软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

Spike-FlexiCAS:支持缓存架构灵活配置的 RISC-V 处理器模拟器*

韩金池[1,2], 王智栋[1,2], 马浩[1,2], 宋威[1,2]

[1](中国科学院信息工程研究所, 网络空间安全防御重点实验室, 北京 100195)

[2](中国科学院大学 网络空间安全学院, 北京 101408)

通讯作者: 宋威, E-mail: songwei@iie.ac.cn

摘 要: 缓存模拟器在缓存架构探索和缓存侧信道研究中起着不可或缺的作用. Spike 作为 RISC-V 指令集的标准实现为基于RISC-V的缓存研究提供了完整的运行环境. 但Spike的缓存模型存在仿真粒度低, 与真实处理器的缓存结构差异大等诸多问题. 为此, 本文修改和扩展 Spike 的缓存模型并取名为 FlexiCAS (*Flexible Cache Architectural Simulator*), 修改后的 Spike 称为 Spike-FlexiCAS. FlexiCAS 能支持多种缓存架构, 具有灵活配置、易扩展等特点并且可以对缓存特性(如一致性协议以及实现方式)进行任意的组合. 此外, FlexiCAS 还能不依赖 Spike 单独地对缓存的行为进行模拟. 性能测试的结果表明, FlexiCAS 对比当前最快的执行驱动型模拟器 ZSim 的缓存模型具有明显的性能优势.

关键词: 缓存架构; 缓存侧信道; RISC-V; 模拟器

中图法分类号: TP311

中文引用格式: 韩金池, 王智栋, 马浩, 宋威. Spike-FlexiCAS: 支持缓存架构灵活配置的 RISC-V 处理器模拟器. 软件学报 http://www.jos.org.cn/xxxx-xxx

英文引用格式: Han JC, Wang ZD, Ma H, Song W. Spike-FlexiCAS: A RISC-V processor simulator supporting flexible cache architecture configuration. Ruan Jian Xue Bao/Journal of Software(in Chinese). http://www.jos.org.cn/xxxx-xxx

Spike-FlexiCAS:A RISC-V processor simulator supporting flexible cache architecture configuration

Han Jin-Chi[1,2], Wang Zhi-Dong[1,2], Ma-Hao[1,2], Song Wei[1,2]

[1](Institute of Information Engineering, Chinese Academy of Sciences, Key Laboratory of CyberSpace Security Defense, Beijing 100195, China)

[2](School of Cyber Security, University of Chinese Academy of Sciences, Beijing 101408, China)

Abstract: Cache simulators play an indispensable role in exploring cache architectures and researching cache side channels. Spike, as the standard implementation of the RISC-V instruction set, provides a complete environment for cache research based on RISC-V. However, Spike's cache model has several issues, including low simulation granularity and significant differences from the cache structures of real processors. To address these issues, this paper modifies and extends Spike's cache model, naming the modified version FlexiCAS (Flexible Cache Architectural Simulator), and refers to the modified Spike as Spike-FlexiCAS. FlexiCAS supports various cache architectures, featuring flexible configuration and easy extensibility, and allows arbitrary combinations of cache features, such as coherence protocols and implementation methods. Additionally, FlexiCAS can simulate cache behavior independently of Spike.The performance test results show that FlexiCAS has a significant performance advantage over the cache model of the currently fastest execution-driven simulator, ZSim.

Key words: Cache architecture; cache side channels; RISC-V; simulator

使用硬件语言(如 Verilog、VHDL 等)开发处理器[1]往往需要投入大量的时间和人力资源, 因此在前期的设计探索阶段通常不会使用硬件语言完整的实现想要测试的处理器结构. 使用软件语言实现的处理器模拟器则

* 基金项目: 国家自然科学基金 (62172406), 中国科学院率先行动“百人计划”青年俊才 (C 类)
收稿时间: 2024-08-25; 修改时间: 2024-10-30; 采用时间: 2024-xx-xx

不仅能实现和评估该领域中的新想法, 还能帮助开发者深入理解程序的行为以及有效的识别出微体系架构中的瓶颈. 例如, Gem5[2]等常用的模拟器提供了对微架构级别的详细建模, 支持多种指令集和体系结构的仿真, 成为研究人员进行微架构设计与优化的重要工具. 随着 RISC-V 开源指令集架构的兴起, 专门针对 RISC-V 的模拟器也变得至关重要. Spike[3]是 RISC-V 的指令集模拟器(Instruction Set Simulator, ISS), 它是 RISC-V 指令集的标准实现(golden model). 所有有关 RISC-V 指令集的更新与扩展都会首先在 Spike 上实现、验证以及评估, 除了使用 GDB 调试器直接调试 Spike 本身, 还可以配合使用 OpenOCD 调试运行在 Spike 之上的 RISC-V 执行程序. 因此 Spike 是目前支持 RISC-V 指令集最完整的运行环境.

缓存作为处理器核和内存间的桥梁, 它的设计对整个计算机系统的性能有重大影响. 缓存的参数众多, 在设计时除了要考虑缓存的大小, 还要考虑缓存的替换策略、缓存一致性协议以及多级缓存间的包含关系等多方面内容. 设计人员需要在对大量的缓存参数组合测试后才能最终确定缓存的结构[4,5]. 与处理器开发类似, 结构简单、快速高效的缓存模拟器可以使开发者在庞大的缓存探索空间中寻找适合设计目标的缓存结构.

缓存模拟器在缓存侧信道攻击和防御也发挥着关键性作用.目前的缓存侧信道攻击已经由同核攻击[6]发展到了跨进程、跨核甚至是跨虚拟机攻击[7,8,37]. 这些攻击的初始想法都会首先通过缓存模拟器进行验证[9,10]. 为了防御缓存侧信道攻击, 一系列的防御措施也相应被提出, 其中一些防御措施是对现有的缓存结构进行修改, 如缓存随机化[11-15]、偏斜缓存(skewed cache)以及将元数据和数据索引关系解耦合的 Mirage Cache[16], 这些防御策略都使用了缓存模拟器进行了功能验证.

综上, 无论是对缓存设计的空间进行探索和性能评估, 还是对缓存侧信道的攻击算法以及新修改的架构进行测试和可行性验证, 缓存模拟器都扮演着不可或缺的角色. 但这也对缓存模拟器提出了如下几个挑战: (1)支持的缓存架构要全面, 这样才能对比不同架构间的性能差异. (2)代码可修改并且易修改, 探索新的缓存架构有可能会对现有的架构做出大幅度的改动, 模拟器需要可改动以支持评估新的架构是否有效并在此基础上快速迭代. (3)既支持只有缓存行为的仿真也支持系统仿真. 在设计新缓存结构的初期阶段往往需要一个无噪声的环境(如没有虚实地址转换的干扰)来验证新的缓存结构是否工作, 而测试缓存时则需要运行测试集(如 SPEC2006[17])获取性能数据, 因此这要求模拟器既能支持简单的读写行为, 也能接入处理器模拟器系统地仿真整个程序的运行.

Spike 是 RISC-V 指令集的标准处理器仿真模型, 但 Spike 的缓存模型的设计目标是尽可能的提升仿真速度, 仿真过程不依赖于缓存, 且不追求对缓存进行真实的细粒度仿真. 因此 Spike 的缓存结构与真实多核处理器的缓存结构有很大不同. 总得来说, Spike 的缓存模型有以下几点缺陷: 缓存结构简单, Spike 的缓存结构仅支持经典多路组相联的二级缓存, 并且无法更换缓存的映射算法和替换算法. 不支持私有缓存, Spike 的多个处理器核心会共享一级指令缓存(Instruction Cache, L1-I)与数据缓存(Data Cache, L1-D)并且不支持多核的缓存一致性协议, 这和大多数真实多核处理器中每个核都有私有的一级和二级缓存的结构相差甚远. 对虚实地址转换的仿真粒度不够. Spike 中仅有用于加速仿真速度的软件旁路转换缓存(Translation Lookaside Buffer, TLB)模块, 缺少真实的硬件旁路转换缓存. 缓存性能指标简单, Spike 的缓存模型仅支持统计简单的读写次数、访问次数、以及缺失次数等指标, 而对于缓存的研究需要更多维度的性能指标. 外部程序与缓存模型交互不灵活, 缺少外部仿真程序与缓存模型动态交互的接口, 外部程序难以动态地获得缓存信息与调整缓存监视器.

本文对 Spike 的缓存模型进行了升级. 具体地, 本文以可配置、易扩展和高性能为目标, 修改和扩展了 Spike 的缓存模型, 并将新的缓存模型命名为 FlexiCAS (*Flexible Cache Architectural Simulator*), 修改后的 Spike 命名为 Spike-FlexiCAS, 新的缓存模型的特点如下:

(1) 灵活配置. 在上下级缓存关系上, 不仅支持常见的包含性(inclusive)缓存, 还支持排它性(exclusive)缓存和非包含非排它性(non-inclusive)缓存. 在缓存一致性协议上, 支持基于广播的和基于目录的 MSI 和 MESI 缓存一致性协议. 此外, 还支持抵御侧信道攻击对缓存进行分区的 skewed cache 以及 Mirage Cache[16]. 用户在使用时可以任意地对不同的缓存特性进行组合, 这是目前其他的处理器模拟器中的缓存模型或单独的缓存模拟器做不到的.

