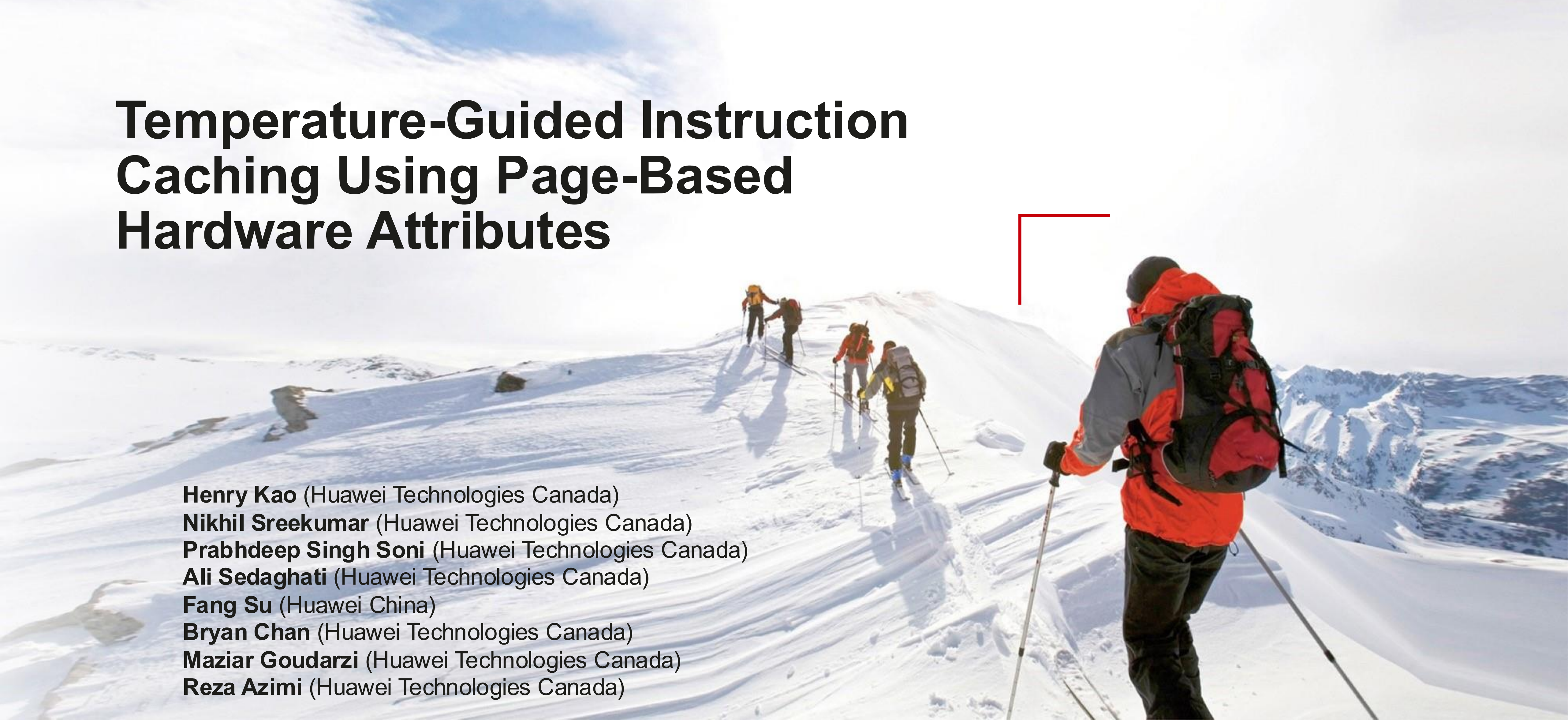
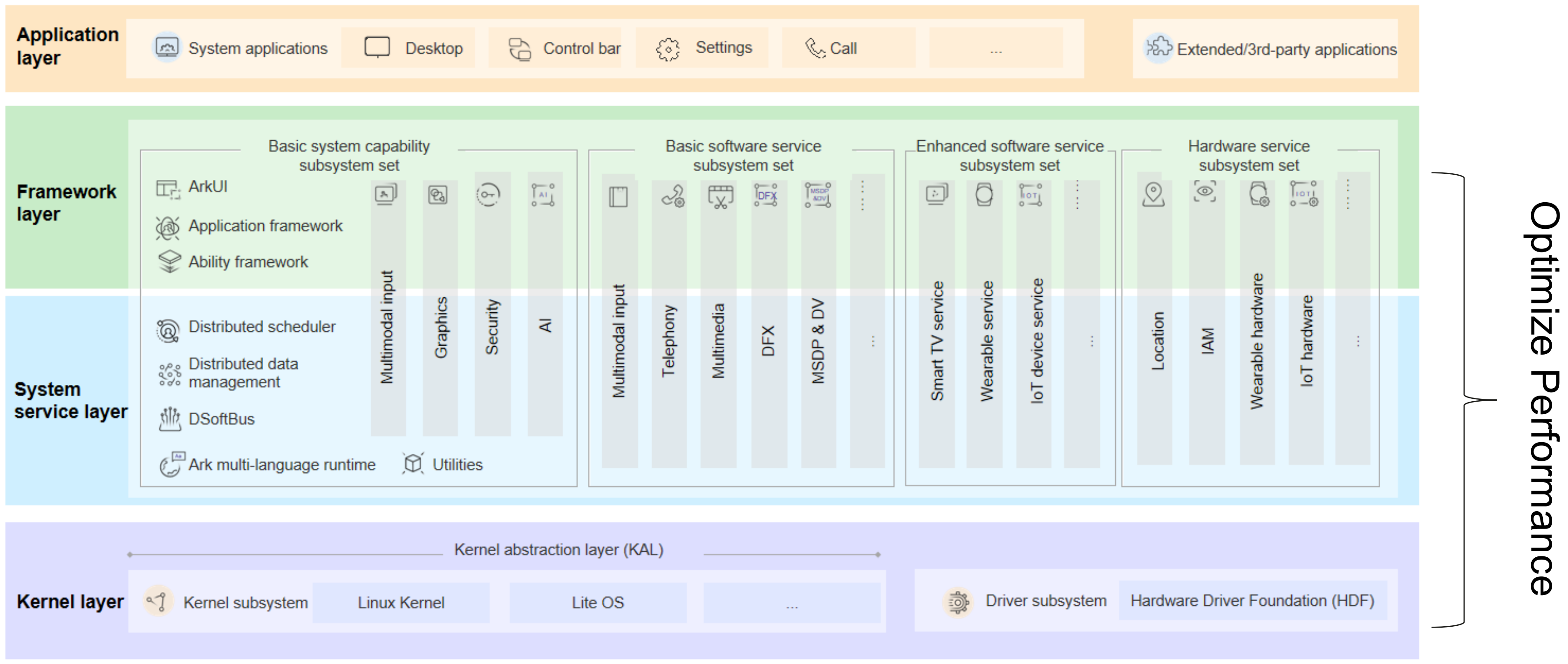
