将体系结构模拟扩展至每秒十亿条指令

Zhouyi Shen HaoTang Ruike Liu中国科学技术大学 中国科学技术大学 中国科学技术大学中国 中国 中国中国合肥 中国合肥 中国合肥squirrel@mail.ustc.edu.cn
th20030818@mail.ustc.edu.cnliuruike@mail.ustc.edu.cn

Jingwei Sun中国科学技术大学 中国合肥sunjw@ustc.edu.cn

Guangzhong Sun中国科学技术大学 中国合肥
gzsun@ustc.edu.cn

arXiv:2611.32397v1 [cs.AR] 24 Nov 2026

摘要

详细体系结构模拟的成本限制了设计空间探索，而在模拟核心之间进行并行化，对于单核心和小规模多核目标只能提供有限的主机并发。我们提出了 GIPSim，这是一种指令驱动模拟器，可根据依赖关系和资源约束推导指令时序，从而减少每个模拟周期更新流水线状态的开销。该模拟器对记录的执行过程采用同步引导的时间分解，在目标核心数之外进一步发掘主机并行性。GIPSim 并行重放由此产生的片段，并通过依赖图汇总其持续时间，以估算总体执行时间；同时利用保留的微体系结构状态、预热和同步引导的缓存驱逐来重建重放上下文。这样无需主机工作线程遵循目标执行顺序，即可计入同步等待。我们使用多核 x86-64、AArch64 和 RV64GC 系统上应用程序及操作系统执行的 QEMU跟踪记录，对 GIPSim 进行了评估。GIPSim 在单节点上的平均模拟吞吐量为：使用一个工作线程时达到每秒 1,380 万条指令（MIPS），使用 96 个工作线程时达到 1,010 MIPS，峰值为 1,421 MIPS。相对于 gem5，执行周期对应的平均绝对百分比误差分别为 7.52% 和 7.43%。

关键词

体系结构模拟、跟踪重放、乱序执行、并行仿真、同步

1 引言

体系结构模拟为处理器设计在实现前提供了定量依据。广泛使用的 gem5 [6]模拟器将周期驱动的详细CPU模型与离散事件框架相结合，由于反复进行状态更新和事件处理而产生了显著开销。已发表的测量结果 [12] 显示，gem5时序模式吞吐量为每秒0.01–0.1百万条指令（每秒百万条指令），约比本机硬件执行速率低五到六个数量级。这种较低的仿真性能延迟了对单个设计选择的反馈，并限制了可评估的工作负载配置对数量，从而限制了体系结构设计空间探索的规模。

为缓解这一瓶颈，已有研究沿着三个互补方向展开：提高时序效率 [13, 44, 51],将仿真工作分配到多个主机工作线程 [9, 12, 30],，以及对执行区域进行采样，以减少详细模拟的工作量 [42, 47, 50]。本文重点关注时序效率和并行仿真。采样与这些工作互为补充：它决定哪些区域需要进行详细模拟，而时序模型和并行仿真则决定如何评估这些区域。

指令驱动仿真通过依据依赖关系和资源约束推导指令时序，而非在每个周期反复评估流水线状态，从而提高时序效率 [44, 51]。它能够以较低的状态维护开销模拟乱序执行和资源竞争，但其准确性取决于对调度和内存交互的建模是否忠实。简化的发射顺序可能改变指令争用资源的方式，而不完整的转换和内存时序则可能扭曲停顿情况，以及计算与内存访问之间的重叠。

并行仿真通过将建模工作分配给多个主机工作线程，弥补时序效率提升的不足。空间分解 将目标核心或组件分配给不同的工作线程，并在执行过程中保留各组件的状态。Graphite、Sniper 和 ZSim 利用核心级并行 [9, 30, 44],，而 parti-gem5 则并行处理 gem5 的详细 CPU 和内存模型 [12]。共享资源交互要求工作线程协调模拟时间上的执行进度。核心级分区也将并发度限制在目标核心数，使得在模拟单核或小规模多核目标时，大型主机的大部分资源闲置。并发运行多个仿真实例可以提高总体吞吐量，但不会缩短单次评估的时间。

时间分解 揭示了沿着执行时间线。屏障区间时间并行 [7]和 ChunkedTe- jas [23] 构建跨越参与核心的粗粒度区间，并联合仿真每个区间内的组件。独立重放每核心跟踪片段，可以揭示执行位置和目标核心上的并行工作，但也带来了两个相互关联的挑战。首先，分解过程必须确定合适的边界，并表示片段的相对位置，以便正确合并其持续时间，从而预测总执行时间。其次，独立重放需要

表1：仿真工作分解和时序组织的代表性方法。多核目标支持：多核目标支持；空间分解/时间分解：空间/时间分解。

相关工作	多核目标支持	空间分解	时间分解	时序组织
gem5 [6]	✓	✗	✗	顺序事件处理
Graphite [30]; Sniper [9]; ZSim [44]; Prophet [51]; parti-gem5 [12]; SplitSim [24]	✓	✓	✗	协调组件 推进
LEGOSim [28]	✓	✓	✗	分阶段组件重放
Nguyen等人 [33]; DiST [14]	✗	✗	✓	独立区间重放 + 时序聚合
Pandey 等人 [36]	✗	✗	✓	独立子轨迹重放 + 校正
屏障区间时间并行 [7]	✓	✗	✓	联合多核仿真在时间间隔内
ChunkedTejas [23]	✓	✓	✓	联合多核仿真在时间间隔内
GIPSim	✓	✓	✓	**独立的逐核重放 + 基于有向无环图的时间重构**

为每个片段提供合适的仿真上下文，包括对其入口状态的近似，以及对执行期间其他核心造成的资源争用的估算。

我们提出了 GIPSim，这是一款基于跟踪的性能模拟器，它将高效的指令驱动时序与同步引导的时间分解 相结合，以实现细粒度独立重放。为实现高效且准确的指令时序，其引擎会在资源约束下显式地以乱序方式调度就绪操作，并使用时间戳将指令进度与地址转换和内存服务关联起来。这样无需反复进行逐周期流水线更新，即可捕获指令调度和内存引发的停顿。

GIPSim 通过将主机重放顺序与目标执行顺序解耦，同时在时序预测中保留执行依赖关系，实现独立并行重放。两种互补设计支持这种解耦：(1) 为了重建整体执行时间，GIPSim 在同步点处分割每个核心的跟踪，并在有向无环图（DAG）中表示各片段及其依赖关系。关键路径分析会汇总片段时长，并计入同步等待。(2) 为了提供重放上下文，GIPSim 利用保留的微体系结构状态和基于历史的预热来近似入口状态。同步引导的缓存驱逐还可近似跨核写入的缓存效应，而解析模型则用于估算活跃核心之间额外的主存争用。因此，工作线程无需前序时序结果，也无需持续进行跨组件协调，即可独立重放。核内片段的独立重放还支持空间分解，如表 1 所示。

我们使用由 QEMU 生成的 SPEC CPU、PARSEC 和 NAS 并行基准测试（NPB）跟踪来评估 GIPSim。目标系统包括单核和八核的 x86、ARM 及 RISC-V 配置。在 48 个工作负载—硬件组合中，单个主机节点上的平均吞吐量从使用 1 个工作线程时的每秒 13.8 百万条指令提升至使用 96 个工作线程时的每秒 1,010 百万条指令。单核目标实现了 68.1–76.0× 的加速比，直接展现了超越目标核心数的并行能力。相对于 gem5，预测周期数的平均绝对百分比误差（MAPE）在使用 1 个工作线程时为 7.52%，使用 96 个工作线程时为 7.43%。吞吐量测量包括初始化、重放和后处理，但不包括跟踪收集。

本文有三项贡献：

- 一种指令驱动的时序模型，通过时间戳显式调度乱序操作，并将核心执行与地址转换和内存服务关联起来，从而保留关键的时序交互，同时无需反复逐周期更新流水线。
- 一种同步引导的时间分解方法，可实现相互独立的并行重放和基于依赖关系的执行时间重构，从而揭示超出目标核心数的主机并发。
- 一种多指令集架构实现及其评估，量化了重放并行性和状态重构如何影响不同工作负载和处理器配置下的预测准确性与吞吐量。

2 相关工作

2.1 体系结构模拟

体系结构模拟对处理器和内存行为进行建模，以便在实现前开展设计探索。功能模拟器重现指令对体系结构状态的影响。QEMU [4]使用动态二进制翻译实现接近原生速度的功能模拟，并作为我们的跟踪前端。性能模拟器还会对微体系结构时序进行建模，以预测执行时间。广泛采用的 gem5 [6, 29] 提供了详细的处理器和内存模型。其能力和预测准确性已得到广泛研究与验证 [1, 20, 46], ，因此我们将其作为准确性参照。

在执行驱动仿真中， [6, 11, 44], 功能执行与时序同步进行，使时序能够影响执行进度和交错执行。在轨迹驱动仿真中，[22, 40], 一个独立的时序后端根据记录的执行过程运行，不向功能前端提供反馈。GIPSim 采用这种设计，是因为预先生成的轨迹简化了在主机工作线程之间分配重放工作的过程，并支持在无需重复功能执行的情况下进行多次时序评估。

采样减少了需要进行详细模拟的工作量。BarrierPoint [8], TaskPoint [17], 和 LoopPoint [42, 43] 利用程序结构选择区域，而 Chronos [26] 则对整个系统进行基于时间的采样。这些方法需要覆盖原始执行过程中的代表性部分；GIPSim 可加速记录的执行过程中的重放，并能够补充此类工作量削减方法。

2.2 核心时序模型

核心时序模型在推进执行和实施流水线约束的方式上有所不同。基于周期驱动的模型（如 gem5 的

乱序 CPU [6]模型会在每个周期显式推进流水线状态，从而呈现出细致的交互，但评估成本也相当高。Sniper 的区间仿真 [9, 13] 则通过分析估算指令重叠和失效惩罚来降低这一成本，使部分操作与资源之间的交互保持隐式 [10]。

