计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
[doi: 10.15888/j.cnki.csa.010316] [CSTR: 32024.14.csa.010316]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

GPU 性能分析综述①

梁鑫妮, 翟高寿

(北京交通大学 计算机科学与技术学院, 北京 100044)
通信作者: 翟高寿, E-mail: gszhai@bjtu.edu.cn

摘　要: 随着图形处理器 (graphics processing unit, GPU) 在高性能计算、人工智能等领域的广泛应用, GPU 性能分析的重要性愈加凸显. 然而, 由 GPU 的并行处理模型、复杂调度机制、多层级存储体系以及 CPU-GPU 异构通信等引发的问题, 使得相关性能分析异常困难. 本文首先回顾了 GPU 执行模型和面向 GPU 性能分析的屋顶线模型, 归纳给出了典型的 GPU 性能瓶颈. 通过梳理现有研究范式, 提出了一种以实现机制为纲、分析粒度与工具形态为目的 GPU 性能分析方法分类体系, 宏观上划分为基于硬件性能计数器的方法、基于微基准测试的方法、基于软件插桩的方法、基于模拟与建模的方法以及基于二进制插桩的方法这 5 类. 在此基础上, 重点讨论了各类方法及技术的优势与不足. 特别地, 以 Neutrino 为例, 分析揭示了传统工具无法观测微观调度行为的局限性. 最后, 总结当前 GPU 性能分析面临的挑战, 并指出下一代性能分析工具的发展方向即可编程、细粒度和智能化.

关键词: GPU 性能分析; 性能瓶颈; 二进制插桩; 微基准; 屋顶线模型

引用格式: 梁鑫妮,翟高寿.GPU 性能分析综述.计算机系统应用. http://www.c-s-a.org.cn/1003-3254/10316.html

Overview of GPU Performance Analysis

LIANG Xin-Ni, ZHAI Gao-Shou

(School of Computer Science & Technology, Beijing Jiaotong University, Beijing 100044, China)

Abstract: With the widespread application of graphics processing units (GPUs) in high-performance computing, artificial intelligence, and other fields, the importance of GPU performance analysis has become increasingly prominent. However, issues arising from GPUs' parallel processing model, complex scheduling mechanisms, multi-level memory systems, and heterogeneous CPU-GPU communication make such performance analysis extremely difficult. This study first reviews the GPU execution model and the roofline model for GPU performance analysis and summarizes typical GPU performance bottlenecks. Based on a review of existing research paradigms, a classification system for GPU performance analysis methods is proposed, with implementation mechanisms as its guiding principle and analysis granularity and tool forms as its objectives. At the macro level, these methods are divided into five categories, including methods based on hardware performance counters, methods based on micro-benchmarks, methods based on software instrumentation, methods based on simulation and modeling, and methods based on binary instrumentation. On this basis, this study discusses the advantages and shortcomings of each type of method and technique. In particular, with Neutrino as an example, the analysis reveals the limitations of traditional tools in observing microscopic scheduling behavior. Finally, the current challenges faced by GPU performance analysis are summarized, and the development directions of next-generation performance analysis tools, including programmability, fine granularity, and intelligence, are identified.

Key words: GPU performance analysis; performance bottleneck; binary instrumentation; micro-benchmark; roof-line model

① 收稿时间: 2026-04-23; 修改时间: 2026-05-12; 采用时间: 2026-06-04; csa 在线出版时间: 2026-08-21

1 引言

1.1 GPU 计算发展背景

1999 年, 英伟达公司首次提出图形处理器 (graphics processing unit, GPU) 的概念. 此后, Khailany 等人[1]提出了一种可加速多媒体应用的流处理器架构; Buck 等人[2]通过扩展 C 语言实现了 GPU 编程支持, 二者共同奠定了 GPU 通用计算基础. 随着统一计算设备架构 CUDA (compute unified device architecture)[3]的推出, GPU 高速并行计算编程更加便利, 而且计算性能相较通用处理器有数十至数百倍的提升. 在摩尔定律放缓的当下, 黄仁勋预言 GPU 计算能力将推动 AI 性能逐年翻倍[4]. 伴随 AI 风靡全球, GPU 重要性愈发凸显, 同时也带来了一系列挑战.

1.2 GPU 性能分析的意义

随着应用规模的扩展和复杂化以及硬件架构的快速迭代, GPU 性能分析已经成为 GPU 计算领域的核心研究内容[5]. 有效的 GPU 性能分析能够帮助开发者快速定位性能瓶颈, 确定优化方向, 为软件和硬件系统的迭代升级提供指导. 在大规模训练和高性能计算任务中, 提升 GPU 的性能不仅能够显著降低成本、提升运行效率, 更能够推动相关研究继续突破. 因此, 理解和掌握 GPU 性能分析的方法、工具及面临的挑战, 对推动计算机领域的发展具有重要意义.

1.3 GPU 性能分析面临的困难

尽管 GPU 性能分析相关研究层出不穷, 但鉴于 GPU 独特的架构, 传统的性能分析方法难以适用. GPU 性能分析面临诸多困难, 主要包括 3 个方面.

(1) 大规模并行及复杂的调度机制. GPU 采用单指令多线程 (single instruction multiple thread, SIMT) 的执行模型, 通过将所有的线程块分配到可用的流式多处理器 (streaming multiprocessor, SM) 上运行, 每个流式多处理器可同时运行多个线程块[6]. 硬件调度器将这些逻辑线程映射到流式多处理器上执行的过程是不透明的. 这种“黑盒”的性质使得性能分析变得困难, 开发者看不到微观的调度冲突, 只能观察到宏观的性能指标.

(2) 复杂的存储层次结构与访存行为. GPU 存储层级包含寄存器、本地内存、共享内存、全局内存、常量内存、纹理内存和多级缓存. 存储设计是分层的, 越靠近流式多处理器核心, 速度越快, 容量越小. 这样的设计导致每一层在容量、带宽和延迟上差异巨大. 并且, 不同的访问模式会有不同的带宽. 因此, 分析内存瓶颈不仅需要明确访存带宽的利用率, 也需要理解数据在存储层次间的移动情况, 这对访存追踪工具的精度与粒度提出了极高的要求.

(3) 快速迭代的硬件架构与新型计算单元. GPU 架构持续引入新的计算单元, 比如在深度学习中的张量核心和在图形渲染中的光线追踪核心[7]. 这些新单元的引入, 出现了许多新的指令集和性能特性, 需要性能分析工具和方法快速适应, 并能够解释相应的性能瓶颈.

1.4 论文贡献

GPU 性能分析领域面临许多独特的难题和挑战. 本文通过对比 5 种技术路线, 指出现有工具在“细粒度”“灵活性”和“开销”方面的局限性, 并论证以“二进制插桩”“开放标准”为代表的新兴方法是打破僵局、实现下一代“可编程、全栈、智能化”性能分析的关键所在. 本文的主要贡献包括以下 3 个方面.

(1) 系统梳理 GPU 基础理论, 建立 GPU 性能分析方法分类体系, 把现有 GPU 性能分析方法划分为基于硬件性能计数器的方法、基于微基准测试的方法、基于软件插桩的方法、基于模拟与建模的方法及基于二进制插桩的方法. 针对每类性能分析方法, 通过具体案例分析相关原理与实现方式.

