]. ACM Transactions on Parallel Computing, 2020, 7(3): 1−27

[17] Meng Ke, Geng Liang, Li Xue, et al. Efficient multi-GPU graph processing with remote work stealing [C]//Proc of the 39th Int Conf on Data Engineering (ICDE). Piscataway, NJ: IEEE, 2023: 191−204

[18] Sengupta D, Song S L. EvoGraph: On-the-fly efficient mining of evolving graphs on GPU[C]//Proc of the High Performance Computing: 32nd Int Conf, ISC High Performance 2017. Berlin: Springer, 2017: 97–119

[19] Mariappan M, Vora K. GraphBolt: Dependency-driven synchronous processing of streaming Graphs[C]//Proc of the 14th EuroSys Conf 2019. New York: ACM, 2019: 1−16

[20] Kumar P, Huang H H. GraphOne: A data store for real-time analytics on evolving graphs[C].//Proc of the 17th USENIX Conf on File and Storage Technologies (FAST'19). Berkeley, CA: USENIX Association, 2019: 249−263

[21] Iyer A P, Pu Qifan, Patel K, et al. TEGRA: Efficient ad-hoc analytics on evolving graphs[C]//Proc of the 18th USENIX Symp on Networked Systems Design and Implementation (NSDI'21). Berkeley, CA: USENIX Association, 2021: 337−355

[22] Martella C, Shaposhnik R, Logothetis D, et al. Practical Graph Analytics with Apache Giraph [M]. Berkeley, CA: Apress, 2015

[23] Gonzalez J E, Xin R S, Dave A, et al. GraphX: Graph processing in a distributed dataflow framework[C]//Proc of the 11th USENIX Symp on Operating Systems Design and Implementation. Berkeley, CA: USENIX Association, 2014: 599−613

[24] Fan Wenfei, Xu Jingbo, Wu Yinghui, et al. GRAPE: Parallelizing sequential graph computations[J]. Proceedings of the VLDB Endowment, 2017, 10(12): 1889−1892

[25] Yang Ke, Zhang Mingxing, Chen Kang, et al. KnightKing: a fast distributed graph random walk engine[C]//Proc of the 27th ACM Symp on Operating Systems Principles. New York: ACM, 2019: 524–537

[26] Tencent: Plato code [EB/OL]. [2024−01−20].https://github.com/Tencent/plato/

[27] Fan Wenfei, He Tao, Lai Longbin, et al. GraphScope: a unified engine for big graph processing[J]. Proceedings of the VLDB Endowment, 2021, 14(12): 2879−2892

[28] Ant Group: TuGraph website [EB/OL]. [2024−01−20].https://www.tugraph.org/

[29] Shi Xuanhua, Zheng Zhigao, Zhou Yongluan, et al. Graph processing on GPUs: A survey[J]. ACM Computing Surveys, 2018, 50(6): 1−35

[30] Huang Jianqiang, Qin Wei, Wang Xiaoying, et al. Survey of external memory large-scale graph processing on a multi-core system[J]. The Journal of Supercomputing, 2020, 76(1): 549−579

[31] Wang Jing, Zhang Lu, Wang Pengyu, et al. Survey on memory system optimization technologies for graph computing[J]. SCIENTIA SINICA Informationis, 2019, 49(3): 295−313 (in Chinese)
(王靖, 张路, 王鹏宇, 等. 面向图计算的内存系统优化技术综述[J]. 中国科学: 信息科学, 2019, 49(3): 295−313)

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

[33] Zhang Yu, Jiang Xinyu, Yu Hui, et al. Research and trend of graph computing architecture and software system[J]. Journal of Computer Research and Development, 2024, 61(1): 20−42 (in Chinese)
(张宇, 姜新宇, 余辉, 等. 图计算体系结构和系统软件关键技术综述[J]. 计算机研究与发展, 2024, 61(1): 20−42)

[34] Heidari S, Simmhan Y, Calheiros R N, et al. Scalable graph processing frameworks: A taxonomy and open challenges[J]. ACM Computing Surveys, 2018, 51(3): 1−53

[35] Sahu S, Mhedhbi A, Salihoglu S, et al. The ubiquity of large graphs and surprising challenges of graph processing[J]. Proceedings of the VLDB Endowment, 2017, 11(4): 420−431

[36] Chakrabarti D, Zhan Yiping, Faloutsos C. R-MAT: A recursive model for graph mining[C]//Proc of the 2004 SIAM Int Conf on Data Mining. Philadelphia, PA: SIAM, 2004: 442−446

[37] Takac L, Zabovsky M. Data analysis in public social networks[C]//Proc of the Int scientific Conf and Int workshop present day trends of innovations, 2012, 1−6

[38] Leskovec J, Lang K, Dasgupta A, et al. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters[J]. Internet Mathematics, 2008, 6(1): 29−123

