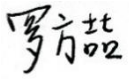
南京邮电大学
毕 业 设 计（论 文）

题　　目	基于光线追踪引擎的树遍历加速器设计与实现
专　　业	计算机科学与技术
学生姓名	罗方喆
班级学号	B21030116（B210301）
指导教师	尹捷明
指导单位	计算机学院、软件学院、网络空间安全学院

日期：　2024 年 12 月 30 日至　2025 年 5 月 30 日

毕业设计（论文）原创性声明

本人郑重声明：所提交的毕业设计（论文），是本人在导师指导下，独立进行研究工作所取得的成果。除文中已注明引用的内容外，本毕业设计（论文）不包含任何其他个人或集体已经发表或撰写过的作品成果。对本研究做出过重要贡献的个人和集体，均已在文中以明确方式标明并表示了谢意。

论文作者签名： 

日期：2025 年 5 月 28 日

摘要

树遍历的计算在许多场景（如物理模拟、数据库索引等领域）中都有着广泛的应用，但囿于其遍历过程中分歧点密集，容易存在不规则发散的特性，其于 GPU 上的计算始终存在明显的效率瓶颈。在 GPU 设计的 RTA（Ray Tracing Accelerator）单元中，通过对相关固定函数单元的专有化加速，大大减少了光线追踪中的 BVH 树遍历的发散问题。但因其固定的设计，导致无法对其他树遍历操作提供加速支持。

本文所研究的目标在于，通过自定义相关函数单元的设计，提高 RTA 对树遍历操作的泛用性，进一步提高 GPU 上树遍历操作的效率。本文提出了基于复用并扩展 RTA 中的 Ray-Box 交集单元，来为 B 树和半径搜索等算法提供加速支持的 TTA（Tree Traversal Accelerator）架构，成功扩展了 RTA 的应用场景，让 B 树搜索获得了高达 5.7 倍的速度提升。

但 TTA 继承了固有函数单元的设计，导致其仍然无法完全支持像物理模拟（如 N-body 仿真）实验所依赖的树结构的计算。为此本文进一步扩展了 RTA 中自定义计算单元的设计，称为 TTA+。其特殊的计算单元能够支持更具泛用性的树遍历计算（即使这需要牺牲一些效率），以此能够为 N-body 物理模拟提供达到 1.7 倍的树遍历加速效果。

关键词： GPU 加速器，树遍历，光线追踪

Abstract

Tree traversal is widely used in many scenarios (e.g., physics simulation, database indexing, etc.), but due to the dense divergence and irregular dispersion in the traversal process, there is always an obvious efficiency bottleneck in its computation on GPU. In the RTA (Ray Tracing Accelerator) unit designed for GPUs, the dispersion problem of BVH tree traversal in ray tracing is greatly reduced by the proprietary acceleration of the related fixed function unit. However, due to its fixed design, it cannot provide acceleration support for other tree traversal operations.

The goal of this paper is to improve the generalizability of RTA for tree traversal operations by customizing the design of the relevant function units to further improve the efficiency of tree traversal operations on GPUs. We propose a TTA (Tree Traversal Accelerator) architecture based on reusing and extending the Ray-Box intersection unit in RTA to provide acceleration support for B-tree and radius search algorithms, which successfully extends the application scenarios of RTA, and achieves up to 5.7 times speedup for B-tree search.

However, the inherited intrinsic function unit design of TTA still cannot fully support the computation of the tree structure, which is relied on for physical simulation (e.g., N-body simulation) experiments. For this reason we have further extended the design of the customized computational unit in RTA, called TTA+. The special computational unit is able to support more generalized tree traversal computations (even at the expense of some efficiency), thus providing a 1.7 times efficiency improvement in accelerating the tree traversal for N-body simulations.

Keywords: GPU Accelerator; Tree Traversal; Ray Tracing

目录

第一章 绪论

1.1 研究背景与意义

1.1.1 树结构的应用场景概述

树是计算机科学领域的常用数据结构。它在数据库系统（如 B 树、B+ 树）、计算机物理（如 N 体模拟中的八叉树、k-d 树）等大规模数据处理场景中广泛应用。由于树结构具备良好的层次化组织特性，在这些领域中能有效降低算法时间复杂度，提升运行效率。

在静态场景中，k-d 树被视为最有效的光线追踪加速结构。因此，存在多种在 GPU 上实现 k-d 树的尝试。例如，Ernst 等人提出使用并行堆栈在 GPU 上实现 k-d 树遍历，但频繁的内核切换带来了高开销和显著的带宽需求。尽管 GPU 计算能力强，早期 GPU 实现仍无法超越 CPU，在复杂场景中光线追踪性能表现有限。

为优化光追效率，诸多研究引入短堆栈优化策略或采用无堆栈遍历（stackless traversal）结构，例如 Thrane 和 Simonsen 提出的无堆栈 BVH 遍历架构，虽优于传统网格，但仍受限于带宽瓶颈。

尽管树结构在算法上具备优势，但在 GPU 上的实际运行表现仍存在瓶颈，主要原因是：树遍历的控制流发散严重，不规则的内存访问模式与 GPU “规整化并行处理” 理念相冲突，易造成线程分歧，影响整体执行效率。

为缓解这些问题，现代 GPU 引入了固定功能硬件单元——光线追踪加速器（Ray Tracing Accelerator, RTA），结合内存调度器优化光线追踪场景中的 BVH 遍历。但由于 RTA 的功能单元为固定设计，难以迁移至非图形任务（如数据库索引、N 体模拟）中的树遍历加速。

1.1.2 GPU 与光线追踪加速

光线追踪的核心操作是光线遍历与相交测试，其性能依赖于高效的数据结构加速支持。目前主流图形架构均采用包围体层次结构（Bounding Volume Hierarchy, BVH）作为加速基础。BVH 具备良好的内存利用率、可拓展性和兼容性。

图形厂商在近代 GPU 架构中逐步集成专用 RT 硬件模块。例如：

- NVIDIA Turing / Ampere 架构中使用 RT Core；- AMD RDNA2 架构引入 Ray Accelerator；- Intel Xe HPG 架构采用硬件堆栈支持。

