计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(9): 2442−2462, 2026

Saturn: 面向 Multi-GPU 平台的高性能图模式挖掘系统

林志恒　孟　轲　谭光明
（中国科学院计算技术研究所　北京　100190）
（中国科学院大学　北京　100049）
（linzhiheng@ncic.ac.cn）

Saturn: High Performance Graph Pattern Mining System on Multi-GPU Platform

Lin Zhiheng, Meng Ke, and Tan Guangming
（*Institute of Computing Technology*, *Chinese Academy of Sciences*, *Beijing* 100190）
（*University of Chinese Academy of Sciences*, *Beijing* 100049）

Abstract　Graph pattern mining, as a core task for analyzing entity relationships, finds extensive applications in fields such as community detection and drug discovery. However, due to its non-polynomial time complexity, large-scale graph processing encounters challenges related to search space explosion and low computational efficiency. Existing GPU-accelerated solutions utilize parallel computing to enhance performance but often suffer from thread idleness caused by the power-law distribution of graph data and excessive CPU intervention in multi-GPU communication, which impede the full utilization of hardware capabilities. We present Saturn, a GPU-based graph pattern mining system, which overcomes the above bottlenecks through computation-communication co-optimization. At computation level, Saturn decouples the loading and querying phases of set operations, employs fine-grained parallel batch processing to dynamically adapt Hash and binary search algorithms, and optimizes memory access efficiency via shared memory caching, significantly improving thread utilization. At communication level, it establishes a CPU-free GPU communication model, leveraging high-bandwidth NVLink interconnect and unified memory mechanism for direct cross-device data access, while reducing redundant transmission and link contention through vertex caching and message aggregation scheduling. Experimental results demonstrate that Saturn outperforms state-of-the-art GPU-based graph mining systems on real-world datasets, achieving superior computational parallel efficiency and cross-device communication effectiveness for large-scale graph pattern mining tasks.

Key words　graph pattern mining; distributed graph processing; GPU; parallel computing; communication optimization

摘　要　图模式挖掘作为分析实体关系的核心任务，在社区检测、药物发现等领域具有广泛应用。但由于其非多项式时间复杂度，大规模图数据处理面临搜索空间爆炸与计算效率低下的挑战。现有基于 GPU 的加速方案虽利用并行计算提升性能，但受限于图数据幂律分布导致的线程空闲问题，以及多 GPU 场景下频繁的 CPU 介入通信开销，难以充分释放硬件算力。提出 GPU 图模式挖掘系统 Saturn，通过计算与通信协同优化突破上述瓶颈：在计算层面，将集合操作的装载与查询阶段解耦，采用细粒度并行批处理策略动态适配哈希与二分查找算法，并结合共享内存缓存优化访存效率，显著提升线程利用率；在通信层面，构建 CPU 无感的全 GPU 通信模式，利用 NVLink 高带宽链路与统一内存机制实现跨设备数据直接访问，

收稿日期：2025-06-26；**修回日期**：2026-04-10
基金项目：国家自然科学基金项目（T2125013, 62032023）
This work was supported by the National Natural Science Foundation of China (T2125013, 62032023).
通信作者：谭光明（tgm@ict.ac.cn）

通过顶点缓存与消息聚合调度降低冗余传输与链路竞争。实验结果表明，Saturn 在真实数据集上相比主流 GPU 图挖掘系统，实现了更高的计算并行效率与跨设备通信能效，为大规模图模式挖掘任务提供了高效解决方案。

关键词 图模式挖掘；分布式图处理；GPU；并行计算；通信优化

中图法分类号 TP338.6

DOI: 10.7544/issn1000-1239.202550501 **CSTR:** 32373.14.issn1000-1239.202550501

图作为一种灵活的数据结构，能够高效表示实体间的关系，已被广泛用于建模现实世界数据。图模式挖掘（graph pattern mining, GPM）任务旨在从给定的数据图 G 中找出所有与模式图 P 同构的子图，而数据图 G 的规模通常远大于模式图 P。这类图任务可挖掘隐藏的数据特征或实现特定模式匹配，在社区检测[1]、分子动力学[2]、药物发现[3]及异常检测[4]等数据分析领域具有广泛应用。然而，图模式挖掘问题目前尚无多项式时间解法，实际处理中需从数据图中每个顶点出发，迭代添加邻边并检查候选子图是否与目标图模式同构，这不仅需要遍历庞大的搜索空间，还需实例化大量的中间结果。

近年来，先进的图模式挖掘系统为提高搜索效率，开始采用以集合为中心的范式[5-7]。该范式的核心思路是充分挖掘模式图的拓扑信息，以此过滤掉无法形成有效最终匹配的中间结果。这类系统常将搜索过程表示为嵌套循环，每层迭代都通过由集合求交（∩）等集合运算构成的表达式进行形式化描述。例如，表达式 $S(v_2)=N(v_0)\cap N(v_1)$ 表示 v_2 的合法候选集（$S(v_2)$）必须是顶点 v_0 和 v_1 的公共邻居。以集合为中心的范式借助关系代数等价变换实现大量优化，例如 AutoMine[5] 通过估算集合计算的代价生成最优匹配顺序（matching order）以缩小搜索空间。GraphPi[8] 结合对称性破坏（symmetry breaking）规则减少冗余搜索。这些系统大多遵循模式感知范式（pattern-aware paradigm），将模式图的拓扑信息转化为集合表达式，通过以集合为中心的抽象分析，最大限度地寻找并复用公共元素。

随着图数据规模不断增长，对高效图模式挖掘的需求愈发迫切。GPU 凭借其大规模并行计算核心，以及高带宽内存（high bandwidth memory, HBM）提供的 TB 级访存带宽[9]，成为加速图模式挖掘应用的理想硬件选择。近年来，基于 GPU 的图模式挖掘加速工作不断涌现，其核心围绕任务并行与算子优化 2 个方面展开。其中，一些系统[10-11]采用 2 阶段任务并行策略将 GPU 线程组分配至不同顶点或边的搜索任务，线程组内并行执行集合操作以实现任务的高效映射。在算子优化方面，则通过 HashIndex[12]，BinarySearch[13] 等算法加速搜索过程中频繁调用的集合算子。但由于现实图数据的幂律分布特性导致集合求交计算量差异显著。现有系统多采用单一或粗粒度并行方案，导致 GPU 线程组在处理小度数顶点时线程的高空闲率问题，制约了 GPU 图挖掘系统的性能。

尽管借助 GPU 的高并行性可加速图模式挖掘算法，但单 GPU 有限的内存容量（如 V100 仅 32 GB）制约了其对大规模图数据的处理能力。为突破这一瓶颈，研究者转而采用图划分策略，将庞大的图数据分散至 Multi-GPU 平台进行处理，期望通过并行计算与内存资源整合提升处理能力。跨 GPU 的顶点邻接表分布导致边连接判断需频繁传输数据，通信开销极大。现有多 GPU 图挖掘系统[14-16]利用 CPU 主存扩展 GPU 处理数据的规模。这些系统需要 CPU 来管理 GPU 显存上的图分区数据驻留和驱逐[15]，或者依赖 CPU 执行显式的同步操作，让多个 GPU 并行处理分区内和分区间（inter/intra partition）的图挖掘任务[14]。然而，频繁的 PCIe 数据交换与 CPU 的深度介入，使得 GPU 计算频繁中断，造成系统 GPU 利用率低下且可扩展性不足。

为了应对上述挑战，本文提出 GPU 图模式挖掘系统 Saturn。Saturn 系统旨在通过计算和通信 2 个层面协同优化来提升图挖掘系统处理大规模图数据时的整体性能。在计算层面，我们将集合操作的加载与查询阶段解耦，通过细粒度并行将同层循环内的多个集合操作批处理，使线程组能利用空闲线程并行处理多个任务，同时根据顶点度数动态适配哈希与二分查找策略，并结合共享内存分层缓存优化访存效率；在通信层面，构建 CPU 无感的全 GPU 通信的图挖掘模式，利用 GPU 间 NVLink 高带宽链路和统一内存机制实现 GPU 直接访问远端图数据。通过高度数顶点缓存减少远程通信，并采用消息聚合分桶调度机制降低链路竞争与冗余传输。

本文的主要贡献包括 3 个方面：

1)详尽分析了 GPU 上现有图模式挖掘系统的并行优化方法的缺陷和性能影响。针对图挖掘任务高度不规则的计算特性,设计了细粒度并行任务调度机制,并结合混合算子策略有效解决了由于数据倾斜导致的线程利用率低下问题。

2)提出 CPU 无感的 Multi-GPU 图挖掘通信方案。该方案利用 GPU 内核直接管理跨设备数据传输,消除了传统模式中频繁的 CPU 干预与主从同步开销。通过引入数据缓存机制和消息聚合策略,进一步降低了冗余通信带来的链路竞争。

3)实现计算-通信协同优化的 GPU 图挖掘系统 Saturn,并在多种真实世界数据集上进行系统性的测试与验证。实验结果表明,Saturn 系统性能显著优于主流基于单 GPU 和多 GPU 平台的图挖掘系统,实现了更高的并行效率与跨设备通信效率。相比于基线系统,Saturn 展现出更优的强扩展性。

1 预备知识

1.1 GPU 架构介绍

在 GPU 架构中,计算模块和存储模块紧密耦合,共同支撑起高效的并行计算任务。如图 1 所示,GPU 采用单指令多线程(single instruction multiple thread, SIMT)的执行模型,以 Warp(通常包含 32 个线程)线程组为粒度,线程以锁步的方式执行同一指令流。多个 Warp 组成线程块(Block),而流式多处理器(streaming multiprocessor, SM)作为核心计算单元,可同时调度多个线程块执行计算任务,实现高效并行处理。另一方面,GPU 存储层次采用分层设计,全局内存(global memory)用于存放计算所需的各类数据,共享内存(shared memory)作为片上高速缓存,用于同一线程块内的数据共享。在各级存储层次中,L2 缓存不可编程,L1 缓存则和共享内存共享同一存储空间,可根据计算任务特性动态调整分配比例。

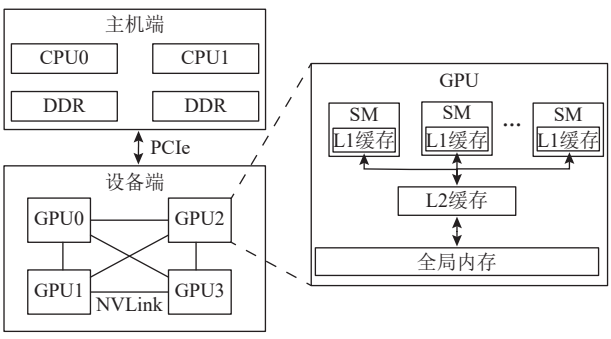


Fig. 1 Multi-GPU hardware architecture

图 1 Multi-GPU 硬件架构

在多 GPU 协同计算场景下,NVLink 高速互连技术突破传统 PCIe 带宽瓶颈,实现了 GPU 间高效直接通信。如图 1 所示,这类 Multi-GPU 的服务器通常由若干个 CPU 和多个 GPU 组成。各设备通过 PCIe 总线或 NVLink 传输线互连。在软件层面,NCCL(NVIDIA collective communications library)和 NVSHMEM 等通信库为多 GPU 并行计算提供了高效的编程接口。其中,NCCL 提供了包括 *NCCL_Send/Recv* 点对点通信以及 *AllGather*, *AllReduce*, *Broadcast* 等集合通信原语。NVSHMEM 则提供了核函数内的通信原语(如 *shmem_put*/*shmem_get*),支持多设备间的单边通信。在存储层次设计上,NCCL 采用分布式内存架构,NVSHMEM 则采用统一内存架构。

1.2 基于集合的图模式匹配算法

图模式挖掘算法旨在从数据图中识别或统计与模式图同构的匹配实例。给定一个包含 m 条边($m>2$)的模式图 $G_{\mathrm{P}}=(V_{\mathrm{P}},E_{\mathrm{P}})$ 和数据图 $G=(V,E)$,图挖掘算法需要找到所有的边映射关系,即 $(e\mapsto\hat{e})$,其中 $e\in E_{\mathrm{P}}$ 且 $\hat{e}\in E$,使得匹配实例子图 $\{\hat{e}_0,\hat{e}_1,\cdots,\hat{e}_{m-1}\}$ 与模式图 G_{P} 同构。近期研究[5-7]表明,图挖掘算法可通过以集合为中心的抽象进行描述,并借助集合求交等集合运算实现计算逻辑。因此,图挖掘算法的搜索过程呈嵌套循环结构:在第 i 层嵌套循环中,生成第 e_i 的候选集供下一层循环枚举。当数据图的所有边均与模式图中的边匹配时,程序终止。为避免搜索过程中产生自同构(automorphism)的子图,图挖掘系统[5,7-8]通常采用对称性破坏规则对匹配顶点进行约束,即 $vid(v_i)<vid(v_j)$,vid 为数据图顶点序号。另外,系统[5, 7-8]会根据模式图特征选择最优的匹配顺序构成程序的嵌套循环。

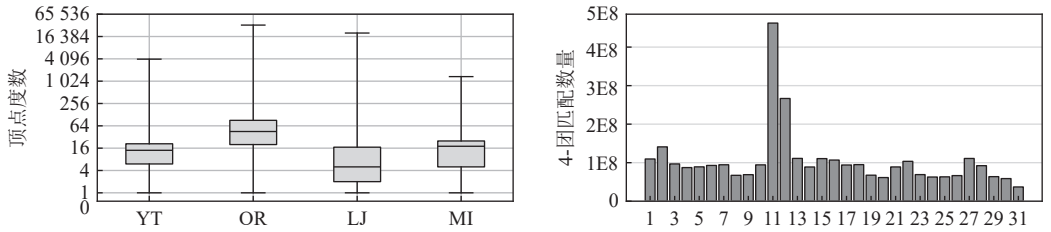
算法 1 展示了 4-团(4-clique)模式挖掘的执行流程伪代码。考虑到对称性破坏规则,要求 $vid(v_0)<vid(v_1)<vid(v_2)<vid(v_3)$。最外层循环遍历数据图中的所有顶点,并将其作为搜索的起始根节点。每个根节点维护各自的搜索状态并保存到候选集合 S 中。迭代轮次和模式图尺寸有关(图中迭代轮次为 4 轮),每轮迭代负责匹配在数据图中匹配模式图的 1 个顶点。在每次迭代开始时,首先读取此前的部分匹配结果,然后枚举当前候选集合 S 中的每个顶点 v_i,根据模式图的连通性,对匹配结果和 $N(v_i)$ 执行集合运算,生成下一轮遍历的新候选集合,同时更新搜索状态。在搜索完成后,程序输出匹配数量或者所有匹配结果。

