

**课程报告**

**课程名称：并行程序设计**

**专业班级： 计算机硕2101班**

**学 号： M202173535**

**姓 名： 蒋子涵**

**指导教师： 石宣化**

**报告日期： 2021.12.29**

**计算机科学与技术学院**

# 大规模图计算系统的并行优化技术综述

## 选题原因

图计算(graph processing或graph computing)是研究客观世界当中事物与事物之间的关系并对其进行完整地刻画、计算和分析的一门处理图的技术, 是大数据关联属性的最佳表达方式。由于图计算能够用于分析数据之间的关联关系，因此其在互联网应用、科学计算、社会计算、商业计算等诸多领域得到应用广泛，已经成为大数据处理的主流模式之一。当前，随着互联网的快速普及，社会的数字化变革以及经济的快速发展，表达数据关联性的图数据的规模正在呈爆发式增长。如何更高效地对图数据进行并行处理，加速图计算算法的执行也成为了大数据研究的主要问题之一。

## 2. 问题解释

### 2.1图计算简介

图是一种由顶点(vertices)和连接顶点的边(edges)构成的数据结构, G = (V, E), 其中 V表示顶点集合, E表示边的集合. e = (vi, vj) 表示从顶点vi到vj的一条有向边. 同时, 每一个顶点与每一条边都有属于自己的属性值. 不同的领域属性值可代表不同的含义. 如社交网络中, 顶点的属性值为个人的热度, 而边的属性值则表示为有关联的两人之间的紧密程度。在此计算过程中，顶点和边都会包含计算过程中的状态数据，然后通过对每个顶点和边进行一系列的迭代计算来完成状态数据的更新。图计算的结果为图中所有顶点和边的最终状态的聚合。

图计算能够挖掘事物之间潜在不易洞察的行为和联系，而这些关联性很难使用传统数据库来描述。这使得图计算在经济建设、国防安全和社会生活等的方方面面都能够发挥重要的作用。

### 2.2图计算相关算法

对于常用的图算法进行总结与归纳, 可以将图算法分为以下两类, 如图2.1所示。

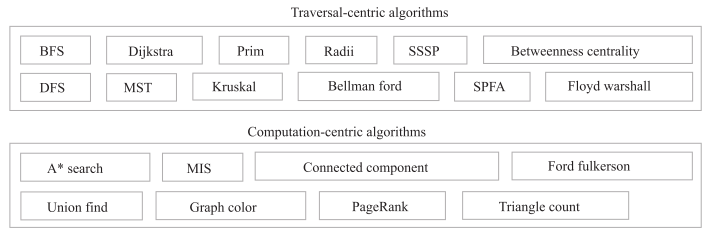


图2.1 图算法分类

1. 遍历为中心的图算法. 该类算法通常需要以特定方式从特定顶点遍历图, 它们的计算内存访问率相对较低, 存在大量随机访问, 大大浪费了存储器带宽。比较经典的算法如广度优先遍历 (breadthfirst traversal, BFS)、单源最短路径 (single source shortest path, SSSP)、中介中心度 (betweenness centrality, BC) 等都属于这种算法。
2. 计算为中心的图算法. 该类算法在一个迭代周期中有大量的运算进行, 在每一次迭代中所有的顶点都会参与. 相比于遍历为中心的图算法, 这类算法的数据局部性较好, 但是在运算过程中会出现大量的浮点运算, 而且每轮迭代所有顶点的参与会增加算法运算过程中对内存的访问频率. 经典的网页排序 (PageRank)、连通分量 (connected component, CC)、三角形计数 (triangle count, TC) 等都属于这种算法。

## 并行图计算编程模型

为了简单高效得实现并行图计算，目前主流图计算系统采用四种编程模型：以点为中心的编程模型、以边为中心的编程模型和以子图为中心的编程模型。采用合适的编程模型，能够提高计算过程中的并行度，从而可以采用多线程或分布式的方式执行图计算程序。

### 3.1以点为中心的编程模型

由于图数据结构内部存在复杂的依赖关系，图计算系统存在不易扩展和难以并行化的问题。为了解决这些问题，需要提出一种适合图计算的编程模型作为系统设计的基础。Google 在2010年发表的Pregel [1]计算框架所采用的TLAV（Think Like a Vertex）思想，正是针对图计算的特点所采用的一种新式编程模型。该模型把计算范围限定在图数据中的单个点，即从单个点的角度考虑图算法的执行过程，包括每个点上的计算以及相邻点之间的消息传递。这样各个点可以实现相互独立的计算，从而进行细粒度的并行。

