计算机科学与探索
Journal of Frontiers of Computer Science and Technology
1673-9418/2023/17(07)-1576-10
doi: 10.3778/j.issn.1673-9418.2111062

面向大图子图匹配的多GPU编程模型

李岑浩[1],崔鹏杰[1],袁　野[2+],王国仁[2]
1. 东北大学 计算机科学与工程学院,沈阳 110000
2. 北京理工大学 计算机学院,北京 100081
+ 通信作者 E-mail: yuanye@mail.neu.edu.cn

摘　要:子图匹配是复杂网络中进行数据挖掘的重要手段。近年来,基于图形处理器(GPU)的子图匹配算法已展现明显的速度优势。然而,由于大图数据的规模宏大以及子图匹配的大量中间结果,单块GPU的内存容量很快成为了处理大图子图匹配算法的主要瓶颈。因此,提出了一种面向大图子图匹配的多GPU编程模型。首先,提出了基于多GPU的子图匹配算法框架,实现了子图匹配算法在多GPU上的协同操作,解决了GPU大图子图匹配的图规模问题。其次,采用了一种基于查询图的动态调节技术来处理跨分区子图集,解决了图划分导致的跨分区子图匹配难题。最后,结合GPU单指令多线程(SIMT)架构特性,提出一种优先级调度策略保证GPU的内部负载均衡,并设计了共享内存的流水线机制优化多核并发的缓存争用。实验表明,多GPU编程模型能够在数十亿级别的数据集上得到正确的匹配结果,与最新的基于GPU的解决方案相比,该算法框架能够获得1.2~2.6倍的加速比。

关键词:图分析;多GPU;大图子图匹配;优先级调度;并行编程模型
文献标志码:A　　**中图分类号:**TP311

Multi-GPU Programming Model for Subgraph Matching in Large Graphs

LI Cenhao[1], CUI Pengjie[1], YUAN Ye[2+], WANG Guoren[2]
1. School of Computer Science and Engineering, Northeastern University, Shenyang 110000, China
2. School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China

Abstract: Subgraph matching is an important method of data mining in complex networks. In recent years, the subgraph matching algorithm based on GPU (graphics processing units) has shown obvious speed advantages. However, due to the large scale of graph data and a large number of intermediate results of subgraph matching, the memory capacity of a single GPU soon becomes the main bottleneck for processing subgraph matching algorithm of large graph. Therefore, this paper proposes a multi-GPU programming model for large graph subgraph matching. Firstly, the framework of subgraph matching algorithm based on multi-GPU is proposed, and the cooperative operation of subgraph matching algorithm on multi-GPU is realized, which solves the problem of graph scale of subgraph matching on GPU. Secondly, a dynamic adjustment technique based on query graph is used to deal with cross-partition subgraph sets, which solves the cross-partition subgraph matching problem caused by graph segmentation. Finally, based on the characteristics of SIMT (single instruction multiple threads) architecture on GPU, a priority scheduling strategy is proposed to ensure the internal load balancing of GPU, and a pipeline

基金项目:国家自然科学基金(61932004,62002054,61732003,61729201);中央高校基本科研业务费专项资金(N181605012)。
This work was supported by the National Natural Science Foundation of China (61932004, 62002054, 61732003, 61729201), and the Fundamental Research Funds for the Central Universities of China (N181605012).
收稿日期:2021-11-11　　**修回日期:**2022-02-21

mechanism of shared memory is designed to optimize the cache contention of multi-core concurrency. Experiments show that the proposed multi-GPU programming model can get the correct matching results on billions of datasets. Compared with the latest GPU-based solution, the proposed algorithm framework can achieve 1.2 to 2.6 times of acceleration ratio.

Key words: graph analysis; multi-GPU; subgraph matching in large graphs; priority scheduling; concurrent programming model

图作为一种数据结构能够简洁地刻画出普遍事物间的联系，基于图的数据挖掘技术无论在学术研究还是工业应用上都享有重要的地位。其中最基本的一个任务是如何在图数据集中找到给定的查询图，也就是子图匹配问题。子图匹配算法近年来在工业研究、生物结构建模、社交网络等方面都有相关应用发展。例如：社区检测[1-3]、频繁子图挖掘[4-5]、生物事件提取[6]等。有很多相关的研究致力于提高子图匹配的搜索效率[7-10]。随着人工智能时代的来临，图数据的种类和规模不断增大，如谷歌和脸书网上的数据已经达到了PB级，子图匹配的序列化匹配方案已经不能满足实际需求。

因此，许多研究开始采用分布式[11-14]和并行处理[15-17]的解决方案。基于分布式的解决方案需要依赖网络间的频繁交互，这对于会产生大量中间结果的匹配算法是一种高通信代价的交互方式。而最近的GPU（graphics processing units）的解决方案利用GPU高并发高带宽的优势使得子图匹配的速度有了较大的提升。然而，由于单块GPU的内存容量有限和子图匹配算法在匹配过程中会生成大量的中间结果，大图子图匹配问题难以在单块GPU上处理，因此需要研究多GPU上的子图匹配问题。但研究基于多GPU的子图匹配又面临着以下挑战：

（1）如何构建有效的面向大图子图匹配的多GPU编程模型；（2）多GPU编程模型涉及图划分，如果直接采用以往的基于分布式的图划分策略生成跨分区匹配解，首先需要GPU之间进行频繁通信，PCIe的带宽将成为高效处理的瓶颈，其次在GPU端处理跨分区子图时会产生大量的条件分歧，直接影响GPU的匹配效率；（3）现实生活中的大图数据结构复杂，顶点度的倾斜分布使得GPU线程负载不均，极大损伤了GPU的并行性能。

为了解决以上挑战，本文提出了一种面向大图中子图匹配的多GPU编程模型来探索多GPU在子图匹配算法中的协同操作。该模型首先利用图划分引擎构建分区内子图和分区间子图，构建好的分区内子图存储到不同的GPU内存中，而分区间子图存储到CPU内存中；然后，该模型的图调度引擎有序调度各类计算资源来搜索分区内和分区间的实例。最后，采用批量同步并行（bulk synchronous parallel，BSP）模型对多GPU上的分区内匹配集和CPU上的跨分区匹配集进行整合。本模型还使用优先级调度策略和共享内存的流水线机制提高算法框架对于幂律图的处理能力和GPU的并发能力，使得该框架可以更好地处理现实世界的大型数据图。