(2) 模块化、易扩展. 该缓存模型采用标准的面向对象设计, 将缓存的各个功能组件进行了抽象、包装和模块化. 用户在使用时可以在原有代码的基础上轻松地设计和扩展出其他的缓存结构. 以支持实现排它性缓存为例, 该缓存模型在缓存一致性的部分上只需修改几个函数即可.

(3) 可集成性强. 新修改的缓存模型实际是做为 Spike 的一个子模块与 Spike 进行集成. 因此它还可以作为单独的缓存模拟器只针对缓存的行为进行仿真. 实际上它还可以与任何的体系结构模拟器进行集成和仿真.

(4) 高性能、仿真准确性高. 该缓存模型对比执行驱动型模拟器 ZSim[18]的缓存模型在对缓存行为模拟上具有明显的性能优势. 此外, 在相同的缓存配置与仿真程序下, 与 Gem5 仿真得到的末级缓存每千条指令的缓存缺失数的 KL 散度(Kullback-Leibler Divergence, KL Divergence)相比其他模拟器最小.

本文第 1 节是相关工作介绍, 简要介绍了目前国内外现有的处理器模拟器和它们支持的缓存特性以及只支持缓存行为仿真的缓存模拟器, 并比较这些模拟器支持的缓存特性与本文设计的缓存模型的相同点和不同点. 第 2 节介绍缓存模型的架构实现, 分别对通信端口和协议、缓存的结构、元数据、数据、缓存一致性协议以及如何单独使用等部分进行阐述, 第 3 节介绍新的缓存模型 FlexiCAS 接入 Spike 时具体做了哪些改动, 第 4 节解释了 Spike-FlexiCAS 如何验证一致性协议的正确性、评估了 Spike-FlexiCAS 在不同配置下的性能以及使用 SPEC2006 测试集[17]和 PARSEC 测试集[19]评估各处理器模拟器的运行速度, 最后是本文的总结以及展望.

本文设计的缓存模型 FlexiCAS 以及与 Spike 集成后的 Spike-FlexiCAS 的代码可以通过以下链接获得(1) https://github.com/comparch-security/FlexiCAS (2) https://github.com/comparch-security/spike-flexicas.

1 相关工作

自 20 世纪 90 年代以来, 学术界中出现了一些模拟器可以被用于缓存的研究[20]. 从功能上区分, 缓存模拟器可以被分为行为级模拟器(behavior simluator)和时序级模拟器(timing simluator)两类. 行为级模拟器不包含时钟, 只在逻辑上模拟缓存的行为, 主要被用于统计缓存的数据指标(如命中率、缺失率等)以及验证缓存架构的正确性. 时序级模拟器则通常包含详细的微架构模型, 它会模拟每个微架构的操作并记录每个操作花费的时间, 这也导致时序级模拟器的仿真速度一般慢于行为级模拟器. 缓存模拟器在模拟的模式上可以分为三种: 执行驱动(execution-driven)型、仿真驱动(emulation-driven)型和仿存序列驱动(trace-driven)型. 执行驱动型的模拟器将需要模拟的程序直接放在本地平台上运行, 这要求程序的指令集(Instruction Set Architectue, ISA)需要和本地的 ISA 兼容. 仿真驱动型的模拟器则会创建一个虚拟环境来运行程序, 这拖慢了程序的运行速度但可以让 ISA 不兼容的程序在本地平台运行. 仿存序列驱动型的模拟器将运行过的程序的仿存序列当作输入, 这种模式不存在 ISA 兼容的问题, 使用者获得仿存序列后可以针对不同的缓存配置进行多次仿真而无需再次运行程序, 但生成的仿存序列一般比较大为仿真带来了额外的时间开销.

Gem5[2]是一个仿真驱动型时序级体系结构模拟器, 它支持对多种 ISA (如 X86、ARM 和 RISC-V 等)的程序进行模拟. Gem5 由 M5 和 GEMS 合并而来, 因此它的内存系统中包含了两个缓存模型: Classic 模型以及 Ruby 模型. Classic 模型只支持基于广播的简化版 MOESI 一致性协议. Ruby 模型则是一个可配置并且高度模块化的缓存模型, 它使用领域规范语言(Specification Language for Implementing Cache Coherence, SLICC)来实现缓存一致性. 使用 SLICC 时需要定义缓存模型的各组件(如目录、缓存等)的状态并声明各状态转移时使用的函数. SLICC 使得开发新的缓存一致性协议成为可能, Gem5 中有很多由 SLICC 声明的一致性协议. 但 SLICC 的开发周期较长并且需要经过大量的测试, 目前还没有其他的模拟器采用 SLICC 来搭建缓存模型. Gem5 由于是完整的体系结构模拟器, 代码体系比较庞大, 因此修改缓存结构的工作量也比较大.

Pin[21]是由 Intel 公司开发的动态二进制插桩框架, 它可以在二进制程序运行的过程中插入各种函数并对程序的行为(如运行指令数、内存访问地址等)进行统计和分析, 因此 Pin 被广泛用于对程序的性能评估以及优化. ZSim[18]和 Sniper[22]都是执行驱动型的时序级体系结构模拟器, 它们在程序运行的过程中通过 Pin 获取指令流等信息, 并以此来驱动整个模拟器的运行. 受 Pin 的限制, ZSim 只能作为 X86 指令集的模拟器, 而

Sniper 在最近的更新改动中还添加了对 RISC-V 的支持[23], 但需要先由 Spike 跑完程序得到仿存序列, 再由 Sniper 的缓存模型进行重放.

Pycachesim[24]和 DineroIV[25]都是轨迹驱动型的模拟器, 它们只包含内存子系统, 因此不能直接模拟程序的运行, 只能以仿存序列做为输入模拟缓存的行为. Pycachesim 支持多级缓存, 上下级缓存间只支持包含性的包含关系, 但它不支持多核缓存. DineroIV 同样只支持包含性的包含关系, 它在支持多级缓存的同时还支持多核缓存, DineroIV 的出现更多是出于教学的目的. CacheFX[26]是一个用于评估缓存安全性的框架, 它实现了 Ceaser[11]、Ceaser-S[12]、Scatter Cache[27]等缓存架构. 由于 CacheFX 的产生是为了评估缓存的安全性, 因此它的仿真模式不同于之前提到的任何一种模拟器. 它会模拟攻击者和受害者使用同一缓存的场景, 并假设一个受害者正在进行一些敏感操作(如 AES 加密等), 以此测试攻击者是否能在某一缓存架构下构架缓存驱逐集或获取受害者的一些信息. 但 CacheFX 的缓存结构只支持一级缓存, 它也无法对程序进行仿真. 因此使用者无法使用 CacheFX 运行测试集评估缓存的性能.

表 1 对比了不同处理器模拟器支持仿真的指令集以及它们的仿真特性的对比, 表 2 对比了各处理器模拟器的缓存模型和单独的缓存模拟器支持缓存特性的对比. 其中 Gem5 的 Ruby 缓存模型虽然支持很多的缓存特性, 但不支持将它们任意的组合在一起, 比如它目前还不支持三级缓存架构下二级缓存作为一级缓存的排它性缓存并且三级缓存是一二级缓存的包含性缓存这种 Intel[28]缓存结构. Sniper 的缓存模型则只支持同一个处理器内私有缓存通过广播进行同步、不同处理器的缓存通过目录进行同步这一种一致性实现方式. 综上所述, FlexiCAS 是目前唯一一个能做到任意组合缓存特性的缓存模型, 这为以后扩展新的一致性协议提供了空间.

表 1 各处理器模拟器的仿真特性比较

	指令集支持			仿真模式	仿真类型	仿真速度
	X86	ARM	RISC-V			
Spike-FlexiCAS	×	×	✓	E	B	~10MIPS
Gem5	✓	✓	✓	E	T	~2MIPS
Sniper	✓	✓*	✓*	X	T	~0.5MIPS
ZSim	✓	×	×	X	T	~60MIPS

注:
E = 仿真驱动型, X = 执行驱动型
B = 行为级, T = 时序级
仿真速度的单位为指令每秒
* Sniper 对于 ARM/RISC-V 指令集是仿存序列驱动型的模拟器

表 2 各处理器模拟器的缓存模型和单独的缓存模拟器支持缓存特性的对比

	多级缓存	支持多核	指令缓存和数据缓存	包含关系			一致性实现方式		一致性协议			缓存特性的任意组合
				包含性	排它性	非包含非排它性	广播	目录	MSI	MESI	MOESI	
Spike-FlexiCAS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Gem5-Classic	✓	✓	✓	×	×	✓	✓	×	×	×	✓	×
Gem5-Ruby	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×
Sniper	✓	✓	✓	✓	×	×	✓	✓	✓	✓	×	×
ZSim	✓	✓	✓	✓	×	×	✓	×	×	✓	×	×
CacheFX	✓	×	×	×	×	×	×	×	×	×	×	×
PyCachesim	✓	×	×	✓	×	×	×	×	×	×	×	×
DineroIV	✓	✓	✓	✓	×	×	×	×	×	×	×	×

2 FlexiCAS 的架构设计

缓存端口、缓存一致性协议和缓存结构是决定缓存如何工作的三个决定性因素，它们代表了缓存不同的功能. 缓存端口是通讯事务处理的主体，一致性协议则规定了通讯处理的状态机，缓存结构负责缓存的数据如何存储并且将缓存底层通用的应用编程接口(Application Programming Interface, API)暴露给端口，供端口上的事务通讯调用. 由于本文的目标是设计一个支持多种一致性协议和缓存结构并且可配置、易修改的缓存模拟器，因此将以上三部分进行了功能抽象并把它们打包为不同的模块. 之后便可以通过模块间的堆叠组成最终的缓存模块. 如图 1 所示，一个支持一致性协议的缓存(Coherent Cache)由通讯内侧端口(Inner)、通讯外侧端口(Outer)、一致性协议(policy)和缓存存储结构(Cache)四个模块组成，其中内侧端口和外侧端口负责传递、处理和转发消息. 此外，内存做为最底层的"缓存"，同样包含一个内侧端口.

