图划分问题研究综述

李其锟

[1)](华中科技大学计算机科学与技术学院 湖北 武汉 430074-)

摘 要 图的划分在许多具有关系数据的大规模分布式计算应用中属于关键步骤。随着现有图数据集在大小和密度上的不断增长，一系列高度可扩展的平衡分区算法逐渐被提出，以满足不同领域的不同需求。作为现有工作的总结，图划分相关的两个迭代分区器家族—基于重新分组的和基于平衡标签传播的（包括 Facebook 的社交哈希分区器）—可以从通用模块化框架的设计决策角度来看待，并且借助于这个模块化的视角，我们可以从更加抽象的角度进行算法研究。除了算法本身，数据处理也是图划分问题中重要的一环。高效的流式图处理系统通过更新计算结果来利用增量处理以反映最新图快照的结构变化。尽管某些基于单调路径的算法通过数值比较并优化中间值来产生正确的结果，但对于需要 BSP 语义的算法来说，直接重用图变化前的计算值并不正确。由于图流中的结构变化导致中间结果无法使用，因此在提供同步处理保证的同时进行增量计算是一个具有挑战性的问题。与此同时，在硬件辅助加速领域，由于图分析经常涉及计算密集型操作，因此常需要使用 gpu 进行加速处理。然而，在社交网络、网络安全和欺诈检测等许多应用中，它们的代表图经常发生变化，人们不得不在 gpu 上重建图结构来整合更新。因此，使用硬件进行图的重建成为高速图处理的瓶颈。对硬件进行规划设计以寻求性能提升也属于解决图划分问题的一个研究方向。

关键词 图算法；图处理；图划分；流划分；分布式系统；

Survey on Graph Partitioning problem

LI Qikun[1)]

[1)](School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074)

Abstract Graph partitioning is a key step in many large-scale distributed computing applications with relational data. With the increase in size and density of existing graph data sets, a series of highly scalable balanced partitioning algorithms have gradually emerged to meet the different needs in different fields. As a summary of existing work, the two iterative partitioner families related to graph In addition to the algorithm itself, data processing is also an important part of the graph partition problem. An efficient streaming graph processing system uses incremental processing to reflect the graph structure changes of the latest graph snapshot by updating the calculation results. Although some algorithms based on monotonic paths produce correct results through numerical comparison and refinement of intermediate values, for algorithms that require BSP semantics, it is not correct to directly reuse the value calculated before the mutation. Because the structural mutation in the graph stream makes the intermediate results unusable, it is challenging to use incremental calculations while

providing synchronization guarantees. At the same time, in the field of hardware, since graph analysis often involves computationally intensive operations, GPUs are often used for acceleration. However, in many applications such as social networks, network security, and fraud detection, their representative graphs often evolve, and people have to rebuild graph structures on GPUs to integrate updates. Therefore, the use of hardware for image reconstruction has become a bottleneck in high-speed image processing. Planning and designing hardware to seek performance improvement is also a research direction to solve the problem of graph partitioning

Key words Graph Algorithms; Graph Processing; Graph Partitioning; Stream Partitioning: Distributed Systems

1 引言

图是计算机科学中普遍存在的一种数据结构，用于表示大量真实世界系统，包括社会和信息网络、生物网络以及物理模拟中的网状域。这类系统的规模会随着时间不断增长，特别是在与在线社交数据相关的领域。现代万维网拥有数百亿个网页（节点），它们之间有数万亿个链接（边）。Facebook 每月为数十亿活跃用户提供服务，加上数以亿计的页面、事件和群组，所有这些都通过网络结构相互交互。类似地，Twitter 每月也有数亿活跃用户通过分享和喜欢彼此的内容进行互动。在所有这些例子中，在排名和推荐问题服务中最显著的全图计算是许多产品和服务核心功能的核心。

不幸的是，大规模计算的成本很高；这些图往往需要占用数以 TB 级的存储空间，而且对这些数据集进行的大多数计算都难以用一台机器来完成。这个问题的典型解决方案是将输入图进行划分并配给多台机器，同时对这些计算使用并行算法，从而降低网络延迟并提高运行时的计算效率。

这样一来，问题就变成了如何“最佳”划分图以获得这些性能增益？这个问题的答案通常是非常具体的。事实上，一些问题最好通过对节点集进行划分来解决，而另一些问题最好通过对边集进行划分来解决。在现有工作中，一种通常思路是重点关注通过划分节点集（无需复制）来驱动的应用程序，以及有效划分大型经验图的节点集的方法，这是一项困难的任务。作为进一步工作的铺垫，平衡节点集划分最近被用于因果推理，通过一个称为图聚类随机化的过程来改进网络环境中的经验设计；这一工作领域特别激励了对非常大的 k 的良好 k 路平衡划分算法的探索。