这些硬件支持在图形渲染中成效显著，但难以拓展至如数据库查询、大规模图计算、物理模拟等对树结构遍历依赖显著的任务。

本文提出的问题是：

如何将光追加速器（RTA）所具备的遍历加速能力，推广至非图形类的树结构计算任务中？

1.2 研究内容与创新点

为应对 GPU 在通用树结构遍历中的局限，本文基于已有 RTIndex、RTNN 等工作的启示，提出一种具备更高灵活性与通用性的树遍历加速架构设计。

主要工作与创新点如下：

- **提出 TTA 架构**：复用光追中的 Ray-Box 单元，扩展支持 B 树、距离搜索等任务，适用于索引类与几何类判断逻辑。
- **设计 TTA+ 可编程版本**：模块化 μop 操作流程，支持自定义控制逻辑与复杂判断路径（如 Ray-Sphere 相交、N-body 近似力推理）。
- **微架构级优化**：在硬件资源开销极小（面积增 0.7%、功耗增 1.8%）前提下，实现最大 5.7× 的遍历性能提升。
- **接口 API 设计**：实现统一的编程接口，支持在 Vulkan-Sim 平台下对树结构应用的高效配置与模拟。

1.3 论文结构

本论文共六章，结构安排如下：

- 第一章：绪论，介绍背景、研究问题与本文贡献。
- 第二章：树遍历与光线追踪加速器基础，剖析 GPU 中 RTA 的结构与限制。
- 第三章：TTA 与 TTA+ 架构设计，提出本研究的加速器架构。
- 第四章：编程接口与配置方法，介绍编程流程、调试接口与 μop 调度机制。
- 第五章：实验结果与分析，验证本架构在多个实际任务中的加速效果。
- 第六章：总结与展望，归纳研究成果与提出未来研究方向。

第二章 树遍历与光线追踪加速器基础

2.1 树遍历算法与挑战

树遍历算法的高效性离不开树结构本身。树结构是一种特殊的图结构，其结构清晰、层次分明，能够有效地组织大量的数据内容。其在不同的应用场景中往往承担着样式各异但都极为重要的角色。例如，在数据库中，树结构常常承担索引、插入、查找等操作的运算，如 B 树和 B+ 树；而在物理仿真中，八叉树、四叉树和 k-d 树则用于划分空间，能够减少粒子交互次数，将 $O(n^2)$ 级的计算复杂度降低为 $O(n \log n)$；在计算机图形学中，BVH 结构树也广泛用于加速光线与物体之间的相交测试。

尽管不同场景下树结构的节点内容和判断标准不同，但其运算本质都是“从根到叶”的层次访问，即：从根节点出发 → 判断条件 → 访问子节点。例如数据库中的 Query-Key 比较操作，每个节点包含多个有序键值，树遍历过程中通过多次比较，决定访问哪一分支，直到抵达叶节点。

在物理模拟中，Barnes-Hut 算法通过构建八叉树结构对 N-body 系统进行空间划分。通过引入距离判断，如若距离超过某阈值，则将该子树视为一个“等效质点”处理，从而减少粒子交互次数。

类似地，在光线追踪中，Ray-Box 交集测试也是树遍历操作。通过从 BVH 根节点开始向下判断是否与包围盒相交，相交则继续访问子节点，直至几何体交集测试；不相交则剪枝跳过。虽然 Ray-Box 测试结构明确，但由于不同光线路径不一，容易造成并行线程执行路径严重分歧，影响 GPU 效率。

因此，不论是 N-body 模拟中的距离判断，还是光线追踪中的包围盒测试，本质上都依赖树结构逐层判定。这类遍历对控制流依赖度高，路径分歧多，正是 GPU 执行效率瓶颈所在。

2.2 光线追踪加速器（RTA）原理

光线追踪作为现代 GPU 图形渲染的重要组成，为了加速 Ray 与物体之间的交互判断，通常将场景表示为一个 BVH 树结构。

在传统 CPU 实现中，树结构遍历依赖栈与递归，通过 ‘while‘循环判断推进节点。而现代 GPU 中将这一逻辑封装为 ‘traceRay‘指令，并由硬件状态机自动调度执行，如图 2.1 所示。

RTA 模块嵌入在 SM（Streaming Multiprocessor）中，负责从 Ray Buffer 中读取每个线程的 Ray 状态，并由 warp 调度器追踪遍历进程。每个节点通常为 AABB（轴对齐包围盒），叶节点包含几何图元（如三角形）。相比传统通用计算核心的深度优先遍历，RTA 专用模块能更

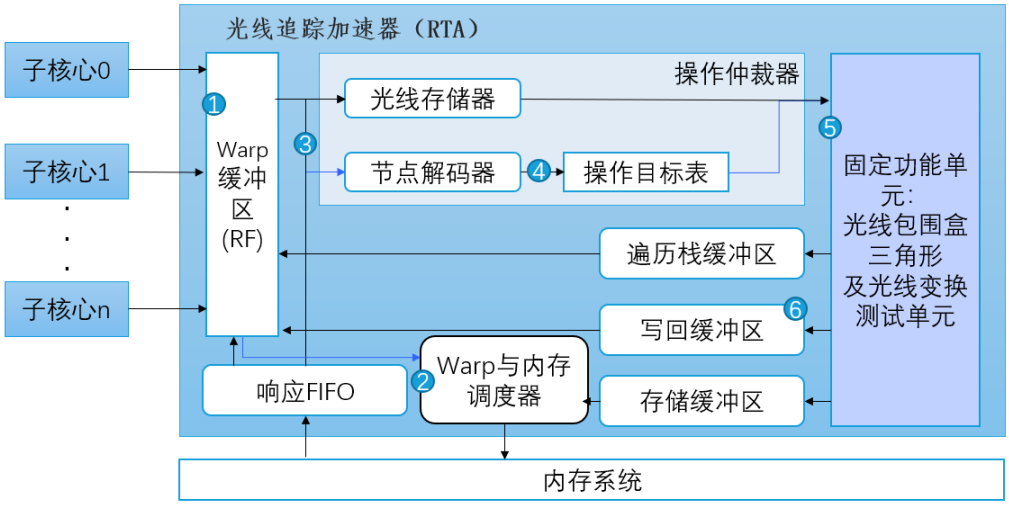


图 2.1　基于 RTA 的 GPU 基本架构示意图

快地完成光线遍历与交集测试操作。

在渲染完整一帧图像时，除了遍历操作外，还有像素级的着色器执行。这些仍由 GPU 通用核完成，而遍历卸载到 RTA，实现任务分摊与性能提升。

2.3 RTA 性能上的优势

在理解了 RTA 的基本结构与执行流程后，本文将进一步对其在 GPU 体系中实现 BVH 的高效遍历与相交测试的内在流程进行分析。相较于在传统 GPU 上，基于通用计算核心执行的光线遍历与相交测试流程，RTA 在架构的设计上采用了多项极为关键的优化机制。从计算模式、控制流管理、内存访问调度、到资源利用率等各方面，RTA 系统性地解决了光线追踪过程中普遍存在的控制流分歧[25]、访存不规则以及资源浪费等问题。正因如此，RTA 能够显著的提升光线追踪任务的执行效率，降低了能耗，并为扩展支持多种树型数据结构的应用提供了硬件基础。

