平台和负载特征感知的在线图分割算法

陆 李 华 蓓

(中国科学技术大学计算机科学与技术学院 合肥 230027)

摘 要 分布式图计算在许多领域有着广泛的应用,图分割是分布式图计算的基础.已有分割算法大多只考虑图的简单拓扑特性,它们将图计算系统视为同构系统,或最多考虑 CPU 计算能力及通信带宽的不同.然而,目前包含 GPU 的异构计算系统已经越来越普遍,由于 GPU 独特的并行计算架构和并行计算模式,不考虑 GPU 计算特点的图分割算法不能获得异构环境下最优的分割方案.本文通过分析及实验发现,计算负载特性对于估算处理节点的图计算时间有很大的帮助.在此基础上,本文提出度变异系数和分片通达度两个负载特征参数,给出了通过数据集采样和离线测试获取负载特征参数到处理器负载计算时间的映射关系的实用方法,并结合以上工作实现了一个平台特性和负载特征感知的在线图分割算法.在真实图数据集上的测试表明,相比于工业界和学术界领先的图分割算法,本文提出的方法可获得最优的图分割方案,可令图计算系统的整体执行时间减少 50%~70%.

关键词 图计算;图分割;GPU;异构系统;负载特征挖掘

中图法分类号 TP391 **DOI 号** 10.11897/SP.J.1016.2020.01230

A Platform-and-Workload Aware Online Graph Partitioning Algorithm

LU Li HUA Bei

(*School of Computer Science and Technology, University of Science and Technology of China, Hefei* 230027)

Abstract Graph is an important data structure for describing and storing relationships between objects, and has been widely used in many fields such as community analyzing, scientific computing and urban planning. To deal with massive graph data, modern graph data processing systems are usually distributed and highly heterogeneous, where different kinds of processing nodes collaboratively conduct the computing tasks. Among them, GPUs are the most common co-processers which can effectively offload workload from CPUs and accelerate the whole system. Graph partitioning is the basis of distributed graph processing. To assign a big graph to multiple computing nodes, graph data is usually partitioned online, during data stream loading. The quality of a graph partition greatly affects load balancing and communication overhead among compute node, and thus becomes an important factor in system performance. Most of the existing graph partitioning solutions are based on traditional k-balanced partition problem, whose goal is to minimize the number of inter-partition edges while maintaining an even distribution of vertices. These works only exploit simple topological properties of the graph, with the assumption of homogeneous computing systems. In fact, Graph partitioning algorithms under heterogeneous environments should take into account the information mining from hardware platform and graph data workload. Some existing works towards heterogeneous system consider the distributed system as computing nodes with different computing and communication power. However, these works only use simple quantification methods such as estimating the computing power with hardware parameters, or measuring the computing and communication power with customized

收稿日期:2019-05-06;在线发布日期:2019-12-12.陆 李,博士研究生,主要研究方向为大数据处理和图计算. E-mail: luly9527@mail.ustc.edu.cn. 华 蓓,博士,教授,博士生导师,中国计算机学会(CCF)会员,主要研究领域为高性能计算、边缘计算、虚拟化等.

microbenchmarks, which is not suitable for modern popular CPU-GPU heterogeneous distributed graph processing system. This is because that GPU is a parallel processor which hardware architecture and programming model is quite different from CPU, therefore its computing power depend on the degree of parallelism available from graph data workload and cannot be quantified by the simple methods mentioned above. Moreover, both graph topology and graph computing tasks are highly irregular. In some applications, the number of vertices involved are very different in each iteration, therefore the actual workload of a partition is hard to determine and cannot be quantified simply by the size of a single partition. In this paper, we focus on CPU-GPU heterogeneous distributed graph processing system. Through careful analysis and experiments, we find that certain workload characteristics have a significant impact on computing power of GPU and workload in actual execution, which are helpful for estimating the graph processing time of a node. On this basis, this paper proposes two workload characteristics, named coefficient variation of degree and accessibility of a partition. Coefficient variation of degree affect computing power of GPU and accessibility of a partition can be used to estimate actual workload. We devise a practical method to obtain the mapping between workload characteristics and workload processing time on a node by means of sampling and test running. Combining the above work, this paper implements a platform-and-workload aware online graph partitioning algorithm. Experiments on four real graph datasets show that compared with the leading graph partitioning algorithms in industry and academia, the proposed algorithm achieves the best partitioning and can reduce the overall graph computing time by 50%～70%.

Keywords　graph computing; graph partitioning; GPU; heterogeneous systems; workload characteristic mining

1　引　言

图是描述现实世界中对象及其关联关系的重要数据结构，由于表达能力丰富而在科学计算、社交网络、文本检索等众多领域有着广泛的应用. 随着移动互联网、物联网、Web2.0 等信息技术的发展，图数据规模呈爆炸式增长，例如万维网(Web)已经拥有超过 200 亿个网页(顶点)和 1600 亿个超级链接(边). 这些图规模巨大、分析算法复杂，已远远超过了单台计算机存储和处理的能力. 分布式计算是解决数据爆炸的重要手段，通过将大图数据分割后分配到多个计算节点上，各节点协同工作来完成整个计算任务.

图分割是分布式图计算的基础，图数据划分是否合理对图计算系统的性能影响巨大. 图分割算法的根本目标是最小化计算任务的总耗时. 为此分割算法通常遵循两个重要原则：一是划分后各子图大小均匀，以使各节点负载均衡；二是各子图间的连通性最小，以最小化网络通信开销. 经典的图分割问题在数学上被描述为一个平衡 k 路图分割问题[1]，它要求将图分割成 k 个顶点数量大致相等的子图，并使得连接不同子图的边(以下称为交叉边)数量最小. 严格的平衡 k 路图分割已被证明是一个 NP 难问题[2]，在实际应用中往往采用启发式方法求解.

已有的图分割算法大致分为离线分割和在线分割两类. 离线分割以当前业界的标杆——METIS① 软件工具为代表. METIS 由明尼苏达大学开发和维护，随着不断更新和优化，已成为当下最优的图分割解决方案之一. METIS 从图的一个随机初始划分开始，通过在子图之间不断交换顶点和边来优化分割. 为了获得较优的分割方案，需要利用图的全局信息并进行很多次迭代，当图规模较大时计算开销很大. 在线分割方案在加载图数据的过程中完成图的分割，假定图数据以节点流或边流的形式到达，分割算法通过一组启发式规则来决定当前到来的顶点或边的放置位置，顶点和边一经放置就不再移动. 启发式规则可以只是简单的哈希运算[3-5]，也可以设计较复

① METIS. http://glaros.dtc.umn.edu/gkhome/metis/metis/faq

杂的目标函数[6]. 在线分割方案可以无缝运行在图数据的加载过程中,并天然支持增长型数据,更适合当前存储和计算分离的图计算环境.

当前大部分的图分割方案基于平衡 k 路图分割问题,这意味着它们假设图计算环境是同构的. 然而,计算环境异构在当前的计算机集群或数据中心已经是非常普遍的现象. 比如,计算节点可能是不同规格型号的多核处理器或协处理器(GPU、FPGA、MIC 等),不同节点间的通信带宽也可能不同,这使得经典问题中的平衡分割方案在实际中并不是最佳的. 已有一些工作关注异构环境中的图分割问题,比如,文献[7]为传统分割算法添加了带宽感知特性,可以根据节点间不均衡的通信能力计算分割方案;文献[8]考虑由多核处理器构成的分布式计算环境,可以根据每个节点的计算能力和每一对节点间的通信带宽设计分割方案. PowerGraph[9] 从一个初始分割开始,在图计算过程中监视各个节点的执行耗时,然后通过将数据从高负载节点迁移至低负载节点来完成对分割的动态调整和现场优化. 在带协处理器的异构环境中,目前工作大多关注 CPU 与协处理器的协同及分布式计算方面的问题,在图分割问题上仅沿用多核异构系统的分割方法. 比如,G2[10] 是分布式 CPU-GPU 异构图计算系统的代表. 对于分割策略,G2 先用 Surfer[11] 分割工具(METIS 的改进版本)将大图划分为多块,然后通过运行若干常见的图算法(例如 BFS 和 PageRank 算法)来测得各节点的执行时间,并按照执行时间的比例为各节点分配数据块.

为平衡各节点的计算时间,图分割算法通常需要在分割前先获取一些先验知识,据此对各个节点的工作负载和处理能力进行精确量化,最终衡量和预测节点的执行时间. 节点的任务负载通常根据分片的顶点数目和边的数量决定,而节点的计算能力则可以通过以下三种方式获得:根据用户预先输入,根据硬件主频和内存容量,以及根据自定义测试基准程序. 例如,METIS 允许用户指定节点计算能力和节点间通信带宽,并基于这些信息计算分割方案. 文献[8]通过运行两种微基准程序(micro benchmark)来估算节点计算能力和任意一对节点间的通信能力,其中节点计算能力是通过让节点执行一千万次的浮点数乘法来估计的.

显然,在异构环境中,先验知识准确与否是影响分割质量最关键的因素. 然而现有工作用来衡量和预测节点执行时间的方式对于图计算系统中最常见的协处理器 GPU 来说完全不适用. 首先,传统方式并不能精确衡量 GPU 节点的计算能力. 这是因为 GPU 采用了适合大规模数据并行的计算架构和单指令多线程执行模式,其计算能力的释放完全依赖于有足够数量的、可被高度并行化、且高度平衡的计算负载,而这与计算负载的特性及程序编写是否高效有很大的关系,因而不能简单地用主频、计算核数量、每秒浮点运算次数等固定的硬件参数来衡量 GPU 的实际运行性能,更不能用这些参数与 CPU 的计算能力(主频、处理核数量、每秒浮点运算次数)进行简单比较. 其次,使用子图规模来衡量节点的工作负载也存在较大偏差,相同规模的子图在实际运行时可能产生不同的计算负载. 这是因为图计算过程本质上由一系列迭代过程组成,在每一轮迭代中计算节点只对本轮的活跃顶点进行计算和通信,并产生下一轮活跃顶点,因而每一轮工作负载与该轮活跃顶点的数量有关. 像深度优先搜索、广度优先搜索等图遍历算法在每一轮迭代中只有部分顶点活跃,下一轮活跃顶点的位置及数量取决于当前活跃顶点的邻接边数量、邻居顶点的位置等负载特性,因而仅以子图规模来衡量工作负载大小也是不够准确的.