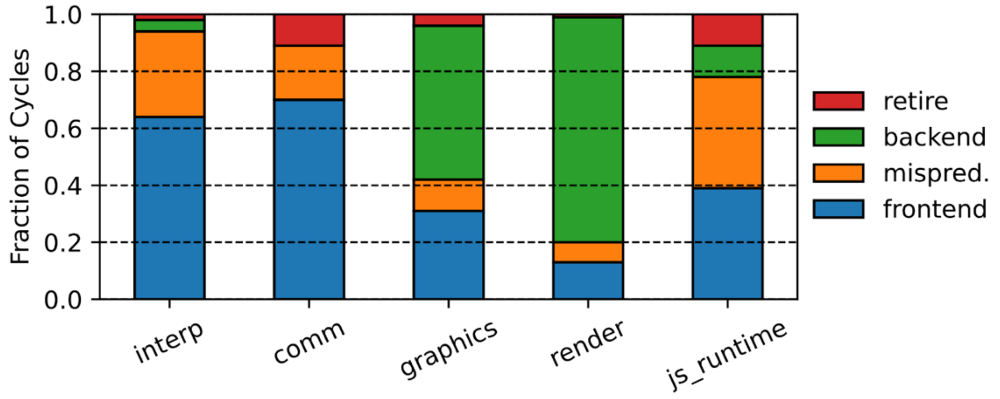