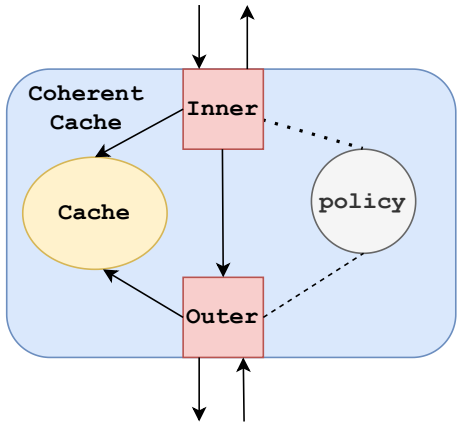


图 1 CoherentCache 结构图

2.1 通信端口

内侧端口和外侧端口是上下级缓存(内存)进行通信的端口以及通讯事务的处理者. 上级缓存作为主节点(Master)，它的外侧端口与作为从节点(Slave)的下级缓存的内侧端口相连接并进行通信. 在一致性缓存内部，内侧端口还会向外侧端口发起事务请求并由其转发给下级缓存的内侧端口. 内侧端口和外侧端口在处理通信事务时会根据事务的类型查询一致性协议以决定如何和上/下级缓存进行交互(如是否向下级缓存发起请求)以及如何操作底层的缓存存储结构(如是否将指定的元数据失效). 图 2 完整的展示了一个包含两个处理器核和三级缓存的系统，每个处理器核都有一个专有的指令缓存和数据缓存. 由于指令缓存只读不写，因此指令缓存被简化为了不参与缓存一致性的缓存，其与下级缓存的通讯也只支持基本的读操作. 为了准确模拟指令缓存的这一行为，FlexiCAS 可将对应的缓存外侧端口设置为不支持一致性的模式，即不接受从下级缓存发出的探测请求，不参与整个系统的一致性通讯.

通信协议 本文参考了开源 RISC-V 处理器 Rocket-Chip[29]和片上系统 Chipyard[30]中所使用的 TileLink 一致性协议的端口设计以及命名格式[31]，表 3 详细阐述了上下级缓存(内存)的内外侧端口通信时使用的消息类型、通信的方向以及它们的功能. 获取(Acquire)请求用于上级缓存发生不命中或者想升级缓存块的权限时使用(如从共享(S)状态升级为修改状态(M))，授予(Grant)则将请求的缓存块的数据返回给上级缓存. 探测(Probe)可以用于维持上下级缓存的包含性关系，以包含性的缓存为例，下级缓存的缓存块被驱逐出缓存时，需要向上级缓存发起探测请求将对应地址的缓存块无效掉. 写回(WriteBack)用于将上级缓存中发生了写请求的脏数据块写回到下级缓存(内存)中. 冲刷(Flush)会将对应的缓存块无效掉并将传刷的消息传递给下级缓存. 但对于指令缓存这一类不参与一致性的缓存，它不会向下级缓存发起写回和冲刷请求，也接受不到下级缓存的探测请求. 完成(Finish)被上级缓存用于通知下级缓存获取请求事务的结束. 末级缓存(Last Level Cache, LLC)和内存

间只需支持读写操作.

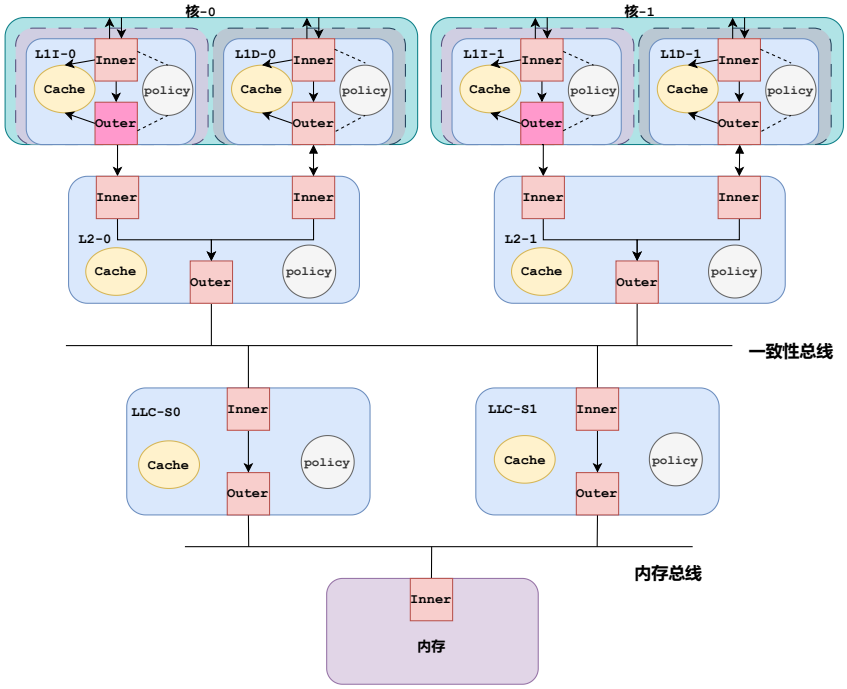


图 2　一个包含两个处理器核和三级缓存的系统

表 3　上下级缓存通信端口间传递的消息类型

消息类型	方向	缓存与缓存	缓存(LLC)与内存
获取(Acquire)	内侧端口向外侧端口	上级缓存请求读一个缓存块或权限升级	LLC 请求从内存读取数据
探测(Probe)	外侧端口向内侧端口	下级缓存请求上级驱逐或写回一个缓存块	未使用
写回(Writeback)	内侧端口向外侧端口	上级缓存写回一个缓存块	LLC 向内存写回一个缓存块
授予(Grant)	外侧端口向内侧端口	下级缓存回复上级缓存的获取请求	LLC 获得之前请求读的数据
冲刷(Flush)	内侧端口向外侧端口	请求冲刷一个缓存块	未使用
完成(Finish)	内侧端口向外侧端口	上级缓存标识请求事务的结束	未使用

除了上下级缓存(内存)需要通信外，一级缓存还需要和处理器核进行通信. 表 4 详细阐述了处理器核和一级缓存通信的消息类型、携带的参数以及它们对应的功能. 其中冲刷操作对应 X86-64 指令集中的 clflush 指令. 指令缓存无需支持写和写回操作. 在处理冲刷缓存请求时，指令缓存采用选择性冲刷策略，仅清除指令数据而非整个缓存内容. 这为实现内存屏障(memory fence)等同步指令提供了支持.

表 4　一级缓存与处理器通信的消息类型

消息类型	携带参数	是否必要	功能
读	地址	是	请求读一个地址的数据
写	地址，写数据	否	请求写一个地址
冲刷	地址	是	请求冲刷一个地址
写回	地址	否	请求写回一个地址
冲刷缓存	无	是	请求冲刷整个缓存

具体实现 上下级缓存的包含关系和缓存的存储结构决定着通信事务处理的流程. FlexiCAS 内侧端口的 UML 框架图如图 3 所示. 内侧基类端口作为抽象基类, 定义了通信协议中的接口函数. 包含性内侧端口继承自内侧基类端口, 它实现了不参与一致性的缓存需要的通信接口, 还定义了通信事务处理需要的通用接口并基于包含性的缓存关系进行了实现. 排它性内侧端口和非包含非排它性内侧端口都继承自包含性内侧端口, 它们分别基于排它性包含关系和非包含非排它性的事务处理流程对通用接口进行了重写. 一致性内侧端口实现了参与缓存一致性的接口, 它可以继承自任意一个不参与一致性的端口从而变为一个能完整处理通信事务的内侧端口. 基于以上搭积木式的框架, 用户可以非常容易的扩展出新的缓存结构需要的通信端口.

