doi:10.3772/j.issn.1002-0470.2022.01.001

基于 GPU 的子图匹配优化技术①

孟 轲②*** 林志恒*** 谭光明③*

(*中国科学院计算技术研究所高性能计算研究中心 北京 100190)

(**中国科学院大学 北京 100049)

摘 要 为了解决图挖掘应用中子图匹配任务的性能问题,本文提出了一种基于图形处理单元(GPU)的顶点预剪枝子图匹配系统(GVSM)。GVSM 采用黑名单剪枝算法和调度排序来减少冗余搜索。利用前缀树数据结构,GVSM 可以对中间结果进行压缩,以便快速索引并降低内存消耗。GVSM 将子图匹配的搜索部分卸载到 GPU 上执行,通过设计软件流水线进行重叠计算和数据移动,在 PCI-E 接口传输数据图拓扑数据的同时激活中央处理器(CPU)与 GPU 上的计算,并用动态负载均衡的方法减少计算资源的浪费。实验结果表明,本文方法能够有效提升子图匹配算法的性能,GVSM 在性能上相比国际同类算法有显著提升,并且能处理更大规模的数据。

关键词 子图匹配;图挖掘;图形处理单元(GPU);高性能;图处理

0 引 言

子图匹配算法在多个领域有着广泛的应用,例如从蛋白质相互作用网络中提取重要子图结构来揭示其三维结构,在语义数据上挖掘推理模式,从资源描述框架(resource description framework,RDF)[1]数据中挖掘关联信息,在社交媒体数据中挖掘有价值的用户关系等。子图匹配算法通常将带标签的模式图(pattern graph)作为输入,并通过在数据图(data graph)中查找这些模式的所有实例来提取出用户感兴趣的子图。这类算法通常围绕着数据图中的子图展开搜索和匹配,而传统的图计算应用(例如宽度优先搜索和佩奇排名)则是围绕着顶点状态展开计算[2-4],一些基于图的学习模型也不会涉及到图的拓扑,而是在每个节点上有更大计算量[5-7]。与之相比,子图匹配算法内存消耗更大,更容易受到数据非规则性的影响。随着应用领域不断扩大,需要处理的图数据的大小也逐年增加,目前图数据的边数已达到百亿级别[8]。遍历这些边带来的大量数据移动和计算量对当下的计算机构成了不小的挑战。图应用作为一种典型的非规则负载,其极低时空局部性使其很难在当前的体系结构平台上获得令人满意的并行效率,对数据结构和负载均衡算法的设计也一直是图算法的研究热点。

近些年来,以图形处理单元(graphic processing unit, GPU)为中心的异构图处理系统不断涌现[9-10]。相比于以中央处理器(center processing unit,CPU)为主的图计算系统而言,GPU 有着更高的带宽和并发度,这使得 GPU 能够更快地遍历图中所有的边并能同时处理更多的请求。而以子图匹配为代表的图挖掘算法恰恰需要极高的并发度和带宽来保证性能。异构计算平台越来越普遍,尤其是在 E 级[8]超算的研发上,异构架构正逐渐成为主流。许多数据中心也开始利用 GPU 来加速各类应用。因此如何利用 GPU 加速子图匹配算法是非常有意

① 国家重点研发计划(2016YFB0201305)和国家自然科学基金(61972377)资助项目。

② 男,1993 年生,博士生;研究方向:计算机系统结构;E-mail:mengke@ncic.ac.cn。

③ 通信作者,E-mail:tgm@ncic.ac.cn。

(收稿日期:2020-12-09)

义而且重要的。然而在 GPU 上部署子图匹配算法仍然有着巨大的挑战，是因为图数据结构非常稀疏，且不同的图数据访存模式差异巨大，采用单一的静态优化方法往往顾此失彼[10]，如何设计对 GPU 友好的图数据结构，针对不同情况选择合适的 GPU 优化策略显得至关重要。

虽然关于子图匹配算法的研究有很多，但大多数都是针对 CPU 算法的研究。解决子图匹配问题最简单的方法是穷举搜索法，但搜索空间随着图的节点规模呈指数型增长，即使图规模很小，计算时间也是不可接受的。文献[11]提出了一种回溯算法，可以大幅减少搜索空间。后来，其他研究者在此基础上提出了 VF2[12]、QuickSI[13]、GraphQL[14]、GADDI[15]等算法。Ullmann 算法[11]并未定义查询顶点的匹配顺序。VF2[12]从一个随机顶点开始，选择与已经匹配的查询顶点相连的下一个顶点。通过利用顶点标签频率的全局统计，QuickSI[13]提出了一种匹配顺序，该顺序将尽可能早地访问具有低频率顶点标签。TurboIso[16]将查询图重写为 NEC 树，该树将与同时具有相同邻居结构的查询顶点匹配，在一定程度上提高了匹配效率。Spath[17]则提出了一种并行子图匹配算法，将查询子图分解为两层高的树，称之为 twig。这些算法用不同的剪枝规则和匹配顺序来提升性能，但是这些算法仍然太慢，且只能处理具有数千个顶点的图。

近些年来，利用 GPU 处理稀疏问题逐渐成为一种充满潜力的方案，因此也有一些研究者试图在 GPU 上部署子图匹配任务。GpSense[18]根据 GPU 的特点设计了对 GPU 友好的数据结构和优化策略来加速子图匹配。GpSM[19]则基于图拆分的方法，设计了一些剪枝算法减少搜索空间，但是其算法依赖人工指定的参数。GSM[20]是一个基于 Gunrock[9]框架的子图匹配算法，它将传统图计算的原语扩展到图挖掘的领域。本文提出一种基于 GPU 的顶点预剪枝子图匹配算法（GPU-based vertex-pruning subgraph matching，GVSM），GVSM 基于图拆分的子图匹配算法，利用高效的剪枝规则和 GPU 的高并发来提升性能。和现有的子图匹配算法进行性能对比，GVSM 显示了显著的性能优势。在已有研究的基础上，本文的主要贡献有以下 3 个方面。

（1）提出了黑名单算法以及调度顺序决定算法。通过快速的预处理，提前大幅减少搜索空间，平衡了并发度和搜索效率。

（2）GVSM 将高并发的匹配过程卸载到 GPU 上执行，而回溯和连接过程则放在 CPU 上执行。设计了流水线处理方案让 CPU 和 GPU 协同处理子图匹配任务，并能够动态地负载均衡。

（3）和现有的子图匹配算法进行了性能对比，详细分析了优化的有效性，优化后的算法显示出显著的性能优势。

1　子图匹配算法概述

1.1　算法定义

给定一个数据图 $D(V_d, E_d, L_d)$ 和模式图 $P(V_p, E_p, L_p)$，其中 V 指图中的点集，E 指图中的边集，L 指节点的标签。子图匹配算法指在 D 中查找所有 P 的实例。即对 $v_p \in V_q$，存在 v_d 与之对应。因此子图匹配任务通常接受两份输入数据，一个是数据图，另一个是模式图，一般数据图较大而模式图较小。例如，如图 1 所示，在给定的数据图中，要查找的模式图为由标签{A, B, C, D}构成的子图。

