计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2021,30(11):3−10 [doi: 10.15888/j.cnki.csa.008346]
©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

基于 RISC-V 的新型硬件性能计数器①

薛子涵[1,2], 解　达[1,2], 宋　威[1,2]

[1](中国科学院 信息工程研究所, 北京 100093)
[2](中国科学院大学 网络空间安全学院, 北京 100049)
通讯作者: 宋　威, E-mail: songwei@iie.ac.cn

摘　要: 经过多年的发展, X86 架构与 ARM 架构的处理器逐渐分别占据了桌面端和移动端市场的主导地位. 虽然无论从技术角度还是从生态体系方面, 这两类架构的处理器性能越来越高, 但是由于其指令集臃肿、技术复杂、授权困难等原因, 使得开发这两类架构的处理器的门槛较高. 研究院所还没有一个合适的指令集用于体系结构的研究和创新. RISC-V 指令集的开源使得这一局面得以缓解. 其具备精简、开源、敏捷开发等特点引起了工业界与学术界的广泛关注与积极参与. 性能计数器 (Hardware Performance Counter, HPC) 是处理器研究和性能调优的重要工具. 由于 RISC-V 制定的标准性能计数器的可拓展性欠佳、可同时捕获事件的数量有限等不足, 本文提出一种新的基于 RISC-V 的分布式硬件性能计数器. 本文使用 Genesys2 开发板作为实验平台, 将这种性能计数器适配到 lowRISC-v0.4 开源 SoC 项目上, 完成了对该设计方案的验证与评估. 该性能计数器只占用 3 个控制状态寄存器 (Control and Status Registers, CSRs) 就可以同时捕获比标准的性能计数器多近乎一个数量级的事件, 在 RISC-V 处理器的性能分析、结构优化、侧信道攻防等方面为研究者提供了翔实的统计数据.

关键词: RISC-V; 分布式硬件性能计数器; 控制状态寄存器

引用格式: 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. http://www.c-s-a.org.cn/1003-3254/8346.html

Hardware Performance Counter Based on RISC-V

XUE Zi-Han[1,2], XIE Da[1,2], SONG Wei[1,2]

[1](Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100093, China)
[2](School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China)

Abstract: With decades of development, X86 and ARM have gradually dominated the markets of desktops and mobile phones. Although these two architectures are becoming increasingly powerful from the standing points of technical advances and software ecosystem, they are not good candidates for architectural research due to their complicated Instruction Set Architecture (ISA) definitions, comprehensive technical designs and intimidating copyright protection issues. Before the introduction of the open RISC-V ISA, there is no appropriate ISA for computer architectural research and innovation. RISC-V has attracted attention and participation from both the industry and academia. Hardware Performance Counter (HPC) is an important tool for researching and optimizing computer processor cores. The original definitions of HPC in the RISC-V standard do not scale properly and the number of events simultaneously monitorable is rather small. For these reasons, we propose a new distributed HPC based on RISC-V in this study. We have integrated this design into the lowRISC-v0.4 open SoC platform and run it on the Genesys2 FPGA board. Our HPC only uses three

① 基金项目: 国家自然科学基金 (61802402); 中国科学院率先行动“百人计划”青年俊才 (C 类)
Foundation item: National Natural Science Foundation of China (61802402); Youth Talent of “Hundred Talents Program” Pioneer Initiative, Chinese Academy of Sciences (Category C)
本文由“RISC-V 技术与生态”专题特约编辑武延军研究员、李玲研究员以及邢明杰高级工程师推荐.
收稿时间: 2021-04-27; 修改时间: 2021-05-21; 采用时间: 2021-06-11; csa 在线出版时间: 2021-10-22

© 中国科学院软件研究所 http://www.c-s-a.org.cn

Control and Status Registers (CSRs) to capture all events. The number of events that can be concurrently monitored is one order of magnitude higher than that the RISC-V standard can support. Meanwhile, our strategy could provide detailed and accurate data for researchers focusing on the performance analysis of RISC-V processors, the architectural optimization, and side-channel attack and defense.

Key words: RISC-V; distributed hardware performance counters; Control and Status Registers (CSRs)

1 引言

现代处理器中一般都拥有若干被称为硬件性能计数器 (Hardware Performance Counter, HPC) 的寄存器[1]. 这种寄存器只会按照软件的配置在指定的事件发生时自增计数, 对处理器的正常工作或者性能表现不会有影响. 这些被记录的信息能够帮助软件工程师对代码进行优化, 也能够协助计算机架构师对处理器结构进行调优, 因此 HPC 几乎成为所有现代处理器的标配. 作为近年来备受学术界和工业界关注的开源指令集 RISC-V, 其处理器在设计之初就把 HPC 考虑了进去并为其制定了相关标准. RISC-V 特权指令规范[2] 定义了若干控制和状态寄存器 (Control and Status Registers, CSRs) 作为计数器和选择器. 其中选择器将一个或多个 (微) 体系结构事件的信号接入计数器, 从而构成了可捕获特定事件的 HPC. 在现代处理器结构愈发复杂的时代, 这种将 HPC 集中在 CSR 模块内的方式不利于 HPC 的拓展, 也限制了可同时监测的体系结构事件数量. 为此, 本文设计了一种分布式的性能计数器. 按照处理器内部模块的划分, 将 HPC 分布在不同区域, 并通过片上互连将 HPC 和 CSR 模块连接, 从而降低了 CSR 模块复杂度, 同时也为 HPC 的拓展提供了便利.