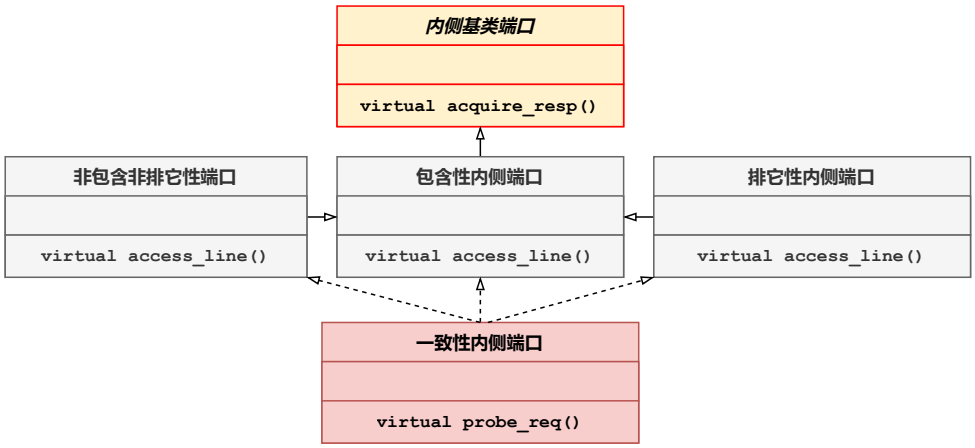


图 3 内侧端口的 UML 框架图

2.2 元数据

元数据(Metadata)作为缓存系统的核心组成部分, 在缓存操作中发挥着如下几个功能: (1)地址匹配: 地址到缓存集合的映射由地址的特定比特决定, 不同的地址有可能映射到同一个缓存集合, 因此需要保存地址的其余比特作为缓存块的标签存储在元数据中用于地址匹配. (2)记录缓存块的状态: 系统使用的缓存一致性协议决定了缓存块可能的状态. 例如, 与 MSI 协议相比, MESI 协议引入了额外的独占状态, 一致性协议需要根据缓存块的状态、当前处理的事务以及预定义的状态机来决定后续的操作. 此外, 在使用写回缓存的策略时, 元数据还需要记录缓存块的脏状态, 即是否包含尚未写回主存的修改数据. (3)共享者跟踪: 在采用目录支持缓存一致性的系统中, 为了优化性能减少不必要的广播通信, 元数据还承担了记录各缓存块在上级缓存的状态的职责, 以此支持一致性维护的操作.

2.3 数据

在当前主流的处理器中, 缓存块的大小通常标准化为 64 字节. FlexiCAS 不仅支持 64 字节的缓存块, 还支持用户动态地调整缓存块的大小. 此外, 由于存储数据却会严重提高缓存模型的内存消耗和降低模拟速度. 因此对于缓存结构的探索和侧信道攻击算法的可行性验证等这些只需要元数据工作就能很好地模拟不同缓存架构下的性能而与具体数据无关的场景, FlexiCAS 也可以选择不存储具体的缓存块数据从而加速模拟的速度. Spike-FlexiCAS 目前默认就是以不带数据的方式调用 FlexiCAS 缓存模型.

2.4 缓存结构与一致性协议

缓存结构存储着缓存的元数据和数据并将元数据和数据操作的底层接口提供给通信端口进行通讯事务的处理. 这些接口的功能可以分为以下几个方面: (1)判断一个地址是否在缓存中命中并且在命中的情况下返回这个地址对应的缓存块的位置(或直接返回对应的缓存块). (2)在给定集合的下标时返回这个集合将要驱逐的路. (3)对读、写、驱逐等缓存事件进行性能统计.

FlexiCAS 的缓存存储结构的组成如图 4 所示: (1)目前缓存侧信道防御提出的新缓存架构多是以缓存分区的偏斜缓存为基础[11,12,16]进行扩展, 为支持缓存侧信道的研究, FlexiCAS 默认支持多分区的缓存结构(图中的分区数为 2), 如果想使用普通的组相连缓存只需指定缓存的分区数为 1. (2)在判断一个地址是否命中以及在不命中的情况下想插入该地址到缓存中都需要先获得该地址到集合的映射. FlexiCAS 将地址到缓存集合的映射这一功能单独抽象为了一个模块(图中为 indexer), 目前支持地址到集合的正常映射(由地址的特定比特决定)以及使用哈希函数的随机映射. (3)缓存的替换策略在很大程度上影响了缓存的性能表现, 它决定了缓存集合要发生驱逐时哪个缓存块被踢出. FlexiCAS 同样把替换策略抽象为了一个模块(图中为 replacer), 目前 FlexiCAS 支持先进先出(FIFO)、最近最少使用(LRU)、再访问间隔预测(RRIP)[32]和随机的替换策略. 用户在现有框架的基础上也可以扩展和测试其他的替换策略的性能. (4)通信端口在处理完一次驱逐、读或写等事务后会执行对应的性能统计接口函数, 更新替换计数器和性能计数器的值. FlexiCAS 将性能计数器的管理模块放在了缓存结构中(图中为性能计数管理器), 用户可以在初始配置阶段选择要挂载哪些计数器.

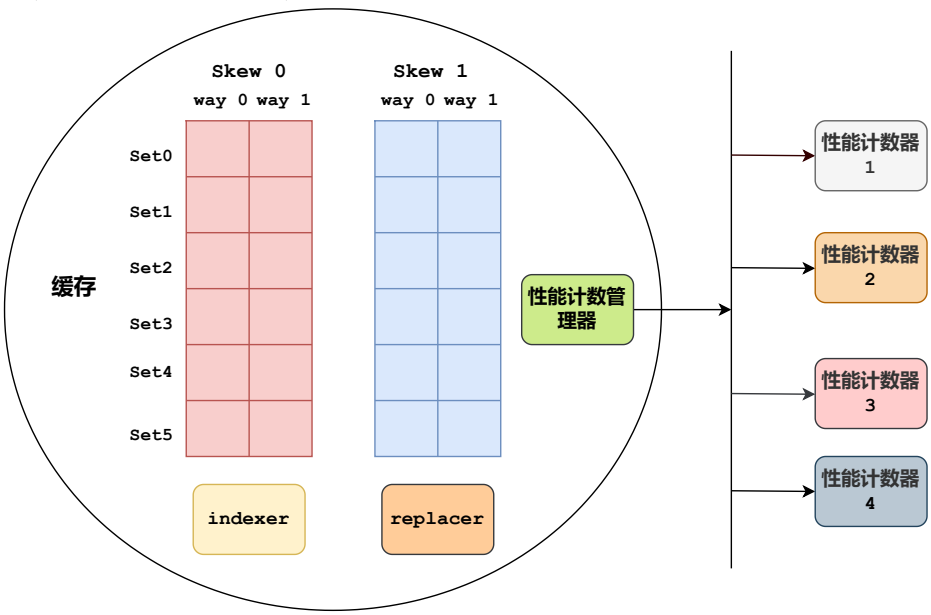


图 4 Cache 结构图

一致性协议做为缓存的状态机, 需要根据端口所处理的事务、缓存和缓存块的状态、缓存的结构等诸多方面去决定下一步的行为. 此外, 它还要在缓存的每一步操作后根据具体使用的一致性协议对缓存块的状态进行修改.

2.5 FlexiCAS的使用方式

FlexiCAS 的代码框架全部使用类模板的实现方式, 大部分类的参数都放到了模板参数中定义. 用户除了能将 FlexiCAS 接入 Spike 使用, 也可以单独的用 FlexiCAS 模拟缓存的行为. 目前 FlexiCAS 已经定义了足够全面的类模板解析函数, 用户可以很快速的搭建起一个可以使用的缓存模型.

图 5 展示了一个如何使用 FlexiCAS 搭建三级缓存的例子. 代码的第 5 行到第 10 行创建了各级缓存、内存以及作为一致性总线向末级缓存的各个切片(Slice)分发消息的分配器, 第 11 行创建了一个记录缓存读写行为的性能计数器, 第 14 到第 19 行负责将各级缓存对应的内外侧端口进行连接, 第 20 行将末级缓存的各个切片和性能计数器连接, 至此便很简洁地完成了三级缓存系统的搭建. 第 22 行到第 23 行用于获取指令和数据缓存面向处理器的接口, 之后用户可以使用表 4 定义的消息函数来驱动这些接口以此模拟缓存的行为.

```
// L1IW、L2IW、L3IW为一二三级缓存集合的数量
// L1WN、L2WN、L3WN为一二三缓存每个集合的路数
// NCore为仿真的核数
int main() {
  auto l1d = cache_gen_l1<L1IW,L1WN,Broadcast,LRU,MSI>(NCore, "l1d"); // 创建私有指令缓存
  auto l1i = cache_gen_l1<L1IW,L1WN,Broadcast,LRU,MSI>(NCore, "l1i"); // 创建私有数据缓存
  auto l2 = cache_gen_l2_exc<L2IW,L2WN,Broadcast,RRIP,MSI>(NCore, "l2"); // 创建排它性的私有L2缓存
  auto l3 = cache_gen_llc_inc<L3IW,L3WN,Directory,RRIP,MESI>(NCore, "l3"); // 创建包含性的L3缓存
  auto dispatcher = new SliceDispatcher<SliceHashIntelCAS>("disp", NCore); // crossbar类型的片上总线
  auto mem = new SimpleMemoryModel("mem"); // 创建内存
  SimpleAccMonitor monitor(true); // 创建性能计数器
  for(int i=0; i<NCore; i++) {
    // 将指令缓存的外侧端口和L2的内侧端口相连接,并设置为不参与一致性的模式
    l1i[i]->outer->connect(l2[i]->inner, l2[i]->inner->connect(l1i[i]->outer, true));
    l1d[i]->outer->connect(l2[i]->inner, l2[i]->inner->connect(l1d[i]->outer));
    dispatcher->connect(l3[i]->inner);
    l2[i]->outer->connect(dispatcher, l3[0]->inner->connect(l2[i]->outer));
    if(i>0) for(int j=0; j<NCore; j++) l3[i]->inner->connect(l2[j]->outer);
    l3[i]->outer->connect(mem, mem->connect(l3[i]->outer)); // 将L3的外侧端口和内存的内侧端口相连接
    l3[i]->attach_monitor(&monitor); // 将末级缓存和性能计数器相连接
  }
  auto core_data = get_l1_core_interface(l1d); // 获取数据缓存面向处理器的接口
  auto core_inst = get_l1_core_interface(l1i); // 获取指令缓存面向处理器的接口
  // ...
}
```

图 5　使用 FlexiCAS 如何搭建三级缓存

3　Spike 接入 FlexiCAS

当通过高速处理器模拟器进行缓存微架构研究时，模拟器的缓存模型需要尽可能贴近真实处理器，同时缓存模型的性能计数器需要尽可能丰富，并支持动态访问、修改. 因此 Spike 无法直接用于处理器缓存微架构的研究，本文实现的全系统处理器模拟器 Spike-FlexiCAS 通过使用 FlexiCAS 代替 Spike 原有的缓存模型，使模拟器缓存行为尽可能的贴近真实处理器.

