MANCHESTER 1824
The University of Manchester

An Analysis of Call-site Patching Without Strong Hardware Support for Self-Modifying-Code

Tim Hartley, Foivos Zakkak,
Christos Kotselidis, Mikel Lujan

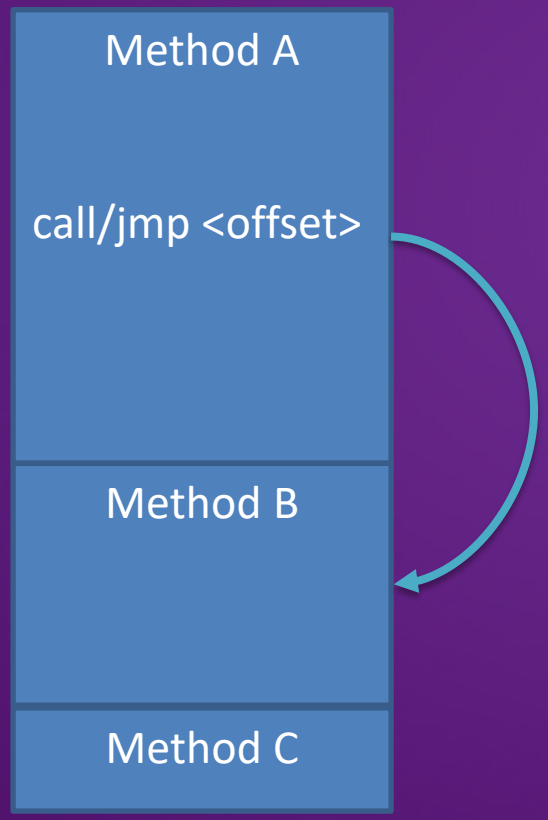
first.last@manchester.ac.uk

MPLR'19 2019-10-22

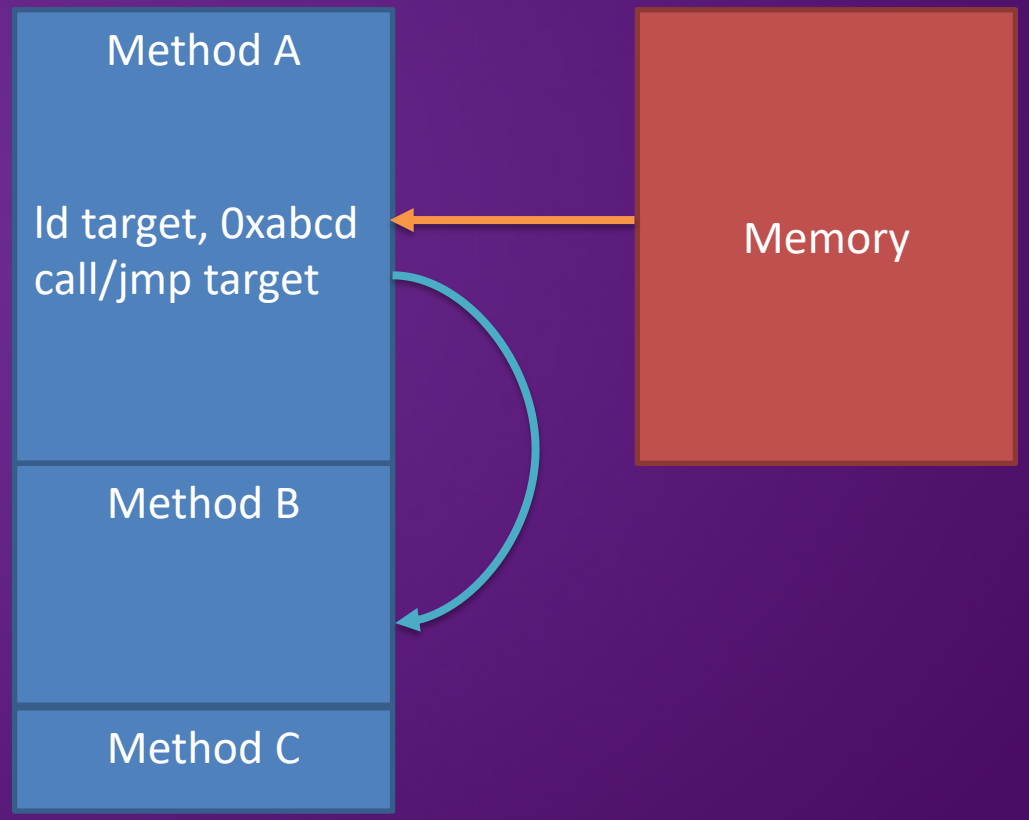
MANCHESTER 1824
The University of Manchester