(2) 针对 5 类性能分析方法, 分别通过具体案例分析相关原理与实现方式, 并对有关方法技术进行优劣对比.

(3) 总结 GPU 性能分析面临的挑战, 展望 GPU 性能分析方法的下一步演化方向, 指出未来应当以二进制插桩和开放化标准的方式获取性能指标, 实现从依赖硬件厂商提供的、固定的被动观测, 走向由用户定义的、灵活的主动编程.

2 GPU 性能分析基础

2.1 GPU 执行模型

GPU 执行模型是理解 GPU 性能分析的基础. 现代通用的 GPU 主要采用单指令多线程模型[8], 其中的每个流式多处理器 (内部架构如图 1 所示) 同时管理着多个线程束, 而每个线程束由 32 个并行线程组成. 由于每个线程可独立执行, 同时, 当某个线程束因访存操作而暂停运行时, 流式多处理器可通过切换至其他就绪的线程束继续执行, 从而将访存延迟隐藏在计算之中. 于是, 流式多处理器得到了高效利用.

多线程指令发射器是单指令多线程模型的调度引

擎, 能够将同一线程束的指令, 分发给下方的多个标量处理器. 与此同时, 一个线程束内的所有线程在控制逻辑的驱动下, 在不同的标量处理器上锁定步调地执行相同的指令, 但各自处理不同的数据, 从而体现了“单指令、多线程”的并行特性.

图 1 GPU 流式多处理器内部架构示意图

因此, 单指令多线程、线程束分组以及线程束切换隐藏访存延迟策略等 GPU 执行模型的核心特性决定了 GPU 的性能表现. 进一步说, 当线程束内出现分支分化或线程束数量不足时, 会降低执行单元的利用率, 从而影响 GPU 的整体计算性能; 若线程束内的线程访存请求无法合并为连续地址访问, 会出现冗余访存事务, 降低带宽的有效利用率. 换言之, GPU 执行模型相关调度方式、组织形式和访存行为将会影响 GPU 的计算效率和访存效率.

2.2 屋顶线模型

GPU 执行模型特性与 GPU 的计算效率和访存效率休戚相关, 而屋顶线模型 (roof-line model) 则从上述两个维度对相关影响进行了量化描述. 屋顶线模型用于分析有关模型在特定计算平台上所能达到的理论计算性能上限[9]. 主要涉及两个指标: 峰值计算性能 (计算屋顶) 和峰值内存带宽 (内存屋顶). 如果程序的计算量大, 算术强度高, 那么性能可能接近计算瓶颈; 如果程序的算术强度低, 但是由于不断地访问数据, 性能会被最低的带宽所限制, 于是可能接近内存瓶颈[10]. 但是, GPU 复杂的硬件层次结构会使得实际情况更加复杂 (如图 2 所示). 数据可能位于全局内存、L1/L2 缓存或共享内存/寄存器中. 如果数据在 L1/L2 缓存命中, 则延迟极低; 如果在全局内存中获取, 则会产生高延迟. 因此程序的性能受限于最慢的访问方式. 内存控制器和高带宽内存的总带宽有限, 如果所有的流式多处理器同时大规模访问, 就会达到内存瓶颈.

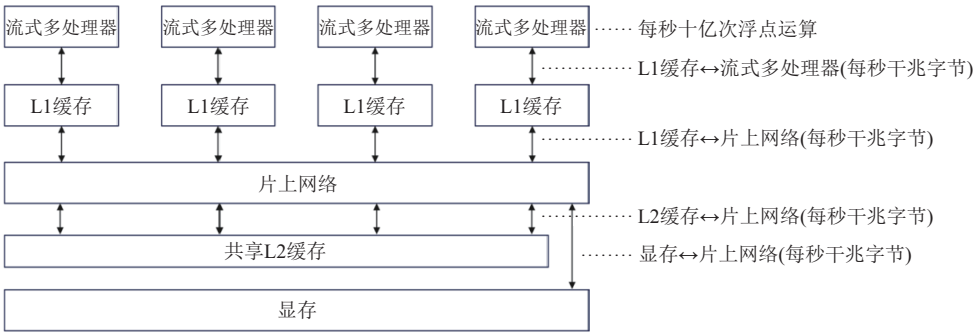

图 2 GPU 分层存储与计算单元硬件架构图

因此, 一个精确的 GPU 屋顶线模型应该包含许多条不同的斜线, 以代表不同层次存储的有效带宽[11]. 在图 3 中, 访存峰值和计算峰值对应实线以上为程序性能无法达到的区域, 或者超过了计算峰值性能, 或者超过了访存峰值带宽. 访存峰值实线与访存 (带宽) 边界虚线之间区域对应性能较好的访存密集型程序, 计算峰值实线与计算边界虚线之间区域对应性能较好的计算密集型程序. 访存 (带宽) 边界虚线和计算边界虚线以下的区域为低效性能区域, 对应那些访存带宽和浮点计算性能都很低的程序.

2.3 GPU 性能瓶颈归类分析

借鉴深度学习模型性能瓶颈的系统性分类, 本文采纳并扩展了相关分析方法[12], 将 GPU 性能瓶颈划分为以下 3 类.

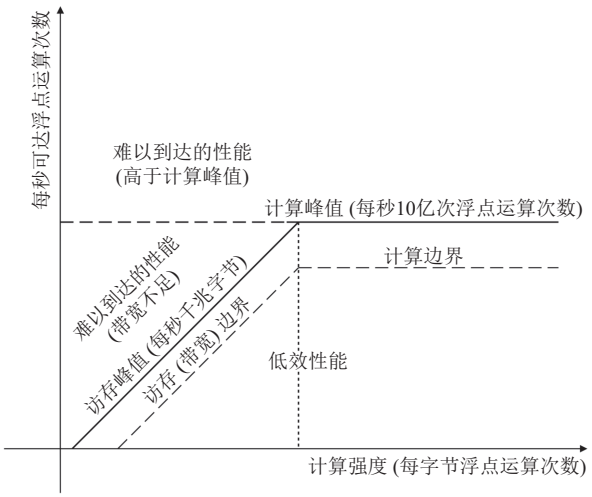

图 3 屋顶线模型图

(1) 计算瓶颈, 指 GPU 因计算单元饱和而使性能受限于算力 (而非显存带宽), 或者因计算单元无法得到充分利用而导致整体性能下降. 从 GPU 执行模型出发来看, 计算瓶颈的形成一般有两方面的原因: 一是线程束内部如果出现了分支分化, 部分线程就会被暂时屏蔽, 进而影响计算单元的有效利用率; 二是活跃线程束数量不够充足时, 流式多处理器无法通过线程束切换充分隐藏访存延迟, 计算单元就会出现空闲周期而造成浪费.

(2) 内存瓶颈[13], 指 GPU 内存访问速度受到限制, 具体表现为 GPU 因等待数据导致系统性能下降. 从 GPU 执行模型出发来看, 内存瓶颈的成因可归结为以下两个方面: 一是线程束内的访存请求如果未能合并为连续地址的访问, 它们就会被拆分成多个重复的访存事务, 进而大幅度降低有效带宽的利用率; 二是当所有流式多处理器同时进行大规模的非合并全局内存访问时, 内存控制器的总带宽会被很快耗尽, 从而形成内存瓶颈.