本文的贡献可概括如下：

（1）提出了面向大图子图匹配的多GPU编程模型，解决图规模对于GPU大图子图匹配的限制。

（2）设计了一种基于查询图的动态调节技术，解决因图划分而带来的跨分区子图匹配问题，并在多GPU处理周期内利用空闲的多核CPU快速处理跨分区子图集。

（3）结合GPU的SIMT（single instruction multiple threads）架构特性，提出优先级调度策略和使用共享内存的流水线机制，减少子图匹配处理过程中的负载不均和空间争用。

（4）利用9个真实数据集和15个生成数据集验证了多GPU编程模型在大规模数据图下的有效性。与当前最先进的基于GPU的子图匹配算法相比，本文的编程模型不仅支持更大规模的数据图，并且在相同规模数据图下还可获得1.2~2.6倍的加速比。

1 相关工作

1.1 子图匹配

给定图 $G=(V_g,E_g,L_g)$ 表示数据图，V_g 表示数据图顶点集，E_g 表示数据图边集，L_g 表示数据图顶点标签集。给定图 $Q=(V_q,E_q,L_q)$ 表示查询图，V_q 表示查询图顶点集，E_q 表示查询图边集，L_q 表示查询图顶点标签集。用 u 和 v 分别表示查询图和数据图的任意顶点，$deg(u)$ 代表顶点 u 的度数，$freq(u.label)$ 代表顶点 u 的标签在数据顶点中出现的频次。对于数据

图 G 和查询图 Q，当且仅当存在一个映射函数 $f:V_q \rightarrow V_g$ 在两个图之间满足对应关系 $\forall(u_1,u_2)\in E_Q$，$\exists(f(u_1),f(u_2))\in E_G$ 时，图 Q 和图 G 是子图同构的。而子图匹配的目标是找到数据图 G 中所有和查询图 Q 子图同构的解。图1和图2展示了一个子图匹配的完整样例，根据查询图可以得到两个匹配集 $\{(u_0,v_0),(u_1,v_1),(u_2,v_2),(u_3,v_4),(u_4,v_5),(u_5,v_6)\}$ $\{(u_0,v_6),(u_1,v_7),(u_2,v_8),(u_3,v_9),(u_4,v_{10}),(u_5,v_{11})\}$。划分策略用虚线表示，其中，第一个匹配集是跨分区匹配集，第二个匹配集是分区内匹配集。

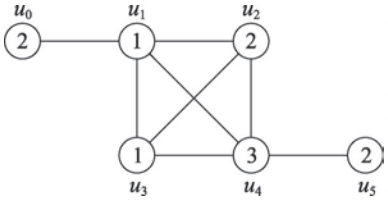

图1 查询图

Fig.1 Query graph

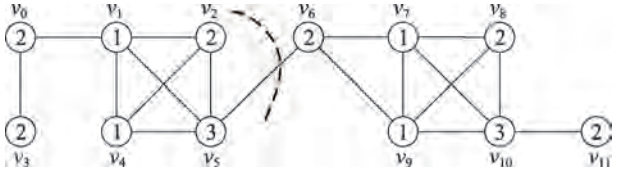

图2 数据图

Fig.2 Data graph

1.2 GPU架构与编程模型

GPU可视为数据并行的众核计算系统，本文主要研究NVIDIA GPU的架构并使用NVIDIA提供的通用并行计算架构（compute unified device architecture，CUDA[18]）。如今，一台服务器可以配备多个GPU处理器，GPU间通过PCIe连接，CPU、GPU和互连的拓扑可以在完整的对等连接和层级交换拓扑之间变化[19-20]。GPU内部包含一组固定的流式多处理器（streaming multiprocessor，SM）和显存，每个GPU内核采用SIMT指令结构，并包含多个内存复制引擎，保证代码同时执行和传输。GPU程序通过调度一个包含大量线程的网格，网格被分组为线程块，在处理器上并行运行。在分配给单个处理器的每个线程块中，32个线程被组成一组并称为线程束（warp），并以锁步方式在内核上执行。此外，同一线程块中的线程可以通过共享内存进行同步和通信，并使用可编程的 L_1 和 L_2 缓存。在下一章中，本文将介绍一个基于以上架构的多GPU的大图子图匹配编程模型。

2 本文的多GPU编程模型

本文的多GPU编程模型所采用的软件架构如图3所示。针对子图匹配算法，最底层为存储层，包含主存、显存和零拷贝内存，主存仅存储跨分区子图集，大量的图数据会被划分到显存中，零拷贝内存主要在面向查询图的动态调节技术中使用。第二层为图拓扑层，包含数据图和查询图的拓扑结构。第三层是通信接口层，发送器和接收器用于计算资源间的通信，路由器用于记录多个计算资源间的路由信息。第四层基于下层编写，包括图划分引擎和图调度引擎，图划分引擎将图数据划分到异构计算资源上，图调度引擎使得系统自动协调分配各类计算资源同时屏蔽了复杂的调度逻辑，以支撑顶层子图匹配算法的计算。提出的编程模型隐藏了编程复杂性，同时保证了高性能和高并发，具体阐述如下。

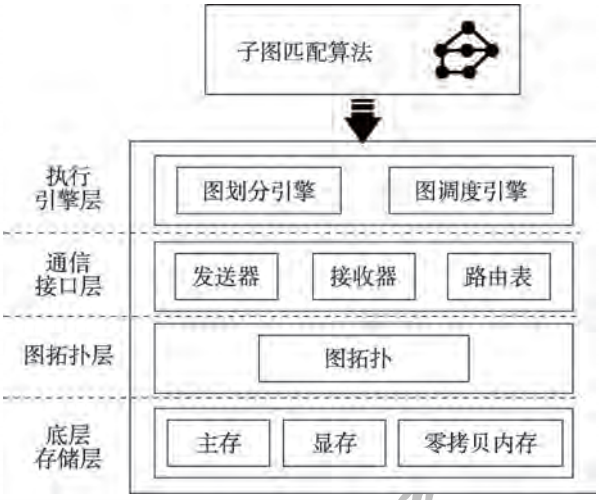

图3 软件架构

Fig.3 Software architecture

2.1 图划分引擎

图划分引擎主要用于将数据图划成满足单块GPU内存的GPU子图，并且基于划分好的GPU子图构建跨分区子图。图划分引擎采用标准的图划分方式METIS（multi-level partitioning algorithm）[21]进行图划分。该划分方式通过寻求最小的跨分区边集来保证将尽可能多的图数据划分到不同GPU内存中，使得多GPU编程模型可以充分利用GPU的高并发和高带宽优势。