算法 1. 基于集合求交的 4-团模式挖掘算法。

输入：数据图$G(V,E)$，模式图$G_P(V_P,E_P)$；
输出：所有 4-团的匹配结果e。
① for $v_0 \in V$ do /* 任务并行 */
② $S_1 = N(v_0)$;
③ for $v_1 \in S_1$ do /* 串行执行 */
④ $S_2 = S_1 \cap N(v_1)$;
⑤ for $v_2 \in S_2$ do /* 串行执行 */
⑥ $S_3 = S_2 \cap N(v_2)$; /* 算子并行 */
⑦ for $v_3 \in S_3$ do
⑧ return e $\{v_0, v_1, v_2, v_3\}$;
⑨ end for
⑩ end for
⑪ end for
⑫ end for

1.3 GPU 图挖掘的并行算法分析

1)图数据特征。真实世界的图数据通常展现出幂律分布的特性，即有少量高度数顶点和大量低度数顶点。图 2(a)展示了不同图数据的顶点度数分布情况，在 OR 和 LJ 图数据集上，最大度数超过 1 万而顶点的邻居数量平均值则在 100 以内。而对于 YT 和 MI 而言，其顶点的平均度数则低于 32。在图挖掘场景中，由于嵌套循环结构特性，程序需要反复遍历图中的顶点和边，这种幂律分布带来的数据倾斜被进一步放大，导致任务的负载不均问题更加严重。我们对 4-团匹配任务进行了分组统计。在图 2(b)中不同数据块上 4-团匹配数量呈现出显著差异，最高和最低的 4-团匹配数量差距近 30 倍。这种负载不均现象并非 4-团模式挖掘任务所独有，在频繁子图挖掘(frequent subgraph mining)、主题模式挖掘(motif finding)等其他图模式挖掘场景中同样普遍存在。这意味着在执行搜索任务时，处理包含高度数顶点的数据块所需要的计算资源和时间远超其他数据块，使得数据图中不同顶点的搜索任务负载不均，制约了图挖掘系统性能。



(a) 不同图数据集的顶点度数分布情况　(b) 不同顶点下的4-团匹配数量分布情况

Fig. 2 Characteristics of graph pattern mining problems

图 2 图模式挖掘问题的特征

2)GPU 图挖掘中的任务并行和算子并行。现有工作[10-11]通常采用 2 阶段并行来提升 GPU 图挖掘系统性能，即线程组内和线程组间并行。在线程组间，系统以固定线程组(比如 GPU 的 Warp)为粒度进行任务划分，并行处理数据图中的单个顶点或者边上的搜索任务(算法 1 中行①的任务并行)。另一方面，集合计算需要频繁的内存访问，检查顶点之间的连接关系。线程组内的所有线程并行处理搜索过程的集合操作以实现算子并行(算法 1 中行⑥的算子并行)。而中间的迭代中，线程组以串行的方式遍历搜索树(算法 1 中行③和行⑤)。因此，GPU 的图挖掘系统呈现出任务和算子 2 个层级的并行特性。

由于图挖掘任务的负载不均问题，一些工作[17]采用混合线程组映射技术进行任务分配。该技术根据顶点的度数对任务进行分桶。采用 GPU 的线程块、Warp, Thread 这 3 个并行粒度处理不同度数的搜索任务。对于高度数顶点，使用线程块协作处理任务可避免过多的任务成为程序拖后腿的部分。相反，对于低度数顶点，使用更多的线程并不会带来性能提升，通常用 Warp 和 Thread 处理。尽管任务并行和算子并行通过分层设计提升了 GPU 利用率，但现有方案[10-13]普遍忽略了搜索树遍历过程中内层循环的串行执行特性，导致 GPU 资源利用率不足。

算法 2. 集合求交的 2 种底层实现。
输入：有序数组 $List_1$ 和 $List_2$；
输出：公共元素数组 $List$。
① function *BinarySearchIntersect*($List_1$, $List_2$)
② for $key \in List_1$ do
③ if *BinarySearch*($List_2$, key) = true then
④ *List.append*(key);
⑤ end if
⑥ end for
⑦ return $List$;
⑧ end function

```
⑨ function HashIndexIntersect(List1, List2)
⑩   for value ∈ List2 do
⑪     Hashtable.insert(value);
⑫   end for
⑬   for key ∈ List1 do
⑭     if Hashtable.lookup(key) = true then
⑮       List.append(key);
⑯     end if
⑰   end for
⑱   return List;
⑲ end function
```

3)集合操作实现。集合求交计算是图挖掘中的核心操作。该操作的输入为 2 个整数有序数组 A 和 B, 在图挖掘场景中表示为顶点的邻居 $N(v_i)$或是候选集 $S(v_i)$。算法输出为这 2 个数组的共同元素。在 GPU 的 SIMT 架构上, 通常采用 2 种实现方法: 二分查找求交(binary search intersection)和哈希查找求交(Hash index intersection)。如算法 2 所示, 基于二分查找的求交算法将 2 个数组中较长的数组组织为二叉树结构(search list), 以较短列表中的元素作为搜索键值(key list)。对于每个搜索键值, 算法通过遍历二叉搜索树来查找匹配项。该方法的时间复杂度是 $O(n\log(n))$。一些研究工作[5,8,10,13]在图挖掘中系统采用二分查找法实现集合操作。另一方面, 基于哈希查找法的求交算法则为其中一个数组构建哈希表, 再以另一个列表的元素为搜索键, 在哈希表中查找公共元素。当发生哈希冲突时, 通常采用线性探测机制解决冲突。该方法的时间复杂度为 $O(n)$, 但需要占据额外的内存空间存储哈希表。另一些研究工作[12,17-18]在图挖掘系统中采用哈希查找法实现集合操作。综合来看, 二分查找法的搜索效率低, 但其不需要占用额外的存储空间。而哈希查找法则采取以空间换时间的思路提升查找效率。

1.4 现有的多 GPU 协同计算图挖掘系统

由于异构设备的显存容量有限, 难以处理大规模图数据。以英伟达的 32 GPU 显存的 V100 显卡为例, 单个 V100 的显存无法处理如 Datagen-sf10k(70 GB)这样的大图数据[19]。一些工作[14-16]利用 CPU 主存扩展图数据处理规模。这些工作通过将原始图数据划分成多个分区并流式传输到不同 GPU 进行处理。由于带匹配的模式图通常尺寸较小, 因此搜索范围不会超过 K 跳邻居(K 为模式图的直径)。利用这一局部性特征, VSGM[15]在 CPU 主存上构建多个具有重叠部分的图数据分区, 在分区内包含其内部顶点搜索所需的全部图拓扑数据。图 3 展示了利用 CPU 主存扩展的多 GPU 协同计算方案。其中 CPU 管理图数据以及分区数据传输, 不同的 GPU 并行处理各自的分区数据, 在本地进行无通信的图挖掘任务。然而, 这种重叠策略在每个分区中冗余存储 K 跳邻居, 导致额外的预处理开销和内存资源浪费。更严重的是, 该方案需要频繁地进行低速 PCIe 数据传输, 忽略了 GPU 互连拓扑和直接通信的特性, 拖慢了多 GPU 的计算效率。

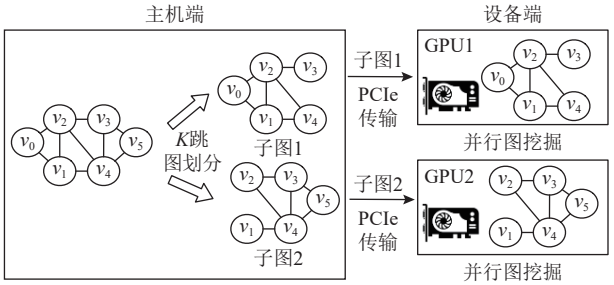


Fig. 3 Host memory based multi-GPU graph pattern mining

图 3 基于主存的 multi-GPU 图模式挖掘

1.5 Saturn 图模式挖掘系统设计概要

本文提出面向 Multi-GPU 的高性能图模式挖掘系统 Saturn 以充分发挥多个异构设备的计算和通信能力。如图 4 所示, Saturn 系统包含 2 个模块: 计算引擎和通信引擎。其中, 计算引擎负责并行本地的图挖掘任务。为提升计算效率, 我们在计算引擎中设计了细粒度并行的集合求交优化方法(将在第 2 节中详细介绍)。另一方面, 通信引擎负责 GPU 设备间的数据传输。与先前方法不同, 通信引擎充分利用 NVLink 高带宽特性, 在内核函数内直接发起通信请求获取远端邻居数据(通信技术将在第 3 节中详细介绍)。在图挖掘执行过程中, 图数据和迭代产生的中间结果保存在各个 GPU 的显存上。总的来说, Saturn 系统通过计算引擎和通信引擎以实现计算和通信的协同优化。

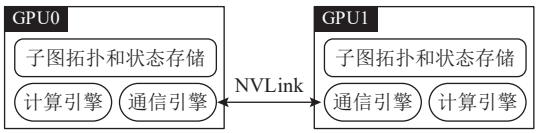


Fig. 4 Architecture of Saturn system

图 4 Saturn 系统架构

1.6 Saturn 与主流图挖掘系统对比

表 1 呈现了当前主流图挖掘系统的核心特性对比。其中, G^{2}Miner, GraphSet, VSGM, G2-AIMD, GAMMA 均属于基于 GPU 架构的图挖掘系统, GraphPi

则为 CPU 图挖掘系统。在计算优化层面，现有系统普遍采用粗粒度并行方案，但该模式难以适配图数据的不规则分布特性，易引发线程负载不均衡问题，制约计算效率。在通信优化层面，G²Miner, GraphSet, GraphPi 通过全图备份策略完全规避设备间通信以换取性能，但这些系统受限于显存容量，无法支撑超设备存储上限的图挖掘任务；VSGM, G2-AIMD, GAMMA 所采用的 K 跳分区策略，因依赖 CPU 参与数据传输，在处理大规模图数据时性能表现受限。相比之下，本文提出的 Saturn 系统实现计算与通信优化的协同设计，通过细粒度并行策略结合 CPU 无感通信技术，在保障高计算性能与通信性能的同时，能够支持大规模图数据挖掘任务的高效执行。具体而言，Saturn 利用完全图划分与 GPU+NVLink 互连技术，有效规避了 CPU 通信开销，实现了更优的多卡扩展性。

Table 1 Comparison of State-of-the-Art Graph Pattern Mining Systems

表 1 主流图模式挖掘系统对比

图挖掘系统	运行平台	并行粒度	图划分策略	通信方式	处理规模	计算性能	通信性能
G²Miner	单 GPU	粗粒度	全图备份	无通信	低	低	无
GraphSet	单 GPU	粗粒度	全图备份	无通信	低	低	无
GraphPi	多 GPU	粗粒度	全图备份	无通信	低	低	无
VSGM	CPU+GPU	粗粒度	K 跳分区	CPU+PCIe	高	低	低
G2-AIMD	CPU+GPU	粗粒度	K 跳分区	CPU+PCIe	高	低	低
GAMMA	CPU+GPU	粗粒度	K 跳分区	CPU+PCIe	高	低	低
Saturn（本文）	多 GPU	细粒度	完全图划分	GPU+NVLink	高	高	高

2 细粒度并行的集合求交优化

2.1 大量低度顶点引发的线程空闲问题

图数据的度数分布特性与 GPU 线程组执行模型之间的不匹配，是制约图挖掘系统性能的关键瓶颈。在图 5 中，我们对真实图数据集 LJ 和 YT 的累积度数进行统计。从图 5 中可以看到 YT 度数小于 32 的顶点占比达 91.17%，度数低于 256 的顶点占比更是超过 99%。在 LJ 中二者比例分别为 86.04% 和 99.48%。这种长尾分布特征导致传统的粗粒度线程分配策略（大度数任务用线程块处理，小度数任务用 Warp 处理）产生大量的线程空闲，造成显著的计算资源浪费。以 4-团模式挖掘算法的内层循环为例（如算法 1 的行⑤⑥所示），我们需要串行遍历候选集 S 中的每个顶点 v_2，对候选集 S 和顶点 v_2 的邻居列表执行集合求交计算。当采用 32 线程的 Warp 作为基本执行单元时，即使操作数规模仅为 4~16，仍需完整地占用 1 个 Warp 周期，导致平均每个周期存在 12.5%~50% 的线程处于空闲状态。

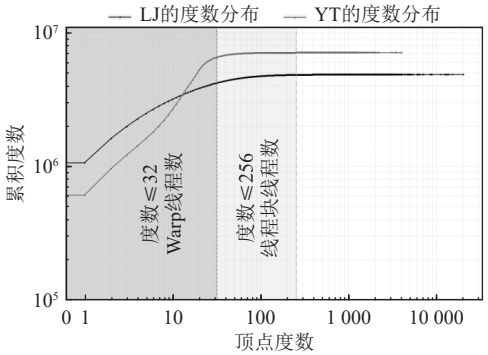


Fig. 5 Cumulative degree distribution of varying graph datasets

图 5 不同图数据集的累积度数分布

图 6 展示了 4-团模式挖掘的内层循环计算的例子。为简化描述，假设 Warp 包含 4 个线程。图 6 中当前匹配的候选集 $S_2=\{1, 2, 3, 4\}$，计算循环需要遍历集合 S_2 内的顶点执行集合求交产生下一层候选集 S_3。由于 S_2 的大小为 4，本层迭代总共执行 4 次集合