节点集划分的一种常见方法是对节点集进行简单的散列，有效地在集群（或机器）中随机均匀地分布节点。但更智能的划分方法可以极大地提高这些分布式算法的运行时间。与一般的图聚类任务相比，此分区任务的一个重要要求是平衡分区中每个集群相关联的计算负载。在这项工作中，通常是将考虑重点放在每个节点的计算负载恒定的情况，但要求选择的算法能够较容易地修改，以适应非均匀/加权负载的情况。

进入我们感兴趣的问题，平衡图划分：给定一个输入图，如何求出节点集的划分使得（1）在 k 个集群或分片上保持平衡负载，同时（2）最小化给定的目标函数。一种通常思路是关注边划分目标——最小化跨越多个分片的边数量，这与该领域的过往工作紧密一致。需要认识到，在实践中，最小化边切割可能无法完全描述工作负载性能，但我们使用此目标作为替代，并对各种设计决策对结果的影响进行分类。目标函数的其他示例包括超图的扇出最小化和图簇随机化中的方差最小化。

不幸的是，找到边划分问题的精确解即使对于是普通图往往也是不可行的：当分片数为 2 时，该问题等同于最小对分问题，这是典型的 NP 难问题，并且没有已知的具有良好近似保证的有效算法。这就是说，在一系列相关的大规模图形数据集上，有大量关于实用（尽管是启发式）算法的工作，这些算法在经验上表现良好。

2 原理和优势

最近关于图划分的可扩展实用算法的研究主要是由管理世界上一些最大关系数据集的公司的研究推动的。在这项工作中，围绕三种主要的算法大类可以构建一个通用框架，这三类算法通常被视为处于或接近最新水平，并且往往用于不同的目标：平衡标签传播算法（BLP）、Social Hash 划分算法（SHP）和重分组线性确定贪婪算法（reLDG）。

BLP 和 SHP 属于基于标签传播的一系列算法。从初始分配开始，它们迭代地执行节点重新定位，以实现更高质量的分区。ReLDG 是所谓的重新排列算法的一个例子，它重复并且连续的处理节点集，每个节点依照设计的用于实现平衡的分配规则来放置。流算法通常由受到高度限制的计算框架来驱动，在这种框架中，当图在传输、移动和/或加载时会进行节点分配。在过往工作中测试过的节点集的顺序有随机、广度优先搜索（BFS）和深度优先搜索（DFS），以模拟网络爬虫或等效过程获得的顺序。现存的工作有以下几个方面（1）针对可伸缩的非流算法进行基准测试，（2）探索策略流排序，即算法考虑节点集的顺序。可以称这些算法为平衡图分区的优先重分组算法。

BLP 算法的相关工作围绕半监督学习和社交检测的标签传播的现有工作进行了有限的研究。它对节点集的初始可行划分（标记）进行迭代、并不断改进，直到达到平衡（或达到最大迭代次数）。这项工作使用最简单的初始化随机平衡分配与其他方法进行比较，尽管仔细初始化已被证明能够实现更好的平衡切割，但初始化的方式仍取决于决策时的环境和可用的元数据。

每个迭代如下进行：对于每个节点 u∈V，计算其移动增益，如果单侧重新定位，则将共定位邻居计数的最大增益定义为

$$C(P) = \{(u, v) \in E \mid P(u) \neq P(v)\},$$

对于所有的 u∈V，显然有$g_u \geq 0$。当$g_u = 0$ 时，节点 u 有效地“满足”，并保持其分片赋值。当$g_u > 0$ 时，将节点放入队列中，按增益递减的顺序移动到目标分片。这些信息被导入一个线性程序中，该程序在迭代过程中解决循环问题，确定从这些队列中移动的最大顶级节点数，从而在遵守每个分片大小的约束的同时获得最大的收益。为所有分片对执行这些节点重新定位构成一个迭代，不断重复该过程，直到没有节点需要移动，或者达到最大迭代次数为止。

在数据处理方向，过去十年中，人们对设计高效算法来处理数据流模型中的大量图产生了极大的兴趣。这包括 PageRank 样式的分数、连接性、spanners、子图计数（如三角形和摘要等问题。然而，这些工作主要集中在线性有界空间的最佳逼近解的理论研究。部分提出的方法较容易地结合现有的图流算法，因为其设计的存储方案可以支持现有算法中使用的大多数图表示。现在已经提出了许多流式数据处理系统，例如 Storm、Spark streaming、Flink。在大规模并行性能的吸引下，一些成功的尝试性工作已经证明了使用 GPU 加速数据流处理从而提升图划分效率的方案的优势。

