中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

A generic Kprobe optimization algorithm for RISC architectures

中科院计算所
陈国凯
chenguokai21sATict.ac.cn
GitHub：chenguokai

背景

- Kprobe
  - Linux Kernel 提供的插桩机制
  - 允许在内核大部分区域设置桩点并注册该点运行前后handler函数
  - 可用于收集性能和调试信息
    - 函数调用次数、调用时间分布
    - 打印桩点寄存器值
  - 实现
    - 基础：在桩点插入断点指令，复用内核已有异常处理通路保存现场，kprobe 主要负责调用 handler
    - **优化**：构造 detour buffer，其内代码负责保存现场、调用 handler、恢复现场；桩点断点指令替换为跳转到 detour buffer 的跳转指令

背景

- 性能开销[1]
  - 理论
    - 断点指令：现代处理器中分支预测器通常不对断点指令做预测，检测到断点指令需要冲刷流水线
    - 间接跳转指令：可用 BTB、ITTAGE[2] 等预测器以较高准确率预测其跳转地址，允许更准确的指令预取
  - 实际
    - On a typical CPU in use in 2005, a kprobe hit takes 0.5 to 1.0 microseconds to process.
    - Typically, an optimized kprobe hit takes 0.07 to 0.1 microseconds to process.

[1] JIM KENISTON P S P, MASAMI HIRAMATSU. Kernel Probes (Kprobes) [Z].

[2] Seznec A. A 64-Kbytes ITTAGE indirect branch predictor[C]//JWAC-2: Championship Branch Prediction. 2011.

优化方案概览

- 内核态
  - 单条指令跳转
  - 多条指令跳转
  - 利用寄存器暂存跳转地址
    - 利用 caller-saved 寄存器加载跳转地址
    - 利用特权态寄存器交换出空闲通用寄存器
    - 压栈通用寄存器
    - 提议方法
- 用户态
  - JIT 改写 binary
  - 指令直接编码地址

可能方案分析 – 内核态

- Kprobe现有实现-单指令跳转
  - x86 ：直接构造带有长立即数的跳转指令
    - RISC **不支持直接**构造该类指令
  - ARM/PowerPC ：使用单条 branch 指令跳转到 detour buffer
    - 同属 RISC，跳转范围相对较大（ ~ **32M** ），但在**近未来也将受到限制**
  - RISC-V* ：目前仅有基础的断点指令实现，**缺少优化版实现**
    - branch/jalr 指令跳转范围 ~ **8KB** jal 指令跳转范围 ~ **2MB**
    - 参考指标：64位defconfig kernel text 段大小**超过2MB**
    - 无法用**单条跳转**指令向 detour bufffer 跳转

*：为简化叙述及与实现保持一致，后续描述默认基于RISC-V指令集

可能方案分析 – 内核态

- 利用多条 jal 指令多次跳转到 detour buffer
  - 开销大，收益少->patch 指令数量随地址范围增大线性增长

    RISC-V 若要实现 32M 范围内跳转，需要寻找空闲空间并 patch 至少 16 条指令
- 利用 caller-saved 寄存器做空闲寄存器
  - 限制大：只能用于函数起始，不可用于函数体内

    函数体内 caller-saved 寄存器也可能已被使用
- 利用特权态空闲 CSR 与通用寄存器交换出空闲寄存器
  - RISC-V 标准未为 supervisor 态定义除 sscratch 外空闲 CSR
    - sscratch 在内核中已用于存储内核栈指针
  - 架构间通用性弱
- 压栈得到空闲寄存器
  - 需替换更多(3-4条)指令，状态更加复杂

可能方案分析 – 用户态

- 插桩工具[1]
  - Intel Pin (x86, AMD64, ARM[2])

  incrementally recompiles and executes code, by copying the compiled code, instrumenting the copies, and then executing the instrumented copy.
  - LiteInst (only for x86-64)

  LiteInst attempts to address the probe suitability issue with the instruction punning algorithm. This algorithm uses jump instructions as probes, and the probes lead execution to corresponding trampolines. Each jump probe has 5 bytes: 1 byte for the opcode of (relative) jmp, and **4 bytes to encode the jump offset that leads to a trampoline**.
  - MAMBO (ARM) [3]

  Just-in-time compilers; Dynamic compilers

[1] Zhao V. Evaluation of dynamic binary instrumentation approaches: Dynamic binary translation vs. dynamic probe injection[J]. 2018.