而对于不可避免的跨分区边集，本文采用提出的面向查询图的动态调节技术来生成以跨分区边集为中心、K 跳范围内的子图集，并将其存储到CPU端，使得多GPU编程模型可以进一步利用多核CPU

的架构优势，如分支预测来处理复杂的跨分区实例。以图4为例来说明图划分引擎的工作原理。对于给定的数据图 G，图划分引擎首先利用METIS将其划分到不同的GPU内存中（G_0 和 G_2），G_0 和 G_2 子图间仅通过一条边互连，因此满足最小的跨分区边集，避免了GPU之间的频繁交互。而对于跨分区子图集（G_1）会利用面向查询图的动态调节技术构建，并将其部署到CPU内存中。跨分区子图的构建将在第3章具体阐述。

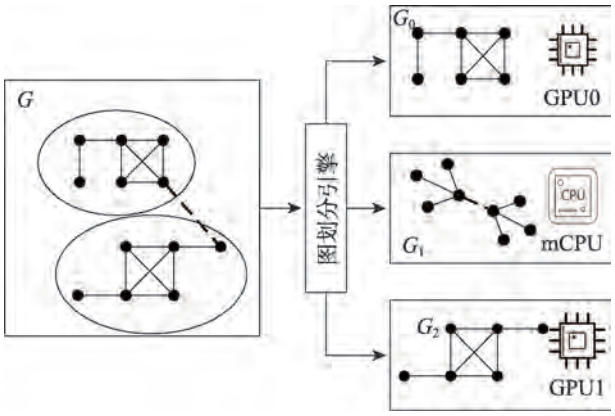

图4 图划分引擎

Fig.4 Graph partition engine

2.2 图调度引擎

图划分引擎构建完毕后，图调度引擎将采用多GPU和CPU的异构工作模式分别处理分区内子图和分区间子图。算法1介绍了本文所设计的图调度引擎。输入由图划分引擎提供，包含分区内子图 *gpu_segGraph* 和分区间子图 *cpu_segGraph*。第2~4行表示发送器、接收器、路由器的建立，用于计算资源间的路由与链接。其中，计算资源是指运行子图匹配时所需的CPU资源和GPU资源。路由器提供了统一便捷的地址查询接口，记录了多个计算资源间的路由信息。路由器内部构建了多个计算节点组成的映射结构，提供可以执行一对多或多对一的传输函数。传输函数只控制数据的发送方式，并根据发送器和接收器携带的信息分配具体目的地。发送器和接收器用于节点内计算资源间的通信，并提供真实的一对一数据传输。第5~7行代码为每个GPU设备创建一个工作线程，每个GPU根据查询图顶点排序函数 $f(u)$ [22-23]生成的查询顺序，基于广度优先搜索的算法思想利用GPU数以千计的线程并行生成分区内实例。$f(u)$ 可以通过式（1）计算：

$$f(u)=\frac{deg(u)}{freq(u.label)} \tag{1}$$

与此同时，空闲的多核CPU会并行处理面向查询图的动态调节技术产生的跨分区子图（第8~10行）。先前的算法研究[9,22,24-25]中使用的查询图顶点数在4~32之间，通过随机提取数据图的方式生成查询图。而3跳以内的查询图最多可生成 N_q 个顶点，足以满足常见的查询图匹配数据图的需求。当部分查询图规模大于3跳时，本文使用第3章介绍的面向查询图的动态调节技术解决图规模过大的难题。N_q 通过式（2）计算：

$$N_q=\frac{1-deg(u)^4}{1-deg(u)} \tag{2}$$

跨分区子图集的规模远小于存储在多GPU内的分区内子图集的规模，并且具有图结构小而多、顶点顺序不连续的特点。因此跨分区子图集更适合CPU进行处理并将计算时间隐藏在GPU的计算过程中。相应的时间比较将在第5章给出。在第11行完成异构处理的同步后，第12~13行使用接收器进行调度，该过程需要整合多个物理设备间的匹配结果，保证匹配解的完整性并对生成的匹配解进行去重，验证结果的正确性，同时保证CPU处理的跨分区匹配结果与多GPU处理的分区内匹配结果的唯一性，并在第14行返回所有结果。

算法1 图调度引擎

输入：分区间子图 *cpu_segGraph* 和分区内子图 *gpu_segGraph*。

输出：查询图 Q 的所有匹配结果 *recv*。

1. procedure *Engine*(*cpu_segGraph, gpu_segGraph*)
2. Sender *send*
3. Router *router*
4. Receiver *recv*
5. For each Device_t *i* in ngpus
6. Thread *d_func*(*workerThread, q_order, ia_results*)
7. End for
8. Parallel for
9. *send.interEngine*(*cpu_segGraph, ir_results*)
10. End for
11. *send.barrier*()
12. *recv.refine*(*ia_results , ir_results, router*)
13. *recv.collect*(*ia_results , ir_results, router*)
14. return *recv*
15. End procedure

GPU端的执行函数如算法2所示,采用生成-过滤-连接策略,第5行依据先前生成的匹配顺序 *q_order* 生成顶点候选集 *cand_dset* 。在第6行进行两阶段过滤,过滤出不能匹配查询点的候选顶点,减少中间候选结果的规模。第7行依据过滤后的顶点集生成侯选边集。候选边集规模庞大,利用GPU的高并发性可以快速生成候选解,并在第8行进行组合产生最终的分区内子图匹配解决方案。整个匹配过程中最耗时的步骤是利用侯选边集生成最后的匹配解,时间复杂度为 $O\left(\prod_{1\sim E_q} C(e)\times \mathrm{lb}\,\nabla(V_g)/(SMs\times threads)\right)$。其中,$C(e)$ 代表候选边集,$SMs\times threads$ 代表每个SM中的工作线程数,时间复杂度与顶点集大小和侯选边集大小有关,并分摊到GPU多线程中快速并行匹配。