Temperature-Guided Instruction Caching Using Page-Based Hardware Attributes

Henry Kao (Huawei Technologies Canada)
Nikhil Sreekumar (Huawei Technologies Canada)
Prabhdeep Singh Soni (Huawei Technologies Canada)
Ali Sedaghati (Huawei Technologies Canada)
Fang Su (Huawei China)
Bryan Chan (Huawei Technologies Canada)
Maziar Goudarzi (Huawei Technologies Canada)
Reza Azimi (Huawei Technologies Canada)

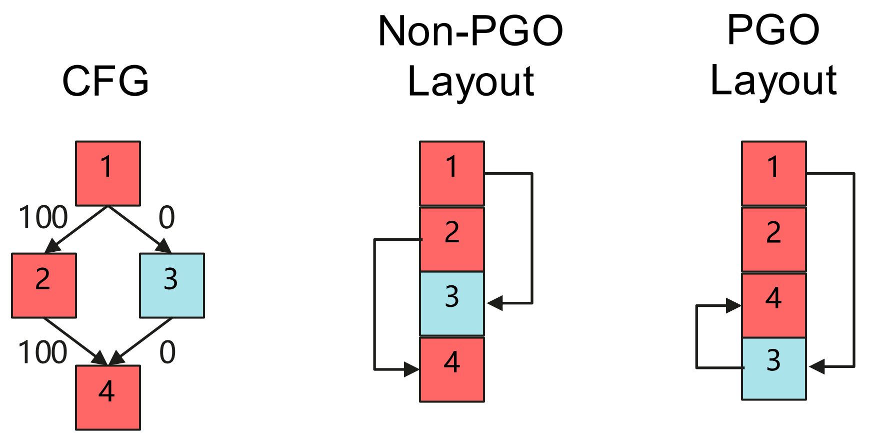
Optimizing Mobile System Libraries



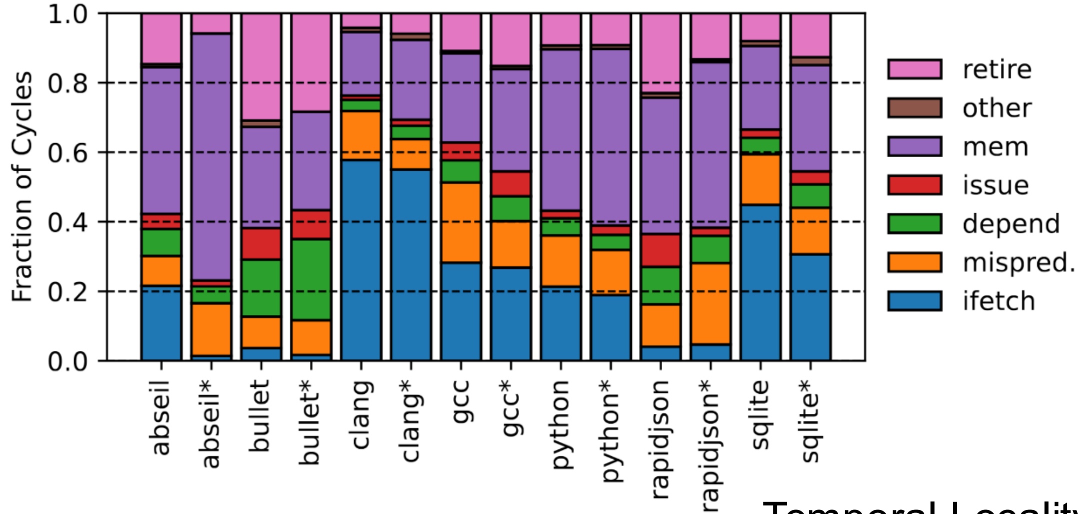


- System performance impacts everything running on the mobile device, not just a single application/workload
- Design compiler optimizations or software-hardware co-design techniques to optimize performance
- Many system libraries show front-end boundedness
- Compiler PGO with code-layout optimizations already show considerable performance improvements

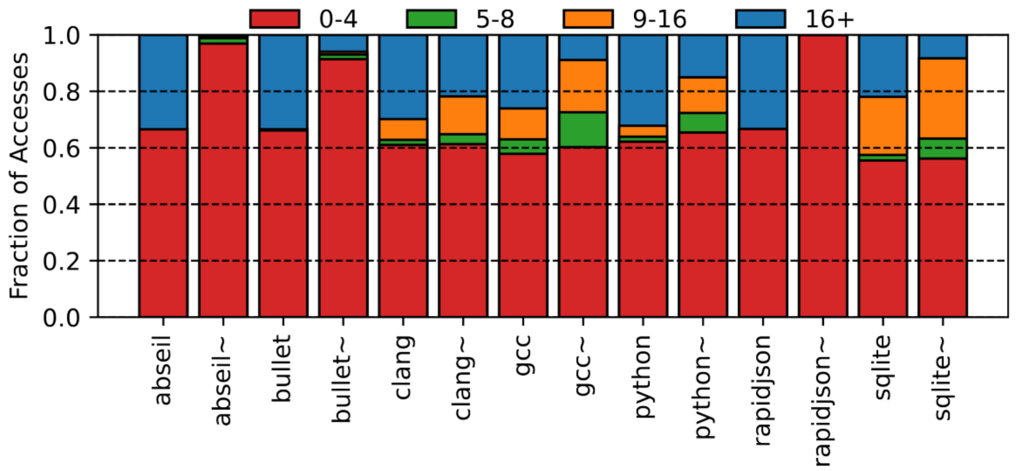
Reasons for Frontend Bottlenecks



Spatial Locality ✓



Temporal Locality ✗



- Key Insight
  > PGO improves spatial locality
  > PGO cannot improve temporal locality
  > Cache replacement tries to predict temporal locality, can we influence that?
- Idea
  > Pass temperature information to the caches

HUAWEI

Challenges as a Compiler Developer

- Develop compiler optimizations to improve performance
- Great, because no hardware changes are needed, performance from pure code optimizations
  - > Function Inlining
  - > Function Placement
  - > Block Placement

Trade-offs to consider:

