计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202111274
60(6): 1337−1345, 2023

一种基于持久化栈的返回地址预测器

谭弘泽　王　剑
（处理器芯片全国重点实验室（中国科学院计算技术研究所）　北京　100190）
（中国科学院大学　北京　100049）
（tanhongze20b@ict.ac.cn）

A Return Address Predictor Based on Persistent Stack

Tan Hongze and Wang Jian
（*State Key Lab of Processors* (*Institute of Computing Technology*, *Chinese Academy of Sciences*), *Beijing* 100190）
（*University of Chinese Academy of Sciences*, *Beijing* 100049）

Abstract　Branch prediction is an essential optimization for both the performance and power of modern processors, enabling instructions ahead of branches to be executed speculatively in parallel. Different from the general branch prediction, procedure return can be conquered with a return-address stack (RAS). By using a speculative emulation of the call stack according to the last-in-first-out rule for procedure calls and returns, the RAS predicts return addresses accurately. However, due to wrong-path corruptions under speculative execution of real processors, the RAS needs a repair mechanism to maintain the accuracy of the storage. Especially for embedded processors which are sensitive to the area, a careful trade-off between the accuracy and the overhead of repair mechanisms could be necessary. To address the redundancy of RAS storage, we introduce hybrid RAS, a return-address predictor based on a persistent stack. By integrating the classical stack, the persistent stack, and the backup prediction with the detection of overflows, our proposal could eliminate wrong-path corruptions and redundancies at the same time. As a result, the return misprediction rate is reduced effectively and efficiently. In addition, the classical stack is decoupled from the persistent stack to further optimize the area. With benchmarks from the SPEC CPU 2000 suite, the experiments show that our proposed RAS can reduce MPKI (mis-predictions per kilo instructions) to 2.4×10^{-3} with a design area of only 1.1×10^{4} μm^2 under design compiler, whose misses are reduced by over 96% compared with the state-of-the-art RAS.

Key words　return address prediction; speculative execution; corruption recovery; persistence; backup prediction

摘　要　分支预测允许处理器并行执行分支之后的指令，由于其高准确率具有性能和功耗方面的双重好处，是一项重要的处理器优化技术.根据分而治之的策略，返回地址栈（return-address stack, RAS）将过程返回类分支单独分出并予以预测.其中，RAS利用过程调用和返回的后入先出规则，可通过猜测执行中调用栈的模拟准确预测返回地址.但是，由于实际处理器猜测执行带来的错误路径污染，该结构需要通过恢复机制来保障所存储数据的准确性.尤其在对面积资源敏感的嵌入式领域，设计者需要在准确率和恢复机制的开销间进行细致的权衡.针对RAS存储中的冗余，通过溢出检测结合传统栈、持久化栈和后备预测3种预测方式，提出一种基于持久化栈的返回地址预测器——混合返回地址栈（hybrid return-address stack, HRAS），避免错误路径污染和对返回地址的冗余存储，从而有效降低返回误预测率.与此同时，设计

收稿日期：2021-12-24；**修回日期**：2022-08-08
基金项目：中国科学院战略性先导科技专项（C类）（XDC05020100）
This work was supported by the Strategic Priority Research Program of Chinese Academy of Sciences (XDC05020100).
通信作者：王剑（jw@ict.ac.cn）

解耦传统栈和持久化栈,进一步降低其面积需求.根据 SPEC CPU 2000 基准测试以及设计编译器的评估结果,HRAS 可利用仅 1.1×10^4 μm^2 的设计面积将过程返回的每千条指令误预测(MPKI)降至 2.4×10^{-3},其误预测相比现有 RAS 可降低 96%.

关键词 返回地址预测;猜测执行;污染恢复;持久化;后备预测

中图法分类号 TP332

分支预测普遍应用于现代 CPU 设计,为处理器提供对分支指令行为的猜测,允许处理器并行执行分支之后的指令,从而可提高其指令级并行性,是一项重要而传统的性能优化技术,至今仍受学术界和工业界的关注[1-3].一旦发生错误预测,处理器需要清除其指令执行路径在错误预测后的部分.这些需被清除的部分为错误预测路径,会浪费执行资源和功耗.高准确率的分支预测可在减少功耗浪费的同时保护指令级并行有效性,从而充分释放硬件资源的工作潜力,使处理器获得性能和功耗方面的双重好处.此外,由于超标量技术提升了处理器指令带宽,乱序执行技术需要通过指令调度窗口发挥作用,这些被普遍采用的技术进一步提高了现代处理器对分支预测的准确率要求.

在所预测分支当中,一类特殊的分支为过程返回,由于其返回地址遵循后入先出的顺序,具有高度的可预测性[4].通过对调用栈的模拟,其专用预测器将过程返回与其调用一一匹配,可有效利用其可预测性,从而被称为返回地址栈(return-address stack, RAS)[5].由于仅需按照先验规则匹配返回地址,无需学习有关分支模式的后验知识,该类预测器拥有理论准确率高和存储资源需求小的特点.

实际上,处理器环境限制了过程返回的预测准确率,导致 RAS 难以充分利用上述可预测性,所包含 4 类限制因素[6]如表 1 所示.其中,错误路径指处理器在错误分支后取出的多余指令,可产生对 RAS 的污染,导致过程返回预测错误.