算法2 GPU执行函数

输入:GPU设备序号 *gpu_id*,查询图顶点排序 *q_order*。

输出:分区内匹配结果 *ia_results*。

1. Procedure *WorkerThread*(*gpu_id, q_order*)
2. Stream *stream*(*gpu_id*)
3. Data_t *cand_dset*
4. Data_t *cand_eset*
5. *initCandSetKernel*(*q_order, graph_seg, cand_dset*)
6. *refineCandSetKernel*(*graph_seg, cand_dset*)
7. *findCandEdgeKernel*(*graph_seg, cand_dset, cand_eset*)
8. *joinCandEdgeKernel*(*graph_seg, cand_eset, ia_results*)
9. *seg.Synchronize*(*stream*)
10. End procedure

3 面向查询图的动态调节技术

分区间子图匹配枚举依赖于面向查询图的动态调节技术。在大部分常规匹配中,3跳内的跨分区子图已经足够正确产生跨分区匹配解,而面对极少部分的特殊查询匹配,利用动态调节扩展跨分区子图的规模,来满足特殊查询的匹配需求。具体阐述如下:

对于跨分区子图搜索,通过 $\varphi(v)$ 判断数据边是否为跨分区边,$\varphi(v)$ 指顶点 v 所在子图 G_i 的序号,即 $\varphi(v)=i$。当存在式(3)时,(u_i,u_j) 被判定为一条分区内边,当存在式(4)时,(u_i,u_j) 被判定为一条跨分区边。

$$u_i,u_j\in V_q,\varphi(f(u_i))=\varphi(f(u_j)) \tag{3}$$

$$u_i,u_j\in V_q,\varphi(f(u_i))\neq\varphi(f(u_j)) \tag{4}$$

基于每条跨分区边 ρ,根据设定的跳数 π,生成以跨分区边为中心、π 跳以内的跨分区子图集,即为 P_π。其中,π 跳指的是对于查询图任意的顶点 u,其通过不大于 π 次深度遍历可以到达查询图的任意顶点 u'。真实世界中,大部分的查询图规模都没有超过3跳,因此,在预处理跨分区子图时,对于每个跨分区子图 P_π^i,都包含查询图 Q 的全部搜索范围。这有利于在多GPU处理大量分区内子图匹配的同时,利用空闲的多核CPU快速处理跨分区子图集。由于常规的子图集具有小而多的结构特点,采用回溯法快速匹配枚举结果是一项很省时的匹配方案。

对于大于3跳的查询图结构,单GPU无法计算出全部的匹配解,本文将采用零拷贝技术从GPU中动态调节跨分区子图集的规模,来保证正确计算出所有的匹配解。相比基于直接内存访问(direct memory access,DMA)的方法,零拷贝技术不仅能保证访问离散数据时的性能,还能避免显式的数据传输[26]。跨分区匹配具体将分成两种情况:(1)当查询图图规模为4跳且在图划分边缘时,单GPU内存中的分区内子图和CPU内存中的跨分区子图并不能完全覆盖4跳规模的查询图,此时GPU利用零拷贝技术从CPU的零拷贝访问内存中拿取跨分区子图并与内部的图分割点融合,以满足4跳规模的查询图匹配需求。(2)当查询图规模扩张到4跳以上且在匹配图划分边缘的子图时,原有的单GPU内存中的分区内子图和CPU内存中的跨分区子图已不能保证匹配到所有结果,此时需要依据CPU上路由器信息和发送器对GPU上的图存储进行调节。如图5所示,细箭头表示逻辑传输,粗箭头表示真正的数据传输。当 GPU_i 需要调节内部存储图时,会将信息发送给路由器(步骤①),由路由器根据路由表传送给发送器(步骤②),通过发

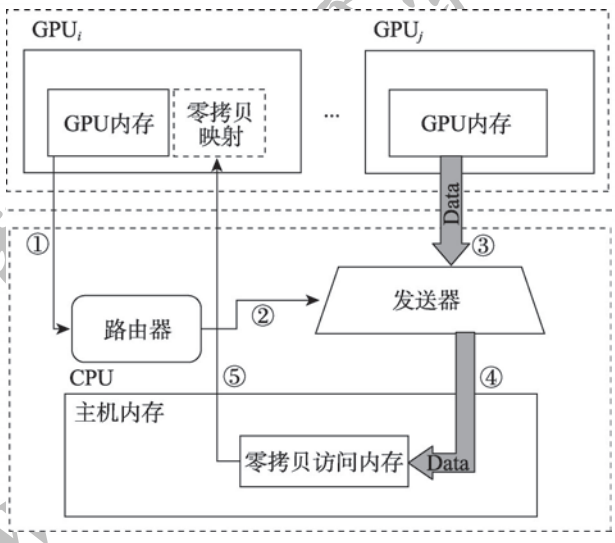

图5 4跳以上的零拷贝调度策略

Fig.5 Zero-copy scheduling strategy more than 4 hops

送器获取 GPU_j 内存中的边缘点集（步骤③）并存储到 GPU 零拷贝访问内存中（步骤④）。GPU_i 利用零拷贝技术可以直接从零拷贝访问内存中获取新的边缘点集并和自身的设备内存中的分区内子图结合（步骤⑤），避免复杂的数据传输并满足4跳以上的查询图匹配需求。

4 GPU内优化操作

4.1 优先级调度

现实生活中的大图数据规模宏大、结构复杂，它们的顶点度大多服从幂律分布，即存在着大量的小度数顶点和少量的大度数顶点。由于GPU内部采用线程束（warps）作为基本的执行单元，当处理幂律图时，只有少部分活跃线程束（active warps）贯穿始终，大部分线程会作为非活跃线程束（inactive warps）等待活跃线程束执行完毕，因此顶点度的偏斜导致了GPU端线程的负载不均，产生线程拖慢，极大损伤了GPU的并行性能。而现有的基于GPU的子图匹配算法并未考虑大图的复杂结构对GPU性能的损伤。