[2] Hazelwood K, Klauser A. A dynamic binary instrumentation engine for the arm architecture[C]//Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems. 2006: 261-270.

[3] Gorgovan C, d'Antras A, Luján M. MAMBO: A low-overhead dynamic binary modification tool for ARM[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2016, 13(1): 1-26.

可能方案 – 用户态 cont.

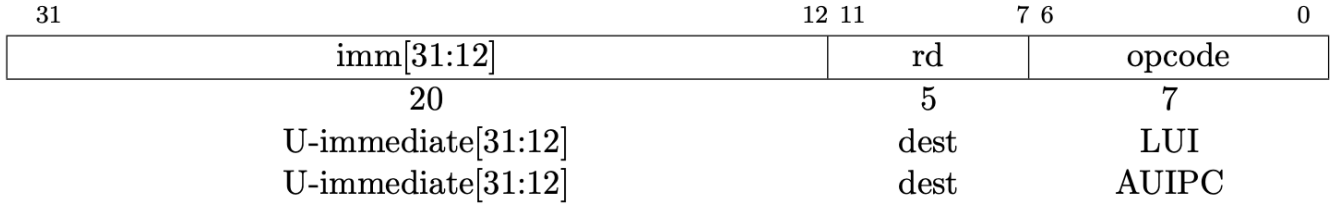
- 移植难点
  - JIT 方案对内核动态修改过大，实现工程量巨大
  - 构造单一跳转指令方案不适于 RISC

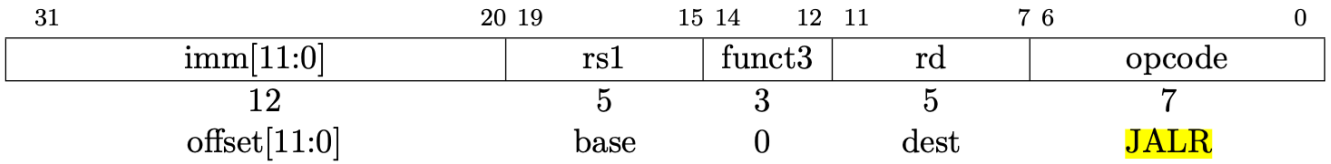
提议方法

- 使用 auipc+jalr 指令对跳转
  - auipc 加载 imm 高 20 位，jalr 加载 imm 低 12 位，共计 32 位立即数可用于 offset 计算～4GB 跳转范围
  - 需要使用寄存器暂存 imm：作为 auipc 的 rd 和 jalr 的 rs1

    auipc **tmp_reg**, %hi(offset)

    jalr x0, %lo(offset)(**tmp_reg**)
  - 该寄存器不能影响 kernel 正常执行流





提议方法

- 问题转化：在任意指令流寻找空闲的临时寄存器

  “In some cases it might be possible to use dead registers, that is not always the case. [1] “

  Proven to be non-trivial!

