超越 BFS：图形处理器上的有根生成树算法比较研究

Abhijeet Sahu、Srikar Vilas Donur 蒂鲁帕蒂
印度理工学院，印度，517619{cs22s501, cs23b049}
@iittp.ac.in

arXiv:2603.11645v1 [cs.DC] 12 Mar 2026

摘要—**有根生成树（RST）是并行图分析中的核心原语，为双连通分量、平面性测试等算法提供基础。在图形处理器上，RST 构建传统上依赖广度优先搜索（BFS），因为它简单且工作效率高。然而，BFS 的步复杂度为 $O(D)$，这严重限制了其在高直径图和幂律图上的并行度。**

我们对现代 GPU 上的替代 RST 构建策略进行了比较研究。我们提出了路径反转有根生成树（PR-RST）算法在 GPU 上的实现，并针对现代图形处理器架构优化了其指针跳跃和广播操作。此外，我们还评估了一种集成方法，它将先进的连通性框架（GConn）与基于欧拉遍历的根化相结合。

在超过 10 个真实世界图上，我们的结果表明，基于 GConn 的方法在高直径图上的加速比最高可达 $300\times$，相较于优化的广度优先搜索更快。这些发现表明，基于连通性的方法的 $O(\log n)$ 步复杂度可能抵消其结构开销在现代硬件上的影响，从而促使人们重新思考 GPU 图分析中的 RST 构建。

索引词—**生成树、有根生成树、图形处理器、广度优先搜索、连通分量**

I. 引言

有根生成树（RST）是并行图分析中的基础构件，为众多算法提供骨干，包括最短路径、耳分解、平面性测试和双连通分量 [1]。这些操作并非仅具有理论意义；它们对于揭示大规模真实世界网络的底层拓扑结构至关重要。

RST 构建的实际应用十分广泛，且跨越多个学科。在计算生物学中，有根生成树用于分析蛋白质相互作用网络并重建系统发育谱系。在电力系统工程中，它们有助于分析电网韧性和应急规划。此外，RST 算法还成功应用于超大规模集成电路设计中的时钟树综合、机器人学中构型空间的路径规划、网络遥测中的无环路由，以及社交网络分析中的有影响力的社群识别。

尽管 RST 构建用途广泛，但其在现代硬件上的效率很大程度上取决于图结构。形式化地说，给定一个无向连通图 $G = (V,\ E)$ 和指定的根节点 r，根生成树（RST）可表示为父数组 P，其中 $P[r] = r$，且每个其他顶点恰有一个父节点。广度优先搜索（BFS）长期以来一直是生成 RST 的主流方法，而且理由充分。其概念简单、易于实现，因此成为图形处理器架构上的默认选择。广度优先搜索保证能找到从根节点到所有其他顶点的最短路径（按跳数计算）。

然而，尽管广度优先搜索无处不在，它对图深度仍存在固有依赖。广度优先搜索通过层同步遍历推进：必须先发现所有与根节点相距 k 的顶点，才能处理任何相距 $k+1$ 的顶点。尽管这种结构在算法上简洁清晰，但它引入了与大规模并行硬件根本相悖的顺序依赖。在直径较小的图上，这种同步开销通常可以忽略不计。然而，在直径较大的稀疏真实世界图上，这可能转化为数千次串行的内核启动和全局同步，从而显著降低性能。设计谱系的另一端，是为连通分量（连通分量） 问题开发的算法，它们可以有效地改造以用于此目的。文献中常被忽视的一个关键观察是：连通性本身并非难题，根化才是。虽然广度优先搜索同时解决连通性与根化问题，但这两项任务可以解耦。

在其开创性工作中，**Shiloach 和 Vishkin（1982年）** [7] 证明，只需极少的额外簿记，任何连通性算法都能在分量合并（union）阶段同时识别生成树边。然而，该方法仅产生无根森林。当**Tarjan 和 Vishkin（1984年）** [6]提出欧拉遍历 技术时，这一缺口得到了弥补；该技术提供了一种优雅的方法，可在对数深度内为无根生成树指定根节点。连通分量算法与欧拉游历技术共同构成了设计深度无关 且工作高效 的有根生成树算法的强大基础，其并行复杂度为 $O(\log n)$。

然而，尽管理论上合理，**Cong 和 Bader（2004年）** [1], 指出，由于定根过程采用了欧拉游历技术，因此还存在另一项实际限制。这种方法需要将标准邻接表转换为循环邻接表，并通过额外的指针定义一条欧拉回路，使其恰好遍历每条有向边一次。构建和维护这种结构会带来额外的预处理开销和不规则内存访问模式，而这两者都会影响并行架构上的性能。为克服这些挑战，**Cong 等人（2004年）** [1], 基于 Shiloach–Vishkin 连通性框架，提出了一种根本性的新