Call-Sites

Direct branching



Indirect branching



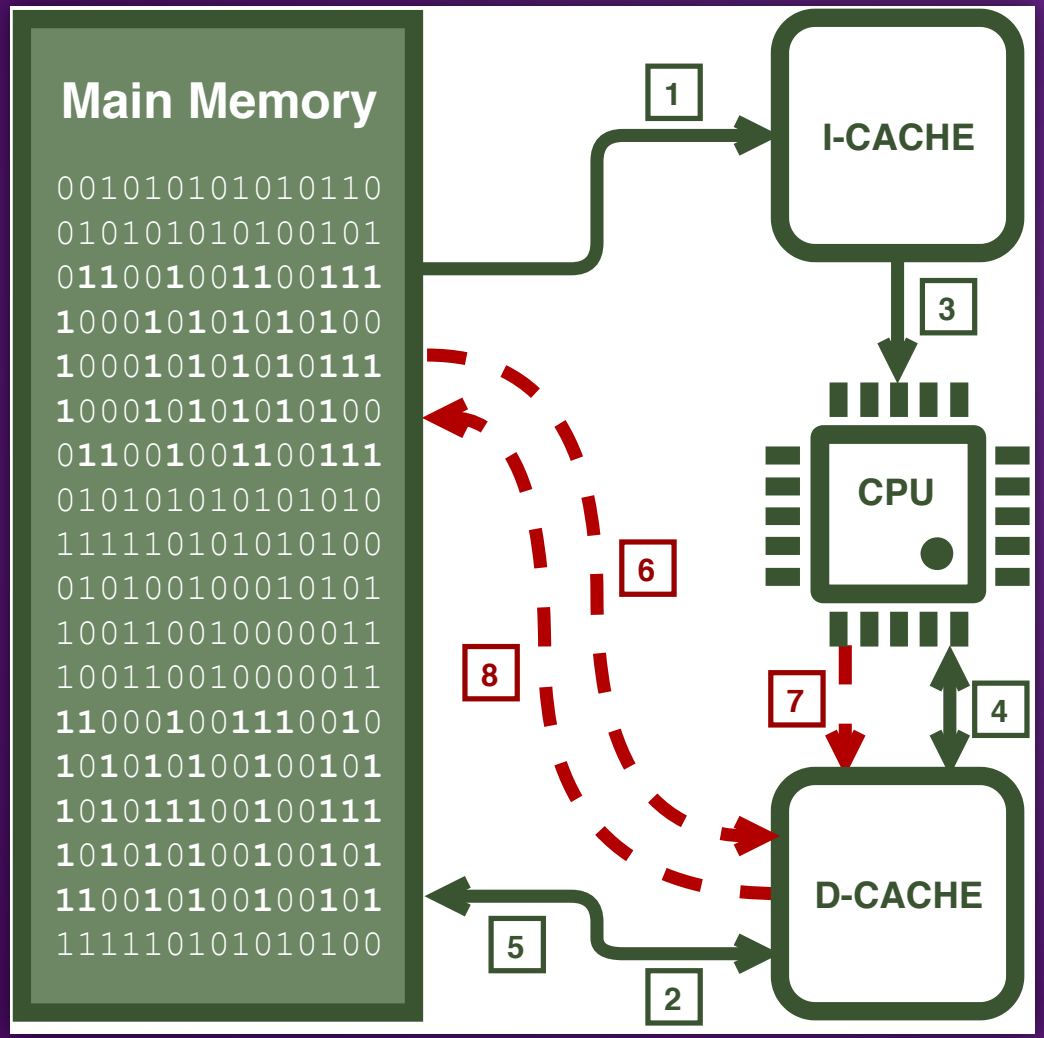
MANCHESTER 1824
The University of Manchester

Call-Site Patching

- Tiered compilation
- De-optimization
- Etc.