综上所述,在目前最常见的由 CPU 和 GPU 构成的图计算系统中,工作负载特性可能会从两个方面影响节点计算时间,一是影响 GPU 节点的计算效率,二是影响节点的实际工作负载. 已有的图分割算法均未考虑这两个因素,它们假设 GPU 的计算效率在任何数据上均相同,且节点工作负载仅由子图的规模决定,这使得这些算法对节点计算时间的估计很不准确,从而不能得到理想的分割方案.

目前,针对异构系统缺乏理想的图分割算法的问题,一般有两种应对方法. 一是根据硬件参数直接调整分配比例,让高性能的节点获得较多的负载[10,12];二是将有偏差的分割方案作为初始划分,在执行图计算的过程中再进行重分割[9]. 第一种方法需要依赖经验知识,缺乏普适性和可靠度. 第二种方法需要在计算过程中进行相关测量和统计,并通过节点间交换数据来调整分割,不仅引入额外的资源消耗,而且会导致计算延迟或暂停. 另外,目前使用较多的图数据处理系统,如 Pregel[3]、PEGASUS[4] 和 GraphLab[5],均采用在数据加载时就对图数据进行划分的在线分割方式,它们在进行分割决策时没有运行数据可用.

本文针对 CPU-GPU 异构系统研究平台和负载

特征感知的在线图分割算法.本文通过实验的方法观察和研究图计算应用的负载特性对 GPU 计算效率及节点实际工作负载的影响,提取出规律,并应用到一个面向 CPU-GPU 异构计算环境的在线图分割算法中.负载特征感知解决了传统方式无法准确预测 GPU 节点的执行时间的难题,本文是首次针对 CPU-GPU 分布式异构系统提出平台和负载特征感知的分割算法.本文的主要贡献如下:

(1)基于对分布式图计算过程及 GPU 并行计算特点的分析,提出两个新的有助于准确估算节点计算时间的负载特征参数,并通过实验验证了它们与节点计算时间之间存在强相关性.

(2)给出一种基于数据集采样和测试运行的先验知识学习方法,用于获取负载特征参数与节点计算时间之间的映射关系.实验表明,这种映射关系与硬件平台和图算法有关,而与特定数据集无关,可以作为准确、可靠的先验知识.

(3)设计和实现了一个面向 CPU-GPU 异构系统的在线图分割算法,并与目前工业界和学术界领先的图分割算法进行比较.在由 8 个节点组成的分布式异构平台上,利用 4 个真实图数据集和 4 种常见图算法的测试表明,本文提出的在线分割算法性能最优,可减少图计算时间 50%～70%,且算法具有良好的 GPU 亲和性和可扩放性.

本文第 2 节介绍论文相关工作和国内外研究现状;第 3 节介绍和分析异构系统中的在线图分割问题;第 4 节提出两种影响执行耗时的特征参数;第 5 节实现一个完整的平台和负载特征感知的在线图分割算法;第 6 节对提出的算法进行实验评测;第 7 节总结全文.

2　相关工作

经典的图分割问题定义为:给定一个无向图 $G=(V,E)$,其中 V 为顶点集合,E 为边集合,要求计算 G 的一个 k 路分割 $P=\{P_i\mid i=1,2,\cdots,k\}=\{(V_i,E_i)\mid i=1,2,\cdots,k\}$,其中 $\bigcup V_i=V$,$\bigcap V_i=\varnothing$,$E_i=\{(v,u)\mid v\in V_i,u\in V\}$,使得 $\big||V_i|-|V_j|\big|<\lambda(i,j=1,2,\cdots,k)$($\lambda$ 为预设的阈值),以及 $|E_c|$ 最小,$E_c=\bigcup E_c(i,j)=\{(v,u)\mid v\in V_i,u\in V_j,i,j=1,2,\cdots,k,i\neq j\}$ 称为交叉边集合.每个 $P_i=(V_i,E_i)$ 称为图 G 的一个分片.

严格的平衡 k 路图分割是一个 NP 难问题,实际中多采用近似算法或启发式算法求解,如文献[1]提出算法复杂度为 $O(\log n)$ 的近似算法,最新版本的 METIS 分割工具吸收采纳文献[13]中的启发式算法并引入并行执行支持和多约束条件控制,是目前使用最广泛的离线图分割工具.近年来基于启发式方法的在线图分割算法也备受关注,Pregel、PEGASUS、GraphLab 使用最简单的哈希算法来分配每个到来的顶点,LDG[14] 提出了一系列基于不同启发式规则的图分割算法,FENNEL[7] 则为这些离线分割算法建立了数学模型和统一框架,文献[15]在 FENNEL 的基础上进一步改进分割质量,文献[16]实现了基于 FENNEL 的流式再分割算法.这些离线流式分割算法简单高效,可以在图数据加载时就完成分割,是各大分布式图处理系统采用的基本算法,也是后来更复杂划分算法的基础,然而它们均着眼于同构的计算环境,未考虑不同节点在计算能力和通信能力上可能存在的差异.

面对客观存在的异构计算环境,一些工作开始关注异构图计算系统中的图分割问题.这些工作将图计算系统抽象成由计算节点和节点间的通信线路构成的网络,并对节点计算能力和通信能力进行量化.最新的 METIS 软件工具支持用户预先设定各个节点的处理能力和节点间通信带宽,并根据这些参数计算分割方案.文献[8]通过运行一千万次浮点数乘法来估计节点的计算能力,并通过在节点间传输大小为 100 KB 的文件来测量每一对节点之间的通信能力,进而根据这些先验知识来指导分割.文献[17]改进了 GraphLab 框架默认的同构分割算法,通过硬件参数估计各个节点的吞吐率,再以此为依据为常用的 5 种在线分割算法添加异构感知能力.在这类算法中,对节点计算能力和实际工作负载的估算准确度直接关系到图分割质量的好坏.

上述工作仅着眼于多核计算环境,另有一部分工作将 GPU 引入到图数据处理中,并开发了运行在 GPU 上的图算法库和图处理框架[18-20].在此基础上,文献[21-24]在配备了 GPU 的单机上实现了异构处理器协同的图处理系统.这些工作采用以 GPU 为中心的负载分配原则,先给 GPU 分配可高度并行化的计算负载,而让 CPU 执行辅助工作及处理剩下的不适合 GPU 处理的负载,该方法的缺点是没有充分发挥所有节点的计算能力.文献[10]是当前分布式 CPU-GPU 异构系统的代表工作,文中明确指出在异构系统中传统算法得到的分割方案距离理想方案有较大的偏差,因此其采用了自定义测试程序来尝试纠正偏差.近年来(2016 年以来)的

图分割工作主要着眼于加快大图分割效率[25-27]，以及在特定场合进一步改进分割质量[28-29]. 据我们所知，针对 GPU-CPU 异构场景下的在线图分割算法目前还是空白，这正是本文的研究重点.

3 问题分析

3.1 在线图分割问题

在线分割是在从文件存储系统加载图数据到分布式图处理系统的过程中完成数据分配的，且顶点一旦被分配后就不再移动. 图数据集通常以顶点流的形式读入，即依次读入顶点和它的所有邻接点和邻接边的信息，因此在线分割也称为流式分割. 在线分割问题定义为：假设 $\boldsymbol{P}$ 为当前时刻的一个分割结果，V_i 为当前时刻分片 P_i 的顶点集合，v 为新读入的一个顶点，$f(\boldsymbol{P})$ 为评估分割质量的代价函数，要求将 v 分配给某个分片 P_i 形成新的分割 $\boldsymbol{P}'$，即 $V_i'=V_i\cup\{v\}$，使得在所有 k 种分配方案中 $f(\boldsymbol{P}')$ 最小.

大多数分布式图处理系统采用 BSP（Bulk Synchronized Processing）计算框架和以顶点为中心的计算方式. 在每一轮迭代中，每个分片内部先对活跃的顶点执行计算，然后分片之间交换信息，再开始下一轮迭代. 对于每一个活跃顶点，首先处理其收到的消息，完成顶点值的更新，然后按照邻接边向其邻居顶点发送新的消息.

图分割方案的最终目标是最小化图计算任务的执行时间，因此可将 $f(\boldsymbol{P})$ 定义为图计算任务在 $\boldsymbol{P}$ 上的执行时间. 由于 BSP 模型按轮次执行，在每一轮中各节点先在各自的分片上执行计算，全局同步后再开始下一轮计算，因此每轮的执行时间相当于该轮中最慢节点的执行时间. 假设一个图计算任务总共执行 m 轮，节点 i 处理分片 P_i，它在第 j 轮的任务耗时（不包括等待其它节点的时间）记为 $ETime(i,j)$，则 $f(\boldsymbol{P})$ 可表示为

$$f(\boldsymbol{P})=\sum_{j=1,\cdots,m}\max_{i=1,\cdots,k}\{ETime(i,j)\} \tag{1}$$

由于每个分片每轮的任务耗时都不尽相同，为方便表达和计算，我们用 $avgETime_i$ 表示分片 P_i 的平均每轮任务耗时，并将式(1)近似表示为

$$f(\boldsymbol{P})=\max_{i=1,\cdots,k}\{avgETime_i\}*m \tag{2}$$

在分割图时并不知道将来执行计算时的总轮数，即 m 未知，但由于计算过程全程同步进行，任何时刻各个分片的当前轮次和总轮次均相同. 这样，我们的目标就可以变化为最小化平均每轮执行时间，即将代价函数定义为

$$f(\boldsymbol{P})=\max_{i=1,\cdots,k}\{avgETime_i\} \tag{3}$$

根据 BSP 计算模型，分片 P_i 的平均每轮任务耗时 $avgETime_i$ 大致分为两部分，在顶点上执行计算的平均计算耗时 t_i^{comp}，以及分片之间交换数据的平均通信耗时 t_i^{comm}，这两部分通常串行执行，即

$$avgETime_i=t_i^{\text{comp}}+t_i^{\text{comm}} \tag{4}$$

接下来我们需要分别求出平均计算耗时 t_i^{comp} 和平均通信耗时 t_i^{comm}. 考虑到每个顶点上的计算过程相同，为方便根据顶点数估算分片的计算耗时，我们引入单位图计算时间 ut^{comp} 的概念.