为此，本文采用三阶段优先级调度模式，用以减少复杂多样的图结构对利用GPU进行子图匹配算法加速的影响。三阶段调度模式分别以线程块为中心、线程束为中心和以线程为中心对GPU的工作线程进行调度划分。优先级 $Pri(v)$ 如式（5）表示，$Deg(v)$、$freq(v.label)$ 分别是顶点度大小和顶点标签出现的频次：

$$Pri(v)=\frac{1}{Deg(v)\cdot freq(v.label)} \tag{5}$$

优先级越小时，说明该匹配集匹配的频次越高。在优先级调度过程中，设置线程块大小作为线程块阈值，线程束大小作为线程束阈值。当 $Pri(v)$ 小于线程块限制阈值时，将一组线程块全部调度给一组工作，当 $Pri(v)$ 大于线程束限制阈值时，利用单个线程处理小而多的工作模块。$Pri(v)$ 在两个阈值中间时，证明该顶点的待匹配集出现的频次居中，更适合使用线程束处理。如图6所示，当顶点 v_{i-1} 的 $Pri(v)$ 判定小于线程块阈值时，说明该顶点邻接顶点多，待匹配集规模大，使用一组线程块的全部线程快速执行匹配。当顶点 v_i 的 $Pri(v)$ 判定在线程块阈值和线程束阈值中间时，使用线程束处理匹配集居中的顶点。此时剩余顶点的匹配集规模都不大，顶点 v_{i+1}、v_{i+2}、v_{i+3} 的 $Pri(v)$ 大于线程束阈值，用单个线程处理

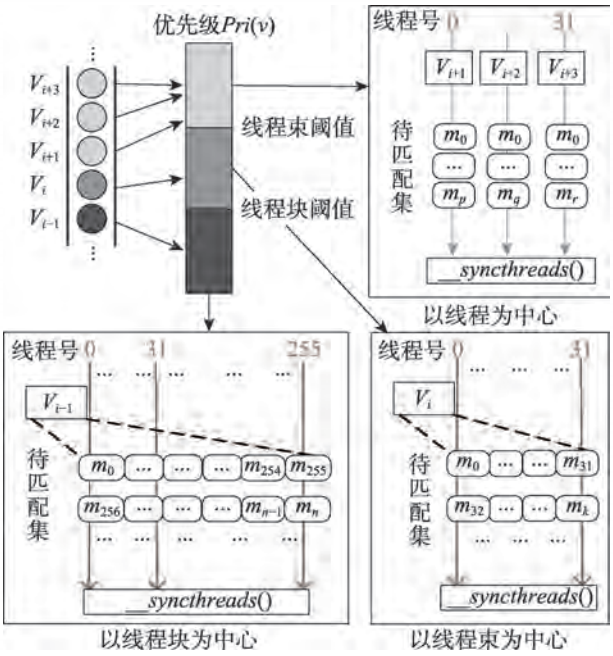

图6 三阶段优先级调度模型

Fig.6 Three-stage priority scheduling model

零星的匹配结果可以有效减少线程等待。三阶段调度模型是一种更加平衡且自适应图结构的调度模式。面对幂律图分布不均衡的顶点性质，将不同规模的待匹配集更均衡的部署在GPU的工作线程上，既减少了块间不平衡，也减少了线程束间的不平衡，同时也避免了过多调度判断导致的条件分支。

4.2 共享内存的流水线机制

在生成待匹配集和匹配过程中，为提高内存访问效率，会使用GPU的共享内存空间来存储高频次访问的图数据。共享内存的空间十分有限，通常以KB级为主，如Tesla T4的每个SM共享内存有48 KB。当每个线程块占用共享内存空间不均衡时，会造成空间的浪费。因此，在多个线程块并发执行时，使用共享内存会决定线程的并发数量，即线程级并发度。由于一个线程块所分配的共享内存空间的生命周期贯穿执行始末，对于共享内存的常规使用会造成线程块的串行执行，成为多个线程块间并行的瓶颈。

为此，本文使用了基于共享内存的流水线机制来解决以上问题。与子图匹配算法在GPU上的计算时间相比，共享内存的占用时间很短，可以将多个线程块中计算区域与共享内存占用区叠加，减少总的执行时间。多个线程块将计算时间与共享内存占用时间重叠，期间通过线程束级的分支条件控制以及同步函数确保正确性，并将除匹配外的小型计算整合，以填补共享内存占用期间的计算空白。

5 实验与评估

本文将提出的大图子图匹配多GPU编程模型与当下最新的子图匹配算法进行实验比较,以展示所提出的多GPU编程模型对子图匹配算法的优化。

由于在先前的研究工作中已经展现了单GPU子图匹配算法对于基于CPU的子图匹配算法的优越性,并且还未有基于多GPU的子图匹配算法出现,故本文只跟当下主流的基于单GPU子图匹配算法进行比较,包括GpSM(GPU-friendly method for subgraph matching)[15]、GSI(GPU-friendly subgraph isomorphism)[16]、GunrockSM[27]。GpSM和GunrockSM都是基于广度优先遍历并面向边连接的单GPU子图匹配算法。GSI是当下最新的基于单GPU并面向顶点连接的子图匹配算法。实验设备为一台安装有ubuntu18.04的GPU服务器。该服务器配备了Intel Xeon Silver 4214 CPU@2.20 GHz并且有128 GB主机内存,4个具有11 GB内存的NVIDIA RTX 2080Ti。

5.1 数据集与查询图

实验在真实数据集和合成数据集上进行,真实数据集用于评测框架的运行时间以及优化效果,合成数据集用于连续测试框架的稳定性。真实数据集如表1所示,包含基于Youtube的社交网络、Texas的道路网络、维基百科数据图Enwiki-2016和Enwiki-2020、书网数据图Fb_se、大规模爬行图Gsh-2015[28]以及图性质符合幂律分布的数据图WebGoogle[29]、LiveJournal[29]、HiggsTwitter[30]。

表1 数据集统计

Table 1 Statistics of datasets

数据集	顶点数/10^6	边数/10^6	大小/GB
Youtube	1.31	1.92	0.056
Texas	2.38	5.92	0.134
Enwiki-2016	4.21	101.40	1.432
Enwiki-2020	10.20	301.40	3.449
Fb_se	35.50	617.20	8.449
Gsh-2015	68.70	1 802.70	29.552
WebGoogle	0.92	5.53	0.166
HiggsTwitter	0.46	14.90	0.324
LiveJournal	4.85	69.00	1.012

在评估了现有子图匹配工作后,本文依据先前研究常用的查询图匹配本文使用的数据集。如图7所示,除了常用的查询图外,还使用超规模查询图测试本文使用的面向查询图的动态调节技术的优劣性(q_6)。实验给出正确的匹配结果并计算执行匹配的时间,每个实验测试多次并给出平均值。