MANCHESTER 1824
The University of Manchester

JIT compilation and Caches



Code-stream vs Data-stream

1. Code gets fetched to I-Cache
2. Data get fetched to D-Cache
3. CPU executes code from I-Cache
4. CPU writes data to D-Cache
5. D-Cache writes-back to memory
6. D-Cache fetches code to be edited
7. CPU writes code to D-Cache
8. D-Cache writes-back code

MANCHESTER 1824
The University of Manchester

Low-power architectures and call-site patching

- Fixed size instructions
  - Limit the range of direct branches/calls
    - +- 128MiB on AArch64
    - +- 1MiB on RISC-V
  - Require multiple instructions to perform long-range calls

AArch64

```
# zakkak @ odroid in ~ on git:master x [13:10:51]
$ java -XX:+PrintFlagsFinal 2>&1 | grep -i codecachesize
    uintx InitialCodeCacheSize                     = 2555904                                 {pd product}
    uintx ReservedCodeCacheSize                    = 134217728                               {pd product}
```

128MiB

x86-64

```
# zakkak @ badbee in ~ [13:13:48]
$ java -XX:+PrintFlagsFinal 2>&1 | grep -i codecachesize
    uintx InitialCodeCacheSize                     = 2555904                                 {pd product} {default}
    uintx ReservedCodeCacheSize                    = 251658240                               {pd product} {ergonomic}
```

240MiB

MANCHESTER 1824
The University of Manchester

Low-power architectures and call-site patching (cont.)

- Weak memory models and self-modifying-code (SMC) support
  - SW explicitly issues memory barriers
  - Code-stream handled separately from data-stream (need to sync them)
- Not all instructions are safe to patch
  - ARM (armv7 and armv8) and IBM (Power) limit the instructions that are safe to be patched while executing
    - Even if using atomic writes

MANCHESTER 1824
The University of Manchester

Patchable call-site implementations in AArch64

Direct Branching (short-range only)

```
B TARGET
```

Relative-Load Indirect Branching

```
CALLEE_1 : .quad 0 x0123456789ABCDEF
           ...
CALLEE_N : .quad 0 x01234ABCDEF56789
START :    ...
           LDR X16, CALLEE_1
           BLR X16
```

Absolute-Load Indirect Branching

```
MOVZ X16, #0xABCD ; Craft the address
MOVK X16, #0xEF89, lsl #16 ; holding
MOVK X16, #0x7654, lsl #32 ; the
MOVK X16, #0x0213, lsl #48 ; target
LDR  X16, [X16]
BLR  X16
```

Trampolines (OpenJDK approach)

```
L:       LDR X16, CALLEE
         BR  X16 ; Don 't link
CALLEE: .quad 0 x0123456789ABCDEF
START:   ...
         BL  SHORT_TARGET ; or L
```