1 相关工作

1.1 指 令

RAS 可通过指令识别函数调用并对函数返回进行预测.这依赖于现代指令集架构(instruction set architecture, ISA)在应用程序二进制接口(application binary interface, ABI)中为函数调用和返回定义了标准行为.例如,根据各自的 ABI 标准,RISCV[7]、OpenRISC[8]和 LoongArch[9]等架构都允许 RAS 通过指令识别函数调用并对函数返回进行预测.其中,各架构中指令对应的 RAS 操作如表 2 所示.本文不增加特殊指令,保证设计可以应用在各架构中.

Table 1 Limiting Factors of RAS Accuracy

表 1 RAS 准确率的限制因素

限制因素	原因	出现频率
错误路径污染	遗留了多余的修改	高
栈溢出	覆盖了有用的返回地址	高
上下文切换	破坏了栈匹配关系	低
非标准过程返回(如 longjmp)	损坏了调用栈层次	低

Table 2 RAS Operations of Instructions in Each Architecture

表 2 各架构中指令对应的 RAS 操作

指令集	函数返回	函数调用
RISCV	jalr x0,x1,0	jal
OpenRISC	l.jr r9	l.jal
LoongArch	jirl $r0,$r1,0	bl

1.2 功能分类

若按照功能进行分类,RAS 存在分支预测和安全检测 2 类应用方向:

1)分支预测方向较为传统,通常保证软件透明性,关注面积资源的利用.此类设计可通过后备预测[10]方法复用间接跳转预测取得性能提升,其涉及文献 [4−5,10−14].

2)安全检测方向主要将 RAS 用于栈溢出攻击的检测,其涉及文献 [6,15−21].在此方向,RAS 需要对比确认栈空间返回地址的完整性.其中,文献 [16−21] 都使用非猜测 RAS,其更新和存储与用于分支预测的 RAS相独立.文献 [6] 通过影子状态寄存器(shadow state register, SSR)的配合实现了同时允许分支预测和安全检测的 RAS. SRAS[17](secure return-address stack)和 RASE[6](return-address stack engine)等设计通过与内存交换数据消除栈溢出和上下文切换问题,但需要操作系统内核支持.

本文主要探讨用于分支预测的 RAS,不引入编译器和内核修改,保证设计便于工业界使用.

1.3 存储结构

若根据栈的存储结构进行分类，RAS 包含传统栈和持久化栈[22]2 类：

1）传统栈按照栈深度存储返回地址. 当传统栈弹出返回地址后，新写入的返回地址可覆盖其原有的内容，节约存储空间. 然而，猜测执行可能导致有效返回地址则被错误地覆盖，从而引起误预测污染. Skadron 等人[12] 和 Wang 等人[14] 提出了对传统栈地址被污染的修复方案，但在面积开销过大的同时仍有性能损失. Sun 等人[13] 通过在预测和提交阶段各存储一套返回地址，实现了返回地址修复，但在乱序处理器中无法准确修复返回地址.

2）持久化栈按照进入顺序存储返回地址. 与传统栈相反，持久化栈先弹出再写入的新返回地址不会覆盖原先栈顶，而是占用新的存储空间. 不同于传统栈，持久化栈的溢出是根据嵌套个数而非嵌套层数，即只要函数调用的嵌套个数超过容量就会发生溢出. Jourdan 等人[11] 所提出的设计直接使用持久化栈进行预测，虽然避免了误预测污染，但是遭受了更严重的栈溢出问题.

传统栈和持久化栈可以相互搭配. Park 等人[6] 利用 SSR 存储对 RAS 的修改，从而在允许 RAS 使用最新信息的同时避免了误预测污染.

1.4 对 比

表 3 总结对比了各 RAS 结构的特征，可分为 4 类问题.

Table 3 Comparison of RAS Structures

表 3 RAS 结构对比

结构	溢出	误预测污染	硬件存储开销	特殊要求
简单 RAS[4-5]	嵌套层数	直接压栈污染栈底； 先弹后压污染栈顶	栈返回地址、 栈顶指针 （基础开销）.	无
SCRAS[11]	间隔次数	污染栈底	栈顶指针、 每条分支需要栈指针.	无
CTRAS[12]	嵌套层数	直接压栈污染栈底； 多层先弹后压污染栈顶	栈顶指针、 每条分支需要若干返回地址.	无
DSRAS[13]	嵌套层数	无	双倍栈返回地址.	要求顺序执行
SARAS[14]	嵌套层数	污染栈底	栈顶指针、 每条分支需要返回地址.	恢复时暂停取指
SSR/RASE[6]	通过内存 扩展避免	无	栈顶指针、 每条分支需要返回地址和指针（SSR）.	需要操作系统管理栈的上下文、 内存空间和地址翻译（RASE）
非猜测 RAS[16-21]	嵌套变化	无	栈返回地址.	不可用于分支预测
HRAS（本文）	嵌套层数	无	栈顶指针每条分支需要栈指针， 其他开销可调节.	无

1）早期 RAS 设计[4-5,11-12] 受溢出和误预测污染影响，准确率表现不佳.

2）后续 RAS 设计[13-14] 提高了 RAS 的准确率，但对处理器提了更多的限制.

3）文献 [6] 通过 SSR 保证了 RAS 准确，但其 SSR 保存全部飞行指令返回地址的特性要求更大的存储空间.

4）非猜测 RAS[16-21] 使用独立与分支预测的存储空间，存在信息冗余.