指令驱动型 模型（如 ZSim [44] 和 Prophet [51]）无需反复扫描周期，即可评估指令和资源约束。Calipers [15] 通过指令事件图和关键路径分析，提供了相关的基于约束的表述。指令级评估以相对较低的成本保留依赖关系和资源约束，但其保真度仍取决于所表示的调度和完成关系。GIPSim 在此基础上，明确关联由就绪状态驱动的发射、资源占用和完成反馈，以解决 §3.1 中探讨的差异。

机器学习 模型提供了另一种降低时序评估成本的方法。SimNet [25] 和 TAO [37] 预测指令时序，而 Concorde [32] 则根据分析组件的估算结果预测区域 CPI。它们的保真度取决于训练覆盖范围和泛化能力。GIPSim 保留了用于指令调度和核心与内存交互的显式时序规则，而非通过学习来推断这些因素的综合影响。

2.3 并行仿真

空间分解将目标核心或组件映射到主机工作线程，如 Graphite [30], ZSim [44], parti-gem5 [12], 和 Split- Sim [24] 所采用的方法。协调式进度可维持组件间时序交互，但会产生通信和同步成本。LEGOSim [28] 中的分阶段协调以及 FCS [27] 中的因果前沿同步可降低这些成本；可用并发度仍受目标硬件结构限制，从而限制了小型目标系统的主机利用率。

时间分解揭示了单次执行中不同位置的工作。早期并行轨迹处理 [33], DiST [14], 以及 Pandey 等人 [36] 进行的 GPU 子轨迹模拟，利用了单核流中的这一机会。对于并行程序，屏障区间时间并行 [7]和 ChunkedTejas [23]分别利用全局屏障以及参考进度或稳定阶段预测来构建系统级区间。这些方法需要

响应线程间的分区边界，从而限制了分区的灵活性。在每个时间间隔内，详细模拟仍在各核心之间耦合进行，以解决它们之间的交互。与 Chun- kedTejas 在时间间隔内联合推进多核的方式不同，GIPSim 将独立的核本地重放与基于依赖的时间重构相结合。

通过独立区域重放来展现并行性，需要同时捕获依赖关系并组合时序。MUSA [16]并行评估 MPI 进程计算阶段，并通过通信重放合并各阶段的时长，同时保留每个阶段内部的多核仿真。对于同步捕获，SynchroTrace [45] 和 ElasticSimMATE [34] 会对同步 API 或运行时进行插桩，而 HETSIM [35] 则使用明确指定的原语。这些机制将显式同步识别与受支持的软件接口绑定起来。GIPSim 使用从指令级释放—获取交接中恢复的依赖关系，

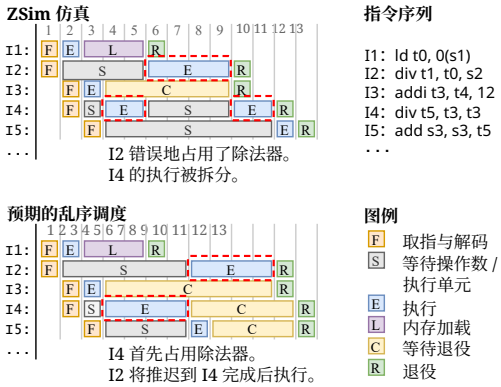


图1：ZSim 的指令驱动模型中，简化调度造成的时序失真示例。

从而减少对特定于库的跟踪的依赖，并通过有向无环图汇总独立计算的片段时长。这支持对核内区域进行细粒度并行重放，无需全局对齐边界或联合推进多核，同时对入口状态和跨核干扰进行近似处理。

更细的区域会使入口状态重建更加频繁。Perelman 等人 [38] 保留过时状态，而 Haskins 和 Skadron [18]将状态保留与近期历史预热相结合，在较短的预热窗口之外保留可能有用的状态。MTR [3] 和 TraceRTL [52] 分别通过压缩内存历史和预热去重减少重建工作。GIPSim 将状态保留和预热与同步引导的缓存驱逐相结合，无需前驱重放结果即可近似局部历史和跨核缓存效应。

3 动机

3.1 精确的指令驱动时序

指令驱动模型避免了重复的流水线更新，但简化的调度可能改变操作争夺有限资源的方式。当较年轻的操作已就绪，而较老的操作仍在等待操作数时，跟踪顺序更新可能会让前者在目标时间中相关影响实际发生之前，就受到后者的影响。

图 1 说明了这个问题。在周期 4，I2 正在等待 I1的结果，而 I4 已准备好发射。由于 ZSim [44] 按程序顺序处理指令，因此在调度 I4之前，会在周期 6–9 为 I2预留除法器。这项预留将 I4 的模拟执行分成两个时间段，扭曲了写回和退役时序，并影响其他指令。在目标乱序执行中，I4 会先占用除法器，而 I2 则会一直等待，直到 I4 完成。

调度与内存服务相互作用，因为完成事件决定操作数就绪并释放有限资源，而长延迟响应可能延迟退役和指令接纳。反过来，调度也决定请求到达时间。地址转换延迟和页表流量也会参与这一反馈，因此它们的影响不止于单次访问的延迟。

为评估这些时序简化所带来的预测影响，我们提取 ZSim 的核心和内存时序模型 [44]并进行重放

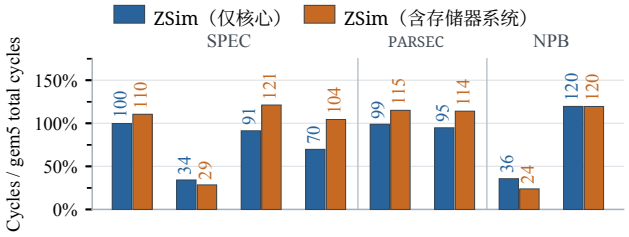


图 2：固定延迟内存和原生内存系统下 ZSim 的周期数，分别以每种设置下对应的 gem5 参考值进行归一化。

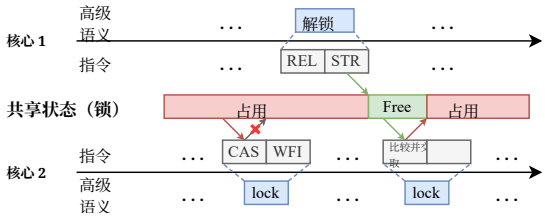


图 3：通过指令级释放–获取交接实现的高级锁同步。

从 gem5 [6] 记录的指令踪迹，以便在每次比较中固定执行路径。受控实验采用四发射 x86 配置，并测量 10^6 条指令（预热 10^5 条指令后）。图 2 报告了固定延迟和原生内存实验的结果，并分别以对应的 gem5 参考结果进行归一化。固定延迟内存抑制了详细的内存层次结构影响，从而揭示核心时序差异：对于 `503.bwaves_r`，ZSim 仅预测出 gem5 周期数的 34.2%；对于 `FT`，则为 35.7%。这些偏差表明，有必要在高效的指令驱动模型中更忠实地执行指令就绪状态和资源约束。

原生内存实验还揭示了内存建模的影响及其与核心执行的交互。对于 605.mcf_s 和 blackscholes，ZSim 预测的周期数分别为相应原生内存 gem5 参考值的 121.2% 和 114.1%。固定延迟内存下的结果则分别为独立固定延迟参考值的 91.3% 和 94.8%。这些不同的误差模式进一步表明，除了单独改进核心模型之外，还需要对内存服务及其对指令调度的反馈进行建模。这些观察促使 GIPSim 将显式的就绪驱动乱序发射与轻量级地址转换和内存层次结构模型相结合，并通过请求与完成时序将它们耦合起来，从而在保留指令驱动效率的同时维持这些交互。

3.2 同步引导的跟踪分解

时间分解能够揭示更多仿真并发性，但现有的粗粒度方法 [7, 23] 在每个时间区间内仍要求跨核心联合重放，因此需要特定的程序结构，或预先了解线程间的相对进度。对更细粒度的每核心区域进行独立重放，可以揭示更多并行工作。然而，仅仅划分每核心跟踪（例如按指令计数划分）仍有两个问题未解决。首先，这些切分点可能会遗漏跨核心依赖端点，无法保留其优先关系。其次，这种划分方式无法确定跨核心的段重叠，因此无法忠实地建模跨核干扰，尤其是取决于

基于精确的访问交错执行。

同步为恢复这些关系提供了结构。并行程序使用锁和屏障来建立正确执行所需的排序和可见性保证。在更底层，这些机制由通过释放-获取交接实现的共享数据通信构成：生产者通过释放操作发布更新，而消费者在使用这些更新之前完成相应的获取操作 [19, 49]。这些交接通过内存访问、原子操作和排序原语实现 [2, 21, 41]。图3 展示了高层的`unlock()`如何映射到释放有序存储，而`lock()` 的成功完成则对应于一次原子获取。这种指令级结构为各种同步实现提供了一个通用的观察点。

匹配交接为解决这两个问题提供了基础。它们标识出段边界和跨核优先关系边，这些信息与同核顺序共同构成依赖有向无环图，用于合并独立计算的段时长。同时，这类交接将先前生产者的写操作与后续消费者的执行关联起来，从而可以在边界处近似处理这些写操作造成的缓存失效效应。因此，在独立计算局部执行时间的同时，同步机制有助于重建跨核时间关系，并估算与交接相关的缓存干扰。

4 GIPSim 的设计

4.1 概述

基于 §3中的观察结果，GIPSim 使用同步引导的时间分解，将执行轨迹划分为多个片段以进行并行重放，并通过依赖关系有向无环图汇总片段时长。指令驱动的核心模型显式支持乱序发射，并结合地址转换与内存时序，可快速、准确地估算片段内的执行周期。