定义 1. 单位图计算时间. 分片 P_i 在节点 i 上的单位图计算时间 ut^{comp} 定义为节点 i 在对分片 P_i 执行图计算时，在每一轮迭代中平均花费在每个顶点上的计算时间.

当确定了分片 P_i 在节点 i 上的单位图计算时间 ut^{comp} 后，分片 P_i 在节点 i 上的平均计算耗时 t_i^{comp} 就可以计算为

$$t_i^{\text{comp}}=ut_i^{\text{comp}}*|V_i| \tag{5}$$

在本文中，我们通过实验的方法得到 ut^{comp}. 需要说明的是，由于本文假设分片 P_i 分配给节点 i，在不产生歧义的情况下我们将混用分片 P_i 和节点 i 这两个术语，并且为了表述上的简洁，省略掉单位图计算时间前面的限定词.

对于平均通信耗时 t_i^{comm}，由于消息传递需要串行进行，分片 P_i 的总通信耗时等于 P_i 与其它分片的通信耗时之和. 每一对节点之间的通信耗时可以表示为它们之间交换的数据总量除以他们之间的通信能力. 由于图算法会为两个分片之间的每一条交叉边设计和维护一个数据结构作为消息包，并在每一轮通信时互相交换，因而任一对分片 P_i 和 P_j 之间的单轮通信总量可以用它们之间的交叉边总数 $|E_c(i,j)|$ 乘以沿单条交叉边交换的数据块（消息包）大小（$Size$）来表示，而它们之间的通信能力 C_{ij}^{comm} 可以用处理节点 i 和 j 之间的链路带宽表示. 综上，分片 P_i 上的平均每轮通信耗时可以用如下方式算：

$$t_i^{\text{comm}}=\sum_{j=1,j\neq i}^{k}\frac{|E_c(i,j)|*Size}{C_{ij}^{\text{comm}}} \tag{6}$$

综上可以算得分片 P_i 在处理节点 i 上的平均每轮运行耗时为

$$avgETime_i=|V_i|*ut_i^{\text{comp}}+\sum_{j=1,j\neq i}^{k}\frac{|E_c(i,j)|*Size}{C_{ij}^{\text{comm}}} \tag{7}$$

至此，式(7)将传统分割的两大目标，计算负载平衡和交叉边数量最小化统一到一个公式中. 这是因为计算负载和交叉边数量都最终影响节点的执行

时间，如果我们能够最小化每个节点每轮的执行时间，那么自然就兼顾了负载平衡和通信开销两个方面的因素.

依据式(3)和式(7)，对一个现有的分割 $\boldsymbol{P}$ 和一个新到来的顶点 v，可以估算出 v 分配给每一个分片将产生的平均每轮执行时间，并总是选择可令该时间最小的分配方案.我们根据对现有在线分割算法相关工作的总结，在此介绍一个一般性的启发式算法框架，如算法 1 所示.

算法 1.　在线图分割算法.

输入：vertex stream S

输出：a k-way partitioning $\boldsymbol{P}$

1. $P_1,\cdots,P_k\leftarrow\varnothing$
2. FOR each v in S DO
3. FOR $i=1$ to k DO
4. $P_i'\leftarrow$assign v and its related edges to P_i
5. $P'\leftarrow\{P_1,\cdots,P_i',\cdots,P_k\}$
6. $ETime[i]\leftarrow f(\boldsymbol{P}')$
7. END FOR
8. $index\leftarrow\arg\min_i ETimei]$
9. assign v and its related edges to P_{index}
10. END FOR
11. RETURN $\boldsymbol{P}=\{P_1,\cdots,P_k\}$

算法在第 1 行进行初始化，在第 2 行至第 10 行的双层循环中完成对图数据的划分.在第 3 行至第 7 行的内层循环中，对新输入的一个顶点 v 尝试 k 种分配方案，对每一种分配方案计算平均每轮执行时间(第 6 行).在完成对每一种方案的测试后，选择 $ETime$ 最小的方案作为 v 的最终分配方案(第 8 行)，第 9 行完成顶点 v 及其所有关联边的分配.

算法核心的部分是第 6 行根据式(7)计算每个分片平均每轮的任务耗时，其中单位图计算时间 ut^{comp} 和计算负载及节点计算能力有关.已有工作大多根据处理器的硬件参数或其执行浮点运算的能力来估计其计算能力，但正如前面所分析的，这种估算方法不适用于 GPU.

3.2　负载特性影响 GPU 的计算效率

GPU 有着非常不同于 CPU 的硬件特点和编程环境.在 CPU-GPU 异构环境中，为充分利用 CPU 和 GPU 的计算资源，同一种图算法通常需要在 CPU 和 GPU 上分别做出不同的软件实现.另外，不同型号 GPU 的流处理器个数、显存大小、主频等均不相同，一些软件实现甚至会根据 GPU 的类型自动调整算法代码中的运行参数.这些因素使得单纯利用硬件参数来估算 GPU 的计算能力非常不准确.

GPU 特殊的并行计算架构和编程模型，还使得其实际计算效率与计算负载的数据特征有关.为理解这一点，首先需要了解 GPU 上的并行图计算过程.在常见的以顶点为中心的计算方式中，一个 GPU 线程会负责处理一个或多个活跃的图顶点，线程在执行完顶点上的计算任务后，依次访问该顶点的邻接顶点并生成需要传递给其邻居的消息.GPU 采用单指令多线程(SIMT)的并行计算模式，以一个线程束(32 个线程)作为基本的调度单位，一个线程束内的线程按锁步方式执行.若某个顶点的邻居顶点很多，则处理该顶点的线程将花费较多的时间访问其邻居顶点和传递消息，而其它线程不得不等待.这说明，即使是同样顶点数量和边数量的图数据，如果顶点的度分布越不均匀，则 GPU 线程之间出现负载不均、互相等待的情况就会越严重，致使 GPU 并行计算的优势无法发挥，图处理系统性能也会严重下降.值得注意的是，这个特性并不会出现在串行执行的 CPU 上.

下面通过一个例子说明顶点度的分布特性如何影响 GPU 的计算效率.我们使用同样的 CPU 和 GPU，在规模相同(顶点数和边数相等)、但度分布特性不同(不规则程度依次增加)的三个图上执行 PageRank 算法.这三个图分别是：(1)使用软件生成的顶点度数完全相等的平衡图；(2)维基百科网页链接图；(3)推特社交网络图.以 CPU 在平衡图上的处理速度为单位 1，图 1 给出了 CPU 和 GPU 在三个图上的相对执行速度.由实验结果可知，CPU 在三个图上的执行速度几乎不变，而 GPU 在三个图上的执行速度差异较大.这说明在硬件、算法、负载大小(图规模)均相同的情况下，负载的度分布特性会对 GPU 的计算效率产生影响.这也充分

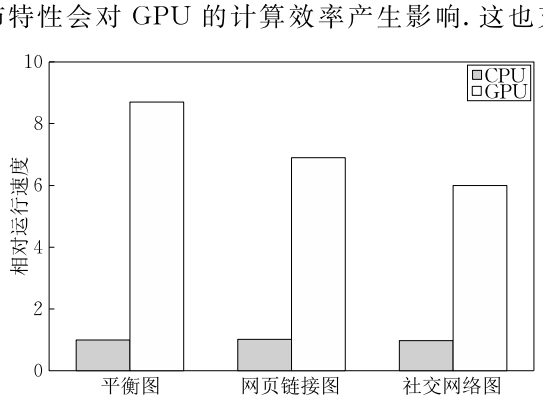


图 1　CPU 和 GPU 在三个图上的相对运行速度

说明,单纯依靠硬件参数或浮点运算能力来估算GPU的实际计算能力是很不准确的.本文4.2节提出使用度变异系数来衡量顶点度分布的不规则性,并研究度变异系数与单位图计算时间的关系.

3.3 负载特性影响分片的实际工作负载

有一部分图算法,如PageRank、多跳邻居搜索算法等,在每一轮迭代中所有顶点都是活跃顶点,在这类算法中分片的工作负载可以用顶点数来衡量.然而仍有相当一部分图算法,如多源BFS算法、模式匹配算法等,在每一轮迭代中只有部分顶点是活跃的.这类算法通常选择一些顶点作为初始点,初始点激活邻居顶点并使其加入计算,整个计算过程不断向外扩散直至达到终止条件.

在部分顶点活跃的图算法中,分片的工作负载难以确定.首先,初始点的分布会影响分片的实际工作负载,比如若所有初始点都在某个分片中,那么其它分片一开始将无事可做,直到有顶点被激活.其次,即使初始点被平均分配到了各个分片中,若某个分片有很多路径通向其它分片,那么该分片中的顶点会有更多的机会被激活,而一些偏僻的分片可能自始至终都只有少量顶点参与到计算中.