[39] Sundaram N, Satish N, Patwary M M A, et al. GraphMat: high performance graph analytics made productive[J]. Proceedings of the VLDB Endowment, 2015, 8(11): 1214−1225

[40] Yang C, Buluç A, Owens J D. GraphBLAST: A high-performance linear algebra-based graph framework on the GPU[J]. ACM Transactions on Mathematical Software, 2022, 48(1): 1−51

[41] Hong Changwan, Sukumaran-Rajam A, Kim J, et al. MultiGraph: Efficient graph processing on GPUs[C]//Proc of the 26th Int Conf on Parallel Architectures and Compilation Techniques (PACT). Piscataway, NJ: IEEE, 2017: 27−40

[42] Liu Hang, Huang H H. SIMD-X: Programming and processing of graph algorithms on GPUs[C]//Proc of the 2019 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2019: 411–427

[43] Green O, Bader D A. cuSTINGER: Supporting dynamic graph algorithms for GPUs[C]//Proc of the 2016 IEEE High Performance Extreme Computing Conf (HPEC). Piscataway, NJ: IEEE, 2016: 1−6

[44] Busato F, Green O, Bombieri N, et al. Hornet: An efficient data structure for dynamic sparse graphs and matrices on GPUs[C]//Proc of the 2018 IEEE High Performance Extreme Computing Conf (HPEC). Piscataway, NJ: IEEE, 2018: 1−7

[45] Winter M, Mlakar D, Zayer R, et al. faimGraph: High performance management of fully-dynamic graphs under tight memory constraints on the GPU[C]//Proc of the SC18: Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2018: 754−766

[46] Sha Mo, Li Yuchen, He Bingsheng, et al. Accelerating dynamic graph analytics on GPUs[J]. Proceedings of the VLDB Endowment, 2017, 11(1): 107−120

[47] Bader D A, Berry J, Kahan S, et al. Graph500 [EB/OL]. [2024−01−20]. http://www.graph500.org

[48] Dong Rongyu, Cao Huawei, Ye Xiaochun, et al. Highly efficient and GPU-friendly implementation of BFS on single-node system[C]// Proc of 2020 IEEE Int Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom). Piscataway, NJ: IEEE, 2020: 544−553

[49] Liu Hang, Huang H H. Enterprise: Breadth-first graph traversal on GPUs[C]//Proc of the Int Conf for High Performance Computing, Networking, Storage and Analysis. New York: ACM, 2015: 1−12

[50] Dijkstra E W. A Note on Two Problems in Connexion with Graphs [M]. Edsger Wybe Dijkstra: His Life, Work, and Legacy. 2022: 287−290

[51] Bellman R. On a routing problem[J]. Quarterly of Applied Mathematics, 1958, 16(1): 87−90

[52] Meyer U, Sanders P. δ-stepping: A parallel single source shortest path algorithm[C]//Proc of the European Symp on algorithms. Berlin: Springer, 1998: 393−404

[53] Wang Kai, Fussell D, Lin Calvin. A fast work-efficient SSSP algorithm for GPUs[C]//Proc of the 26th ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2021: 133–146

[54] Dong Xiaojun, Gu Yan, Sun Yihan, et al. Efficient stepping algorithms and implementations for parallel shortest paths[C]//Proc of the 33rd ACM Symp on Parallelism in Algorithms and Architectures. New York: ACM, 2021: 184–197

[55] Dhulipala L, Blelloch G, Shun Julian. Julienne: A framework for parallel graph algorithms using work-efficient bucketing[C]//Proc of the 29th ACM Symp on Parallelism in Algorithms and Architectures. New York: ACM, 2017: 293–304

[56] Zhang Yuan, Cao Huawei, Zhang Jie, et al. A bucket-aware asynchronous single-source shortest path algorithm on GPU[C]// Proc of the 52nd Int Conf on Parallel Processing Workshops. New York: ACM, 2023: 50–60

[57] Beamer S, Asanović K, Patterson D. Reducing PageRank communication via propagation blocking[C]//Proc of the 2017 IEEE Int Parallel and Distributed Processing Symp (IPDPS). Piscataway, NJ: IEEE, 2017: 820−831

[58] Lakhotia K, Kannan R, Prasanna V. Accelerating PageRank using partition-centric processing[C]//Proc of the 2018 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2018: 427–440

[59] Pandey S, Li X S, Buluc A, et al. H-INDEX: Hash-indexing for parallel triangle counting on GPUs[C]//Proc of the 2019 IEEE High Performance Extreme Computing Conf (HPEC). Piscataway, NJ:

IEEE, 2019: 1−7

[60] Yaşar A, Rajamanickam S, Wolf M, et al. Fast triangle counting using Cilk[C]//Proc of the 2018 IEEE High Performance extreme Computing Conf (HPEC). Piscataway, NJ: IEEE, 2018: 1−7