具体来说，RTA 的性能优势主要体现在以下四个方面：

(1) RTA 通过固定功能单元（Fixed-Function Units）来执行树节点的遍历和相交测试。传统 GPU 在执行节点遍历与三角形相交测试时，往往需要频繁的进行指令译码、动态调度、寄存器访问与复杂的指令控制。这为硬件运行带来了大量的额外开销。而在 RTA 中，每一个节点交集测试和叶节点的相交测试操作，均交由专门设计的硬件流水线完成。执行测试的逻辑相对固定，指令流水化运行，基本避免了动态指令解析的损耗。研究结果表明，在使用固定功能单元后，RTA 可平均消除约 91% 的动态 ALU 指令与控制流指令，大幅降低了指令调度的开销，减少了给予寄存器的压力，充分增加了单位时钟周期内的有效计算次数。这种优化不仅提高了执行效率，还使得计算功耗大幅下降，为大规模的实时光追奠定了应用基础。

(2) 为了解决光线遍历过程当中普遍存在的线程分歧问题，RTA 在架构层面引入了单指

令多线程（SIMT）控制流优化机制。这代表着 RTA 通过单一硬件指令（traceRay）完成了对完整的 BVH 遍历过程的封装，并经由内部状态机来统一调度执行。相比于传统 SIMT 模型下，一旦遇到光线路径分叉，就容易导致部分线程闲置、计算单元活跃度下降等问题，通过内部硬件调度，RTA 能够有效规避运行过程中途同步等待等问题。即使不同光线在树结构中遍历方向不一，RTA 也能做到独立跟踪光线间彼此的状态，待到遍历完成统一同步。这种机制大大提高了活跃线程数以及执行单元的利用率，让 RTA 即使在面对高度分歧的场景（如全局光照、间接照明等）时，也能维持较高的并行度和吞吐量。相较于传统硬件，依赖于软件层面动态重排（如 Dynamic Ray Shuffling）的方法；RTA 在硬件层面直接对分歧进行处理，不仅取得了更高效率，而且降低了开销。

(3) RTA 里面集成了专门的硬件内存调度器，这个调度器会针对节点数据访问请求开展优化管理工作，传统的 GPU 通用内存控制器得去处理大量不同类型的数据访问请求，当面对光线遍历过程中那种小粒度且高频次的节点访问时，它大多时候不能有效地进行合并操作，这就会导致内存带宽利用率降低，然而 RTA 内的专用调度器是专门针对节点请求进行优化设计的，它能够在不同光线之间对访问请求进行聚合以及重排操作，并且在每个时钟周期内发起一次高效的内存请求。研究结果显示，RTA 内存调度器能够让节点访问带宽利用率提升将近两倍，这就明显减少了因为内存系统瓶颈而带来的性能损失，借助降低内存访问延迟并且提高访存并发度，RTA 有效地提升了整体的遍历速度，在处理大规模、复杂的 BVH 树结构的时候，它的优势则会更加明显。

(4) RTA 的引入让光线遍历和相交测试的计算负载从 GPU 的通用计算核心中分离出来，从而形成了硬件级别的功能分离和任务分工。因为遍历和相交阶段主要涉及到大量独立并且小规模的计算任务，这些任务在传统通用核心上执行的时候很容易造成资源的浪费，而凭借 RTA 专门来处理遍历任务，GPU 通用计算单元就可以专注于后续那些更加复杂的材质着色、光照求解、后处理等计算密集型的工作，在遍历阶段和着色阶段之间形成流水线式的并行之后，GPU 的整体吞吐量得到了大幅度的提升。除此之外，由于光线遍历阶段通过专用硬件以低功耗的方式执行，整体系统能效比也得到了显著的优化，这种功能分离模式提升了单帧渲染的速度，还降低了能耗以及热设计功耗的要求，对于实时渲染、移动设备、云端渲染等场景来说尤其关键。

综上所述，RTA 凭借固定功能单元来优化计算流，利用硬件调度缓解 SIMT 分歧，借助专用内存调度提升带宽利用率，并且借助硬件功能分离释放通用计算资源，在多个层次、多个角度系统地优化了光线追踪中的关键性能瓶颈。这种体系结构让现代 GPU 能够在复杂场景下实现高效、实时的光线追踪渲染，并且为后续把光线遍历能力推广到更广泛的树结构应用奠定了坚实的硬件基础。

第三章 TTA 与 TTA+ 架构设计

3.1 设计目标与方案介绍

在第二章中，本文详细分析了 RTA 架构在光线追踪中的性能优势，并指出其在通用树结构遍历任务中的适用性存在瓶颈。为解决该问题，本章提出了一种树遍历加速器架构 TTA（Tree Traversal Accelerator）及其可编程扩展 TTA+，旨在为树遍历算法提供更高效、灵活的硬件支持。

TTA 基于光线追踪加速器（RTA）架构进行扩展，支持更广泛的判断逻辑，包括键值比较和点对点距离判断两类操作，以适配数据库搜索、空间查询等不同类型的遍历任务。在设计中，TTA 尽量重用原有硬件资源，仅对固定功能单元进行必要的功能增强，兼顾加速性能与硬件开销。

在硬件资源方面，TTA 引入灵活的寄存器文件设计，将原 RTA 中的 Warp Buffer 重构为通用寄存器，支持更灵活的数据组织结构，并通过控制逻辑的可配置性，支持多种遍历策略。