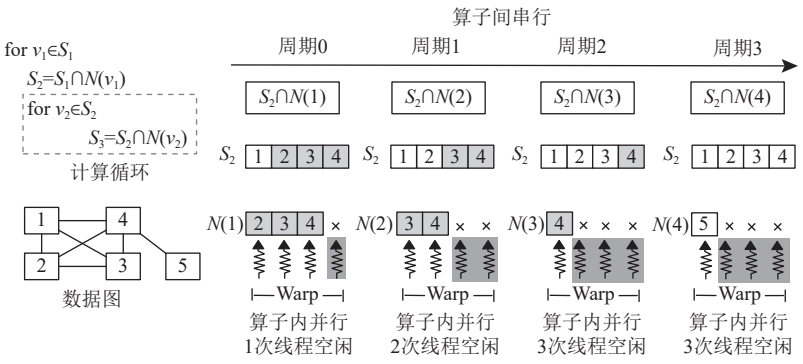


Fig. 6 Thread idle issue caused by coarse-grained parallelism

图 6 粗粒度并行引发的线程空闲问题

操作, 即$S_2 \cap N(1), S_2 \cap N(2), S_2 \cap N(3), S_2 \cap N(4)$, 占用4个Warp执行周期。在执行$S_2 \cap N(4)$时, 由于顶点4仅有一个邻居顶点5(根据破坏对称性规则, 仅考虑顶点*vid*比当前顶点大的情况), 导致Warp内仅有1个活跃线程。在图6中展示的4个Warp执行周期中共产生9次线程空闲, 有效的线程执行率仅56.25%, 造成严重的计算效率损耗。类似地, 对于大度数顶点处理采用的线程块级并行策略同样存在算力浪费问题。这种以固定线程组粒度处理单一集合操作的策略, 本质上延续了串行算法的执行模式, 无法适应图数据的幂律分布特性。当处理小规模操作数时, 线程组内大量线程难以避免地处于计算空闲状态。

2.2 优化机会

现有方法仅考虑算子内并行是导致线程空闲的原因。在图挖掘应用中, 传统集合操作的处理模式呈现“算子内并行、算子间串行”的局限性。当前流程通常为“数据准备”与“集合求交”的交替循环执行, 即每次仅能处理单一集合操作的2个操作数, 由线程组完成当前任务后再进入下一轮计算。这种粗粒度的算子并行的核心问题在于任务粒度与线程组粒度的静态绑定: 当处理小规模操作数, 如小度数顶点邻居$N(v_i)$时, 固定线程组(包含32个线程的Warp)因负载不足导致大量线程空闲, 而串行执行的多个集合操作被割裂为独立任务单元, 进一步加剧资源浪费。图挖掘算法的迭代循环特性(如第i层循环需对候选集S与多个顶点邻居$N(v_i)$依次求交)隐含重要优化机会: 同层循环内的多个集合操作共享公共操作数S, 且各操作的计算逻辑高度一致。若能打破“单次操作独占线程组”的静态约束, 将线程组从单一操作的固定绑定中释放, 使其能够利用空闲线程并行处理循环内的多个集合操作(如同时计算不同顶点的$N(v_i)$与S的交集), 则可显著提升资源利用率。

集合操作的细粒度展开是获取算子间并行性的关键。集合操作通常包含装载与查询这2个阶段: 前者需将邻居列表或哈希表加载为待查列表, 后者则需在待查列表中检索另一列表的键值。针对图挖掘迭代循环中多个集合操作共享公共操作数S的特性, 装载阶段可优化为仅需一次性并行加载共享操作数S, 避免重复数据搬运; 而线程空闲主要发生于于查询阶段: 循环内依次将$N(v_i)$中的数据作为键值在共享操作数S中查询时产生了大量线程闲置。现有方法是粗粒度地对集合操作按线程组计算, 忽略了装载阶段的操作数重复和查询阶段列表的长度差异。如图7所示, 若能对查询阶段的键值进行细粒度并行, 让线程组内每个线程负责单个键值的查询任务, 则可使得线程空闲次数减少到1次。由此可见, 细粒度展开集合操作的2阶段流程, 针对查询阶段的多操作并行性进行调度优化, 是解决线程空闲的潜在机会。

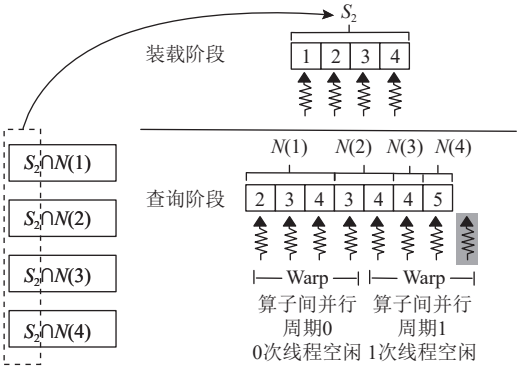


Fig. 7 Two-stage decoupling of set operations

图7 集合操作的2阶段解耦

2.3 细粒度并行

为了利用上述优化机会, 我们提出了细粒度并行的计算优化方案来处理线程空闲的问题, 将单次循环内的多个集合操作进行批处理。考虑到集合操作的这2个阶段, 我们利用线程组并行加载待查列表, 而在查询阶段让线程组内的每一个线程并行查询循环内操作数的每个键值。

1)查询阶段的细粒度并行。区别于传统的以集合操作为中心的粗粒度并行, 细粒度并行将集合操作的装载和查询阶段解耦。首先, 我们通过前缀和(prefix sum)计算邻居列表的长度和偏移地址。线程组内每个线程根据偏移索引到对应的顶点邻边和对应的邻居顶点*vid*。每个线程根据*vid*获取单个顶点作为键值, 在待查列表中查找是否命中, 执行顶点连接关系判断。每个线程独立计算索引并进行查询, 查询过程可以并行执行且无数据依赖。当前周期内线程查找完毕, 所有线程读取线程组内最后一个线程处理的顶点*vid*, 更新各自要处理的下一个任务索引。重复上述过程直到循环内所有集合操作的查找任务结束。在索引更新阶段, 以Warp为粒度的线程组采用*__shfl_sync*和*__syncwarp*原语进行Warp内线程索引通信。以线程块为粒度的任务并行则采用共享内存的索引数组*shared_idx*[]和*__syncthreads*原语进行通信。

图8展示了细粒度并行的计算流程。该流程描述了用Warp处理图7中$S_2 \cap N(1), S_2 \cap N(2)$这2个集合操作。首先每个线程判断*tid*是否超过邻居边界, 由于$N(1) = \{2, 3, 4\}$, 线程3重新计算待处理的顶点

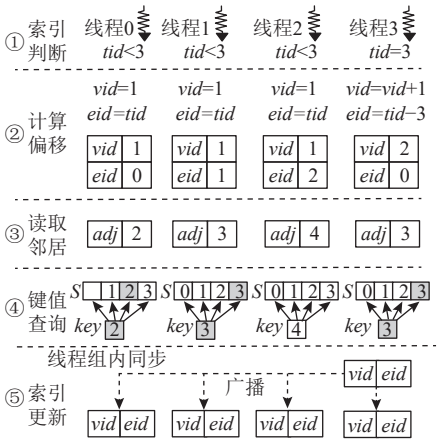


Fig. 8 Fine-grained parallelism computing process

图 8 细粒度并行计算流程

和边。在读取邻居阶段，每个线程根据 *vid* 和 *eid* 读取各自的邻居数据作为键值，并在查询阶段并行执行查询任务。在索引更新阶段，线程 3 则在线程组内广播 *vid* 和 *eid*，为下一个 Warp 执行周期做数据准备。

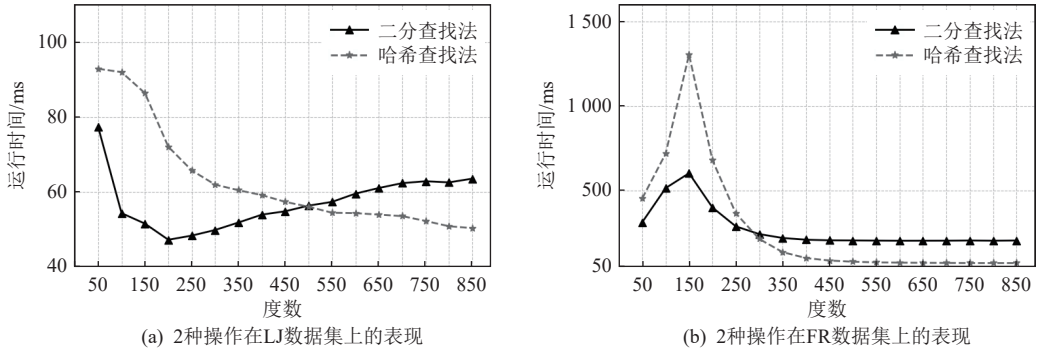
2）装载阶段的混合算子策略。现有的工作采用 2 种不同的集合操作实现，即哈希查找法和二分查找法。对于哈希查找法，其装载阶段需要对查列表构建哈希表。而对于二分查找法则是构建二叉搜索树。而我们发现真实世界图数据中的度数的大小会影响这 2 种方法的效果。在图 9 中，我们展示了 2 种集合计算方法在 LJ 和 FR 这 2 个数据集上不同度数区间的性能差异。我们可以看到在大度数任务中，哈希查找法的效果优于二分查找法。而在处理小度数任务时，由于装载阶段引入了建表的开销，哈希查找法的加载效率低于二分查找法。因此，我们通过设置度数阈值的方式来混合使用 2 种集合计算方法。对于度数大于阈值的顶点，我们采用哈希查找方法；而对于度数小于该阈值的顶点，则使用二分查找法。度数阈值是一个经验参数，在实验设置中，我们采用阈值 256 来区分这 2 种方法。

此外，装载阶段可以进一步对访存进行优化，对于低度数顶点，可将构建的二叉搜索树缓存至共享内存中。由于小度数顶点的邻居规模较小，共享内存的高速访问特性能够显著减少不规则的二分查找对访存的影响。对于大度数的哈希表结构，我们将哈希表的前 *k* 层数据缓存至共享内存，若未命中则退化成全局内存的哈希表查找。分层缓存策略能够提升高频访问的哈希表关键层读取性能，降低查询阶段的访存延迟。

3 CPU 无感的 GPU 间通信技术

3.1 基于分布式内存架构的通信方案

在分布式内存的系统中，GPU 间通过 NCCL 通信库进行消息传递。在这种通信模式下，GPU 负责将通信数据在核函数内准备好，再通过 CPU 发起通信请求将数据通过 NVLink 发送到目的 GPU。在分布式图模式挖掘场景中，由于图数据分布在不同的 GPU 设备上，其通信主要是在传递邻居列表。由于 NCCL 库依赖于 CPU 控制通信，每个 GPU 无法直接访问其他 GPU 的设备内存来获取顶点及其邻居数据。在这种情况下只能通过 CPU 调用 *ncclSend*/ *ncclRecv* 等显式通信方式完成。另外，为了最小化通信启动开销并充分利用带宽，需将发往相同 GPU 的数据进行聚合并批量发送。图 10 介绍了在分布式内存场景下多 GPU 间如何进行通信来协作完成图模式挖掘。图 10 中展示的 2 个 GPU 分别存放奇数顶点和偶数顶点及其邻居数据。在搜索过程中，GPU0 需要顶点 0, 1, 2, 3 的邻居数据，GPU1 则需要顶点 4, 5, 6, 7 的邻居数据。由于数据分布在本地和远端，此时 GPU 需要对所需数据按目标设备进行重排和分桶（如图 10 中的步骤①所示）。在目标 GPU 接收顶点之前，需先传输顶点数量，再传输实际顶点 *vid*（如图 10 中步骤



(a) 2种操作在LJ数据集上的表现 (b) 2种操作在FR数据集上的表现

Fig. 9 Comparison of binary search intersection and Hash index intersection

图 9 二分查找求交和哈希查找求交的对比

②所示)。每个 GPU 在获取所有顶点 *vid* 后,在步骤③对其他 GPU 所需邻居数据执行本地批量写入操作。随后在步骤④中,所有聚合后的顶点邻居数据将被回传,在多个 GPU 间执行邻居交换操作。由此可见,若采用 CPU 协同的多 GPU 图挖掘方案,不仅需要大量通信操作,还需多次同步操作以避免读写冲突,这导致了更高的通信量和内存带宽占用。

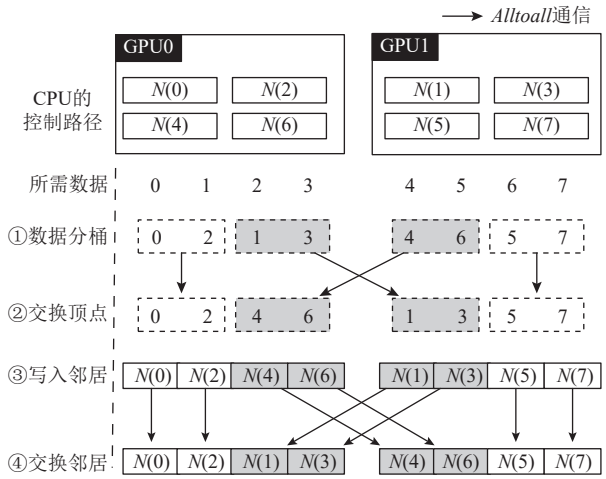


Fig. 10 Distributed-memory based GPM communication mode

图 10 基于分布式内存的图模式挖掘通信模式

3.2 基于统一内存架构的高效通信方案

考虑英伟达原生的 GPU 通信技术和编程框架。其中最值得关注的是 NVSHMEM 库,它为细粒度(几字节到几十字节)的 GPU 间通信提供了内核级接口,如图 11 所示。本地 GPU 的数据可直接访问,而远端数据的读取则通过 NVSHMEM API 函数 *nvshmemx_TYPENAME_get_warp* 实现,该函数将所需数据传输至本地共享内存(*TYPENAME* 可为 int/float/double 等常见数据类型)。类似地,远端访问支持按 Thread/Warp/线程块为粒度进行通信。另外,NVSHMEM 提供统一内存注册函数 *nvshmem_malloc* 将数据放置共享内存区域。

```
1  /* 在GPU上初始化NVSHMEM的上下文 */
2  nvshmem_init ( )
3  /* 分配所有GPU可见的共享内存 */
4  d_shared_mem = (void*) nvshmem_malloc (num_bytes);
5  /* 分配仅当前GPU可见的本地内存 */
6  cudaMalloc(( void**) & d_mem, num_bytes);
7  /* 以Warp为粒度的远端内存访问 */
8  __device__ nvshmem_int_get_warp(void *dst, const void *src,
                size_t nelems, int src_gpu_id);
9  /* 以线程块为粒度的远端内存访问 */
10 __device__ nvshmem_int_get_block(void *dst, const void *src,
                size_t nelems, int src_gpu_id);
11 /* 在CPU上释放NVSHMEM的上下文 */
12 nvshmem_free(d_shared_mem);
```