- 假相关
  - 使用同一寄存器但操作数据间不存在依赖
  - 产生原因
    - 循环体结构
    - 变量数量多于可用寄存器数量

- WAW

```
addi a0, a1, a2
xor t0, t0, t0
sub t1, t1, t2
addi a0, a3, a4
```

- WAR

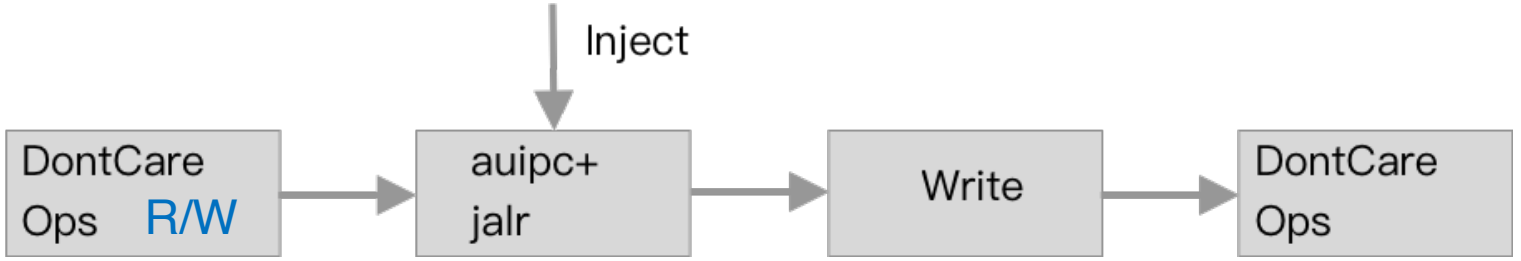
```
addi a1, a0, a2
xor t0, t0, t0
sub t1, t1, t2
addi a0, a3, a4
```

两种假相关

1. Gorgovan C, d'Antras A, Luján M. MAMBO: A low-overhead dynamic binary modification tool for ARM[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2016, 13(1): 1-26.

提议方法 cont.

- 基于假相关空闲寄存器寻找算法
  - 思想：逻辑寄存器数目有限，存在大量 WAW、WAR→该类寄存器在 W/R 操作后、W 前的值不重要。
  - 硬件视角：固有可利用的指令级并行
  - 软件视角：可利用的空闲寄存器



  - 从起始位置向后寻找不曾作为源寄存器的目的寄存器，则其为一空闲寄存器

提议方法 cont.

- 算法技术细节
  - 寄存器寻找终止条件
    1. branch、jal 指令：理论上可以继续分析*，实际收益不大
    2. jalr 指令：目标不定，不可继续分析
    3. 特权类指令：不可继续分析（可能触发异常，本身也不应进入寄存器分析阶段）
  - RISC-V 的 0 号寄存器恒 0，不作为结果返回

*：branch需要分析其两条路径空闲寄存器取交集

Kprobe优化实现细节

- 可优化点
  - 通用要求[1]
    - 被跳转指令替换的区域（被优化区域）完全位于一个函数内（跨越函数边界则很可能干扰正常执行流）
    - 函数不能含有间接跳转（不含跳转地址不定的指令）
    - 函数不能含有会触发异常的指令（处理代码可能跳转到优化区域内部）
    - 函数内不允许跳转到被优化区域非开头位置（不能从桩点内部执行，会破坏正常执行流）
  - 提议方案额外要求
    - 为简化逻辑，要求指令必须可以直接异位执行

[1] JIM KENISTON P S P, MASAMI HIRAMATSU. Kernel Probes (Kprobes) [Z].

Kprobe优化实现细节 cont.

- 指令流劫持实现细节
  - 进入 detour buffer：利用**桩点处**空闲寄存器作为临时寄存器利用auipc+jalr加载当前桩点到 detour buffer 起始地址的偏移量
  - detour buffer 内：
    - 通用寄存器与 supervisor 态 CSR 压栈保存现场
    - 调用 kprobe handler
    - 恢复各寄存器值，还原现场
    - 执行桩点处原本两指令
    - 回到桩点后继续执行
  - 返回桩点之后：需利用**桩点后**位置的空闲寄存器作为临时寄存器加载 detour buffer 末尾到桩点后的偏移量
- 指令热修改
  - 内核内原有接口一次仅能替换一条指令（至多 4 byte）
  - 使用 stop_machine 机制实现替换8 byte功能来确保两条指令替换原子性

Kprobe优化实现细节

- detour buffer 构建
  - 预先实现保存、恢复现场代码
  - 留空部分在注册 kprobe 时动态填充
    - 桩点位置指令留空
    - kprobe 结构体地址留空
    - kprobe 处理函数地址留空
    - 从detour buffer 返回 auipc+jalr 寄存器号留空

Kprobe优化实现细节 cont.