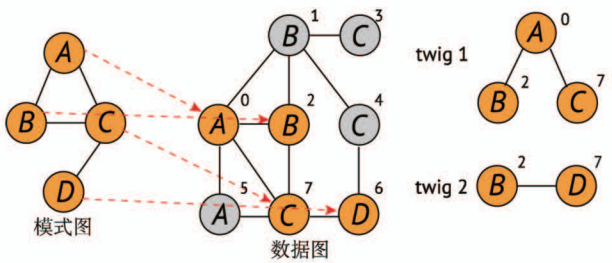

图 1　子图匹配算法与并行化

子图匹配算法的两大挑战是：（1）产生大量的中间结果，对存储和检索的压力巨大；（2）由于图数据本身的非规则性，负载不均衡问题十分严重，这对 GPU 这类核数多但是单核弱的平台带来了巨大挑战。

而目前国内外针对子图匹配问题的解决方案没有很好地注意到匹配过程可以剪枝的地方，导致做了许多冗余的搜索，浪费了算力，同时缺少针对

GPU 这类平台进行系统性的优化。

1.2　查询图拆分优化

Spath[17]是对查询图进行拆分的子图匹配算法,它将查询子图分解为 twig(一个两层高的树),如图 1 所示。在拆分完模式图之后,算法执行扩展-连接过程。在扩展阶段,算法按一定顺序匹配每一个拆分出来的 twig,即遍历数据图中每个点及其一阶邻居,查询其是否能匹配当前 twig。在连接阶段,将每个 twig 匹配到的待选点集进行笛卡尔积操作,将这些分割开来的待选点组合成完整的子图。基于图拆分子图匹配算法过程可以抽象成 2 个步骤。第 1 步骤只匹配单独的 twig (本文中称为扩展操作,expand),它只检查单个点在数据图中的匹配关系,而不检查子图的拓扑关系;第 2 步骤检查生成的子图的拓扑关系是否满足模式图的拓扑关系(本文中称为连接操作,join),它需要对子图的整体连通性进行检查。

1.3　基于前缀树的压缩优化

为了存储产生的大量的中间结果,必须要对其进行数据压缩,来减少其消耗的存储空间。由于产生的中间结果有许多的重合部分,因此可以采用前缀树的方式来压缩中间结果的共同部分,如图 2 所示。产生的 2 个中间结果$\{A, B, C\}$和$\{A, B, D\}$有着相同的部分$\{A, B\}$,因此可以存储到同一个根上。将这些中间结果称为图的一个嵌入。

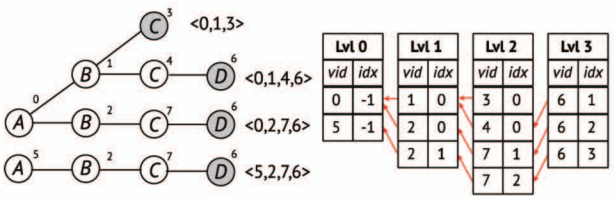

图 2　前缀树存储子图

前缀树的每一部分保存 2 个值$\{vid, idx\}$。其中 idx 表示该嵌入的前序点的索引,vid 表示该嵌入的当前节点的标签。而前缀树的每一层所存储的节点的标签都是相同的。前缀树将随着子图匹配算法的过程进行构建,子图匹配过程中每匹配一个 twig 就会增加前缀树的一层,由于相同的前缀总是只需保存一份,相比于直接存储子图可以节省大量的存储空间。但是在连接过程中,整个子图需要还原到原始的结构。此时可以从前缀树的叶子节点开始沿着 idx 指针进行遍历直到抵达根节点,此时路径上的节点就构成了当前迭代下已经匹配的子图。

2　子图匹配算法优化

本节将详细介绍子图匹配的算法优化,包括缩小搜索空间、搜索时剪枝优化、调度顺序优化,这些算法能够显著减少搜索空间,并减少需要存储的子图的数目。

2.1　缩小搜索空间

为了减少子图匹配过程中产生的中间结果,本文设计了黑名单算法来提前排除数据图 G 中的部分点和边。如图 1 所示,顶点 v_1 虽然有着和模式图匹配的标签 B,但是它没有标签为 D 的孩子节点,因此在后序的匹配中必然不能形成有效匹配。如果将这个子图拖到最后才排除,那么势必会浪费优先的计算资源和存储空间。因此在开始子图匹配算法之前,先对子图的部分节点进行标记,使其不参与后序的匹配过程。对子图的标记通过位图(Bitmap)来完成,每一位代表该节点是否被标记为被排除的点。这一部分可以在预处理完成。GPU 在执行匹配的过程中,首先读取该位图中的信息,如果某个线程发现当前节点被位图标记为应当排除的节点,那么就直接跳过该节点的匹配,从而减少搜索时间。

最初位图中所有位全部初始化为 0,此时所有的顶点都是合法的。黑名单算法将迭代进行筛选,在上一轮被加入黑名单的节点在本轮迭代中不会参与统计,因此会影响它的邻居节点,使得更多的节点被加入黑名单。该算法将遍历模式图 P 中的每个顶点,同时也将其邻居节点的标签进行遍历,记录其邻居标签的种类和数量,并将这些信息单独存入数组中。之后循环遍历数据图 D 中的每个节点,记录当前节点 v 的邻居节点的标签数目和种类。随后查询 P 中是否存在顶点 u,其邻居节点的状态能够匹配节点 v。若不能匹配,则将 v 加入黑名单。这说明节点 v 无论如何都无法和模式图中的某个节点进行匹配,故可以提前剪枝。这个算法由于不需要存储中间结果,每次只需要遍历一次边集即可完成标记,

因此开销非常低，第 4 节将详细分析该算法的开销。

在 1.2 节中，介绍了基于图拆分的子图匹配算法，它将模式图 P 拆成两层高的树（twig）来和数据图中的节点进行匹配，这意味着在匹配过程中不仅要匹配节点自己的标签，也要匹配节点的一阶邻居的标签。黑名单算法也基于类似的规则，如果数据图中的某个节点 v 不能匹配模式图中拆分出来的任意一个 twig，那么这个节点就是不合法的，可以提前筛除。而且筛除节点 v 将影响到它的所有邻居节点，这将导致它的某个邻居 u 从合法状态变为不合法状态，是因为 u 的合法性可能依赖于 v。例如节点 u 必须借助邻居 v 才能和模式图 P 中拆出来的 twig 进行匹配。若 v 变为不合法，那么 u 也不合法。因此黑名单算法是一个递归的过程，在每一轮迭代中，不断筛除不合法的节点。在最初的几轮迭代中，会标记大量的节点到黑名单中，但随着迭代的推进，标记的效率会大幅下降。因此黑名单算法可以只执行最初几轮迭代就停止，来平衡算法开销和剪枝效率。