[61] Wasserman S, Faust K. Social network analysis: Methods and applications[J]. American Ethnologist, 1997, 24(1): 219−220

[62] Mislove A, Marcon M, Gummadi K P, et al. Measurement and analysis of online social networks[C]//Proc of the 7th ACM SIGCOMM Conf on Internet Measurement. New York: ACM, 2007: 29−42

[63] Ye Chang, Li Yuchen, He Bingsheng, et al. GPU-accelerated graph label propagation for real-time fraud detection[C]//Proc of the 2021 Int Conf on Management of Data. New York: ACM, 2021: 2348–2356

[64] Jiang Jiaxin, Li Yuan, He Bingsheng, et al. Spade: A real-time fraud detection framework on evolving graphs[J]. Proceedings of the VLDB Endowment, 2022, 16(3): 461−469

[65] Alam M A, Faruq M O. Finding shortest path for road network using Dijkstra's algorithm[J]. Bangladesh Journal of Multidisciplinary Scientific Research, 2019, 1(2): 41−45

[66] Xiao Yunpeng, He Xi, Yang Chen, et al. Dynamic graph computing: A method of finding companion vehicles from traffic streaming data[J]. Information Sciences, 2022, 591: 128−141

[67] Huang Zan, Chung Wingyan, Chen Hsinchun, et al. A graph model for E-commerce recommender systems[J]. Journal of the American Society for Information Science, 2004, 55(3): 259−274

[68] Mcauley J, Pandey R, Leskovec J. Inferring networks of substitutable and complementary products[C]//Proc of the 21st ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2015: 785–794

[69] Yang Xiaoyong, Zhu Yadong, Zhang Yi, et al. Large scale product graph construction for recommendation in E-commerce [J]. arXiv preprint, arXiv: 2010.05525, 2020

[70] Afarin M, Gao Chao, Rahman S, et al. CommonGraph: Graph analytics on evolving data[C]//Proc of the 28th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2023: 133–145

[71] Zou Lei, Zhang Fan, Lin Yinnian, et al. An efficient data structure for dynamic graph on GPUS[J]. IEEE Transactions on Knowledge and Data Engineering, 2023, 35(11): 11051-11066

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

[73] 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

[74] Zhang Yu, Liao Xiaofei, Jin Hai, et al. DiGraph: An efficient path-based iterative directed graph processing system on multiple GPUs[C]//Proc of the 24th Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2019: 601–614

[75] Zhang Yu, Peng Da, Liao Xiaofei, et al. LargeGraph: An efficient dependency-aware GPU-accelerated large-scale graph processing[J]. ACM Transactions on Architecture and Code Optimization. New York: ACM, 2021, 18(4): 1−24

[76] 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

[77] Lin Heng, Zhu Xiaowei, Yu Bowen, et al. ShenTu: Processing multi-trillion edge graphs on millions of cores in seconds[C]//Proc of the SC18: Int Conf for High Performance Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2018: 706−716

[78] Tian Yanyan, Balmin A, Corsten S A, et al. From "think like a vertex" to "think like a graph"[J]. Proceedings of the VLDB Endowment, 2013, 7((3):): 193−204

[79] Kang U, Tsourakakis C E, Faloutsos C. PEGASUS: A peta-scale graph mining system implementation and observations[C]//Proc of the 9th IEEE Int Conf on Data Mining. Piscataway, NJ: IEEE, 2009: 229−238

[80] Vora K, Gupta R, Xu Guoqing. KickStarter: Fast and accurate computations on streaming graphs via trimmed approximations[C]// Proc of the 22d Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2017: 237–251

[81] Jiang Xiaolin, Xu Chengshuo, Yin Xizhe, et al. Tripoline: Generalized incremental graph processing via graph triangle inequality[C]//Proc of the 16th European Conf on Computer Systems. New York: ACM, 2021: 17–32

[82] Islam A a R, Dai Dong. DGAP: Efficient dynamic graph analysis on persistent memory[C]//Proc of the Int Conf for High Performance Computing, Networking, Storage and Analysis. New York: ACM, 2023: 1−13

[83] Sengupta D, Sundaram N, Zhu Xia, et al. GraphIn: An online high performance incremental graph processing framework[C]//Proc of the 22nd Int European Conf on Parallel and Distributed Computing (Euro-Par 2016). Berlin: Springer, 2016: 319−333

[84] Zhu Huanzhou, He Ligang, Leeke M, et al. WolfGraph: The edge-centric graph processing on GPU[J]. Future Generation Computer Systems, 2020, 111: 552−569

[85] Sheng Feng, Cao Qiang, Yao Jie. Exploiting buffered updates for fast streaming graph analysis[J]. IEEE Transactions on Computers, 2021, 70(2): 255−269