3.1　缓存模型接口

Spike-FlexiCAS 通过将 Spike 的 MMU 中的所有缓存请求同步到 FlexiCAS 缓存模型接口，替代了原有的缓存模型. 在 Spike 的 MMU 中，有三类行为会访问缓存模型:数据读写、软件指令缓存访问和缓存块清理/失效操作. 当 MMU 进行数据读写时，经过软件 TLB 模块完成地址转换后，将相应的读写请求同步到 FlexiCAS 的读写接口. Spike 原有的缓存模型使用了简单的软件指令缓存来提升仿真速度. 当访问指令缓存时，如果命中，直接通过软件指令缓存进行访问；如果未命中，则通过重新填充软件指令缓存进行访问. 为了不降低仿真速度, Spike-FlexiCAS 保留了软件指令缓存结构. 同时，无论软件指令缓存是否命中，地址转换完成后，都会将指令读取请求同步到 FlexiCAS 的读接口. 清理/失效缓存块行为用于支持 RISCV 指令集中 cbo_clean、cbo_flush 和 cbo_inval 等缓存一致性维护指令. 执行清理/失效缓存块时，将清理/失效请求同步至对应的 FlexiCAS 写回/冲刷接口.

3.2　硬件TLB

Spike 原有的软件 TLB 模块采用单级 TLB 结构，表项仅记录主机地址与物理地址相对于虚拟地址的偏移，不访问缓存，其设计目的是加速仿真速度. Spike-FlexiCAS 在保留软件 TLB 的基础上，添加了硬件 TLB，以兼顾仿真速度和真实模拟 TLB 对缓存的影响. Spike-FlexiCAS 的硬件 TLB 架构基于 Coffee Lake (Intel Gen 9)的

二级 TLB 设计[33], 其中一级指令 TLB 与数据 TLB 共享二级 TLB.

Spike-FlexiCAS 的硬件 TLB 主要提供三个接口供 MMU 使用: (1)地址转换: 将虚拟地址转换为物理地址. 如果 TLB 命中, 返回 TLB 表项; 如果未命中, 则需要进行页表遍历(walk)操作并将结果填充到 TLB 中, 同时访问缓存模型的一级缓存. (2)冲刷 TLB: 清空 TLB 中的所有表项. (3)地址转换函数模块: 封装地址转换函数, 以供 CSR 接口调用.

3.3 CSR接口

为了实现外部仿真程序对 FlexiCAS 缓存模型的动态访问, 并能够灵活调整缓存模型的性能计数器, Spike-FlexiCAS 扩展了一个用户模式下的非标准 CSR (本文称为“Spike 交互 CSR”), 用于实现外部程序与 FlexiCAS 缓存模型之间的交互.

Spike 交互 CSR 的地址空间映射了 FlexiCAS 缓存模型中所有交互行为的命令编码. FlexiCAS 缓存模型内部定义了 CSR 命令编码与交互行为之间的映射关系. 外部程序可以通过 csrw 指令将命令编码写入 FlexiCAS 缓存模型的 CSR 写接口. 收到命令编码后, 缓存模型将执行对应的交互行为. 如果该行为产生返回值, 返回值将被存储在 Spike 交互 CSR 中. 外部程序可以通过 csrr 指令读取 Spike 交互 CSR 以获取交互行为的返回值.

4 实验评估

本文主要从正确性和性能两个方面评估 Spike-FlexiCAS. 正确性评估集中在本文新增的缓存模型一致性协议的准确性, 这通过运行并行测试程序 Litmus[34]以及与 Gem5 比较运行同一测试集后的缓存性能数据来验证. 性能评估则包括以下几个方面: (1)Spike-FlexiCAS 在不同配置下的性能测试. (2)将 Spike-FlexiCAS 与其他处理器模拟器的性能进行对比.

本文使用的实验平台为一台配备了 16 核 32 线程的 AMD Ryzen Threadripper 3955WX CPU 以及 128GB 内存的计算机, 使用的操作系统为 Ubuntu 22.04, 编译器的优化等级均设置为 O3. 所有模拟器无特殊说明的情况下默认使用的缓存一致性协议均为 MESI, 一级数据缓存和指令缓存的大小为 8 路、32KB, 二级缓存的大小为 16 路、1MB, 三级缓存的大小为 16 路、2MB, 使用的测试集均为 SPEC2006 测试集[17], Spike-FlexiCAS 的运行线程数为 1.

4.1 正确性验证

Litmus[34]是用来验证计算机体系结构中内存一致性模型的工具, 它允许研究人员编写并行的测试程序, 通过运行这些测试, 可以观察并验证硬件(软件)是否遵循预期的内存一致性模型. 本文通过在 Spike-FlexiCAS 上运行 Litmus 并使用程序的运行结果以及比较 Spike 内存模型的和 FlexiCAS 内存模型的一致性来验证修改后的缓存模型的正确性. 本文指定 Litmus 的测试核数为双核, 测试的缓存配置为三级缓存(一级和二级缓存的包含关系为排它性, 二级和三级缓存的包含关系为包含性, 使用的一致性协议为 MESI), 最终测试得到的程序运行结果通过了正确性的验证.

本文还比较了包含性二级缓存、缓存协议为 MESI 一致性协议配置下 Spike-FlexiCAS、Gem5[2]、Sniper[21]和 ZSim[18]运行 SPEC2006 测试集[17]得到的缓存性能数据的准确性. 其中使用到的具体配置为: Spike-FlexiCAS 使用的内核为支持运行用户态程序的 riscv-pk, 核数为 1; Gem5 的 CPU 类型为 TimingSimpleCPU, 内存类型为 SimpleMemory, 核数为 1, 运行的模式为用户模式模拟(User-Space Simulation, SE); Sniper 的 CPU 类型为 nehalem, 内存类型为 constant, 核数为 1; ZSim 的 CPU 类型为 SimpleCore, 内存类型为 SimpleMemory, 核数为 1. 具体地, 本文测试了各模拟器运行 SPEC2006 测试集各测试样例 10G 条指令后末级缓存的每千条指令缓存缺失数(Misses Per Kilo Instructions, MPKI), 得到的结果如图 13 所示. 由于 Gem5 是学术界和工业界最广泛使用的模拟器, 因此本文认为使用 Gem5 模拟得到的结果最为准确. KL 散度可以描述信息的损失程度以及评估不同的分布间存在的信息差异, 对于两个离散的概率分布 P 和 Q，如果χ是所有可能的离散状态集合, P(x)是事件 x 在概率分布 P 中的概率, Q(x) 是事件 x 在概率分布 Q 分布的概率, 那么分布 P 和 Q 的 KL 散度可以

通过以下公式计算:

$$D_{KL}(P||Q) = \sum_{x\in\chi} P(x) log \frac{P(x)}{Q(x)} \quad (1)$$

因此为了衡量其余三个模拟器的结果和 Gem5 的差距, 本文计算了其他三个模拟器对比 Gem5 的 MPKI 的 KL 散度. 得到的结果为 Spike-FlexiCAS 与 Gem5 的 KL 散度为 0.0035, Sniper 和 ZSim 与 Gem5 的 KL 散度则为 0.0276 和 0.5652. Spike-FlexiCAS 与 Gem5 的 MPKI 的 KL 散度最小, 即两个模拟器在 MPKI 的结果差异最小, 这在一定程度上说明了 Spike-FlexiCAS 针对缓存仿真的准确性和正确性.