由此可见,对于每一轮迭代中只有部分顶点活跃的图算法,即使分片的规模相同,其实际工作负载也不相同,从而其单位图计算时间也不同.本文4.3节提出使用通达度来衡量一个分片与其它分片的联系的紧密程度,并研究分片通达度与单位图计算时间的关系.

4 负载特性与单位图计算时间

4.1 实验准备

本节通过实验的方法,观察并归纳出影响单位图计算时间的两个负载特征参数.本文选择了四种经典的图算法在四个真实的数据集上进行实验,实验采用了不同型号的CPU和GPU(图算法、图数据集、硬件信息可以见6.1节).为减少误差,每种图算法不断重复运行足够长时间(5 min以上),每组实验进行20次并记录下平均值作为结果.

4.2 度变异系数与单位图计算时间

3.2节的分析表明,顶点度的分布特性会影响GPU执行图处理任务的计算效率.本文借鉴概率论和统计学中变异系数(Coefficient of Variation)的概念,使用度变异系数来刻画顶点度分布的不规则程度.

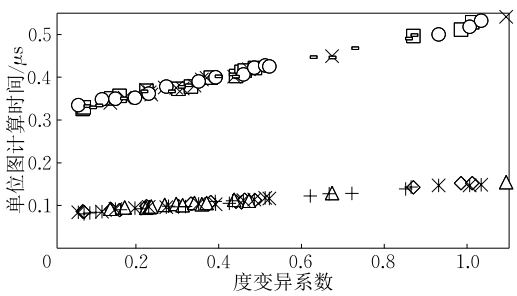
定义2. 度变异系数.图的度变异系数α定义为图的顶点度标准差与图的顶点度平均值之比.若以$SD(i)$和$MN(i)$分别表示分片P_i中顶点度的标准差和平均值,则分片P_i的度变异系数$\alpha_i=SD(i)/MN(i)$.

度变异系数越大,表明顶点度分布的不规则程度越大,则GPU在执行图计算的过程中会更多地出现线程等待的情形,导致单位图计算时间增加.为验证我们的猜想,并观察度变异系数和单位图计算时间之间的规律,我们设计了如下实验.对4个真实的图数据集分别进行采样,计算每个样本图的度变异系数.在2种不同型号的CPU和2种不同型号的GPU上,分别对每个样本图执行PageRank算法和全源最短路径算法,每个处理器计算一个样本,测量并记录每个处理器在每个样本图上的单位图计算时间(执行时间除以样本图的顶点数).

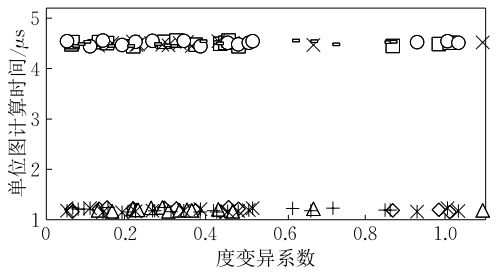
需要特别说明的是,本文采样目的与常见的统计采样并不相同,采集到的样本集并不是为了尽可能无偏地反映和代表原数据集,而是为了测试在不同数据特征下GPU的性能,所以一些不具有代表性的极端的样本集反而可能更有参考价值.采样次数原则上越多越好,但至少需要涵盖实际计算过程中可能遇到的所有参数范围(此处为度变异系数范围).作为初步探索,在具体实验中我们进行了2000次以上的大量采样,目的是尽可能保证每个参数范围内都有样本以供实验分析.对于采样规模,由于在实验评测章节原图需要被分割到4个节点上执行图计算,每个分片规模约为原图的25%,所以采样规模也设定约为原图的25%.

以度变异系数为横轴,单位图计算时间为纵轴,将以上实验数据绘制成散点图,得到如图2(a)、图2(b)所示的结果.为清楚起见,GPU1和GPU2的实验结果显示在图2(a),CPU1和CPU2的实验结果显示在图2(b),相同形状的点为在采自同一个原始图的样本图上得到的实验数据.为防止杂乱,每张散点图中仅显示了每个纵轴区间(此处为度变异系数的一个区间)内具有代表性的数据,其余数据呈现相同的规律.

从图2可以看到,随着度变异系数增大,GPU的单位图计算时间是上升的,而CPU的单位图计算时间基本保持不变,这与3.2节的分析基本一致.从图2还可以看到,对于一个特定的硬件,其单位图计算时间与度变异系数之间的关系与样本图来自哪个图数据集无关.这个实验表明,我们可以利用硬件在某个图数据集的样本上得到的度变异系数与单位



(a) 度变异系数-单位图计算时间散点图(上GPU2/下GPU1)

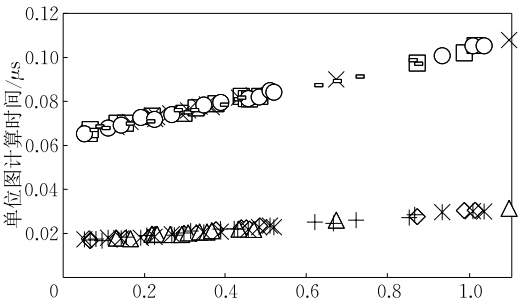


(b)度变异系数-单位图计算时间散点图(上CPU2/下CPU1)

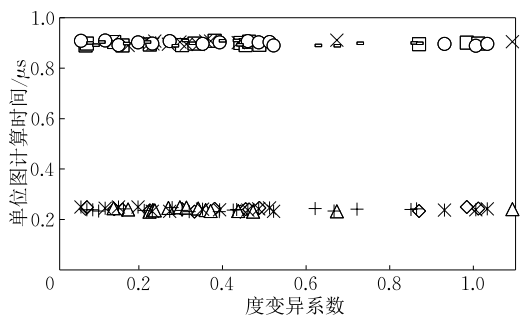
图 2　PageRank 算法下度变异系数-单位图计算时间关系图

图计算时间的关系，去预测该硬件在其它陌生图数据集上的单位图计算时间.

在相同的实验配置下用全源最短路径算法进行测试，得到的结果如图 3 所示.



(a) 度变异系数-单位图计算时间散点图(上GPU2/下GPU1)



(b) 度变异系数-单位图计算时间散点图(上CPU2/下CPU1)

图 3　全源最短路径算法下度变异系数-单位图计算时间关系图

由图 3 可以看到，全源最短路径的单位计算时间比 PageRank 算法要小，这是由算法复杂度和代码实现细节决定的. 另外全源最短路径算法相对 PageRank 算法，GPU 和 CPU 上的单位执行时间对度变异系数仍然呈现相近的规律，即 GPU 上正相关，而 CPU 上基本保持不变. 这说明度变异系数在不同算法下都对 GPU 执行时间有着影响.

4.3　分片通达度与单位图计算时间

对于每一轮中只有部分顶点活跃的图算法，图计算任务通常会给出一些判定条件，用来指出哪些顶点是初始点. 根据这些条件，分割算法在一个顶点到来时可以确定该顶点是否为初始点.

从某个初始顶点开始的计算任务按照如下过程传递负载：初始顶点完成计算后，激活其邻居顶点. 邻居顶点完成计算后，再激活其邻居顶点. 以此类推，最终分片的边界顶点被激活，再通过交叉边激活其它分片中的顶点. 因此，在计算一个分片的单位图计算时间时，还需要考虑从其它分片传递过来的工作负载，本文称其为隐性工作负载.

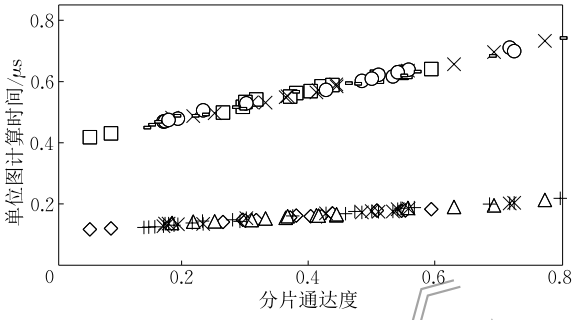
由于隐性工作负载沿着从初始顶点开始的路径传递，对于一个不在 P_i 中的初始顶点 v，v 给 P_i 带来的隐性工作负载与 v 到 P_i 的路径数量有关. 本文使用分片通达度来表示外部初始顶点到某个分片的路径密度.

定义 3.　分片通达度. 分片通达度定义为分片中每个顶点与外部初始顶点之间的平均路径数量. 设 V_o 为初始顶点集合，V_{o-i} 为不在 P_i 上的初始顶点集合，$|path(v,u)|$ 为顶点 v 和 u 之间的路径总数，则 P_i 的通达度 β_i 按照式(8)进行计算：

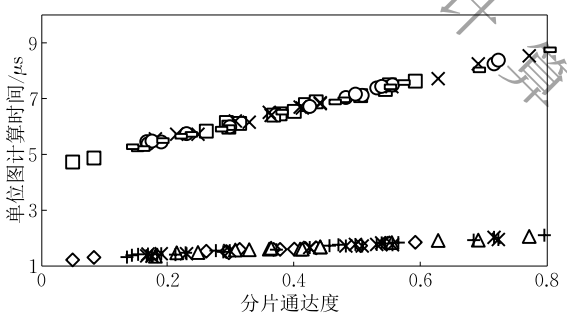
$$\beta_i=\sum_{u\in V_i,v\in V_{o-i}}\frac{|path(v,u)|}{|V_i|} \tag{8}$$

由于隐性负载的存在及差异性，即使两个分片的规模和度分布相同，其单位图计算时间也可能不同. 直观上，隐性负载越大，单位图计算时间越长. 为验证这个猜想，并观察分片通达度与单位图计算时间的关系，我们采用 4.2 节的方法设计了一个类似的实验. 按照 4.2 节的方法对 4 个图数据集进行随机采样，每个样本图的规模约为原图的 50%. 将每个样本图随机分割为大小基本相等的 A、B 两个分片，以 A 分片作为观察和研究的对象，计算出 A 分片的度变异系数. 在 4.1 节给出的 4 种硬件上，分别对每个样本图执行多源 BFS 算法(B 分片运行在同一台计算机的另一个处理器硬件上)，每一次随机选取 2%的点作为初始点，计算 A 分片的分片通达度和单位图计算时间. 以分片通达度为横轴，单位图计算时间为纵轴，将实验数据绘制成散点图. 由于单位

图计算时间还与度变异系数有关，为屏蔽度变异系数的影响，本文将在度变异系数相似的样本图上得到的数据放在一起显示，图 4 中的数据是在度变异系数为 0.4 附近的样本图上得到的. 为清楚起见，GPU 的数据绘制在图 4(a)中，CPU 的数据绘制在图 4(b)中，相同形状的点为在来自同一个原始图的样本图上得到的数据. 为防止杂乱，每张散点图中仅显示了每个区间具有代表性的数据，其余数据呈现相同的规律.