本文组织架构如下: 第 2 节介绍了相关的背景知识. 第 3 节介绍了 SiFive U74-MC[3] 的性能计数器并分析其不足之处. 第 4 节针对现有的 HPC 方案存在的问题, 提出了分布式的 HPC 方案. 第 5 节完成了将分布式的 HPC 部署到 lowRISC-v0.4[4], 通过运行 SPEC CPU2006[5] 分析结果并评估该方案效果. 第 6 节总结本文.

2 研究背景

2.1 RISC-V

RISC-V[6] 是一种优秀的开源指令集架构, 由加州大学伯克利分校于 2010 年发布. RISC-V 充分吸取了其他指令集优点: 结构优雅便于实现、灵活性强方便拓展、免费开源无须高昂的授权费用、社区活跃工具链完善等[7,8]. 这些优点使其得到了众多科研团队和商业公司的青睐. 目前已有平头哥、SiFive 在内的商业公司推出了基于 RISC-V 指令集的处理器. 然而 RISC-V 定义的 HPC 存在不足, 无法同时监测多种体系结构事件, 在使用中局限性较大.

2.2 RocketChip

RocketChip[9] 是加州大学开源的基于 RISC-V 的 64 位处理器. 其中 RocketChip 的 Rocket 处理器核采用 5 级标量顺序流水线, 采用 Chisel (Constructing hardware in a Scala embedded language) 语言开发[10]. 作为加州大学为推广 RISC-V 而开发的开源处理器, RocketChip 为全世界的科研团队提供了针对体系结构研究的优良平台.

2.3 lowRISC-v0.4

lowRISC-v0.4 是剑桥大学开发的基于 RocketChip 的开源 SoC, 其基本信息如表 1 所示[4]. 相比于其他开源 SoC 项目 lowRISC-v0.4 加入的 OSD (Open SoC Debug) 模块[11] 提供了函数追踪、特权模式监测、程序下载 (通过 UART 接口) 等功能. 这些功能在代码调试过程中提供了详细的调试信息. 本文利用 OSD 模块与上位机结合的方式, 使得 lowRISC-v0.4 能够自动运行 SPEC CPU2006 benchmark[5], 并在运行结束后读取 HPC, 获取体系结构事件的统计值, 从而验证 HPC 能否正常工作.

表 1 lowRISC-v0.4 基本信息

模块	参数
指令集	RISC-V IMAFD
主频	50 MHz (Genesys2 board[12])
流水线结构	5级顺序标量
一级指令缓存	32 sets × 4 ways
指令TLB	8 entries (full-associative)
一级数据缓存	32 sets × 4 ways (include 2 mshrs)
数据TLB	8 entries (full-associative)
二级(末级)缓存	1 bank × 128 sets × 8 ways
内存	1 GB 800 MHz DDR3 (Genesys2 board)
外设	UART SPI (SD) RTC
调试模块	Open SoC Debug

© 中国科学院软件研究所 http://www.c-s-a.org.cn

2.4 硬件性能计数器

硬件性能计数器是一组能够记录 (微) 体系结构事件发生次数的寄存器. 这些事件通常包括时钟周期数、已执行指令数、分支预测失败次数、各级缓存 (cache) 的缺失/命中次数、TLB (Translation Look-aside Buffer) 的缺失/命中次数等[1]. 对于会造成流水线阻塞的事件, 某些处理器中的 HPC 还会记录该事件导致的阻塞周期数[13,14]. 这些寄存器通常作为系统寄存器 (比如 ARM 中的 System Register, RISC-V 中的 CSR), 从而能够被指令直接访问. 尽管 HPC 不是处理器正常工作的必要组成单元, 也不会对处理器性能有影响, 然而 HPC 提供了硬件运行过程中实时的状态信息. 利用这些信息我们能够更加高效地对系统状态进行监测[1]、对硬件资源进行高效利用[15]、对功耗进行合理管理[16]、对恶意代码进行有针对性的检测[17,18]、对计算机系统结构[19]进行优化. 因此, 几乎所有现代处理器都会配置该计数器. HPC 的管理和配置工作通常由性能监测单元 (Performance Monitor Unit, PMU) 完成. PMU 为每一个 HPC 都分配了一个事件类型选择寄存器, 当 (微) 体系结构中发生的事件和该寄存器选中的事件匹配时, PMU 就会控制对应的 HPC 增加计数值. 除了计数器和选择寄存器这两个成对出现的寄存器之外, 某些处理中的 PMU 还包含了其他寄存器, 从而可以提供更丰富、更强大的额外功能: 比如访问控制、特权级过滤 (如过滤用户态下的事件) 等. HPC 的基本配置和访问较为简单, 通常只需在选择寄存器中, 选中要监控的事件. 对于较为复杂的需要则可以直接使用 Linux 下的开源工具—perf[20]. 经过多年的发展, 商业处理器 HPC的硬件结构和软件配套已经十分完善, 能够满足各种应用需求. 然而, 迄今为止开源 RISC-V 处理器 (如 RocketChip, lowRISC) 中的 HPC 存在功能简陋、拓展性弱、能够同时统计的体系结构事件少等不足, 无法满足对于体系结构的研究需求. 基于此本文设计、实现了一种分布式 HPC, 并使用该 HPC 进行我们相关的体系结构研究工作.