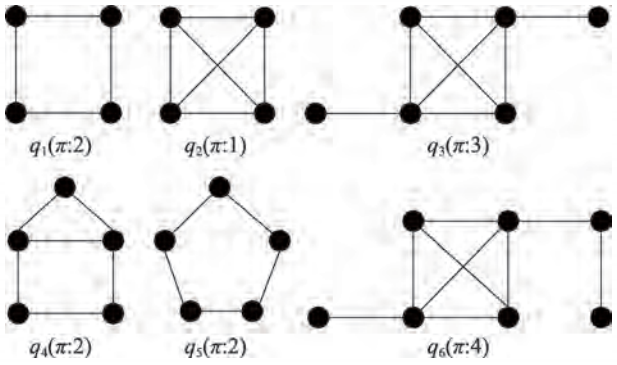

图7 查询图

Fig.7 Query graphs

5.2 多GPU子图匹配框架性能分析

图8展示了本文提出的框架与当前最新的GPU解决方案的运行时间,缺失的数据表示运行因崩溃、内存不足导致的失败。如图8所示,在前3个小规模数据集上,本文方法与GpSM和GunrockSM相比具有明显优势,加速比在4倍以上。主要原因在于本文提出的编程模型屏蔽了分区间的PCIe通信,使得多GPU间执行耦合度降低,并利用多GPU更强的并行能力快速生成子图匹配结果。而在小数据集上和最新的GPU解决方案GSI相比,优势不明显,主要由于多GPU的调度时间无法被计算时间掩盖。在第四个数据集Enwiki-2020上,数据规模已经达到了数亿级,本文的多GPU模型对于大型数据集的处理能力开始显现。当数据集规模不断扩大,GpSM和GunrockSM已经因为超过内存限制而无法继续进行子图匹配计算。在第五个数据集上,本文的方法模型已经可以实现2.6倍的加速比。当数据集扩展到数十亿级别时,GSI利用单GPU也不能继续计算,本文的方法在多GPU上依旧可以处理规模庞大的匹配运算。

在利用多GPU对子图匹配算法进行加速的同时,还需使用CPU处理因图划分带来的跨分区子图集。CPU具有良好的分支预测功能,更适合处理小而多的图结构。图9展示了利用多GPU处理分区内子图集与多核CPU处理跨分区子图集的耗时比较。其中最顶部的部分代表CPU的计算耗时,其余部分是每个GPU上的计算耗时。与单个GPU的用时相比,CPU与GPU的耗时差距在10倍以上。因此,多核CPU有足够多的时间进行匹配计算并隐藏在GPU的计算耗时中。