[86] Feng Guanyu, Ma Zixuan, Li Daixuan, et al. RisGraph: A real-time streaming system for evolving graphs to support sub-millisecond per-update analysis at millions Ops/s[C]//Proc of the 2021 Int Conf on Management of Data. New York: ACM, 2021: 513–527

[87] Cheng R, Hong Ji, Kyrola A, et al. Kineograph: Taking the pulse of a fast-changing and connected world[C]//Proc of the 7th ACM European Conf on Computer Systems. New York: ACM, 2012: 85–98

[88] Ju Wuyang, Li Jianxin, Yu Weiren, et al. iGraph: An incremental data processing system for dynamic graph[J]. Frontiers of Computer Science, 2016, 10: 462−476

[89] Jaiyeoba W, Skadron K. Graphtinker: A high performance data

structure for dynamic graph processing[C]//Proc of the 2019 IEEE Int Parallel and Distributed Processing Symp (IPDPS). Piscataway, NJ: IEEE, 2019: 1030−1041

[90] Zhang Yuan, Cao Huawei, Liang Yan, et al. FSGraph: Fast and scalable implementation of graph traversal on GPUs[J]. CCF Transactions on High Performance Computing, 2023, 5(3): 277−291

[91] Sengupta D, Song S L, Agarwal K, et al. GraphReduce: Processing large-scale graphs on accelerator-based systems[C]//Proc of the Int Conf for High Performance Computing, Networking, Storage and Analysis. New York: ACM, 2015: 1−12

[92] Xue Rui, Han Haoyu, Zhao Tong, et al. Large-scale graph neural networks: The past and new frontiers[C]//Proc of the 29th ACM SIGKDD Conf on Knowledge Discovery and Data Mining. New York: ACM, 2023: 5835–5836

[93] Ma Lingxiao, Yang Zhi, Miao Youshan, et al. Neugraph: Parallel deep neural network computation on large graphs[C]//Proc of the 2019 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2019: 443–457

[94] Mariappan M, Che J, Vora K. DZiG: Sparsity-aware incremental processing of streaming graphs[C]//Proc of the 16th European Conf on Computer Systems. New York: ACM, 2021: 83–98

[95] Gong Shufeng, Tian Chao, Yin Qiang, et al. Automating incremental graph processing with flexible memoization[J]. Proceedings of the VLDB Endowment, 2021, 14(9): 1613−1625

[96] Yu Song, Gong Shufeng, Zhang Yanfeng, et al. Layph: Making change propagation constraint in incremental graph processing by layering graph [J]. arXiv preprint, arXiv: 2304.07458, 2023

[97] Jiang Zihan, Mao Fubing, Guo Yapu, et al. ACGraph: Accelerating streaming graph processing via dependence hierarchy[C]//Proc of the 2023 60th ACM/IEEE Design Automation Conf (DAC). Piscataway, NJ: IEEE, 2023: 1−6

[98] Xu Xianghao, Jiang Hong, Wang Fang, et al. GraphSD: A state and dependency aware out-of-core graph processing system[C]//Proc of the 51st Int Conf on Parallel Processing. New York: ACM, 2023: 1−11

[99] Gera P, Kim H, Sao P, et al. Traversing large graphs on GPUs with unified memory[J]. Proceedings of the VLDB Endowment, 2020, 13(7): 1119−1133

[100] Dhulipala L, Blelloch G E, Shun Julian. Low-latency graph streaming using compressed purely-functional trees[C]//Proc of the 40th ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 2019: 918–934

[101] Zhu Xiaowei, Feng Guanyu, Serafini M, et al. LiveGraph: A transactional graph storage system with purely sequential adjacency list scans[J]. Proceedings of the VLDB Endowment, 2020, 13(7): 1020−1034

[102] Kim J, Dally W J, Scott S, et al. Technology-driven, highly-scalable dragonfly topology[C]//Proc of 2008 Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2008: 77−88

[103] Min S W, Mailthody V S, Qureshi Z, et al. EMOGI: Efficient memory-access for out-of-memory graph-traversal in GPUs[J]. Proceedings of the VLDB Endowment, 2020, 14(2): 114−127

[104] Zhou Shijie, Kannan R, Prasanna V K, et al. HitGraph: High-throughput graph processing framework on FPGA[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(10): 2249−2264

[105] Chen Xinyu, Tan Hongshi, Chen Yao, et al. ThunderGP: HLS-based graph processing framework on FPGAs[C]//Proc of 2021 ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2021: 69–80

[106] Dai Guohao, Huang Tianhao, Wang Yu, et al. NewGraph: Balanced large-scale graph processing on FPGAs with low preprocessing overheads[C]//Proc of 2018 IEEE 26th Annual Int Symp on Field-Programmable Custom Computing Machines (FCCM). Piscataway, NJ: IEEE, 2018: 208−208