然而，上述系统主要集中在一般的流处理上，缺乏对图流处理的支持。Stinger 是一个并行解决方案，用于支持单机上的动态图分析。最近，Kineograph、CellIQ 和 GraphTau 等系统被提出以用于解决分布式场景下随时间变化的图的处理需求。然而，现有的集中在基于 CPU 的时间演化图处理的工作将在 GPU 上不在高效，因为 CPU 和 GPU 是两种架构，在并行执行中具有不同的设计原则和性能关注点。

图分析处理本质上是数据密集型和计算密集性的。大规模并行 GPU 加速器功能强大，可实现许多应用程序的最高性能。与 CPU 这种具有大缓存容量和高单核处理能力的通用处理器相比，GPU 将其大部分芯片面积用于大量简单算术逻辑单元（ALU），并以 SIMT（单指令多线程）方式执行代码。在大量 ALU 的帮助下，GPU 在具有充分并行性的应用程序中展现出比 CPU 高几个数量级的计算吞吐量。这引发了一系列研究以探索使用 gpu 来加速图分析和显示巨大的潜力。示例如广度优先搜索（BFS）、子图查询、PageRank 和许多其他算法。在 GPU 上部署特定图算法的成功推动了通用 GPU 图处理系统的设计，如 Medusa 和 Gunrock。然而，上述面向 GPU 的图算法和系统都以静态图为前提。为了处理动态图场景，现有的工作必须在图的更新到达时在 gpu 上执行图重建。根据现有工作，DCSR 是唯一的解决方案，由于它基于边块链表和尾部追加技术, 因此它是为仅插入场景而设计的，即它不支持删除或高效搜索。由此出现的

GPMA 系统能够以细粒度的方式在 gpu 上实现高效的动态图更新(即插入和删除操作)。此外，现有的图分析和为 gpu 优化的系统可以使用 GPMA 系统直接轻松替换它们的存储层，因为现有工作中使用的基本图存储方案可以直接在 GPMA 系统相关工作提出的存储方案之上实现

3 研究进展

3.1 基于矛盾值的重组线性确定性贪婪算法（ReLDG）

重组线性确定性贪婪算法（ReLDG）属于迭代算法的一个称为重复流算法的子类。这个类由单通道线上图所驱动，在此类图的相关运行环境中，程序解析一个图文件，并串行地将图数据从源读取到目标集群。有相关工作提出了该方法的多通道/迭代版本，并考虑得出结论:重新分组方法可以通过改进以取得与离线、非流式算法相似的竞争力。

ReLDG 算法源自 LDG，将节点列表重复进行流传输，直到达到最大迭代次数。具体来说，ReLDG 在每次迭代时都执行以下操作：对于每个 u∈V，将 u 赋给满足以下条件的分片

$$\arg\max_{i\in[k]} |V_i^{(t)} \cap N(u)| \cdot \left(1 - \frac{x_i^{(t)}}{C}\right).$$

毋庸置疑，流顺序中节点的位置对该节点周围生成的分区的质量起着很大的作用。流开始处的节点尚未受到乘法权重的影响，而流结束处的节点分配可能受该项支配。过去对 LDG 和 ReLDG 的研究主要集中在随机顺序上，在最初的 LDG 工作中考虑了 BFS/DFS 顺序。

该工作考虑到了以下几种算法特征:

Synchronous vs.streaming assignment. BLP 和 SHP 通过利用来自先前分区的静态快照的信息同时执行所有节点重新定位。而在 ReLDG 中，节点从节点列表上的串行传递，每次有一个节点参与分配，从而更改流中稍后节点的分配情况。这种区别产生了同步与流式分配两类算法。

Flow-based vs.pairwise constraint handling. 在两种同步算法之间，BLP 使用线性规划来保持平衡，在每个分片的节点净流入等于净流出的约束条件下，最大化重定位增益，直至达到所需的不平衡参数。另一方面，KL-SHP 只是确保相同数量的节点在分片对之间移动。由于前者具有流体动力学解释，因此称之为基于流的约束处理策略。后者称之为成对约束处理。

Incumbency preference. 在 BLP 、SHP-I 和 SHP-II 中，只有增益g_u>0 的节点才有资格重新定位。所有其他节点都被重新分配到其以前的分片分配。另一方面，KL-SHP 允许通过重新定位“次优”节点进行次优移动，以实现全局正交换。ReLDG 在每次迭代时串行地分配每个节点，可能会将流中后期在上一次迭代中分配给所需分片的节点逐出。BLP 和受限 SHP 算法因此具有在位偏好，始终允许节点保留其先前的分配。 为了参数化这种偏好，可以为算法的“在位”级别引入了一个阈值 c，定义为允许$g_u \le c$ 的节点保留其最后一次分配。换句话说，只有g_u>c 的节点才有资格重新定位。Vanilla restreaming（随机流顺序）对应于 c=-∞（无在位）的选择，而 BLP 对应于 c=0 的选择。任何算法都可以很容易地修改，以将 c 容纳为方法的输入参数，