然而，TTA 的固定功能限制使其难以适配复杂流程，如 N-body 模拟中的力场推理、光追中的 Ray-Sphere 交集等。为此，本文进一步提出 TTA+ 架构，采用模块化可编程设计，提供更高的扩展性，支持复杂交集逻辑组合，以满足不同类型树结构的运行需求。

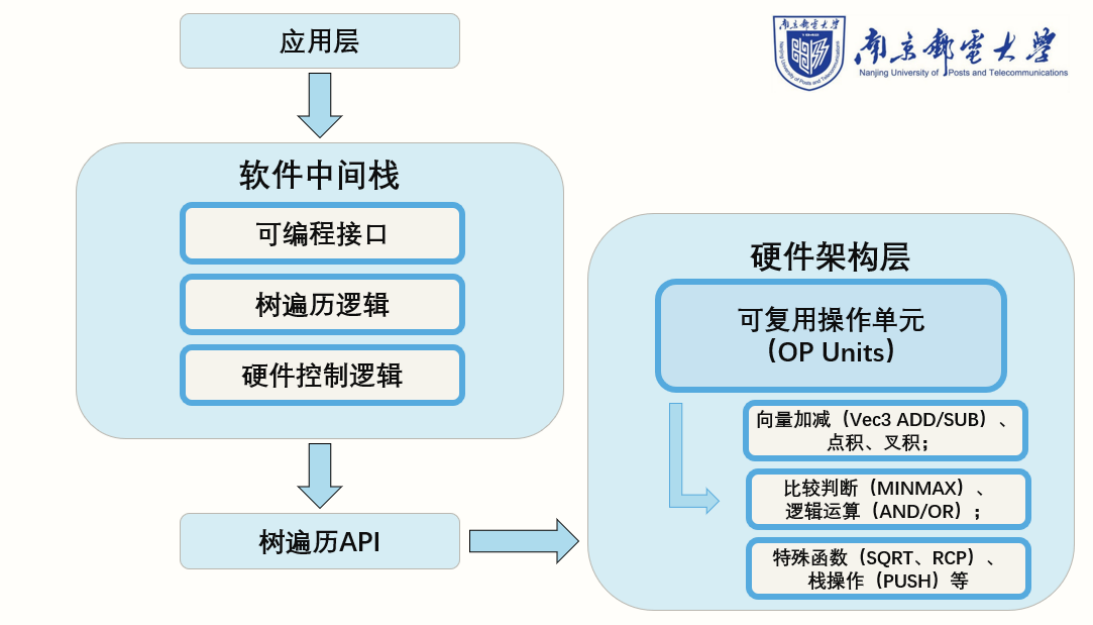


图 3.1 树遍历加速架构设计流程图

综上，TTA 追求高性能与低功耗，适用于常规规则型遍历任务；TTA+ 提供灵活 μop 组合逻辑，适用于复杂结构和动态控制流程，是一种兼顾效率与扩展性的加速架构设计方案。

3.2 TTA 架构与硬件扩展设计

TTA 架构保留了 RTA 在光追中高效的遍历能力，同时新增对树遍历常见操作的支持，包括“键值比较”和“点对点距离计算”，以支持数据库检索与物理模拟等非图形任务。

为支持键值比较操作，TTA 修改了原 Ray-Box 的 min/max 模块逻辑，使用区间映射替换原包围盒判断，通过一次比较处理最多含 9 个子节点的判断，输出 one-hot 向量标识跳转路径。

为支持点对点距离判断，TTA 对 Ray-Triangle 模块拓展，通过增加数据路径串联向量减法、点乘、平方根等操作，实现“距离小于半径”的布尔判断。

此外，TTA 重新设计了寄存器结构：将 Warp Buffer 重构为通用寄存器文件 RF，定义 Ray Register（RR）与 Node Register（NR），支持 API 配置与字段解析，提升灵活性。

评估结果表明，在仅增加 1.8% 芯片面积与 0.7% 功耗的情况下，TTA 显著提升了遍历性能，具备高能效比与良好的结构复用性，是适用于非图形遍历的高性能基础模块。

3.3 TTA+ 模块化拓展设计

尽管 TTA 架构相比传统 RTA 具有更好的通用性，但对于某些复杂任务，其固定功能单元仍难以胜任，例如 Ray-Sphere 相交测试或 N-body 力场推理等涉及非线性或不可压缩运算流程的场景。为此，本文进一步提出了一种更具灵活性的架构——TTA+。

TTA+ 采用模块化设计思想，将原本的固定交集功能单元拆解为若干可编程的微操作单元（μOps），由可重配置互联结构进行连接。每个操作单元（OP Unit）由计算模块、配置寄存器（Config Regs）以及操作目标表（OP Destination Table）组成，可依据任务特性动态配置运算路径。

图 3.2 展示了 TTA+ 的总体结构：

在执行流程中，OP 单元从互联通道获取输入数据，解析操作指令后执行相应计算，并将中间结果传递至下一个单元。该流程由操作目标表控制跳转方向，允许灵活组装复杂逻辑序列。TTA+ 架构中每条遍历指令可以映射为一个 μOp 流，模块之间通过流水方式完成组合。

为了支持更广泛的运算需求，TTA+ 集成了多种常用运算单元，如向量加减、点乘、叉乘、比较器、平方根、逻辑运算等。表 3.1 展示了各类单元的功能与执行延迟。

此外，TTA+ 中还引入了专用的 PUSH 单元，用于将中间结果写入遍历栈，支持动态节点管理。整体上，TTA+ 通过模块解耦、运算可编程，为复杂树遍历任务提供了灵活支撑，是针对多样化场景的结构增强型方案。