如图3.1所示，最早的TLAV是建立在BSP（Bulk Synchronous Parallel）模型之上,即把计算过程分割为多个超级步，超级步之间通过屏障（Barrier）来保证信息被同时传送和接受，从而避免了数据竞争和死锁。但以点为中心的模型并不局限于上述的同步执行模型，还支持异步执行模型。与同步执行模型不同的是，异步执行模型不需要屏障来限定超级步之间的信息传输，即每个点的更新值可以在当前的超级步中立即可见，从而加速算法的收敛速度。但是，异步执行模型需要考虑数据竞争、一致性等问题。

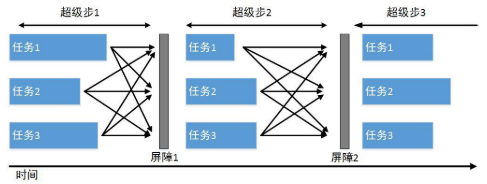


图3.1 BSP模型

以顶点为中心的计算模型可以很方便地表示各种图算法并且可以实现顶点的高并行, 所以这种编程模型已被广泛用于许多图加速器[2∼4]。然而, 以该模型实现的图计算系统需要对顶点数据和边数据进行大量的随机访存操作，这将导致较大的时间开销。由于现代计算机体系结构的特点，随机访存往往比顺序访存的性能低。因此如何采用顺序访存进行图计算的问题受到业界广大的关注。同时，以点为中心的编程模型需要对原始的图数据进行排序等预处理操作，这也增大总的时间开销。

### 3.2以边为中心的编程模型

如前所述，以点为中心的编程模型存在大量的随机访存和排序预处理等问题，为了解决这些问题，以边为中心的模型被提出[5]。

以边为中心的模型不再采用随机访存的方式来访问边数据，而是采用顺序访存的方式。X-Stream[5]是第一个使用以边为中心的模型，即GAS模型来处理每条边, 主要通过3个步骤迭代执行：(1)收集其源顶点的信息 (gather); (2)更新 (apply); (3)将更新后的值发送到各目标顶点 (scatter)。通过试验证明以边为中心模型运行算法比采用以点为中心模型运行算法在处理速度上提高了2到3倍。同时，以边为中心的模型不需要对原始的图数据进行排序等预处理操作，从而降低了时间开销。

现有的图加速器通常使用以边为中心的模型来提高其有限内存带宽的利用率[6, 7]。但与以顶点为中心的编程模型相比, 以边为中心的模型缺乏灵活的调度能力, 同时该模型还可能导致对顶点的大量随机访问。因此, 通常需要额外的优化, 例如细粒度分区和定制的顶点更新策略[6, 8]。

### 3.3以子图为中心的编程模型

以点或边为中心的编程模型是一种细粒度的编程模型，由于其状态信息在每个超级步内只能被传播一跳，导致某些特殊的算法不能有效的执行。针对细粒度编程模型的局限性，一种以子图为中心（Subgraph-Centric，也称为以块为中心Block-Centric）的粗粒度编程模型被提出[9]，且子图是由相互关联性较强的图顶点构成。

以子图为中心的模型先把图数据划分为不同的子图，然后更新子图中所有的点直到子图收敛，最后把子图更新的状态信息传送到其它子图。这种模型可以加快算法收敛的速度，但该模型存在一些挑战，比如对图数据进行子图划分等问题。

## 图计算硬件并行加速技术

加速器因具备丰富的带宽资源、超高的并行能力及极低的数据传输延迟等技术优势，是实现高效图计算的重要技术手段之一。按照加速器物理器件性质来看，现有的面向图计算的硬件加速方案大致可分为基于 GPU 的图计算加速器、基于 MIC 的图计算加速器、以及基于 FPGA的图计算加速器等。

### 4.1基于 GPU 的图计算加速技术

与CPU相比，GPU因集成众多计算单元，可提供更强的并行计算能力，以更高效地支撑大规模图顶点遍历与更新。图4.1描述了NVIDIA Tesla P100[10]的结构图。每个GPU内部包含多个流处理器（Streaming Multiprocessor），流处理器共享全局存储单元，同时每个流处理器都有各自的私有缓存。流处理器内部含有多个线程单元，连续固定长度（一般为 32）线程为一个调度基本单位，通常称为一个线程warp，warp内部线程执SIMD（Single Instruction Multiple Data）计算模型。每个流处理器上可以同时执行多个线程 warp，连续一定数量 warp 称为一个CTA（Compute Thread Array）,GPU 对流处理器进行任务分发的基本粒度为CTA，同一个CTA 内部可以通过共享存储器进行通信。一方面，通过多级粒度的任务并发，GPU 可以实现超高的计算并行度，同时具备不同粒度的任务调度。另一方面，GPU内部有多个内存控制器，通过并发线程可以满足高带宽的要求。例如Tesla P100具有3584个处理单元，其内部带宽可以高达732GB/s。