2.2　剪枝优化

在搜索之前执行黑名单算法可以有效减少搜索空间。在搜索的过程中，也可以进行一些剪枝操作来提前终止搜索分支以减少内存和算力的开销。最基础的剪枝条件就是数据图中当前节点的标签和正在匹配的 twig 的根节点标签要保持一致。而进一步严格的检查则需要对它们的邻居节点进行一一匹配，这是一个耗时的步骤，它需要检索大量的边。由于输入数据常常存在着拥有大量边集的“超级节点”，因此数据图 D 中常常存在蕴含关系，即对于 D 中 2 个顶点 v 和 u，它们具有相同的标签且 u 的邻居集合是 v 邻居集合的子集，记做 $v \geqslant u$。形式化表达如下，如果 $v \geqslant u$，当且仅当 $L(v) = L(u)$ 且 $Adj(u) \subseteq Adj(v)$，其中 $L(v)$ 表示 v 的标签，$Adj(v)$ 是 v 的邻居顶点集。根据蕴含关系可以得知，当 $v \geqslant u$ 时，如果 v 匹配失败，那么此时 u 的匹配也必然失败。如图 3 所示，节点 v_1、v_2、v_3 都能匹配当前标签 B，且节点 v_2、v_3 的邻域是 v_1 的邻域的子集，因此 $v_1 \geqslant v_2$ 且 $v_1 \geqslant v_3$。由于 v_1 没有标签为 E 的邻居，因此 v_1 匹配失败，根据蕴含关系，不需要进行数据比对也可以推出 v_2、v_3 匹配将失败，因此之后的匹配可以跳过 v_2、v_3。可以利用该依赖关系对子图匹配的搜索过程进行剪枝。因此对于数据图中某一节点 u 和其在模式图中的匹配节点 u' 设立 3 条规则来实现该剪枝过程。

（1）顶点 u' 和 u 标签必须相同，且 $deg(u) >= deg(u')$（$deg(u)$ 表示顶点 u 的度）。

（2）顶点 u 的邻居顶点拥有的标签种类必须比顶点 u' 多。

（3）若 D 中存在 $v \geqslant u$，则要求 v 必须能匹配 u'。

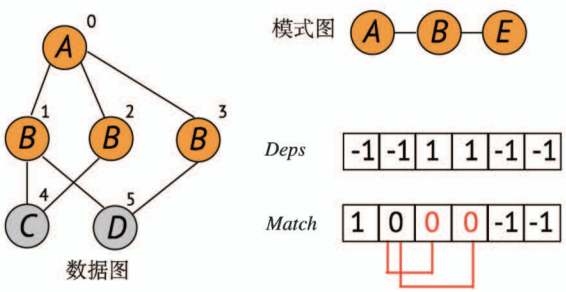

图 3　蕴含关系剪枝优化

其中前两条规则可以简单地判断当前顶点 u 是否有足够的邻居节点来完成后续的匹配，最后一条规则利用了蕴含关系来进行提前剪枝，其实现方式如图 3 所示。通过提前对数据图进行预处理可以得到一个依赖数组 $Deps$，表示节点 $Deps[i]$ 蕴含节点 i，即 $Deps[i] \geqslant i$。在匹配过程中，维护一个匹配数组 $Match$，其中 $Match[i]$ 表示节点 i 可以在模式图中得到一个合法的匹配，其中 0 表示不合法，1 表示合法，−1 表示尚未处理。通过查询 $Match[Deps[i]]$ 即可得知节点 i 是否需要被剪枝处理。为了减少开销，在本文中，只关注一层蕴含关系，即不关注嵌套的蕴含关系。即便如此，实验证明该剪枝规则能减少大量的不必要的搜索。

另一方面，蕴含关系也能够减少内存消耗，在 1.3 节中介绍了利用前缀树的方式存储子图匹配的中间结果的方法，前缀树需要存储在 GPU 的显存中，而这一部分的存储空间往往较少，即使使用了前缀树的方式进行了压缩，子图匹配产生的大量的中间结果仍然是一个巨大的压力。特殊地，若 $v \geqslant u$

且 $u \geqslant v$,即 v 和 u 的邻居节点相同,但是占据着前缀树不同的空间,可以把这些节点合并到一起,以链表的方式存储到一个前缀树分支里,在遍历该前缀树的时候只需处理一次,来代替被合并的链表里的所有节点。在匹配结束后,可以将这些节点依次替换,将其还原,可以节省前缀树存储的空间。

2.3　最优调度顺序

在搜索过程中存在着大量的冗余搜索。这是因为一些搜索分支最终无法匹配模式图,但是在搜索过程中由于无法判断是否能够形成匹配,不得不将其存储到接下来的迭代中。事实上在数据图中顶点的信息量存在着差别,信息量越大的节点匹配越严格,可以更早地排除没有潜力的节点[21]。如图 1 所示,查询图 P 和数据图 D 中,如果不考虑任何剪枝优化,假设匹配顺序为 $\{C, A, B, D\}$,那么在第一轮迭代中数据图有 3 个 C 符合标签,因此将需要存储 3 个中间结果,而若匹配顺序为 $\{D, B, A, C\}$,那么第一轮迭代只需存储一个中间结果。因此,对于数据图里顶点的匹配顺序将影响搜索效率。为了减少中间结果的产生,尽量将较难匹配的节点或者匹配结果较少的节点提前进行匹配,来减少匹配过程中产生的中间结果。对于一个模式图中的节点,它的邻居节点数目和标签的种类越多,其匹配的难度越高。同时它在数据图中的直接匹配越少,则预期产生的中间结果数目就越少,采用 $g(v)$ 来衡量模式图中节点 v 的预期产生中间结果的多少,可设 $g(v) = (freq(v) - blacklist(v))/(deg(v) \times labels(Adj(v)))$。其中 $freq(v)$ 表示 v 的标签在数据图中出现的频度,$blacklist(v)$ 表示 v 的标签在黑名单中出现的频度,$deg(v)$ 表示 v 的节点度数,$labels(Adj(v))$ 表示 v 的邻居节点标签的种类数目。当 $g(v)$ 越高时,匹配 v 产生的中间结果就可能越多。因此调度顺序可以通过利用 $g(v)$ 来启发式地决定。