图 4展示了 GIPSim 预测目标工作负载执行时间的工作流程。功能前端（例如功能模拟器或代理机器）执行工作负载，并生成包含表 2所列信息的执行轨迹。在开始独立并行重放之前，GIPSim 会将每核心跟踪划分为多个片段，并构建一个有向无环图$G = (V,\ E)$，其边编码每个核心内的程序顺序以及跨核心的同步依赖。

在阶段 1 中，主机工作线程使用我们的指令驱动的核心模型及其耦合存储器系统（§ 4.2），并行独立地计算片段时长（§ 4.4）。对于片段 v，我们计算

$$d_v = \mathcal{M}(\mathcal{T}_v, \theta; \widehat{x}_v, \widehat{h}_v), \tag{1}$$

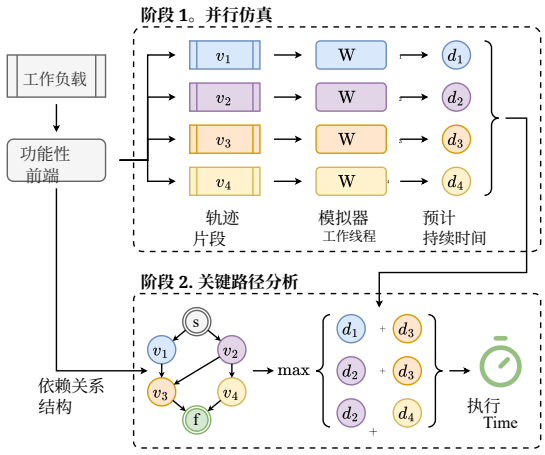


图4：GIPSim 工作流程，从执行轨迹到执行时间预测。

表2：GIPSim 所需的轨迹信息。

记录类型	所需信息
指令	指令地址、微操作分解以及分支结果
内存访问	访问地址、大小和读写方向
同步	匹配的同步端点
地址转换	虚拟到物理映射、页大小和地址转换失效

其中 $\mathcal{T}_v$ 是该片段的轨迹，θ 指定目标配置，$\widehat{x}_v$ 近似其入口状态，h_v 估计共享资源干扰。历史重建和干扰估计提供了独立重放所需的上下文，无需依赖前序时序结果（§ 4.3）。每个时长 d_v 均以目标周期表示。

在阶段 2 中，片段时长会成为 G中的节点权重。关键路径分析会在排序约束下汇总这些权重，以预测程序完成时间（§ 4.5）。

4.2 轨迹分解与任务构建

GIPSim 在同步端点处分割每个核心跟踪，并使用生成的片段作为顶点构建有向无环图 $G=(V,\ E)$。该图包含两种边：

$$E = E_{\text{seq}} \cup E_{\text{sync}}. \tag{2}$$

中的边 E_{seq} 连接同一核心上的连续片段。中的边 E_{sync} 将生产者的释放端点连接到受相应成功获取约束的消费者片段，如图3所示；未成功的尝试不会建立这些交接边。

为减少同步频繁时独立重放所需的高昂上下文重建开销，GIPSim 将每个核心跟踪中的连续片段聚合为一个包含 10^6 条指令的任务 。跨越任务边界的片段会被拆分，从而在 E_{seq}中引入额外的边。

工作线程动态领取任务，并连续重放其中的所有片段，同时分别记录每个片段的持续时间。同一任务内各片段之间会继承微体系结构状态，仅在相关边界处应用与同步相关的缓存行驱逐。

4.3 独立重放的时序上下文

GIPSim 使用多个模拟器工作线程独立重放任务，每个任务都维护自己的时序上下文。为了在近似准确的上下文中模拟片段，GIPSim 会重建入口状态，并对其他片段造成的干扰进行建模。

局部历史。 工作线程复用从先前重放中保留的陈旧微体系结构状态，以避免重复进行冷启动初始化。在任务开始时，从同一目标核心此前约 10^4 条指令中进行预热，使缓存、TLB、预测器和预取器适应近期执行情况。预热周期不计入测量时长。任务内后续片段继承该上下文，无需额外预热，从而避免在每个同步边界都重复进行状态重构。

同步相关缓存状态。 为近似模拟交接相关的缓存效应，GIPSim 会收集每次匹配的释放操作之前约 10^4 条指令中写入的物理缓存行。在消费者进入相应片段之前，这些缓存行会从其本地数据缓存层级中驱逐，并打上标记以指示其远端来源。后续访问若匹配这些标记，则会经过一条时序路径，模拟从另一核心的缓存而非 DRAM 获取数据。对于来自同一生产者核心的连续交接，高水位标记会跟踪写入历史中已处理的位置，确保仅对新覆盖的写入应用失效操作。

增量 DRAM 排队。 GIPSim 使用简化的排队模型近似 DRAM 争用。在预处理期间，它遍历依赖有向无环图，并使用片段指令计数作为代理时长，以估计 α_v，即片段 v期间活跃目标核心的平均数量。在重放期间，根据近期 DRAM 请求密度估计局部利用率 ρ_{local}。将该利用率按 α_v 进行缩放，可近似表示活跃核心产生的总体压力。使用 M/D/1 [5] 平均排队表达式，

$$W(\rho) = \frac{\rho\,\tau_s}{2(1-\rho)}, \qquad 0 \le \rho < 1, \tag{3}$$

其中 τ_s 是有效的请求服务时间。对于 $\alpha_v\rho_{\text{local}} < 1$，基线增量等待时间为

$$\Delta W_v = \max\{0,\ W(\alpha_v\rho_{\text{local}}) - W(\rho_{\text{local}})\}. \tag{4}$$

有限上限会将饱和状态附近及其以上的等待时间估计值限制在一定范围内。这些方程描述的是一种分析近似，并不假设 DRAM 流量严格遵循 M/D/1 过程。缩放局部强度无法再现异构的逐核流量或全局有序的存储体冲突，但无需逐请求协调即可估计争用。

4.4 指令驱动的核心与内存时序

GIPSim 对一种具备按序取指与分派、乱序发射与执行以及按序提交（重新提交）的乱序流水线进行建模，如图5所示。依赖关系、资源占用、

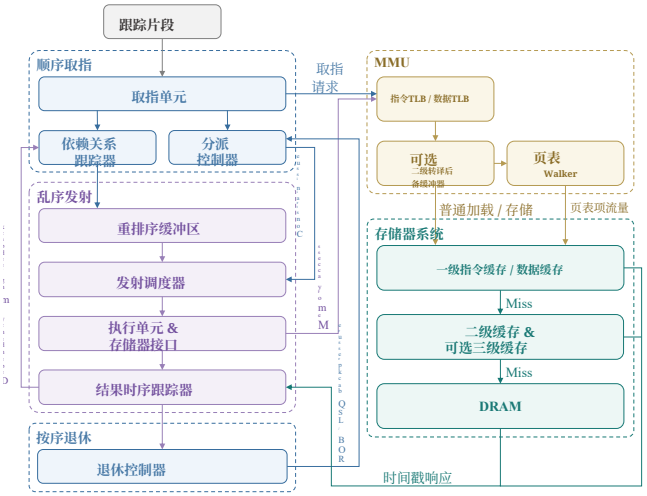


图 5：GIPSim 中简化的单核系统模型。

以及完成时间戳将这些阶段联系起来。用于地址转换、缓存访问和 DRAM 服务时序的基于时间戳的内存模型与核心模型相耦合，请求完成情况会反馈到指令就绪状态、资源可用性和流水线进度中。

按序取指与分派。流水线前端按程序顺序消耗指令跟踪，并考虑取指宽度、指令缓存访问和重命名延迟。分派受限于有限的重排序缓冲区（ROB）、加载队列（LQ）和存储队列（SQ）容量；所需条目不可用时便会停顿。对于已接纳的操作，将建立寄存器生产者—消费者关系。此阶段还会对分支预测进行建模，包括分支方向和目标预测。预测器包括分支目标缓冲区（BTB）、返回地址栈（RAS）和间接分支预测器。当预测结果与轨迹中的对应结果不同时，就会发生预测错误，使后续取指停顿一段恢复延迟。

乱序发射与执行。已分派的 µops 根据操作数就绪情况和资源可用性竞争执行。对于操作 i，设 a_i 为其分派时间，D_i 为其数据依赖关系，$b_{j\to i}$ 为生产者 j 的结果对其可用的时间。其最早的数据就绪时间为

$$r_i = \max\left(a_i, \max_{j\in\mathcal{D}_i} b_{j\to i}\right), \tag{5}$$

对于空依赖集合，使用 $r_i = a_i$。依赖跟踪涵盖寄存器依赖、指令内微操作依赖和转发延迟。

依赖计数器用于跟踪尚未计算结果可用时间的生产者。一旦所有必需的可用时间都已知，该操作就会被插入一个由时间戳桶实现的优先队列中。就绪操作会按就绪周期非递减的顺序发射，同时遵守每周期发射宽度限制。随后，直接根据各阶段的开始周期以及相应阶段的延迟，计算执行、加载、存储和写回阶段的完成周期。一个就绪周期尚未确定的操作必须等待其生产者完成执行，因此其就绪时间不会早于等待调度的已知最早就绪周期。

按序提交与反压。提交仅从重排序缓冲区（ROB）队首进行，并受退役带宽限制。较年轻且提前完成的操作会留在ROB中，直到较早的操作退役。GIPSim通过将已完成的操作保留在ROB槽位数组中，并按照提交宽度限制按序退役，来模拟这一行为。因此，位于队首的长延迟操作可能阻塞退役、填满ROB，并阻塞流水线前端。这保留了从乱序执行，经由按序提交，再到按序分派的反馈机制。