- Code Size
- Compile Time

```c
#include <stdio.h>
#include <stdlib.h>

extern void dummy(int x);

int compute(int x) {
  int temp1 = x * x;
  int temp2 = temp1 + x;
  int temp3 = temp1 / temp2 - 25;
  int temp4 = temp3 ^ 0xffff & 0xadbd;
  int temp5 = (temp4 * x) + x;
  return temp5;
}

int main(int argc, char *argv[]) {
  int num1 = atoi(argv[0]);
  int num2 = atoi(argv[1]);
  int result1 = compute(num1);
  int result2 = compute(num2);
  dummy(result1);
  dummy(result2);
  return 0;
}
```

Not Inlined

```
compute:
        mul     w9, w0, w0
        mov     w8, #44477
        add     w10, w9, w0
        sdiv    w9, w9, w10
        sub     w9, w9, #25
        eor     w8, w9, w8
        madd    w0, w0, w8, w0
        ret

main:
        stp     x29, x30, [sp, #-32]!
        stp     x20, x19, [sp, #16]
        mov     x29, sp
        ldr     x0, [x1]
        mov     x19, x1
        mov     x1, xzr
        mov     w2, #10
        bl      strtol
        ldr     x8, [x19, #8]
        mov     x19, x0
        mov     x1, xzr
        mov     w2, #10
        mov     x0, x8
        bl      strtol
        mov     x20, x0
        mov     w0, w19
        bl      compute
        mov     w19, w0
        mov     w0, w20
        bl      compute
        mov     w20, w0
        mov     w0, w19
        bl      dummy
        mov     w0, w20
        bl      dummy
        mov     w0, wzr
        ldp     x20, x19, [sp, #16]
        ldp     x29, x30, [sp], #32
        ret
```

Inlined

```
compute:
        mul     w9, w0, w0
        mov     w8, #44477
        add     w10, w9, w0
        sdiv    w9, w9, w10
        sub     w9, w9, #25
        eor     w8, w9, w8
        madd    w0, w0, w8, w0
        ret

main:
        stp     x29, x30, [sp, #-32]!
        str     x19, [sp, #16]
        mov     x29, sp
        ldr     x0, [x1]
        mov     x19, x1
        mov     x1, xzr
        mov     w2, #10
        bl      strtol
        ldr     x8, [x19, #8]
        mov     x19, x0
        mov     x1, xzr
        mov     w2, #10
        mov     x0, x8
        bl      strtol
        mul     w8, w19, w19
        add     w9, w8, w19
        sdiv    w8, w8, w9
        mul     w9, w0, w0
        add     w10, w9, w0
        sdiv    w9, w9, w10
        sub     w8, w8, #25
        mov     w10, #44477
        eor     w8, w8, w10
        madd    w8, w19, w8, w19
        sub     w9, w9, #25
        eor     w9, w9, w10
        madd    w19, w0, w9, w0
        mov     w0, w8
        bl      dummy
        mov     w0, w19
        bl      dummy
        mov     w0, wzr
        ldr     x19, [sp, #16]
        ldp     x29, x30, [sp], #32
        ret
```

HUAWEI

Challenges Designing Co-Designed Solutions

- Software/Compiler offline analysis can provide a lot of performance optimization hints to the hardware
  - > Software prefetching
  - > Execution frequency of instructions/data
  - > Branch taken/not-taken information (ARM BRBE)

- How to pass information from offline analysis to hardware?

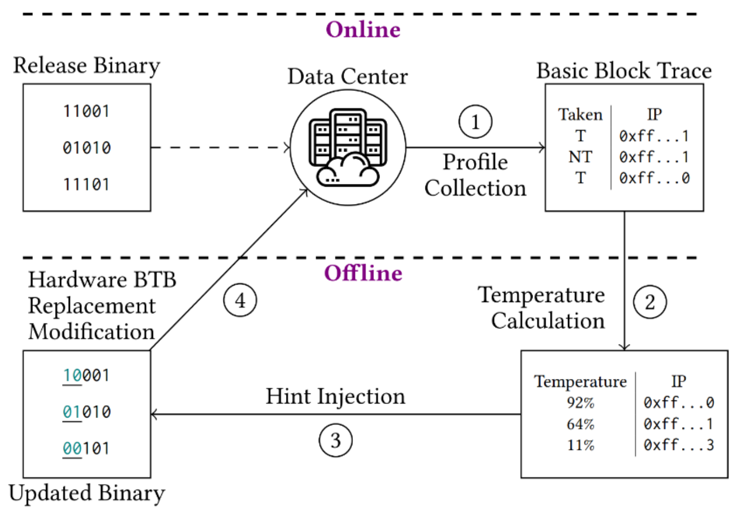


Figure 10: High level design of *Thermometer*

Song et al., "Thermometer: Profile-Guided BTB Replacement for Data Center Applications", ISCA'22

New Instructions?

- > ISA license (e.g., ARM) restrictions
- > Dynamic instruction overhead
- > New HW to process instructions

Metadata?