为解决这 4 类问题，本文提出了一种基于持久化栈的返回地址预测器——混合返回地址栈（hybrid return-address stack, HRAS）. HRAS 对处理器不存在如文献 [13−14] 的限制问题. 类似于结合 RAS 和 SSR[6] 的设计，本文结合传统栈、持久化栈和后备预测[10]3 种预测方式，同时解耦传统栈和持久化栈 2 部分设计，不必保存全部飞行指令返回地址. 在此设计的基础上，本文定量分析返回地址误预测的来源，提出一种 RAS 同面积准确率的优化方式. 此外，传统栈部分非猜测更新的特性允许与用于安全检测的非猜测 RAS[16-21] 共用存储空间.

2 HRAS 结构设计

2.1 HRAS 结构总览

HRAS 包含提交栈（retired stack, RS）和猜测队列（speculative queue, SQ）2 个子模块，总体结构如图 1 所示. HRAS 通过将子模块预测与间接跳转预测相结合. 子模块预测通过组合可在及时更新的同时提供精确的错误路径恢复，间接跳转预测缓解了 RAS 的容量限制.

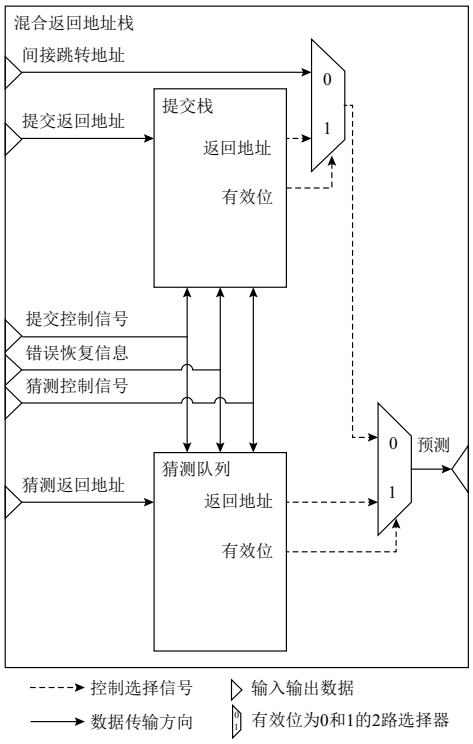


Fig. 1 The overall structure of HRAS

图 1 HRAS 总体结构

HRAS 根据处理器猜测和提交阶段的特点为子模块选择了高效的数据结构. 处于猜测阶段的指令可被随时取消. 当处理器取消部分指令时，为防止错误路径污染，RAS 需要将栈状态恢复为所取消指令的对应版本. 而可持久化栈能以最小的存储空间保留栈的中间版本并通过指针指示当前版本. 通过采用这一数据结构，SQ 可通过指针实现栈版本的精确恢复. 与此相反，提交阶段的指令总是精确的，不会被取消. 根据这一特点，RS 可采用传统栈结构，回收历史版本的存储资源.

2.2 RS

RS 存储已提交返回地址，采用传统栈结构，根据过程的调用和返回加减栈顶指针，通过将更新推迟到提交阶段避免错误路径污染，包含写栈顶(top of stack to write, TOSW)指针、读栈顶(top of stack to read, TOSR)指针、栈底(bottom of stack, BOS)指针、提交表和溢出判断逻辑 5 部分，其结构如图 2 所示.

为保证返回地址栈的准确性，RS 根据提交信息维护 TOSW 指针和提交表，当接收到过程调用信号时将提交返回地址写入提交表，利用提交阶段更新序列的准确性避免错误路径污染. 例如，当猜测阶段过程调用依次产生返回地址 *A*, *B*, *C*, *D*, *E* 时，RS 暂时不会向提交表写入返回地址. 若错误预测路径产生的返回地址 *D*, *E* 被取消，提交表只会写入通过提交阶段确定的返回地址 *A*, *B*, *C*, 从而避免误预测污染.

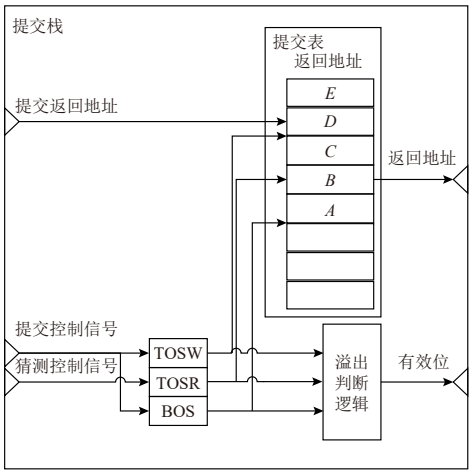


Fig. 2 The structure of RS

图 2 RS 结构

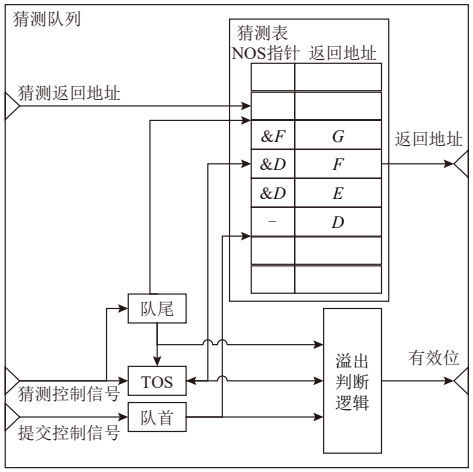
为在猜测阶段提供预测，RS 根据猜测控制信号更新选择 TOSR 指针，追踪猜测阶段的过程调用栈层次，根据 TOSR 指针读出用于预测的返回地址. 为清除错误路径污染，RS 根据错误恢复信息或 TOSW 指针恢复其 TOSR 指针，保证无猜测过程调用或返回时其读写指针相同. 只要中间实际执行的指令不会导致溢出，无论内部嵌套的过程调用或返回是否提交、无论中间有无错误预测路径，RS 总能根据 TOSR 指针读出正确的返回地址. 例如，错误预测路径产生的返回地址 *D*, *E* 被取消，而 TOSR 也被恢复到指向返回地址 *C*. 此时，无论 *C* 是否已提交，只要执行流再次返回到了需要返回地址 *B* 的位置，RS 就能根据 TOSR 提供所需返回地址.