地址转换与存储器时序。GIPSim模拟内存管理单元（MMU），以处理地址转换时序，并模拟基于时间戳的存储器系统，以处理请求服务。MMU包含独立的指令TLB和数据TLB、可选的共享二级TLB，以及并发能力有限的页表遍历器。存储器系统由带有缺失处理资源、预取和写回支持的多级缓存，以及一个DRAM模型组成；该模型会跟踪行状态、银行与通道竞争以及写缓冲。

对于指令取指和数据访问，GIPSim 首先从轨迹中获取虚拟地址和物理地址，并通过内存管理单元确定地址转换时序。一级TLB命中被建模为与缓存访问重叠，不会增加额外延迟；而未命中时，如果已配置，则会发起二级查找。未解决的地址转换会触发页表遍历，其依赖的读取操作会遍历所配置的内存路径。一旦确定地址转换时序，便会通过缓存层次结构，并在需要时通过 DRAM，对原始访问进行计时。所得的完成周期将返回给核心模型，从而确定指令取指时序和数据请求完成时间。对于内存存储，缓存写入发生在微操作退役之后，因此其时序影响主要来自存储队列的持续占用，以及由此对存储分派产生的反压。

4.5 时间重构

局部重放根据上述流水线和内存约束确定每个片段的持续时间 d_v。重放完成后，这些持续时间会成为依赖图中的节点权重，从而确定它们在目标时间线上的位置。令 $\mathrm{pred}(v)$ 表示 v的直接前驱。通过拓扑遍历计算其开始时间和结束时间：

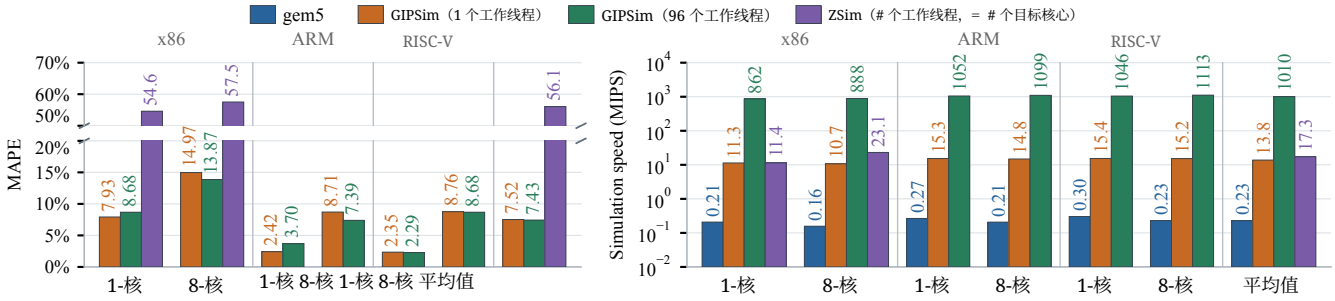
$$\begin{aligned} S_v &= \max\left(\{0\} \cup \{F_u \mid u \in \mathrm{pred}(v)\}\right), \\ F_v &= S_v + d_v. \end{aligned} \tag{6}$$

预计完成时间为

$$\widehat{T} = \max_{v\in V} F_v. \tag{7}$$

如果某个消费者原本会在其同步前驱完成之前启动，相应的边就会推迟它的启动时间；否则，该边不会增加等待时间。遍历耗时为

$$O\left(|V| + |E|\right)。$$



（a）相对于 gem5 的 ROI 周期 MAPE。

（b）模拟吞吐量。

图6：不同指令集架构和目标核心数量下的时序准确性与模拟吞吐量。

表 3：目标系统配置。

组件	配置
指令集架构 / 核心	x86-64、AArch64、RV64GC；1 个或 8 个核心
流水线	8发射乱序核心
资源约束	重排序缓冲区：192；IQ：64；加载队列/存储队列：32/32
分支预测器	锦标赛预测器；4096 项分支目标缓冲区；16 项返回地址栈
一级指令缓存 / 一级数据缓存	各 32 KiB，8 路；缓存行 64 字节；标签/数据延迟：1/1 周期
L2	256 KiB，16 路，私有；标签/数据延迟：10/10 周期
预取	用于数据访问的 4 度步长预取
主存	3 GiB，单通道 DDR4-3200
DRAM 组织结构	两个秩；每个秩 16 个存储体；FR-FCFS 参考配置

表4：目标工作负载。

基准测试套件	工作负载	缩写。
SPEC CPU 2017	500.perlbench_r:test	PB
	503.bwaves_r:test	BW
	605.mcf_s:test	MC
	654.roms_s:test	RO
PARSEC	bodytrack:test	BT
	blackscholes:simlarge	BS
NPB	FT:S	FT
	IS:A	IS

5 评估

我们从时序准确性和模拟效率两个方面评估 GIPSim。首先概述整体结果，然后考察各工作负载的时序准确性、模拟速度与可扩展性，以及不同硬件配置下的准确性。随后，我们报告轨迹采集成本，并评估结果对预热和缓存驱逐历史长度的敏感性。

5.1 实验设置

主机平台。 主机平台是一台双插槽 Intel Xeon Platinum 8558P 服务器，配备 96 个物理核心、两个非统一内存访问节点和 256 GiB 内存。每个核心启用一个硬件线程，频率策略配置为 3.1 GHz。该服务器运行 Ubuntu 24.04.3 和 Linux 6.8.0。

目标系统。 我们在采用 x86、ARM 和 RISC-V 指令集架构的目标系统上评估 GIPSim，每种系统分别配置一个和八个核心。表 3 总结了目标配置。

基线模拟器。我们使用 gem5 [6] 版本 25.0.0.1 作为计时参考，并在全系统模式下使用 O3CPU 对感兴趣区域（ROI）进行仿真。ZSim [44] 为模拟吞吐量提供了另一项基线。由于 ZSim 仅对用户空间执行进行建模，其周期计数不包括内核执行，因此可能低于全系统估计值。Prophet [51] 与 GIPSim 密切相关，但我们的复现尝试显示，其模拟吞吐量低于 0.1 每秒百万条指令，执行时间预测误差最高达 80%。因此，我们将其排除在应用级比较之外。

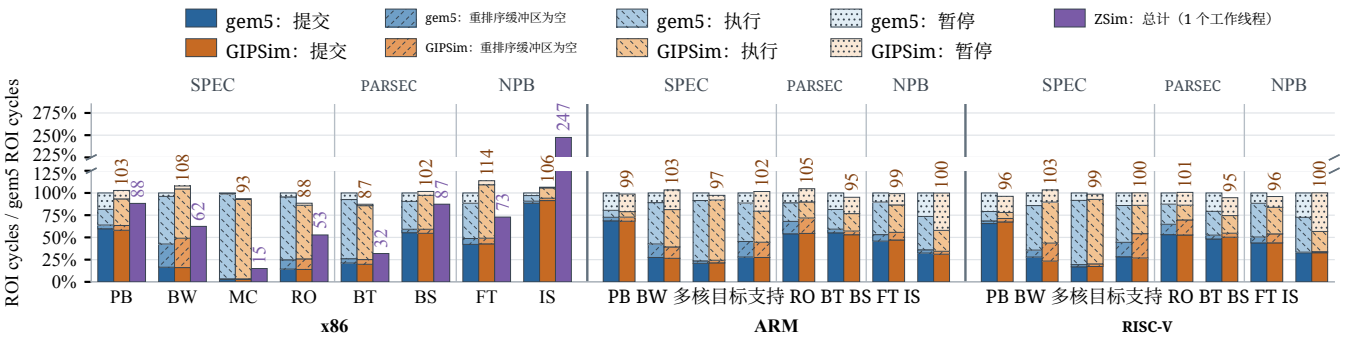
我们在第 [23]2.3节对 ChunkedTejas进行了定性讨论，因为我们无法找到其分块扩展的公开实现。

工作负载与感兴趣区域。所有配置均使用表 4中列出的八个常见工作负载。对于 SPEC CPU 2017 [48]和 PARSEC [39], , 我们使用测试输入；对于 NPB [31], , 我们使用 S 类，但有两个例外：`blackscholes` 使用 `simlarge`，而 `IS` 使用 A 类，以获取足够长的执行轨迹。每个客户机工作负载的并行度均配置为与目标核心数相匹配。感兴趣区域从工作负载启动时开始，到工作负载终止或所有目标核心的指令计数总和达到 10^9（单核目标）或 8×10^9（八核目标）时结束，以先发生者为准。只统计已退休指令；推测执行后被清除的指令不计入。

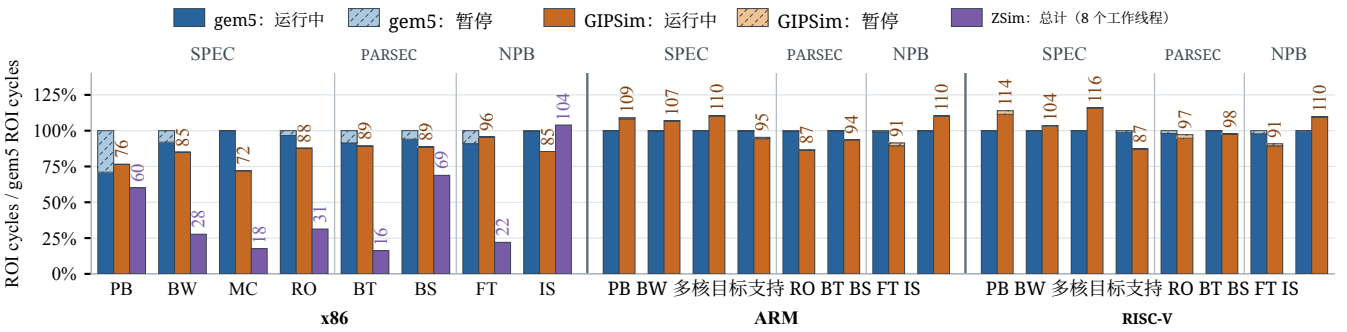
评估指标。我们使用相对于 gem5 的周期计数误差来评估仿真准确度，并报告各工作负载的平均绝对百分比误差，每个工作负载的权重相同。模拟吞吐量以每秒百万条指令为单位，根据感兴趣区域内的指令数和主机端仿真总时间计算。主机端仿真总时间包括启动、重放和后处理。