- > Code size increase
- > Dynamic code footprint increase

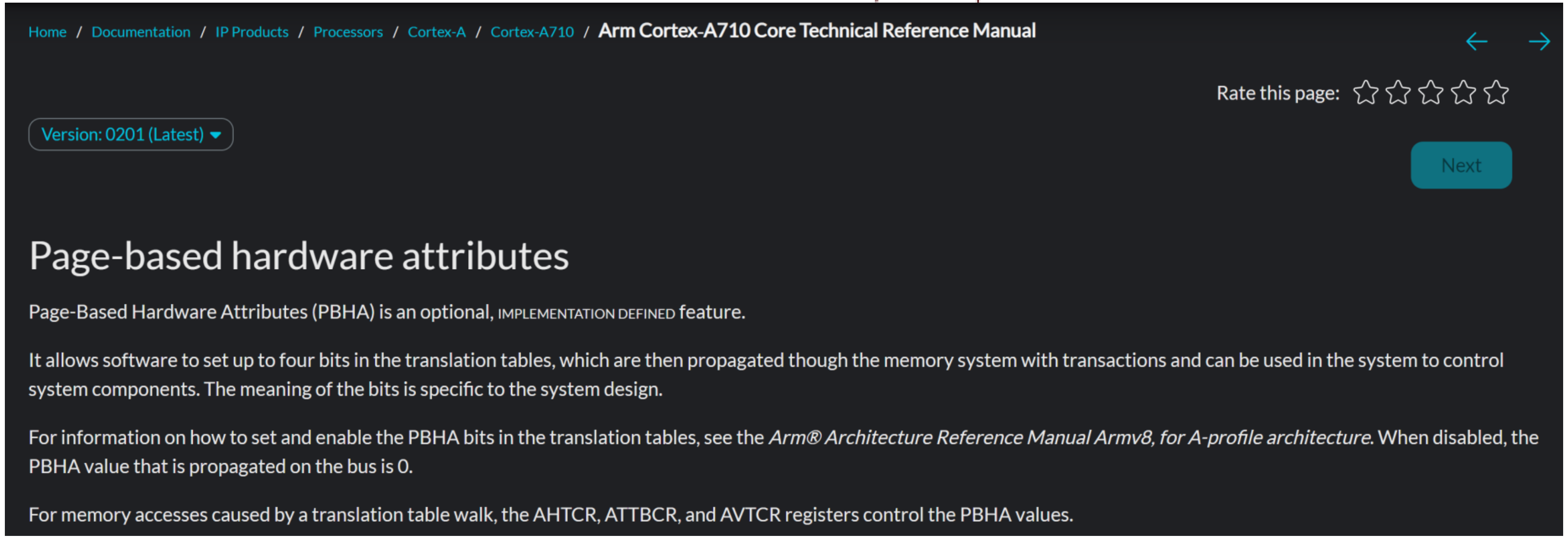
Summary of Challenges

Software

- Keep code size increase minimal
- Keep compile time increase minimal

Hardware

- No changes to ISA
- Power and area cost minimal (especially on mobile devices)



Home / Documentation / IP Products / Processors / Cortex-A / Cortex-A710 / Arm Cortex-A710 Core Technical Reference Manual
Rate this page:
Version: 0201 (Latest)
Next

Page-based hardware attributes

Page-Based Hardware Attributes (PBHA) is an optional, IMPLEMENTATION DEFINED feature.

It allows software to set up to four bits in the translation tables, which are then propagated though the memory system with transactions and can be used in the system to control system components. The meaning of the bits is specific to the system design.

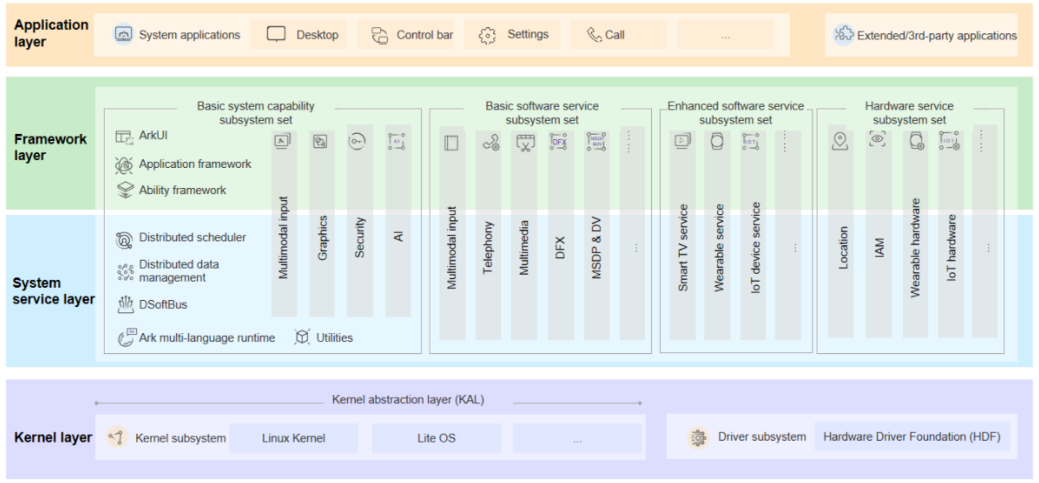
For information on how to set and enable the PBHA bits in the translation tables, see the *Arm® Architecture Reference Manual Armv8, for A-profile architecture*. When disabled, the PBHA value that is propagated on the bus is 0.