(3) 数据传输瓶颈. 在 CPU 与 GPU 或 GPU 与 GPU 之间, 数据传输可能形成瓶颈. 例如, 迭代过程中频繁交换数据时, 数据传输时间会大大增加. 尤其在多 GPU 或分布式场景下, 这种开销会显著制约 GPU 整体性能[14]. 从 GPU 执行模型的运行机制来看, 数据传输瓶颈的根源在于, 计算单元要求数据尽可能驻留在设备内存中以实现高效访问, 但 PCIe 或 NVLink 等链路的带宽通常会低于片上内存带宽, 所以当数据在多个设备之间进行传输的时候, 带宽差距会造成 GPU 的计算流水线被迫停滞和等待数据准备就绪. 在迭代计算的场景下, 频繁的数据交换会让传输开销不断累积, 进而成为制约系统整体性能的关键因素. 典型场景包括多 GPU 间的显存访问和分布式训练中的梯度同步等.

3 GPU 性能分析方法分类体系构建

研究 GPU 性能分析方法能够帮助开发者定位性能瓶颈, 提升开发效率. 本文将 GPU 性能分析方法分为 5 类: 基于硬件性能计数器的方法、基于微基准测试的方法、基于软件插桩的方法、基于模拟与建模的方法以及基于二进制插桩的方法. 本节将依次对这 5 类方法进行阐述与比较. 它们在分析粒度、核心原理及核心性能指标 3 个维度的差异如表 1 所示.

表 1 GPU 性能分析方法分类对比

方法	代表工具	分析视角	核心原理	指标
基于硬件性能计数器	NVIDIA Nsight Compute	内核级/微架构级	读取GPU内部性能监控单元的固定计数器	吞吐量、利用率、事件计数
基于微基准测试	Micro-benchmark	微架构级	编写极简测试程序, 通过性能测量逆向推断硬件特性	硬件特性参数、存储系统效率
基于软件插桩	PyTorch Profiler TensorFlow Profiler NVIDIA Nsight Systems	应用级/系统级	在软件栈不同层次插入探针, 记录函数调用和内核执行等事件	执行时间、调用参数、内存分配量
基于模拟与建模	GPGPU-Sim DNNPerf	微架构级	通过软件模拟器周期精确模拟硬件或建立数学模型预测性能	周期细节、预测值
基于二进制插桩	Neutrino	指令级	在汇编级动态插入自定义探测代码	自定义事件、指令级耗时

本文采用分层分析范式, 根据观测粒度将分析视角划分为应用级、系统级、内核级、指令级和微架构级, 并且以此探析 GPU 性能分析方法与工具.

(1) 应用级: 关注完整软件应用的整体行为与端到端的性能指标, 如吞吐量、延迟、训练时间等, 在深度学习项目中较为常见.

(2) 系统级: 在操作系统或系统软件层面进行操作和控制, 重点关注 CPU、GPU 和内存等多个硬件组件

及操作系统的进程或线程之间的交互和资源竞争[15].

(3) 内核级: 聚焦在单个 GPU 内核的执行效率, 主要分析内核的执行时间、占用率、计算吞吐量、内存带宽利用率、算术强度等[16].

(4) 指令级: 处理器中以单条指令为单位进行分析、调度和执行的层次.

(5) 微架构级: 主要关注硬件内部单元的具体行为, 如线程束调度、缓存层次等[17].

3.1 基于硬件性能计数器的方法

基于硬件性能计数器的方法 (工作流程如图 4 所示) 直接使用 GPU 内部集成的性能监控单元. 该单元在指令执行、内存访问等事件发生时, 自动累加计数器, 从而实现数据采集[18,19].

硬件性能计数器是嵌入在处理器中的专用寄存器, 能够监控和记录处理器的各种性能事件, 如 GPU 的基本使用情况、特定事件发生的次数等. 早期的计算性能分析始于 IBM 大型机的计时器中断, 之后 Unix 提供了 prof/gprof[20]工具可直接读取硬件计数器. 随着 CPU 架构的复杂化, 电源管理单元和硬件计数器成为标准的配置, 相关的代表工具有 Linux 的 perf[21]. 与此同时, 伴随 AI 的快速崛起, GPU 成为计算主力, 其性能分析也催生了英伟达中的 Nsight Compute 工具 (关键步骤如图 5 所示), 该工具围绕屋顶线模型, 提供更加便捷的计数器访问.

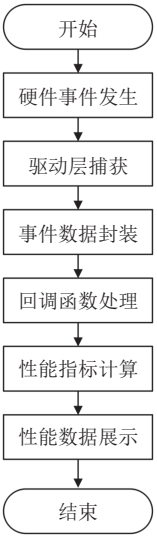

图 4 硬件性能计数器工作流程图

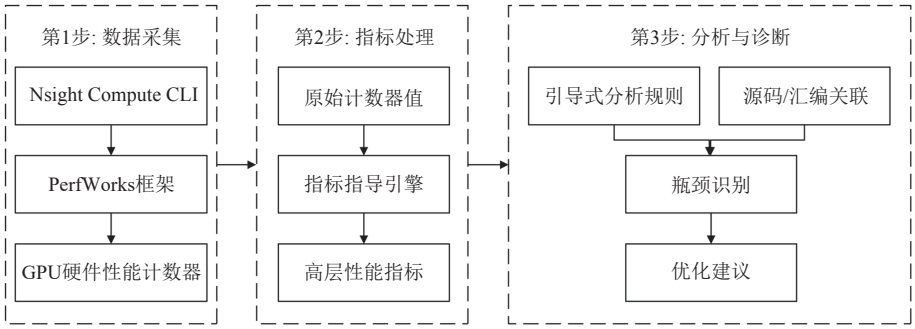

图 5 Nsight Compute 工具关键步骤示意图

Nsight Compute 通过英伟达的 PerfWorks 框架读取流式多处理器、内存控制器、缓存等单元上内置的硬件性能计数器, 之后通过内置的一套指标推导引擎将原始数据转换为有意义的指标, 最后 Nsight Compute 通过引导式分析自动识别性能限制因素, 并利用源码关联功能将抽象的指标直接定位到具体的代码行, 甚至关联到汇编指令, 从而为开发者提供精准的优化建议[22].

此外, 英伟达提供的 CUPTI (CUDA profiling tools interface) 为开发者提供了底层访问 GPU 性能计数器的编程接口, 支持在应用程序中直接嵌入性能数据的采集和分析功能, 实现了更加灵活的性能监控与分析[23]. 随着技术的发展, 硬件性能计数器涌现许多特色的研究方向: 一方面, 国产操作系统的飞速发展, 出现了基于龙芯等处理器的性能计数器性能分析计数. 还有基于 RISC-V 的新型硬件性能计数器[24]. 另一方面, Rust 等现代系统编程语言对硬件性能计数器更好地支持, 以及更多开发者关注硬件与软件的协同优化, 这一技术将会在性能优化领域发挥更加重要的作用.

3.2 基于微基准测试的方法

基于微基准测试的方法是通过编写简单、可控的微型测试程序, 系统性地测量执行事件或吞吐量, 然后反推硬件的特性. 该方法主要测量运算单元性能、存储系统效率、互联网络吞吐量及操作系统调度开销等关键性能指标.