Priority ordering. 在这项工作中，优先级被定义为如何考虑非在位节点进行重新定位的顺序。BLP 和 KL-SHP 在进行节点重新定位时都优先考虑增益。它们利用重定位队列中的排序，优先移动收益最高的节点，从而直接优化边划分。另一方面，Vanilla ReLDG 在重新定位节点时不会对任何度量进行优先级排序；节点按照流顺序随机进行排序。这一事实突出了这一系列算法的改进机会，该工作即基于此，提出了一种新的度量，即矛盾性，作为流顺序的依据，提出以矛盾性降序顺序排列流。也就是说，强烈倾向于移动或原地不动的节点被放置在流的前部中，从而使节点有很好的机会进入理想的分区，而更“矛盾”的节点则被延迟。将节点 u，a_u 的矛盾性定义为在对比当前分配与最佳可能外部分配时，共同分配邻居的（负）最大差异：

$$a_u = -\max_{i\in[k]\setminus P(u)} \left||N(u)\cap V_i| - |N(u)\cap V_{P(u)}|\right|.$$

a_u 越高(越接近 0)，节点当前分配和其他最佳分片之间的邻居同位置计数差距越小。由于矛盾值的范围从负到 0，这种排列有将度数较低的节点推向流末尾的趋势。可以观察到矛盾值和节点度数之间的高度相关性，同时，能够证明期望的初始矛盾是节点度的单调（线性）函数的上下界。

对于该改进对相关算法的性能提升，该工作设

计了相关实验来回答以下问题（1）所提出的平衡图划分算法在切割质量方面如何进行比较？（2）模块化在这些方法的性能中扮演什么角色？（3）优先 ReLDG 算法的性能如何随着 k 的增加而提升？（4）流顺序如何影响 ReLDG 的性能？

	Synchronous				Streaming (reLDG)						
Graph	SHP-I	SHP-II	KL-SHP	BLP	Random	CC	BFS	Degree	Ambivalence	Gain	METIS
pokec	0.578	0.595	0.585	0.532	0.675	0.681	0.698	**0.716**	0.712	0.618	0.827
livejournal	0.626	0.648	0.625	0.617	0.674	0.666	0.731	0.745	**0.749**	0.671	0.899
orkut	0.535	0.555	0.534	0.531	0.650	0.628	0.665	**0.689**	0.679	0.626	0.711
notredame	0.783	0.635	0.652	0.612	0.882	0.864	**0.929**	0.902	0.924	0.878	0.982
stanford	0.737	0.711	0.697	0.629	0.856	0.844	0.891	0.900	**0.916**	0.793	0.973
google	0.670	0.603	0.616	0.606	0.848	0.814	0.868	0.959	**0.964**	0.799	0.989
berkstan	0.701	0.652	0.658	0.585	0.858	0.805	0.895	0.913	**0.918**	0.766	0.988

图 1 基于矛盾值的改进对算法带来的性能提升

由数据可知具有随机流顺序的 ReLDG 算法在所有网络中的性能都比同步方法好很多，考虑到 ReLDG 通常被认为受到其为线上图特殊设计而被进一步限制，这是一个令人惊讶的结果。此外，矛盾顺序在七个图中的四个图上产生最佳划分（并且与所有七个图上的最佳结果相竞争）。这些算法的相对性能的一些细节值得在以后的工作中中进一步分析和评论。

通过分析同步方法的结果，直觉表明 BLP 将优于基于 SHP 的算法，因为基于流的约束处理扩展了允许的重新定位空间（相对于基于 SHP 的方法的成对处理）。然而，不仅 KL-SHP 在所有网络上都优于 BLP，而且 SHP-I 和 SHP-II（KL-SHP 的限制形式）提供了更高质量的分区，其中 SHP-II 在社交网络上表现最好，SHP-I 在网络上表现最好（在同步方法中）

这篇论文在工作中剖析了用于平衡分区划分的具有高度可伸缩性的迭代算法中涉及的设计决策。并在此基础上引入了一个新的类，即优先级流算法，它来源于同步算法中流顺序设置的优先级思想。本工作为流算法提供了一种新的优先级顺序，即矛盾值顺序。当在各种社交和网络图上进行测试时，该工作发现流算法不会受到基于 BLP 或 SHP 的算法采用的同步分配过程中观察到的病态现象（即分处两个分片中的相邻节点不断交换位置）。与 BLP 和 KL-SHP 相比，即使是普通 reLDG 算法（随机流顺序）结果也能在所有测试图上生成更高质量的分区。

最好的划分结果来自矛盾值和度数排序，在七个测试图中的六个图上处于领先。矛盾值和度数排序具有高度相关性，如果计算矛盾值很麻烦，则采用静态的度数排序也能取得相近的效果。虽然被设计主要面向线上图的划分，但实验结果表明，高度可扩展的线下图划分也可以选用 ReLDG 系的算法并保持主要竞争力。