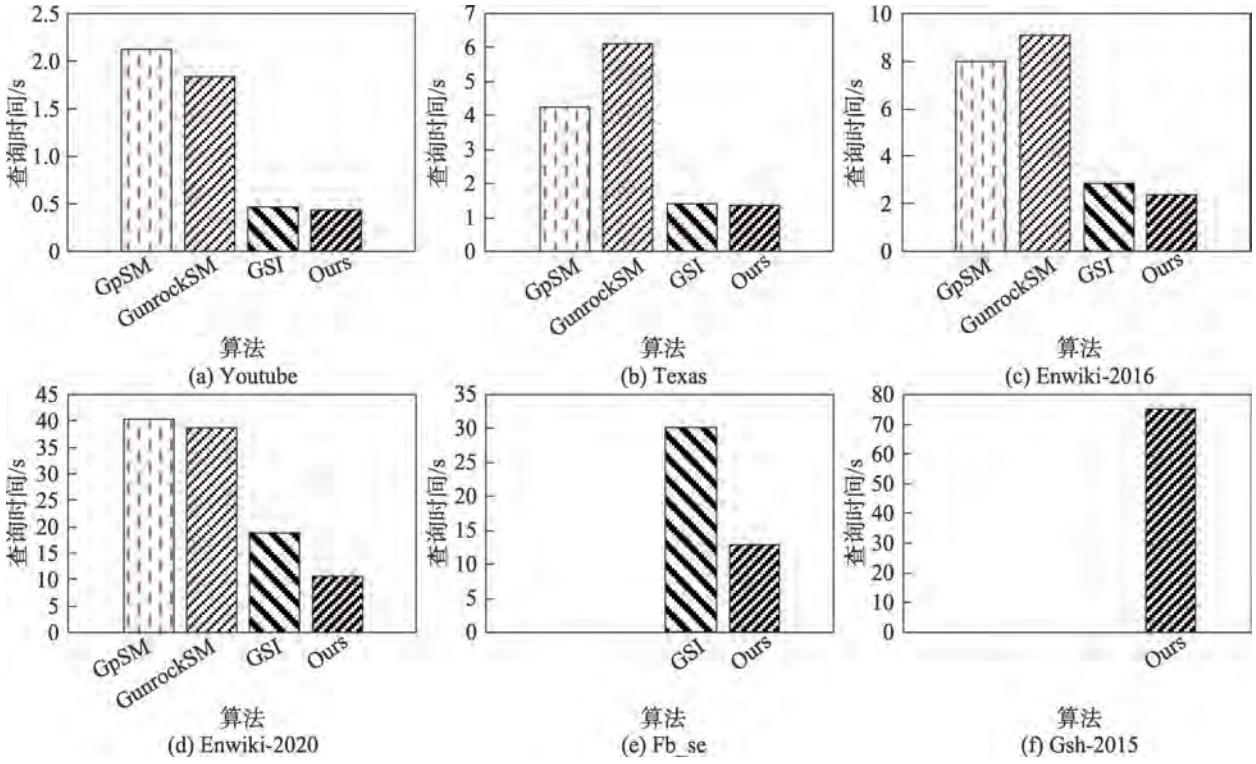

图8 性能比较

Fig.8 Performance comparison

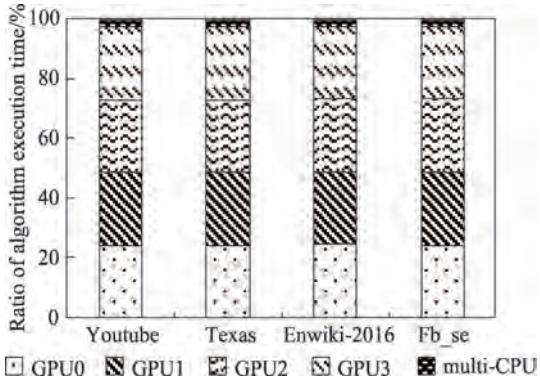

图9 CPU与多GPU算法执行时间占比

Fig.9 CPU to multi-GPU algorithm execution time ratio

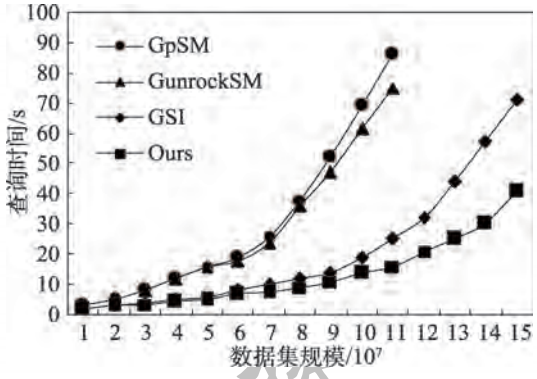

图10 稳定性分析

Fig.10 Stability analysis

5.3 多GPU子图匹配框架稳定性分析

由于很难找到连续平稳增加的数据集，对于提出的框架稳定性分析利用生成图进行测试。生成图集保证图的连通性和图生成的随机性。如图10所示，生成图集序号1~15代表1 000万~1.5亿规模的数据集。随着数据规模的不断增加，GpSM和GunrockSM呈现指数型增长，并且当数据规模达到亿数量级时，这两种算法由于内存不足不能正确匹配数据。最新的GSI算法由于在连接操作上的优化使其能够处理更大的数据图，不过随着数据规模增大，处理时间也会陡增。相比较而言，本文的框架利用多GPU的并行性更高，处理大规模数据的性能更好，对于数据图规模的增大趋势并没有那么敏感，同时该框架也具有良好的可扩展性。

5.4 多GPU子图匹配框架优化比较

对于多GPU子图匹配框架基础上的共享内存流水线机制的利用，主要是为了解决利用共享内存加速的GPU内核函数受限于共享内存空间而降低并行度的问题。本文在使用流水线加速机制前后分别对本文算法框架进行了评估。

如表2所示，由于多线程块并发执行时才会对共享内存进行抢用，这种性能优化在数据集规模足够

表2 共享内存优化分析

Table 2 Shared memory optimization analysis

数据集	大小/GB	共享内存优化
Youtube	0.056	1.03x
Texas	0.134	1.08x
Enwiki-2020	3.449	1.13x
Fb_se	8.449	1.25x
Gsh-2015	29.552	1.37x

庞大时,会更凸显这种流水线机制的作用。当大量的核函数计算时间足够与共享内存占用时间重叠时,这种加速效果尤为明显。

而对于本文框架处理幂律图时,本文所提出的优先级调度优化开始展现优势。如表3所示,在非幂律图上这种调度策略对时间的影响微小,主要是因为常规的使用线程束处理数据图是比较均衡的处理模式。而在幂律图上,不加调度的基础方式会因为负载不均衡导致拖尾效应,难以发挥多GPU在大规模数据匹配上的优势。采用优先级调度方式会更适合处理密度分布不均衡的数据图。当数据的 $Pri(v)$ 值小于线程块阈值时,调用线程块集中处理会将线程的负载降低87%,而当数据的 $Pri(v)$ 值大于线程束阈值时,调用单独线程处理匹配解会扩大线程间的并行度,因而减少内核函数执行的总时间。

表3 优先级调度优化分析

Table 3 Priority scheduling optimization analysis

数据集	是否是幂律图	分支调度优化
Youtube	否	0.98x
Texas	否	0.99x
Enwiki-2016	否	1.01x
Enwiki-2020	否	1.03x
Fb_se	否	1.02x
Gsh-2015	否	1.01x
WebGoogle	是	1.14x
HiggsTwitter	是	1.21x
LiveJournal	是	1.33x

6 结论

本文提出了一种面向大图子图匹配的多GPU编程模型。该模型可以在多GPU上并行处理子图匹配问题,解决了图规模对于GPU大图子图匹配的限制。其次,采用了一种基于查询图的动态调节技术来处理跨分区子图集,解决了图划分导致的跨分区子图匹配难题。最后,结合GPU的SIMT架构特性,提出一种优先级调度策略,保证GPU的内部负载均衡,并设计了共享内存的流水线机制优化多核并发的空间争用。实验表明,本文的多GPU编程模型能够在数十亿级别的数据集上得到正确的匹配结果,与最新的基于GPU的解决方案相比,该算法框架能够获得1.2~2.6倍的加速比。

参考文献:

[1] KAIRAM S R, WANG D J, LESKOVEC J. The life and death of online groups: predicting group growth and longevity [C]//Proceedings of the 5th ACM International Conference on Web Search and Data Mining, Seattle, Feb 8-12, 2012. New York: ACM, 2012: 673-682.

[2] WANG J, CHENG J. Truss decomposition in massive networks [J]. Proceedings of the VLDB Endowment, 2012, 5(9): 812-823.

[3] 张晓琳, 袁昊晨, 李卓麟, 等. 面向子图匹配的社会网络隐私保护方法[J]. 计算机科学与探索, 2019, 13(9): 1504-1515.
ZHANG X L, YUAN H C, LI Z L, et al. Subgraph matching oriented privacy preserving method for social network[J]. Journal of Frontiers of Computer Science and Technology, 2019, 13(9): 1504-1515.

[4] JENA B S, KHAN C, SUNDERRAMAN R. High performance frequent subgraph mining on transaction datasets: a survey and performance comparison[J]. Big Data Mining and Analytics, 2019, 2(3): 159-180.

[5] UGANDER J, BACKSTROM L, KLEINBERG J. Subgraph frequencies: mapping the empirical and extremal geography of large graph collections[C]//Proceedings of the 22nd International Conference on World Wide Web, Rio de Janeiro, May 13-17, 2013. New York: ACM, 2013: 1307-1318.

[6] LIU H, KESELJ V, BLOUIN C. Biological event extraction using subgraph matching[C]//Proceedings of the 4th International Symposium for Semantic Mining in Biomedicine, Cambridge, Oct, 2010: 110-115.