算法，称为 PR-RST，用于构建有根生成树（RST），并消除了将连通性与定根阶段分开处理的必要。该路径反转有根生成树算法不再先构建一片未定根的生成森林，再通过欧拉巡回为其定根，而是将这两项任务作为一个统一问题来处理。

下表总结了 RST 构建主要方法的理论复杂度。

表 I 有根生成树算法的深度与工作量比较。

算法	并行深度	工作复杂度
广度优先搜索 [4]	Θ(直径(G))	$O(V+E)$
连通分量 + 欧拉遍历	$O(\log V)$	$O((V+E)$ 对数 $V)$
PR-RST [1]	$O(\log V)$	$O((V+E)$ 对数 $V)$

A. 我们的贡献

本文作出以下贡献：

- 我们提出了 Cong 和 Bader 的 PR-RST 算法的图形处理器实现，该实现根据其原始多核实现进行了调整，并评估了其在现实图形处理器执行约束下的表现。
- 我们表明，RST 构建可以自然地分解为一个由两个阶段组成的过程：首先进行连通分量计算，随后使用欧拉遍历执行定根阶段；这两个阶段都能高效地映射到图形处理器架构上。
- 通过对 30 多个真实世界图数据集开展广泛的实验研究，我们确定了最适合 GPU 的 RST 构建策略。

II. 相关工作

Harish 和 Narayanan（2007 年） [2]首次使用 CUDA 在图形处理器上实现了广度优先搜索。Merrill 等人（2012 年） [4], 随后通过采用细粒度前缀和进行负载均衡，并在边级别而非顶点级别扩展前沿，实现了改进，使其达到了渐近最优的O（$V + E$）工作复杂度。后续的 GPU 图处理框架和库（例如 Gunrock 和 Groute）进一步探索并优化了广度优先搜索。这些工作巩固了广度优先搜索作为 GPU 上最实用遍历原语的地位，同时也暴露了其架构局限性。

尽管取得了这些进展，所有基于遍历的方法（基于广度优先搜索的 RST）从根本上仍然容易受到图直径（D）的影响。在“稀疏”图中，该算法被迫执行 D 次串行迭代。

基于遍历的 RST 构建的另一种方案依赖于并行连通性算法。Soman 等人首次展示了在 GPU 上大规模采用连通性算法的方法，证明 Shiloach–Vishkin 风格的指针跳跃方法能够自然地映射到 GPU 架构上。此后，基于 GPU 的连通分量研究取得了长足进展，涵盖标签传播、并查集变体，以及在工作效率和收敛速度之间进行权衡的混合方法。其中，GConn [3]代表了当前的最先进水平。

III. 方法论

本节将介绍本文研究的算法及其并行 GPU 实现。

A. 广度优先搜索（*BFS*）

给定一个源顶点，广度优先搜索会按距离递增的顺序探索图，生成层次结构和一棵隐式生成树。从概念上讲，广度优先搜索按轮次（或层次）进行。在每一轮中，上一轮发现的所有顶点（前沿）都会尝试访问尚未发现的邻居。新发现的顶点构成下一轮前沿。这个过程持续进行，直到所有尚未发现但可达的顶点都被访问。我们采用 Merrill 等人的以边为中心的广度优先搜索实现 [4]作为基线。

B. 基于连通性的算法

其核心思想基于两种交替执行的操作：连接 （也称为钩接）和 压缩（也称为捷径化或指针跳跃）。

a) 钩接（连接）。：给定一条边 (u, v)，如果 u 和 v 当前属于不同的连通分量，则将一个连通分量连接到另一个连通分量，方法是令其中一个连通分量的根节点指向另一个的根节点。此操作称为钩接。当$hook$ 成功时，相应的边 (u, v)会被标记为生成树边，因为它有助于构建生成森林。

钩接可以通过多种方式进行（例如，基于顶点编号、随机优先级或秩），但目标始终相同：在避免形成环的同时减少连通分量的数量。

b) 指针跳跃：一轮钩接后，形成的结构是一片森林。然而，这些树可能深度较大且形状不规则，形成长链或类似星形的结构。这样的深度会对后续轮次的性能产生负面影响，因为遍历较长的父节点链代价高昂。为解决这一问题，连通性算法采用指针跳跃：每个顶点 i 都会将其父指针更新为 ($P[i] = P[P[i]]$)，从而在每次迭代中有效地将每个顶点到其根节点的距离减半。