此领域的开创性工作是 Wong 等人[25]对英伟达 Fermi 架构的微基准测试. 这项工作提出了一个称为“micro-benchmark”的套件, 并且运用这个套件测量了 CUDA 编程模型中的英伟达 GT200 架构. 这个套件关注两个影响 GPU 性能的方面: 算术运算核心和为这些算术运算核心提供指令与数据的内存层级. 每个微基准测试由 GPU 的核心代码组成, 包含一段将会在 GPU 某个电路上运行的代码片段以及该代码片段前后的事件统计代码. 同时, 一个基准内核需要完整运行两次, 抛弃第 1 次运行结果, 因为第 1 次运行时可能受到缓存未命中的影响. 具体测试流程如图 6 所示.

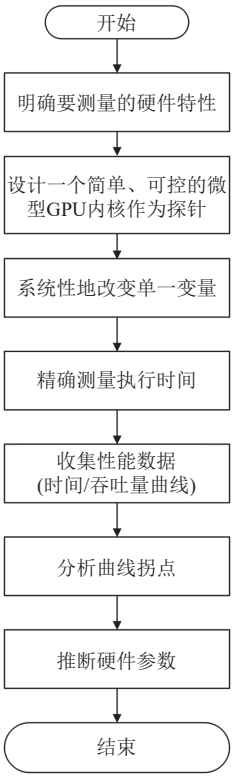

图 6 微基准测试具体流程图

该方法通过设计简单的、能够控制变量的 GPU 内核程序作为探针, 对某个特定的硬件部件, 每次改变一个参数, 观察性能变化情况. 性能曲线中的“拐点”是推断硬件参数的关键. 例如, 当访问的数组总大小超过某级缓存容量时, 平均延迟会突然跃升, 并以此反推, 这个跃升点就是对应数据大小的缓存容量. 基于这一方法, Wong 等人[25]成功揭示了 GT200 架构算术流水线延迟与吞吐、多级缓存与 TLB 层次结构, 以及 Warp 调度与分支同步机制等关键微架构参数.

在微基准测试方法的基础上, 2019 年 Jia 等人[26]在 Turing 架构的基础上进行了应用, 独立揭示了并发整数指令或浮点指令执行单元、全新的 L0 或 L1 指令缓存层次以及异步内存拷贝引擎等关键微架构创新. 由此, 出现了一系列在该方法基础之上的专用微基准工具与自动化框架, 这些工具能够有针对性地收集和分析性能数据[27]. 并且能够通过静态分析减少微基准套件的执行次数, 大幅缩短执行时间, 有利于集成到持续集成与持续交付或部署流程上[28]. 在最新的研究中通过微基准测试和多级分析剖析了 Hopper 架构, 发现该架构在实际应用中展示出了显著的加速潜力[29]. 同时, GPU 性能分析不仅需要关注硬件本身的执行效率, 还需诊断其所在的复杂软件运行的环境. 因此, 该方法被拓展到深度学习领域中, 通过在训练流程中插入标准的 GPU 内核, 可用来分析内存分配器效率, 内核启动开销等, 从而找到 GPU 在深度学习框架中性能不佳的原因[30].

3.3 基于软件插桩的方法

基于软件插桩的方法通过在软件栈的特定位置动态或者静态地插入“探针”, 记录函数调用、内核启动等时间, 生成带有时间戳和上下文信息的执行信息. 核心思想是在保留程序原有逻辑的前提下, 注入额外的观测代码. 根据工具的关注点可以细分为 3 类.

(1) 应用级方法: 这类工具经常在深度学习框架中体现, 有 PyTorch Profiler[31]和 TensorFlow Profiler[32]工具, 能够通过上下文管理器或者 API 调用, 追踪 GPU 活动的时间线, 支持分布式训练的性能分析.

(2) 系统内核级方法: eBPF 技术允许用户在不修改内核源代码的情况下, 安全地向 Linux 内核注入自定义逻辑, 实现系统监控、网络控制和系统优化[33,34]. 与前述应用级方法不同, eBPF 不依赖特定深度学习框架, 具有低开销、可动态启停的优势, 但其分析范围限于 CPU 端的 GPU 驱动和运行时调用, 无法直接对 GPU 内核内部进行指令级追踪. 在面向分布式大模型推理场景下, eInfer 利用 eBPF 技术实现了端到端的性能追踪, 能够在不修改应用程序的前提下关联 CPU 与 GPU 事件, 为生产环境的实时性提供了新的思路[35].

(3) 系统级方法: NVIDIA Nsight Systems[36]是目前 GPU 异构计算最核心、最强大的系统级性能分析工具. 其核心技术是采样与追踪的混合方法, 能够捕获整个系统所有 CPU 线程、GPU 上下文、CUDA API 调用、GPU 内核执行等时间, 并且能够直观地展示出 GPU

和 CPU 活动是否重叠, 数据传输是否出现瓶颈等问题. 此外, KPerfIR 将性能分析能力以编译器为中心集成到 Triton 基础设施中, 实现了 GPU 内部的细粒度测度和算子内部计算瓶颈的精确分析, 且开销仅为 8.2%[37].

上述 3 类方法在 GPU 性能分析中各有侧重. 应用级方法聚焦于机器学习框架, 能将性能事件直接关联到模型代码, 适用于算法瓶颈定位; 系统内核级方法在操作系统层进行低开销监控, 适合生产环境下的实时观测; 系统级方法则提供 CPU-GPU 协同执行的时间线视图, 擅长诊断跨设备交互导致的宏观瓶颈.

3.4 基于模拟与建模的方法

基于模拟与建模的方法一方面通过软件模拟 GPU 硬件每一时钟周期的行为, 另一方面建立数学分析模型去预测或复现程序的性能.

在模拟方面, 主要使用周期精确模拟器, 模拟指令从取址、译码、发射、执行到写回的完整流水线, 模拟器以时钟周期为步长, 追踪任意硬件单元在任何周期的状态. GPGPU-Sim[39]是一个广泛使用的开源 GPU 周期精确模拟器, 基于英伟达 Tesla 架构, 执行 CUDA 程序的并行线程执行 (parallel thread execution, PTX)[38] 中间表示, 并在模拟的时钟周期内推进所有硬件状态. 开源的 GPU 模拟器已经出现很多, 都面向不同的目标架构和仿真方法, 表 2 展示了不同的模拟器在指令集支持、仿真模式等方面存在差异.

当前不透明的工业设计以及公共研究的差距, 需要一个能够容易与工业设计并齐的模拟器, 因此出现了 Accel-Sim[40]模拟器, 该模拟器表现出较好的综合性能, 通过引入新的 GPU 模拟器前端、更新 GPGPU-Sim 的性能模型和新的前端和性能模型构建的基础设施 3 种机制, 让 GPU 模拟器追上工业步伐. 首先 Accel-Sim 使用英伟达的 NVBit[46]工具对 CUDA 的应用程序进行二进制插桩, 生成流式汇编 (streaming assembly, SASS) 格式的跟踪文件 trace. 之后将生成的跟踪文件导入 GPGPU-Sim 中进行模拟. 最后, Accel-Sim 提供 Correlator 工具能够将模拟结果与真实的硬件性能数据进行对比和分析, Tuner 工具能够优化模拟器的性能. Accel-Sim 在保持高精度的同时, 提升了对新一代 GPU 架构的适配能力.