MANCHESTER 1824
The University of Manchester

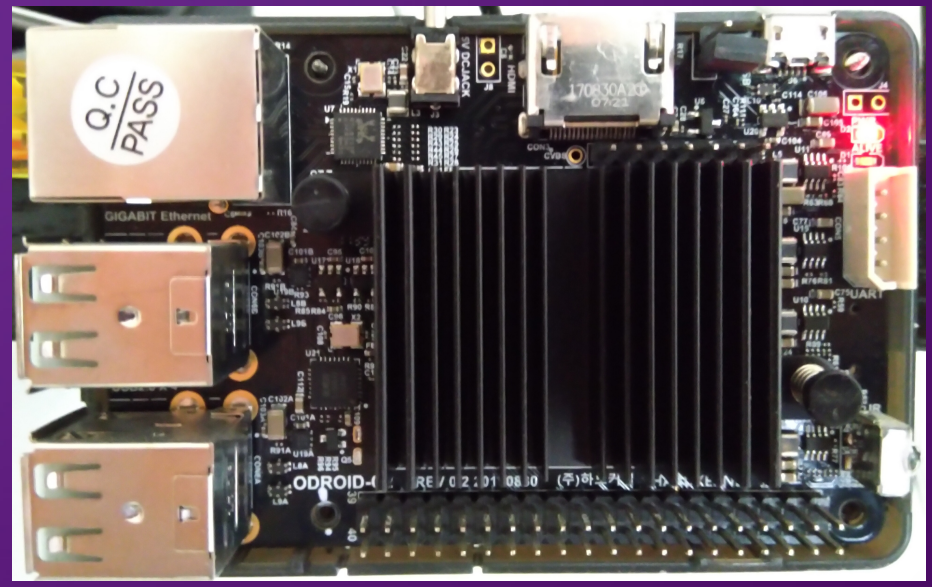
Comparison of call-site implementation approaches

	Direct	Absolute-load Indirect	Relative-load Indirect	Trampolines	Any | Safe-point
Characterisation	Code	Data	Hybrid	Hybrid	Any
Requires SMC support	Yes	No	No	Yes	Maybe
Size (in 32-bit instructions)	1	6	2 +2	1 +4 to 3 +2	Any (≥ 1)
Patching Complexity	Medium	Low	Medium	High	Any
Supported Call Range	Limited	Any	Any	Any	Any
Stop-the-world pause	No	No	No	No	Yes

MANCHESTER 1824
The University of Manchester

Evaluation Setup

- Odroid-C2
  - Quad-core Cortex-A53 @ 1.54GHz (pinned)
    - 8-stage pipelined processor with 2-way superscalar, in-order pipeline
  - 2 GB DDR3 RAM
  - Ubuntu 18.04.02 LTS
  - Kernel: Odroid 3.16..68-41
  - GCC 8.3.0
  - MaxineVM 2.8.0
  - OpenJDK 8 u212