经过 $O(\log D)$ 次指针跳跃后，其中 D 是树的直径，同一连通分量中的所有顶点都会直接指向同一个根节点。这种扁平化 会大幅降低树的深度，并为下一阶段的钩接做好准备。钩接与压缩交替进行，使算法能够快速收敛，并自然地映射到并行架构上。

C. 路径反转根生成树（*PR-RST*）

我们所知，**路径反转 RST（PR-RST）**算法最初由 **Cong 和 Bader** [1],提出，作为一种多核方法，此前尚未在 GPU 上进行探索。在这项工作中，我们针对现代 GPU 改造了 PR-RST，并引入了一系列 GPU 专用优化。

作为一种基于连通性的算法，PR-RST 遵循前文描述的相同钩接和指针跳跃框架。其核心思想是，在处理所有根节点–顶点对时，通过连通分量合并高效地更新树

$((u_i,\ r_i) \in R)$ 并行处理。对于每一对，算法会同时识别树上从 u_i 到 r_i 路径上的所有顶点。一旦确定这些路径上的顶点，就会并行反转它们之间的父子关系：对于每条满足 (u= parent[v]) 的边，通过设置 (parent[u] =v) 来翻转方向。

为支持高效的路径识别，每个顶点 (u) 都维护一个大小为 $O(\log n)$ 的数组，称为其特殊祖先数组。该数组存储 (u) 在距离为 2 的幂次处的祖先，并通过指针跳跃构建。利用这些特殊祖先，可在 $O(\log n)$ 次并行迭代中识别从 u_i 到 r_i 路径上的所有顶点，从而实现可扩展且适用于 GPU 的路径反转。

接下来，我们将讨论为提升这些操作在 GPU 上的效率而采用的优化。

钩接。 在钩接过程中，我们交替进行最大 钩接和最小钩接；经验表明，这种优化能改善收敛性和负载均衡。对于每条边 (u, v)，选择一个端点的代表元 $\max(\text{rep}_u, \text{rep}_v)$ 或 $\min(\text{rep}_u, \text{rep}_v)$ 作为钩接目标。

路径反转。 我们不显式存储父节点链，而是在捷径化过程中使用一个 'onPath' 数组记录指针跳跃历史。在随后的迭代中，利用这一历史，通过在 ($\lceil \log n \rceil$) 轮中传播标记来重建反向路径。随后，第二个内核沿着标记路径反转父指针，将树的方向从旧根节点重新定向至嫁接节点。这种设计避免了串行遍历，保持了合并内存访问，并确保路径反转完全采用数据并行方式。

指针跳跃。 每个钩接阶段之后，我们执行指针跳跃来压缩树。最终指针会收敛到连通分量根节点，并作为下一次迭代的代表元。记录快捷路径历史对于实现高效的路径反转至关重要，且无需额外同步。为减少内核启动和同步开销，我们在进行全局同步之前，让每个线程执行五个指针跳跃步骤。实验证明，与每次单独跳跃后都进行同步相比，这种策略性能更优。

D. 欧拉遍历

虽然 Polak 等人 [5] 此前已将欧拉遍历算法改编用于 GPU，但其实现依赖旧版 mgpu 库，且仅限于连通树。我们通过使用现代**CUB (CUDA Unbound)**库，对该实现进行了仔细的重新工程化，并将算法推广到能够处理非连通森林。

给定一个包含 N 个顶点和 N-1 条无向边的森林，我们生成 2(N-1) 条有向边，并按字典序对其排序。这会将每个顶点的所有出边连续分组，从而确定性地生成邻接顺序。根据排序后的排列，我们使用三个数组隐式构造邻接表：'first[v]' 和 'last[v]' 表示顶点 (v) 的出边范围，而 'next[e]' 则连接具有相同源顶点的连续边。

表 II 图统计数据。直径通过广度优先搜索生成树的深度进行近似；M 表示百万。

数据集	顶点数	边数	直径	Note
WB	0.69百万	13.30百万	973	web-BerkStan
AS	1.70百万	22.19百万	757	as-Skitter
HT	0.46百万	25.02百万	157	higgs-twitter
CD	0.54百万	30.49百万	14	coPapersDBLP
SO	2.60百万	56.41百万	23,581	sx-Stack Overflow
RU	23.95百万	57.71百万	6,143	美国道路
LJ	4.85百万	85.71百万	1,877	社交网络 LiveJournal 1
K20	1.05百万	89.75百万	253,378	kron_g500-logn20
EU	50.91百万	108.11百万	19,932	欧洲开放街道地图
K21	2.10百万	183.19百万	553,161	kron_g500-logn21
CO	3.07百万	234.37百万	6	com-Orkut
UK	18.52百万	523.65百万	38,360	uk-2002