最优调度顺序生成算法实际上是最小生成树 prime 算法的变体。在最开始的一轮迭代里选择 $g(v)$ 值最小的节点,将其加入到已挑选集合 O 中。随后的每一轮迭代里,在剩下的节点里挑选一个与 O 联通且 $g(v)$ 最小的节点,在 GVSM 的实现中,可以通过优先队列来维护当前 $g(v)$ 最小的节点。对于子图匹配而言,实际上其搜索空间并不是整个数据图,而是受限于查询图的特征。通过选择最优的调度顺序,将每一轮迭代可能产生的中间结果控制在最小,该算法可以在预处理阶段进行。若模式图 P 中有 n 个节点,计算出每个节点的 $g(v)$ 值并以此为据排序,需要 $O(n \log n)$ 代价,故生成调度顺序的算法需要 $O(n^2 \log n)$ 的代价。由于 n 的数值一般很小,因此该算法带来的预处理开销可以忽略不计。

3　基于 GPU 的子图匹配系统的设计

针对 GPU 高并发、高带宽的特点,在第 2 节的基础上,实现了基于 GPU 的子图匹配系统 GVSM。GVSM 能充分利用异构架构的特点,协同完成匹配任务。3.1 节将介绍 CPU-GPU 的流水线设计,3.2 节将提出 GPU 和 CPU 之间的自适应负载均衡优化。

3.1　流水线设计

如第 1 节所述,子图匹配算法是一个需要大量搜索的算法,如果能利用 GPU 大量的线程进行搜索将能够极大地提升搜索效率。但 GPU 更适合做规则的运算,如果将整个子图匹配算法都放到 GPU 上,可能会导致严重的负载均衡问题,使得系统无法充分发挥 GPU 的高并发性。另外,由于 GPU 的显存大小限制,子图匹配的搜索过程产生的大量中间结果如果保存在 GPU 显存内,则可能面临显存溢出的问题。因此,本文采用 CPU 和 GPU 协同计算的模式进行子图匹配,利用 GPU 执行每轮的计算任务,将 GPU 计算时产生的大量中间结果和候选子图保存至 CPU 主存上克服 GPU 的内存溢出问题,同时 CPU 的多线程技术也能为 GPU 分担一些简单的计算任务,使得两者能够协同计算。

在第 1 节中,本文介绍了基于扩展-连接的子图匹配算法。如图 4 所示,在 GPU 的实现中,扩展过程考虑的是子图和顶点/边的关系,它在已经找到的中间结果上,通过遍历边界点的邻居节点来产生新的待选子图,因此逻辑比较简单。而 GPU 擅长利用大量线程做一些简单的任务,这样的特性适合用来执行扩展操作,GPU 只需要输入前缀树的一部分数据就可以进行扩展操作。由于 GPU 的内存容量小,

而 CPU 端的内存容量往往远大于 GPU 内存，因此本文通过借助 CPU 内存来保存中间结果。对于连接操作而言，其处理的是子图内部的关系，这相比于扩展操作往往更加复杂，它需要按照前缀树的索引将整个待选子图还原，这不仅需要复杂的逻辑也需要大量的非连续内存访问。而 CPU 擅长处理逻辑复杂的操作，因此本文利用 CPU 的多线程技术来处理连接过程。这就需要 CPU 和 GPU 进行协同计算，利用数据移动解决内存空间不足的问题。

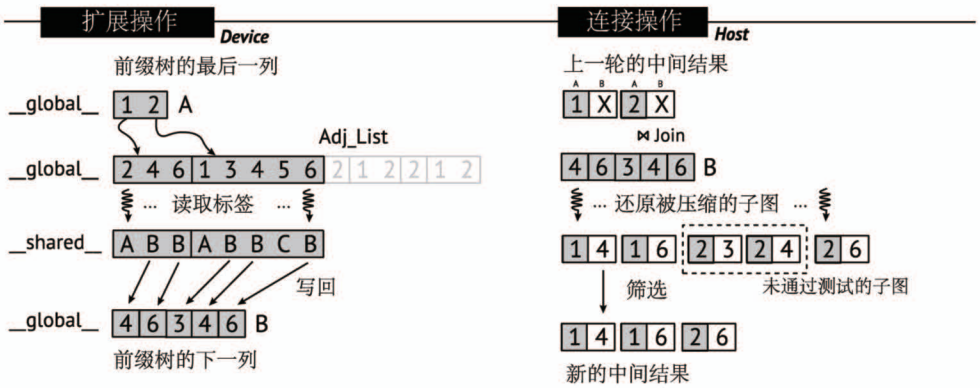

图 4　GPU-CPU 协同处理的 Expand-Join 操作

为了充分发挥 GPU 和 CPU 各自的优势，本文设计了流水线方案来隐藏 CPU 和 GPU 之间数据传输。在迭代开始时，CPU 端生成初始待匹配的集合。为了让 GPU 执行扩展操作，系统需要将数据通过高速串行计算机扩展总线（peripheral component interconnect express，PCI-E）传输到 GPU 显存。GPU 在接收到数据之后，对当前的子图进行搜索，生成候选匹配的中间结果集合。接着需要将这些中间结果从 GPU 设备端拷贝到 CPU 主存，让 CPU 执行连接操作检查这些子图，CPU 将合格的子图保存在主存上。在下一轮迭代中，将部分数据继续拷贝到 GPU 上重复执行上述步骤。当迭代结束时，所有挖掘到的子图将保存在 CPU 主存上。整个流程简化后可以分为以下 4 个步骤。

（1）数据拷贝到 GPU。

（2）GPU 执行扩展操作。

（3）数据拷贝回 CPU。

（4）CPU 执行连接操作。

按这样的 4 个步骤，系统可以依次串行执行整个图挖掘过程，在第 i 轮时，按串行思路需要依次做完步骤（1）~（4）之后才会执行第 $i+1$ 轮迭代。但这样的方式使得当 GPU 在执行扩展操作时，由于 CPU 还没有接收到数据无法处理而处于等待状态。GVSM 通过流水线的方式，让 GPU 和 CPU 可以同时进行计算，同时重叠数据传输和计算，如图 5 所示。

为防止内存溢出，系统每次仅拷贝本轮的一部分数据至 GPU 进行扩展操作，因此本文对前缀树进行了切分，每次计算时只传输前缀树中一层的一部分数据到 GPU 中参与扩展，这是因为 GPU 执行扩展操作时会消耗大量内存。由于采用了流水线设计，GPU 中需要同时保存第 $i-1$、i、$i+1$ 轮迭代的数据，与之对应 CPU 也需要同时保存第 $i-1$、i、$i+1$ 轮迭代的数据，因此需要将存储空间分为 3 个缓冲区，用来保存 3 个相邻迭代的数据。对于 GPU 来说，同一时刻，除了正在处理第 i 轮的数据以外，还可以同时保存下行的 $i+1$ 轮的数据和上行的 $i-1$ 轮迭代的数据。如图 5 所示，在数据交换的过程中，需要依据对应的内存缓冲区拷贝数据，得益于英伟达 GPU 的双拷贝（dual-copy）引擎，第 $i-1$ 份扩展后的候选数据的设备到主存传输（device to host，D2H）（步骤（3））和第 $i+1$ 份待扩展数据的主存到设备传输（host to device，H2D）（步骤（1））可以进行数据传输重叠，与此同时 GPU 上第 i 轮的扩展操作和 CPU 第 $i+2$ 轮的连接操作也可以同时进行。利用这样的流水线模式，系统能够充分利用 CPU 和 GPU 的计算能力，重叠了计算也节省了数据的传输时间。