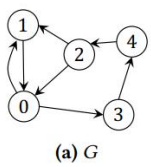
（

3.2 基于依赖的同步流式图处理技术

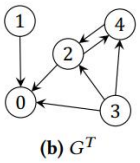
该论文主要面向图计算过程中的数据处理工作，提出了 GraphBolt 系统来处理动态图，同时通过依赖驱动的增量处理来保证 BSP 语义。GraphBolt 的编程模型允许分解复杂的聚合以合并增量值更改，同时还支持简单聚合的直接增量更新。研究评估表明，GraphBolt 可以有效地处理具有不同突变率的流式图，从单边突变到最多百万级别边突变的情况均能提供良好支持。

在本工作之前，已经存在一些工作探讨过对动态变化图的处理，例如 KickStarter、GraphIn 和 Tornad，这些高效流式图处理系统的核心是一个动态图，它的结构通过一系列图更新而迅速变化，其中包含一个增量算法，对图结构的变化作出反应，从而为最新的图快照生成最终结果。工作图不断地被由边和顶点的插入和删除组成的△G 更新流。算法通过对图的最新快照进行现场计算来产生最终结果。为了保持一致性，在迭代过程中执行计算时，将更新批处理到△G 中，并在开始下一次迭代之前将其合并。

虽然对动态图进行增量处理被认为是有效的，但直接重用中间结果通常会导致算法产生错误的结果。尽管某些单调算法（如最短路径和广度优先搜索）总是产生正确的结果，但那些需要同步处理语义的算法就不会收敛到正确的值。图 2 显示了一个从 G 变为G^T的流式图，以及相应的标签传播结果，这是在评估中使用的同步图算法。在没有增量处理的情况下，$S^*(G^T, \mathrm{I})$收敛到正确的结果；但是，从$S^*(\mathrm{G}, \mathrm{I})$进行增量计算违反了同步处理语义，结果收敛到了$S^*(G^T, R_G)$，这是不正确的。



(a) G



(b) G^T

	0	1	2	3	4
I	0.32	0.72	0.29	0.74	0.27
$S^*(G, I)$	0.32	0.326	0.329	0.317	0.327
$S^*(G^T, I)$	0.32	0.717	0.737	0.741	0.737
Incorrect incremental computation using values of G to compute for G^T					
$S^*(G^T, R_G)$	0.32	0.326	0.322	0.317	0.321

(c) Label Propagation on G & G^T. Incorrect values are highlighted in red.

图 2 动态图增量处理示例

为了增量地转换R_G^k，该工作研究并开发了一个依赖性驱动的增量处理模型，它以值依赖的形式捕获R_G^k的计算方式，然后使用捕获的信息来整合图结构中变化的影响。为此，首先需要按照同步处理语义的定义描述连续迭代中的值之间的依赖关系，然后随着迭代的进行跟踪这些依赖关系。当△G 到达时，按照迭代次序优化捕获的依赖项，以增量方式生成迭代 k 的$R_{G^T}^k$，然后直接用于以同步方式向前计算。通过执行上述过程，可以保证在每次迭代结束时同步处理语义，从而确保最终结果的正确性。

然而，开发这种依赖驱动的增量处理带来了一些挑战。首先，在线跟踪依赖关系可能非常消耗性能，因为依赖关系信息的数量与边 E 的数量成正比。该论文通过仔细地将依赖信息转换为位于顶点上的聚集值的形式，并进一步认识到依赖结构（即值如何相互影响）可以从输入图结构中导出，从而克服了这一挑战，从而将依赖项信息降到顶点 V 的数量级。此外，由于真实世界的图是稀疏的和倾斜的，这导致聚合值随着迭代的进行而稳定。因此，该工作提出修剪机制，保守地修剪要跟踪的依赖性信息，而不会导致额外的分析（例如反向传播）来重新计算未跟踪的值。

依赖驱动的增量处理对于复杂聚合（例如，在向量上运行的 MLDM 聚合）来说变得很困难，因为它们的增量对应项通常不容易推断。为解决这个问题，，该论文介绍了其开发的一个通用的增量编程模型，该模型允许分解复杂聚合，以纳入由△G 产生的增量值变化。编程模型支持表达分解复杂聚合的基本工作流程，并再现根据值变化更新的旧贡献，由此构建复杂聚合的增量。此外，像 UMGET 这样的简单聚合可以直接表达在该增量编程模型中，而无需进行工作流分解。