(a) 分片通达度-单位图计算时间散点图(上GPU2/下GPU1)



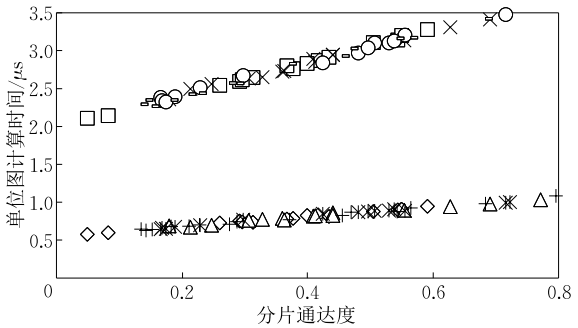
(b) 分片通达度-单位图计算时间散点图(上CPU2/下CPU1)

图 4 多源 BFS 下分片通达度-单位图计算时间关系图

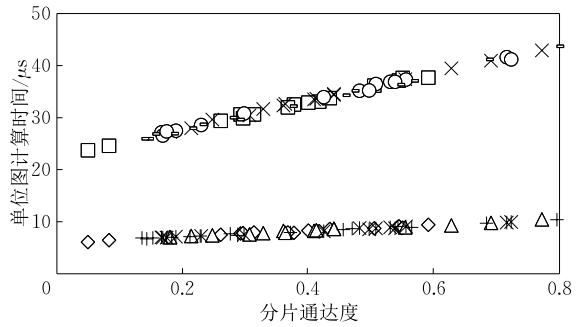
从图 4 可以看到，随着分片通达度的上升，CPU 和 GPU 的单位图计算时间都是上升的. 并且，对于一个特定的硬件，其单位图计算时间与分片通达度的关系与样本图来自哪个图数据集无关. 这说明，我们利用硬件在某个图数据集的样本上得到的分片通达度-单位图计算时间关系，也适用于该硬件和其它图数据集的组合.

在相同的实验配置下用图路径匹配算法进行测试，得到的结果如图 5 所示.

可以看到，图路径匹配算法下的图计算时间比多源 BFS 算法长很多，这是因为前者较为复杂，需要较多计算和访存. 另外图路径匹配算法相对多源 BFS 算法，GPU 和 CPU 上的单位执行时间对分片通达度仍然呈现相近的规律，即 GPU 和 CPU 上都呈现正相关. 这说明分片通达度在不同算法下都对执行时间有着影响.



(a) 分片通达度-单位图计算时间散点图(上GPU2/下GPU1)



(b) 分片通达度-单位图计算时间散点图(上CPU2/下CPU1)

图 5 图路径匹配算法下分片通达度-单位图计算时间关系图

5 基于平台和负载特征的在线图分割

5.1 获取先验知识

采用本文提出的图分割算法，需要事先根据计算平台和所要运行的图算法做一些采样测试工作，获得分割图时所需的先验知识.

共有 3 类先验知识需要获取，同时作为分割算法的输入：首先要获得单条交叉边上的通信消息数据块大小，这是根据算法设计中的消息数据结构和网络协议封装格式等决定的. 设一轮计算中的总通信量为 V，发生通信的交叉边数量为 $|E_{cc}|$，则通信消息数据块大小为 $V/|E_{cc}|$. 其次需要测试各个节点之间的带宽，这可以通过在不同节点之间传递测试文件，设测试文件大小为 S，传输时间为 t，则带宽为 S/t. 最后需要根据第 4 节的描述获得负载特征参数到单位图计算时间的拟合函数.

获取拟合函数的过程中，最重要的目的是要采样获得不同参数特征的样本图用于学习，为此可将预估的参数范围进行等间隔划分，对采样得到的样本图进行参数特征计算，并按所属区间进行统计，当每个参数区间均有代表性样本时采样结束. 需要注意的是，该过程只是对图数据进行采样，并不实际执行图计算，且参数区间的大小可以调整，因此该采样

过程的开销是可控的.

在进行探究性实验时，我们是按照实际运行的分片规模对数据集进行采样的.为获取性价比较优的采样规模，本文设计了一组实验来观察在不同规模的样本图上学习到的参数特征—单位图计算时间关系.以在 GPU1 上使用数据集 Amazon0601 执行 PageRank 算法为例，将采样规模分别设为原先采样规模的 50%，25%，10%和 1%，绘制度变异系数和单位计算时间图的关系，如图 6 所示.

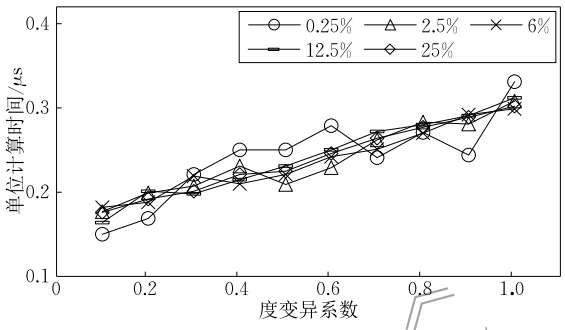


图 6　在不同采样规模下学习到的度变异系数-单位图计算时间关系图

可以看到采样规模为原先的 50%，25%和 10%时(即原图的 12.5%，6%，2.5%)时，呈现的规律与 100%采样规模时基本相同，1%(即原图的 0.25%)时有一定的偏差.故在实验中我们采取如下规则，以实际运行分片规模的 10%采样，若最终分片数目为 N，原图规模为 $|G|$，最终采样规模则为 $|G|/N\times 10\%$.

若所要运行的图算法在每一轮迭代中会激活全部顶点，需要针对每一种处理器节点学习其单位图计算时间 ut^{comp} 与度变异系数 α 的关系($ut^{\text{comp}}\sim\alpha$).若图算法在每一轮迭代中只激活部分顶点，需要针对每一种处理器节点学习其 ut^{comp} 与度变异系数 α 和分片通达度 β 的关系($ut^{\text{comp}}\sim(\alpha,\beta)$).需要注意的是，以上采样和测试工作对于处理器-图算法的每一种组合只需做一次，当该组合用于处理其它图数据集时，已获得的先验知识可以直接使用.

由于采样过程中得到的数据都是一些散点数据，当需要估计某个分片 P_i 的单位图计算时间时，需要进行数据拟合.本文中我们利用滑动窗口平均的方法进行拟合.对于每一轮迭代中所有顶点都活跃的图算法，首先计算 P_i 的度变异系数 α_i，然后选取与 α_i 最近的 5 个样本点，计算它们的函数平均值作为单位计算时间.对于每一轮迭代中只有部分顶点活跃的图算法，首先计算 P_i 的度变异系数 α_i 和分片通达度 β_i，然后选取几何空间中最靠近(α_i，β_i)的 5 个样本点，计算它们的函数平均值作为单位图计算时间.

至此通过获取先验知识，我们可以得到在任何特征参数下的单位图计算时间 ut^{comp}.

5.2　在线分割算法的实现

算法 2 是在算法 1 的框架下对算法所做的细化.算法的输入为：顶点流 S，通过先验知识获得的负载特征参数与平均计算时间的拟合函数 f，单条交叉边上的通信数据块大小 $Size$，以及所有节点两两之间的链路带宽.算法输出图分割结果.

算法 2.　平台和负载特征感知的在线图分割算法.

输入：vertex stream S, fitting function $f(\alpha,\beta)$, message size $Size$, link bandwidth C^{comm}

输出：a k-way partitioning $\boldsymbol{P}$

1. $P_1,\cdots,P_k\leftarrow\varnothing$
2. $V[k][k],\alpha[k][k],\beta[k][k],Ec[k][k][k]\leftarrow 0$
3. FOR each v in S DO
4.　FOR $i=1$ to k DO
5.　FOR $j=1$ to k DO
6.　$P_i'\leftarrow$ assign v and its related edges to P_i
7.　calculate $v[i][j],\alpha[i][j]$ on P_i'
8.　calculate $\beta[i][j],E_c[i][j][l]$ on $\boldsymbol{P}'=\{P_1,\cdots,P_i',\cdots,P_k\}$
9.　$ut^{\text{comp}}[i][j]\leftarrow f(\alpha[i][j],\beta[i][j])$
10.　$ETime[i][j]\leftarrow V[i][j]*ut^{\text{comp}}[i][j]+\sum_{l=1,l\neq j}^{k}\frac{E_c[i][j][l]*Size}{C_{jl}^{\text{comm}}}$
11.　END FOR
12.　END FOR
13.　$index\leftarrow\arg\min_i\{\max_j\{ETime[i][j]\}\}$
14.　assign v and its related edges to P_{index}
15. END FOR
16. RETURN $\boldsymbol{P}=\{P_1,\cdots,P_k\}$

对于新读入的一个顶点 v，流式分割算法通过将其模拟分配给每一个分片来形成 k 个尝试性方案，算法使用二元组(i，j)来表示第 i 号尝试性方案(将顶点 v 分配给 P_i)中的第 j 号分片.第 2 行使用 $V[i][j]$来保存分片的顶点数，$\alpha[i][j]$和 $\beta[i][j]$分别保存分片的度变异系数和分片通达度，$Ec[i][j][l]$保存该分片与其它各个分片(用 l 表示)的交叉边数量.第 4 行至第 12 行的两层循环中，外层循环遍历所有 k 个尝试性方案，内层循环遍历某个尝试性方案的所有 k 个分片，第 7，8 行更新分片的所有参数，第 9 行利用特征参数和先验知识计算分片的

单位图计算时间,第 10 行根据式(7)计算分片的平均每轮任务耗时 $ETime[i][j]$. 结束循环后,第 13 行取每一个尝试性方案下最大的 $ETime[i][j]$作为该方案的代价,并选择代价最小的尝试性方案作为最终的分配方案(接收顶点 v 的分片序号),第 14 行完成对顶点 v 及其所有关联边的分配.