RS 通过溢出判断逻辑检查其指针的相对位置关系，从而检测返回地址的有效性，在 SQ 无效的情况下为 HRAS 提供用于选择最终预测结果的有效位. 栈的最新和最旧有效元素分别由其 TOSW 指针和 BOS 指针指示. 其 BOS 指针根据栈的溢出和下溢出被相应加减. 二者形成的有效区间(BOS, TOSW]表示了有效返回地址范围，可用于溢出判断逻辑，根据 TOSR 指针判断返回地址有效性，从而在其失效时，HRAS 可借助间接跳转地址尝试修复返回地址.

2.3 SQ

SQ 存储未提交返回地址，采用支持精确恢复的持久化栈结构，通过增量更新维护了猜测执行中栈的各个版本，避免对旧有数据的修改，包含队尾指针、队首指针、栈顶[11](top of stack, TOS)指针、猜测表和溢出判断逻辑 5 部分，其结构如图 3 所示. 其中，队尾

指针、队首指针和猜测表组成一个循环队列，其元素可通过 NOS 指针链接为单向链表，而其 TOS 指针指向链式栈的当前栈顶.



“–”表示指针指向无效区域

Fig. 3 The structure of SQ

图 3 SQ 结构

由于猜测表仅存储未提交返回地址，其循环队列元素的生命周期从进入猜测执行开始到退出猜测执行结束. 在收到猜测过程调用信号时，为存储新的猜测返回地址，SQ 分配新的结点并将之入队，即在将队尾指针加 1 的同时，将结点写入原队尾指针指向的表项. 由于指令提交时顺序和取指时相同，结点遵守先入先出的顺序，从而，当收到提交过程调用信号时，SQ 通过将队首指针加 1 依次令队首结点出队. 此外，当复位或例外等事件发生时，由于处理器不再有任何未提交的指令，SQ 清空其循环队列.

循环队列中的链式栈表示了逻辑 RAS，通过 TOS 指针指示其栈顶，可提供用于预测的返回地址. 在取指阶段，当过程调用时，SQ 创建新的结点，并将 TOS 指针指向该结点. 其 NOS 指针指向原栈顶，即原 TOS 指针所指. 而当过程返回时，SQ 可根据 TOS 指针读出当前栈顶结点并将其返回地址用于预测，为实现弹栈操作，仅需将其 TOS 指针指回其读出的 NOS 指针. 例如，若 SQ 中当前链式栈栈顶返回地址为 *D*，而新的猜测返回地址为 *E*，则其新建结点存储返回地址 *E* 和指向 *D* 所在表项的指针 *&D*. 此后，过程返回可将新结点的 NOS 指针 *&D* 重新恢复到其 TOS 指针，从而重新读出正确的返回地址 *D*. 在弹栈后，SQ 队尾不变，栈顶返回地址恢复为 *D*.

SQ 在队列中分配结点，通过指针链接栈结构，实质上是一种持久化栈结构，可通过指针精确恢复其状态，无需恢复其猜测表数据，可节约功耗和硬件复杂度. 作为持久化栈，SQ 可通过 TOS 指针访问栈的中间版本，使得其找回逻辑栈版本只需恢复 TOS 指针. 与此同时，为保证取指时进入 SQ 的返回地址可在提交时按序退出，通过在分支误预测后恢复队尾指针，该设计可回收队尾的无效结点. 以图 3 为例，猜测表通过 4 个结点分别表示了(*D*)，(*E*, *D*)，(*F*, *D*)，(*G*, *F*, *D*)4 个版本的逻辑栈. 若从当前状态取消 1 次弹栈操作，SQ 仅需将 TOS 指针从 *&F* 恢复为 *&G*. 从而，其逻辑栈内容会被从(*F*, *D*)恢复为(*G*, *F*, *D*). 又如，若从当前状态取消 2 次压栈操作，SQ 仅需在恢复 TOS 指针的同时再将队尾指针从 *&G* 后恢复为 *&E* 后.

为提供返回地址的选择依据，SQ 通过溢出判断逻辑计算有效位. 其队尾指针和队首指针分别指示了循环队列的队尾和队首，标记了其有效区域. 有效位有效即 TOS 指针位于循环队列有效区域，可表明所读出返回地址尚未被提交. 其中，猜测表项数无需超过其有效区域最大项数，即处理器猜测执行的分支指令容纳数量. 然而，当猜测表项数较少时，循环队列新分配的队尾元素可能覆盖队首元素，导致返回地址失效. 为检测此类溢出，SQ 可通过额外引入的 Max 指针[10] 记录 SQ 队尾的覆盖位置，并在栈顶处于被覆盖但尚未被提交的情况下无效化 RS 的预测.

2.4 HRAS 结果选择

HRAS 最终预测结果选自 SQ、RS 和间接跳转预测器 3 处. 其选择根据预设优先级和子模块判断的有效性. 如果 SQ 的返回地址有效，那么当前栈层次的最新返回地址就是 SQ 的返回地址，从而选择逻辑会优先使用来自 SQ 的返回地址；否则当前栈层次的最新返回地址已提交，从而选择逻辑需要从 RS 获取返回地址. 然而，当二者无法提供有效的返回地址时，选择逻辑使用外部的间接跳转地址作为补充.