3 U74-MC 的性能计数器

U74-MC[3] 是由 SiFive 公司发布的基于 RISC-V 指令集的 64 位高性能多核处理器, 可应用于数据中心、通信基站等对性能要求较高的场景. 作为目前性能最优的 RISC-V 处理器之一, U74-MC 拥有功能全面的 HPC. 因此本文对 U74-MC 的 HPC 进行了着重分析. U74-MC 的 HPC 可以分为两类: 第 1 类按照 RISC-V 特权指令规范, 将若干 CSR 作为 HPC 寄存器, 软件可以通过 CSR 指令配置和访问这些 HPC. 第 2 类专门用来统计 L2 的微体系结构事件, 这类 HPC 不会占用 CSR, 而是像外设一样通过 MMIO (Memory-Mapped I/O, 即内存映射) 与处理器核连接. 为了便于区分, 本文把第 1 类称为核内 HPC, 第 2 类称为外设 HPC.

3.1 核内 HPC

按照 RISC-V 特权指令规范[3], U74-MC 分配了 4 个 CSR 作为 HPC (这些计数器称为 mhpmcounter), 其中两个 (mcycle 和 minstret) 用于统计时钟周期和已执行指令数, 其余两个是可编程的 CSR, 可用于够捕获选定的事件, 并完成计数. 事件的选择由 mphevent 寄存器控制, 能够选中某一种或某几种事件. 为了安全起见, 用户级 (user-level) 的程序不能配置该寄存器, 也无法直接访问 mhpmcounter, 只能在机器级 (machine-level). 程序在 [m|s]counteren 中开启了对应的计数器访问权限后, 用户级程序才被允许通过访问 hpmcounter, 从而间接获取 mhpmcounter 中的存储的计数值. 即 hpmcounter 是 mhpmcounter 的影子寄存器 (shadow CSR).

如图 1 所示, 核内 HPC 的寄存器位于 CSR 模块内部, 事件信号来自不同模块. 因此这种结构的 HPC 拥有极低的访问延时, 软件通过 CSR 命令能够直接获取 HPC 数据. 虽然核内 HPC 结构简单, 便于实现且能够满足大部分应用场景, 但仍存在不足: 首先, 无法同时监测大量事件, 当监测事件大于核内 HPC 数量时, 只能由软件不断切换监测的事件类型. 这种方式不仅损害了性能, 同时还降低了测量精度. 其次, 计数器都集中在 CSR 区域, 而 (微) 体系结构事件却分布在如处理器核、缓存、直接存储器访问 (Direct Memory Access, DMA) 等各个模块中. 繁多的计数信号来源给前端设计和后端布线带来了巨大挑战.

3.2 外设 HPC

现代处理器内部拥有种类众多的模块: 处理器核、缓存、DMA 等, 将这些模块的事件信息全部汇总在核内 HPC 是很不明智的做法. 将同一模块的 HPC 集中在一起并作为外设连接到 MMIO, 这样就避免了核内 HPC 计数信号过于分散的问题, 而且通过访存指令 (LOAD/STORE) 就能对这些 HPC 进行控制和访问. U74-MC 的 L2 HPC 就使用了这种设计方来案监测 L2 缓存的 (微) 体系结构事件.

如图 2 所示, 外设 HPC 就像外设一样通过 MMIO

© 中国科学院软件研究所 http://www.c-s-a.org.cn

连接至处理器核, 由于 IO 空间范围较大, 此结构具有较好的拓展性. 外设 HPC 弥补了核内 HPC 缺陷, 能够同时监测更多的 (微) 体系结构事件. 但仍有一些不足: 首先, IO 空间无法对用户态下的程序直接可见, 访问前需要经过操作系统的映射, 从而产生不小的代价. 其次, MMIO 由诸多外设共享, HPC 数据可能被恶意外设获取, 进而被利用发起侧信道攻击, 引起难以监测的安全问题. 最后, 相比于 CSR 访问命令, LOAD/STORE 指令影响 L1D (一级数据缓存) 的缺失率, 即访问 HPC 会影响测量的准确性.