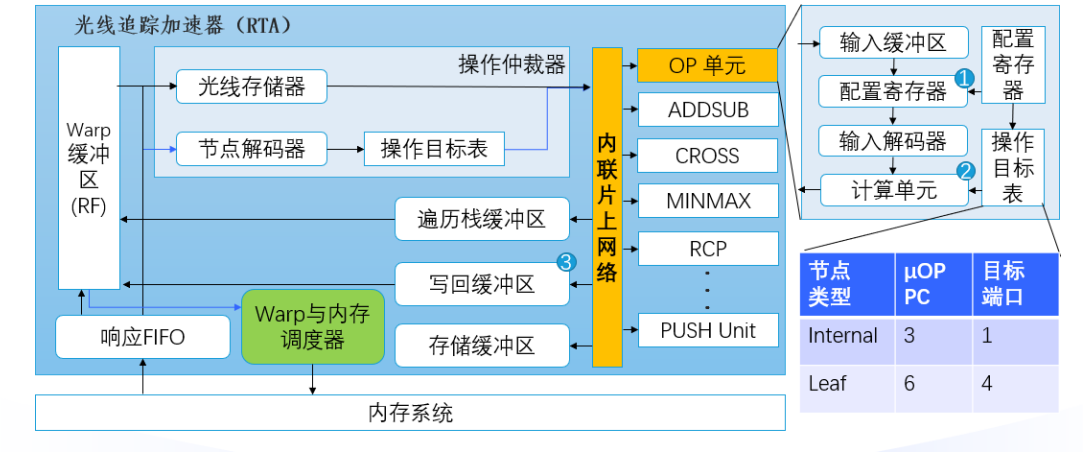


图 3.2 TTA+ 架构基本结构示意图

表 3.1 TTA+ 架构中的运算单元类型与延迟

运算单元类型	描述	延迟周期数
Vec3 加/减	三维向量加减 $(\vec{v}_1 \pm \vec{v}_2)$	4
乘法器	FP32 标量乘法	4
倒数 (RCP)	计算 $1/x$	4
叉乘单元	三维向量叉乘 $(\vec{v}_1 \times \vec{v}_2)$	5
点乘单元	三维向量点乘 $(\vec{v}_1 \cdot \vec{v}_2)$	5
Vec3 比较	每分量执行 $a \le b$ 判断	1
MINMAX 单元	$MIN(a, MAX(b, c))$ 类操作	1
MAXMIN 单元	$MAX(a, MIN(b, c))$ 类操作	1
逻辑运算单元	AND / OR / XOR / NOT	1
开方单元 (SQRT)	计算 $\sqrt{x}$	11
R-XFORM 单元	光线变换矩阵乘法	4

3.4 本章小结

本章围绕树遍历任务的加速需求，提出了两种新型 GPU 加速器架构：TTA 与 TTA+。

TTA 架构在保留 RTA 固定功能单元的基础上，拓展了键值比较与距离判断模块，并通过寄存器重构、逻辑复用，实现对通用树结构任务的支持。其硬件开销低、执行效率高，适用于索引搜索与空间判断类任务。

针对复杂控制流和结构动态性高的遍历任务，TTA+ 进一步提出模块化微操作设计，通过组合多种 OP 单元构建复杂判断逻辑，兼顾灵活性与扩展性，适用于高复杂度几何计算与非图形树结构遍历任务。

TTA 与 TTA+ 分别代表树遍历加速器设计中对性能与通用性的不同取向，二者共同为构建高能效、可配置的 GPU 遍历加速系统提供了架构基础。

第四章 编程接口与配置方法

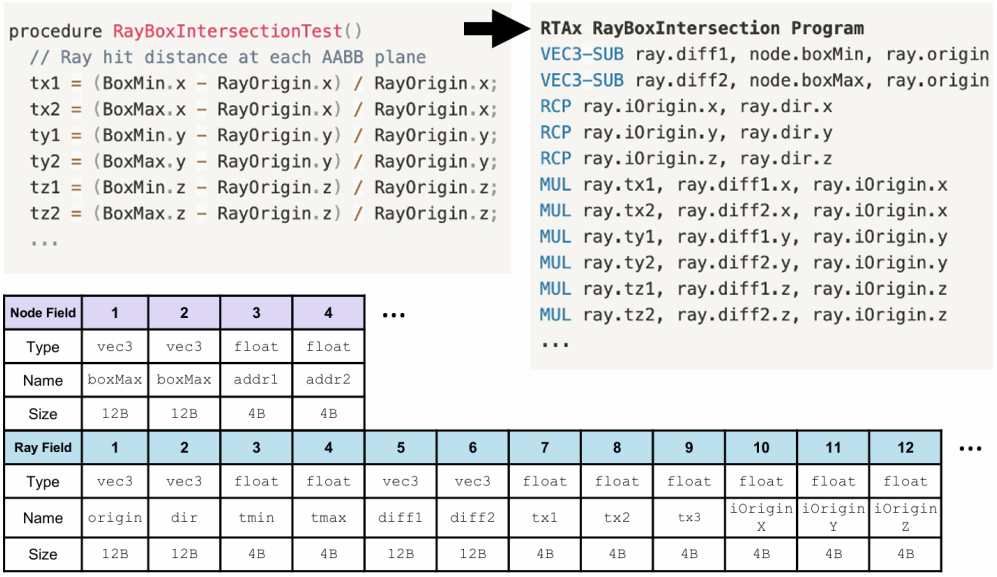
为了在通用树结构中实现可编程 TTA/TTA+ 的运行及调试，本章将介绍相关 API 接口的设计与配置流程，主要包括 μop 调度、字段配置、数据流控制等接口逻辑介绍。

4.1 μop 调度流程

为了支持 TTA+ 所需的更灵活的运算路径，本文扩展了指令配置机制，引入了对“μop”（微操作）列表的支持。具体配置方法如下：

- **输入缓冲区配置（Input Buffer）：** 包含光线或查询点的参数，如位置、方向等；
- **配置寄存器（Config Regs）：** 用于指示各个运算单元所需的输入字段（例如从 Ray Buffer 中读取哪个字段作为被比较值）；
- **操作目标表（OP Dest Table）：** 维护运算结果的目标端口，根据 μop 的程序计数器（PC）以及节点类型，判断下一个数据应路由至哪个运算模块；
- **μop 程序序列：** 每个 μop 对应一个特定操作（如 SUB、DOT、MINMAX、RCP、SQRT 等），并依序通过内部互联总线执行。

配置方式如图 4.1 所示，每个操作单元内部拥有独立的执行资源，还配备了配置寄存器与目标表，在支持流水执行的同时确保 μop 间数据依赖关系的正确传递。



Node Field	1	2	3	4	...
Type	vec3	vec3	float	float	
Name	boxMax	boxMax	addr1	addr2	
Size	12B	12B	4B	4B	

Ray Field	1	2	3	4	5	6	7	8	9	10	11	12	...
Type	vec3	vec3	float	float	vec3	vec3	float	float	float	float	float	float	
Name	origin	dir	tmin	tmax	diff1	diff2	tx1	tx2	tx3	iOrigin X	iOrigin Y	iOrigin Z	
Size	12B	12B	4B	4B	12B	12B	4B	4B	4B	4B	4B	4B	