对于每条有向边 e，我们将其欧拉遍历后继计算为

$$\text{succ}(e) = \begin{cases} \text{next}(\text{rev}(e)), & \text{if next}(\text{rev}(e)) \neq -1, \\ \text{first}(\text{from}(\text{rev}(e))), & \text{otherwise.} \end{cases}$$

直观地说，遍历完（$e = (u \to v)$）后，遍历会从（v）继续，在 v 的邻接表中越过反向边（rev（e）$= (v \to u)$）前进，必要时绕回表头。此后继函数会为每棵树生成一个循环链表。

为了推广到非连通森林，我们分别断开每个环。对于每个根节点（r），我们找出其最后一条出边 'last[r]'，计算对应的反向边（（last$[r] + E/2$）mod E），并将其后继设为（-1）。这样，每个欧拉回路都会转换为一个线性链表，从而为每棵树生成一次独立遍历。

然后，我们应用并行链表排序，为每条边分配其在对应欧拉遍历中的位置（排名）。由于这些环已被断开为独立链表，因此每棵树都有各自独立的排名。

最后，根据边秩推导父数组。对于每条边（$e = (u \to v)$）及其反向边（rev（e）$= (v \to u)$），如果（rank$[e]$ >rank[rev（e）]），则 'e' 对应返回遍历，且 'u' 是父节点[v]；否则，v 是父节点[u]。

IV. 实验

我们使用多种不同类型的图，在 NVIDIA L40s GPU 上评估了所有算法。所有代码均使用 NVCC v12.9（-O3，--sm_89）编译。每个数据集均先进行一次预热运行，再进行五次计时运行，并报告运行时间的中位数。我们首先对三种有根生成树算法进行总体性能比较。接下来，我们分析性能随图直径变化的情况，重点展示层级同步方法与基于连通性的方法之间截然不同的行为。最后，我们考察所得生成树的结构特性，并评估其对下游算法的影响。