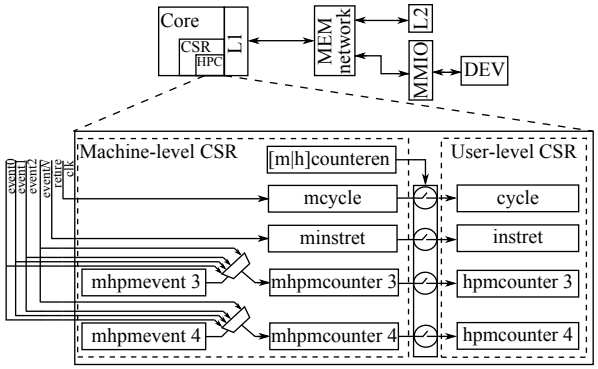

图 1 核内 HPC 结构

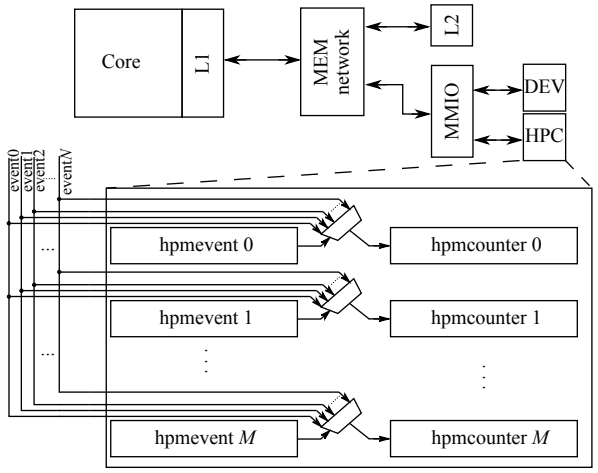

图 2 外设 HPC 结构

4 分布式性能计数器

针对以上不足, 本文提出一种分布式的性能计数器方案, 该方案充分结合了核内/外设 HPC 两者优点: 首先, 该结构拥有众多 HPC, 能同时监测上百种事件. 其次, 易于使用, 利用 3 个 CSR 就够控制和访问这些 HPC, 不影响 L1D 性能有更好的准确性, 不占用 MMIO, 因此不会带来地址映射的额外开销. 最后, 有着更好的安全特性, 能够通过 CSR 的权限控制管理 HPC 的访问, 从而避免了 HPC 被非法使用, 进而导致侧信道攻击.

4.1 结构

核内 HPC 结构模型中的每一个计数器都占据了 1 个 CSR, 当需要同时监测更多事件时, 就需要更多的 CSR. 如果将这些计数器从 CSR 中剥离出来, 根据其事件类型和触发信号的来源, 将其分布在不同模块中, 并通过片上互连, 将各个计数器和对应的 CSR 相连, 这样就可以通过使用少量的 CSR 来控制和访问数量庞大的 HPC, 按照这种策略, 本文设计了如图 3 所示的分布式性能计数器. 从图 3 中可以清晰地看出它由 3 个重要部分组成: HPCManager, HPCClient 和 HPCInterconnect. 而 HPC 的寄存器被分散在了 HPCManager 中.

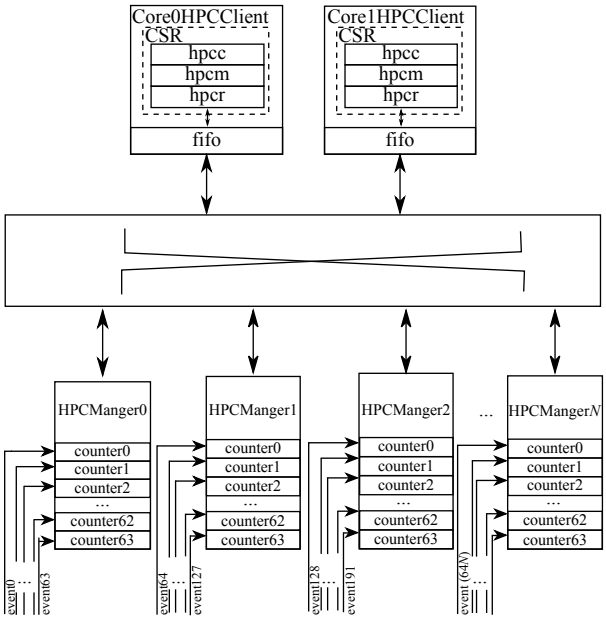

图 3 分布式性能计数器结构

HPCManager 主要负责接收和处理来自 HPCClient 的数据请求, 根据请求命令去查找指定的 HPC, 并把该 HPC 的数据返回给 HPCClient. 1 个 HPCManager 最多可以容纳 64 个 HPC, 这些 HPC 并无编程功能, 每个计数器只能够记录特定的 (微) 体系结构事件. HPC 按照所属模块进行划分, 比如处理器核中的事件统一划分给一个 HPCManager, L2 (二级缓存) 分片中的事件统一划分给另一个 HPCManager. 当模块中的事件超过 64 种时, 可为该模块分配多个 HPCManagerID. 显然, 相比于计数器集中在 CSR 中的方式, 这种按模块

© 中国科学院软件研究所 http://www.c-s-a.org.cn

去划分的分布式方案拓展性更强, 拥有更规整、更清晰的结构, 同时带给后端布线的压力也更小.

HPCClient 主要负责向 HPCManager 发起 HPC 数据请求以及接收返回的数据, 其结构如图 4 所示. 可以看出, HPCClient 位于处理器核中的 CSR 模块, 它占用了 3 个 CSR: hpcm (hardware performance counters bitmap), hpcc (hardware performance counters config) 以及 hpcr (hardware performance counters receive). 通过这 3 个 CSR, 软件能够获取到任意 HPC 的数据. 它们的详细内容将在本文的 4.2 小节展开.