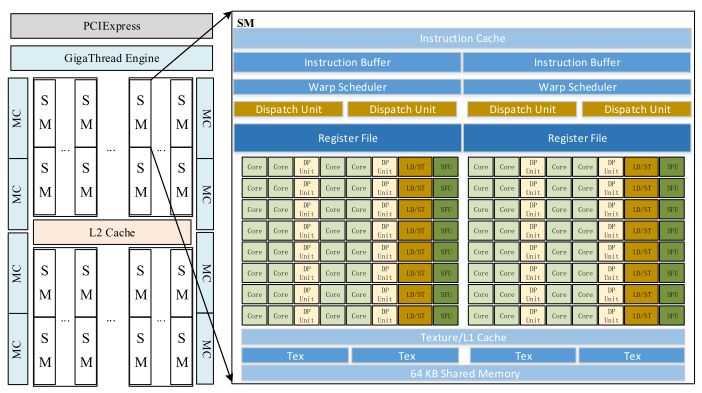


图4.1 NVIDIA Tesla P100结构图

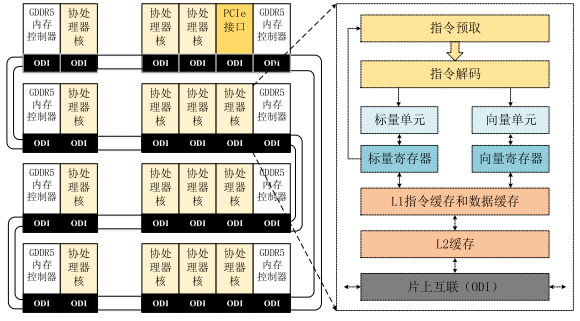
图计算具有高并发的特点，无论是vertex-centric或者edge-centric编程模型都隐藏着大量数据并行语义，可以使用GPU进行并行加速，且图计算是一种数据密集型应用，GPU 高达上百 GB/s 的带宽相对于CPU也具有明显的优势，因此单位时间内可传输的数据更多，并充分利用GPU单指令多数据（Single Instruction Multiple Data，SIMD并行优势进行加速。

Gunrock[11]设计了一种基于GPU的图计算通用加速库，评测结果显示，相较于基于 CPU 的经典图处理框架，Gunrock 在性能上可高出至少一个数量级。通过 Gunrock 设计的通用API，开发人员可以基于GPU实现更加复杂的图算法。除此之外，Gunrock设计并封装了基于GPU的常用图优化技术。例如，针对负载均衡问题，采用了类似于B40C的线程细粒度调度和基于warp/CTA的混合调度策略，以实现不同粒度间的计算任务负载均衡。针对访存带宽问题，采用了CSR和Edge list的混合数据结构提高聚合访存的效率，对点操作采用CSR结构，对边操作采用Edge list访问，最大程度上减少了乱序访存带来的额外开销。

为了进一步提升GPU带宽利用率，Graphie[12]针对大规模图数据的处理调度进行了优化。为了解决GPU在访问顶点相应点数据时，顶点数据不连续所导致的带宽资源浪费问题，Graphie 提出了一种重编码图结构，将无出度的顶点编号放置编号集的尾部，这样可保证所有源点集合访问的连续性，从而提高图计算的聚合访存效率。Graphie支持GPU的大图处理，将边表分为固定大小的数据块，将活跃点数据块传输到GPU进行处理，为方便记录每个数据块中是否存在活跃节点，通过插入虚拟顶点，Graphie保证每个数据块起点的连续性且范围长度相同。

### 4.2基于 MIC 的图计算加速技术

Intel Xen Phi[13]（简称 MIC 众核处理器）的结构如图4.2所示，与GPU结构类似，MIC 构架是一种支持一致性内存访问的对称多处理器结构。MIC 的多个处理核之间通过环形双向总线进行核间通信。每个处理核配置相应容量的一级缓存和二级缓存，同时每个处理核拥有一个标量处理单元和一个向量处理单元，向量处理单元拥有长度为512位的SIMD处理能力。MIC构架的优势在于，能提供与GPU 类似的 SIMD 性能加速能力的同时，仍可较好地支持OpenMP、MPI、Intel TBB、Cilk等多核处理器的编程环境，具备很好的代码可移植性。



由于 MIC 构架与 GPU构架同是基于 SIMD 技术进行加速，且均具有很高的带宽，因此其面临的问题与 GPU类似。与GPU 不同的是，MIC 所采用的是 X86 指令集，这意味着能够运行在 Intel CPU平台的代码也能运行在 MIC 平台上，具有天然的异构计算优势。

MOSAIC[14]是基于CPU+MIC异构平台实现的核外图处理框架，其采用的设计思路是将图数据切分成固定大小（称为 tile）,然后每个tile内部的数据在MIC协处理器端进行局部加速归约，对归约得到的结果在 CPU端进行全局归约，同时协处理器的计算任务和主机端任务能较好的流水化，充分利用二者的计算能力。