5.3 实现问题

流式分割是典型的启发式贪心算法,只能根据当前情形求得局部最优解,较为依赖初始解. 在理论上,在算法刚开始时由于信息不足,算法可能被迫进行随机分配. 为防止初始分割质量不佳影响最终结果,我们曾经尝试将最开始读入的部分的顶点作为一个整体,利用 METIS 工具进行分割,并将分割后的结果作为初始解,在此基础上再进行后续分割. 然而,经实验发现,初始解对最终结果的影响效果可以忽略不计.

与采用简单启发式规则的在线分割算法相比,本文算法引入了一定的计算开销,主要包括各个分片的度变异系数和分片通达度的计算,以及根据负载特征参数计算各个分片的单位图计算时间等. 然而,各个分片的度变异系数和分片通达度可以通过增量计算获得,不需要每次都重新计算. 另外,由于图数据大多从磁盘或其它外存储系统读入,在线分割是在数据读入的过程中同步进行;与磁盘的读写速度相比,本算法引入的计算时间可以忽略,不会降低数据的加载速度.

6 实验评测

6.1 实验设定

为评估本文提出的在线图分割算法,我们使用了 4 个真实的图数据集,分别是 Amazon0601、Wiki-Talk、Live-Journal 和 cit-Patents. 这些图数据集均可从网上免费获得①,其详细信息见表 1 和表 2. 实验时,这些图均转化成了无向图.

表 1 实验使用的 4 个图数据集

数据集名称	顶点数	边数	描述
Amazon0601	403394	3387388	亚马逊购物记录图
Wiki-Talk	2394385	5021410	维基聊天网络图
Live-Journal	4847571	68993773	交友网站社交网络图
cit-Patents	3774768	16518948	专利相互引用关系图

这 4 个数据集是从大量公开的数据集中通过分析对比遴选出来的,原则是保证数据集尽可能具有不同的结构特征,并能涵盖和代表真实世界常见的数据集. 常见的反映图数据结构特征的参数有:稀疏度,反映图的边数与点数的比例;偏移度,反映图中度分布的均匀性;连通度,反映图中连通分片或低连通分片的数量. 本文选择的 4 个图数据集的结构特征如表 2 所示,其中"高"、"中等"、"低"的评价是相对于我们收集到的候选数据集而言的.

表 2 4 个图数据集的结构特征

数据集名称	描述	疏密	偏移度	连通度
Amazon0601	数据库记录	中等	低	中等
Wiki-Talk	社群	疏	低	高
Live-Journal	社交网	密	高	低
cit-Patents	标签关联	中等	中等	高

本文使用了 4 种经典的图算法,用于寻找规律和评估图分割算法,详细信息见表 3. 对于每轮迭代中全部顶点活跃的图算法,使用度变异系数作为其负载特征;对于只有部分顶点活跃的图算法,采用度变异系数和分片通达度作为其负载特征.

表 3 实验使用的 4 个图算法

算法名称	描述	所有点参与计算?
PageRank	根据顶点度计算顶点重要度	是
全源最短路径	计算图中任意两点距离	是
多源 BFS	从初始点开始访问所有顶点	否
路径匹配	从初始点开始寻找匹配路径	否

本文使用四台计算机来模拟异构计算环境,每台计算机都包含一个多核 CPU 和一块 GPU 显卡,将它们看成 8 个计算节点. 这 8 个计算节点的硬件参数均不相同,详细信息如表 4 所示. 6.2 节~6.4 节使用 2 台计算机(CPU1+GPU1,CPU2+GPU2)进行实验,6.5 节增加两台计算机(CPU3+GPU3,CPU4+GPU4)进行扩放性实验.

表 4 实验使用的 8 个计算节点

节点名称	型号	计算核数	主频/GHz
CPU1	Intel I7-7600K	8 core	4.0
CPU2	Intel E7-7200	2 core	2.53
CPU3	Intel I5-2537	2 core	1.4
CPU4	Intel I7-4500u	2 core	1.8
GPU1	Nvidia G1080	2560 SP	1.6
GPU2	Nvidia C1060	448 SP	1.15
GPU3	Nvidia G750	512 SP	1.02
GPU4	Nvidia G650	384 SP	1.06

我们使用以下两个图分割工作作为比较对象:第一个是工业界质量最优的离线分割软件工具 METIS. 近年多数分割工作均以 METIS 作为对比

① Snap Dataset. http://snap.stanford.edu/data/index.html, 2018, 9, 20

基准[25-28]. 第二个是文献[8]提出的异构环境感知的在线分割算法(以下简称 HEA_Alg)，是学术界目前针对异构环境效果最好的在线分割算法. 文献[8]基于不同的贪心策略共设计了 MW、MI、BMI、CB、CPH、CMH 六种方案，本文采用其论文中报告效果最好的 CB 方案进行比较. 本文提出的平台和负载特性感知的在线分割算法简称 P&W_Alg.

我们按照第 5 节描述的方法，首先在上述 8 个计算节点上，利用数据集 Amazon0601 学习不同图算法下负载特征参数与单位图计算时间的对应关系，然后利用这些先验知识对 4 个图数据集进行在线分割.

另外为减少误差，任意实验都运行足够长时间(5 min 以上)，并进行 20 次，记录下平均值作为结果.

我们共设计了四组实验. 第一组比较 METIS、HEA_Alg 和 P&W_Alg 算法的分割效果，即分割后图计算任务的完成时间；第二组分析本文提出的两种负载特征对图分割质量的影响；第三组验证本文提出的图分割算法对于 GPU 架构是友好的；第四组在更多节点下检验算法的可扩放性.

6.2　三种算法分割效果评估

我们在由 2 台计算机(4 个计算节点)构成的图计算平台上，分别采用 METIS、HEA_Alg 和 P&W_Alg 算法对 4 个图数据集进行在线分割，在分割完成的图上执行 PageRank 算法、全源最短路径算法、多源 BFS 算法和图路径匹配算法，测量每一次图计算任务的完成时间. 对于多源 BFS 算法，随机选取图中 2%的节点作为初始顶点. 图 7、图 8、图 9 和图 10 是

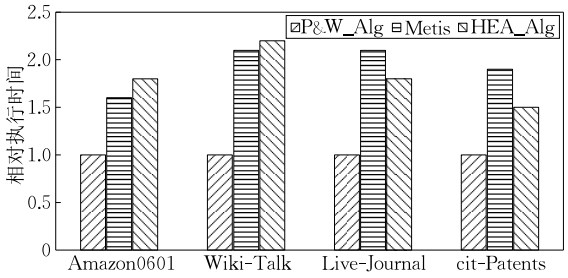


图 7　采用不同图分割算法后 PageRank 的执行时间

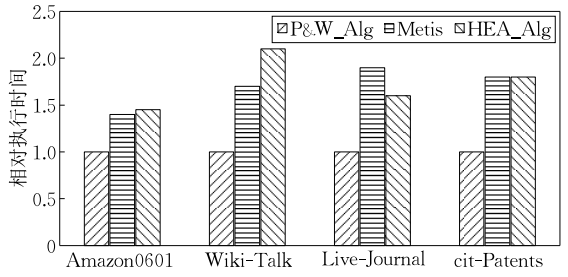


图 8　采用不同图分割算法后全源最短路径的执行时间

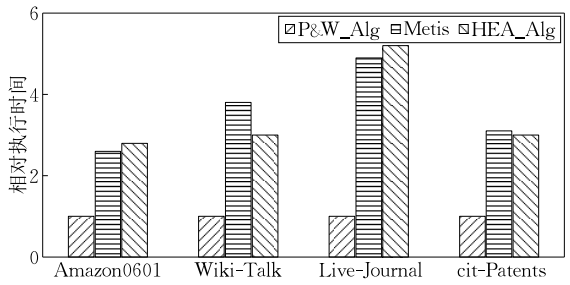


图 9　采用不同图分割算法后多源 BFS 的执行时间

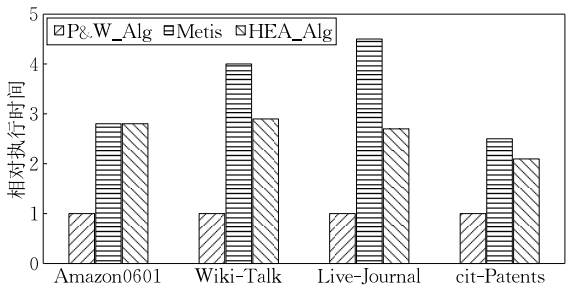


图 10　采用不同图分割算法后图路径匹配的执行时间

在 4 个数据集上，采用 3 种图分割算法完成图的分割后，运行 4 种图算法所用的时间，在每个图数据集上以 P&W_Alg 分割后的算法执行时间作为单位时间 1.