表 2 不同 GPU 模拟器的对比

模拟器	指令集	前端模式	架构	负载数	仿真速率	多线程支持	手动调整英伟达库
GPGPU-Sim[39]	vISA+NVIDIA mISA (GT200)	Execution	Fermi	14	3	否	否
gem5-APU[41]	vISA+AMD mISA	Execution	AMD	10	—	否	否
MGPU-Sim[42]	vISA+AMD mISA	Execution	AMD	7	28	是	否
MacSim[43]	vISA+Intel mISA	Trace	Fermi	—	—	否	否
Multi2Sim[44,45]	vISA+NVIDIA/AMD mISA	Execution	Kepler	24	0.8	否	否
Accel-Sim[40]	vISA+NVBit生成的mISA	Trace and Execution-driven	Kepler/Pascal/Volta/Turing	80	12.5	否	是

在建模方面, 基于程序的特征和硬件参数, 通过一组数学公式推导性能. 大部分方法都是在屋顶线模型的基础上进行演进. 在屋顶线模型提出的同年, 已经有研究在 CUDA 平台上建立性能预测模型, 该模型通过指令计数和延迟求和的方法, 将内核执行分为 3 个阶段, 计算阶段统计各类指令的延迟或吞吐、内存访问阶段的相关延迟以及调度阶段的资源竞争, 为后续的细粒度性能分析奠定了基础[47]. 但传统的建模方法需要为每种不同的 GPU 架构手动提取参数, 过程比较繁琐且不通用, 因此如何为未知的硬件快速拟合一个可用的模型是当时的挑战. 为应对这一挑战, Baghsorkhi 等人[48]提出了一个参数化模型框架, 通过轻量级的探测和学习, 利用搜索或者优化算法自动调整模型参数, 让模型的误差降低, 再用校准后的模型预测新程序的性能. 这个方法的提出为后续的神经网络性能模型奠定了基础. DNNPerf[49]是一个基于机器学习的工具, 使用图神经网络预测深度学习模型的性能, 并且预测 GPU 内存消耗和训练时间. 这标志着建模方法已经从内核级性能预测拓展至端到端的深度学习模型性能预测中.

3.5 基于二进制插桩的方法

基于二进制插桩的方法是直接对 GPU 内核的可执行代码进行修改, 在指令间插入自定义的探测代码, 进而获取相关数据对 GPU 性能进行分析. 主要的插桩层次在 SASS 指令上插桩和 PTX 指令上插桩.

SASS 是英伟达 GPU 的原生机器指令集, 是 GPU 硬件直接执行的底层机器码的汇编形式. PTX 是在 SASS 的基础上抽象了一层, 并且 PTX 指令集是完全开放的,

能够在多代英伟达 GPU 之间保持兼容. 两者的关系体现在 CUDA 程序的编译过程中, 如图 7 所示[50,51].

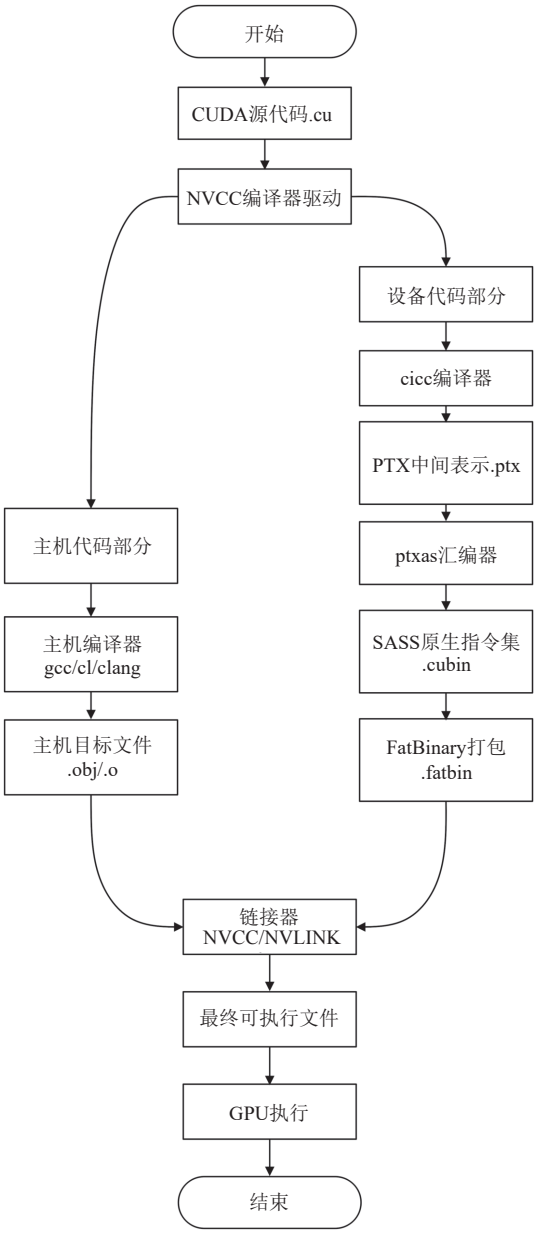

图 7 CUDA 程序编译流程图

在公开的学术研究领域, Neutrino[52]工具取得了关键的进展, Neutrino 选择在更高层次的 PTX 中间表示层进行插桩, 在保持强大可编程能力的同时, 显著提升了工具的可访问性、可移植性和易用性. 如图 8 展示了 Neutrino 简化的工作流程: 用户输入层包含用户编写的探针代码以及指定的目标程序, 先将探针代码编译成 PTX 汇编代码, 并且目标程序运行时, Hook 驱动捕获 GPU 二进制文件. 随后探针引擎从二进制文件中提取 PTX 汇编, 在指定的跟踪点中插入探针代码, 并且添加结构化映射存储数据; 最后, 执行插桩后的内核, 通过回调函数生成性能数据或可视化图.