For memory accesses caused by a translation table walk, the AHTCR, ATTBCR, and AVTCR registers control the PBHA values.

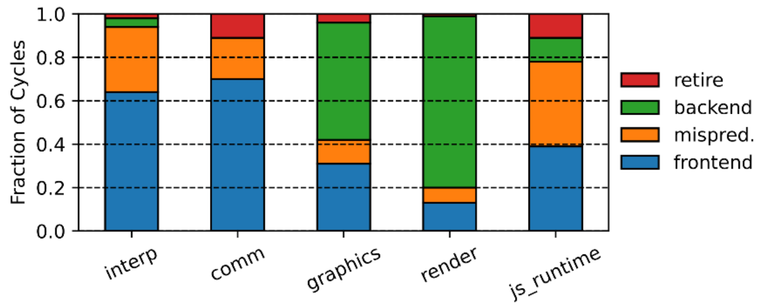
HUAWEI

Mise En Place

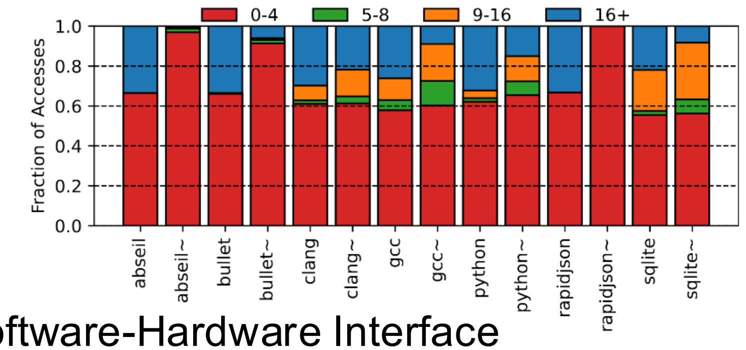
Mobile Libraries



Front-end Bottlenecks

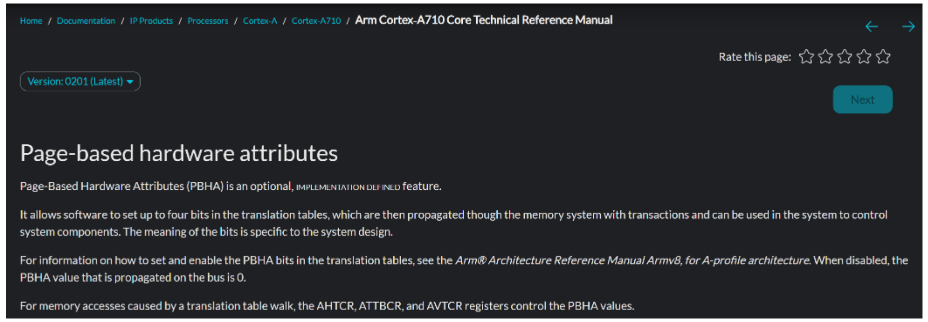


Temporal Locality of Hot Code is an Issue



Compiler (BiSheng/LLVM) PGO

Convenient Software-Hardware Interface



Home / Documentation / IP Products / Processors / Cortex-A / Cortex-A710 / Arm Cortex-A710 Core Technical Reference Manual

Rate this page: ☆☆☆☆☆

Version: 0201 (Latest)

Next

Page-based hardware attributes

Page-Based Hardware Attributes (PBHA) is an optional, IMPLEMENTATION DEFINED feature.