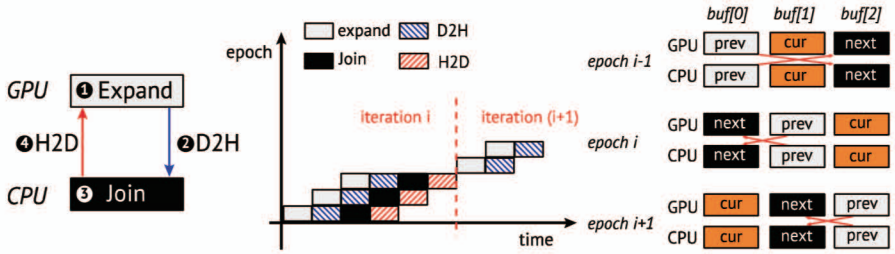

图 5　基于流水线的子图匹配算法流程

3.2　自适应负载均衡

由于子图匹配在扩展操作过程中会产生不可控数量的中间结果,同时产生的中间结果随着迭代的深入而逐渐增多,如果每次流水线固定拷贝一整层数量的节点到 GPU,可能会导致实际需要处理的边集相差较大。在子图匹配的早期阶段,需要的内存量很小,整个扩展-连接过程都可以在 GPU 上直接完成,此时拷贝到 CPU 上反而导致了额外的开销。由于不是所有的情况下使用 GPU 都能起到加速效果,比如在数据量较少的时候将数据拷贝到 GPU 上反而会使性能下降。因此本文设计一种动态负载均衡方法,通过设置一个阈值 th,当前缀树的某一层数量超过这个阈值时,才将连接操作通过流水线的方式交由 CPU 处理,否则所有的流程都在 GPU 上处理。

为了保证流水线能更好地重叠计算,需要 GPU 和 CPU 的负载尽可能相同,否则会导致算力的浪费。在算法的前期,子图匹配过程将产生大量的中间结果,而且增长速度非常快。GVSM 采用分块的方法进行处理,每次从前缀树中拷贝 $chunk\_size$ 个顶点到 GPU 中去,但是其个数将随着 CPU 的负载变动。由图 5 可知,为了保证重叠率,其块大小 $chunk\_size$ 需要保证 GPU 处理第 i 轮扩展操作的时间要尽可能和 CPU 处理第 $i-2$ 轮的连接操作时间相同。但是由于不同的节点的邻居节点数量不一样,有的节点可能连接着数百万的节点,而有的节点可能只有数个节点。那么每次拷贝固定大小的块到 GPU 上仍然会导致严重的负载不均衡问题,从而影响流水线的运行。因此根据出度数作为每个节点的权重,通过提前保存当前层节点权重的前缀和来进行任务块的划分,通过二分的方法确定当前块的大小,保证每次拷贝到 GPU 上的任务块需要处理的边的数目和连接操作需要处理的边的数目满足线性关系,即 $chunk\_size \times avg\_degree = k \times subgraph\_size$。其中 $subgraph\_size$ 是第 $i-2$ 次迭代需要连接操作的子图的数量,avg_degree 表示当前块的平均出度数。而 k 值需要通过多次实验采用回归训练确定。通过这种方法能够以几乎忽略不计的开销保证每次迭代需要处理的任务块有相近的处理时间。

另一方面,扩展操作通过接受前缀树中的一段节点,以及模式图中的一个 twig,在数据图中寻找对应的匹配,这一过程通常在 GPU 上进行。而在之后的连接操作,则将新匹配出的节点和之前匹配出来的中间结果进行笛卡尔积操作,来判断所形成的新的子图是否符合要求。在此过程中,每次只添加一个节点。事实上,子图匹配也可以在一次内核调用中处理 2 个 twig,即同时匹配 2 个节点,相当于增加了搜索的步长。尤其在算法后期,节点数目较少时,可以增加搜索的步长,加快优化。

4　实验评测

4.1　实验环境和基准测试程序

在本节中,使用本文提出的技术构建了一个完整的子图匹配算法 GVSM,并选择 6 个真实数据集作为数据图。如表 1 所示,这 6 个数据集的顶点规模在十万至百万级别,边规模在百万至千万级别。其中 nv 表示对应数据集中顶点的数量,ne 表示对应数据集中边的数量。$Ave\text{-}degree$ 为数据图中每个顶点的度的平均数。它在一定程度上反映了图的稠密程度,一般来说,平均度数越高说明图越稠密,反之则说明图越稀疏。这些真实数据集普遍具有无尺度(scale-free)的特点,即每个顶点拥有的边的数量呈幂律分布,这导致部分节点拥有大量的边与之相连,

而绝大数顶点只有少数边相连。通过预处理，对实验数据分配随机数量的标签，并选择各种不同规模的模式图进行子图匹配任务。

表 1　GVSM 测试的数据集

数据集	*nv*	*ne*	*Ave-degree*	*labels*
enron	69 244	549 216	4	47
gowalla	196 591	950 327	5	12
mico	100 000	2 160 312	22	29
patents	2 745 761	27 930 818	10	37
road _ central	14 081 816	33 866 826	2	41
orkut	3 072 441	234 370 166	76	138

实验运行平台的服务器为搭载架构为 Intel Xeon 的处理器并装有 Nvidia 的 tesla P100 显卡，配备 16 GB 显存。操作系统为 ubuntu 16.04，内存为 94 GB，CUDA 版本为 CUDA 10.0。每次测试时间为 10 次运行的平均时间。选择近些年发布的开源的基于 GPU 的子图匹配算法 GpSM[19] 和 GSM[20] 作为参考对象。GpSM 针对 GPU 结构做了优化，加速了子图匹配算法的计算流程。GSM 则是基于 Gunrock[9] 的基础上所构建的子图匹配算法。其实现也都是基于图拆分的子图匹配，分为扩展和连接两步。而 Gunrock 是 GPU 上非常流行的图算法库，有着优化良好的库函数来处理 GPU 上的稀疏数据。