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

[108] Chen Xinyu, Chen Yao, Cheng Feng, et al. ReGraph: Scaling graph processing on HBM-enabled FPGAs with heterogeneous pipelines[C]//Proc of the 55th IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2022: 1342−1358

[109] Wang Qinggang, Zheng Long, Huang Yu, et al. GraSU: A fast graph update library for FPGA-based dynamic graph processing[C]//Proc of 2021 ACM/SIGDA Int Symp on Field-Programmable Gate Arrays. New York: ACM, 2021: 149–159

[110] Ham T J, Wu Lisa, 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: 1−13

[111] 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

[112] Dadu V, Liu Sihao, Nowatzki T. PolyGraph: Exposing the value of flexibility for graph processing accelerators[C]//Proc of the 48th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2021: 595–608

[113] Yang Yifan, Li Zhaoshi, Deng Yangdong, et al. GraphABCD: Scaling out graph analytics with asynchronous block coordinate descent[C]//Proc of the ACM/IEEE 47th Annual Int Symp on Computer Architecture. Piscataway, NJ: IEEE, 2020: 419–432

[114] Rahman S, Abu-Ghazaleh N, Gupta R. GraphPulse: An event-driven hardware accelerator for asynchronous graph processing[C]//Proc of the 53rd Annual IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2020: 908−921

[115] Rahman S, Afarin M, Abu-Ghazaleh N, et al. JetStream: Graph analytics on streaming data with event-driven hardware accelerator[C]//Proc of the 54th Annual IEEE/ACM Int Symp on Microarchitecture. New York: ACM, 2021: 1091–1105

[116] Zhang Yu, Liao Xiaofei, Jin Hai, et al. DepGraph: A dependency-driven accelerator for efficient iterative graph processing[C]//Proc of the 2021 IEEE Int Symp on High-Performance Computer Architecture (HPCA). Piscataway, NJ: IEEE, 2021: 371−384

[117] Zhang Kaiyuan, Chen Rong, Chen Haibo. NUMA-aware graph-structured analytics[C]//Proc of the 20th ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2015: 183–193

[118] Zhang Yunming, Kiriansky V, Mendis C, et al. Making caches work for graph analytics[C]//Proc of the 2017 IEEE Int Conf on Big Data (Big Data). Piscataway, NJ: IEEE, 2017: 293−302

[119] Shun Julian, Dhulipala L, Blelloch G E. Smaller and faster: Parallel processing of compressed graphs with Ligra+[C]//Proc of the 2015 Data Compression Conf. Piscataway, NJ: IEEE, 2015: 403−412

[120] Vora K. LUMOS: Dependency-driven disk-based graph processing[C]//Proc of the 2019 USENIX Conf on USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2019: 429–442

[121] Cheng Jiefeng, Liu Qin, Li Zhenguo, et al. VENUS: Vertex-centric streamlined graph computation on a single PC[C] //Proc of the 2015 IEEE 31st Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2015: 1131−1142

[122] Ai Zhiyuan, Zhang Mingxing, Wu Yongwei, et al. Squeezing out all the value of loaded data: An out-of-core graph processing system with reduced disk I/O[C]//Proc of the 2017 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2017: 125–137

[123] Matam K K, Hashemi H, Annavaram M. MultiLogVC: Efficient out-of-core graph processing framework for flash storage[C]//Proc of the 2021 IEEE Int Parallel and Distributed Processing Symp (IPDPS). Piscataway, NJ: IEEE, 2021: 245−255

[124] Chen Rong, Shi Jiaxin, Chen Yanzhe, et al. PowerLyra: Differentiated graph computation and partitioning on skewed graphs[J]. ACM Transactions on Parallel Computing, 2019, 5(3): 1−39

[125] Zhong Jianlong, He Bingsheng. Medusa: Simplified graph processing on GPUs[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(6): 1543−1552

[126] Meng Ke, Li Jiajia, Tan Guangming, et al. A pattern based algorithmic autotuner for graph processing on GPUs[C]//Proc of the 24th Symp on Principles and Practice of Parallel Programming. New York: ACM, 2019: 201–213

[127] Pai S, Pingali K. A compiler for throughput optimization of graph algorithms on GPUs[C]//Proc of the 2016 ACM SIGPLAN Int Conf on Object-Oriented Programming, Systems, Languages, and Applications. New York: ACM, 2016: 1–19

[128] Gharaibeh A, Santos-Neto E, Costa L, et al. Efficient large-scale graph processing on hybrid CPU and GPU systems [J]. arXiv preprint, arXiv: 1312.3018, 2013

[129] Gharaibeh A, Costa L B, Santos-Neto E, et al. A yoke of oxen and a thousand chickens for heavy lifting graph processing[C]//Proc of the 21st Int Conf on Parallel Architectures and Compilation Techniques. New York: ACM, 2012: 345–354