利用选择逻辑，HRAS 可通过猜测和提交 2 阶段更新信息相互补充，结合传统栈、持久化栈和间接跳转预测器 3 种结构的优点准确预测返回地址. 猜测更新具有及时性，但需要栈支持版本恢复. 借助于持久化栈结构，通过留存猜测阶段指令执行窗口中可被恢复的返回地址，SQ 支持了对栈的恢复. 注意到指令一经提交则不再会被取消，RAS 不再需要栈的历史版本，其额外留存返回地址的特性会浪费存储空间. 利用提交阶段的更新信号的准确性，RS 避免了对可恢复性的要求，从而可以采用更加紧凑的传统栈结构. 此外，当容量不足导致 HRAS 失效时，间接跳转预测器可根据失效的返回指令地址补充预测，从

而缓解深层次过程调用返回的 HRAS 失效.

3 实验及数据分析

3.1 实验配置

本文性能评估工作基于 EVE(emulation verification engineering)仿真加速器. 其中, EVE 仿真加速器是新思科技公司旗下一款硬件验证仿真加速平台, 已被全世界重要的半导体和电子系统公司所采用, 可通过挂载文件系统在操作系统中运行若干中小型测试程序[23]. 该平台可搭载 DDR3(double data rate 3)-1000 双通道内存, 仿真一款商业 CPU 核心, 启动 Linux 操作系统并运行基准测试, 从而准确地反映处理器结构和操作系统行为等实际因素对返回预测的影响, 并保证仿真时间可控.

该商业 CPU 核心基于 LoongArch 指令集, 其基本配置如表 4 所述, 可为本设计提供真实的处理器核环境.

Table 4 Basic Configuration Simulated by EVE

表 4 EVE 仿真基础配置

结构参数	配置
每周期获取指令个数	4
每周期发射指令个数	3
每周期提交指令个数	4
分支部件个数	1
定点部件个数	2
访存部件个数	2
浮点部件个数	2
重排序缓冲项数	128
分支队列项数	32
虚地址长度/b	40
指令 L1 高速缓存容量/KB	32
数据 L1 高速缓存容量/KB	32
共享 L2 高速缓存容量/MB	1
流水线级数	10
误预测惩罚时钟周期数	9

为反映实际应用负载中返回地址的预测准确率及其对性能的影响, 本文选择标准性能评估公司(Standard Performance Evaluation Corporation, SPEC) CPU 2000 基准测试, 在真机上预先编译其所有基准测试程序. 所用编译器版本为 GCC(GNU compiler collection) 8.3.0. 所用编译选项统一为“-Ofast -static”. 为节约测试时间, 本文主要对比其中过程返回次数最多的 9 个基准测试. 如无特殊说明, 下文中平均返回误预测率均指以上 9 个基准测试过程返回每千条指令误预测(mis-predictions per kilo instructions, MPKI)的算术平均值.

为评估获取各个基准测试程序的运行时间和返回 MPKI, 本文通过 perf 工具获取了运行时间、动态指令数、过程返回次数和过程返回误预测次数等性能事件计数. 其中, perf 是 Linux 操作系统下的一款众所周知的性能分析工具, 其基于处理器硬件计数器.

基于上述评估平台, 本文测试了实际场景中的 RAS 的 MPKI.

3.2 重要性分析

本节分析实际系统运行测试时返回 MPKI 对性能的影响, 从而说明返回预测对处理器性能的重要性.

图 4 分析了返回 MPKI 对处理器每周期指令数(instructions per cycle, IPC)的影响, 显示了返回地址预测对性能的重要性. 其中, 为了分析从包含随机因素的数据中提取 RAS 结构设计对性能的统计影响, 该分析采用了线性回归法. 实验表明, 当返回 MPKI 值在 0.2 以内时, 返回 MPKI 值每增加 1 可为该处理器的每周期指令数带来约 0.07 的损失. 即平均每次误预测约消耗 23 个时钟周期, 其平均延迟明显大于流水线深度. 由于乱序处理器中返回指令前后的其他指令存在并行性, 分支误预测的实际性能损失除分支的执行延迟外, 还包含所破坏的并行性. 若设计在误预测时检测栈溢出攻击, 每次返回误预测的开销可达 500 个时钟周期[16].

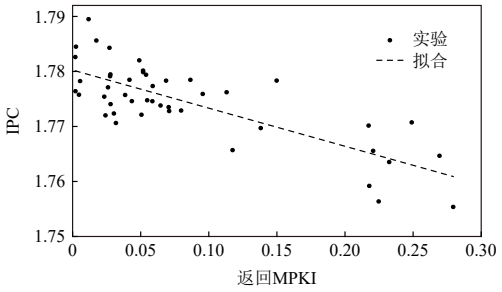


Fig. 4 IPC comparison on different return MPKI values

图 4 不同返回 MPKI 值的 IPC 比较

3.3 性能分析

首先, 本节对函数返回进行分类, 提出一种返回误预测的分解方式. 然后, 本节评估了 RS 项数和 SQ 项数分别对 2 类函数返回误预测的影响. 最后, 本节验证了所提分解方式并优化了 HRAS 的等面积性能.