Fig. 11 NVSHMEM APIs listing

图 11 NVSHMEM 接口列表

传统的 GPU 通信框架通常将数据传输分为 2 条路径:数据实际移动的路径,以及包含消息准备、触发数据移动和向远程进程发送信号的通信控制路径。大多数框架虽支持 GPU 到 GPU 的数据移动路径,但仍将控制路径实现在 CPU 上,这导致了额外的控制延迟和同步开销。在本工作中,我们采用 CPU 无感的 GPU 通信模式,将这 2 条路径均实现在 GPU 上,从而显著降低了控制路径延迟,并减少了跨设备同步开销。这些改进共同使得核函数内通信变得可行且具有潜在优势。具体而言,在具有 NVLink 互连的多 GPU 系统上,我们能对同一节点上远程 GPU 内存进行直接读写操作,从而将整个集群构建成一个统一内存架构(unified memory architecture, UMA),后文简称“统一内存”。在底层实现上,我们采用 NVIDIA 的 NVSHMEM 库及其提供的 *GET*/*PUT*/*ATOMIC* 原语。这一机制将所有控制逻辑保留在 GPU 上,无需 CPU 干预。通过这种设计,我们不仅充分利用了 GPU 的并行计算能力处理通信控制逻辑,还避免 CPU 与 GPU 间的数据传输开销,实现更高效的核函数内通信。

在统一内存架构的多 GPU 系统中,每个 GPU 可以直接访问其他 GPU 的设备内存,因此只需要在每个 GPU 上在内核函数内读写远端地址即可完成通信操作。如图 12 所示,GPU 在执行本地图挖掘的搜索任务时,如果所需的邻居数据不在本地,则根据远端数据所在的地址直接读取远端内存并将邻居列表写入本地,继续执行后续的搜索任务。这种方法消除了分布式内存下多 GPU 通信时的数据重排和跨 GPU 同步开销。在这个过程中 GPU 上不同的 SM 将同时发起单边通信请求。这样的单边通信模式仅需 1 个步骤,减少了冗余的通信量和占用更低的网络带宽。

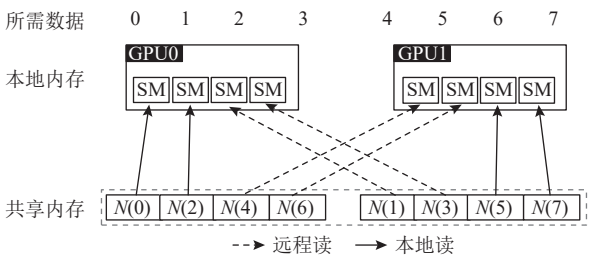


Fig. 12 UMA-based GPM communication mode

图 12 基于统一内存架构的图模式挖掘通信模式

3.3 高度数顶点缓存的图数据共享存储

考虑分布式环境,我们需要进行图划分来处理更大规模的图数据,本文采用 1 维图划分方法[20]。将输入图的顶点集 V 划分为 N 个部分($V_0, V_1, \cdots, V_{N-1}$),

其中 N 为 GPU 设备数量。GPU 设备 $i(0 \leqslant i < N)$ 会在其全局内存中保存与 V_i 相关的所有图数据，即与 V_i 相关的所有边和邻居数据。为确保数据分布均衡，我们采用与 Khuzdul[20] 一致的图划分哈希函数。顶点所属的 GPU 设备号由哈希函数 $H(v)$ 决定，该函数将顶点 v 映射到对应的设备 ID 上（$0 \leqslant ID < N$）。每个 GPU 设备上包含映射到它的顶点子集以及与这些顶点相连的所有边。

我们采用压缩稀疏行（compress sparse row, CSR）格式存储图结构，并将其划分为多个数据分区。每个分区包含一组顶点及其对应的邻居顶点数据，索引和实际数据均位于统一内存空间中。为支持内存共享下的快速邻居访问，我们为每个 GPU 维护一个邻居地址指针，该指针记录了每个顶点在统一内存中的起始和结束地址。我们通过邻居地址指针确定目标顶点所在的 GPU 并获取其地址。本地 GPU 上的数据可直接访问，而远程数据则按 Warp 为通信粒度传输到本地。例如，若顶点 v_i 存储在 GPUi 上，指针 m 的第 i 个条目记录顶点 v_i 在 GPUi 上的内存地址，第 $i+1$ 个条目则记录顶点 v_{i+1} 的内存地址。

传统 1 维图划分方法与 GPU 内存共享设计结合时，会因未充分考虑高度数顶点对通信的影响而产生显著通信开销。为此，我们提出基于高度数缓存（high degree caching, HDC）的优化策略，即将高度数顶点的邻居数据备份存储至每个 GPU 设备内存中，通过将高度数顶点的远程访问转换为本地读取，显著降低通信量。尽管备份高度数顶点会引入数据冗余并增加存储需求，但这一代价可被通信开销的显著降低所抵消，该特性在处理具有偏斜度分布的真实世界图数据时尤为关键。

图 13 展示了基于统一内存架构的图数据存储：统一内存区域存储了图的 CSR 索引结构和图拓扑数据。在私有内存区域，每个 GPU 保存图挖掘过程中的搜索状态和高度数顶点的邻居数据备份。在搜索过程中，系统根据哈希函数决定是从本地还是远端获取邻居列表（包含顶点索引和加载邻居 2 个步骤），在得到邻居列表后，即可执行后续的集合操作。

3.4 统一内存架构下的图挖掘计算模式及通信优化

1）CPU 无感的图模式挖掘算法。算法 3 展示了 CPU 无感的图模式挖掘流程，该算法主要通过单边通信实现跨 GPU 数据访问。在 CPU 端，CPU 主机端先通过 1 维图划分生成子图分区（算法 3 的行⑩），再将顶点列表和边列表注册到统一内存空间并加载至各 GPU（算法 3 的行⑪⑫），为内核函数的跨设备直接访问提供数据基础，后续的图模式挖掘过程中的计算和通信则完全交由 GPU 核函数负责。在 GPU 端，*gpu_graph_mining* 核函数遍历本地顶点集合并对每个顶点 v_0 通过 *get_neighbours* 以单边通信方式获取其邻居列表 v_0_*adj*，再递归遍历 v_0 的邻居 v_1 并获取 v_1_*adj*，通过计算 2 组邻居的交集统计子图模式数量（算法 3 的行②~⑧，以 4-团匹配为例）。

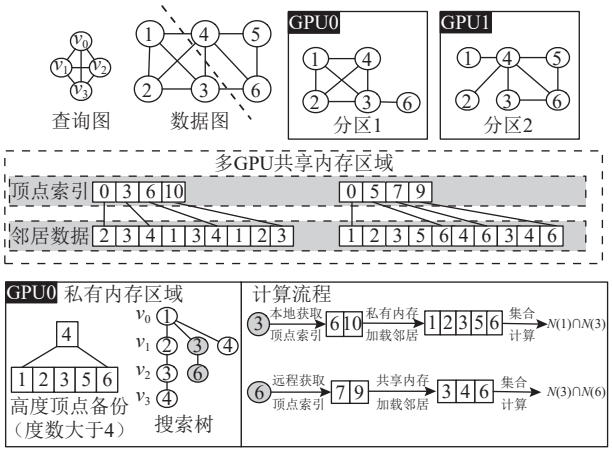


Fig. 13 UMA based graph data storage

图 13 基于统一内存架构的图数据存储

算法 3. CPU 无感的图模式挖掘算法。

① function *GPU_graph_mining*(*g*)
② for each v_0 in $g.V$
③ v_0_*adj* = *g.get_neighbours*(v_0);
/* 本地访存 */
④ for each v_1 in v_0_*adj*
⑤ v_1_*adj* = *g.get_neighbours*(v_1);
/* 单边通信 */
⑥ *count* += *intersect*(v_0_*adj*, v_1_*adj*);
⑦ end for
⑧ end for
⑨ end function
⑩ function *graph_mining_program*( )
⑪ *g* = *graph_1D_partition*( );
⑫ *register_memory*(*v_list*, *e_list*);
/*注册统一内存*/
⑬ *load_graph*(*g*, *v_list*, *e_list*);
⑭ *GPU_graph_mining* (*g*);
⑮ end function

2）基于消息聚合的通信优化。在多 GPU 的分布式图挖掘场景中，远端数据访问面临显著挑战。一方面，不同 SM 可能同时发起对同一远端 GPU 数据的请求，受限于 NVLink 端口资源，易引发链路冲突与带宽竞争，从而降低数据传输效率；另一方面，多

次重复邻居数据请求会产生大量冗余通信,且图挖掘的动态性与不确定性导致远端数据访问时机难以预测,进一步加剧通信低效问题。

为解决上述问题,我们提出基于消息聚合的通信优化策略。具体而言,在本地设备内存上系统为各GPU配置多消息队列,依据目的GPU对SM发起的数据请求进行分桶调度(如图14所示,通过*GPU_ID*与*vid*映射关系,将请求定向分配至对应GPU队列),从而规避同一远端GPU链路的竞争冲突;针对同一目的GPU的请求队列,在获取邻居数据之前,我们对重复邻居数据通信采取聚合处理,合并冗余请求。该策略适配图挖掘任务特性,有效缓解链路竞争与冗余传输问题,提升GPU带宽利用率。

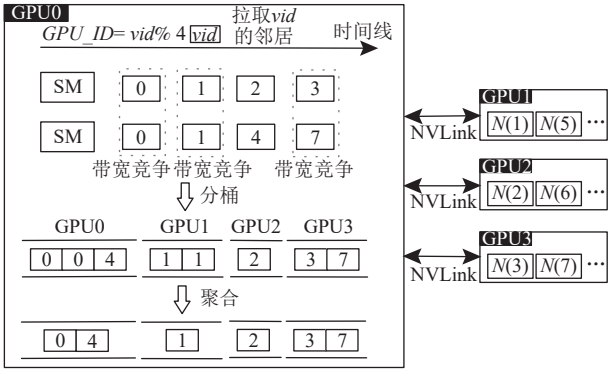


Fig. 14 "Binning-Merge" based communication optimization

图14 基于"分桶-聚合"的通信优化

4 实验与结果分析

4.1 实验设置

1)硬件和软件环境。实验在2个硬件平台进行:在单节点平台上,测试采用1台配有8块A100 GPU的Linux服务器,单卡的显存容量是40 GB,GPU之间通过300 GBps带宽的NVLink实现高速互连,CPU使用2颗AMD EPYC 7763,总共128核。在多节点平台上,测试采用1个8节点集群(每个计算节点包含4块V100 GPU),单卡的显存容量是32 GB,GPU间采用150 GBps带宽的NVLink互联,节点间采用100 Gbps带宽的InfiniBand互联。软件环境方面,操作系统版本为Ubuntu 22.04。采用的CUDA版本是12.2,NVSHMEM版本是3.0.6。GPU程序编译使用nvcc编译器,启用-O3优化选项以提升执行效率。构建工具CMake的版本为3.28,gcc编译器版本为11.4.0。

2)数据集和模式图。实验中我们采用了来自SNAP[21]和LDBC[19]的真实世界和人工生成图数据。如表2所示,这些数据集包含7个中小规模数据集(图边数分别达百万级和千万级),用于评估单GPU图挖掘系统性能,以及2个大规模图数据FR和DG(图边数达10亿级)用于测评多GPU协同计算的图挖掘系统。具体而言,CP是专利之间的引用图;EN是电子邮件通信图;AM是亚马逊电商产品购买数据图;CA则是道路网络图;在社交网络类别中,MI刻画作者合作网络,YT属于网页图类别,FR和LJ则是具有倾斜顶点的社交网络图。DG是用LDBC工具生成的大规模社交网络数据。此外,如图15所示,实验选取9个典型模式图(顶点数量为3~6个)作为测试基准,这些模式图在图挖掘系统评测领域已被广泛应用[6-7,14-15],能够有效验证系统在不同计算场景下的性能表现。

Table 2 Graph Datasets in Real World

表2 真实世界图数据集

数据	名称缩写	图数据名称	顶点数量 V	边数量 E	最大度数
社交	MI	Mico	0.10×10^6	2.00×10^6	1 359
邮件	EN	Enron	0.69×10^6	0.27×10^6	1 391
电商	AM	Amazon0601	0.40×10^6	2.40×10^6	2 747
道路	CA	roadNet-CA	1.90×10^6	2.80×10^6	12
专利	CP	Cit-Patents	3.70×10^6	33.00×10^6	793
社交	YT	YouTube	7.00×10^6	114.00×10^6	4 017
社交	LJ	LiveJournal	4.85×10^6	85.70×10^6	20 333
社交	FR	Friendster	66.00×10^6	$3\,600.00\times10^6$	5 214
人工合成	DG	Datagen-sf10k	98.70×10^6	$18\,800.00\times10^6$	4 485