MANCHESTER 1824
The University of Manchester

Microbenchmark

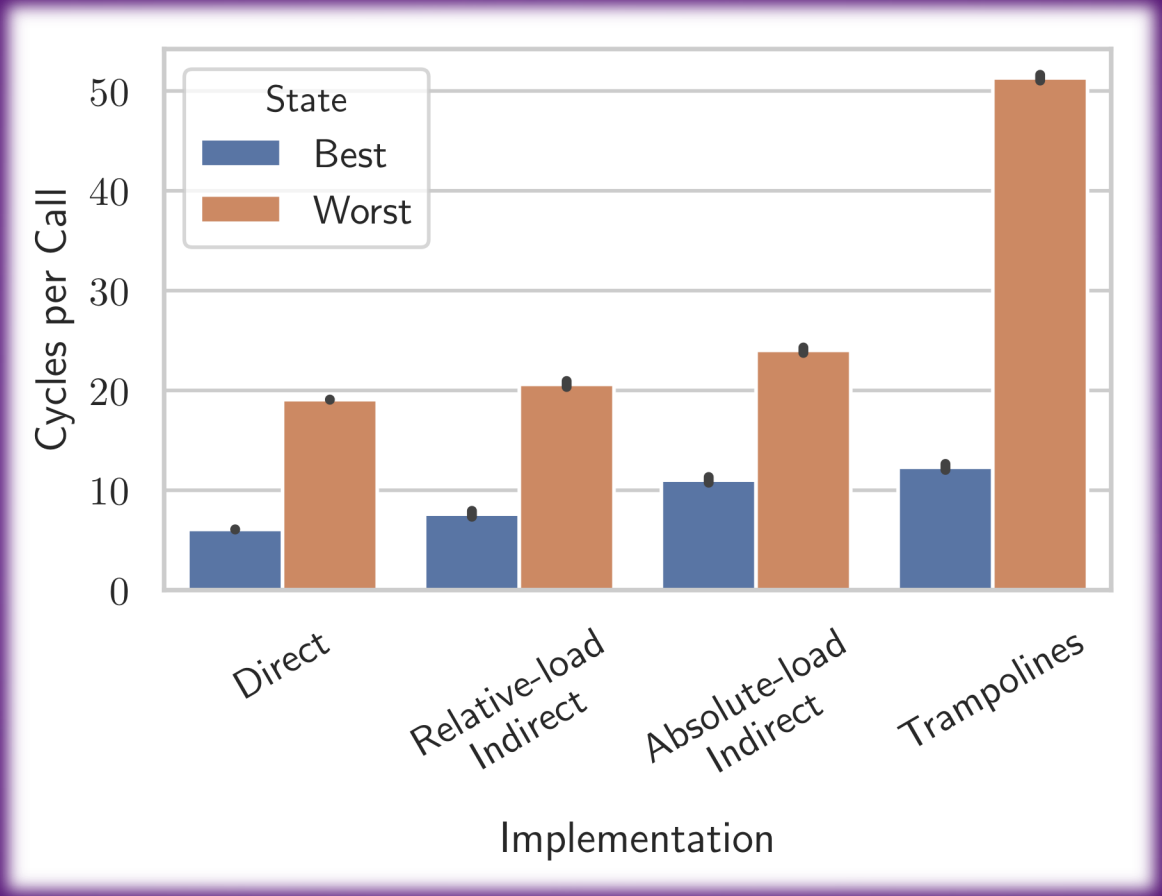
```
START:
        b #12
CALL_0: .quad PATCHER
        ldr x16, CALL_0
        blr x16
        b #12
CALL_1:.quad CALLEE_1
        ldr x16, CALL_1
        blr x16
        b #12
```

```
CALLEE_0:
        ret
CALLEE_1:
        ret
PATCHER:
        /* test to see if patching is enabled */
        cbnz w0, #8 /* jump over ret if non zero */
        ret
        sub w0, w0, #1
        adr x9, PATCHER
        add x10, x30, #4
        str x9, [x10]
        dc cvau, x10 /* write-back */
        dsb ish
```

- Generates inline call-sites
- Callers are ret-only methods
- To patch we call a patcher method instead of a ret-only
- Patcher always patches the next call-site (allows us to control number of patches
- Patcher performs the necessary barriers as it would in a real system

MANCHESTER 1824
The University of Manchester

Microbenchmark results



	Direct	Absolute-load Indirect	Relative-load Indirect	Trampolines
Cycles	45	12	24	56
L1I-refills on caller's site	1	0	0	0–2
L1D-refills on caller's site	0	1	1	0–1