### 4.3基于 FPGA 的图计算加速技术

FPGA作为一种介于通用芯片（如CPU，GPU）与定制化芯片（ASIC）之间的计算平台，一方面可以提供大量的计算资源以保证较高的并行度，另一方面也提供了较好的可重构性以保证较低的能源损耗。因此，大量研究人员与机构开始尝试使用 FPGA解决传统架构中存在的问题。

由于图应用的数据局部性差和访存随机性大的特点，因而其访存延迟通常明显大于传统应用。针对图计算访存延迟高的问题，现有基于FPGA的图计算工作在执行模型和数据划分等方面开展了大量工作。

在执行模型方面，由于传统的Vertex-centric模型[1]局部性较差，因此现有的 FPGA工作多数基于Edge-centric模型[5]以提升边数据访问的局部性。在数据划分方面，传统的划分方法以等分方式所有边数据以同等计算量为尺度进行划分。这种方法可以一定程度上保证较好的负载均衡，却加剧了各划分内点数据访问的随机性。因此现有工作通常采用网格划分（grid-partition）的方式提升点数据访问的局部性[15]。

通过在执行模型与数据划分等方面的优化技术，现有的FPGA工作有效缓解了图计算在传统架构下访存延迟过高的问题。而针对并行度不足的问题，现有工作通常通过在片上提供大量的处理单元的方式以利用图计算中潜在的并行性[15]。

## 总结

本文对图计算领域的一些并行计算方法进行了调研和讨论，主要对并行图计算编程模型和硬件并行加速技术进行了比较详细的阐述，对不同的模型和方案进行了相互的比较，列举了各自的优缺点以及一些领域内的工作。

## 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]. Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. 2010:135-146.
2. Ham T J, Wu L, Sundaram N, et al. Graphicionado: a high-performance and energyecient accelerator for graph analytics. In: Proceedings of the 49th Annual IEEE/ACM International Symposium on Microarchitecture, Taipei, 2016. 1–13
3. Ozdal M M, Yesil S, Kim T, et al. Energy ecient architecture for graph analyticsaccelerators. In: Proceedings of the 23rd ACM/IEEE Annual International Symposium on Computer Architecture, Seoul, 2016. 166–177
4. Dai G H, Chi Y Z, Wang Y, et al. FPGP: graph processing framework on FPGA a case study of breadth-first search. In: Proceedings of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, 2016. 105–110
5. Roy A, Mihailovic I, Zwaenepoel W. X-stream: Edge-centric graph processing using streaming partitions. Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. 2013: 472-488.
6. Dai G, Huang T, Chi Y, et al. Fore-graph: exploring large-scale graph processing on multi-FPGA architecture. In: Proceedings of the 25th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, 2017. 217–226
7. Zhou S J, Prasanna V K. Accelerating graph analytics on CPU-FPGA heterogeneous platform. In: Proceedings of the 29th International Symposium on Computer Architecture and High Performance Computing, Campinas, 2017. 137–144
8. Song L, Zhuo Y, Qian X, et al. GraphR: acceleratinggraph processing using ReRAM. In: Proceedings of the 24th IEEE International Symposium on High-Performance Computer Architecture, Vienna, 2018. 531–543
9. Tian Y, Balmin A, Corsten S A, et al. From think like a vertex to think like a graph. Proceedings of the VLDB Endowment, 2013, 7(3): 193-204.
10. Pascal Architecture Whitepaper – Nvidia. [https://images.nvidia.com/content/pdf/tesla/whit epaper/pascal-architecture-whitepaper.pdf](https://images.nvidia.com/content/pdf/tesla/whit%20epaper/pascal-architecture-whitepaper.pdf)
11. Wang, Yangzihao, et al. "Gunrock: A high-performance graph processing library on the GPU." Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 2015: 265-266.
12. Han W, Mawhirter D, Wu B, et al. Graphie: Large-Scale Asynchronous Graph Traversals on Just a GPU. Proceedings of the International Conference on Parallel Architectures and Compilation Techniques. 2017:233-245.
13. Jeffers, James, James Reinders, Avinash Sodani. Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition. Morgan Kaufmann, 2016.
14. Maass S, Min C, et al. Mosaic: Processing a Trillion-Edge Graph on a Single Machine. Proceedings of the Twelfth European Conference on Computer Systems. 2017:527-543.
15. G. Dai, T. Huang, Y. Chi, et al. “ForeGraph: Exploring Large-Scale Graph Processing on Multi-FPGA Architecture,” Proceedings of ACM International Symposium on Field-Programmable Gate Arrays, 2017:217-226