4.2　性能对比

为了验证算法的高效性，在 6 个真实图数据上，通过和开源的已发表的工作 GpSM 和 GSM 进行单节点的性能对比，以测试算法执行时间与查询图中顶点数量之间的关系，结果如图 6 所示，从图 6 可以发现，GpSM、GSM 和 GVSM 的处理时间均随着查询图节点的上升而上升。因为模式图节点的增加将导致需要匹配的边集呈指数上升。但是相比于 GSM 和 GpSM，GVSM 有着更快的处理时间以及更平滑的曲线。这说明 GVSM 更有潜力处理规模更大的查询。由于 GVSM 有高效的剪枝优化手段，去掉许多冗余的没有必要的搜索分支，这使得整个匹配过程中需要载入的数据和需要的计算量大大缩小。

在查询图顶点较少的时候，GVSM 的性能略低于 GSM 和 GpSM，这是因为 GVSM 有额外的预处理时间，黑名单算法的额外开销以及流水线的在启动阶段没有办法重叠 PCI-E 总线拷贝导致的。由于查询图较少的时候再需要进行的搜索并不多，优化带

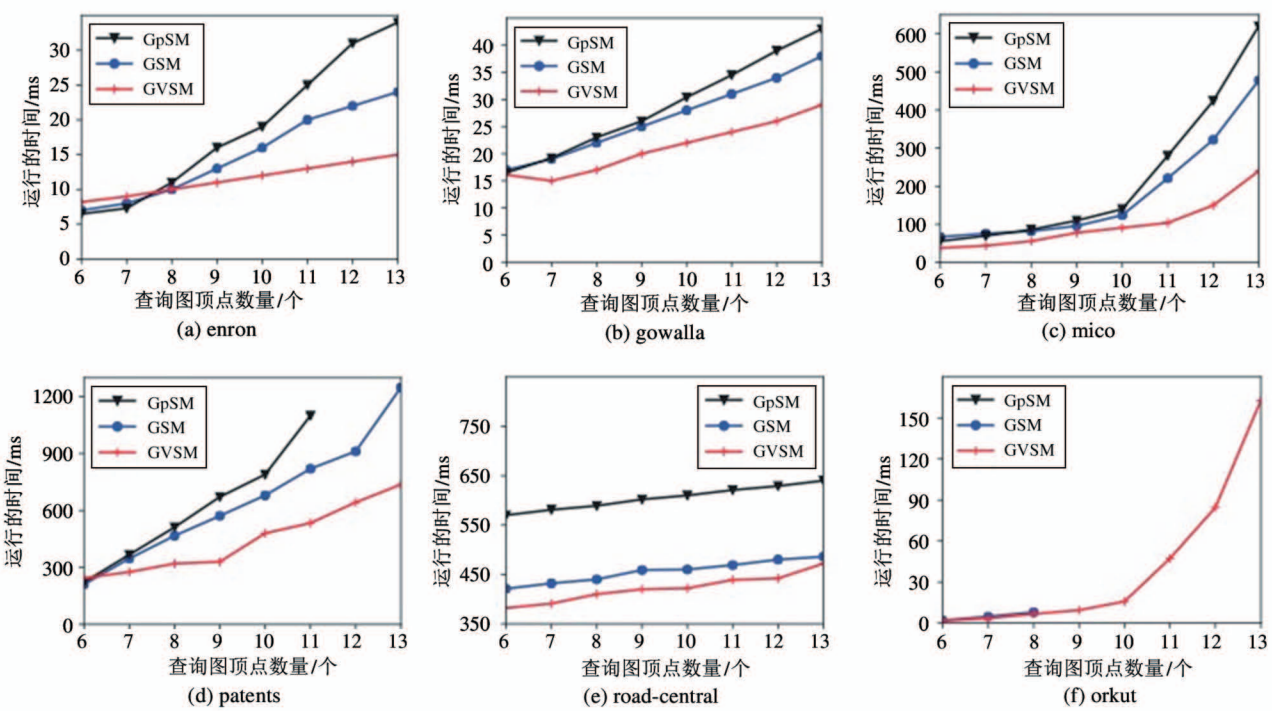

图 6　实验结果对比

来的性能提升不足以完全掩盖其带来的开销。因此 GVM 更适合处理查询图较大的子图匹配任务,不适合处理查询图较小的子图匹配任务。但随着查询规模的增长,GVSM 性能很快就超过了 GSM 并逐渐拉开了差距。除此之外,如图 6(f)所示,GSM 和 GpSM 在边数较多的大图 orkut 上无法处理较大的查询图,GSM 在查询图的顶点数量超过 8 个时,会导致需要存储的中间结果数量超过 GPU 的显存而导致程序异常退出。但是 GVSM 由于采用了前缀树对中间结果进行了压缩,并采用流水线的方式,将中间结果转存到 CPU 上,并利用 CPU 分担了一部分不适合并行处理的计算工作,这使得 GVSM 能处理的问题规模大幅增加。

4.3　算法剪枝优化性能分析

在第 2 节中,介绍了 3 种缩小搜索空间的剪枝优化手段,分别是黑名单算法、蕴含关系剪枝、以及调度顺序优化。在 GPU 上实现这 3 种优化,并和没有开启这 3 种优化的原始程序在 6 个真实的图数据上进行性能上的对比,来验证本文优化工作的有效性。

图 7 展示了黑名单算法的有效性及其带来的预处理的开销。黑名单算法通过提前的预处理操作剔除掉一些不可能发展为最终子图的节点,并将它们提前标记,这种筛除并不是基于简单的标签匹配而是依赖更深层次的拓扑关系。从图 7 左图可以看出,利用该算法在多个数据集上能提升 15% ~50% 的性能,对于标签数量相对较多的图性能提升更显著。图 7 右图则展示了该算法在不同数据集上的预处理开销占总运行时间的比例,从中可知在所有的数据集上开销都少于总运行时间的 17%。对于规模较小的数据集 enron、gowalla、mico 而言,由于总运行时间较短,因而黑名单算法开销占比较高,而对于规模较大的数据集 patents、road-central 和 orkut 而言,其开销低于 7%。可以推测在更大的数据规模