MANCHESTER 1824
The University of Manchester

Dacapo and MaxineVM

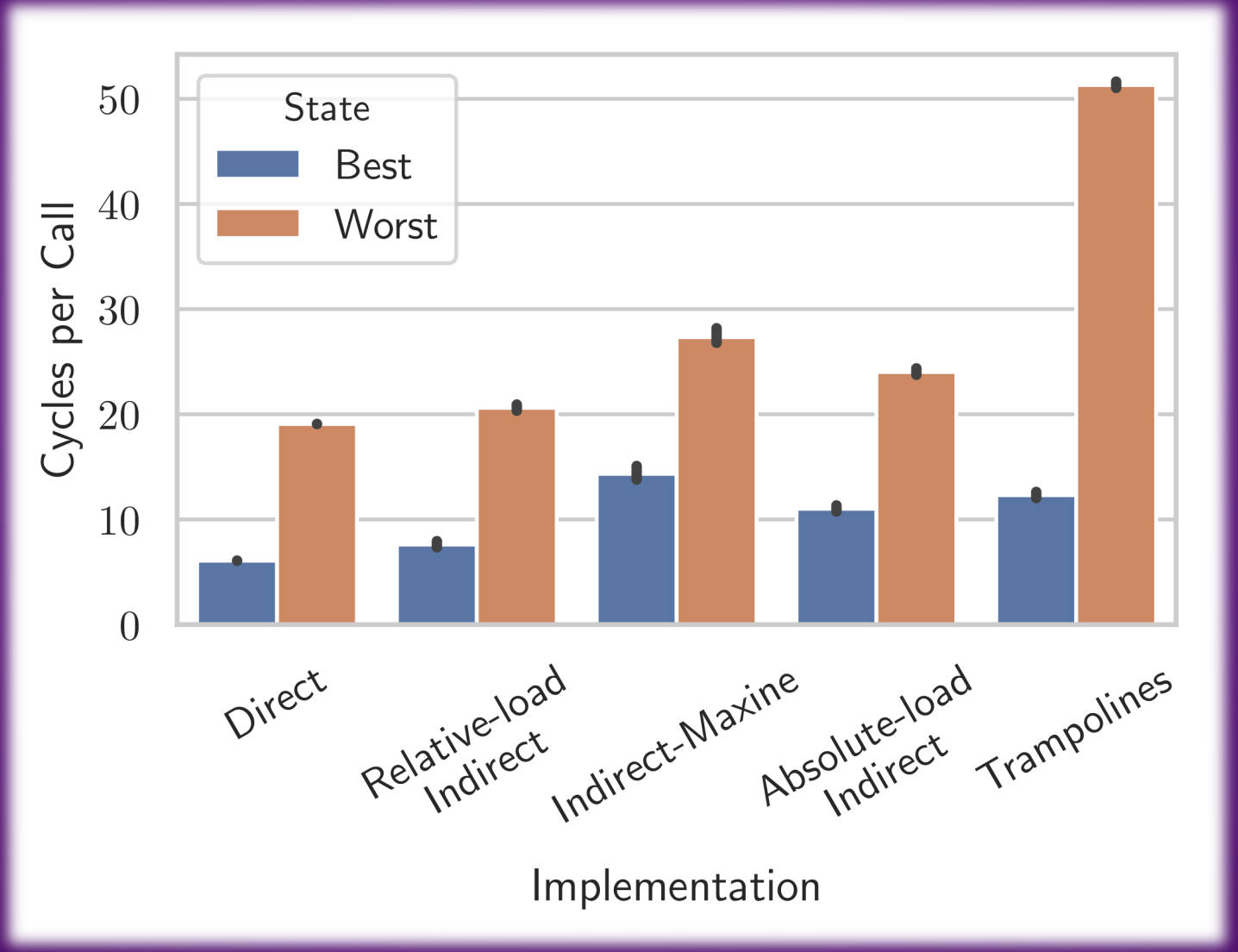
- We take the best two performing approaches (Direct and Relative-Load Indirect) and evaluate them with DaCapo using MaxineVM
- We had to tweak Relative-Load Indirect to make it work with MaxineVM
  - Due to its metacircular nature, MaxineVM can only operate with offsets (relative branches), since at boot image creation the absolute targets are not known yet

Indirect-Maxine

```
        ADR X17, CALL       ; Get address of BLR
        LDR X16, OFFSET     ; Load offset
        ADD X16, X16 , X17  ; Add them
        B #8                ; Jump over inline offset
OFFSET: .int CALL - CALLEE_1
CALL:   BLR X16
```