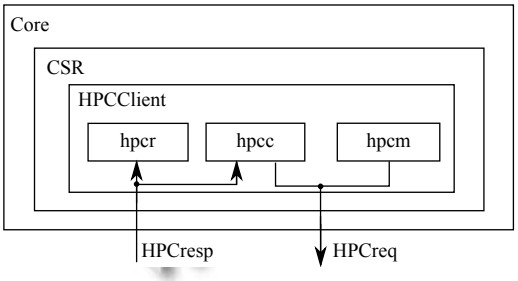

图 4 HPCClient 结构

HPCInterconnect 则把 HPCClient 和 HPCManager 连接在一起, 并且把这两种模块的命令、数据传递到指定位置. HPCInterconnect 采用循环优先级, 它保证了在多核结构中, 每个核对 HPCManager 都拥有相同的优先级, 在多核同时向同一 HPCManager 发起请求时, 可以有效避免“饿死”的情况.

4.2 性能计数器占用的 CSR

在 CSR 中定义了 3 个寄存器, 即 hpcc, hpcm, hpcr, 通过这 3 个寄存器来完成对 HPC 的管理和访问.

hpcc 即 hardware performance counters config. 其比特位分配情况如图 5 所示. 作为硬件性能计数器的功能配置寄存器, 该 CSR 功能包括: 向指定的 HPCManager 发起 HPC 请求、指示接收状态、指示软件读取是否失败等.

以下内容是依据图 5 对 hpcc 的功能划分进行说明:

(1) trigger: hpcc[0] ReadWrite

该位主要功能是发起和取消 HPC 请求. 如果此位为 0, 则表示当前没有正在进行的 HPC 请求, 这时写入 1 就可以触发一次新的 HPC 请求. 若此位为 1, 则表示有正在进行的 HPC 请求, 软件可以通过写入 0 的方式来撤销当前未完成的请求. 当 HPCClient 接收完所有指定的数据后, trigger 会自动复位, 表示完成了 1 次请求, 即所有需要的 HPC 数据均已存入 fifo.

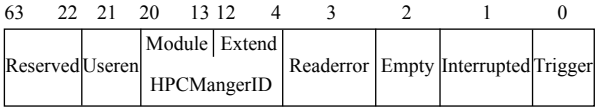

图 5 hpcc 寄存器

(2) interrupted: hpcc[1] ReadOnly

该位用于表示 HPC 在请求过程中, 是否发生过上下文切换. 当软件写 hpcm 时, 该位自动复位, 当上下文切换时, interrupted 自动置位, 表示 HPC 请求阶段可能出现数据错误, 需要重新发起请求. 软件只拥有该位的读取权限.

(3) empty: hpcc[2] ReadOnly

该位用于表示接收返回数据的 fifo 状态. 当 fifo 空时, 该位会被置位, 否则被复位. 当软件写 hpcm 时该位自动置位, 软件只拥有该位的读取权限. 利用 empty, 我们可以实现更快速地 HPC 读取, 当 empty 复位后, 软件就可以开始读取该数值, 而不必等到接收完 HPCManager 返回的所有数据 (此时 trigger 自动复位) 才开始读取工作. 因而把该位当作读取开始的判断依据, 能够节约一定的时钟周期.

(4) readerror: hpcc[3] ReadOnly

该位用于表示软件读取 HPC 时是否发生了错误. 当软件写 hpcm 时, 该位自动复位, 当软件从空的 fifo 中读出了数据时, 该位置位, 表示读到了错误的数据. 软件只拥有该位的读取权限.

(5) HPCMangerID: hpcc[20:4] ReadWrite

即 HPCManager 的 ID. 软件向该位写入正确的 ID 号, 从而在选中预期的 HPCManager 之后, 置位 trigger 就能向该 HPCManager 发起 HPC 请求. 为了增强扩展性, 该位无法独立工作, 需要与 hpcm 寄存器配合使用. 事实上, HPC 可以按照任意规则划分到 HPCManager, 本着“一种逻辑清晰的划分规则, 既能给使用者带来便利也能够减轻后端布线的压力”的思想, 本文采取按照“模块化”的划分策略, 将 HPCMangerID 分割成 module 和 extend 两部分, module 指的是处理器核 0, 处理器核 1, 末级缓存分片 0, 末级缓存分片 1 等模块. 而复杂模块内的事件数量通常会超过 64, 因此会使用 extend 对模块分配多个 HPCManagerID.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

useren: hpcc[21] ReadWrite

该位用于对用户级程序 HPC 访问权限的控制, 用户级的程序对该位只有读取的权限, 只有机器级或特权级下的程序能够对该位进行更改. 若此位为 1, 则用户级的程序能够发起 HPC 请求, 否则用户级的程序在置位 trigger 时会触发异常. 该位的置位复位可以由操作系统完成, 这样能够对任意进程进行 HPC 访问权限的控制. 避免了 HPC 数据被恶意程序利用, 有着很好的安全特性.