[130] Han Wei, Mawhirter D, Wu Bo, et al. Graphie: Large-scale asynchronous graph traversals on just a GPU[C]//Proc of 2017 26th Int Conf on Parallel Architectures and Compilation Techniques (PACT). Piscataway, NJ: IEEE, 2017: 233−245

[131] Kim M S, An K, Park H, et al. GTS: A fast and scalable graph processing method based on streaming topology to GPUs[C]//Proc of the 2016 Int Conf on Management of Data. New York: ACM, 2016: 447–461

[132] 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 2017 USENIX Conf on Usenix Annual Technical Conf. Berkeley, CA: USENIX Association, 2017: 195–207

[133] Wang Pengyu, Wang Jing, Li Chao, et al. Grus: Toward unified-memory-efficient high-performance graph processing on GPU[J]. ACM Transactions on Architecture and Code Optimization, 2021, 18(2): 1−25

[134] Jia Zhihao, Kwon Y, Shipman G, et al. A distributed multi-GPU system for fast graph processing[J]. Proceedings of the VLDB Endowment, 2017, 11(3): 297−310

[135] Dathathri R, Gill G, Hoang L, et al. Gluon: A communication-optimizing substrate for distributed heterogeneous graph analytics[C]//Proc of the 39th ACM SIGPLAN Conf on Programming Language Design and Implementation. New York: ACM, 2018: 752–768

[136] Ji Yuede, Liu Hang, Huang H H. SWARMGRAPH: Analyzing large-scale in-memory graphs on GPUs[C]//Proc of the 2020 IEEE 22nd Int Conf on High Performance Computing and Communications; IEEE 18th Int Conf on Smart City; IEEE 6th Int Conf on Data Science and Systems (HPCC/SmartCity/DSS). Piscataway, NJ: IEEE, 2020: 52−59

[137] Brahmakshatriya A, Zhang Yunming, Hong Changwan, et al. Compiling graph applications for GPUs with GraphIt[C]//Proc of the 2021 IEEE/ACM Int Symp on Code Generation and Optimization (CGO). Piscataway, NJ: IEEE, 2021: 248−261

[138] Yuan Pingpeng, Xie Changfeng, Liu Ling, et al. PathGraph: A path centric graph processing system[J]. IEEE Transactions on Parallel and Distributed Systems. Piscataway, NJ: IEEE, 2016, 27(10): 2998−3012

[139] Chi Yuze, Dai Guohao, Wang Yu, et al. NXGraph: An efficient graph processing system on a single machine[C]//Proc of the 2016 IEEE 32nd Int Conf on Data Engineering (ICDE). Piscataway, NJ: IEEE, 2016: 409−420

[140] Xu Xianghao, Wang Fang, Jiang Hong, et al. A hybrid update strategy for I/O-efficient out-of-core graph processing[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(8): 1767−1782

[141] Zhang Mingxing, Wu Yongwei, Zhuo Youwei, et al. Wonderland: A novel abstraction-based out-of-core graph processing system[C]//Proc of the 23rd Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2018: 608–621

[142] Zheng Long, Li Xianliang, Zheng Yaohui, et al. Scaph: Scalable GPU-accelerated graph processing with value-driven differential scheduling[C]//Proc of the 2020 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2020: 573−588

[143] Tang Ruiqi, Zhao Ziyi, Wang Kailun, et al. Ascetic: Enhancing cross-iterations data efficiency in out-of-memory graph processing on GPUs[C]//Proc of the 50th Int Conf on Parallel Processing. New York: ACM, 2021: 1−10

[144] Vora K, Xu Guoqing, Gupta R. Load the edges you need: a generic I/O optimization for disk-based graph processing[C]//Proc of the 2016 USENIX Conf on Usenix Annual Technical Conf. Berkeley, CA: USENIX Association, 2016: 507–522

[145] Pan Zhenxuan, Wu Tao, Zhao Qingwen, et al. GeaFlow: A graph extended and accelerated dataflow system[C]//Proc of the ACM on Management of Data. New York: ACM, 2023, 1(2): 1−27

[146] Ediger D, Mccoll R, Riedy J, et al. STINGER: High performance data structure for streaming graphs[C]//Proc of the 2012 IEEE Conf on High Performance Extreme Computing. Piscataway, NJ: IEEE, 2012: 1−5

[147] Macko P, Marathe V J, Margo D W, et al. LLAMA: Efficient graph analytics using large multiversioned arrays[C]//Proc of the 2015 IEEE 31st Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2015: 363−374

[148] Leo D D, Boncz P. Teseo and the analysis of structural dynamic graphs[J]. Proceedings of the VLDB Endowment, 2021, 14(6): 1053−1066