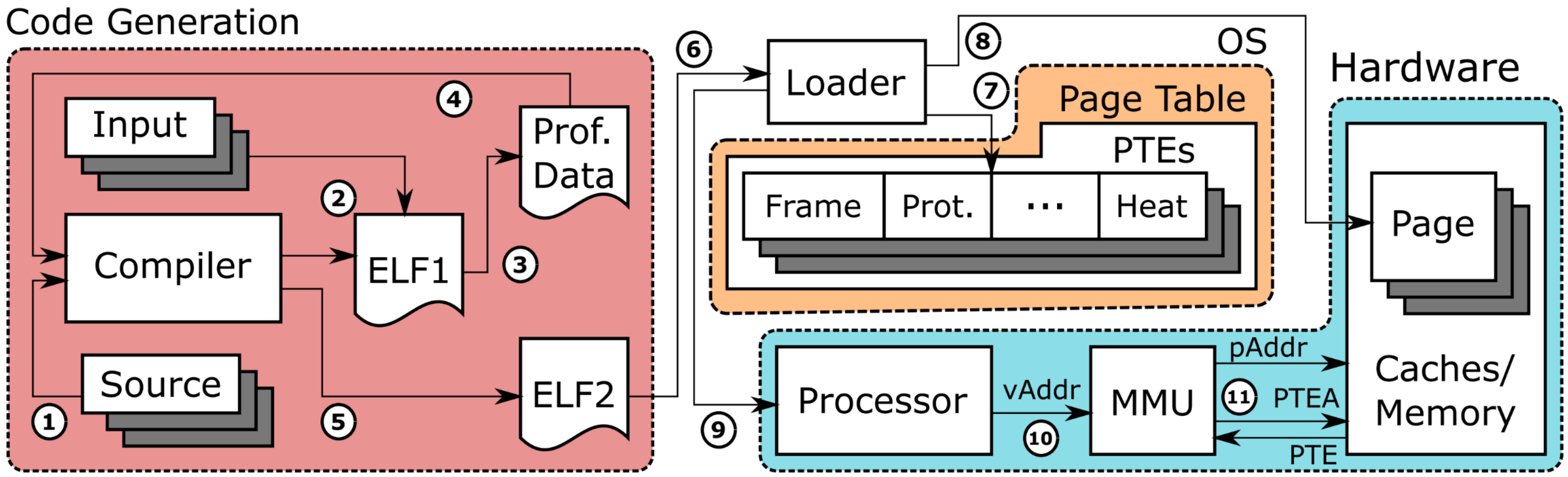
It allows software to set up to four bits in the translation tables, which are then propagated though the memory system with transactions and can be used in the system to control system components. The meaning of the bits is specific to the system design.

For information on how to set and enable the PBHA bits in the translation tables, see the *Arm® Architecture Reference Manual Armv8, for A-profile architecture*. When disabled, the PBHA value that is propagated on the bus is 0.

For memory accesses caused by a translation table walk, the AHTCR, ATTBCR, and AVTCR registers control the PBHA values.

HUAWEI

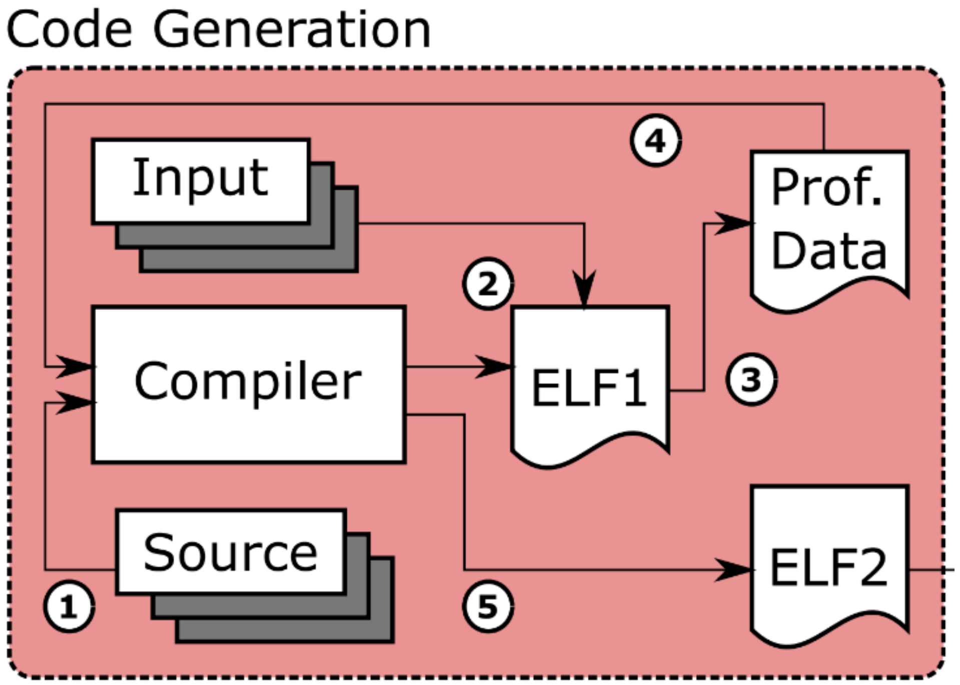
Temperature Guided Instruction Cache Replacement