hpcm 即 hardware performance counters bitmap. 作为硬件性能计数器的位图选择器, 该寄存器的每一比特都对应 HPCManager 中的 1 个 HPC. 当 hpcc.trigger 处于复位状态时, 软件拥有 hpcm 的写权限, 否则软件只能读取该寄存器. 当 HPCClient 按照 HPCMangerID 指示向目标 HPCManager 发起请求时, hpcm 选定了要读取的 HPC, 同时该寄存器自动复位. 之后每接收到 HPCManager 返回的 1 个 HPC, 就将对应的比特位置位, 通过读取该位, 软件可以知晓已接收到的 HPC. 受限于寄存器最大位宽, 因此每次选取的 HPC 不能超过 64 个, 从而理论上, 1 个 HPCManager 最多管理 64 个计数器. 为不同模块分配单独的 HPCManager, 记录了不同模块内部的 (微) 体系结构事件, 目前支持的事件和对应的 hpcm 位图如表 2 所示. read 表示读取缓存的次数. readmiss 表示读取缓存, 但发生数据缺失的次数. write 表示向缓存写入数据的次数. writemiss 表示向缓存写入数据, 但是发生缺失的次数. writeback 表示将缓存中的数据写入下一级缓存或者内存中的次数.

表 2 记录的事件和其所在位置

hpcm	module (core)	module (L2分片)
[0]	I.read	read
[1]	I.readmiss	readmiss
[2]	D.read	write
[3]	D.readmiss	writeback
[4]	D.write	—
[5]	D.writemiss	—
[6]	D.writeback	—

hpcr 作为硬件性能计数器的接收寄存器, 软件只有该寄存器的读取权限. 通过读取该寄存器, 软件可以获得 HPC 数值. hpcr 是直接连接到 fifo 输出端口, 软件每读取一次 hpcr, fifo 就会弹出一个新的数据存入 hpcr 寄存器. 当 fifo 为空时就会把过时或错误的数据存入该寄存器. 发起 HPC 请求后, 软件可以在 hpcc.trigger 由 1 变 0 后才开始读取 hpcr (表示当前的 HPC 全部接收完毕), 也可以在 hpcc.empty 由 1 变 0 后立刻读取 (表示当前接收到有返回的 HPC). 在没有异常或例外的情况下, 两者得到的数值以及数值顺序完全相同, 且后者更节约时钟周期, 尤其在请求的 HPC 数量较多的情况中.

4.3 编程模型

在裸机环境 (bare metal environment) 中, 不存在多个线程对 HPC 控制权的争抢问题, 配置并访问 HPC 十分简单. 算法 1 适用于裸机环境下获取 HPC 数值: 行①使用 CSR 置位指令, 将 bitmap 写入 hpcm 寄存器, 其中 bitmap 代表选中 HPCManager 中的某几个 HPC. 行② id 用来选中 HPCManager, 置位 trigger 会触发对 HPCManager 的请求. 行④等待 HPCManager 返回数据 (empty 复位). 行⑤empty 复位后, 软件读取 hpcr 并将得到的 HPC 数据存入数组.

算法 1. 裸机环境 HPC 的访问

输入: HPCM 的 id 和 HPC 对应的 bitmap
输出: HPC 记录的数据

```
Function get_HPCs(id, bitmap):
① set_csr(hpcm, bitmap);
② set_csr(hpcc, id&trigger);
③ for(i=0; i<=hpcnumber; i++) {
④     while(get_csr(hpcc.empty));
⑤     hpcs[i]=get_csr(hpcr);
⑥ }
```

在多线程环境下, 可能会出现同一个处理器核中的多个线程同时向 HPC 发起请求. HPC 的获取需要 HPC 的 3 个寄存器相互配合, 当配置工作被其他线程打断, 则会出现意想不到的错误. 针对这种问题, 我们通过使用“软件锁”, 防止多个程序同时对 HPC 的 3 个寄存器进行控制和访问. 然而“软件锁”不仅实现复杂而且开销较大, 对测量精度有影响. 为此, 我们通过监测 hpcc.interrupted 位, 来判断 HPC 的请求程序是否发生过中断. 当系统中线程个数与中断次数均较少时, 这种方式的失败率可以忽略. 这种方法编程简单、易于实现, 不会给系统带来过多的额外开销, 更加适合我们的研究工作. 其具体算法如算法 2 所示: 行②复位 hpcc 寄存器, 终止可能正在进行的 HPC 请求. 行③至行⑧如算法 1, 获取 HPC 数据并存入数组. 行⑨判断上述操作是否出现上下文切换, 若出现过上下文切换, 则返回到行②处.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

算法 2. 多线程中 HPC 的访问

输入: HPCM 的 id 和 HPC 对应的 bitmap
输出: HPC 记录的数据

```
Function get_HPCs(id, bitmap):
① do {
②     clear_csr(hpcc.trigger);
③     set_csr(hpcm, bitmap);
④     set_csr(hpcc, id&trigger);
⑤     for(i=0; i<=hpcnumber; i++) {
⑥     while(get_csr(hpcc.empty));
⑦     hpcs[i]=get_csr(hpcr);
⑧     }
⑨ } while(get_csr(hpcc.interruped))
```