[149] Basak A, Lin Jilan, Lorica R, et al. SAGA-Bench: Software and hardware characterization of streaming graph analytics workloads[C]//Proc of the 2020 IEEE Int Symp on Performance Analysis of Systems and Software (ISPASS). Piscataway, NJ: IEEE, 2020: 12−23

[150] Islam A A R, Dai Dong, Cheng Dazhao. VCSR: Mutable CSR graph format using vertex-centric packed memory array[C]//Proc of the 2022 22nd IEEE Int Symp on Cluster, Cloud and Internet Computing (CCGrid). Piscataway, NJ: IEEE, 2022: 71−80

[151] Wang Rui, He Shuibing, Zong Weixu, et al. XPGraph: XPline-friendly persistent memory graph stores for large-scale evolving graphs[C]//Proc of the 2022 55th IEEE/ACM Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2022: 1308−1325

[152] Chen Hongtao, Zhang Mingxing, Yang Ke, et al. Achieving sub-second pairwise query over evolving graphs[C]//Proc of the 28th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2023: 1−15

[153] Shen Sijie, Yao Zihang, Shi Lin, et al. Bridging the gap between relational OLTP and graph-based OLAP[C]//Proc of the 2023 USENIX Annual Technical Conf (USENIX ATC 23). Berkeley, CA: USENIX Association, 2023: 181−196

[154] Vaziri P, Vora K. Controlling memory footprint of stateful streaming graph processing[C]//Proc of the 2021 USENIX Annual Technical Conf (USENIX ATC 21). Berkeley, CA: USENIX Association, 2021: 269−283

[155] Zhang Chenglong, Cao Huawei, Wang Guobo, et al. Efficient optimization of graph computing on high-throughput computer[J]. Journal of Computer Research and Development, 2020, 57(6): 1152−1163 (in Chinese)
(张承龙, 曹华伟, 王国波, 等. 面向高通量计算机的图算法优化技术[J]. 计算机研究与发展, 2020, 57(6): 1152−1163)

[156] Sha Mo, Li Yuchen, Tan K L. GPU-based graph traversal on compressed graphs[C]//Proc of the 2019 Int Conf on Management of Data. New York: ACM, 2019: 775–792

[157] Chen Zheng, Zhang Feng, Guan Jiawei, et al. CompressGraph: Efficient parallel graph analytics with rule-based compression [J] // Proc of the ACM on Management of Data. New York: ACM, 2023, 1: 1−31

[158] Lee E, Kim J, Lim K, et al. Pre-select static caching and neighborhood ordering for BFS-like algorithms on disk-based graph engines[C]//Proc of the 2019 USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2019: 459–473

[159] Yang T Y, Liang Yuhong, Yang M C. Practicably boosting the processing performance of BFS-like algorithms on semi-external graph system via I/O-efficient graph ordering[C]//Proc of the 20th USENIX Conf on File and Storage Technologies (FAST 22). Berkeley, CA: USENIX Association, 2022: 381−396

[160] Wei Hao, Yu J X, Lu Can, et al. Speedup graph processing by graph ordering[C]//Proc of the 2016 Int Conf on Management of Data. New York: ACM, 2016: 1813–1828

[161] Balaji V, Lucia B. When is graph reordering an optimization? studying the effect of lightweight graph reordering across applications and input graphs[C]//Proc of the 2018 IEEE Int Symp on Workload Characterization (IISWC). Piscataway, NJ: IEEE, 2018: 203−214

[162] Guan Nan, Stigge M, Yi Wang, et al. Cache-aware scheduling and analysis for multicores[C]//Proc of the 7th ACM Int Conf on Embedded software. New York: ACM, 2009: 245–254

[163] Han W S, Lee S, Park K, et al. TurboGraph: A fast parallel graph engine handling billion-scale graphs in a single PC[C]//Proc of the 19th ACM SIGKDD Int Conf on Knowledge Discovery and Data Mining. New York: ACM, 2013: 77–85

[164] Lin Shuai, Wang Rui, Li Yongkun, et al. Towards fast large-scale graph analysis via two-dimensional balanced partitioning[C]//Proc of the 51st Int Conf on Parallel Processing. New York: ACM, 2023: 1−11

[165] Gong Shufeng, Zhang Yanfeng, Yu Ge. HBP: Hotness balanced partition for prioritized iterative graph computations[C]//Proc of the 2020 IEEE 36th Int Conf on Data Engineering (ICDE). Piscataway, NJ: IEEE, 2020: 1942−1945

[166] NVIDIA. Unified memory for cuda beginners [EB/OL]. [2024−01−20].https://developer.nvidia.com/blog/unified-memory-cuda-beginners/

[167] NVIDIAL. Nvlink and nvswitch [EB/OL]. [2024−01−20].https://www.nvidia.com/en-us/data-center/nvlink/