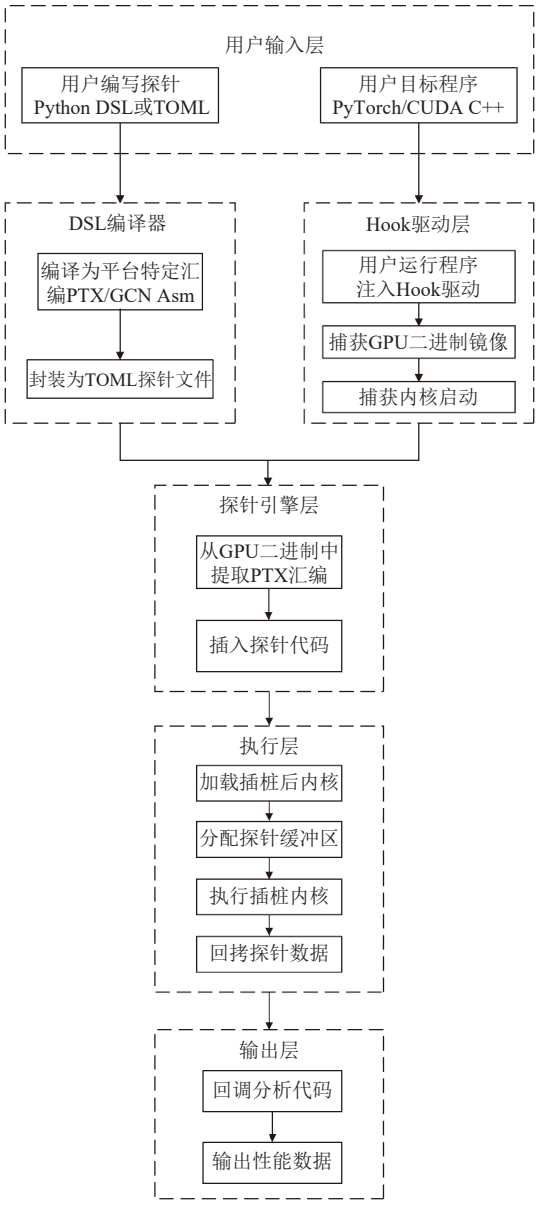

图 8 Neutrino 工作流程简化图

Neutrino 在 GPU 内核的汇编指令级动态插入探针, 实现细粒度、可编程的性能分析. 其探针设计包含 3 个核心要素: 代码片段、结构化映射和独立寄存器组. 探针可插入指令前后、内核入口/出口、分支点及内存访问指令等关键位置, 从而捕获指令级时间戳、寄存器值、内存地址及线程束调度信息. 因此, Neutrino 在保持低开销的同时也具备强大的细粒度分析能力. 对于轻量级探针, 内核执行时间开销仅为 1.04 倍, 额外

寄存器使用平均约 4 个; 即使对于重探针, 开销也控制在可接受范围内. 通过对比 Nsight Compute 等硬件计数器工具, Neutrino 能够捕获前者无法提供的指令级调度信息.

此外, GPU 内存分析工具 CUTHERMO[53]直接基于 GPU 二进制代码展开分析, 无需对硬件、操作系统或应用程序源码进行修改. 该工具通过生成热力图以直观展现内存访问场景, 从而方便识别内存低效问题并提供性能优化策略.

4 各类 GPU 性能分析方法优劣比较

基于硬件性能计数器的方法、基于微基准测试的方法、基于软件插桩的方法、基于模拟与建模的方法和基于二进制插桩的方法从不同维度进行性能分析, 它们在技术原理、适用场景等方面各具特点. 这 5 类方法主要优势与潜在局限总结如下.

(1) 基于硬件性能计数器的优势在于提取的数据精准且开销低, 反映了硬件层的客观指标. 缺陷在于 GPU 的黑盒特性, 提取的数据固定, 不能够自行编程.

(2) 基于微基准测试的优势在于主动揭示硬件的位置参数, 结果可重现, 便于逆向推导硬件特性. 缺陷是测试环境与复杂的应用行为差距大, 且手工测试工作量大, 需要随着硬件迭代而不断更新.

(3) 基于软件插桩的优势在于能够在系统级上展示 CPU-GPU 的并发与依赖, 语义关联强, 便于理解. 缺陷是开销较高, 细粒度观测有限, 有可能扰动程序行为.

(4) 基于模拟与建模的优势在于整个过程完全可见与可控, 支持未来架构探索, 能够探索理论性能上界. 缺陷是模拟器运行速度慢, 且建模的精度受限于对未公开硬件细节的假设, 验证难度高.

(5) 基于二进制方法的优势在于可以编程, 并且满足细粒度的追踪, 灵活度强. 缺陷是难度高, 当前研究尚不成熟, 开销与扰动也大.

当前 GPU 性能分析逐步形成多层次、多目标的方法体系, 每种方法在细粒度、开销和灵活性上都有权衡. 当前的研究无法均衡这 3 种维度, 需要对多种方法进行深度融合[54], 向智能化、全栈化和可编程的方向前进, 打造更加全面的性能分析方法.

5 未来研究方向展望

2025 年黄仁勋[55]在 GPU 技术大会演讲表明当前 AI 发展迅猛, 进入了“代理 AI”和“物理 AI”阶段, 未来的性能分析工具需要更加强大和完善的功能, 理解并追踪复杂、动态的工作流. 同时, 推理需求增加, 计算量预计增大“100 倍”, 亟须发展低开销、无干扰的性能采样技术, 且能够应对百倍级计算规模. 新一代的新型计算单元和硅光子学互连等新硬件的演进, 具有全新的性能特性和瓶颈, 因此, 性能分析必须紧跟硬件迭代. 未来对 GPU 性能分析方法的研究仍面临的 3 大挑战.

其一, 性能分析的深度与开销之间的矛盾. 细粒度的方法能够提供具体的性能数据, 但是需要极大的开销去满足, 无法部署到大规模的生产环境中; 而低开销的方法无法观察到细粒度的性能数据. 这一矛盾在 GPU 性能分析领域已有实证研究: 2025 年 Lahiry 等人[56]指出单一数据轨迹的庞大规模和复杂性使得 GPU 性能分析不仅计算成本高而且非常耗时. 因此, 面对当下成百倍的 AI 计算规模, 相关矛盾会被放大.

其二, 各类方法之间缺乏协同. 本文提出的 5 类方法具有较强的互补性, 但目前仍各自独立, 缺少有效的整合机制. 近年来已有研究开始探索这 5 类方法之间的协同, 其中常见的方式是将基于硬件性能计数器的方法与其他方法进行结合. 譬如, 综合运用随机森林建模和硬件性能计数器数据开展基于 GPU 系统的应用性能的评估[57]. 然而, 相关工作往往局限于特定场景或特定方法的局部协同, 尚未形成多种方法集成的统一框架, 相关研究有待进一步深入.

其三, 对获取的数据分析不到位. 现有工具仅能输出性能指标, 缺乏自动化的性能瓶颈定位与优化建议生成能力. Filipovič等人[58]提出了一种基于硬件性能计数器的自动调优方法, 通过收集 GPU 硬件计数器事件, 以加快应用移植到不同硬件或处理不同特性数据的自动调优. Opal[59]将硬件性能计数器与屋顶线分析相结合, 进而利用大模型尝试生成 GPU 性能优化建议. 但是相关方法或者局限于特定场景或者分析能力受限于大模型对 GPU 架构的理解程度, 对通用场景和复杂瓶颈的自动诊断仍具有较大的提升空间.

针对上述挑战, 未来研究可以从以下 3 个方向进行优化与突破.

(1) 可以通过构建分层自适应性能分析框架, 从粗到细进行协同分析, 实现精度与开销的动态平衡. 这个框架可以由 3 个层级构成: 第 1 层是系统级快速筛查, 借鉴 SysOM-AI[60]的跨层观测理念, 通过使用轻量级软

件插桩, 以较低开销捕获 CPU-GPU 活动时间线, 采用规则或轻量级机器学习模型自动识别瓶颈类型. 第 2 层是内核级定向分析, 根据第 1 层输出的标签, 自动调用对应的深度分析工具. 第 3 层是指令级可变成验证, 当第 2 层无法定位原因时, 启用二进制插桩框架, 完成原因分析.

(2) 可以开发语义感知的智能化性能分析技术. 将底层性能事件自动映射回高层任务语义, 并且在此基础上构建自动定位与优化建议模型. 具体而言, 借鉴 Opal 的思路, 将大模型与硬件计数器结合, 搭建智能化的性能分析框架. 同时, 进一步探索大模型与性能模型的深度融合, 提升大模型对 GPU 架构的理解, 突破当前研究对复杂瓶颈分析不到位的问题.