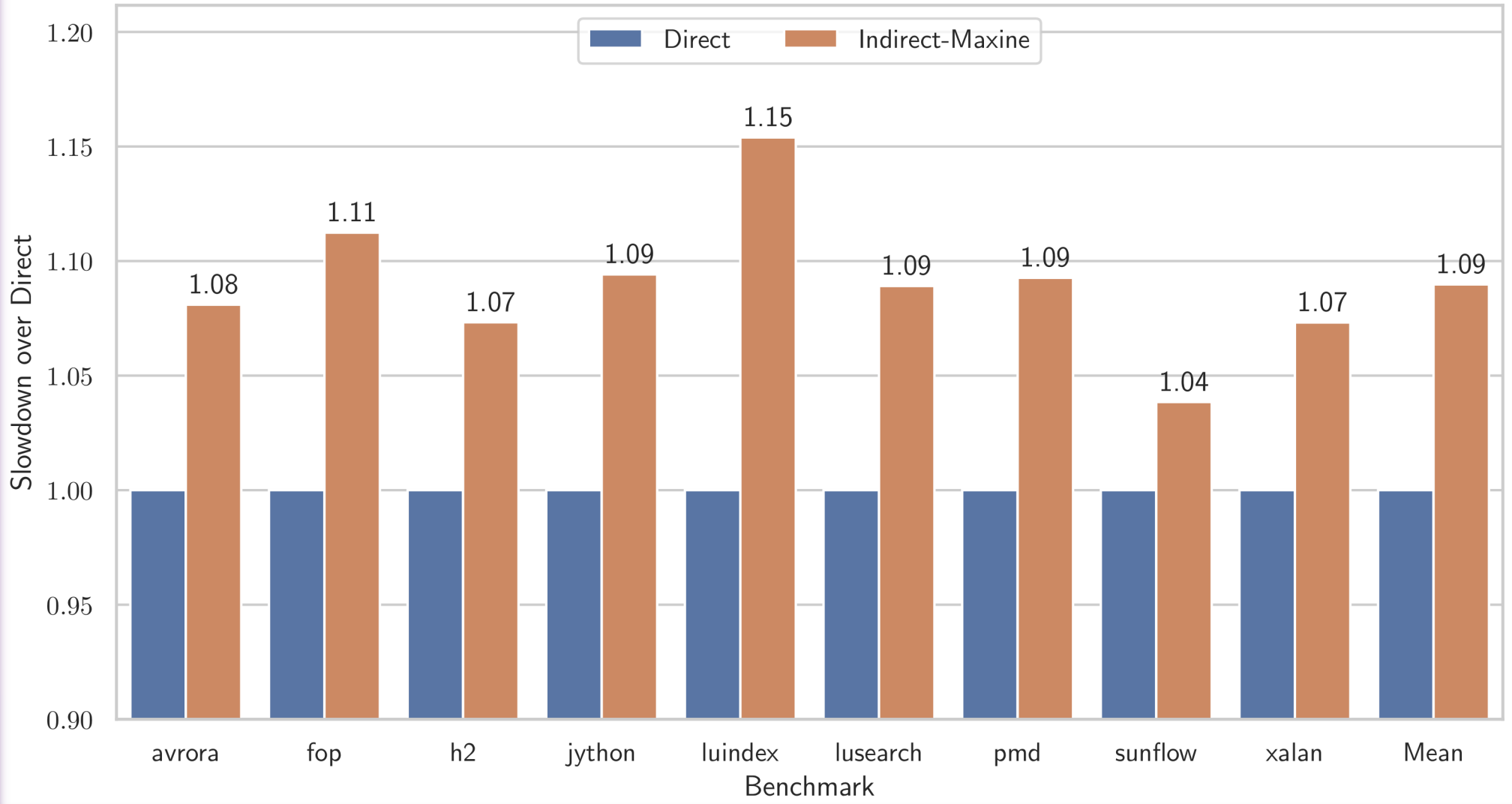
MANCHESTER 1824
The University of Manchester

Indirect-Maxine in Microbenchmark results



MANCHESTER 1824
The University of Manchester

DaCapo Results



MANCHESTER 1824
The University of Manchester

Conclusions

- OpenJDK's method seems the best for AArch64 since it penalizes only long-range branches and avoids explicit instruction cache invalidations on callers.
  - If you have a higher $\frac{\#long\text{-}range\ calls}{\#short\text{-}range\ calls}$ ratio then maybe Relative-Load is better
- The most promising approach **in theory** would be combining the following *gadgets*

Direct (short-range only)

```
B TARGET
```

Indirect (long-rang)

```
ADRP X16, CALLEE
ADD  X16, X16, :lo12:CALLEE
BLR  X16
```

  - On AArch64 this is not possible though since ADRP and ADD cannot be safely overwritten if they are being executed concurrently with the modifications.