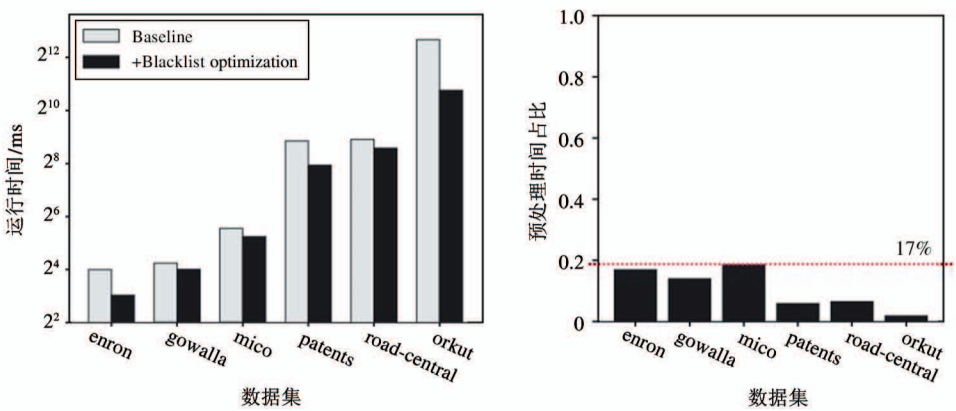

图 7　黑名单算法的效果和开销

下,黑名单算法的开销会更低。因此黑名单算法更适合规模较大的数据集而不适合小规模数据集。考虑到其带来的性能提升,该算法的额外开销是可接受的。

图 8 展示了搜索中利用蕴含关系剪枝在 6 个数据集上的优化效果。通过蕴含关系剪枝,GVSM 维护一个额外的数据结构来记录节点间的蕴含关系,并据此对搜索过程进行提前剪枝。尤其是在进行黑名单算法筛除了大量边和顶点以后,将产生更多的蕴含关系,这进一步提升了剪枝效率。该优化对中小规模的图能提供约20% 的性能提升。而对部分大

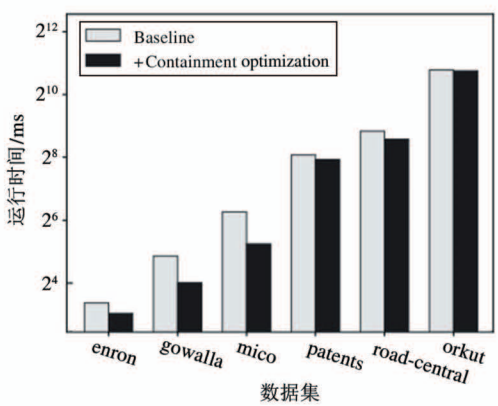

图 8　剪枝优化效果

图数据性能提升不高，这是因为节点数目增多，边的连接方式变多，这导致蕴含关系较少，同时需要维护的数据结构开销也逐渐增大，因此带来的性能提升无法掩盖需要维护额外数据结构带来的开销。

图 9 展示了不同的调度顺序对性能的影响，在这里显示了 mico 图数据在进行一个 4 节点的查询图的子图匹配任务时采用不同的匹配顺序而导致的性能差异。在全部的 6 种顺序中，以 *C-A-B-D* 的顺序进行匹配性能最好，比最坏的情况 *B-A-D-C* 性能要高出 1 倍。这是因为提前进行严格的匹配有助于控制子图匹配过程中产生的中间结果，GVSM 采用启发式方法对匹配顺序进行排序，在这种情况下找到顺序也是 *C-A-B-D*，与枚举出来的最优调度顺序相符合。在 6 个真实图数据的测试中，在 87% 的情况下，GVSM 能够根据启发式算法选中最优的调度顺序。

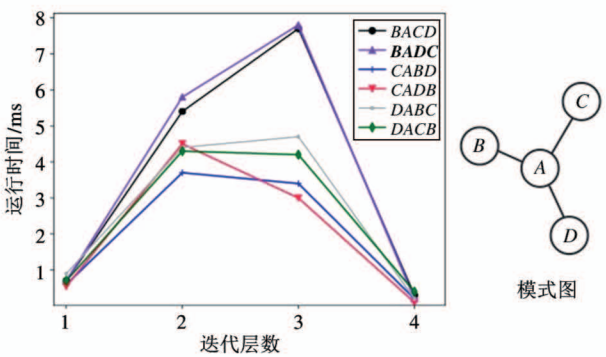

图 9　不同调度顺序对性能的影响

4.4　系统实现优化性能分析

第 3 节介绍了 2 种针对 GPU 的子图匹配算法在实现上的设计，分别是利用流水线重叠计算和数据移动，以及负载均衡任务划分。在 GPU 上实现这 2 种优化，将没有和开启这 2 种优化的原始程序在 6 个真实的图数据上进行性能对比，来验证本文设计的有效性。

图 10 展示了流水线优化在 6 个真实图数据中的性能。从图中可以发现，对数据规模较小的图而言，使用流水线的方式与不使用流水线进行重叠差别不大。这是因为在此时图的数据可以直接在 GPU 中存下，因此流水线优化更适合较大规模的图数据。GVSM 在处理小规模数据时，根据设立的阈值 *th*，所有的计算都在 GPU 内完成，因此不需要通过 PCI-E 总线来交换数据。而对数据规模较大的图而言，不进行流水线的重叠会导致 GPU 因等待数据而空转，所以执行时间较长。在这种情况下开启流水线优化可以将性能提升 2 ~ 2.8 倍。而直接在 GPU 上处理这个规模的数据，可能会由于查询图节点过多而导致无法在 GPU 中存下大量的中间数据而任务失败。利用流水线将 GPU 和 CPU 结合起来可以高效地处理更大的数据集，性能也不低于基于纯 GPU 的实现 GSM。

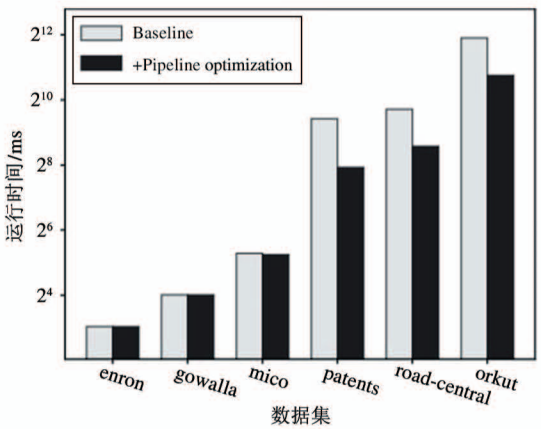

图 10　流水线优化效果

图 11 展示了采用动态块大小方法的负载均衡优化在 Mico 图数据上执行的每层迭代的时间。由于每次迭代只处理一块数据，采用静态的方法每次选取固定数量的节点，而采用动态方法每次选择和 CPU 上相近的负载。可以发现，在算法初期，采用静态方法的版本有大量的中间结果需要确定，因此会出现一些关键迭代层，在这些迭代中，会搜索到大量的边集，从而决定了整个算法的运行时间。由于负载不均衡，导致在关键迭代层计算和负载不匹配，无法和前后两层的任务进行有效的重叠，而采用动态方法，对每次拷贝进 GPU 的数据进行控制，保证前后两次迭代的任务差距不会过大，因此重叠效果更好。另一方面，从图中可以发现，在算法后期，由于中间结果变少，需要搜索的边也逐渐变少，导致后面的迭代需要处理的节点数量变少，此时数据通过 PCI-E 总线传输的延迟逐渐成为性能的瓶颈。因此采用一次处理 2 条边而不是 1 条边的方法可以大幅