(3) 可以设计自动化的新硬件适配技术与跨架构性能建模方法. 通过自动分析新硬件指令集, 进而自动生成覆盖关键性能的微基准测试套件. 在此基础上, 利用迁移学习技术将已有的性能模型快速适配至新的架构, 以减少从头开始训练模型所需的大量数据和时间. 同时, 可借鉴 LEO[61]的跨厂商分析思路, 建立支持不同厂商的 GPU 性能瓶颈统一诊断模型, 实现分析工具与硬件平台的解耦.

6 总结

本文系统梳理了现有研究成果, 建立了 GPU 性能分析方法分类体系, 即基于硬件性能计数器的方法、基于微基准测试的方法、基于软件插桩的方法、基于模拟与建模的方法以及基于二进制插桩的方法. 在此框架下揭示相关核心原理与代表性工具, 通过对比有关方法技术的优势与不足, 指出了 GPU 性能分析从单一方法向多维度融合与智能化演进的必要性, 以应对 AI 的飞速发展和超级计算要求. 进一步说, 未来该领域将聚焦于低开销、智能化与可编程的研究方向, 从而为下一代异构计算系统提供更多的支撑.

参考文献

1 Khailany B, Dally WJ, Kapasi UJ, *et al*. Imagine: Media processing with streams. IEEE Micro, 2001, 21(2): 35–46. [doi: 10.1109/40.918001]

2 Buck I, Foley T, Horn D, *et al*. Brook for GPUs: Stream computing on graphics hardware. ACM Transactions on Graphics, 2004, 23(3): 777–786. [doi: 10.1145/1015706.1015800]

3 NVIDIA. Scalable parallel programming with CUDA. Proceedings of the 2008 IEEE Hot Chips 20 Symposium. California: IEEE, 2008. 1–2. [doi: 10.1109/HOTCHIPS.2008.7476516]

4 Mims C. Huang's law is the new Moore's law, and explains why NVIDIA wants arm. The Wall Street Journal. https://www.wsj.com/articles/huangs-law-is-the-new-moores-law-and-explains-why-nvidia-wants-arm-11600488001. (2020-09-19) [2026-04-21].

5 Lopez-Novoa U, Mendiburu A, Miguel-Alonso J. A survey of performance modeling and simulation techniques for accelerator-based computing. IEEE Transactions on Parallel and Distributed Systems, 2015, 26(1): 272–281. [doi: 10.1109/TPDS.2014.2308216]

6 赵海燕, 李志凯, 钱诗友, 等. 面向分布式集群的 GPU 性能分析与建模方法: 现状及展望. 小型微型计算机系统, 2026, 47(1): 58–72. [doi: 10.20009/j.cnki.21-1106/TP.2025-0278]

7 Markidis S, Chien SWD, Laure E, *et al*. NVIDIA tensor core programmability, performance & precision. Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium Workshops. Vancouver: IEEE, 2018. 522–531. [doi: 10.1109/IPDPSW.2018.00091]

8 Lindholm E, Nickolls J, Oberman S, *et al*. NVIDIA Tesla: A unified graphics and computing architecture. IEEE Micro, 2008, 28(2): 39–55. [doi: 10.1109/MM.2008.31]

9 Williams S, Waterman A, Patterson D. Roofline: An insightful visual performance model for multicore architectures. Communications of the ACM, 2009, 52(4): 65–76. [doi: 10.1145/1498765.1498785]

10 Wang YS, Yang C, Farrell S, *et al*. Time-based roofline for deep learning performance analysis. Proceedings of the 2020 IEEE/ACM Fourth Workshop on Deep Learning on Supercomputers. Atlanta: IEEE, 2020. 10–19. [doi: 10.1109/DLS51937.2020.00007]

11 Ding N, Williams S. An instruction roofline model for GPUs. Proceedings of the 2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems. Denver: IEEE, 2019. 7–18. [doi: 10.1109/PMBS49563.2019.00007]

12 Yang C, Wang YS, Farrell S, *et al*. Hierarchical roofline performance analysis for deep learning applications. arXiv:2009.05257, 2020.

13 Gholami A, Yao ZW, Kim S, *et al*. AI and memory wall. IEEE Micro, 2024, 44(3): 33–39. [doi: 10.1109/MM.2024.

3373763]

14 Awan AA, Hamidouche K, Hashmi JM, *et al*. S-caffe: Co-designing MPI runtimes and caffe for scalable deep learning on modern GPU clusters. ACM SIGPLAN Notices, 2017, 52(8): 193–205. [doi: 10.1145/3155284.3018769]

15 Shama E, Grant RE. NAV: A comparative analysis tool for nsight systems GPU traces. Proceedings of the 33rd Euromicro International Conference on Parallel, Distributed, and Network-based Processing. Turin: IEEE, 2025. 536–543. [doi: 10.1109/PDP66500.2025.00082]

16 van Lanker L, Taboada H, Brunet E, *et al*. Predicting GPU kernel's performance on upcoming architectures. Proceedings of the 30th European Conference on Parallel and Distributed Processing. Madrid: Springer, 2024. 77–90.

17 Iliakis K, Xydis S, Soudris D. Repurposing GPU microarchitectures with light-weight out-of-order execution. IEEE Transactions on Parallel and Distributed Systems, 2022, 33(2): 388–402. [doi: 10.1109/TPDS.2021.3093231]

18 程克非, 张聪, 汪林林, 等. 基于硬件性能计数器的软件性能数据采集与分析研究. 计算机应用, 2005, 25(10): 2431–2433.

19 Kurzynski M, Aga S, Wu D. Chopper: A multi-level GPU characterization tool & derived insights into LLM training inefficiency. arXiv:2512.08242, 2025.

20 Graham SL, Kessler PB, McKusick MK. gprof: A call graph execution profiler. ACM SIGPLAN Notices, 2004, 39(4): 49–57. [doi: 10.1145/989393.989401]

21 Corbet J, Rubini A, Kroah-Hartman G. Linux Device Drivers. Sebastopol: O'Reilly Media, 2005.

22 NVIDIA. NVIDIA nsight compute documentation. https://docs.nvidia.com/nsight-compute/. [2026-04-21].

23 NVIDIA Corporation. CUPTI: CUDA profiling tools interface. https://docs.nvidia.com/cupti/. [2026-04-21].

24 薛子涵, 解达, 宋威. 基于 RISC-V 的新型硬件性能计数器. 计算机系统应用, 2021, 30(11): 3–10. [doi: 10.15888/j.cnki.csa.008346]

25 Wong H, Papadopoulou MM, Sadooghi-Alvandi M, *et al*. Demystifying GPU microarchitecture through microbenchmarking. Proceedings of the 2010 IEEE International Symposium on Performance Analysis of Systems & Software. White Plains: IEEE, 2010. 235–246. [doi: 10.1109/ISPASS.2010.5452013]

26 Jia Z, Maggioni M, Smith J, *et al*. Dissecting the NVIDIA Turing T4 GPU via microbenchmarking. arXiv:1903.07486, 2019.

27 Grambow M, Kovalev D, Laaber C, *et al*. Using microbenchmark suites to detect application performance changes. IEEE Transactions on Cloud Computing, 2023, 11(3): 2575–2590. [doi: 10.1109/TCC.2022.3217947]

28 Japke N, Grambow M, Laaber C, *et al*. µOpTime: Statically reducing the execution time of microbenchmark suites using stability metrics. ACM Transactions on Software Engineering and Methodology, 2025, 34(8): 231. [doi: 10.1145/3715322]

29 Luo WL, Fan RB, Li ZY, *et al*. Dissecting the NVIDIA hopper architecture through microbenchmarking and multiple level analysis. arXiv:2501.12084, 2025.