5.2 总体结果

图 6 总结了 § 5.1中所述六种指令集架构与核心数配置组合的时序准确性和模拟吞吐量。使用一个主机工作线程时，ROI 周期 MAPE 平均为 7.52%；使用 96 个工作线程时，平均为 7.43%。按工作负载和配置进行算术平均后，模拟吞吐量从每秒 13.8 百万条指令提升至每秒 1,010 百万条指令，而 gem5 为每秒 0.23 百万条指令，ZSim 为每秒 17.3 百万条指令。这些吞吐量测量包括初始化、重放和



（a）单核目标。



（b）八核目标。

图 7：各工作负载的模拟周期数，按相应的 gem5 感兴趣区域周期数归一化。GIPSim 使用一个主机工作线程，而 ZSim 为每个目标核心使用一个主机工作线程。

后处理时间，但不包括跟踪收集。ZSim 的平均值仅涵盖 x86 目标。

5.3 时序准确性

图 7 展示了 GIPSim 使用一个主机工作线程时各工作负载的准确性。对于单核目标，x86、ARM 和 RISC-V 的 ROI 周期 MAPE 分别为 7.93%、2.42% 和 2.35%。周期分解还揭示了提交、ROB 为空和执行组件之间的差异，从而可以结合总周期误差考察局部时序行为。x86 的较大偏差包括 +13.77%（FT）以及 −12.68 %（BT）。

对于八核目标，x86、ARM 和 RISC-V 的 ROI 周期 MAPE 分别为 14.97%、8.71% 和 8.76%。该分解将运行周期与所有目标核心均处于暂停状态的时间段区分开来。例如，x86 PB 的总周期误差为 −23.51%，排除全核暂停后则为 +7.63%。暂停占其 gem5 ROI 周期数的 28.95%。这个案例表明，尽管运行周期预测要准确得多，暂停时长差异仍可能主导总周期误差。

5.4 速度与可扩展性

图 8 展示了各工作负载的模拟吞吐量。使用 96 个主机工作线程时，x86、ARM 和 RISC-V 单核目标的平均模拟吞吐量分别达到 862、1,052 和 1,046 每秒百万条指令；相应的八核结果分别为 888、1,099 和 1,113 每秒百万条指令。单核结果表明时间分解能够沿着单个目标核心的执行时间线挖掘出大量主机并行性。

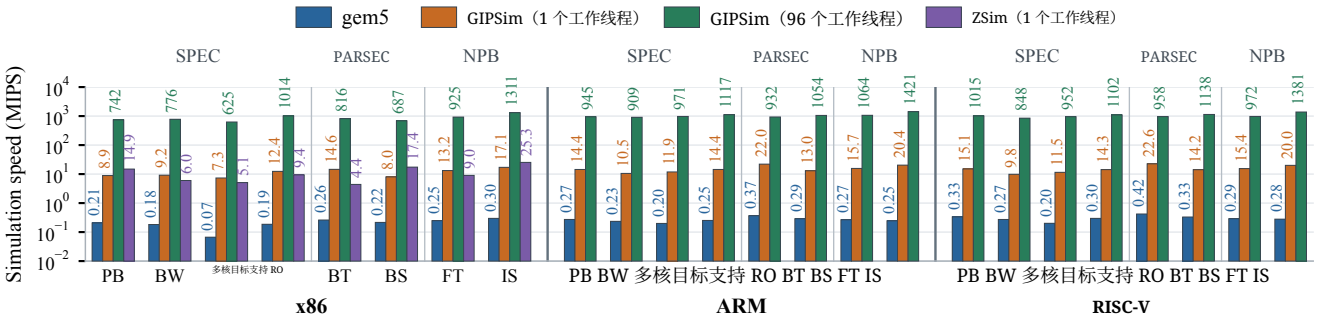
图 9(a) 独立改变主机工作线程数和目标核心数。在 16 个工作线程时，吞吐量相比一个工作线程提升约 15.4–15。8× 。在 96 个工作线程时，六个系统的加速比分别为 68.1–83。0×，单核目标的加速比为 68.1–76。0× 。

随着并行重放加速，初始化和后处理所占比重愈发明显。在 96 个工作线程时，这些阶段占总模拟时间的 3.3– 4.8%。仅重放阶段的加速比达到 70.6– 86.0×，而完整后端的加速比为 68.1–83.0× 。这些测量结果揭示了详细重放的可扩展性与其周边串行工作之间的差异。

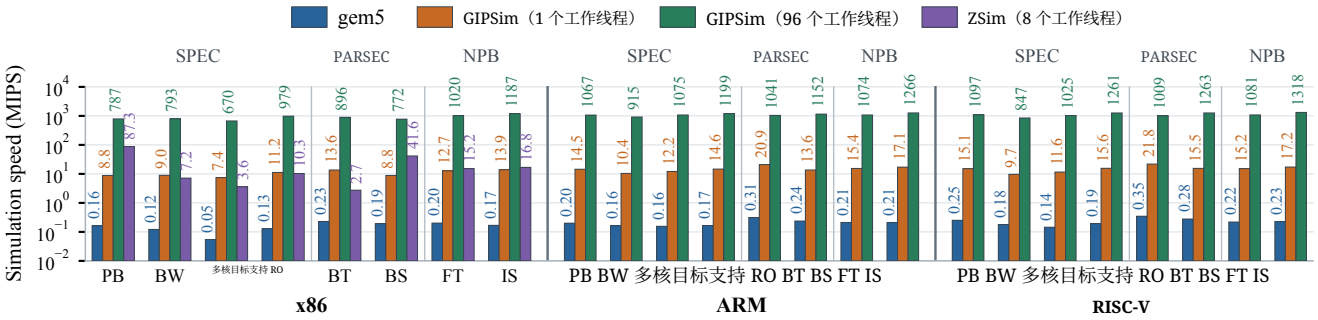
图 9(b) 将这些吞吐量提升与其对准确性的影响并列展示。六个系统的平均 ROI 周期 MAPE 在整个工作线程数扫描范围内保持在 7.01% 至 7.52% 之间。随着重放上下文变化，不同配置呈现出不同的误差趋势；但汇总结果表明，在所测试的主机并行性范围内，准确性变化有限。

5.5 不同硬件配置下的准确性

我们分别或组合调整核心与内存系统参数，评估了另外三种 x86 配置，分别称为 替代核心、替代内存和替代核心与内存。每种配置都分别使用一个和八个目标核心进行评估。相较于表 3，替代核心采用四发射核心和包含 224 个条目的重排序缓冲区。替代内存增加了一个 2 MiB 的共享三级缓存，以及容量各为 1 KiB 的指令/数据页遍历缓存，并且

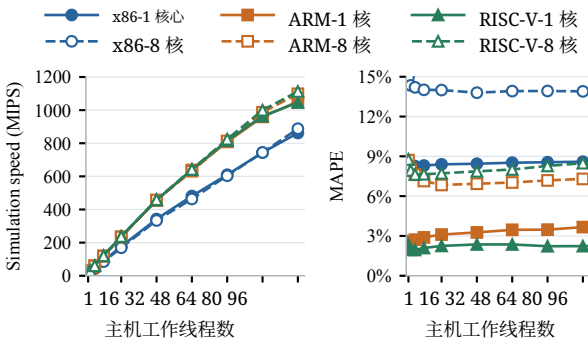


（a）单核目标。



（b）八核目标。

图 8：每个工作负载的模拟　　吞吐量（使用 1 个和 96 个主机工作线程），以及 gem5 的参考测量值　　ts.



（a）仿真吞吐量。

（b）ROI 周期 MAPE 相对于 gem5。

图9：随着主机工作线程数量增加，GIPSim 的模拟吞吐量和时序误差。

将 L1/L2/L3 标签和数据延迟设置为 3/14/67 个周期。替代核心与内存结合了这些更改。

图 10 评估了这些参数变化下的预测准确性。对于单核目标，替代核心、替代内存和替代核心与内存的 ROI 周期 MAPE 分别为 6.37%、5.49% 和 5.31%。相应的八核误差为 5.60%、22.24% 和 19.98%，表明替代内存和替代核心与内存的并行执行对这些变化更加敏感。