减少迭代层数,从而减少额外的数据传输开销。

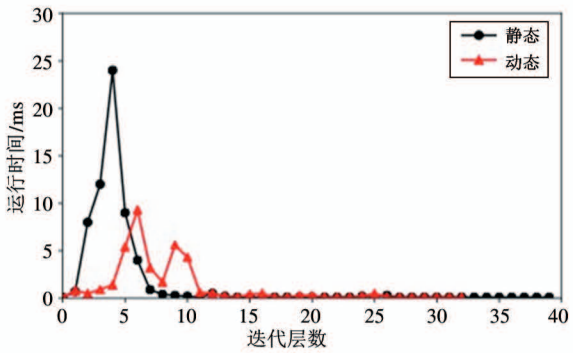

图 11　负载均衡优化效果

5　结　论

本文对单节点的子图匹配算法及其优化技术进行了介绍,并在查询图拆分的算法上提出了一种基于 GPU 的子图匹配方法 GVSM。在该算法上设计了黑名单算法,通过提前筛除不可能构成最终子图的节点,减少了搜索需要的计算量和内存空间,同时通过蕴含关系进行剪枝,并通过决策匹配顺序,大幅减少了冗余的搜索分支。在系统优化上,采用 GPU 与 CPU 协同计算的方式,利用流水线优化技术,重叠了计算与数据移动,并利用 GPU 的高带宽高并发来完成待选点集的发掘。同时采用了动态负载均衡算法,保证 GPU 与 CPU 之间的工作量的平衡,保证了流水线的稳定。最终的实验结果表明,本文方法能显著提升子图挖掘任务的性能,相比同类型的实现,可将处理性能提升 2 倍,并能应对更大规模的数据集。本文方法具有一定的通用性,这些优化技术也能在其他的图挖掘应用上使用。在未来的工作当中,将会把本文的优化方法扩展到其他类型的图挖掘算法中,并把 GVSM 扩展到大规模 GPU 集群上,以处理更大规模图挖掘任务。

参考文献

[1] MILLER E. An introduction to the resource description framework[J]. *Bulletin of the American Society for Information Science and Technology*, 1998, 25(1): 15-19

[2] 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, Indianapolis, USA, 2010: 135-146

[3] LOW Y, GONZALEZ J E, KYROLA A, et al. Graphlab: a new framework for parallel machine learning [EB/OL]. http://arxiv.org/ps_cache/arXiv/pdf/1006/1006.4990v1.pdf: arXiv, (2010-06-25), [2020-12-09]

[4] TIAN Y, 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

[5] LI Z, CUI Z, WU S, et al. Fi-GNN: modeling feature interactions via graph neural networks for CTR prediction [C]// Proceedings of the 28th ACM International Conference on Information and Knowledge Management, Beijing, China, 2019: 539-548

[6] LI Z, CUI Z, WU S, et al. Semi-supervised compatibility learning across categories for clothing matching[C]// 2019 IEEE International Conference on Multimedia and Expo, Shanghai, China, 2019: 484-489

[7] CUI Z, LI Z, WU S, et al. Dressing as a whole: outfit compatibility learning based on node-wise graph neural networks[C]// The World Wide Web Conference, New York, USA, 2019: 307-317

[8] LIN H, ZHU X, YU B, et al. Shentu: processing multi-trillion edge graphs on millions of cores in seconds[C]// SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, USA, 2018: 706-716

[9] WANG Y, DAVIDSON A, PAN Y, et al. Gunrock: a high-performance graph processing library on the GPU [C]// Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Barcelona, Spain, 2016: 1-12

[10] MENG K, Li J, TAN G, et al. A pattern based algorithmic autotuner for graph processing on GPUs[C]// Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming, Washington, USA, 2019: 201-213

[11] CORDELLA L P, FOGGIA P, SANSONE C, et al. A (sub) graph isomorphism algorithm for matching large graphs[J]. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2004, 26(10): 1367-1372

[12] SHANG H, ZHANG Y, LIN X, et al. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism[J]. *Proceedings of the VLDB Endowment*, 2008, 1(1): 364-375

[13] HE H, SINGH A K. Graphs-at-a-time: query language and access methods for graph databases[C]// Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, Vancouver, Canada, 2008: 405-418

[14] SUN Z, WANG H, WANG H, et al. Efficient subgraph matching on billion node graphs[C]// The 38th International Conference on Very Large Data Dases, Istanbul, Turkey, 2012: 788-799

[15] ZHANG S, LI S, YANG J. GADDI: distance index based subgraph matching in biological networks[C]// Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, New York, USA, 2009: 192-203

[16] HAN W S, LEE J, LEE J H. 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, USA, 2013: 337-348

[17] ZHAO P, HAN J. On graph query optimization in large networks[J]. *Proceedings of the VLDB Endowment*, 2010, 3(1-2): 340-351

[18] TRAN H N, CAMBRIA E. Gpsense: a GPU-friendly method for commonsense subgraph matching in massively parallel architectures[C]// International Conference on Intelligent Text Processing and Computational Linguistics, Konya, Turkey, 2016: 547-559

[19] TRAN H N, KIM J, HE B. Fast subgraph matching on large graphs using graphics processors[C]// International Conference on Database Systems for Advanced Applications, Hanoi, Vietnam, 2015: 299-315

[20] WANG L, WANG Y, OWENS J D. Fast parallel subgraph matching on the GPU [EB/OL]. https://arXiv.org/pdf/2003.01527.pdf: arXiv, (2020-03-01), [2020-12-09]

[21] MAWHIRTER D, WU B. AutoMine: harmonizing high-level abstraction and high performance for graph mining [C]// Proceedings of the 27th ACM Symposium on Operating Systems Principles, Huntsville Ontario, Canada, 2019: 509-523

Optimizing GPU-based subgraph matching algorithm

MENG Ke[*,**], LIN Zhiheng[*,**], TAN Guangming[*]

([*] High Performance Computing Research Center, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190)

([**] University of Chinese Academy of Sciences, Beijing 100049)

Abstract

In order to solve the performance problem of subgraph matching, a graphic processing unit (GPU)-based vertex-pruning subgraph matching (GVSM) system is proposed. GVSM adopts a blacklist pruning algorithm and an order-selection to reduce redundant searching. By using a prefix tree, GVSM can compress the intermediate results, which reduces the memory footprint. Meanwhile, GVSM processes graphs using GPU cores while streaming topology data of graphs via PCI-E interfaces with a fully pipelined algorithm is designed to overlap the computing and data movement. A dynamic load-balance method is used to keep balanced workloads in GPU and center processing unit (CPU) in each iteration to reduce the waste of computing resources. The experimental results show that the proposed method can solve the subgraph matching task effectively and efficiently. GVSM has significantly outperformed the state-of-the-art work and has the ability to process larger dataset.

Key words: subgraph matching, graph mining, graphic processing unit (GPU), high performance, graph processing