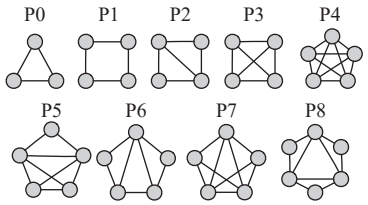


Fig. 15 Pattern diagram used in experiment

图15 实验中所用的模式图

3)对比系统。为全面评估本文方法的性能,我们选取多个主流GPU上的图挖掘系统作为对照,具体包含G^{2}Miner[10], GraphSet[11], VSGM[15], G2-AIMD[16], GAMMA[22]。在单GPU系统方面,G^{2}Miner和GraphSet将整个数据图存储在GPU的显存上,并采用高效并行策略与高性能集合算子优化计算效率。Multi-GPU系统GAMMA在处理小图时支持该种图存储模式。因此我们选择这些系统来评估本文方法在处理小规模数据上的计算优化效果。

在多 GPU 协同计算场景中，VSGM, G2-AIMD, GAMMA 采用 CPU-GPU 混合内存架构存储数据图，以支持更大规模图挖掘任务。因此我们选择这些系统来评估 Saturn 在处理大规模数据上的效果。考虑到图挖掘问题的 NP 难特性，所有对比系统均采用相同的匹配顺序（matching order）和对称性破坏（symmetry breaking）规则，以确保实验结果的公平性。

实验评测方法分为以下 3 个维度：

1）端到端性能对比。4.2 节通过单 GPU 与多 GPU 场景下的端到端性能测试，验证 Saturn 在不同规模数据处理中的效率优势。首先通过单 GPU 环境下的对比测试，验证 Saturn 在不依赖通信优化时的计算效率优势；其次展示了在多 GPU 协同计算场景中，对比 Saturn 与现有系统的整体性能表现；最后展示了单 GPU 上不同系统在处理主题计数应用的性能差异。

2）性能优化效果评估。4.3 节聚焦 Saturn 核心优化策略的量化效果，通过消融实验揭示各模块对系统性能的贡献度，评估系统在 GPU 内和 GPU 间整体的算力利用率以及可扩展性。首先深入分析计算效率提升的主要来源；然后重点评估数据缓存与消息聚合策略对通信性能的优化效果；接着讨论 CPU 无感调度机制对多 GPU 协同的影响；最后通过多卡集群测试验证 Saturn 的强扩展性表现。

3）系统微基准测试分析。4.4 节通过微基准测试评估系统运行时的计算、存储及通信开销行为，以体现优化方法的低开销特性。首先探究细粒度并行技术对 GPU 线程组利用率的提升；然后讨论不同阈值选择对 GPU 混合算子策略性能的影响；接着展示系统运行时的时间分布和内存占用情况；最后则讨论了多卡协同计算时采用统一内存架构和 NCCL 传统分布式内存架构在 NVLink 带宽利用率和延迟的差异。

4.4 节评估系统在 GPU 内和 GPU 间整体的算力利用率以及可扩展性。首先探索细粒度并行技术对 GPU 线程组利用率的提升；其次讨论 CPU 无感调度机制对多 GPU 协同的加速作用；最后实验通过 8 卡集群测试验证 Saturn 的强扩展性表现。

4.2 端到端性能对比

1）单 GPU 图挖掘系统对比。为了评估 Saturn 系统的计算效率，我们将其与 G^{2}Miner, GraphSet 和单 GPU 版本的 GAMMA 这 3 个主流的单 GPU 图模式挖掘系统进行对比。我们选择 7 个中小规模的数据图（CP, MI, YT, LJ, EN, AM, CA）和全部的 9 个模式图（P0, P1, P2, P3, P4, P5, P6, P7, P8）展开性能评估，所有数据图均存储于 GPU 显存中。端到端性能对比结果如图 16 所示。实验结果显示，Saturn 在所有测试场景中性能均优于 G^{2}Miner, GraphSet, GAMMA，相对三者的平均加速比（几何平均值）分别达 2.51、3.73 和 8.76，最高加速比分别达 7.28、12.32 和 42.81。整体来看，Saturn 的运行速度在各类场景下都有明显优势。

在数据图方面，小图普遍存在平均度数较低的特征，这使得传统图挖掘系统在执行过程中产生大量线程空闲，制约了计算效率。而 Saturn 的细粒度并行方法可以充分利用线程组的并行性，在小数据集（如 MI 等大部分顶点度数偏低的数据集）上对比 G^{2}Miner 平均取得 2.8 的加速比。在模式图方面，我们发现随着模式图结构复杂度提升，Saturn 的性能优势越发显著。以处理 YT 数据图为例，在挖掘简单模式（如模式图 P0）时，Saturn 系统相比于 GraphSet 取得 1.7 的加速比，而当面对复杂的模式图（如 P3 和 P4）时，最高可取得 2.6 的加速比。在处理 P8 这种包含 6 个顶点模式图时取得了 5.95 的加速比。这是由于复杂模式图涉及更多嵌套循环，线程负载不均问题更为突出，进一步加剧了线程空闲。而我们的计算优化方法能够有效缓解负载不均衡问题，更好地适配复杂模式图的挖掘过程。总的来说，本文的计算优化方法能够应对数据图和模式图的不同组合场景。

另外，GraphSet 作为 GPU 自动代码生成系统，为支持任意模式图引入额外运行时开销，这使其在处理非规则模式图（P1 和 P2）时，性能显著低于 G^{2}Miner 和 Saturn 系统。

2）多 GPU 图挖掘系统对比。如图 17 所示，我们在 8 个 A100 GPU 的服务器上将 Saturn 与 VSGM, G2-AIMD 和 GAMMA 这 3 个多 GPU 协同计算的图挖掘系统进行性能对比。为屏蔽计算优化对通信性能的影响，在实验中我们采用基础版本 Saturn(w/o opt)（和基线保持相同计算能力）和开启第 2 节介绍的计算优化版本 Saturn 进行测试。我们使用 2 个大规模数据图 FR 和 DG，在全部 9 个模式图（P0, P1, P2, P3, P4, P5, P6, P7, P8）。由于数据图和中间结果内存开销超过单 GPU 显存，多 GPU 协同计算系统需要对数据图进行图划分。实验结果表明：优化后的 Saturn 能处理更大的图数据，而 GAMMA 在 P0～P7 时，DG 图数据上会用尽内存（图中 OOM），在最复杂的 P8 上，GAMMA、VSGM、未开启优化的 Saturn 运行时间远超 4 h，而优化后的 Saturn 能在数百秒内完成计算；Saturn 较 VSGM, G2-AIMD, GAMMA 在不同数据及模式图下，平均加速比分别达 12.3（最高 20.4）、6.2（最高 13.8）和 59.7

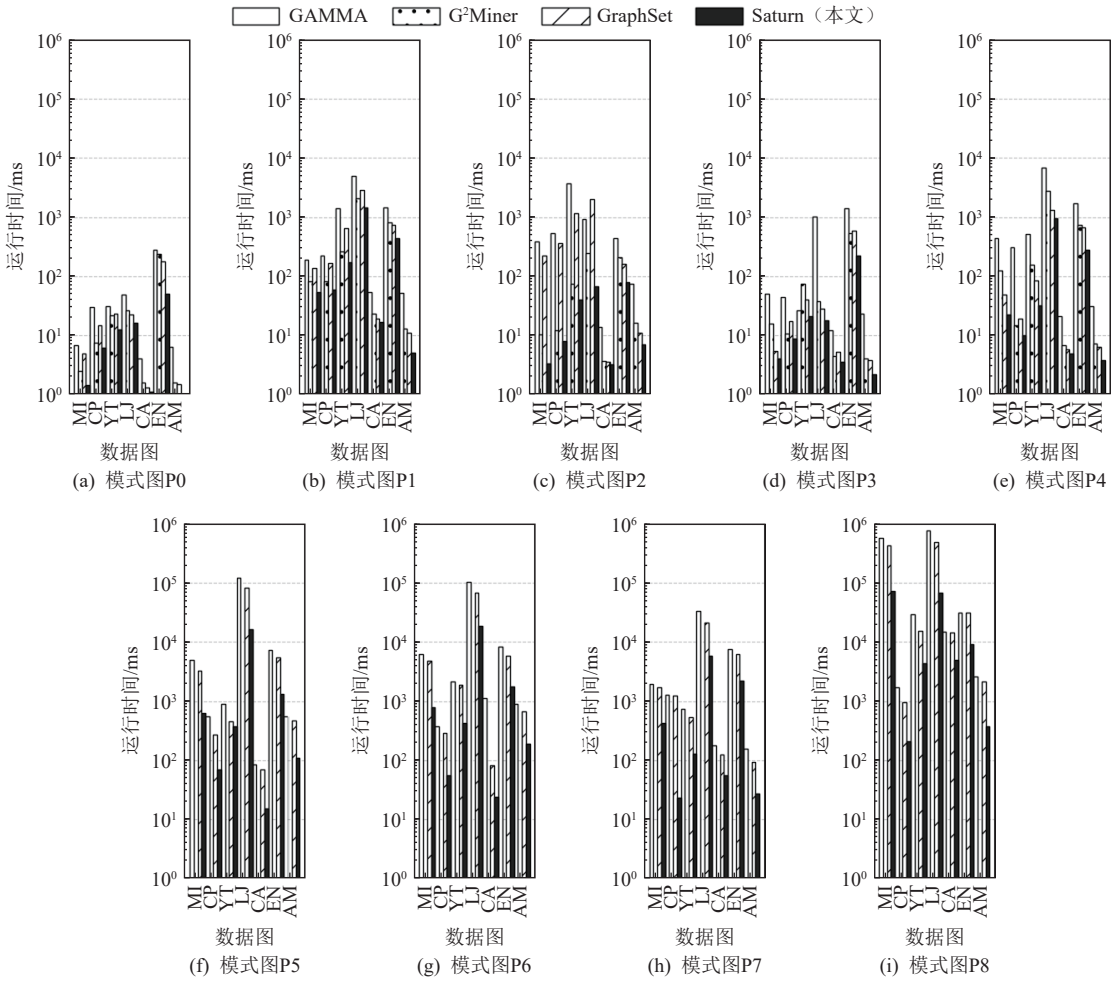


Fig. 16 Execution time comparison of Single-GPU GPM systems on small/medium-scale data graph

图 16 单 GPU 图模式挖掘系统在中小规模数据图的运行时间对比

(最高 85.5)；另外，Saturn 较 Saturn(w/o opt) 平均性能提升 1.44 倍。性能优势主要来源于 2 个方面：VSGM 和 G2-AIMD 依赖于 CPU 传输数据图，性能受 PCIe 低带宽的限制。Saturn 采用 CPU 无感的调度策略，充分利用 GPU 间 NVLink 的高带宽特点。而 GAMMA 并没有将 CPU 的 PCIe 数据传输和 GPU 计算进行掩盖，导致处理大图时性能劣于其他 3 个系统。另一方面，在处理复杂模式图时，VSGM 和 G2-AIMD 采用的 K 跳分区方法会增大分区直径，导致数据通信量激增。Saturn 通过消息聚合和数据缓存优化，降低了 GPU 间通信开销，由此实现相较于对比系统的性能优势。

3)图挖掘应用性能对比。我们选取主题计数(motif counting)和频繁子图挖掘(frequent subgraph mining, FSM)作为典型图挖掘应用展开性能分析。主题计数旨在识别固定顶点数量的高频模式，其搜索需同时查找多个模式。频繁子图挖掘的核心目标是在带标签的属性图中，挖掘出现频率高于预设阈值(即支持度)的子图结构。

在主题计数应用上，我们测试了单 GPU 配置下不同系统在处理 3-motif 和 4-motif 这 2 类应用在 4 个数据集上的性能表现，结果如表 3 所示。所有场景下，Saturn 的性能均优于 G²Miner 和 GraphSet，平均加速比分别达 4.59 和 3.21，最高则为 7.66 和 5.67。这表明在处理多模式查询时，Saturn 的细粒度并行机制能发挥更大优势，显著减少冗余计算与线程闲置。

在频繁子图挖掘应用上，我们选取带标签版本的属性图 MI-29 和 YT-29(这 2 个图上顶点均有 29 类标签)上验证 Saturn 系统处理异构图的能力，其中 2-fsm 和 3-fsm 这 2 类应用的性能对比如表 4 所示。实验结果显示，Saturn 系统相较于 G²Miner 和 GraphSet 分别取得平均 5.84 和 1.87 的加速比，在标签数量多、结构复杂度高的异构图挖掘任务中依旧保持稳定高效，充分验证了 Saturn 在异构图上执行频繁子图挖掘任务的高效处理能力。