[7] LAI L, QIN L, LIN X, et al. Scalable distributed subgraph enumeration[J]. Proceedings of the VLDB Endowment, 2016, 10(3): 217-228.

[8] SUN S, LUO Q. In-memory subgraph matching: an in-depth study[C]//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, Portland, Jun 14-19, 2020. New York: ACM, 2020: 1083-1098.

[9] BI F, CHANG L, LIN X, et al. Efficient subgraph matching by postponing Cartesian products[C]//Proceedings of the 2016 International Conference on Management of Data, San Francisco, Jun 26-Jul 1, 2016. New York: ACM, 2016: 1199-1214.

[10] REN X, WANG J. Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs[J]. Proceedings of the VLDB Endowment, 2015, 8(5): 617-628.

[11] 许文, 宋文爱, 富丽贞, 等. 面向大规模图数据的分布式子图匹配算法[J]. 计算机科学, 2019, 46(4): 28-35.

XU W, SONG W A, FU L Z, et al. Distributed subgraph matching algorithm for large scale graph data[J]. Computer Science, 2019, 46(4): 28-35.
[12] LAI L, QIN L, LIN X, et al. Scalable subgraph enumeration in MapReduce[J]//Proceedings of the VLDB Endowment, 2015, 8(10): 974-985.
[13] QIAO M, ZHANG H, CHENG H. Subgraph matching: on compression and computation[J]. Proceedings of the VLDB Endowment, 2017, 11(2): 176-188.
[14] WANG X, CHAI L, XU Q, et al. Efficient subgraph matching on large RDF graphs using MapReduce[J]. Data Science and Engineering, 2019, 4: 24-43.
[15] TRAN H N, KIM J, HE B. Fast subgraph matching on large graphs using graphics processors[C]//LNCS 9049: Proceedings of the 20th International Conference on Database Systems for Advanced Applications, Hanoi, Apr 20-23, 2015. Cham: Springer, 2015: 299-315.
[16] ZENG L, ZOU L, ÖZSU M T, et al. GSI: GPU-friendly subgraph isomorphism[C]//Proceedings of the 36th International Conference on Data Engineering, Dallas, Apr 20-24, 2020. Piscataway: IEEE, 2020: 1249-1260.
[17] GUO W, LI Y, SHA M, et al. GPU-accelerated subgraph enumeration on partitioned graphs[C]//Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, New York, Jun 14-19, 2020. New York: ACM, 2020: 1067-1082.
[18] CUDA parallel computing platform[EB/OL]. [2021-09-04]. http://www.nvidia.com/object/cuda home new.html.
[19] ALAM M, PERUMALLA K S, SANDERS P. Novel parallel algorithms for fast multi-GPU-based generation of massive scale-free networks[J]. Data Science and Engineering, 2019, 4: 61-75.
[20] GALLET B, GOWANLOCK M. Heterogeneous CPU-GPU epsilon grid joins: static and dynamic work partitioning strategies[J]. Data Science and Engineering, 2021, 6: 39-62.
[21] KARYPIS G, KUMAR V. A fast and high quality multilevel scheme for partitioning irregular graphs[J]. SIAM Journal on Scientific Computing, 1998, 20(1): 359-392.
[22] WOOKS H, JINSOO L, JEONGH L. TurboISO: towards ultrafast and robust subgraph isomorphism search in large graph databases[C]//Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, New York, Jun 22-27, 2013. New York: ACM, 2013: 337-348.
[23] SUN Z, WANG H, WANG H, et al. Efficient subgraph matching on billion node graphs[J]. Proceedings of the VLDB Endowment, 2012, 5(9): 788-799.
[24] BIBEK B, HANG L, HUANG H H. CECI: compact embedding cluster index for scalable subgraph matching[C]// Proceedings of the 2019 International Conference on Management of Data, Amsterdam, Jun 30-Jul 5, 2019. New York: ACM, 2019: 1447-1462.
[25] MYOUNGJI H, HYUNJOON K, GEONMO G, et al. Efficient subgraph matching: harmonizing dynamic programming [C]//Proceedings of the 2019 International Conference on Management of Data, Amsterdam, Jun 30-Jul 5, 2019. New York: ACM, 2019: 1429-1446.
[26] SEUNG W M, VIKRAM S M, ZAID Q, et al. EMOGI: efficient memory-access for out-of-memory graph-traversal in GPUs[J].Proceedings of VLDB Endowment, 2020, 14 (2): 114-127.
[27] WANG L, WANG Y, OWENS J D. Fast parallel subgraph matching on the GPU[C]//Proceedings of the 2016 ACM Symposium on High-Performance Parallel and Distributed Computing, Kyoto, May 31-Jun 4, 2016. New York: ACM, 2016.
[28] WebGraph Project Website. Laboratory for web algorithmics [EB/OL]. [2021-09-04]. http://law.di.unimi.it/index.php.
[29] LESKOVEC J, LANG K J, DASGUPTA A, et al. Community structure in large networks: natural cluster sizes and the absence of largewell-defined clusters[J]. Internet Mathematics, 2009, 6(1): 29-123.
[30] DOMENICO M D, LIMA A, MOUGEL P, et al. The anatomy of a scientific rumor[J]. Scientific Reports, 2013, 3(1): 1-9.

李岑浩(1997—),男,辽宁锦州人,硕士研究生,CCF学生会员,主要研究方向为基于新硬件的大图数据分析。
LI Cenhao, born in 1997, M.S. candidate, student member of CCF. His research interest is large graph data analysis based on new hardware.

崔鹏杰(1993—),男,博士,主要研究方向为基于新硬件的大图数据分析。
CUI Pengjie, born in 1993, Ph.D. His research interest is large graph data analysis based on new hardware.

袁野(1981—),男,博士,教授,主要研究方向为图数据库、概率数据库、数据隐私保护、云计算。
YUAN Ye, born in 1981, Ph.D., professor. His research interests include graph databases, probabilistic databases, data privacy-preserving and cloud computing.

王国仁(1966—),男,博士,教授,主要研究方向为XML数据管理、查询处理和优化、生物信息学、高维索引、并行数据库系统、P2P数据管理。
WANG Guoren, born in 1966, Ph.D., professor. His research interests include XML data management, query processing and optimization, bioinformatics, high-dimensional indexing, parallel database systems and P2P data management.