5 结果与分析

我们通过将本文设计的分布式硬件性能计数器, 部署在 lowRISC-v0.4 上, 其消耗的资源如表 3 所示. 可以看出本文的设计硬件开销较小: LUTs 和 Registers 分别占用了 1.66% 和 6.29%. 其中, HPCClient 使用寄存器相对较多, 其原因是用于接收 HPC 数据的缓冲区容量较大. lowRISC-v0.4 能够以 50 MHz 的频率工作在 Genesys2 开发板[12], 其工作频率并没有受到 HPC 的影响. 通过使用 OSD 模块, 我们实现了 lowRISC-v0.4 可以自动运行 SPEC CPU2006 的各个整数测试集, 在运行完一百亿条指令后, 采用算法 1 读取 HPC. 每个基准的测试用例结果的平均值如表 4 所示, 可以看出 L2 的读取次数等于指令缓存和数据缓存的缺失次数之和, 符合 L1 和 L2 包含关系. 结合表 3、表 4, 我们可以得出一个结论: 分布式的 HPC 能够以极少的硬件资源为代价, 实现了准确地记录时钟周期、执行指令数以及 L1 与 L2 各分片的读写次数和缺失次数.

表 3 性能计数器资源消耗情况

模块	Slice LUTs	Slice Registers	Block RAM
lowRISC-v0.4	52 108	31 422	50.5
HPCClient	540	1243	0
HPCManager (L2分片)	80	266	0
HPCManager (core)	151	461	0
HPCInterconnect	92	6	0
合计 (%)	1.66	6.29	0

表 4 SPEC CPU2006 (整数) 运行结果

事件	perlbench	bzip2	gcc	gobmk	hmmer	sjeng	libquantum	omnetpp	astar	xalancbmk
cycle	4.122E+10	2.368E+10	3.048E+10	3.397E+10	2.813E+10	2.730E+10	2.898E+10	3.727E+10	2.996E+10	4.379E+10
instret	1.084E+10	1.046E+10	1.047E+10	1.038E+10	1.056E+10	1.046E+10	1.061E+10	1.031E+10	1.060E+10	1.015E+10
L1I.read	1.369E+10	1.182E+10	1.256E+10	1.277E+10	1.292E+10	1.283E+10	1.137E+10	1.277E+10	1.232E+10	1.344E+10
L1I.readmiss	3.881E+08	2.378E+04	1.052E+08	3.690E+08	3.491E+05	2.310E+08	4.456E+03	5.642E+07	1.316E+06	1.529E+08
L1.Dread	3.107E+09	2.693E+09	2.833E+09	2.846E+09	4.218E+09	2.605E+09	2.639E+09	2.835E+09	3.276E+09	3.753E+09
L1D.readmiss	4.038E+08	1.459E+08	1.931E+08	1.897E+08	2.173E+08	1.586E+08	2.524E+07	3.942E+08	2.458E+08	4.484E+08
L1D.write	1.167E+09	1.049E+09	1.501E+09	1.198E+09	9.485E+08	8.691E+08	5.796E+08	7.037E+08	4.902E+08	5.996E+08
L1D.writemiss	7.579E+07	4.574E+07	9.363E+07	8.087E+07	5.180E+07	4.531E+07	1.369E+07	1.540E+07	2.528E+07	3.434E+07
L1D.writeback	1.188E+08	7.217E+07	1.135E+08	1.182E+08	6.694E+07	7.223E+07	9.205E+07	1.962E+07	7.880E+07	6.513E+07
L2.read	8.676E+08	1.917E+08	3.919E+08	6.395E+08	2.695E+08	4.350E+08	2.661E+08	4.660E+08	2.724E+08	6.357E+08
L2.readmiss	1.175E+08	1.082E+08	1.598E+08	1.054E+08	5.117E+07	5.079E+07	2.513E+08	3.609E+08	1.078E+08	3.592E+08
L2.write	1.188E+08	7.217E+07	1.135E+08	1.182E+08	6.694E+07	7.223E+07	9.205E+07	1.962E+07	7.880E+07	6.513E+07
L2.writeback	1.892E+07	4.791E+07	7.952E+07	3.746E+07	3.140E+07	1.910E+07	9.038E+07	6.506E+06	5.000E+07	2.839E+07

6 结论与展望

表 5 将分布式的 HPC 和 SiFive 公司的 U74-MC 处理器所包含的两类 HPC 进行对比: U74-MC 的核内型 HPC 结构简单安全性强, 但是在监测大量事件时, 需要依靠软件不断切换监测事件, 既增加了编程代码也降低了测量精度. U74-MC 的外设型 HPC 将计数器部署在 IO 空间, 因此有很大的拓展空间, 能够同时监测大量事件, 然而由于通过 MMIO 连接, HPC 数据可能泄露给恶意外设, 对处理器安全有一定影响. 而且外设型 HPC 需要 LOAD/STORE 指令访问, 这些指令对 L1D 的缺失率有一定影响. 本文设计的分布式 HPC 为每个事件都单独分配了计数器, 因而测量精度得到了保障. 根据事件种类将计数器部署在不同 HPCManager 中, 并利用独立的 HPC 互连将这 HPCManager 和处理器核中的 HPCClient 进行连接, 因此分布式 HPC 有着很大的拓展空间和很好的安全性. 除此之外该方案仅使用 3 个 CSR 就能访问所有的 HPC, 避免了对 CSR 资源的过多占用. 这些优点给我们之后的相关研究工作带来了极大的便利. 但仍存在诸多缺陷: 首先分布式的 HPC 只适用于 64 位处理器, 通用性较差. 其次精力

© 中国科学院软件研究所 http://www.c-s-a.org.cn

所限, 本文提出的方案没有经过严格测试无法保障稳定性, 只适合研究工作. 再次相较于商业处理器提供的众多监测事件, 本方案目前监测的事件种类少, 只统计了我们研究需要的事件. 这些不足均需要后续完善.