最后，论文阐述了一个计算感知的混合执行策略，当依赖信息因剪枝而不可用时，它可以在依赖驱动的增量处理和传统的增量处理之间动态切换。

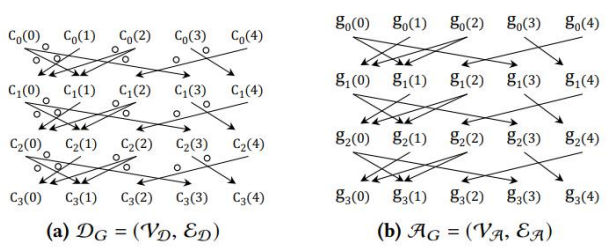


图 3 样例图的依赖关系迭代

图 3 显示了图 2 中 G 在执行 4 次迭代时的依赖关系图。随着迭代计算的进行，D_G增加了| V |顶点和| E |边。当保存D_G时，会详尽地捕获整个执行历史，以便能够对随后的图突变进行C_L的增量校正，这种对值的依赖关系的跟踪会导致对于总计 t 次迭代需要维护 O（|E|*t）数量的信息，这会显著增加内存占用，从而使整个过程内存受限。为了减少必须跟踪的依赖信息量，必须仔细分析值是如何通过依赖关系参与C_L的计算

Tracking as Value Aggregations. 给定一个顶点 v，它的值是基于其传入邻居的值，分两个子步骤计算的：首先，将来自上一次迭代的传入邻居的值聚合为单个值；然后，使用聚合值计算当前迭代的顶点值。该计算公式如下

$$c_i(v) = \oint_{\forall e=(u,v)\in E} \left(\bigoplus (c_{i-1}(u)) \right)$$

由于流经边的值最终组合成顶点处的聚合值，因此本工作提出可以跟踪这些聚合值，而不是单个依赖项信息。通过这样做，可以通过递增地校正聚合值并在整个图的后续聚合中传播校正来校正图变异时的值依赖性。

为了研究这种聚合方案带来的性能提升，该论文对几种使用不同的聚合函数的算法进行了性能测试

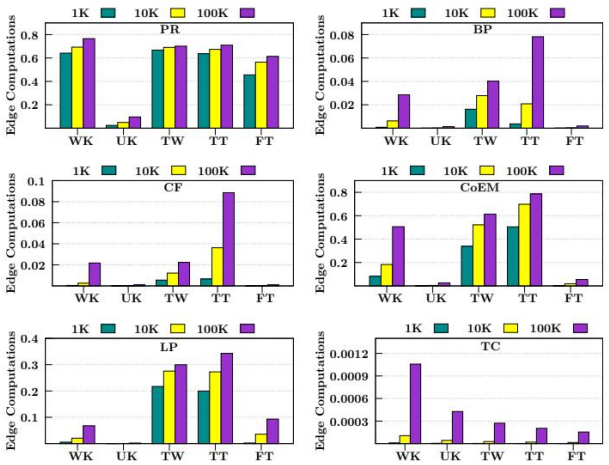


图 4 几种不同算法在不同图突变率下的性能表现

如图 4 所示，与其他图相比，GraphBolt 对 YH 执行的边缘计算要低得多（如图 6 所示）；除 CoEM 小于 12%（即 0.12 比率）外，所有情况下，它都小于 0.5%。这主要是因为与 GraphBolt 相比，GB Reset 需要做更多的工作，即与 GraphBolt 相比，GB Reset

更多地利用了并行性, 这表明 GraphBolt 的基于依赖驱动的增量计算足够强大，可以减少计算量，而无需依靠增加计算能力来实现高性能。注意，边突变的影响取决于图的结构和图算法的性质。例如，PR 上的 GraphBolt 实现了 UK 比 FT 更高的节省，而 BP 和 CF 实现了 UK 比 FT 更高的加速比

3.3 基于GPU的动态图处理加速(GPMA)

本工作主要研究如何在 gpu 上高效的动态更新图结构。首先，该工作介绍了一个 GPU 动态图分析框架，使得现有的面向 GPU 的静态图算法能够支持高性能的演化图分析。其次，为了避免在 gpu 上进行图结构的重建成为处理动态图的瓶颈，该论文提出 GPMA 和 GPMA+并行支持增量动态图维护, 从理论上证明了 GPMA+的可扩展性和复杂性，并通过大量实验评估了其效率。作为未来的工作，该论文提出希望能够探索一种用于动态图处理的混合 CPU-GPU 方法，并为相关应用程序进行更有效的优化。

为了实现 GPU 上的动态图计算优化, 本文提出了动态图存储应该遵循的相关设计原则 **Design Principles** 提出的 GPU 上的图存储方案需要考虑以下因素:

(1) 提出的动态图存储应该有效地支持多样化的更新操作，包括插入、删除和修改。此外，它应该具有良好的局部性，以适应 GPU 的高度并行内存访问特性，从而实现高内存效率。