可以看出，对于所有的图算法-图数据集组合，P&W_Alg 算法的分割效果最好，即在其生成的分割上运行图算法的时间最短. 与 METIS 和 HEA_Alg 生成的分割方案相比，P&W_Alg 生成的分割方案可使 PageRank 算法的执行时间平均缩短近 50%，可使多源 BFS 算法的执行时间至少缩短 70%. 为分析性能改进的原因，我们以在数据集 Amazon0601 上运行 PageRank 算法为例，测量了在执行 1000 次迭代的过程中每个计算节点的任务总耗时(即不包括等待其它节点的时间)，结果如图 11 所示.

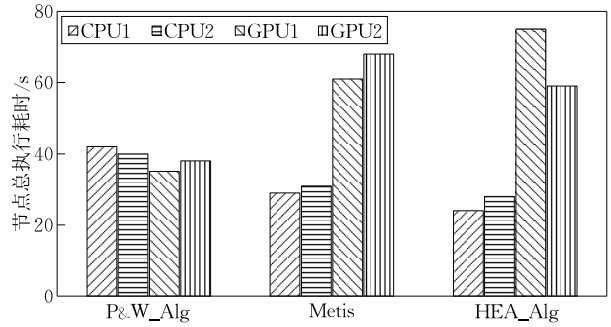


图 11　3 种分割算法下的 4 个节点上的总执行时间

可以看到，在 P&W_Alg 生成的分割方案上，4 个计算节点的任务耗时差异最小，表明该分割方案的负载均衡性最好；而在 METIS 和 HEA_Alg

生成的分割方案上，GPU 节点的任务耗时远大于 CPU 的任务耗时，说明 CPU 在计算过程中等待时间较长，其计算资源没有得到充分利用. 这说明 METIS 和 HEA_Alg 对于图计算场景下 GPU 的实际计算能力以及图计算负载对于节点计算时间的影响估计不准，导致对节点计算时间的判断产生较大的偏差，最终导致得到的图分割方案负载不均衡. 以上实验结果表明，本文提出的在线分割算法在异构计算环境下优于已有算法.

考虑到上述实验仅在数据集 Amazon0601 上采样和获取了先验知识，为加强验证，我们分别又在另外三个数据集上获取了先验知识并进行测试. 测试结果呈现与图 7 至图 10 类似的优异结果. 这说明，在已有数据集上学习到的先验知识迁移到陌生数据集上也是可行的.

6.3 负载特征对于分割质量的影响

对于多源 BFS 和图路径匹配算法，我们同时使用了两个负载特征来指导分割，本实验分析这两个负载特征对图分割效果的贡献. 本实验运行三个分割算法：P&W_Alg 算法、只使用度变异数的 P&W_Alg 算法（用 P&W_α 表示）和只使用分片通达度的 P&W_Alg 算法（用 P&W_β 表示）. 在 4 个图数据集上分别采用这三个算法进行图分割，每次随机选取图中 2%的节点作为初始顶点，测量每一次图计算任务的完成时间. 实验结果显示在图 12 和图 13 中，在每个图数据集上以 P&W_Alg 分割后的多源 BFS 算法或路径匹配算法执行时间作为单位时间.

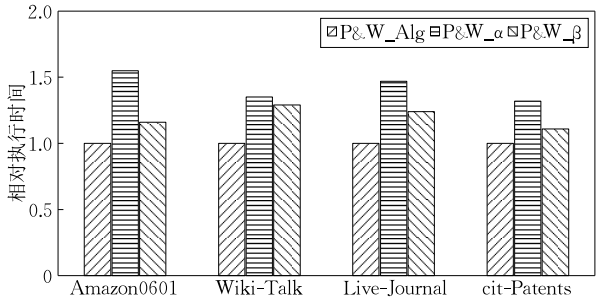


图 12 基于不同负载特征分割图后多源 BFS 的执行时间

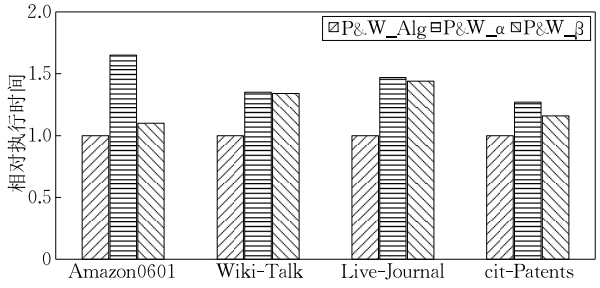


图 13 基于不同负载特征分割图后路径匹配的执行时间

从图 12 和图 13 可以看到，两个负载特征均对优化图的分割质量有贡献，其中分片通达度的贡献更大一些. 这是因为这类算法在执行过程中只有部分顶点活跃，这使得整个分片的度变异系数不能很好地反映这部分顶点的度分布特性.

6.4 图分割算法对于 GPU 的友好性

本实验测试三种图分割算法对于 GPU 的友好性. 我们在 4 个图数据集上分别使用 METIS、HEA_Alg 和 P&W_Alg 生成图分割，并执行 4 种算法，记录并统计每个 GPU 在图计算过程中的实际运行时间（不包含 GPU 的等待时间），用 GPU 上的计算负载（用实际参与计算的顶点总数表示）除以 GPU 实际运行时间，得到 GPU 的平均吞吐率. 实验结果如图 14 和图 15 所示，在每个图数据集上，以在 P&W_Alg 算法分割后的图上测量得到的 GPU 吞吐率作为单位吞吐率.

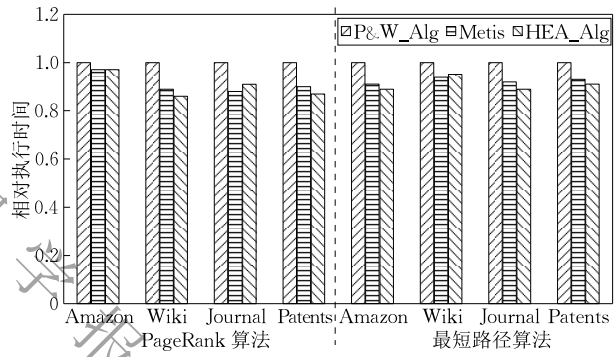


图 14 采用不同分割算法后 GPU 执行图算法的吞吐率（PageRank 和最短路径算法）

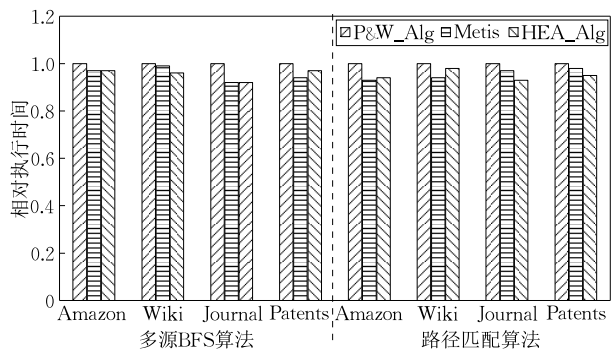


图 15 采用不同分割算法后 GPU 执行图算法的吞吐率（多源 BFS 算法和路径匹配算法）

可以看到，对于所有的图算法—图数据集组合，GPU 在由 P&W_Alg 算法分割得到的图上运行的吞吐率最高，与 METIS 和 HEA_Alg 算法相比，吞吐率平均提升约 8%. 这是因为 P&W_Alg 算法总是寻找可令任务耗时最短的解，在节点分配上倾向于将节点分配给最适合对其进行处理的节点. 以度

变异系数为例，P&W_Alg 算法在分配节点时倾向于降低 GPU 上的度变异系数，这将使得 GPU 上的各顶点的度数接近，从而使得各个 GPU 线程的计算负载接近，这样可以减少线程等待时间，提高 GPU 计算效率. 实验结果表明本文提出的图分割算法对于 GPU 更友好.

6.5　图分割算法的可扩放性

本实验测试本文提出的图分割算法的可扩放性. 我们将 CPU1 和 GPU1 作为 2 个节点的分布式系统，加入 CPU2 和 GPU2 作为 4 个节点的分布式系统，再加入 CPU3 和 GPU3 作为 6 个节点的分布式系统，最后加入 CPU4 和 GPU4 作为 8 个节点的分布式系统. 以数据集 Amazon0601 为例，分别采用 METIS、HEA_Alg 和 P&W_Alg 算法进行在线分割，分别分割成 2 块、4 块、6 块和 8 块. 在分割结果上分别运行 4 种算法，测量每一次图计算任务的完成时间，取 20 次实验结果的平均值. 实验结果显示在图 16 和图 17 中，对于每一种图算法，以 2 节点系统在 P&W_Alg 算法分割得到的图上执行算法的总时间为单位 1.