函数返回指令可根据其对应调用指令提交的时间顺序分为长距离和短距离 2 种延迟类型. 其中, 长

距离返回指直到对应调用指令提交后才需要进行预测的返回指令，短距离返回反之. 在本设计中，长、短距离返回分别需要由 RS 和 SQ 进行预测. 若忽略返回地址预测对返回指令延迟类型的影响，各返回指令可被固定划分为长、短距离返回，从而其误预测率 M_{RS} 与 M_{SQ} 分别只与 RS 和 SQ 的项数 N_{RS} 和 N_{SQ} 相关. 因此，HRAS 的总误预测率 M 近似分解为误预测率 M_{RS} 与 M_{SQ} 之和，有

$$M(N_{RS}, N_{SQ}) = M_{RS}(N_{RS}) + M_{SQ}(N_{SQ}). \quad (1)$$

为提高等面积性能，HRAS 需要为 SQ 和 RS 合理分配存储空间，使得总项数 $N_{RS}+N_{SQ}$ 一定时总误预测率 M 最低. 根据式(1)分解关系，本文可分别测试 RS 项数和 SQ 项数与误预测的关系，并以此计算出最小化总误预测率的项数取值.

在 RS 项数固定为 32 的情况下，图 5 对比了不同 SQ 项数的返回 MPKI 值. 测试结果显示，SQ＝8 时即可正确猜测实际测试中的短距离返回，而项数更小的 SQ 会开始产生误预测，并在项数减少到 4 时产生的 MPKI 值为 0.05. 因此，根据评估结果，即使每 4 条飞行分支指令共享 1 项猜测表，SQ 都能给出正确的预测结果. 相比为每条分支指令存储 1 个返回地址的设计[6,12,14]，该设计只需 25% 的存储空间.

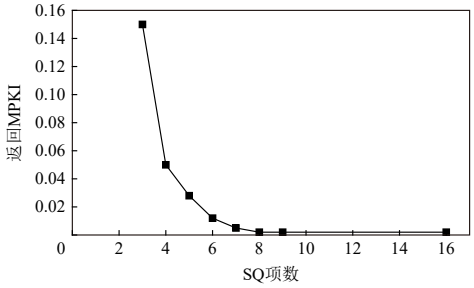


Fig. 5 Return MPKI values comparison on different numbers of SQ

图 5 不同 SQ 项数的返回 MPKI 值比较

在 SQ 项数固定为 16 的情况下，图 6 对比了 RS 不同项数的返回 MPKI 值. 结果显示，RS 项数的增加可有效抑制栈溢出，其返回 MPKI 值在项数为 32 时可降至 0.002.

配合图 5、6 的数据，本文通过测试其他配置的 MPKI 验证了式(1)的分解关系. 实验表明，在所测范围内，其预估的 MPKI 值相比于实际的 MPKI 值误差小 3%，远小于项数变化带来的 MPKI 变化，适用于项数配置的选取. 进而，本文估计了使得 MPKI 值最低的存储项数配置并验证了其 MPKI 符合预期. 所得各规模 HRAS 中 RS 和 SQ 的最优配置如表 5 所示.

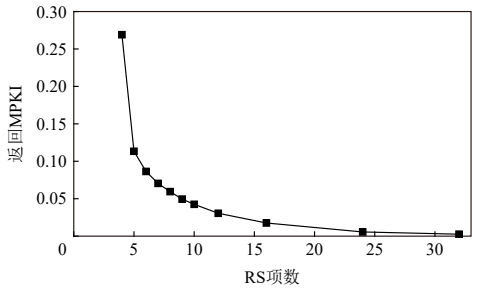


Fig. 6 Return MPKI values comparison on different numbers of RS

图 6 RS 不同项数的返回 MPKI 值比较

Table 5 Best Configures of RS and SQ on Different Sizes of HRAS

表 5 各规模 HRAS 中 RS 和 SQ 的最优配置

总项数	SQ 项数	RS 项数
10	4	6
12	6	6
14	6	8
≥20	8	≥12

为进一步分析 HRAS 产生误预测的原因，本文对仿真加速器波形进行了离线分析. 结果表明，所测系统在时钟中断时的上下文切换会污染 RAS. 若要消除此类误预测，RS 需要独立存储内核和用户的返回地址. 鉴于此类误预测的 MIPKI 值已经不足 0.001，本文没有分离内核和用户的返回地址存储.

3.4 性能对比

本节将 HRAS 与已知设计进行了性能对比.根据该处理器的条件，在现有设计中进行了筛选. SCRAS 相比 CTRAS，同面积时 MPKI 值更高[10]，无需重复测试. 由于本处理器集成时的适配问题，本文也不测试其他设计. HRAS 存在的问题为：

1)文献 [13] 要求处理器顺序执行分支指令；

2)文献 [14] 要求处理器支持暂停取指的功能；

3)文献 [6] 要求操作系统内核支持；

4)文献 [16−21] 不适用于分支预测.

综上所述，在本商用处理器核中可用的 SOTA 方案包括搭配了后备预测[10] 的简单 RAS[4-5] 和 CTRAS[12] 2 种设计. 本文根据文献描述对其进行了实现.

图 7 对比了 HRAS、简单 RAS 和 CTRAS 在项数同为 32 时的返回 MPKI. 在全部 SPEC CPU 2000 基准测试中，254.gap 是过程返回预测最为困难的测试. 在该测试中，HRAS 的 MPKI 值仅为 0.01，相比其他预测器降低了 90% 以上. 综上所述，HRAS 可有效利用过

程返回的可预测性,在任何测试中均比现有 RAS 设计具有更高的等面积准确率.