4.3 性能优化效果评估分析

1)计算优化的性能分析。为了评估 Saturn 系统

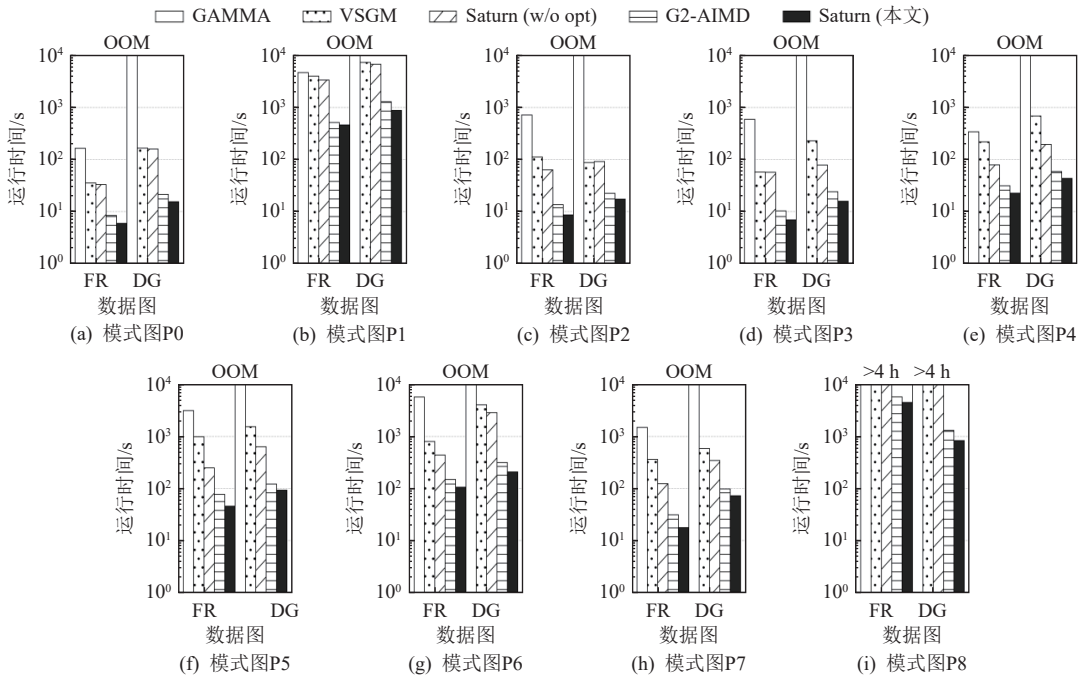


Fig. 17 Execution time comparison of multi-GPU GPM systems on large-scale data graph

图 17 多 GPU 图模式挖掘系统在大规模数据图的运行时间对比

Table 3 Execution Time Comparison on Motif-Counting Applications

表 3 在主题计数应用上的运行时间对比 s

数据集	3-motif			4-motif		
	G²Miner	GraphSet	Saturn（本文）	G²Miner	GraphSet	Saturn（本文）
CP	1.23	0.91	0.33	1.87	2.38	0.44
MI	0.46	0.34	0.06	1.34	0.71	0.42
LJ	1.89	1.53	0.44	45.38	9.85	5.87
YT	2.65	2.28	0.82	7.89	7.33	3.21

在单 GPU 上计算优化对性能的影响，我们使用 YT 和 LJ 这 2 个数据图和 4 个模式图（P1, P2, P3, P4）进行测试。如图 18 所示，实验以未做任何优化的基础版本作为性能基线。其中，细粒度并行聚焦于集合算子的查询阶段，通过并行处理同一层循环内的多个集合操作，大幅降低 GPU 线程组内的空闲线程比例，有效缓解传统串行执行模式导致的计算资源浪费问题，在不同数据图和模式图场景下，该策略平均实现了 2.12 倍的性能提升。而访存优化则是在细粒度并行的基础上进一步升级，采用混合算子策略，针对数据图中大度数顶点与小度数顶点的特性，分别设计差异化的集合算子实现方案，同时将待查询数据提前预加载至共享内存，避免多次重复读取数据带来的性能损耗，该优化策略带来了 22% 的平均性能增益。实验结果表明，面对数据图与模式图的多种组合，Saturn 系统的计算优化方法均展现出了显著的性能提升效果。

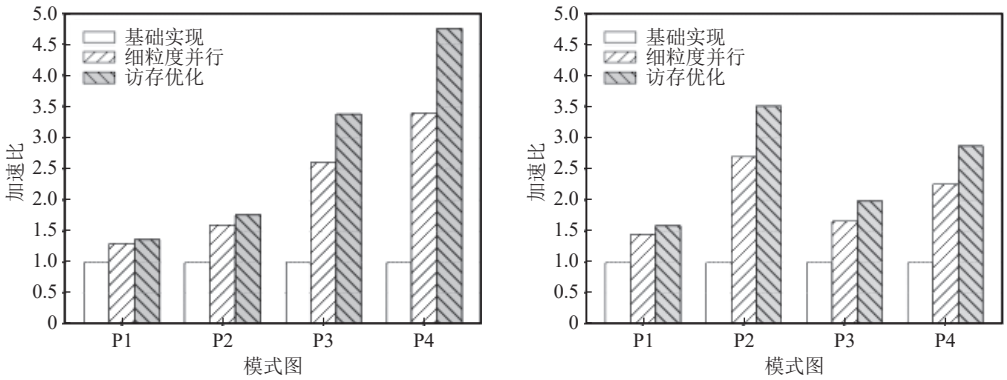
2）通信优化的性能分析。为了评估多 GPU 协同计算的场景下通信优化技术带来的性能提升，我们在 FR 和 DG 大图上对 4 个模式图（P5, P6, P7, P8）执行图挖掘任务。我们主要评测 2 种优化方法：第 1 种是数据缓存优化。该优化是在多 GPU 的设备内存上

Table 4 Execution Time Comparison on Frequent Subgraph Mining Applications

表 4 在频繁子图挖掘应用上的运行时间对比

带标签数据集	2-fsm				3-fsm			
	支持度	G²Miner/s	GraphSet/s	Saturn（本文）/s	支持度	G²Miner/s	GraphSet/s	Saturn（本文）/s
MI-29	0.3×10^3	0.84	0.25	0.14	10×10^3	运行失败	21.37	13.55
	0.5×10^3	0.78	0.23	0.09	15×10^3	运行失败	20.86	13.21
YT-29	1×10^3	3.65	2.14	1.27	1×10^3	运行失败	280.39	137.94
	5×10^3	3.38	1.97	0.82	5×10^3	运行失败	5 784	3 258

注：数据图为异构图，顶点包含不同标签，表示为数据图-标签数。

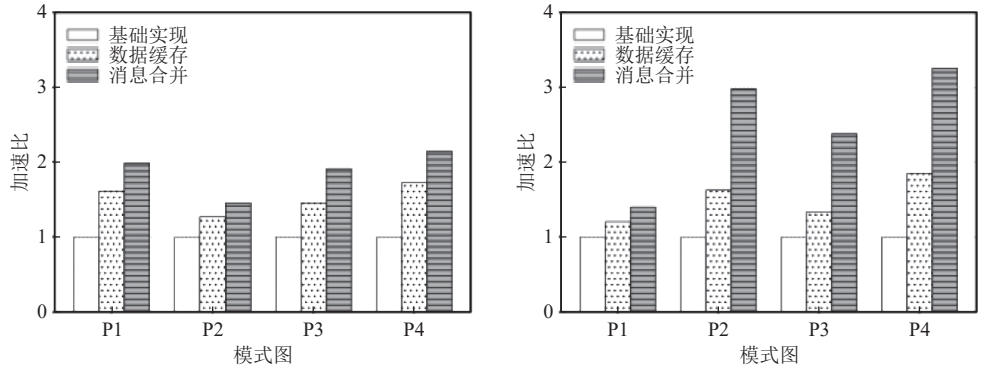


(a) 在YT图上的计算优化效果 (b) 在LJ图上的计算优化效果

Fig. 18 Computational optimization effect evaluation

图 18 计算优化效果评估

备份了高度数顶点的邻居数据,避免了 GPU 间频繁的高度数顶点通信。第 2 种是消息合并优化,该优化聚焦于多 GPU 间通信的链路竞争问题,将发往相同 GPU 的数据分桶,并在桶内将重复的邻居数据请求合并,在提升带宽利用率的同时降低了冗余数据通信。如图 19 所示,经过数据缓存优化,Saturn 系统的整体性能提升了 51.4%。在此基础上,叠加消息合并优化,可以带来平均 1.46 的加速比。



(a) 在FR图上的计算优化效果 (b) 在DG图上的通信优化效果

Fig. 19 Communication optimization effect evaluation

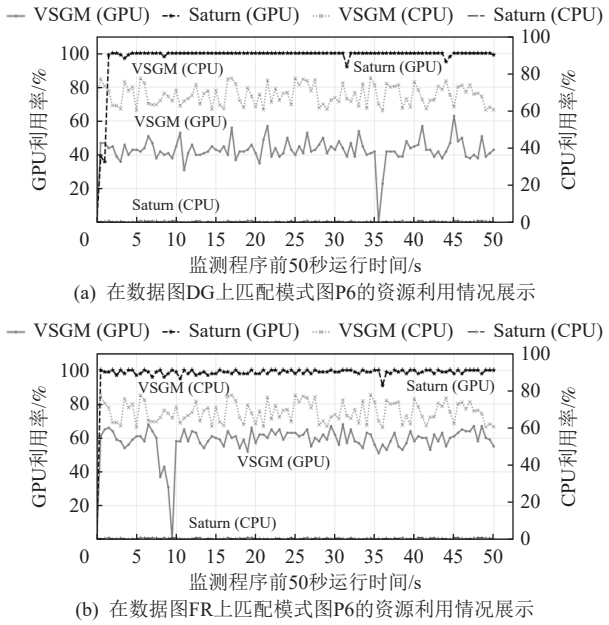
图 19 通信优化效果评估

3)GPU 整体利用率的分析。为探究不同 Multi-GPU 系统在图挖掘任务中的 CPU 和 GPU 资源利用效率,我们在配备 8 个 A100 GPU 的服务器上执行 P6 模式挖掘任务,并对比 VSGM 与 Saturn 系统的 CPU 和 GPU 整体利用率,见图 20。

实验数据显示,VSGM 系统受限于 CPU 与 GPU 间的数据传输瓶颈,在 FR 和 DG 这 2 个数据集上的平均 GPU 利用率仅为 42.9% 和 58.8%,峰值利用率亦仅达 68%。这主要归因于其执行图挖掘任务时,需频繁在 CPU 主存与 GPU 显存间进行数据搬移,导致 GPU 计算资源出现大量闲置。这种频繁的数据搬运不仅拉长了整体执行时间,也严重制约了 GPU 算力的实际发挥。在这一过程中,CPU 需频繁执行图分区的构建、索引与传输操作,导致 VSGM 在运行时 CPU 平均利用率高达 68.7%。

相较之下,Saturn 展现出更优的资源利用效率:一方面系统将 GPU 利用率稳定维持在 96% 以上,另一方面通过将计算与通信整合到单一完整内核中,无需 CPU 参与干预,使得系统运行时 CPU 利用率维持在 5% 以下。该优势得益于 2 个方面:其一,通过 CPU 无感的通信机制,将数据传输的控制路径与数据路径完全卸载至 GPU,规避 CPU 与 GPU 间的协同延迟;其二,系统对 GPU 计算单元进行细致调度,确保所有计算单元在图挖掘过程中保持计算或是通信状态,从而实现计算资源的高效复用。

4)系统的强扩展性。为验证 Saturn 系统在大规模计算环境下的强扩展性,我们在 1 台配备 NVLink 互连的 8 个 A100 GPU 服务器和 8 个节点构成的 32 卡 V100 GPU 集群上,针对 P2, P5, P6, P7 这 4 个模式图及 FR, DG 这 2 个数据图开展性能测试(其他数据



(a) 在数据图DG上匹配模式图P6的资源利用情况展示

(b) 在数据图FR上匹配模式图P6的资源利用情况展示

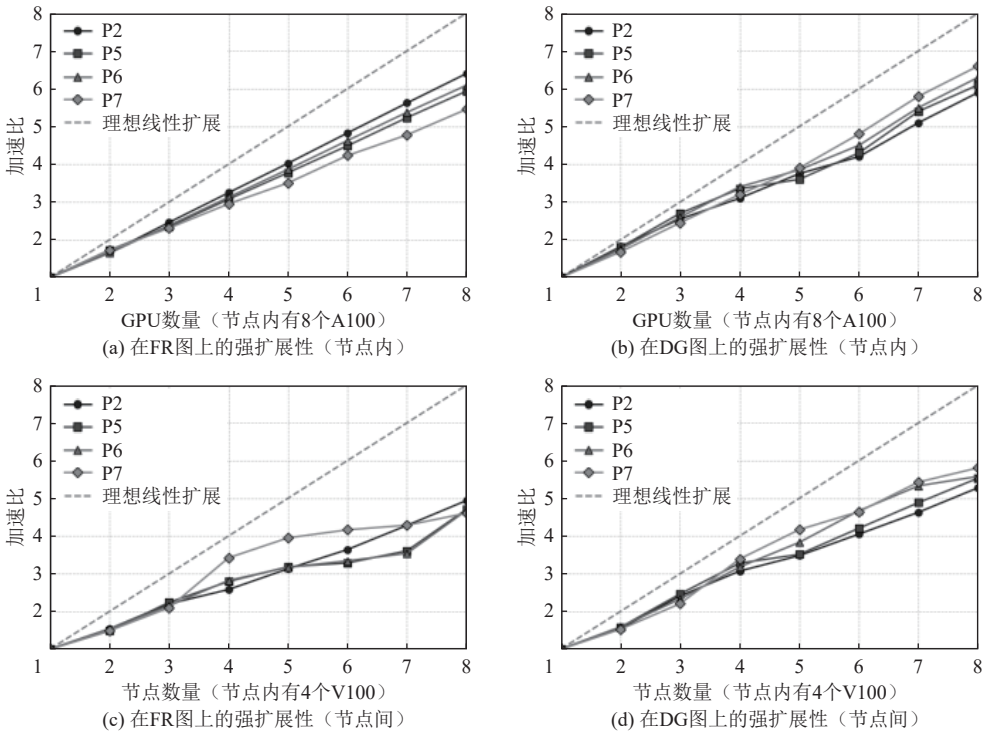
Fig. 20 Comparison of overall utilization for GPU and CPU in eight collaborative computing scenarios

图 20 在 8 卡协同计算场景下 GPU 和 CPU 整体利用率对比

集与模式图呈现相似趋势)。图 21 所示的实验结果显示, Saturn 系统在单机 8 卡 A100 和 32 卡 V100 的不同配置下均展现出优异的并行处理能力。其中, 在 8 卡 A100 的单节点内 Saturn 的并行效率在不同查询场景中稳定处于 73.8%~82.5%, 平均并行效率达 77.8%。而在 32 卡 V100 的分布式场景中, Saturn 的并行效率稳定于 57.5%~72.6%, 平均并行效率达 64.2%。这一方面主要归功于我们的 CPU 无感的通信策略, 避免了在 GPU 执行图挖掘任务过程中 CPU 的频繁干预。另一方面, 我们通过数据缓存和消息合并优化, 有效减少冗余数据传输与链路资源竞争, 实现多 GPU 间的高效协同计算。