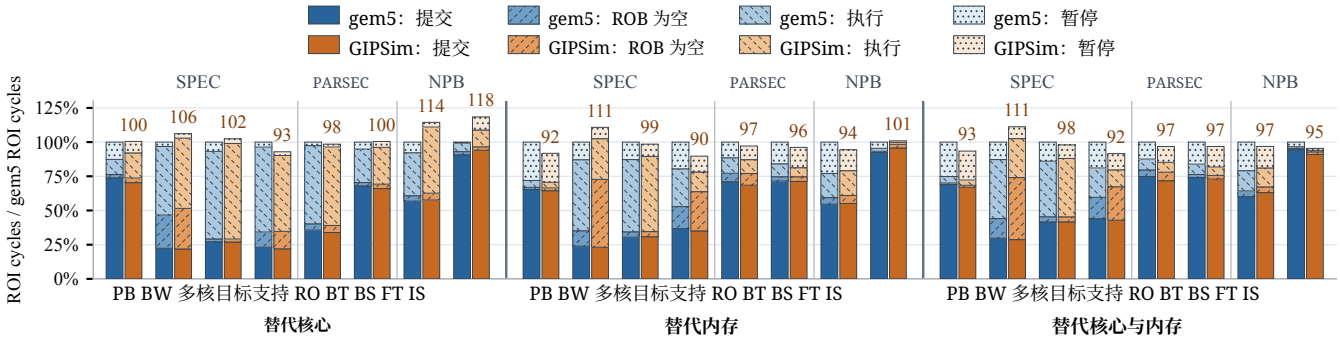
表5：轨迹获取和存储开销，均值置于方括号中。

目标	指令（×10⁹）	捕获（秒）	轨迹（GiB）
x86 / 1-核	0.40–1.00 [0.89]	80–191 [152]	8.5–29.5 [22.4]
ARM / 1-核	0.34–1.00 [0.87]	83–166 [138]	7.6–29.0 [20.9]
RV64 / 1-核	0.35–1.00 [0.87]	35–112 [87]	7.7–27.5 [20.9]
x86 / 8-核	0.40–8.00 [5.08]	59–1296 [306]	8.7–236.1 [136.3]
ARM / 8-核	0.34–8.00 [4.95]	64–896 [245]	7.7–232.3 [124.2]
RV64 / 8-核	0.36–8.00 [5.03]	21–871 [198]	7.8–220.5 [122.9]

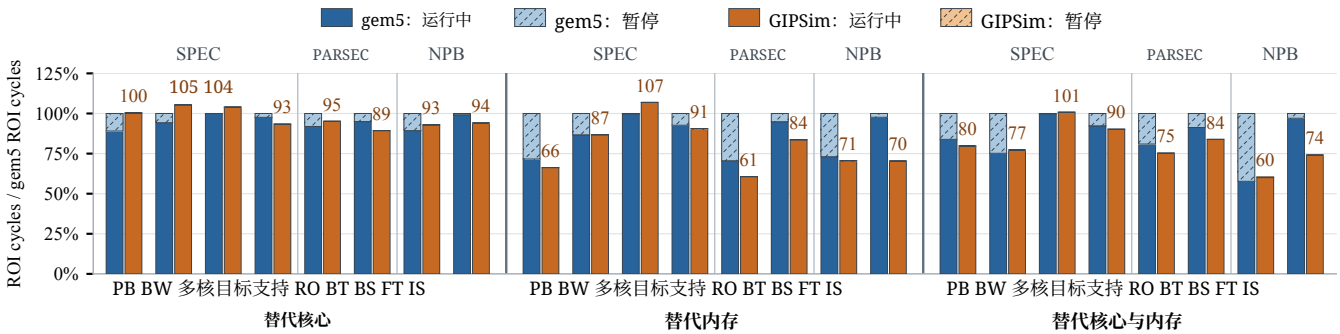
周期分解显示，在这些情况下，暂停时长差异产生了显著影响。排除全核暂停后，替代内存和替代核心与内存的八核平均绝对百分比误差分别降至 9.33% 和 6.73%。例如，在八核替代核心与内存配置下，FT 的总周期误差为 –39.59%，而运行周期的误差为 +4.05%；全核暂停占 gem5 ROI 的 42.29%。这些结果表明，在评估不同硬件配置下的准确性时，有必要区分主动执行预测与暂停时长建模。

5.6 轨迹生成和存储开销

表 5 量化了未纳入后端吞吐量测量的轨迹采集和存储成本。对于每种目标配置，我们先对每个工作负载的重复捕获结果取平均值，然后报告八种工作负载的范围和算术平均值。



（a）单核目标。



（b）八核目标。

图10：使用一个主机工作线程时，不同 x86硬件配置下的时序准确性。

采集时间涵盖完整的 QEMU 捕获过程，包括客户机启动；存储量则按记录的跟踪文件总大小计算。文中还列出了实际的感兴趣区域（ROI）指令计数，以表明捕获的执行量。

在不同指令集架构中，单核目标的平均获取时间为87至152秒，八核目标则为198至306秒。相应的平均轨迹大小分别为20.9–22.4 GiB和122.9–136.3 GiB。这些开销对应着不同数量的记录的执行过程：在配置级别上，单核目标的平均感兴趣区域长度为0.87–0.89 billion条指令，八核目标则为4.95–5.08 billion条指令。

5.7 ReplayContext重构的敏感性

我们考察预热长度W和缓存驱逐历史长度 E对重建的重放上下文的影响。对于每个工作负载，应用程序扫描都会使用固定轨迹，并针对每种设置执行一次重放。我们报告各工作负载的平均绝对周期偏差和总模拟时间。

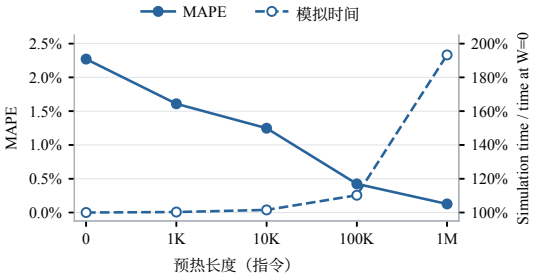
预热长度。 图 11a 使用八个单核 x86 工作负载、96 个主机工作线程以及 $W \in \{0、1K、10K、100K、1M\}$。周期偏差是相对于同一轨迹在没有预热或周期性任务边界的情况下进行单工作线程重放的结果测量的。该参考在整个感兴趣区域内保留了微体系结构历史，并近似表示不间断仿真，从而能够评估边界状态重构的影响。仿真时间以 96 个工作线程的测量结果进行归一化，使用 $W = 0$。

将W从零增加到 10K 后，平均周期偏差从 2.27% 降至 1.25 %，仿真时间增加 1.54%。100K 窗口可将偏差降至 0.42%，但时间增加 10.23%；1M 窗口则可将偏差降至 0.13%，但时间增加 93.31%。更长的前缀能够提高与连续重放的一致性，但会重复更多仿真工作。我们采用 10K 作为默认预热长度，以平衡这一改进与重构成本。

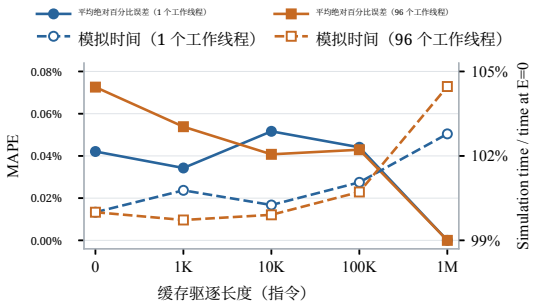
缓存驱逐历史长度。图11b采用八个八核 x86 工作负载，分别使用 1 个和 96 个主机工作线程，预热长度为 10K。我们在相同的五种历史长度上改变 E 。周期偏差使用相应的 $E = 1M$ 重放作为参考进行比较，在测试的最长窗口下，生产者历史记录更短。仿真时间针对每个工作线程数归一化为 $E = 0$ 。

在 $E = 10K$ 下，使用一个工作线程时的平均周期偏差为 0.052%，使用 96 个工作线程时为 0.041%。在所有测试设置中，偏差均低于 0.08%。在 10K 下，相应的仿真时间变化相对于 $E = 0$ 分别为 $+0.26\%$ 和 -0.09%；在 1M 下，时间分别增加 2.78% 和 4.47%。因此，这些应用对生产者历史记录窗口的时序敏感性有限。成本的小幅增长可能部分源于高水位标记，因为它避免了对重叠的生产者写入历史记录进行重复处理。

为了研究有界窗口能够捕获哪些共享数据更新，我们另外采用了四种双核 x86 共享模式。每种模式均使用 16 KiB 的共享负载、一个主机工作线程和 10K 次预热。消费者遍历地址相关的随机链，以限制内存级并行和顺序预取。



(a) 预热长度。



(b) 缓存驱逐历史长度。

图11：对用于重放上下文重构的历史长度的敏感性。

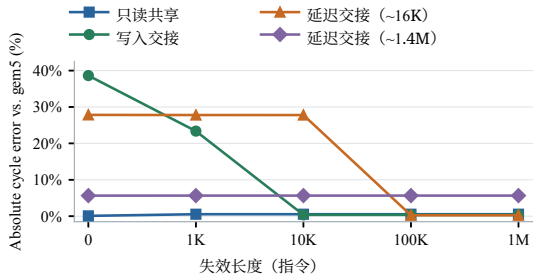


图12：在受控共享模式下，相对于 gem5 对缓存驱逐历史长度的敏感性。

某种模式的所有窗口设置都会重放相同的 gem5 退役轨迹，图 12 报告了相对于对应 gem5 感兴趣区域的绝对周期误差。

对于即时写交接，在禁用驱逐的情况下，周期数会被低估 38.63%。10K 窗口将绝对误差降至 0.37%，而窗口更长时变化不大。只读模式在整个测试范围内的误差保持在 0.09–0.55%。如果交接延迟到最后一次有效载荷写入后约 16K 条指令，10K 窗口的误差仍为 27.78%，而 100K 窗口可将误差降至 0.25%。在第四种模式中，发布发生在最后一次有效载荷写入后约 1.4M 条指令处。即使是测试的最大窗口也无法覆盖到该次写入，因此整个测试范围内的误差始终约为 5.66%。

这些结果表明，对于所评估的应用和即时交接，10K 是低成本重构的合适默认值。延迟共享的情况表明，窗口的覆盖范围取决于发布距离，因此有必要提供可配置的生产者历史长度。