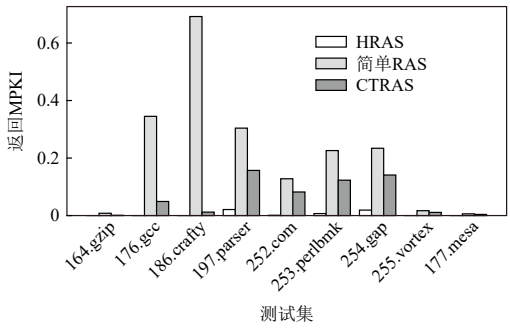


Fig. 7 Returning MPKI values comparison among different RAS implementations

图 7 各 RAS 实现的返回 MPKI 值比较

根据设计编译器的时序检查,RAS 设计均不影响处理器的时序关键路径,不会限制所测处理器的主频. 这主要基于访问延迟和流水线结构 2 方面因素. 在访问延迟方面,HRAS 与其他可用现有方案一样可并行进行溢出检测与返回地址读取,且相比处理器中物理寄存器堆等其他结构延迟明显较小;在结构方面,RAS 可提前将栈顶返回地址锁存到寄存器中用于预测,使得其访问延迟不影响外部结构.

3.5 开销对比

为准确评估 RAS 电路实现的面积,假设其目标频率为 2.0 GHz,基于设计编译器(design compiler, DC)以 28 nm 工艺综合了简单 RAS、CTRAS-1、CTRS-2 和 HRAS,并进行了面积评估. 而所评估项数介于 8~32 之间,为处理器中的典型项数[10].

图 8 对比了多种 RAS 设计在不同面积配置下的 MPKI. 其中,HRAS 采用如表 5 的最优配置. 在面积约为 4.5×10^3 μm² 时,HRAS 相比于简单 RAS,MPKI 可降低约 52%. 在面积约为 1.1×10^4 μm² 时,HRAS 相比简单 RAS 和 CTRAS,MPKI 可分别降低约 99% 和 96%. 对比结果表明,HRAS 在相同面积下具有最高准确率.

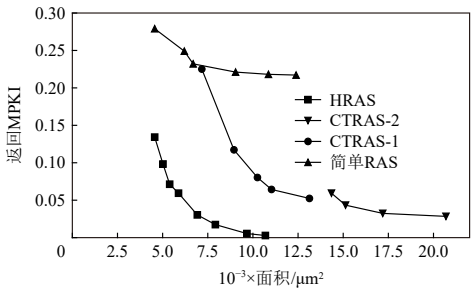


Fig. 8 Return MPKI values comparison on different areas of RAS implementations

图 8 各 RAS 实现在不同面积配置下的返回 MPKI 值比较

相比现有设计,HRAS 具有更好的可扩展性. 由于更激进的现代乱序处理器可容纳多达 100 条分支,设计应尽量减少每条分支固定的存储开销. HRAS 对处理器检查点的存储需求与简单 RAS 相近,无需为每条分支存储返回地址,可避免在处理器规模扩展时引入过多硬件资源开销. 此外,由于 RS 在提交时更新的特性,可以与用于栈溢出攻击检测的 RAS[16-21] 共享存储空间.

相比现有设计,HRAS 在同等准确率时系统开销更小. 相比文献 [13],HRAS 无需限制处理器分支指令的执行顺序,适用于现代的乱序处理器. 相比文献 [14],HRAS 避免了多周期修复带来的额外阻塞控制,简化了处理器的设计要求. 相比文献 [6],HRAS 由纯硬件实现,无需操作系统支持,相比本研究所用处理器的取指模块,即使将返回 MPKI 降至 0.01 以下,HRAS 仍仅需其面积的 4%.

相比其他设计,HRAS 在功耗方面没有明显缺点. 由于分猜测和提交 2 阶段存储,HRAS 在每次过程调用时需要 2 次写,而在每次过程返回时需要同时读 2 张表. 相比于简单 RAS,HRAS 虽然增加了动态功耗,但是其带来的准确率提高能够降低整个处理器核在错误路径上所消耗的能量. 相比于 CTRAS,HRAS 将对处理器检查点返回地址的读写换为了对 SQ 的读写,不会增加动态功耗.

以上分析和 DC 评估实验的结果表明,HRAS 在同准确率时面积最小,同面积时准确率最高,且具有可扩展性.

4 总结与展望

本文提出了一种混合返回地址栈(HRAS),其基于可持久化栈实现了精确恢复,结合传统栈结构实现了高效的返回地址存储,并基于实际的处理器核和操作系统测试环境对其进行了评估. 结果表明,该返回地址栈可消除错误路径污染,从而充分利用过程返回的可预测性,在 1.1×10^4 μm² 的设计面积下相比现有 RAS 降低 96% 的返回误预测,同面积每周期指令数相比带后备预测的简单 RAS 可提升 1%.

作者贡献声明:谭弘泽负责全文的设计和实现,以及文章的撰写和修改;王剑提出指导意见.