[168] Lin Yen, Jeng J Y, Liu Y Y, et al. A review of PCI express protocol-based systems in response to 5G application demand[J]. Electronics, 2022, 11((5):): 678

[169] Wang Jing, Li Chao, Liu Yibo, et al. Fargraph+: Excavating the parallelism of graph processing workload on RDMA-based far memory system[J]. Journal of Parallel and Distributed Computing, 2023, 177: 144−159

[170] Wang Jing, Li Chao, Wang Taolei, et al. Excavating the potential of graph workload on RDMA-based far memory architecture[C]//Proc of the 2022 IEEE Int Parallel and Distributed Processing Symp

(IPDPS). Piscataway, NJ: IEEE, 2022: 1029−1039
[171] Cui Pengjie, Yuan Ye, Li Cenhao, et al. RGraph: Effective distributed graph data processing system based on RDMA[J]. Journal of Software, 2022, 33(3): 1018−1042 (in Chinese)
(崔鹏杰, 袁野, 李岑浩, 等. RGraph: 基于 RDMA 的高效分布式图数据处理系统[J]. 软件学报, 2022, 33(3): 1018−1042)
[172] Brock B, Buluç A, Yelick K J a P A. RDMA-based algorithms for sparse matrix multiplication on GPUs [J]. arXiv preprint, arXiv: 2311.18141, 2023
[173] Kwak H, Lee C, Park H, et al. What is Twitter, a social network or a news media?[C]//Proc of the 19th Int Conf on World Wide Web. New York: ACM, 2010: 591–600
[174] Boldi P, Santini M, Vigna S. Laboratory for web algorithmics [EB/OL]. [2024−01−20]. http://law.di.unimi.it/
[175] Rossi R, Ahmed N. The network data repository with interactive graph analytics and visualization [J] //Proc of the AAAI Conf on Artificial Intelligence. Palo Alto, CA: AAAI, 2015, 29: 1−2
[176] Yang J, Leskovec J. Defining and evaluating network communities based on ground-truth[C]//Proc of the ACM SIGKDD Workshop on Mining Data Semantics. New York: ACM, 2012: 1−8
[177] Leskovec J, Sosič R. Snap: A general-purpose network analysis and graph-mining library[J]. ACM Transactions on Intelligent Systems and Technology, 2016, 8(1): 1−20
[178] Boldi P, Vigna S. The webgraph framework I: Compression techniques[C]//Proc of the 13th Int Conf on World Wide Web. New York: ACM, 2004: 595–602
[179] Boldi P, Rosa M, Santini M, et al. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks[C]//Proc of the 20th Int Conf on World Wide Web. New York: ACM, 2011: 587–596

Zhang Yuan, born in 1990. PhD candidate, engineer. Member of CCF. Her main research interests include high-performance computing and graph algorithm optimization.
张 园, 1990 年生. 博士研究生, 工程师. CCF 会员. 主要研究方向为高性能计算、图算法优化.

Cao Huawei, born in 1989. PhD, associate professor master supervisor. Member of CCF. His main research interests include parallel computing and high throughput computing architecture.
曹华伟, 1989 年生. 博士, 副研究员, 硕士生导师. CCF 会员. 主要研究方向为并行计算、高通量计算架构.

Zhang Jie, born in 1999. Master, assistant engineer. Her main research interests include high-performance computing and graph algorithm optimization.
张 婕, 1999 年生. 硕士, 助理工程师. 主要研究方向为高性能计算、图算法优化.

Shen Yue, born in 1999. Master candidate. Her main research interests include parallel computing and dynamic graph processing.
申 玥, 1999 年生. 硕士研究生. 主要研究方向为并行计算、动态图处理.

Sun Yiming, born in 1997. PhD candidate. His main research interests include parallel computing and graph algorithm optimization.
孙一鸣, 1997 年生. 博士研究生. 主要研究方向为并行计算、图算法优化.

Dun Ming, born in 1997. Master, assistant engineer. Her main research interests include high-performance computing and sparse algorithm optimization.
敦 明, 1997 年生. 硕士, 助理工程师. 主要研究方向为高性能计算、稀疏算法优化.

An Xuejun, born in 1966. PhD, senior engineer, PhD supervisor. Member of CCF. His main research interests include computer system architecture and high-performance interconnection networks.
安学军, 1966 年生. 博士, 高级工程师, 博士生导师. CCF 会员. 主要研究方向为计算机系统结构、高性能互连网络.

Ye Xiaochun, born in 1981. PhD, professor, PhD supervisor. Member of CCF. His main research interests include algorithm paralleling and optimizing software simulation, and architecture for high throughput computer.
叶笑春, 1981 年生. 博士, 研究员, 博士生导师. CCF 会员. 主要研究方向为算法并行与优化软件仿真、高通量计算机体系结构.