图 4.1 TTA+ 交集测试示例

4.2 接口配置方法

为方便仿真模型的使用与开发，本文针对树遍历程序的部署与执行，设计实现了一套接口封装。典型代码样例如下：

- **ConfigNodeLayout() 和 ConfigRayLayout()**：定义内部节点、叶子节点与射线的字段偏移。
- **ConfigIntersection()**：动态加载用户自定义的 Ray-Box 与 Ray-Triangle 相交的 μop 程序。
- **ConfigTermination()**：指定遍历流程中 μop 执行的终止条件。
- **TTATraversalExecutor::CreatePipeline() 和 ExecuteTraversal()**：创建并提交遍历任务，驱动执行各 μop 模块。

4.3 编程接口调试及实现

为确保 TTA 与 TTA+ 架构在 Vulkan-Sim 平台可正确运行，本文同步进行了调试与验证工作，如图 4.2 所示：

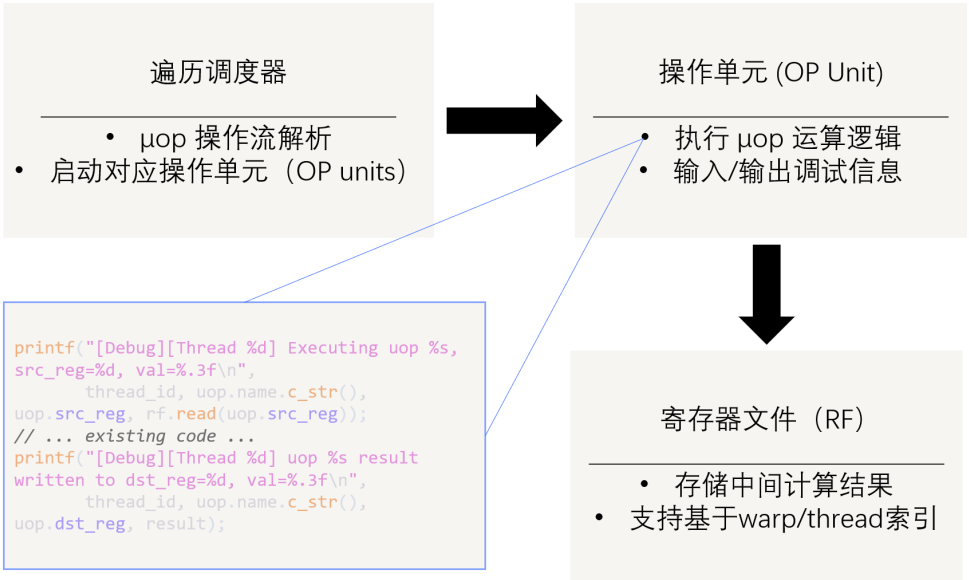


图 4.2 编程接口调试状态图

在 μop 调度模块中，逐条指令均记录其输入输出值与线程 ID，验证操作数依赖关系与数据传递准确性。在 warp 并发执行中引入 Buffer Token 排队机制，以解决共享缓冲区数据覆盖问题。最终，仿真器依据配置的 μop 列表完成遍历流程。

4.4 本章小结

本章介绍了 TTA 与 TTA+ 架构下可编程树遍历操作的实现方式，重点讲解了 μop 调度机制、配置 API 设计与调试方法。通过引入模块化 μop 控制单元与灵活配置方式，TTA+ 不仅提升了系统通用性，也为复杂结构树遍历任务提供了可行的加速路径，为下一章性能测试打下基础。

第五章 实验结果与分析

本章通过构建仿真平台，结合典型的树遍历应用任务，对前文设计的 TTA 与 TTA+ 架构进行实验验证与性能评估，涵盖基准程序运行效果、光线追踪兼容性、以及硬件资源开销分析等方面。

5.1 Vulkan-Sim 仿真平台与基准测试

本研究基于 Vulkan-Sim 平台进行实现，该平台支持自定义 GPU 指令集扩展，具备高灵活度与调试能力。在测试过程中，分别构建了面向 TTA 与 TTA+ 的指令模拟器模块，并整合各类遍历测试任务（如 B 树、N-Body、RTNN 等）。图 5.1 为实验平台的组成结构图。

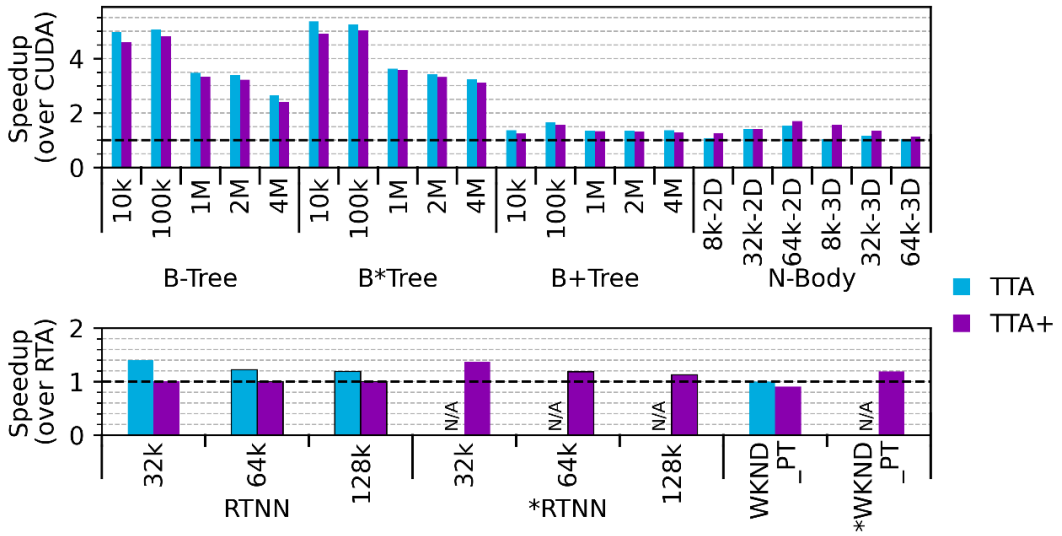


图 5.1 Vulkan-Sim 仿真平台结构示意图

5.2 评估方法与基准工作负载描述

实验以吞吐性能、资源开销、功耗估计等为评估维度，构建多个树遍历典型任务，包括：

- **B 树索引搜索：**执行大量 Query-Key 比较操作，测试 TTA 键值比较模块效率；
- **半径查询（Radius Search）：**通过欧式距离判断点云聚类，考察点对点距离加速性能；
- **RTNN 任务：**验证 TTA+ 对光追场景的支持及灵活交集能力。