4.4 系统微基准测试分析

1)GPU 线程组利用率。在单 GPU 环境下执行图挖掘任务时, 线程组内线程利用率是衡量计算效率的关键指标。为验证 Saturn 系统的细粒度并行技术相对于传统算子内并行方法的优势, 我们在图 22 中对比展示了不同数据图(MI 和 LJ)与模式图(P2, P3, P4)组合下的线程利用率表现。传统的“算子内并行-算子间串行”计算模式在处理小度数顶点的查询任务时存在显著缺陷, 即线程组内大量线程因等待数据或任务分配而处于空闲状态, 导致整体利用率低下。实验数据显示, 该模式下线程利用率平均值仅为 22.4%, 在处理复杂模式图(如 P4)或低平均度数数据图(如 MI, 利用率为 19.5%)时, 利用率最低可降至 13.8%。Saturn 系统的细粒度并行优化通过重构查询阶段的执行逻辑, 显著提升了线程利用率。该技术将同一层循环内的多个集合操作进行并行化处理,



(a) 在FR图上的强扩展性（节点内）

(b) 在DG图上的强扩展性（节点内）

(c) 在FR图上的强扩展性（节点间）

(d) 在DG图上的强扩展性（节点间）

Fig. 21 Scalability of Saturn system on Multi-GPU platform

图 21 Saturn 系统在 Multi-GPU 平台上的可扩展性

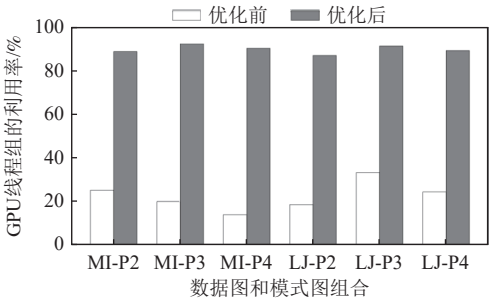


Fig. 22 Evaluation of thread group utilization on single GPU

图 22 单 GPU 上线程组利用率评估

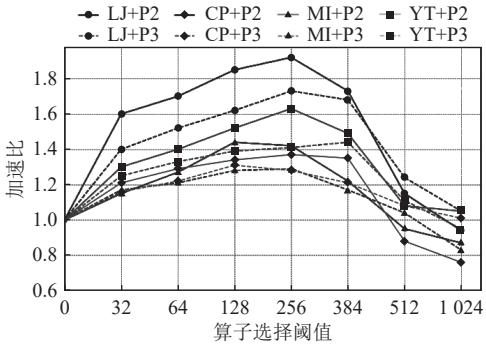


Fig. 23 Impact of various thresholds on hybrid operator selection

图 23 不同阈值对混合算子选择的影响

确保每个线程在执行周期内均有明确的查询任务。优化后的数据表明, 线程组内利用率平均达到89.98%, 峰值利用率可达 92.5%, 且在不同数据图与模式图组合下均保持稳定的高利用率表现。这一改进不仅解决了传统模式中的线程空闲问题, 还通过任务的动态分配机制有效平衡了计算负载, 实现了GPU 资源的高效利用。

2)混合算子策略的阈值选择。图 23 呈现了不同度数阈值对混合算子策略性能的影响。实验以单 GPU 配置下选取 P2 和 P3 这 2 个模式图, 在 LJ, YT, MI, CP 这 4 个数据集上开展评估, 度数阈值范围设定为 0~1 024。其中, 阈值 0 对应二分查找法, 其性能作为实验基准。结果表明, 当模式图 P2 和 P3 的度数阈值设为 256 时, 平均加速比分别达到 1.58 与 1.42; 而当阈值处于 64~384 时, 可至少实现 1.4 的加速效果。基于上述结果, 系统性能对比实验中, 混合算子策略的静态度数阈值统一设置为 256 已足够展示 Saturn 的优势。尽管采用更精准的自适应阈值能进一步挖掘潜力, 但其涉及的算法复杂度和运行时开销与本研究的算子优化相互正交。因此, 本文将阈值 256 作为实验评估的标准配置, 而更复杂的自动化参数寻优机制可作为未来的研究方向进一步探索。

3)运行时开销评估。为阐明细粒度并行优化与多卡通信优化不会产生过多开销, 我们在 2 个数据图与 2 个模式图上开展图挖掘任务, 以此评估运行时间和内存占用状况。图 24 呈现了多 GPU 场景下的运行时间开销与内存开销。执行时间分别对计算时间、通信时间以及额外开销时间进行统计, 这 3 部分的平均占比依次为 4%, 37%, 59%; 额外开销源于 2 方面: 一是计算索引更新过程中所依赖的 Warp 内线程广播通信, 二是通信内存管理与消息聚合操作, 这 2 类操作仅涉及本地显存的索引更新, 不会产生跨设备的数据传输开销, 故而额外开销占比处于较低水平。运行时内存消耗主要分为 3 个部分: 用于临时通信数据存放的通信内存、跨节点高度邻居数据的缓存内存和用于本地执行图挖掘任务的计算内存,

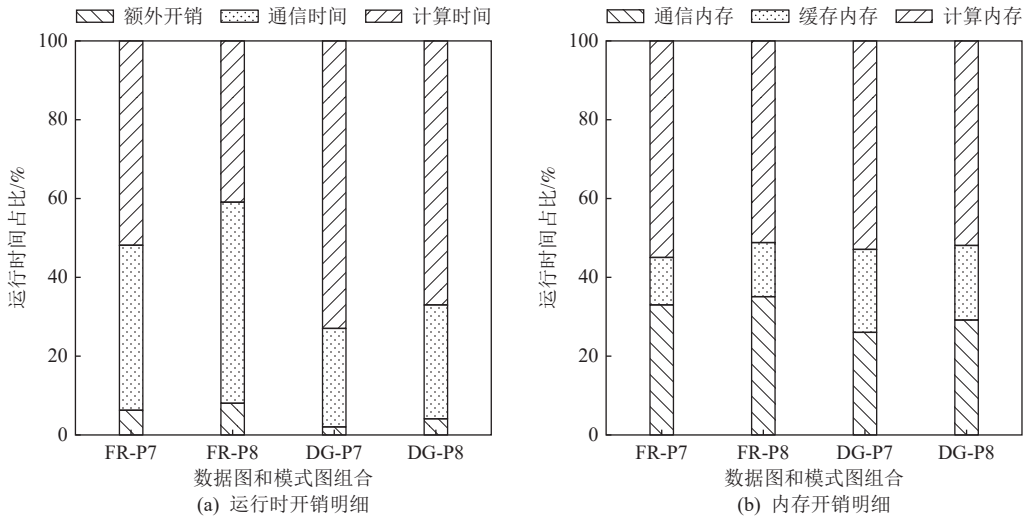


Fig. 24 Details of runtime and memory cost

图 24 运行时间和内存开销明细

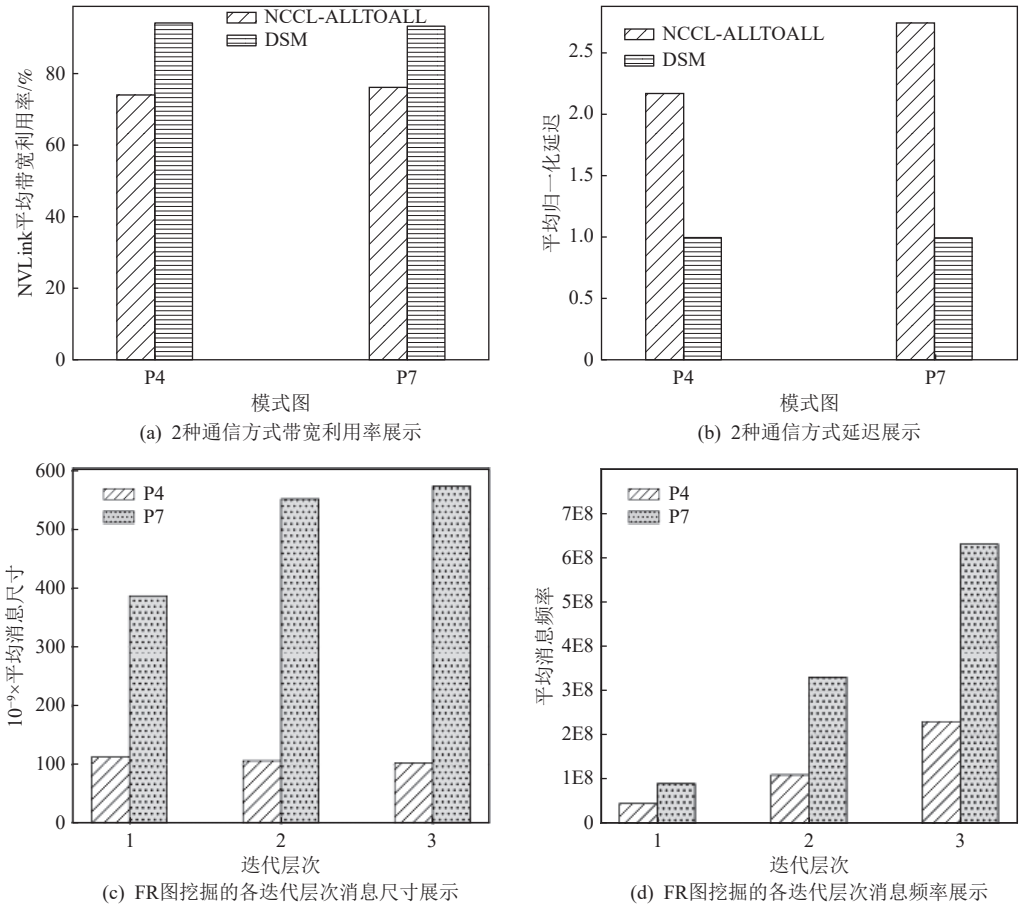
它们的平均占比分别为 31%, 16%, 53%。借助 Saturn 的共享内存机制, 在维持内存分布稳定的同时, 还能确保较低的运行时通信开销, 实现了性能与内存开销的良好平衡。

4) 系统通信行为分析。为测试 Saturn 的通信效率, 我们在 FR 数据图上使用 P4, P7 这 2 个模式图执行 4 个 GPU 协同计算的图挖掘任务, 重点评估 3 方面性能: 统一内存架构下的 NVLink 带宽利用率与平均延迟, 以及不同迭代层次的平均消息尺寸和通信频率。实验对比了传统通信库 NCCL 与 NVSHMEM 在多 GPU 图挖掘任务中的通信效率, 结果如图 25 所示, NCCL 的平均带宽利用率为 77%, 而 NVSHMEM 可达 88%; 由于 NCCL 需要 CPU 进行多次显式同步与数据重排, 其平均延迟达 NVSHMEM 的 2.4 倍。另一方面, 由于图挖掘这类算法的任务特性, 消息大小通常较小(例如, 本实验中小于 600×10^9), 但消息频率较高, 随迭代层数增加, 消息尺寸与频率均呈增长趋势, 我们的消息聚合机制能够将小消息合并且消除重复邻居数据访问, 从而能有效提升通信效率。这也进一步验证了基于 NVSHMEM 与消息聚合的设计更适配图挖掘高频率、小消息的通信特征。

5 相关工作

5.1 CPU 上的图挖掘系统

Arabesque[23] 是提出“按实例思考”(think like embedding, TLE)编程抽象的分布式图模式匹配系统。该系统采用广度优先搜索算法逐步生成部分匹配实例, 该方法虽然能增量得到匹配结果, 但会导致较高的内存开销。Peregrine[6] 提出了基于模式感知(pattern aware)的图挖掘系统。该系统通过直接计算最终结果的方式在单节点上处理大规模图数据, 避免对无关子图的无效遍历。AutoMine[5] 将图模式挖掘问题形式化为集合问题, 通过集合代数(如集合求交、集合求差等)实现图挖掘算法。该系统基于关系代数生成最优查询计划, 并通过调整匹配顺序, 在匹配初始阶段对伪候选集(false candidates)进行剪枝, 同时复用中间计算结果以提升搜索效率。GraphPi[8] 旨在解决模式图的对称性导致的大量冗余计算和不同匹配顺序对性能的影响。该系统构建了精确的性能预



(a) 2种通信方式带宽利用率展示 (b) 2种通信方式延迟展示

(c) FR图挖掘的各迭代层次消息尺寸展示 (d) FR图挖掘的各迭代层次消息频率展示

Fig. 25 System communication behavior analysis

图 25 系统通信行为分析

测模型,动态确定最优匹配顺序与非对称约束组合,实现高效图模式挖掘。Khuzdul[20]是一个基于CPU多节点的分布式图挖掘系统。该系统提出了可扩展实例(extendable embedding)的编程抽象来表达模式枚举算法,支持细粒度的任务调度和计算通信掩盖,同时采用了静态的缓存策略降低多节点间的通信负担。现有的CPU系统存在显著性能瓶颈,其设计侧重抽象机制与编程灵活性,未能有效利用异构设备的高并发特性。

5.2 单GPU上的图挖掘性能优化

Pangolin[24]是一个GPU上的图挖掘系统,其针对GPU架构提出了基于"扩展—规约—过滤"(extend-reduce-filter)的执行模型,并设计了结构化数组(structure of arrays, SoA)的子图存储结构适配GPU的高并发特点。但由于其编程接口不够灵活,系统仅能支持有限种类的图挖掘应用。G^{2}Miner[10]根据数据图、模式图和GPU架构这3个维度的特征为用户自动生成高性能的GPU代码,其在GPU内核函数中采用边上任务划分结合GPU的Warp任务并行,同时实现了基于Warp原语的集合求交算子。STMatch[25]和EGSM[26]考虑了GPU上图挖掘的负载均衡问题。其中,每个线程组通过维护独立的栈结构,并通过任务窃取机制拆分搜索任务。STMatch采用单纯的深度优先搜索方案而TDFS和EGSM则采用了广度搜索和深度搜索的混合方案。GraphFold[27]聚焦于GPU在处理图挖掘任务的冗余搜索问题,采用合并匹配(fused matching)算法避免了重复的集合操作。然而,这些GPU上的图挖掘系统仅考虑整个数据图存放在设备内存的场景,无法处理超出单个显存容量的图数据。