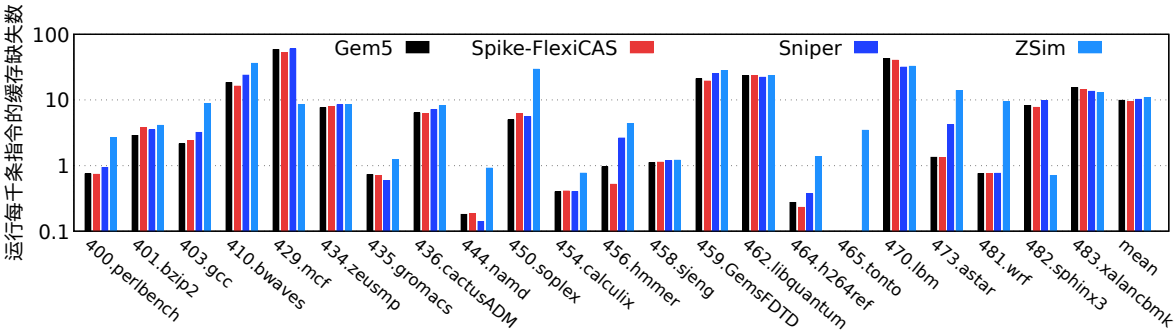


图 6 各模拟器运行 SPEC2006 测试集 10G 条指令的 MPKI 大小对比

4.2 不同配置下Spike-FlexiCAS的性能测试

评估缓存携带数据运行对仿真速度的影响 本文评估了 Spike-FlexiCAS 仿真时缓存携带数据和不携带数据对整体仿真速度的影响以及对缓存仿真速度的影响. 具体地, 本文测试了在 Spike-FlexiCAS 运行 SPEC2006 测试集时, 仿真 100G 条指令携带数据和不携带数据所需的时间, 测试的结果如图 7 所示. 所有基准测试在携带数据运行时在缓存操作花费的时间和整体仿真的时间均高于不携带数据运行. 特别地, 对于某些基准测试, 如 454.calculix 和 481.wrf, 携带数据时的缓存操作时间较不携带数据增加了 35%以上, 整体仿真时间的增幅也超过了 10%. 对于所有的基准测试, 携带数据时的缓存花费时间与整体仿真时间的增幅分别约为 15%和 5%. 仿真时间增加的原因是由于携带数据运行会涉及到对数据的拷贝和迁移, 因此降低了仿真的速度.

不同缓存层级对仿真速度的影响 本文评估了不同的缓存层级对 Spike-FlexiCAS 的整体仿真速度影响以及对缓存仿真速度的影响. 具体地, 本文首先测试了 Spike-FlexiCAS 在缓存层级为一级、二级、三级缓存以及都设置为包含性的上下级缓存关系时, 仿真 SPEC2006 测试集 100G 条指令花费的时间, 测试的结果如图 8 所示. 所有基准测试在缓存操作花费时间和整体仿真时间都随着缓存层级的提升有所延长. 其中二级缓存和三级缓存对比一级缓存在缓存操作花费的时间分别平均增长了 36.6%和 60.3%, 整体仿真的时间分别平均增加了 9.9%和 16.7%. 仿真时间增加的原因是由于缓存层级的增加会导致维护缓存一致性的操作变多, 因此仿真的速度也会变慢.

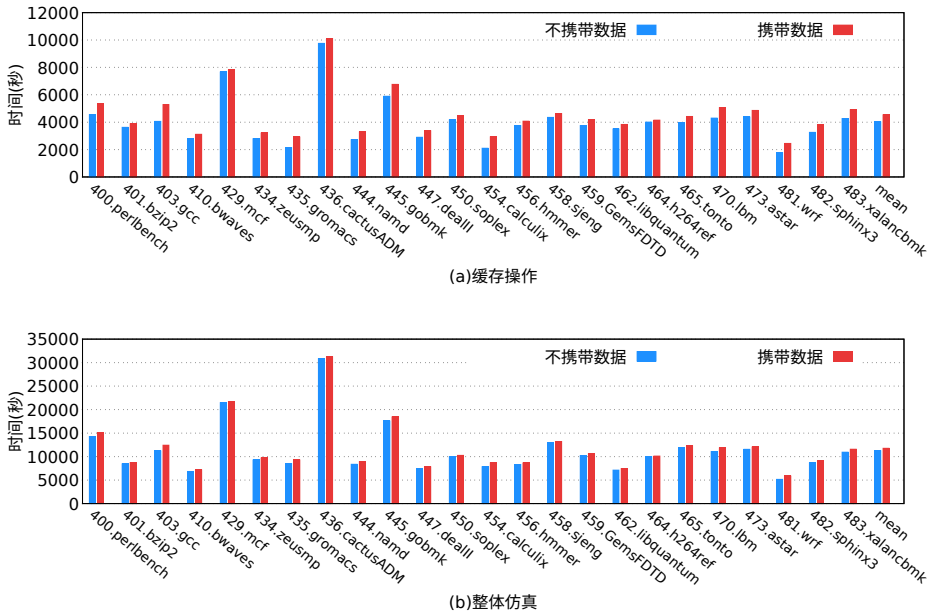


图 7 Spike-FlexiCAS 携带数据和不携带数据运行 SPEC2006 测试集 100G 条指令花费时间对比

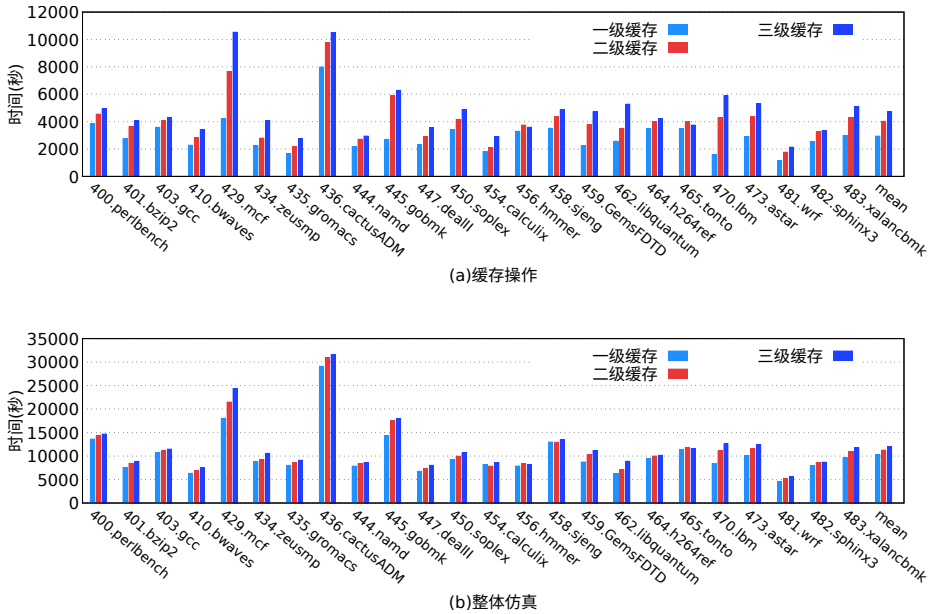


图 8 Spike-FlexiCAS 在一级、二级、三级缓存层级下运行 SPEC2006 测试集 100G 条指令花费时间对比

不同上下级缓存包含关系对仿真速度的影响 本文评估了不同的上下级缓存包含关系对 Spike-FlexiCAS 的整体仿真速度影响以及对缓存仿真速度的影响. 具体地, 本文首先测试了 Spike-FlexiCAS 在缓存层级为三级, 二级缓存是一级缓存的包含性缓存, 三级缓存是一级和二级缓存的包含性和排它性缓存时, 仿真 SPEC2006 测试集 100G 条指令花费的时间, 测试的结果如图 9 所示. 在不同的基准测试下,包含性缓存和排它性缓存的缓存运行时间和整体仿真时间并无绝对的优劣. 如对于 429.mcf 和 465.tonto 来说, 包含性缓存的仿真性能更优, 但对于 436.cactusADM 和 464.h264ref 等基准测试, 排它性缓存的仿真性能则更胜一筹. 导致这种现象的原因是, 排它性缓存相对包含性缓存能容纳的缓存数据更多, 但维护缓存一致性的操作也更复杂, 因此不同的数据访问行为会导致仿真性能的差异.