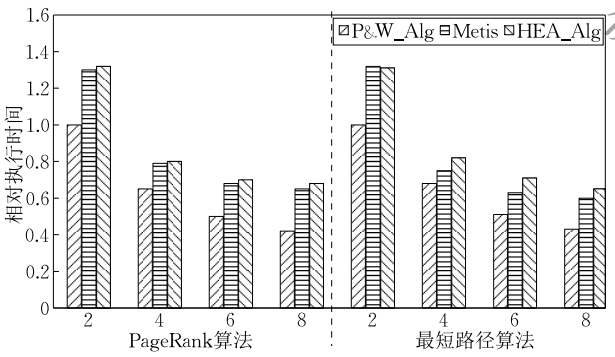


图 16　4 种分割算法分割成不同数量分片后的执行时间（PageRank 和最短路径算法）

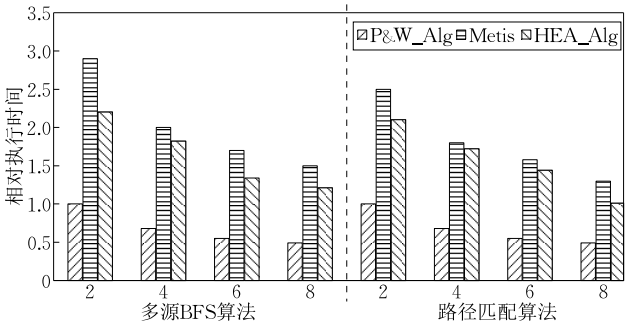


图 17　4 种分割算法分割成不同数量分片后的执行时间（多源 BFS 算法和路径匹配算法）

可以看到，P&W_Alg 在所有情形下的分割效果都是最佳的，从 2 个分片到 8 个分片 P&W_Alg 均可以使图计算任务的总体执行时间减少 50%至 70%. 值得一提的是，对于多源 BFS 算法和路径匹配算法（部分顶点活跃的图算法），利用已有分割算法进行图分割并采用 8 个节点进行计算，其执行时间比利用本文算法进行图分割并采用 2 个节点进行计算的时间还长，这说明本文算法在预测部分顶点活跃的图计算任务的执行时间方面更为准确.

细心的读者会发现，使用 8 个节点的执行时间仅为使用 2 个节点的执行时间的 50%，并未呈现线性下降的趋势. 这里有两个原因，一是第 3、第 4 台计算机中的 CPU 和 GPU 相比于第 1、第 2 台计算机中的 CPU 和 GPU 性能差很多，无法承担太多的计算任务；二是增加计算节点虽然可以分担负载，但是分片增多也增加了通信开销，抵消了增加节点带来的一些好处.

7　结束语

本文针对由 CPU 和 GPU 组成的异构图计算系统研究平台和负载特征感知的在线图分割问题. 基于对分布式图计算过程及 GPU 并行计算特点的分析，本文指出已有图分割算法采用的简单启发式规则没有充分反映处理节点及计算负载的特性，从而不能生成在实际运行过程中负载平衡的图分割. 本文提出两个有助于准确估算处理节点运行时间的负载特征参数，度变异系数和分片通达度，并给出基于这两个特征参数的在线图分割算法. 本文同时给出了利用图数据集采样和处理器测试运行来学习负载特征到处理器单位图计算时间映射关系的实用方法. 在真实图数据集上的实验表明，与当前工业界和学术界领先的图分割算法相比，本文算法的分割效果最好，可减少图计算时间 50%～70%.

参　考　文　献

[1] Andreev K, Racke H. Balanced graph partitioning. Theory of Computing System, 2006, 39(6): 929-939

[2] Garey M R, Johnson D S, Stockmeyer L. Some simplified NP-complete problems//Proceedings of the 6th Annual ACM Symposium on Theory of Computing. Seattle, USA, 1974: 47-63

[3] Malewicz G, Austern M H, Bik A J C, et al. Pregel: A system for large-scale graph processing//Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. Indianapolis, USA, 2010: 135-146

[4] Deelman E, Singh G, Su M H, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. Scientific Programming, 2005, 13(3): 219-237

[5] Low Y, Gonzalez J E, Kyrola A, et al. GraphLab: A new framework for parallel machine learning. arXiv preprint, arXiv:1408.2041, 2014

[6] Tsourakakis C, Gkantsidis C, Radunovic B, et al. Fennel: Streaming graph partitioning for massive scale graphs//Proceedings of the 7th ACM International Conference on Web Search and Data Mining. New York, USA, 2014: 333-342

[7] Meyerhenke H, Sanders P, Schulz C. Parallel graph partitioning for complex networks. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(9): 2625-2638

[8] Xu N, Cui B, Chen L, et al. Heterogeneous environment aware streaming graph partitioning. IEEE Transactions on Knowledge and Data Engineering, 2014, 27(6): 1560-1572

[9] Gonzalez J E, Low Y, Gu H, et al. PowerGraph: Distributed graph-parallel computation on natural graphs//Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation. Hollywood, USA, 2012: 17-30

[10] Zhong J, He B. Towards GPU-accelerated large-scale graph processing in the cloud//Proceedings of the 2013 IEEE 5th International Conference on Cloud Computing Technology and Science. Bristol, UK, 2013: 9-16

[11] Chen R, Yang M, Weng X, et al. Improving large graph processing on partitioned graphs in the cloud//Proceedings of the 3rd ACM Symposium on Cloud Computing. San Jose, USA, 2012: 3

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

[13] Schloegel K, Karypis G, Kumar V. Parallel static and dynamic multi-constraint graph partitioning. Concurrency and Computation: Practice and Experience, 2002, 14(3): 219-240

[14] Stanton I, Kliot G. Streaming graph partitioning for large distributed graphs//Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Beijing, China, 2012: 1222-1230

[15] Tsourakakis C. Streaming graph partitioning in the planted partition model//Proceedings of the 2015 ACM on Conference on Online Social Networks. Palo Alto, USA, 2015: 27-35

[16] Nishimura J, Ugander J. Restreaming graph partitioning: Simple versatile algorithms for advanced balancing//Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Chicago, USA, 2013: 1106-1114

[17] LeBeane M, Song S, Panda R, et al. Data partitioning strategies for graph workloads on heterogeneous clusters//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Austin, USA, 2015: 1-12

[18] Harish P, Narayanan P J. Accelerating large graph algorithms on the GPU using CUDA//Proceedings of the International Conference on High-Performance Computing. Goa, India, 2007: 197-208

[19] Zhong J, He B. Medusa: Simplified graph processing on GPUs. IEEE Transactions on Parallel and Distributed Systems, 2013, 25(6): 1543-1552

[20] Wang Y, Davidson A, Pan Y, 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. Barcelona, Spain, 2016, Article 11

[21] Zhang T, Zhang J, Shu W, et al. Efficient graph computation on hybrid CPU and GPU systems. The Journal of Supercomputing, 2015, 71(4): 1563-1586

[22] Ma L, Yang Z, Chen H, et al. Garaph: Efficient GPU-accelerated graph processing on a single machine with balanced replication//Proceedings of the USENIX Annual Technical Conference 2017. Santa Clara, USA, 2017: 195-207

[23] Shi X, Luo X, Liang J, et al. Frog: Asynchronous graph processing on GPU with hybrid coloring model. IEEE Transactions on Knowledge and Data Engineering, 2017, 30(1): 29-42

[24] Sengupta D, Song S L, Agarwal K, et al. GraphReduce: Processing large-scale graphs on accelerator-based systems//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Austin, USA, 2015: 1-12

[25] Simpson T, Pasadakis D, Kourounis D, et al. Balanced graph partition refinement using the graph p-Laplacian//Proceedings of the Platform for Advanced Scientific Computing Conference. Basel, Switzerland, 2018: 8

[26] Huang X, Shen Z. ACTDP: An adaptive chunk tool for database partition in shared-nothing distributed database//Proceedings of the 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS). Beijing, China, 2018: 903-906

[27] Liu X, Zhou Y, Guan X, et al. A feasible graph partition framework for parallel computing of big graph. Knowledge-Based Systems, 2017, 134: 228-239

[28] Zheng A, Labrinidis A, Pisciuneri P H, et al. PARAGON: Parallel architecture-aware graph partition refinement algorithm//Proceedings of the 19th International Conference on Extending Database Technology. Bordeaux, France, 2016: 365-376

[29] Zhang M, Zhuo Y, Wang C, et al. GraphP: Reducing communication for PIM-based graph processing with efficient data partition//Proceedings of the 2018 IEEE International Symposium on High Performance Computer Architecture. Vienna, Austria, 2018: 544-557

LU Li, Ph.D. candidate. His research interests include big data processing and graph computing.

HUA Bei, Ph. D., professor. Her research interests include high performance computing, edge computing, virtualization, etc.

Background

Graph computing plays an important role in data mining. Modern graph computing systems are usually distributed and highly heterogeneous, where CPUs and co-processers (such as GPUs) collaboratively conduct the computing tasks. Graph partitioning is the technique that divides a big graph into several subgraphs and assigns each subgraph to a compute node. An optimal partitioning is achieved if the workloads are balanced among computing nodes and the cross-node communication is minimized.

Most of the graph partitioning algorithms try to solve the classical k-balanced partitioning problem, which supposes the graph computing system is homogeneous and whose goal is to minimize the number of inter-partition edges while maintaining an even distribution of vertices. Some works have paid attention to heterogeneous systems where the compute nodes are considered to have different computing and communication power. However, they give very simple quantification methods, such as estimating the computing power with processor parameters, or measuring the computing and communication power with customized microbenchmarks. A common measure of computing power is the number of floating-point multiplications per second.

However, none of the above quantification methods is suitable for CPU-GPU heterogeneous systems. GPU has quite different hardware architecture and programming model from CPU, whose computing power can only be stimulated by highly parallelizable and regular computing tasks, therefore its runtime performance cannot be characterized by static hardware parameters or offline benchmarking. Moreover, graph computing is known as a typical irregular task, not only the graph topology is highly irregular, but also the number of vertices involved in each iteration may differ greatly, therefore the real workload of a partition cannot be quantified simply by the number of vertices.

This paper proposes a platform-and-workload aware online graph partitioning algorithm that takes into account GPU's hardware/software architecture and the irregularity of graph computing. The main contribution of this paper is that it explores and defines two new feature parameters that well capture the irregularity of graph computing workloads, and designs a practical method to obtain the relationship between feature parameters and node's computing power. The proposed algorithm achieves the best partitioning and can reduce the overall graph computing time by 50%～70%. This paper is the first work that targets at high efficient graph partitioning towards CPU-GPU heterogeneous systems.