A. 有根生成树算法的性能比较

我们的结果呈现出一个令人惊讶的结论：尽管欧拉游历变换存在已知的理论开销，GCONN 仍脱颖而出，成为速度最快的算法。平均而言，

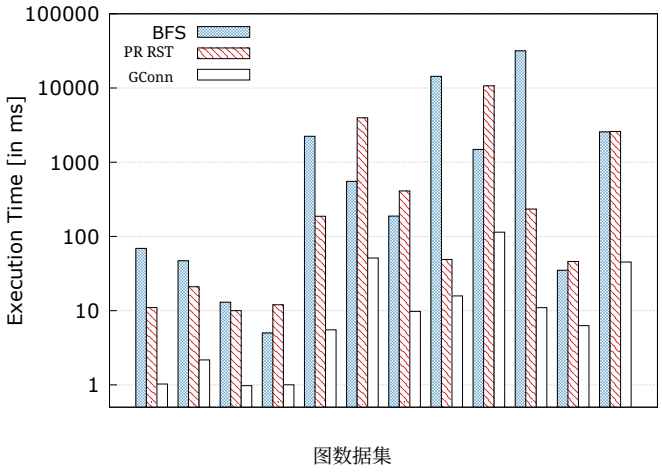


图1. 广度优先搜索、PR-RST 和 GCONN 的运行时间比较。

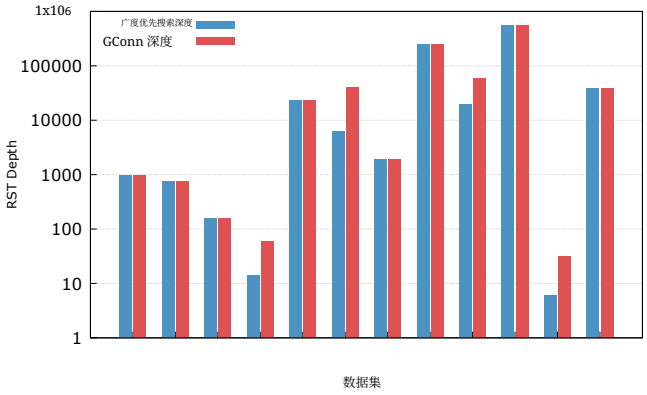


图2. 广度优先搜索 RST 深度与 GConn RST 深度的比较。

GCONN 相较于 BFS 和 PR-RST 实现了 **300× 加速**，如图 1 所示，甚至优于那些专门设计用来避免这些结构性开销的方法。

这一发现有力地表明，人们认为欧拉回路变换开销较大的看法，是旧式并行模型遗留下来的。对于现代图形处理器架构而言，这一“瓶颈”已不再是显著的性能因素。这一见解与 Polak 等人近期的研究完全吻合：他们在《欧拉遇上 GPU》中展示了这些经典的、以理论为基础的技术如何在 GPU 上重获新生，并发挥出显著的实际效用。经典理论的优雅之处曾被视为不切实际，如今则因现代硬件而得以释放。这一发现挑战了长期以来的既有假设，也预示着 GPU 上并行生成树构造的范式转变。

B. 图直径的影响

如图1所示，广度优先搜索的运行时间会随着生成的生成树深度增加而增长。如前所述，广度优先搜索采用层同步遍历，严格按照顺序逐层探索图。虽然这种方法在概念上很简单，但在 GPU 等大规模并行架构上会成为性能瓶颈。

每一层都必须处理完毕后，才能开始处理下一层，这使各层之间存在固有的顺序依赖。

对于直径较大的图，例如具有 1000 多个层级的道路网络，这种行为会导致数千次顺序内核启动。因此，当图直径较大时，广度优先搜索按层同步的特性从根本上限制了其利用现代 GPU 上海量并行能力的能力。

相比之下，**Gconn + 欧拉遍历**的性能在不同图直径下基本保持稳定。作为一种基于连通性的算法，Gconn 避免了严格的逐层同步，而是依赖并行挂接和压缩操作。这使其即使面对深层图也能展现出充分的并行性，因此对图直径的增加基本不敏感。然而，如图2所示，这种方式的代价是生成更深的生成树，凸显了基于遍历的方法与基于连通性的方法之间在深度和性能方面的权衡。

V. 讨论

尽管广度优先搜索仍然是构建根生成树的主流范式，但我们的结果表明，有必要转变看法。在多个数据集中，我们观察到，基于连通性的算法通常会生成明显更深的树。尽管这种结构差异给传统上围绕广度优先搜索树设计的下游任务带来了新的算法问题，但它也凸显了重新思考图算法的必要性，使其能够更有效地利用任意生成树。

VI. 结论

尽管基于图形处理器的连通性算法已经取得了显著进展，广度优先搜索远未过时；其在最短路径发现或环检测方面固有的优势仍然至关重要。然而，对于只需任意生成树的应用而言，传统遍历与结合欧拉遍历的现代基于连通性的方法之间的性能差距过于显著，不容忽视。对于动态图或增量图，PR-RST 等专用方法或许能为维护根树提供一条有前景的路径，同时避免完全重构的开销。我们将这些发现作为未来高性能计算图框架的指导原则：算法的选择应由问题决定，而不应受传统影响。

参考文献

[1] G. Cong 和 D. A. Bader，“欧拉游历技术与并行根生成树”，载于国际并行处理会议，*2004*年。*ICPP 2004*。IEEE，2004年，第 448–457 页。[2] P. Harish 和 P. J. Narayanan，“使用 CUDA 加速 GPU 上的大型图算法”，载于国际高性能计算会议（*HiPC*）。Springer，2007年，第 197–208 页。[3] C. Hong、L. Dhulipala 和 J. Shun，“探索 GPU 上静态与增量图连通性算法的设计空间”，载于*ACM* 国际并行架构与编译技术会议论文集，2020年，第 55–69 页。[4] D. Merrill、M. Garland 和 A. Grimshaw，“可扩展的 GPU 图遍历”，*ACM Sigplan Notices*，第 47 卷，第 8 期，第 117–128 页，2012年。

[5] A. Polak、A. Siwiec 和 M. Stobierski，“欧拉遇上 GPU：具有理论保证的实用图算法”，载于*2021*年*IEEE*国际并行与分布式处理研讨会（*IPDPS*）。IEEE，2021年，第 233–244 页。[6] R. E. Tarjan 和 U. Vishkin，“一种高效的并行双连通性算法”，《*SIAM*计算杂志》，第 14 卷，第 4 期，第 862–874 页，1985年。[7] Y. S. U. Vishkin 和 Y. Shiloach，“一种 O(log n) 并行连通性算法”，《算法杂志》，第 3 卷，第 57–67 页，1982年。