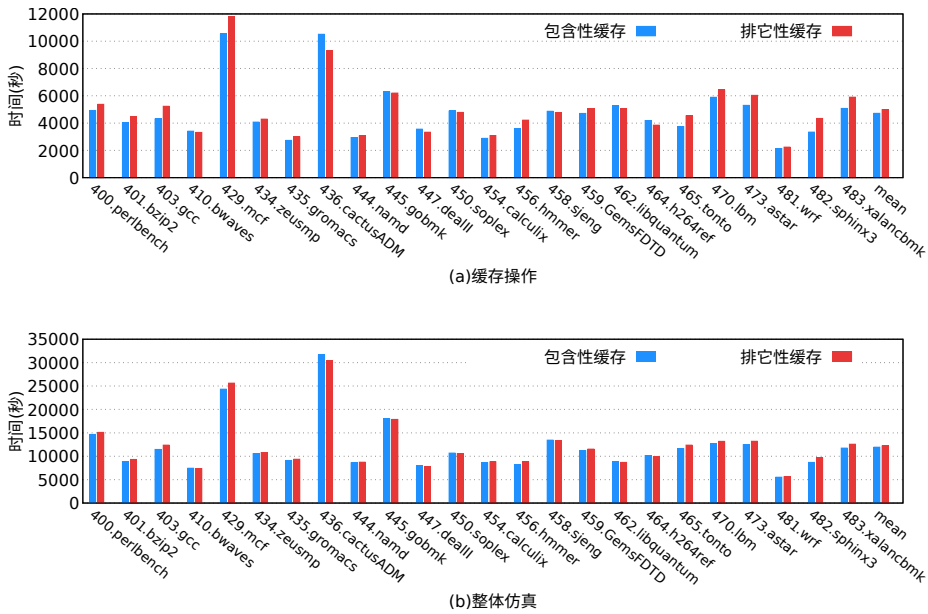


图 9 Spike-FlexiCAS 在三级缓存，不同缓存包含关系运行 SPEC2006 测试集 100G 条指令花费时间对比

多核仿真对仿真速度的影响 一款模拟器能否启动 Linux 操作系统是评价其实用性的重要参考依据，在 Spike-FlexiCAS 上能够成功运行 Linux 操作系统(内核版本为 6.6.2，根文件系统为 tmpfs). 本文测试了在不同核数的设定下启动 Linux 操作系统的时间(从开始启动操作系统到输入用户帐号的时间). 测试的结果如图 10(a)所示，由于 Spike-FlexiCAS 是单线程运行,因此随着运行核数的增加，启动操作系统的时间也逐渐增加. 而由于 Spike-FlexiCAS 是单线程运行，因此在 Spike-FlexiCAS 上进行多核仿真时，理论上在多核的每个核上跑相同负载的真实时间应与在单核上跑一个负载的真实时间成线性增加的关系. 为了评估修改了缓存模型的 Spike-FlexiCAS 是否满足该关系，本文测试了在不同核数设定下的 Spike-FlexiCAS 上运行 Linux 操作系统，并将每个核的负载都设为相同的情况下(本文将负载设置为运行 SPEC2006 测试集中的 400.perlbench 的 10G 条指令)，负载运行完成花费的时间. 测试的结果如图 10(b)所示，随着运行核数的增加，负载运行完成花费的时间基本都和核数维持在正比例的关系.

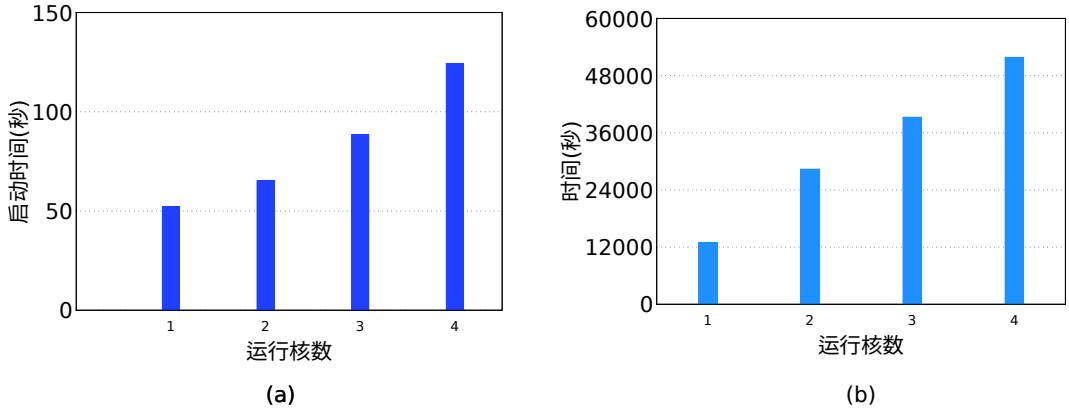


图 10 Spike-FlexiCAS 不同核数的设定下在 Linux 操作系统上操作花费的时间对比: (a) 启动 Linux 操作系统的时间(b) 每个核的负载都设为相同时，负载运行完成花费时间对比

4.3 测试集速度测评

本文比较了包含性二级缓存、缓存协议为 MESI 一致性协议配置(其它配置和本文 4.1 节中说明的模拟器配置相同)下 Spike-FlexiCAS、Gem5、Sniper 和 ZSim 在同一机器上运行 SPEC2006 测试集的运行速度, 其中 Spike-FlexiCAS 和 Gem5 作为仿真驱动型的模拟器运行 RISC-V 架构的测试集, Sniper 和 ZSim 作为执行驱动型的模拟器运行 X86 架构的测试集, 测试它们运行 100G 条指令的速度. 各模拟器设置运行的核数均为一.

四个模拟器运行 SPEC2006 测试集的速度如图 11 所示. ZSim 作为执行驱动型的模拟器会将测试的程序直接在本地运行因此速度最快, 仿真速度约为 60M 条指令每秒. 速度第二快的模拟器是 Spike-FlexiCAS, 仿真速度能达到 10M 条指令每秒. 最后是 Gem5 和 Sniper, 两者的仿真速度大约为 2M 条指令每秒和 0.5M 条指令每秒.

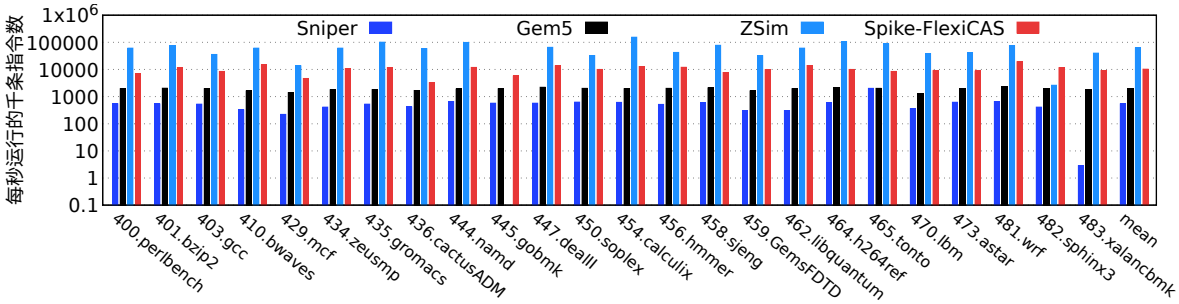


图 11 各模拟器运行 SPEC2006 测试集的速度对比

为了测试多核配置下的性能，本文还测试了不同模拟器运行并行测试集 PARSEC[18]的运行速度. 为了支持 PARSEC 测试集的运行，Spike-FlexiCAS 和 Gem5 都是首先运行 Linux 操作系统(因此 Gem5 的运行模式调整为全系统模式模拟), 之后再在 Linux 环境上进行测试. 所有模拟器设置运行的核数均为二, 其余配置则和前文描述的保持一致.

四个模拟器运行 PARSEC 测试集 simsmall 规模的速度如图 12 所示. ZSim 依旧是运行速度最快的模拟器, Spike-FlexiCAS 则紧随其后.

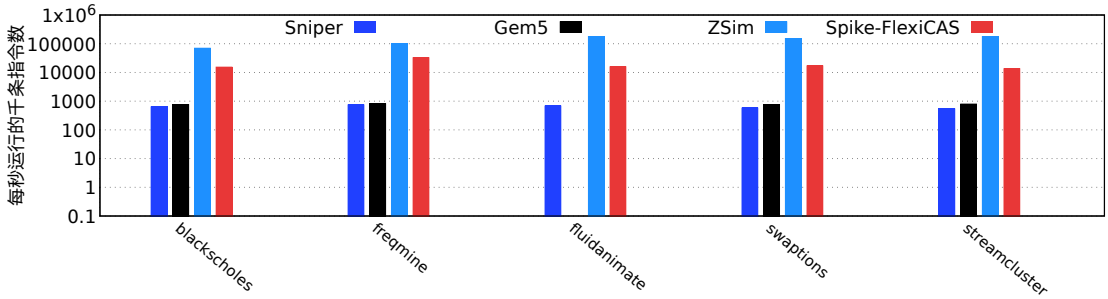


图 12 各模拟器运行 PARSEC 测试集的速度对比

本文还将 ZSim 的缓存架构进行了分离, 使其能够单独地模拟缓存行为, 以此对比 ZSim-Cache 和 FlexiCAS 的性能. 具体地, 本文首先使用 Spike-FlexiCAS 运行 SPEC2006 测试集并收集执行 10G 条指令的仿存序列. 之后, 在相同的缓存配置和编译优化配置下, 测试 ZSim-Cache 和 FlexiCAS 对这些仿存序列重放的时间, 结果如图 13 所示. 对于所有基准测试的仿存序列, FlexiCAS 的重放时间均少于 ZSim-Cache 的重放时间. 特别地, 对于某些基准测试, 如 400.perlbench 和 401.bzip2, ZSim-Cache 的重放时间甚至达到了 FlexiCAS 重放时间的 2.8-3.2 倍. 总体而言, ZSim-Cache 对于所有基准测试的仿存轨迹的平均重放时间是 FlexiCAS 的 2.31 倍.