(2) 物理存储策略应支持通用的逻辑存储格式。这样才能很好的适配现有的图分析方案

GPMA 主要启发自一种新型结构，即压缩内存阵列(Packed Memory Array)（PMA），该结构旨在通过留出间隙以适应具有有界间隙比的快速更新，同时以部分连续的方式保持排序元素。PMA 是一种自平衡二叉树结构。给定一个 N 个条目的数组，PMA 将整个内存空间划分为长度为 O(logN)的叶段，并将非叶段定义为其子段所占用的空间。对于位于高度 i（叶子节点高为 0）的任何分段，PMA 设计了一种方法，将分段的下限和上限密度阈值分别指定为ρ_i和τ_i，以实现 O(log^2N)的均摊更新复杂性。一旦插入/删除导致段的密度超出(ρ_i, τ_i)定义的范围，PMA 将尝试通过重新分配存储在段的父节点中的所有元素来调整密度。调整过程以递归方式调用，直到所有分段的密度再次满足 PMA 密度阈值定义的范围内时，调整过程才会终止。对于有序数组, 修改方式是显而易见的。因此，主要以向现有的有序结构进行插入操作为讨论重点，而删除操作在 PMA 中可以看做是插入的双重操作, 因此不再赘述。

图 5 显示了 PMA 插入的示例。每个段由相应树节点中显示的间隔（数组的起始和结束位置）唯一标识，例如，根段是段[0,31]，因为它覆盖所有 32 个空间。存储在 PMA 中的所有值都显示在数组中。图中的表格显示了预定义的参数，包括段大小、密度阈值(ρ_i, τ_i)的分配以及树不同高度处相应的最小和最大入口大小。在本工作中使用这些设置作为运行示例。在 PMA 中，首先通过二叉搜索识别相应的叶段，并将新条目放置在叶段的后面。插入导致叶段的密度（=4）超过阈值（t=3）。因此，需要确定最近的祖先片段，该片段可以在不违反阈值的情况下容纳插入，即段-[16,31]。最后，通过在段-[16,31]中均匀地重新分配所有条目来完成插入。

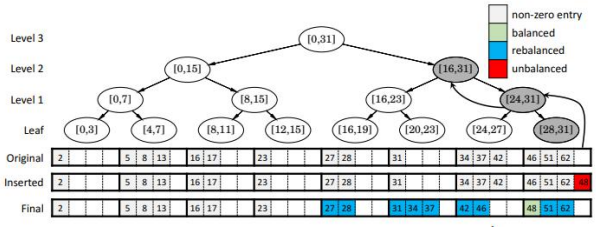


图 5 PMA 插入示例

Concurrent Insertions in GPMA. 受 CPU 上 PMA 的启发，该论文提出 GPMA 以在 GPU 上进行并发插入。直观地说，GPMA 为线程分配一个插入，并使用基于锁的方法为每个线程并发执行 PMA 算法，以确保一致性。更具体地说，插入的所有叶段都是预先确定的，然后每个线程检查插入的叶段是否仍然满足自下而上的阈值。对于每个特定段，都以互斥的方式访问。此外，更新位于相同树高的所有段后，所有线程都会同步，以避免可能的冲突，因为较低高度的段完全包含在较高级别的段中。

图 6 为 GPMA 并发插入提供伪代码, 强调了添加到原始 PMA 更新算法中的行，以实现 GPMA 的并发更新。对于第 9 行所示的每个迭代，所有线程都从叶段开始，并以自下而上的方式尝试插入。如果某个线程在第 11 行的互斥竞争中失败，它将立即中止并等待下一次尝试。否则，它将检查当前段的密度。如果当前段不满足密度要求，它将在下一

个循环迭代中尝试父段（第 13-14 行）。一旦某个祖先段能够容纳插入，它将合并第 16 行中的新条目，并将该条目从插入集中删除。随后，更新的段将均匀地重新分派其所有条目，并且进程终止

Algorithm 1 GPMA Concurrent Insertion

```
1: procedure GPMAInsert(Insertions I)
2:     while I is not empty do
3:         parallel for i in I
4:             Seg s ← BinarySearchLeafSegment(i)
5:             TryInsert(s, i, I)
6:         synchronize
7:         release locks on all segments

8: procedure TryInsert(Seg s, Insertion i, Insertions I)
9:     while s ≠ root do
10:        synchronize
11:        if fails to lock s then
12:            return                          ▷ insertion aborts
13:        if (|s| + 1)/capacity(s) ≥ τ then
14:            s ← parent segment of s
15:        else
16:            Merge(s, i)
17:            re-dispatch entries in s evenly
18:            remove i from I
19:            return                          ▷ insertion succeeds
20:    double the space of the root segment
```