参 考 文 献

[1] Garza E, Ajorpaz S M, Jimenez D A, et al. Bit-level perceptron prediction for indirect branches [C] // Proc of the 46th Annual Int Symp on Computer Architecture. New York: ACM, 2019: 27−38

[2] Zangeneh S, Pruett S, Lym S, et al. BranchNet: A convolutional neural network to predict hard-to-predict branches [C] //Proc of the 53rd Annual Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 2020: 118−130

[3] Adiga N, Bonanno J, Collura A, et al. The IBM z15 high frequency mainframe branch predictor industrial product [C] //Proc of the 47th Annual Int Symp on Computer Architecture. New York: ACM, 2020: 27−39

[4] Kaeli D R, Emma P G. Branch history table prediction of moving target branches due to subroutine returns [C] //Proc of the 18th Annual Int Symp on Computer Architecture (ISCA). New York: ACM, 1991: 34−42

[5] Desmet V, Sazeides Y, Kourouyiannis C, et al. Correct alignment of a return-address-stack after call and return mispredictions [C] //Proc of the 32nd Annual Int Symp on Computer Architecture (ISCA). New York: ACM, 2005: 25−33

[6] Park Y, Lee G. Repairing return address stack for buffer overflow protection [C] //Proc of the 1st ACM Int Conf on Computing Frontiers. New York: ACM, 2004: 335−342

[7] Waterman A, Asanovic K. The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213 [S/OL]. 2019 [2022-04-08]. https://riscv.org/technical/specifications/

[8] Lampret D. OpenRISC 1000 Architecture Manual [S/OL]. OpenRISC Community, 2022 [2022-04-08]. https://openrisc.io/

[9] R&D Department of Chip. The LoongArch Reference Manual, Volume I: Basic Architecuture [S/OL]. Beijing: Loongson Technology Corporation Limited, 2021[2022-04-08]. https://www.loongson.cn/(in Chinese)
(芯片研发部. 龙芯架构参考手册, 卷1: 基础架构 [S/OL]. 北京: 龙芯中科技术股份有限公司, 2021 [2022-04-08]. https://www.loongson.cn/)

[10] Vandierendonck H, Seznec A. Speculative return address stack management revisited[J]. ACM Transactions on Architecture and Code Optimization, 2008, 5(3): 15:1−15:20

[11] Jourdan S, Stark J, Hsing T, et al. Recovery requirements of branch prediction storage structures in the presence of mispredicted-path execution[J]. Internatinal Journal of Parallel Programming, 1997, 25(5): 363−383

[12] Skadron K, Ahuja P S, Martonosi M, et al. Improving prediction for procedure returns with return-address-stack repair mechanisms [C] // Proc of the 31st Annual Int Symp on Microarchitecture (MICRO). Piscataway, NJ: IEEE, 1998: 259−271

[13] Sun Caixia, Zhang Minxuan. Dual-stack return address predictor [C] // Proc of the 1st Int Conf on Embedded Software and Systems (ICESS). Piscataway, NJ: IEEE, 2004: 172−179

[14] Wang Guopeng, Hu Xiangdong, Zhu Ying, et al. Self-aligning return address stack [C] // Proc of the 7th IEEE Int Conf on Networking, Architecture and Storages (NAS). Piscataway, NJ: IEEE, 2012: 278−282

[15] Aleph One. Smashing the stack for fun and profit [J/OL]. Phrack Magazine, 1996 [2022-04-09]. https://phrack.org/issues/49/14.html

[16] Ye Dong, Kaeli D. A reliable return address stack: Microarchitectural features to defeat stack smashing[J]. SIGARCH Computer Architecture News, 2005, 33(1): 73−80

[17] Xu Jun, Kalbarczyk Z, Patel S, et al. Architecture support for defending against buffer overflow attacks, UILU-ENG-02-2205, CRHC-02-05[R/OL]. Urbana, America: Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 2002 [2022-04-08]. http://hdl.handle.net/2142/74493

[18] Ozdoganoglu H, Vijaykumar T N, Brodley C E, et al. SmashGuard: A hardware solution to prevent security attacks on the function return address[J]. IEEE Transactions on Computers, 2006, 55(10): 1271−1285

[19] Alam M, Roy D B, Bhattacharya S, et al. SmashClean: A hardware level mitigation to stack smashing attacks in OpenRISC [C/OL] // Proc of the 14th ACM/IEEE Int Conf on Formal Methods and Models for System Design (MEMOCODE). Piscataway, NJ: IEEE, 2016 [2022-04-04]. https://ieeexplore.ieee.org/document/7797764

[20] Bresch C, Hély D, Papadimitriou A, et al. Stack redundancy to thwart return oriented programming in embedded systems[J]. IEEE Embedded Systems Letters, 2018, 10(3): 87−90

[21] Li Jinfeng, Xu Qizhen, Li Yongyue, et al. Efficient return address verification based on dislocated stack[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(11): 3398−3407

[22] Driscoll J R, Sarnak N, Sleator D D, et al. Making data structures persistent[J]. Journal of Computer and System Sciences, 1989, 38(1): 86−124

[23] Wu Ruiyang, Wang Wenxiang, Wang Huandong, et al. Design of Loongson GS464E processor architecture[J]. SCIENTIA SINICA Informationis, 2015, 45(4): 480−500 (in Chinese)
(吴瑞阳, 汪文祥, 王焕东, 等. 龙芯GS464E处理器核架构设计[J]. 中国科学: 信息科学, 2015, 45(4): 480−500)

Tan Hongze, born in 1996. PhD candidate. Student member of CCF. His main research interest includes computer architecture.

谭弘泽, 1996 年生. 博士研究生. CCF 学生会员. 主要研究方向为计算机系统结构.

Wang Jian, born in 1971. PhD, professor. Senior member of CCF. His main research interests include processor micro-architecture, software-hardware co-designed virtual machine, and operating system.

王 剑, 1971 年生. 博士, 正高级工程师. CCF 高级会员. 主要研究方向为处理器微体系结构、软硬件协同设计虚拟机和操作系统.