表 5 3 种 HPC 特点对比

对比项	U74-MC核内型	U74-MC外设型	本文的分布式
硬件结构	最简单	一般	一般
扩展性	弱	强	强
测量精度	低	一般	高
安全性	强	一般	强
编程复杂度	一般	简单	一般
通用性	良好	良好	差
事件种类	众多	众多	很少

参考文献

1 邹文. 基于硬件性能监视的性能测试技术研究 [硕士学位论文]. 长沙: 国防科学技术大学, 2004.

2 Waterman A, Asanović K. The RISC-V instruction set manual, Volume II: Privileged architecture. https://riscv.org/wp-content/uploads/2019/08/riscv-privileged-20190608-1.pdf. (2019-06-08) [2021-04-06].

3 SiFive, Inc. SiFive U74-MC core complex manual. https://sifive.cdn.prismic.io/sifive/f24c0f97-cd86-4a88-9f2d-af23e8e32a10_u74mc_core_complex_manual_21G1.pdf. [2021-04-06].

4 Kimmitt J, Song W, Bradbury A. Tutorial for the v0.4 lowRISC release. https://www.lowrisc.org/docs/minion-v0.4/. [2021-04-06].

5 Henning JL. SPEC CPU2006 benchmark descriptions. ACM SIGARCH Computer Architecture News, 2006, 34(4): 1–17. [doi: 10.1145/1186736.1186737]

6 Waterman A, Asanović K. The RISC-V instruction set manual, Volume I: Unprivileged ISA. https://riscv.org/wp-content/uploads/2019/12/riscv-spec-20191213.pdf. (2019-12-13) [2021-04-06].

7 王诲喆, 唐丹, 余子濠, 等. 开源芯片、RISC-V 与敏捷开发. 大数据, 2019, 7(4): 50–66.

8 余子濠, 刘志刚, 李一苇, 等. 芯片敏捷开发实践: 标签化 RISC-V. 计算机研究与发展, 2019, 56(1): 35–48. [doi: 10.7544/issn1000-1239.2019.20180771]

9 Asanović K, Avizienis R, Bachrach J, *et al*. The rocket chip generator. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.pdf. (2016-06-15) [2021-04-06].

10 Bachrach J, Vo H, Richards B, *et al*. Chisel: Constructing hardware in a Scala embedded language. Proceedings of the 49th Annual Design Automation Conference. San Francisco: Association for Computing Machinery, 2012. 1216–1225. [doi: 10.1145/2228360.2228584]

11 The open SoC debug documentation library. https://opensocdebug.readthedocs.io/en/latest/. (2017-06-09) [2021-04-06].

12 Genesys 2 FPGA board reference manual. https://reference.digilentinc.com/_media/reference/programmable-logic/genesys-2/genesys2_rm.pdf. (2017-08-24) [2021-04-06].

13 Eyerman S, Eeckhout L, Karkhanis T, *et al*. A performance counter architecture for computing accurate CPI components. ACM SIGARCH Computer Architecture News, 2006, 34(5): 175–184. [doi: 10.1145/1168919.1168880]

14 Intel. Dual-core update to the Intel Itanium 2 processor reference manual for software development and optimization revision 0.9. https://www.intel.cn/content/www/cn/zh/products/docs/processors/itanium/dual-core-update-itanium-2-processor-manual.html. [2021-04-06].

15 刘玉. 基于性能监测硬件支持的片上缓存资源管理技术 [博士学位论文]. 合肥: 中国科学技术大学, 2013.

16 Chetsa GLT, Lefèvre L, Pierson JM, *et al*. Exploiting performance counters to predict and improve energy performance of HPC systems. Future Generation Computer Systems, 2014, 36: 287–298. [doi: 10.1016/j.future.2013.07.010]

17 李晓虹. 基于行为的 Cache 攻击检测系统 [硕士学位论文]. 长沙: 湖南大学, 2018.

18 周巧凤. 基于性能计数器的 Cache 侧信道攻击检测研究 [硕士学位论文]. 北京: 北京交通大学, 2018.

19 吴金磊. 处理器核的性能分析及其分支预测结构优化 [硕士学位论文]. 长沙: 国防科学技术大学, 2016.

20 Singh A, Buchke A. A study of performance monitoring unit, perf and perf_events subsystem. http://rts.lab.asu.edu/web_438/project_final/CSE_598_Performance_Monitoring_Unit.pdf. [2021-04-06].

© 中国科学院软件研究所 http://www.c-s-a.org.cn