6 讨论与局限性

模拟模式。 当前 GIPSim 的实现由轨迹驱动。在时序评估期间，跨硬件配置复用同一轨迹可以避免重复的功能执行，因此该方法非常适合设计空间探索。在固定配置下评估不同工作负载时，这一优势较为有限，因为每个工作负载都需要单独的轨迹，从而带来存储需求和磁盘 I/O 开销。由执行驱动的实现可以减少这些开销。为实现这一扩展，可以复用局部时序和基于有向无环图的重建机制，并在功能执行期间逐步生成任务和同步依赖。**跨核交互。** GIPSim 的独立重放利用同步关系来近似跨核数据交换。因此，当前的分解方法面向无数据竞争且不存在伪共享的程序。数据竞争可能导致行为取决于未由同步排序的访问。伪共享可能会在对同一缓存行上不同对象的其他独立访问之间引入一致性流量。这些交互可能影响超出已记录同步图的局部时序，并需要额外的跨区域信息。

暂停时间。 有向无环图会考虑记录的依赖关系所隐含的同步等待。没有活动目标核心时，由睡眠引起的时间间隔还取决于唤醒时序，而这无法仅凭指令开销或依赖顺序确定。当无法获得准确的目标睡眠时长时，当前模型会忽略这一部分。对于配备一个主机工作线程的八核 x86 目标，相较于 gem5，总周期的平均绝对百分比误差为 14.97%，而运行周期的误差为 9.99%。对于替代内存，相应数值分别为 22.24% 和 9.33%；对于替代核心与内存，则分别为 19.98% 和 6.73%。若要纳入这些时间间隔，就需要采用与所建模目标相一致的唤醒时序。

7 结论

较长的仿真时间限制了架构研究的规模，尤其是在可用主机并行性超过目标核心数时。GIPSim 通过结合基于指令的时序与时间设计来解决这一瓶颈；该设计利用同步信息，既支持独立重放，也能重建执行时间。这样一来，主机重放顺序便与目标执行顺序解耦，在考虑同步等待的同时释放更多并行性。

在单个主机节点上，96 个工作线程可达到每秒 1,010 百万条指令，峰值为每秒 1,421 百万条指令。相对于 gem5，执行周期的平均绝对百分比误差为 7.43%。这些结果展现了缩短重复架构计时分析周转时间的潜力。

未来的工作将探索 GIPSim 的执行驱动实现，将功能执行与独立局部计时以及基于依赖关系的重构整体执行时间相结合。我们还将研究对其他同步

机制进行建模，以拓宽工作负载覆盖范围。此外，我们将探索估算暂停时间的方法，以改进整体时序预测。

致谢

OpenAI ChatGPT 和 Codex 协助起草和修订了本文的部分内容。作者对最终内容的准确性、原创性、出处标注和完整性承担全部责任。

参考资料

[1] Ayaz Akram 和 Lina Sawalha。2019。《计算机体系结构仿真技术与工具综述》。IEEE Access 7（2019），78120–78145。doi:10. 1109/ACCESS.2019.2917698[2] Arm Limited。 [n. d.]。《Arm A-profile 架构参考手册》。Arm Limited。https://support.arm.com/documentation/ddi0487/maa/ [3] Kenneth C. Barr、Heidi Pan、Michael Zhang 和 Krste Asanović。2005。使用内存时间戳记录加速多处理器仿真。载于 IEEE 系统与软件性能分析国际研讨会（ISPASS）。https://publications.csail.mit.edu/abstracts/abstracts05/kbarr/kbarr.html [4] Fabrice Bellard。2005。QEMU：快速且可移植的动态翻译器。载于 2005 年 USENIX 年度技术会议 FREENIX 赛道。USENIX 协会。https://www.usenix.org/conference/2005-usenix-annual-technical-conference/qemu-fast-and-portable-dynamic-translator [5] Dimitri P. Bertsekas 和 Robert G. Gallager。1992。《数据网络》（第 2 版）。Prentice Hall。https://web.mit.edu/dimitrib/www/datanets.html [6] Nathan Binkert、Bradford Beckmann、Gabriel Black、Steven K. Reinhardt、Ali Saidi、Arkaprava Basu、Joel Hestness、Derek R. Hower、Tushar Krishna、Somayeh Sardashti、Rathijit Sen、Korey Sewell、Muhammad Shoaib、Nilay Vaish、Mark D. Hill 和 David A. Wood。2011。gem5模拟器。ACM SIGARCH 计算机体系结构新闻 39, 2 2011 1 7 10 1145 2024716 2024718 ( ), –。doi:。/. Paul [7] D. Bryan、Jason A. Poovey、Jesse G. Beu 和 Thomas M. Conte。2012。通过屏障区间时间并行加速多线程应用仿真。载于 IEEE 计算机与电信系统建模、分析与仿真国际研讨会（MASCOTS）。doi:10.1109/MASCOTS。2012.23[8] Trevor E. Carlson、Wim Heirman、Kenzo Van Craeynest 和 Lieven Eeckhout。2014。BarrierPoint：多线程应用的采样模拟。载于 IEEE 系统与软件性能分析国际研讨会（ISPASS）。doi: 10 1109 2014 6844456 。/ISPASS。。Trevor E. Carlson、Wim [9] Heirman 和 Lieven Eeckhout。2011。Sniper：探索可扩展且精确的并行多核仿真的抽象层次。载于高性能计算、网络、存储与分析国际会议论文集（SC '11）。计算机协会，52:1–52:12。doi:10.1145/2063384.2063454[10] Trevor E. Carlson、Wim Heirman、Stijn Eyerman、Ibrahim Hur 和 Lieven Eeckhout。2014。高层机制核心模型评估。ACM 体系结构与代码优化汇刊 11，3（2014），28:1–28:25。doi:10.1145/2629677[11] Humberto Carvalho、Geoffrey Nelissen 和 Pavel Zaykov。2020。mcQEMU：使用 QEMU 对多核平台进行时序精确模拟。载于第 23 届 Euromicro 数字系统设计会议论文集（DSD '20）。IEEE，81–88。doi:10.1109/DSD51259.2020.00024[12] José Cubero-Cascante、Niko Zurstraßen、Jörn Nöller、Rainer Leupers 和 Jan Moritz Joseph。2026。使用 parti-gem5 进行可扩展且精确的并行时序模拟。《并行编程国际期刊》54（2026）。doi:10.1007/s10766-026-00819-5 文章 16。[13] Davy Genbrugge、Stijn Eyerman 和 Lieven Eeckhout。2010。区间仿真：提升体系结构模拟的抽象层次。载于 IEEE 高性能计算机体系结构国际研讨会（HPCA）。https://biblio. ugent.be/publication/ 845663 [14] Sylvain Girbal、Gilles Mouchard、Albert Cohen 和 Olivier Temam。2003。DiST：一种简单、可靠且可扩展的方法，可显著减少处理器体系结构模拟时间。载于 2003 年 ACM SIGMETRICS 计算机系统测量与建模国际会议论文集。计算机协会。doi: 10 1145 781028 781029 。/。 [15] Hossein Golestani、Rathijit Sen、Vinson Young 和 Gagan Gupta。2022。Calipers：用于建模处理器性能的临界性感知框架。载于第 36 届 ACM 国际超级计算会议论文集。1–14。doi:10.1145/3524059.3532390[16] Thomas Grass、César Allande、Adrià Armejach、Alejandro Rico、Eduard Ayguadé、Jesús Labarta、Mateo Valero、Marc Casas 和 Miquel Moretó。2016。MUSA：面向下一代高性能计算机器的多级仿真方法。载于 SC16：高性能计算、网络、存储与分析国际会议。 526 537 10 1109 2016 44 – 。doi: 。/SC。 。Thomas Grass、 [17] Alejandro Rico、Marc Casas、Miquel Moretó 和 Eduard Ayguadé。2016。TaskPoint：基于采样的基于任务的程序仿真。载于 IEEE 系统与软件性能分析国际研讨会（ISPASS）。doi:10.1109/ISPASS.2016.7482104[18] John W. Haskins, Jr. 和 Kevin Skadron。2005。面向采样微架构仿真的加速预热。ACM 体系结构与代码优化汇刊 2, 1 2005 78 108 10 1145 1061267 1061272 ( ), –。doi: 。/ 。John L. [19] 2017 Hennessy 和 David A. Patterson。。计算机体系结构：量化方法（第 6 版）。Morgan Kaufmann。[20] Quentin Huppert、Timon Evenblij、Manu Perumkunnil、Francky Catthoor、Lionel Torres 和 David Novo。2021。基于真实硬件顺序执行核心的内存层次结构校准，实现精确模拟。载于欧洲设计、自动化与测试会议及展览（DATE '21）论文集。IEEE，707–710。doi:10.23919/DATE51398.2021.9474108[21] 英特尔公司。 [不适用。] Intel 64 和 IA-32 体系结构软件开发人员手册。英特尔公司。 h ttps:// www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html[22] Radhika Jagtap、Stephan Diestelhorst、Andreas Hansson、Matthias Jung 和 Norbert Wehn。2016。使用弹性跟踪探索系统性能：快速、精确且可移植。载于嵌入式计算系统：体系结构、建模与仿真国际会议（SAMOS）。https://www.gem5.org/documentation/general_docs/cpu_models/TraceCPU [23] Rajshekar Kalayappan、Avantika Chhabra 和 Smruti R. Sarangi。2020。ChunkedTejas：一种并行化轨迹驱动体系结构模拟器的基于分块的方法。ACM 建模与计算机仿真汇刊 30，3（ 2020 15 1 15 21 10 1145 3375397 ），：–：。doi：。/ Hejing Li、 [24] Marvin Meiers、Jialin Li 和 Antoine Kaufmann。2025。SplitSim：面向系统研究的实用大规模全系统模拟。ACM 网络学报 3，CoNEXT4（2025），52:1–52:19。doi:10. 1145/3768999[25] Lingda Li、Santosh Pandey、Thomas Flynn、Hang Liu、Noel Wheeler 和 Adolfy Hoisie。2022。SimNet：使用深度学习进行精确且高性能的计算机体系结构仿真。ACM 测量与分析计算系统学报 6, 2 2022 25 1 25 24 10 1145 3530891 ( ), ：–：。doi：。/ [26] Shanqing Lin 、 Yuanlong Li 、 Mohammad Alian 、 Babak Falsafi 和 Konrad K. Lai。2026。精确且可扩展的全系统基于时间的采样。载于第 59 届 IEEE/ACM 国际微体系结构研讨会论文集。https://infoscience.epfl.ch/handle/20.500.14299/267412[27] Tiantian Lin、Tengjun Long、Mingzhe Zhang、Yingtao Jiang、Xiaohang Wang、Chao Zhang、Amit Kumar Singh、Jieming Yin 和 Kui Ren。2027。FCS：面向模块化模拟器的前沿提交时同步方法。作者发布的摘要。https://mingzhe-zhang.github.io/publications/HPCA2027-2 列为 HPCA 2027 即将发表的论文。[28] Tiantian Lin、Cheng Qiu、Xiaohang Wang、Ling Wang、Zhulin Zheng、Yingtao Jiang、Amit Kumar Singh、Jieming Yin、Sihai Qiu、Xiaodong Li、Xin Tang、Jie Song、Mingzhe Zhang 和 Kui Ren。2025。LEGOSim：面向多芯粒异构集成的统一并行仿真框架。载于第 58 届 IEEE/ACM 国际微体系结构研讨会论文集。1347–1362。doi:10. 1145/3725843.3756068[29] Jason Lowe-Power、Abdul Mutaal Ahmad、Ayaz Akram、Mohammad Alian、Rico Amslinger、Matteo Andreozzi、Adrià Armejach、Nils Asmussen、Bradford Beck- mann、Srikant Bharadwaj、Gabe Black、Bobby R. Bruce、Daniel Rodrigues Car- valho、Jeronimo Castrillon、Lizhong Chen、Nicolas Derumigny、Stephan Diestel- horst、Wendy Elsasser、Marjan Fariborz、Amin Farmahini-Farahani、Pouya Fo- touhi、Ryan Gambord、Jayneel Gandhi、Thomas Grass、Andreas Hansson、Swap- nil Haria、Timothy Hayes、Syed Ali Raza Jafri、Timothy M. Jones、Matthias Jung、Subash Kannoth、Tushar Krishna、Jason Lowe-Power、Christian Menard、Miquel Moreto、Nikos Nikoleris、Lena E. Olson、Marc Orr 和 David A. Wood。2020。gem5模拟器：版本 20。0+。arXiv:2007。03152 [cs.AR] https: //arxiv.org/abs/ 2007 03152 。Jason E. Miller、Harshad [30] Kasture、George Kurian、Charles Gruenwald, III、Nathan Beckmann、Christopher Celio、Jonathan Eastep 和 Anant Agarwal。2010。Graphite：面向多核处理器的分布式并行模拟器。载于第 16 届 IEEE 国际高性能计算机体系结构研讨会论文集（HPCA ' 10 1 12 10 1109 2010 5416635 ）。IEEE，–。doi：。/HPCA。。 [31] [ ] NASA高级超级计算部门。不适用。。NAS并行基准测试。https://www.nas.nasa.gov/software/npb.html。访问日期：2026 年 10 月 23 日。[32] Arash Nasr-Esfahany、Mohammad Alizadeh、Victor Lee、Hanna Alam、Brett W. Coon、David Culler、Vidushi Dadu、Martin Dixon、Henry M. Levy、Santosh Pandey、Parthasarathy Ranganathan 和 Amir Yazdanbakhsh。2025。Concorde：通过组合式分析-机器学习融合实现快速且精确的 CPU 性能建模。载于第 52 届年度国际计算机体系结构研讨会论文集。1480–1494。doi:10.1145/3695053.3731037[33] A.-T. Nguyen、Pradip Bose、K. Ekanadham、A. K. Nanda 和 M. M. Michael。1997。并行轨迹驱动体系结构模拟的精度与加速。载于第 11 届国际并行处理研讨会论文集。IEEE，39–44。