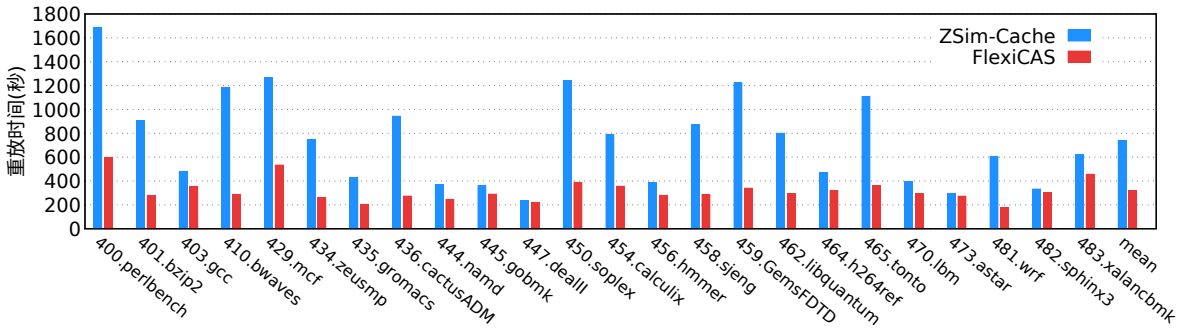


图 13　ZSim-Cache 和 FlexiCAS 重放 SPEC2006 测试集 10G 条指令的仿存序列花费时间对比

5　总结与展望

在这篇文章中，本文对 RISC-V 处理器模拟器 Spike 的缓存模型进行了升级，新升级的缓存模型具有可配置、易扩展以及高性能的特性，此外它还支持对任意不同的缓存特性进行组合．性能测试的结果表明，FlexiCAS 对比当前最快的执行驱动型模拟器 ZSim 的缓存模型具有明显的性能优势．例如，它们在对相同的仿存序列仿真时，FlexiCAS 的仿真性能是 ZSim 缓存模型的 2.31 倍．后续工作除了为 FlexiCAS 添加更多的缓存特性外，本文还计划参考和借鉴这些[35,36]工作升级 Spike-FlexiCAS 以使其支持多线程并行的多核仿真并与真实处理器进行比较与校准．

References:

[1] Sagar Karandikar et al. “FireSim: FPGA-accelerated Cycle-exact Scale-out System Simulation in the Public Cloud”. In: Proceedings of the 45th Annual International Symposium on Computer Architecture. ISCA ’18. 2018, pp. 29–42.

[2] Krste Asanovic et al. “The rocket chip generator”. In: EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17 (2016).

[3] Andrew Waterman et al. Spike RISC-V ISA Simulator. https://github.com/riscv/riscv-isa-sim.

[4] Pat Conway, Nathan Kalyanasundharam, Gregg Donley, Kevin Lepak, and Bill Hughes. “Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor”. In: IEEE Micro (2010), pp. 16–29.

[5] Li Zhao, Ravi R. Iyer, Srihari Makineni, Don Newell, and Liqun Cheng. “NCID: a non-inclusive cache, inclusive directory architecture for flexible and efficient cache hierarchies”. In: Proceedings of the 7th Conference on Computing Frontiers. 2010, pp. 121–130.

[6] Dag Arne Osvik, Adi Shamir, and Eran Tromer. “Cache attacks and countermeasures: the case of AES”. In: The Cryptographers’Track at the RSA Conference. 2006, pp. 1–20.

[7] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. “Last-Level Cache SideChannel Attacks are Practical”. In: IEEE Symposium on Security and Privacy, SP. 2015, pp. 605–622.

[8] Yuval Yarom and Katrina Falkner. “FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: Proceedings of the 23rd USENIX Security Symposium. USENIX Association, 2014, pp. 719–732.

[9] Antoon Purnal, Lukas Giner, Daniel Gruss, and Ingrid Verbauwhede. “Systematic analysis of randomization-based protected cache architectures”. In: 2021 IEEE Symposium on Security and Privacy (SP). 2021, pp. 987–1002..

[10] Zihan Xue, Jinchi Han, and Wei Song. “CTPP: A Fast and Stealth Algorithm for Searching Eviction Sets on Intel Processors”. In: Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses. 2023, pp. 151–163.

[11] Moinuddin K. Qureshi. “CEASER: Mitigating Conflict-Based Cache Attacks via Encrypted-Address and Remapping”. In: 51st Annual IEEE/ACM International Symposium on Microarchitecture, MICRO. 2018, pp. 775–787.

[12] Moinuddin K. Qureshi. “New attacks and defense for encrypted-address cache”. In: Proceedings of the 46th International Symposium on Computer Architecture, ISCA. 2019, pp. 360–371.

[13] Wei Song, Boya Li, Zihan Xue, Zhenzhen Li, Wenhao Wang, and Peng Liu. “Randomized LastLevel Caches Are Still Vulnerable to Cache Side-Channel Attacks! But We Can Fix It”. In: 42nd IEEE Symposium on Security and Privacy, SP. 2021, pp. 955–969.

[14] Wei Song, Zihan Xue, Jinchi Han, Zhenzhen Li, and Peng Liu. “Randomizing Set-Associative Caches Against Conflict-Based Cache Side-Channel Attacks”. In: IEEE Transactions on Computers vol. 73, no. 4 (2024), pp. 1019–1033.

[15] Zhenghong Wang and Ruby B. Lee. “New cache designs for thwarting software cache-based side channel attacks”. In: 34th International Symposium on Computer Architecture,ISCA. 2007, pp. 494–505.

[16] Gururaj Saileshwar and Moinuddin Qureshi. “MIRAGE: Mitigating Conflict-Based Cache Attacks with a Practical Fully-Associative Design”. In: 30th USENIX Security Symposium. 2021, pp. 1379-1396.

[17] John L. Henning. “SPEC CPU2006 benchmark descriptions”. In: SIGARCH Comput. Archit. News vol. 34, no. 4 (Sept. 2006), pp. 1–17.

[18] Daniel Sánchez and Christos Kozyrakis. “ZSim: fast and accurate microarchitectural simulation of thousand-core systems”. In: The 40th Annual International Symposium on Computer Architecture, ISCA. 2013, pp. 475–486.

[19] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. “The PARSEC benchmark suite: Characterization and architectural implications”. In: Proceedings of the 17th international conference on Parallel architectures and compilation techniques. 2008, pp. 72–81.

[20] Hadi Brais, Rajshekar Kalayappan, and Preeti Ranjan Panda. “A Survey of Cache Simulators”. In:ACM Comput. Surv. vol. 53, no. 1 (2020).

[21] Chi-Keung Luk et al. “Pin: building customized program analysis tools with dynamic instrumentation”. In: Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation. 2005, pp. 190–200.

[22] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. “Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation”. In: SC ’11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. 2011, pp. 1–12.

[23] Neethu Bal Mallya, Cecilia Gonzalez-Alvarez, and Trevor E Carlson. “Flexible timing simulation of RISC-V processors with sniper”. In: Workshop on Computer Architecture Research with RISC-V (CARRV). 2018.

[24] pycachesim. https://github.com/RRZE-HPC/pycachesim.

[25] dinero. https://pages.cs.wisc.edu/~markhill/DineroIV

[26] Daniel Genkin, William Kosasih, Fangfei Liu, Anna Trikalinou, Thomas Unterluggauer, and Yuval Yarom. “CacheFX: A Framework for Evaluating Cache Security”. In: Proceedings of the ACM Asia Conference on Computer and Communications Security, ASIA CCS. 2023, pp. 163–176.

[27] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. “ScatterCache: Thwarting Cache Attacks via Cache Set Randomization”. In: 28th USENIX Security Symposium, USENIX Security. 2019, pp. 675–692.

[28] coffee_lake. https://en.wikichip.org/wiki/intel/microarchitectures/coffee_lake.

[29] A Krste Asanovic et al. “The rocket chip generator”. In: EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17 (2016).

[30] Alon Amid et al. “Chipyard: Integrated Design, Simulation, and Implementation Framework for Custom SoCs”. In: IEEE Micro vol. 40, no. 4 (2020), pp. 10–21.

[31] Henry Michael Cook. Productive design of extensible on-chip memory hierarchies. University of California, Berkeley, 2016.

[32] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely Jr., and Joel S. Emer. “High performance cache replacement using re-reference interval prediction (RRIP)”. In: 37th International Symposium on Computer Architecture (ISCA). 2010, pp. 60–71.

[33] Andrei Tatar, Daniël Trujillo, Cristiano Giuffrida, and Herbert Bos. “TLB;DR: Enhancing TLBbased Attacks with TLB Desynchronized Reverse Engineering”. In: 31st USENIX Security Symposium,USENIX Security. 2022, pp. 989–1007.

[34] Jade Alglave, Luc Maranget, Susmit Sarkar, and Peter Sewell. “Litmus: Running tests against hardware”. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Vol. 6605. Springer. 2011, pp. 41–44.

[35] Radhika Jagtap, Stephan Diestelhorst, Andreas Hansson, Matthias Jung, and Norbert Wehn. “Exploring system performance using elastic traces: Fast, accurate and portable”. In: International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation, SAMOS 2016, pp. 96–105

[36] Karthik Sangaiah et al. “SynchroTrace: Synchronization-Aware Architecture-Agnostic Traces for Lightweight Multicore Simulation of CMP and HPC Workloads”. In: ACM Trans. Archit. Code Optim. vol. 15, no. 1 (2018), 2:1–2:26

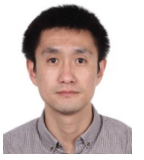
附中文参考文献:

[37] 王崇, 魏帅, 张帆, and 宋克. “缓存侧信道防御研究综述”. In: 计算机研究与发展 vol. 58, no. 4 (2021), pp. 794－810.

韩金池(2000－),男,硕士生,主要研究领域为计算机体系结构安全.

王智栋(2001－),男,硕士生,主要研究领域为计算机体系结构安全.

马浩(1994－),男,博士生,主要研究领域为随机化缓存侧信道防御..



宋威(1983－), 男,副研究员,博士生导师,主要研究领域包括安全处理器设计、计算机体系结构安全、编译器的安全优化技术、基于 RISC-V 的处理器设计.