图 6 GPMA 并发插入算法伪代码

图 7 展示了一个具有五个插入的示例，即{1,4,9,35,48}，用于并发 GPMA 插入。初始结构与示例 1 相同。识别要插入的叶段后，负责 Insertion-1 和 Insertion-4 的线程将竞争相同的叶段。假设 Insertion-1 成功获得互斥，Insertion-4 将中止。由于段有足够的可用空间，Insertion-1 成功插入。即使 Insertion-9、35、48 没有叶段竞争，他们也应该继续检查相应的父段，因为插入后没有任何左叶段满足密度要求。Insertion-35,48 仍然竞争同一级别的 1 段，Insertion-48 胜。对于本例，有三个插入成功，结果如图 4 底部所示。Insertion-4、35 在此迭代中被中止，并将等待下一次尝试。

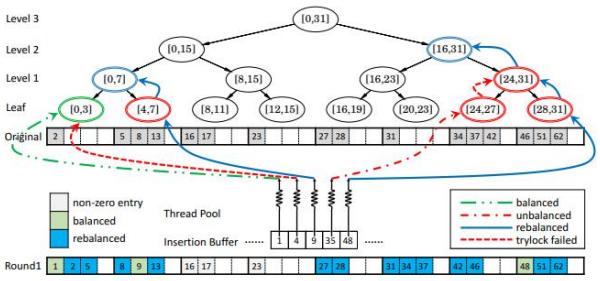


图 7 GPMA 并发插入示例

为了解 GMPA 系统相较现有系统的性能提升，图 8 展示了相关实验数据

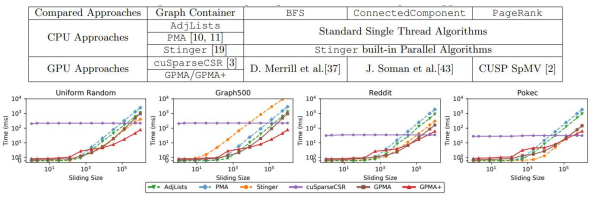


图 8 GPMA 相较其他 GPU 解决方案的性能表现

可以观察到，当边更新批的规模较小时，基于 PMA 的方法在处理更新时非常有效。当边更新批变大时，PMA 和 GPMA 的性能会迅速降低到简单重建的性能。尽管对于小更新量，GPMA 比 GPMA+具有更好的性能，并发更新条目不太可能发生冲突，但对于大批量，线程冲突会变得严重。由于其无锁特性，GPMA+显示出优于 PMA 和 GPMA 的性能。特别是，相对于 PMA 和 GPMA，GPMA+的加速比分别高达 20.42 倍和 18.30 倍。Stinger 在大多数情况下显示出令人印象深刻的更新性能，因为 Stinger 以并行方式高效地更新其动态图结构，并且代码运行在强大的多核 CPU 系统上。目前，对于纯随机数据结构维护，多核 CPU 系统被认为比 GPU 更强大，但成本更高。此外，还可以注意到，Stinger 在 Graph500 数据集中的性能非常差。根据现有的研究，这种现象是由于 Stinger 对每个边的块执行固定尺寸。由于 Graph500 是一个严重歪斜的图，因为图遵循幂律模型，这种歪斜会导致 Stinger 在内存利用方面的严重性能缺陷。

4 总结与展望

本文主要从图划分算法，图数据处理，图计算硬件加速三个角度总结了图划分领域的最新工作。算法层面，基于矛盾值的重组线性确定性贪婪算法被认为具有很高的潜力，该工作为现有的流算法提供了一种新的优先级度量方式，即矛盾值顺序，并通过实验验证这种新的基于流的算法不会受到基于 BLP 或 SHP 的算法采用的同步分配过程中观察到的病态现象，从而提升了现有流算法的性能表现与稳定性。图数据处理方面，针对动态线上图环境开发的 GraphBolt 系统采用了依赖驱动的增量处理策略，在保证 BSP 语义同时，采用增量聚合的方式降低了动态图跟踪过程中需要记录的依赖数量，从而实现了对变化突变速率下的动态图进行高效处理。硬件加速方面，基于 GPU 的 GPMA 系统创造

性的将 PMA 结构引入 GPU 数据处理体系，并以此为基础实现了数据的无锁化并行处理，使得 GPU 的动态图处理加速能力上升到一个新的台阶。

参 考 文 献

[1] Awadelkarim A, Ugander J. Prioritized Restreaming Algorithms for Balanced Graph Partitioning[C]//Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2020: 1877-1887.

[2] Mariappan M, Vora K. Graphbolt: Dependency-driven synchronous processing of streaming graphs[C]//Proceedings of the Fourteenth EuroSys Conference 2019. 2019: 1-16.

[3] Sha M, Li Y, He B, et al. Technical report: Accelerating dynamic graph analytics on gpus[J]. arXiv preprint arXiv:1709.05061, 2017.