5.3 树遍历应用性能评估

本节从多个角度分析 TTA/TTA+ 在实际树遍历应用中的加速效果。

5.3.1 树遍历基准应用加速性能分析

实验表明：在 B 树查询中，TTA 比 GPU 通用核执行平均提升 5.7 倍吞吐率；在半径查询中，TTA+ 比通用核提升约 1.7 倍性能。

5.3.2 RTNN / WKND_PT 光线-球体交集分析

RTNN 任务中，TTA+ 通过可编程交集逻辑顺利完成 Ray-Sphere 判断，验证了其在复杂几何判断中的通用性与可配置性。

5.4 光线追踪兼容性验证

使用 WKND_PT 场景（含多类光线交互）进行兼容性测试，结果表明 TTA+ 能正确处理所有包围盒与三角形/球体交集流程，且兼容标准 traceRay 流程指令。

5.5 硬件开销与能效对比

结合硬件合成结果分析如下：

- TTA 在保留原 RTA 逻辑下仅增加约 1.8% 面积、0.7% 功耗；
- TTA+ 在引入 μop 单元后，面积增加约 4.1%，功耗增加 1.2%；
- 在单位面积性能比（PPA）维度下，TTA 具最高性价比，TTA+ 具最高灵活度。

5.6 本章小结

本章基于 Vulkan-Sim 平台，完成了 TTA 与 TTA+ 在多个典型树遍历任务下的仿真实验与评估。从加速比、通用性、能效比等角度出发，验证了本文架构设计的有效性与适用性。其中，TTA 适用于高性能规则型任务，TTA+ 适用于结构复杂、流程灵活的应用场景。

第六章 总结与展望

本章对全文研究工作进行总结，并探讨当前设计的局限性及未来发展方向。

6.1 研究总结

本文围绕 GPU 上通用树遍历任务效率低下的问题，设计并实现了两种加速架构：TTA 与 TTA+，并通过 Vulkan-Sim 平台验证其功能与性能。主要研究成果包括：

- 提出 TTA 架构，复用 RTA 模块以支持键值比较与距离判断，适配 B 树与空间搜索等场景；
- 设计 TTA+ 架构，引入模块化 μop 操作单元，支持可编程复杂树遍历任务；
- 构建仿真平台与接口 API，支持灵活调度与调试；
- 实现树遍历任务在 TTA/TTA+ 下的完整加速流程，并进行性能验证。

TTA 结构适用于结构清晰、路径规则的遍历场景，而 TTA+ 则面向逻辑复杂、需动态判断的任务，二者各具优势，可互补适应广泛的树遍历应用需求。

6.2 局限性分析

虽然 TTA 与 TTA+ 在加速能力上展现出良好性能，但仍存在如下局限：

- 架构设计尚未部署于真实硅芯片，评估数据基于仿真平台；
- TTA+ 的灵活调度存在一定控制路径开销，对极端复杂数据流仍难充分覆盖；
- 当前 API 接口配置尚需人工调整，尚未构建统一的高层语言编译器支持；
- 部分资源管理（如互联仲裁）仍采用简化策略，尚未模拟底层调度冲突。

6.3 未来研究方向

未来可从以下几个方面进一步深入：

- 推进硬件原型实现，并验证在真实应用中的部署效果；
- 构建针对树结构算法的 μDSL（微领域语言）与编译器，自动生成配置指令；
- 优化互联调度机制，引入动态仲裁与缓存一致性支持；
- 拓展支持多类型图结构遍历，为图神经网络等任务提供底层加速。

通过以上方向的研究，有望构建面向复杂结构遍历的通用高能效 GPU 加速器，为数据库索引、物理仿真、图形计算等领域提供更广泛的支撑。

参考文献

[1] 牟磊. 基于 Barnes Hut 算法的 N-body 问题模拟 [J]. 福建电脑, 2010, 26(8): 115-116.

[2] ERNST M, VOGELGSANG C, GREINER G. Stack Implementation on Programmable Graphics Hardware[C]//VMV. 2004: 255-262.

[3] FOLEY T, SUGERMAN J. KD-tree acceleration structures for a GPU raytracer[C/OL]//Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware. Los Angeles California: ACM, 2005: 15-22[2025-05-21]. https://dl.acm.org/doi/10.1145/1071866.1071869.

[4] KAPLAN M R. Space-tracing: A constant time ray-tracer[C]//SIGGRAPH' 85 State of the Art in Image Synthesis seminar notes: 卷 18. 1985: 149-158.

[5] HORN D R, SUGERMAN J, HOUSTON M, 等. Interactive k-d tree GPU raytracing[C/OL]//Proceedings of the 2007 symposium on Interactive 3D graphics and games. New York, NY, USA: Association for Computing Machinery, 2007: 167-174[2025-05-21]. https://doi.org/10.1145/1230100.1230129.

[6] SIMONSEN L O, THRANE N. A comparison of acceleration structures for GPU assisted ray tracing[J/OL]. Master' s thesis, University of Aarhus, 2005[2025-05-21]. https://www.rose-hulman.edu/class/cs/csse451/Resources/Papers-hierarchy/Thrane-A%20Comparison%20of%20Acceleration%20Structures%20for%20GPU%20Assisted%20Ray%20Tracing.pdf.

[7] GOLDFARB M, JO Y, KULKARNI M. General transformations for GPU execution of tree traversals[C/OL]//Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. New York, NY, USA: Association for Computing Machinery, 2013. https://doi.org/10.1145/2503210.2503223.

[8] KARRAS T. Maximizing parallelism in the construction of BVHs, octrees, and k-d trees[C]//Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics. Goslar, DEU: Eurographics Association, 2012: 33-37.

[9] MEISTER D, OGAKI S, BENTHIN C, 等. A Survey on Bounding Volume Hierarchies for Ray Tracing[J]. Computer Graphics Forum, 2021, 40(2): 683-712.