5.3 多GPU协同计算的图处理系统

在传统的图计算领域,张珩等人[28]设计了基于Multi-GPU平台的图计算系统,提出了Grid图数据切分策略,动态地从SSD加载图数据至多个GPU上并行处理。蒋筱斌等人[29]提出了混合感知的细粒度通信模型,优化多GPU间的通信效率以支持大规模图处理(最短路径、PageRank等图计算应用)。然而,图挖掘问题与图计算任务特征的不同导致优化技术难以迁移。在图挖掘领域,PBE[14]将图数据保存在CPU主存上,由CPU负责对大图数据进行切分,将分区数据分别加载给多个GPU设备进行并行处理。VSGM[15]提出了基于视图(view-based)的技术来掩盖分数数据的计算和通信从而提升GPU的利用率。G2-AIMD[16]使用拥塞控制算法改进了VSGM的调度流程,降低了GPU产生中间结果的显存开销。但现有的基于多GPU的图挖掘方案均面临显著的CPU-GPU数据传输开销问题,且普遍忽略GPU间的高效通信链路。

5.4 通用图处理系统

当前图处理领域已涌现出众多功能分化的图系统,涵盖图计算、图挖掘及图学习等核心方向。在传统图计算领域,PowerGraph[30], Pregel[31], GraphX[32]等经典系统,主要聚焦于BFS, Pagerank等基础图遍历算法的高效执行;而图学习领域中,DGL[33], PyTorch Geometric[34]等主流框架,则提供了GCN[35], GIN[36], GAT[37]等丰富的图神经网络模型,支撑深度学习任务开展。为实现复杂图任务的端到端处理,这类系统常需协同工作,例如Yu等人[38]设计的Vineyard系统,致力于提供通用的图处理支撑能力。以欺诈检测场景为例,通常先通过图计算系统的遍历功能从海量用户数据中构建图结构,再将生成的数据图输入图挖掘系统,借助K-团识别、环形模式匹配等图挖掘算法产出分析结果并为图中顶点赋值属性,最后将带属性的数据图输入图学习系统以完成诈骗团伙预测。本文设计的Saturn系统接收数据图和模式图为输入,提供结构查询功能,输出顶点属性特征,为下游图学习任务的高效执行提供支撑。

6 结 论

本文针对图模式挖掘在GPU加速中面临的线程利用率低、多GPU通信开销大等挑战,提出了高效的GPU图模式挖掘系统Saturn。通过深入分析现有并行优化方法的缺陷,我们设计了细粒度并行策略与混合算子优化,将集合操作的加载与查询解耦,动态适配哈希与二分查找策略,显著提升了GPU线程利用率,有效解决了幂律图数据下小度数顶点处理时的线程空闲问题。在多GPU协同方面,提出CPU无感的全GPU通信模式,利用NVLink高带宽链路和统一内存机制,结合高度数顶点缓存与消息聚合分桶调度,大幅降低了跨设备通信开销,避免了CPU介入导致的计算中断与PCIe数据传输瓶颈。实验结果表明,Saturn系统在多个真实数据集上实现了对主流GPU图挖掘系统的性能超越,不仅提升了单GPU环境下的计算并行效率,还在多GPU场景中展现出优异的通信能效与可扩展性。

作者贡献声明:林志恒提出了方案、完成实验并

撰写论文；孟轲指导论文思路与写作；谭光明提供研究思路并给予指导意见。

参 考 文 献

[1] Spirin N, Han Jiawei. Survey on Web spam detection: Principles and algorithms[J]. ACM SIGKDD Explorations Newsletter, 2012, 13(2): 50−64.

[2] Guo Zhichun, Guo Kehan, Nan Bozhao, et al. Graph-based molecular representation learning[C]//Proc of the 32nd Int Joint Conf on Artificial Intelligence (IJCAI'23). New York: ACM, 2023: 6638−6646

[3] Ranu S, Singh A K. Indexing and mining topological patterns for drug discovery[C]//Proc of the 15th Int Conf on Extending Database Technology (EDBT'12). New York: ACM, 2012: 562−565

[4] Akoglu L, Tong H, Koutra D. Graph based anomaly detection and description: A survey[J]. Data Mining and Knowledge Discovery, 2015, 29(3): 626−688

[5] Mawhirter D, Wu B. AutoMine: Harmonizing high-level abstraction and high performance for graph mining[C]//Proc of the 27th ACM Symp on Operating Systems Principles (SOSP '19). New York: ACM, 2019: 509−523

[6] Jamshidi K, Mahadasa R, Vora K. Peregrine: A pattern-aware graph mining system[C]//Proc of the 15th European Conf on Computer Systems (EuroSys '20). New York: ACM, 2020: 13: 1–13: 16

[7] Mawhirter D, Reinehr S, Holmes C, et al. GraphZero: A high-performance subgraph matching system[J]. SIGOPS Operating Systems Review, 2021, 55(1): 21−37.

[8] Shi Tianhui, Zhai Mingshu, Xu Yi, et al. GraphPi: High performance graph pattern matching through effective redundancy elimination[C]//Proc of the 32nd Int Conf for High Performance Computing, Networking, Storage and Analysis (SC'20). Piscataway, NJ: IEEE, 2020: 1−14

[9] JEDEC. High bandwidth memory (HBM) DRAM[EB/OL]. (2021-03-01)[2025-06-13]. https://www.jedec.org/standards-documents/docs/jesd235a

[10] Chen Xuhao, Arvind. Efficient and scalable graph pattern mining on GPUs[C]//Proc of the 16th USENIX Symp on Operating Systems Design and Implementation (OSDI'22). Berkely, CA: USENIX Association, 2022: 857−877

[11] Shi Tianhui, Zhai Jidong, Wang Haojie, et al. GraphSet: High performance graph mining through equivalent set transformations[C]//Proc of the 35th Int Conf for High Performance Computing, Networking, Storage and Analysis (SC'23). New York: ACM, 2023: 32: 1−32: 14

[12] Pandey S, Li Xiaoye, Buluc A, et al. H-index: Hash-indexing for parallel triangle counting on GPUs[C]//Proc of the 23rd IEEE High Performance Extreme Computing Conf (HPEC). Piscataway, NJ: IEEE, 2019: 1−7

[13] Hu Yang, Liu Hang, Huang H. Tricore: Parallel triangle counting on GPUs[C]//Proc of the 30th Int Conf for High Performance Computing, Networking, Storage and Analysis (SC'18). Piscataway, NJ: IEEE, 2018: 171−182

[14] Guo Wentian, Li Yuchen, Sha Mo, et al. GPU-accelerated subgraph enumeration on partitioned graphs[C]//Proc of the 40th ACM SIGMOD Int Conf on Management of Data (SIGMOD'20). New York: ACM, 2020: 1067–1082

[15] Jiang Guanxian, Zhou Qihui, Jin Tatiana, et al. VSGM: View-based GPU-accelerated subgraph matching on large graphs[C]//Proc of the 34th Int Conf for High Performance Computing, Networking, Storage and Analysis (SC'22). Piscataway, NJ: IEEE, 2022: 1−15

[16] Yuan Lyuheng, Ahmad A, Yan Da, et al. G2-AIMD: A memory-efficient subgraph-centric framework for efficient subgraph search on GPUs[C]//Proc of the 40th IEEE Int Conf on Data Engineering. Piscataway, NJ: IEEE, 2024: 3164−3177

[17] Pandey S, Wang Zhibin, Zhong Sheng, et al. Trust: Triangle counting reloaded on GPUs[J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 32(11): 2646−2660

[18] Wang Zhibin, Meng Ziheng, Li Xue, et al. Smog: Accelerating subgraph matching on GPUs[C]//Proc of the 27th IEEE High Performance Extreme Computing Conf (HPEC). Piscataway, NJ: IEEE, 2023: 1–7

[19] Iosup A, Hegeman T, Ngai W L, et al. LDBC graph-analytics: A benchmark for large-scale graph analysis on parallel and distributed platforms[J]. Proceedings of the VLDB Endowment, 2016, 9(13): 1317−1328

[20] Chen Jingji, Qian Xuehai. Khuzdul: Efficient and scalable distributed graph pattern mining engine[C]//Proc of the 28th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems (ASPLOS'23). New York: ACM, 2023: 413−426

[21] Leskovec J, Krevl A. SNAP datasets: Stanford large network dataset collection[EB/OL]. 2014[2025-06-13]. http://snap.stanford.edu/data

[22] Lin Hu, Lin Yinnian, Zou Lei, et al. A graph pattern mining framework for large graphs on GPU[J]. The VLDB Journal, 2025, 34(1): 117−143.

[23] Teixeira C H C, Fonseca A J, Serafini M, et al. Arabesque: A system for distributed graph mining[C]//Proc of the 25th Symp on Operating Systems Principles (SOSP'15). New York: ACM, 2015: 425–440

[24] Chen Xuhao, Dathathri R, Gill G, et al. Pangolin: An efficient and flexible graph mining system on CPU and GPU[J]. Proceedings of the VLDB Endowment, 2020, 13(8): 1190−1205

[25] Wei Yihua, Jiang Peng. STMatch: Accelerating graph pattern matching on GPU with stack-based loop optimizations[C]//Proc of the 34th Int Conf for High Performance Computing, Networking, Storage and Analysis(SC'22). Piscataway, NJ: IEEE, 2022: 53: 1–53: 13

[26] Sun Xibo, Luo Qiong. Efficient GPU-accelerated subgraph matching[J]. Proceedings of the ACM on Management of Data, 2023, 1(2): 181: 1−181: 26

[27] Lin Zhiheng, Meng Ke, Shui Chaoyang, et al. Exploiting fine-grained redundancy in set-centric graph pattern mining[C]//Proc of the 29th ACM SIGPLAN Annual Symp on Principles and Practice of Parallel Programming (PPoPP'24). New York: ACM, 2024: 175–187

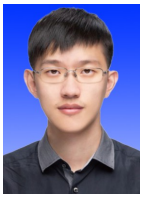
[28] Zhang Heng, Zhang Libo, Wu Yanjun. Large-scale graph processing on Multi-GPU platforms[J]. Journal of Computer Research and Development, 2018, 55(2): 273−288 (in Chinese)
(张珩, 张立波, 武延军. 基于 Multi-GPU 平台的大规模图数据处理[J]. 计算机研究与发展, 2018, 55(2): 273−288)

[29] Jiang Xiaobin, Xiong Yixiang, Zhang Heng, et al. ChattyGraph: Highly scalable graph computing system for heterogeneous multi

accelerators[J]. Journal of Software, 2023, 34(4): 1977−1996 (in Chinese)
(蒋筱斌, 熊轶翔, 张珩, 等. ChattyGraph: 面向异构多协处理器的高可扩展图计算系统[J]. 软件学报, 2023, 34(4): 1977−1996)
[30] Gonzalez Joseph E, Low Y, Guerin R, et al. PowerGraph: Distributed graph-parallel computation on natural graphs[C]//Proc of the 10th USENIX Symp on Operating Systems Design and Implementation (OSDI 12). Berkeley, CA: USENIX Association, 2012: 17−30
[31] Malewicz G, Austern M H, Bik A J C, et al. Pregel: A system for large-scale graph processing[C]//Proc of the 30th ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2010: 135−146
[32] Gonzalez J E, Xin R S, Dave A, et al. GraphX: Graph processing in a distributed dataflow framework[C]//Proc of the 11th USENIX Symp on Operating Systems Design and Implementation (OSDI 2014). Berkeley, CA: USENIX Association, 2014: 599−613
[33] Wang Minjie, Yu Ling, Zheng Da, et al. Deep graph library: Towards efficient and scalable deep learning on graphs[J]. arXiv preprint, arXiv: 1909.01315, 2019
[34] Fey M, Lenssen J. Fast graph representation learning with PyTorch Geometric[J]. arXiv preprint, arXiv: 1903.02428, 2019
[35] Zhang Si, Tong Hanghang, Xu Jiejun, et al. Graph convolutional networks: A comprehensive review[J]. Computational Social Networks, 2019, 6(1): 1−23
[36] Xu Keyulu, Hu Weihua, Leskovec J, et al. How powerful are graph neural networks?[J]. arXiv preprint, arXiv: 1810.00826, 2018
[37] Veličković P, Cucurull G, Casanova A, et al. Graph attention networks[J]. arXiv preprint, arXiv: 1710.10903, 2017
[38] Yu Wenyuan, Zhang Liang, Yang Chao, et al. Vineyard: Optimizing data sharing in data-intensive analytics[J]. Proceedings of the ACM on Management of Data, 2023, 1(2): 1−27



Lin Zhiheng, born in 1997. PhD. Student member of CCF. His main research interests include high-performance computing, heterogeneous computing, and large-scale graph data processing.
林志恒, 1997年生。博士。CCF学生会员。主要研究方向为高性能计算、异构计算、大规模图数据处理。



Meng Ke, born in 1993. PhD, associate professor. Member of CCF. His main research interests include high-performance computing, parallel algorithm design and optimization, and graph computing.
孟　轲, 1993年生。博士, 副研究员。CCF会员。主要研究方向为高性能计算、并行算法设计与优化、图计算。



Tan Guangming, born in 1980. PhD, professor. Senior member of CCF. His main research interests include parallel programming and algorithms design, domain-specific architecture, bioinformatics, and big data.
谭光明, 1980年生。博士, 研究员。CCF高级会员。主要研究方向为并行程序与算法设计、领域专用体系结构、生物信息学、大数据。