- Kprobe指令分类

指令分类	特性	举例
可直接执行指令	指令所在PC地址与执行效果无关，可直接放置于detour_buffer	运算类指令、访存类指令
需要模拟执行指令	指令执行效果与所在 PC 有关，但可模拟执行	分支指令、auipc 指令
不可出现在 Kprobe 点指令	指令执行效果与所在 PC 有关且无法模拟	特权类指令

- 原始 Kprobe 实现
  - 只替换 1 条指令，指令种类有限，允许替换可直接执行指令及需模拟执行指令
- 优化 Kprobe 实现
  - 需替换 2 条指令，指令间存在较复杂关系组合，仅允许替换 2 条可直接执行指令可避免复杂关系分析

Kprobe优化实现细节 cont.

- 可能出现的指令组合

指令组合	特性
2 条指令均可直接执行	与当前实现一致
第 1 条指令可直接执行，第 2 条指令为跳转指令	第 1 条指令也需要模拟执行*
第 1 条指令为跳转指令，第 2 条指令可直接执行	第 2 条指令需要存放于 inst_slot，根据第一条指令执行结果选择性执行**
2 条指令都为跳转指令	均需要模拟执行

*：现有机制不允许直接执行指令（需要已经恢复现场）后再次进行模拟操作

**：inst slot 本身使用 ebreak 标记指令结束，会拉低执行效率

RISC-V 64上技术问题

- RV32 总地址空间 4GB
  - 任何情况下detour buffer地址均位于4GB范围内
- RV64 总地址空间大于 4GB
  - Linux 当前分配到的地址距离 kernel text 段范围超过 4GB→无法直接跳转
  - patch alloc_insn_page() 确保获取到的内存地址在 4GB 范围内[1]

[1] https://lore.kernel.org/lkml/mhng-0e2f7f49-2680-4341-83dc-0e7cd042a3fa@palmer-mbp2014/T/

通用性

- 普遍适用于 RISC 指令集

ISA	Absolute	PC-relative
MIPS	lui+jalr	auipc+jalr (R6 only)
ARM32	mov+movt	ldr+bx
ARM64	mov+movk+br	ldr+br
LoongArch	lui12i.w+jirl	pcaddu12i+jirl
RISC-V	lui+jalr	auipc+jalr

加载跳转地址的指令组合

优势分析

- 相对直接替换分支指令
  - 有效增加了可能的跳转范围（2/32M->4G），令这一优化算法实用性更强
- 相对二进制翻译技术
  - 实现简单，对原始二进制修改少且局部性强

评估

- 可行性-空闲寄存器存在比例
  - 对 Linux Kernel 64 bit defconfig 做静态分析，单点存在概率约 68.3%，auipc+jalr 返回需要间隔 8 byte 双点，总体成功率约 51%
  - 实践中函数起始一般能够找到
  - 实践策略分析
    - 使用统一寄存器完成进入/返回 detour buffer
      - 总体成功率将降低到 30%
    - 遇到 branch、jal 指令沿指令流继续分析
      - 成功率提升上限 4%

评估

- 效率提升
  - XiangShan Nexus-AM 裸机环境进行的定性测试
    - 1000 iterations (近似1000 probes)
  - 测试方法
    - 在正常的workload中插入 ecall 或 jalr 指令分别模拟正常与优化版 Kprobe
    - ecall 使用 Nexus-AM 的标准异常处理路径
    - jalr 指令跳转到模拟 detour buffer，模拟保存恢复上下文等操作

	Inst	Cycle	IPC	Cycle Diff (Inst - base Inst)
base	3,109,157	2,596,590	1.197	N/A
ecall (近似ebreak)	3,388,160	2,807,023	1.207	210433
jr	3,288,160	2,699,749	1.218	103159

主线进展

- v2 版本补丁
  - http://lists.infradead.org/pipermail/linux-riscv/2022-September/019398.html

可能改进点

- 评估处理 RVC 扩展
  - 2 byte/4 byte 指令替换边界问题相对复杂
- 允许模拟执行分支指令

致谢

- 感谢开源软件供应链点亮计划对本项目的支持
- 感谢 OpenEuler 社区特别是廖畅老师对方案设计细节的指导

Q & A

Wechat : Xim-1999