30 Rodrigues CF, Riley G, Luján M. Fine-grained energy profiling for deep convolutional neural networks on the Jetson TX1. Proceedings of the 2017 IEEE International Symposium on Workload Characterization. Seattle: IEEE, 2017. 114–115.

31 Raikundalia S. PyTorch Profiler. https://pytorch.org/tutorials/recipes/recipes/profiler_recipe.html. (2021-01-29)[2026-04-21].

32 TensorFlow Team. Optimize TensorFlow performance using the Profiler. https://www.tensorflow.org/guide/profiler. [2026-04-21].

33 Zhong YH, Li HY, Wu YJ, *et al*. XRP: In-kernel storage functions with eBPF. Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation. Carlsbad: USENIX Association, 2022: 375–393.

34 Zheng YS, YU T, YANG YW, *et al*. gpu_ext: Extensible OS policies for GPUs via eBPF. arXiv:2512.12615, 2025.

35 Chu KX, Su JC, Zhang YF, *et al*. Einfer: Unlocking fine-grained tracing for distributed LLM inference with eBPF. Proceedings of the 3rd Workshop on eBPF and Kernel Extensions. Coimbra: ACM, 2025. 76–83. [doi: 10.1145/3748355.3748372]

36 NVIDIA. NVIDIA Nsight Systems user guide. https://docs.nvidia.com/nsight-systems/. [2026-04-21].

37 Guan Y, Fang YW, Zhou KR, *et al*. KPerfIR: Towards an open and compiler-centric ecosystem for GPU kernel performance tooling on modern AI workloads. arXiv:2505.21661, 2025.

38 NVIDIA Corporation. Parallel thread execution ISA version 8.4. https://docs.nvidia.com/cuda/archive/12.4.0/parallel-thread-execution/index.html. (2024-03-02)[2026-04-21].

39 Bakhoda A, Yuan GL, Fung WWL, *et al*. Analyzing CUDA workloads using a detailed GPU simulator. Proceedings of the 2009 IEEE International Symposium on Performance

Analysis of Systems and Software. Boston: IEEE, 2009. 163–174. [doi: 10.1109/ISPASS.2009.4919648]

40 Khairy M, Shen ZS, Aamodt TM, *et al*. Accel-Sim: An extensible simulation framework for validated GPU modeling. Proceedings of the 47th ACM/IEEE Annual International Symposium on Computer Architecture. Valencia: IEEE, 2020. 473–486. [doi: 10.1109/ISCA45697.2020.00047]

41 Beckmann B, Gutierrez A. The AMD gem5 APU simulator: Modeling heterogeneous systems in gem5. Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture., 2015.

42 Sun YF, Baruah T, Mojumder SA, *et al*. MGPUSim: Enabling multi-GPU performance modeling and optimization. Proceedings of the 46th ACM/IEEE Annual International Symposium on Computer Architecture. Phoenix: IEEE, 2019. 197–209.

43 Kim H, Lee J, Lakshminarayana N B, *et al*. MacSim: A CPU-GPU heterogeneous simulation framework user guide. Georgia Institute of Technology, 2012: 1–57.

44 Gong X, Ubal R, Kaeli D. Multi2Sim Kepler: A detailed architectural GPU simulator. Proceedings of the 2017 IEEE International Symposium on Performance Analysis of Systems and Software. Santa Rosa: IEEE, 2017. 269–278.

45 Ubal R, Jang B, Mistry P, *et al*. Multi2Sim: A simulation framework for CPU-GPU computing. Proceedings of the 21st International Conference on Parallel Architectures and Compilation Techniques. Minneapolis: IEEE, 2012. 335–344.

46 Villa O, Stephenson M, Nellans D, *et al*. NVBit: A dynamic binary instrumentation framework for NVIDIA GPUs. Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture. Columbus: ACM, 2019. 372–383. [doi: 10.1145/3352460.3358307]

47 Kothapalli K, Mukherjee R, Rehman MS, *et al*. A performance prediction model for the CUDA GPGPU platform. Proceedings of the 2009 International Conference on High Performance Computing. Kochi: IEEE, 2009. 463–472.

48 Baghsorkhi SS, Delahaye M, Patel SJ, *et al*. An adaptive performance modeling tool for GPU architectures. ACM SIGPLAN Notices, 2010, 45(5): 105–114. [doi: 10.1145/1837853.1693470]

49 Gao YJ, Gu XY, Zhang HY, *et al*. Runtime performance prediction for deep learning models with graph neural network. Proceedings of the 45th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice. Melbourne: IEEE, 2023. 368–380. [doi: 10.1109/ICSE-SEIP58684.2023.00039]

50 NVIDIA Corporation. CUDA Compiler Driver NVCC Reference Guide. Santa Clara: NVIDIA, 2022.

51 Lee S, Min SJ, Eigenmann R. OpenMP to GPGPU: A compiler framework for automatic translation and optimization. ACM SIGPLAN Notices, 2009, 44(4): 101–110. [doi: 10.1145/1594835.1504194]

52 Huang SL, Wu CS. Neutrino: Fine-grained GPU kernel profiling via programmable probing. Proceedings of the 19th USENIX Symposium on Operating Systems Design and Implementation. Boston: USENIX Association, 2025. 331–355.

53 Zhao YB, Cui JK, Li ZC, *et al*. CUTHERMO: Understanding GPU memory inefficiencies with heat map profiling. arXiv:2507.18729, 2025.

54 Zhang KX, Cui YF, Zhang SH, *et al*. SynPerf: A hybrid analytical-ML framework for GPU performance prediction. arXiv:2601.14910, 2026.

55 Huang J. GTC 2025 Keynote. https://www.nvidia.com/en-us/on-demand/session/gtc25-s72484/. [2026-04-21].

56 Lahiry A, Pokharel A, Banday B, *et al*. A distributed framework for causal modeling of performance variability in GPU traces. arXiv:2510.18300, 2025.

57 Madougou S, Varbanescu AL, de Laat C, *et al*. A tool for bottleneck analysis and performance prediction for GPU-accelerated applications. Proceedings of the 2016 IEEE International Parallel and Distributed Processing Symposium Workshops. Chicago: IEEE, 2016. 641–652.

58 Filipovič J, Hozzová J, Nezarat A, *et al*. Using hardware performance counters to speed up autotuning convergence on GPUs. Journal of Parallel and Distributed Computing, 2022, 160: 16–35. [doi: 10.1016/j.jpdc.2021.10.003]

59 Zaeed M, Islam TZ, Inđić V. Opal: A modular framework for optimizing performance using analytics and LLMs. arXiv:2510.00932, 2025.

60 Zheng YS, Mao WA, Cheng SY, *et al*. SysOM-AI: Continuous cross-layer performance diagnosis for production AI training. arXiv:2603.29235, 2026.

61 Xia YN, Mellor-Crummey J. LEO: Tracing GPU stall root causes via cross-vendor backward slicing. arXiv:2604.20032, 2026.

(校对责编: 李慧鑫)