doi:10.1109/IPPS.1997.580842
[34] Alejandro Nocua、Florent Bruguier、Gilles Sassatelli 和 Abdoulaye Gamatié。2019。用于 OpenMP 工作负载快速架构探索的 gem5 轨迹驱动仿真器。《微处理器与微系统》67（2019），42–55。doi:10.1016/j. micpro.2019.01.008[35] Subhankar Pal、Kuba Kaszyk、Siying Feng、Björn Franke、Murray Cole、Michael O'Boyle、Trevor Mudge 和 Ronald G. Dreslinski。2020。HETSIM：使用轨迹驱动、同步与依赖感知框架模拟大规模异构系统。载于 2020 IEEE 国际工作负载特征分析研讨会。13–24。doi:10.1109/IISWC50251.2020.00011[36] Santosh Pandey、Lingda Li、Thomas Flynn、Adolfy Hoisie 和 Hang Liu。2022。在 GPU 上实现可扩展的基于深度学习的微架构仿真。载于 SC22：高性能计算、网络、存储与分析国际会议。
1 15 10 1109 41404 2022 00084 –。doi：。/SC。。Santosh Pandey、[37] Amir Yazdanbakhsh 和 Hang Liu。2024。TAO：重新思考基于深度学习的微架构仿真。ACM 测量与分析计算系统学报 8，2（2024），28:1–28:25。doi:10.1145/3656012[38] Erez Perelman、Greg Hamerly、Michael Van Biesbrouck、Timothy Sherwood 和 Brad Calder。2003。使用 SimPoint 实现精确且高效的仿真。载于 2003 年 ACM SIGMETRICS 计算机系统测量与建模国际会议论文集。
318 319 10 1145 781027 781076 –。doi：。/。普林斯[39] [ ] 顿大学。不适用。。PARSEC 基准测试套件。https://parsec.cs.princeton.edu/。项目网站。[40] Alejandro Rico、Alejandro Duran、Felipe Cabarcas、Yoav Etsion、Alex Ramirez 和 Mateo Valero。2011。多线程应用程序的轨迹驱动仿真。载于 IEEE 系统与软件性能分析国际研讨会论文集（ISPASS '
11 87 96 10 1109 2011 5762718 ）。IEEE，–。[41] 2024 doi：。/ISPASS。。RISC-V International。。RISC-V 指令集手册：原子指令扩展 A，版本 2.1。https://docs.riscv.org/reference/isa/v20240411/unpriv/a-st-ext.html 已批准的规范。[42] Alen Sabu、Harish Patil、Wim Heirman 和 Trevor E. Carlson。2022。LoopPoint：面向多线程应用程序的检查点驱动采样模拟。载于 IEEE 国际高性能计算机体系结构研讨会（HPCA）。doi:10.1109/HPCA53966.2022.00051[43] Alen Sabu、Zhantong Qiu、Harish Patil、Changxi Liu、Wim Heirman、Jason Lowe- Power 和 Trevor E. Carlson。2026。使用循环边界检查点加速并行工作负载的仿真。《ACM 架构学报与代码优化
23, 1 2026 10 1145 3799430 ( )。doi: 。/[44] 2013 Daniel Sanchez 和 Christos Kozyrakis。ZSim：千核系统的快速、精确微体系结构模拟。载于第40届年度国际计算机体系结构研讨会（ISCA '13）论文集。计算机协会，475–486。doi:10.1145/2485922.2485963[45] Karthik Sangaiah、Michael Lui、Radhika Jagtap、Stephan Diestelhorst、Siddharth Nilakantan、Ankit More、Baris Taskin 和 Mark Hempstead。2018。SynchroTrace：用于芯片多处理器和高性能计算工作负载轻量级多核模拟的同步感知、架构无关跟踪。ACM 体系结构与代码优化汇刊
15, 1 2018 2 1 2 25 10 1145 3158642 ( )，：–：。doi: 。/[46] Yakun Sophia Shao、Sam Likun Xi、Vijayalakshmi Srinivasan、Gu-Yeon Wei 和 David M. Brooks。2016。使用 gem5-Aladdin 协同设计加速器和片上系统接口。载于第49届年度 IEEE/ACM 国际微体系结构研讨会（MICRO '16）论文集。IEEE，48:1–48:12。doi:10.1109/ MICRO.2016.7783751[47] Timothy Sherwood、Erez Perelman、Greg Hamerly 和 Brad Calder。2002。自动表征大规模程序行为。载于第十届编程语言与操作系统体系结构支持国际会议（ASPLOS）论文集。https://cseweb.ucsd.edu/~calder/ abstracts/ASPLOS-02-SimPoint.html[48] 标准性能评估公司。 [无日期]。SPEC CPU 2017。https: //www.spec.org/cpu2017/。访问日期：2026年10月23日。[49] The Open Group。2018。The Open Group基础规范，第7期，2018年版。https://pubs.opengroup.org/onlinepubs/9699919799/ IEEE标准1003.1-2017。[50] Roland E. Wunderlich、Thomas F. Wenisch、Babak Falsafi 和 James C. Hoe。2003。SMARTS：通过严谨的统计抽样加速微体系结构模拟。载于国际计算机体系结构研讨会（ISCA）。https: //users.ece.cmu.edu/~jhoe/doku/doku.php?id=smarts_simulation_sampling [51] Weihua Zhang、Xiaofeng Ji、Yunping Lu、Haojun Wang、Haibo Chen 和 Pen-Chung Yew。2017。Prophet：面向并行指令的众核模拟器。《IEEE并行与分布式系统汇刊》28卷第10期（2017），2939–2952。doi:10.1109/TPDS.2017.2700307[52] Zifei Zhang、Yinan Xu、Sa Wang、Dan Tang 和 Yungang Bao。2026。TraceRTL：面向微体系结构探索的敏捷性能评估。载于2026年IEEE国际高性能计算机体系结构研讨会（HPCA）。doi:10.1109/HPCA68181.2026.11408626