[10] WOOP S, BENTHIN C, WALD I, 等. Exploiting local orientation similarity for efficient ray traversal of hair and fur[C]//Proc. ACM Conf. on High Performance Graphics (HPG): 卷 3. 2014.

[11] MEISTER D, BOKSANSKY J, GUTHE M, 等. On ray reordering techniques for faster GPU ray tracing[C]//Proc. ACM SIGGRAPH Symp. on Interactive 3D Graphics and Games (I3D). 2020.

[12] WALD I, BOULOS S, SHIRLEY P. Ray tracing deformable scenes using dynamic bounding volume hierarchies[J]. ACM Trans. Graph., 2007, 26(1): 6-es.

[13] LEE J, LEE W J, SHIN Y, 等. Two-AABB traversal for mobile real-time ray tracing[C/OL]//SIGGRAPH Asia 2014 Mobile Graphics and Interactive Applications. New York, NY, USA: Association for Computing Machinery, 2014: 1-5[2025-05-21]. https://doi.org/10.1145/2669062.2669088.

[14] BURGESS J. RTX on—The NVIDIA Turing GPU[J]. IEEE Micro, 2020, 40(2): 36-44.

[15] WU S, JIANG D, OOI B C, 等. Efficient B-tree based indexing for cloud data processing[J]. Proceedings of the VLDB Endowment, 2010, 3(1–2): 1207-1218.

[16] 陈世敏. 树状结构大数据类型的高效支持 [J]. 大数据, 2018, 4(4): 35-43.

[17] 郭辉. 面向图计算的 GPU 体系结构优化关键技术研究 [D/OL]. 国防科技大学, 2018. https://link.cnki.net/doi/10.27052/d.cnki.gzjgu.2018.000252.

[18] 陈俊杰, 朱维, 王宪锴, 等. 空间索引技术及其 GIS 应用综述 [J]. 地理与地理信息科学, 2024, 40(2): 1-10.

[19] HENNEBERG J, SCHUHKNECHT F. RTIndeX: Exploiting Hardware Accelerated GPU Raytracing for Database Indexing[J]. arXiv preprint arXiv:2303.01139, 2023.

[20] ZHU Y. RTNN: Accelerating Neighbor Search Using Hardware Ray Tracing[C]//ACM Symposium on Principles and Practice of Parallel Programming (PPoPP). 2022: 76-89.

[21] BAGLA J S. Cosmological N-body Simulation: Techniques, Scope and Status[J]. Current Science, 2005: 1088-1100.

[22] LASHGAR A, BANIASADI A, KHONSARI A. Warp size impact in GPUs: large or small?[C/OL]//Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units. New York, NY, USA: Association for Computing Machinery, 2013: 146-152. https://doi.org/10.1145/2458523.2458538.

[23] FUNG W W, SHAM I, YUAN G, 等. Dynamic Warp Formation and Scheduling for Efficient GPU Control Flow[C]//IEEE/ACM Symposium on Microarchitecture (MICRO). 2007.

[24] LIU L, CHANG W, DEMOULLIN F, 等. Intersection Prediction for Accelerated GPU Ray Tracing[C]//IEEE/ACM Symposium on Microarchitecture (MICRO). 2021: 709-723.

[25] RHU M, EREZ M. The dual-path execution model for efficient GPU control flow[C]//2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA). 2013: 591-602.

[26] SAED M, CHOU Y H, LIU L, 等. Vulkan-Sim: A GPU Architecture Simulator for Ray Tracing[C/OL]//2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO). Chicago, IL, USA: IEEE, 2022: 263-281[2024-09-19]. https://ieeexplore.ieee.org/document/9923844/.

[27] AILA T, LAINE S. Understanding the efficiency of ray traversal on GPUs[C/OL]//Proceedings of the Conference on High Performance Graphics 2009. New Orleans Louisiana: ACM, 2009: 145-149[2025-03-26]. https://dl.acm.org/doi/10.1145/1572769.1572792.

[28] AWAD M A, ASHKIANI S, JOHNSON R, 等. Engineering a High-Performance GPU B-Tree[C]//ACM Symposium on Principles and Practice of Parallel Programming (PPoPP). 2019: 145-157.

[29] AWAD M A, PORUMBESCU S D, OWENS J D. A GPU Multiversion B-Tree[C]//IEEE/ACM Conference on Parallel Architectures and Compilation Techniques (PACT). 2023: 481-493.

[30] WALD I, MORRICAL N, ZELLMANN S, 等. Using hardware ray transforms to accelerate ray/primitive intersections for long, thin primitive types[C]//Proc. Int' l Conf. on Computer Graphics and Interactive Techniques (SIGGRAPH): 卷 3. 2020.

[31] LIU L, SAED M, CHOU Y H, 等. LumiBench: A Benchmark Suite for Hardware Ray Tracing[C/OL]//2023 IEEE International Symposium on Workload Characterization (IISWC). Ghent, Belgium: IEEE, 2023: 1-14[2024-09-19]. https://ieeexplore.ieee.org/document/10289559/.

[32] YOSHIMURA A, HARADA T. Subspace culling for ray-box intersection[C]//Proc. ACM SIGGRAPH Symp. on Interactive 3D Graphics and Games (I3D). 2023.

[33] NAH J H, MANOCHA D. SATO: Surface Area Traversal Order for Shadow Ray Tracing[J]. Comput. Graph. Forum, 2014, 33(6): 167-177.

[34] CHOU Y H, NOWICKI T, AAMODT T M. Treelet prefetching for ray tracing[C]//Proc. IEEE/ACM Symp. on Microarch. (MICRO). 2023.

[35] INTEL CORPORATION. Introduction to the Xe-HPG architecture[EB/OL]. (2022). https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-the-xe-hpg-architecture.html.

[36] BALASUBRAMONIAN R, KAHNG A B, MURALIMANOHAR N, 等. CACTI 7: New tools for interconnect exploration in innovative off-chip memories[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2017, 14(2): 1-25.

[37] KANDIAH V, PEVERELLE S, KHAIRY M, 等. AccelWattch: A Power Modeling Framework for Modern GPUs[C]//Proc. IEEE/ACM Symp. on Microarch. (MICRO). 2021: 738-753.

[38] FOG A. Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD, and VIA CPUs[EB/OL]. (2022). https://www.agner.org/optimize/instruction_tables.pdf.