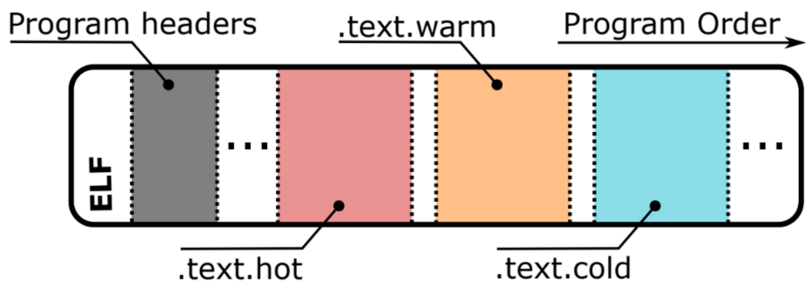


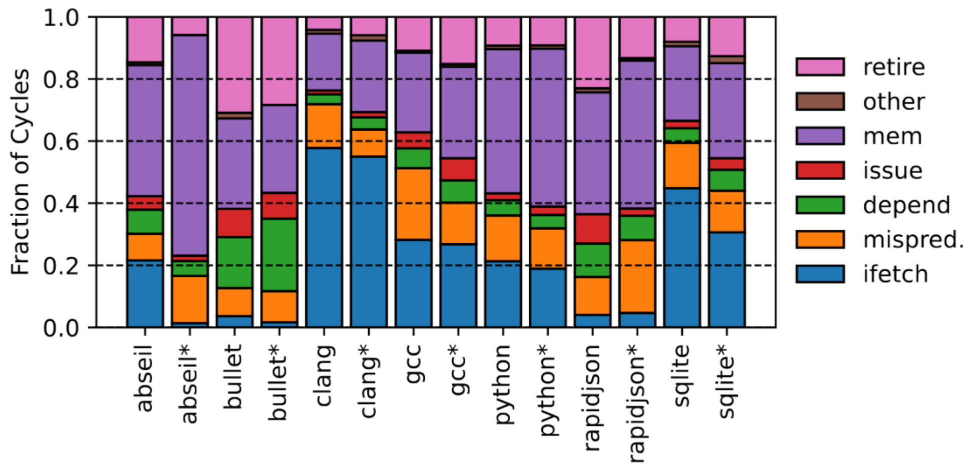
Code Generation and Optimization

```
> readelf –S <binary>

Section Headers:
 [Nr] Name              Type             Address           Offset
      Size              EntSize          Flags  Link  Info  Align
 [ 0]                   NULL             0000000000000000  00000000
      0000000000000000  0000000000000000           0     0     0
...
 [13] .text             PROGBITS         0000000000038ae0  00037ae0
      0000000000030bc9  0000000000000000  AX       0     0     16
 [14] .init             PROGBITS         00000000000696ac  000686ac
      000000000000001b  0000000000000000  AX       0     0     4
 [15] .fini             PROGBITS         00000000000696c8  000686c8
      000000000000000d  0000000000000000  AX       0     0     4
 [16] .text.unlikely    PROGBITS         00000000000696e0  000686e0
      00000000000624d6  0000000000000000  AX       0     0     16
 [17] .text.hot         PROGBITS         00000000000cbbc0  000cabc0
      0000000000088f59  0000000000000000  AX       0     0     16
 [18] .plt              PROGBITS         0000000000154b20  00153b20
      00000000000003b0  0000000000000000  AX       0     0     16
...
```



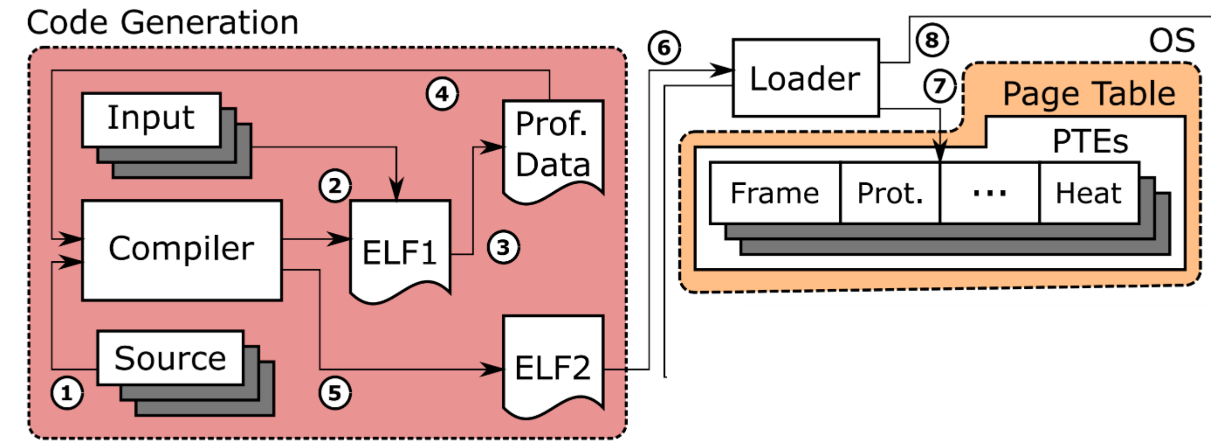


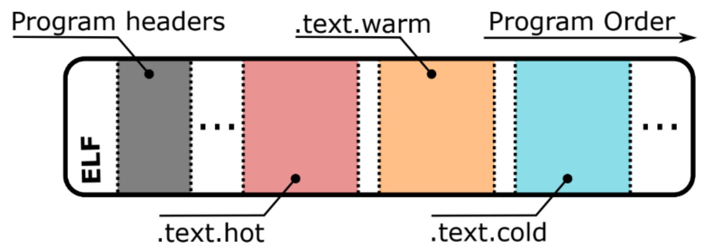


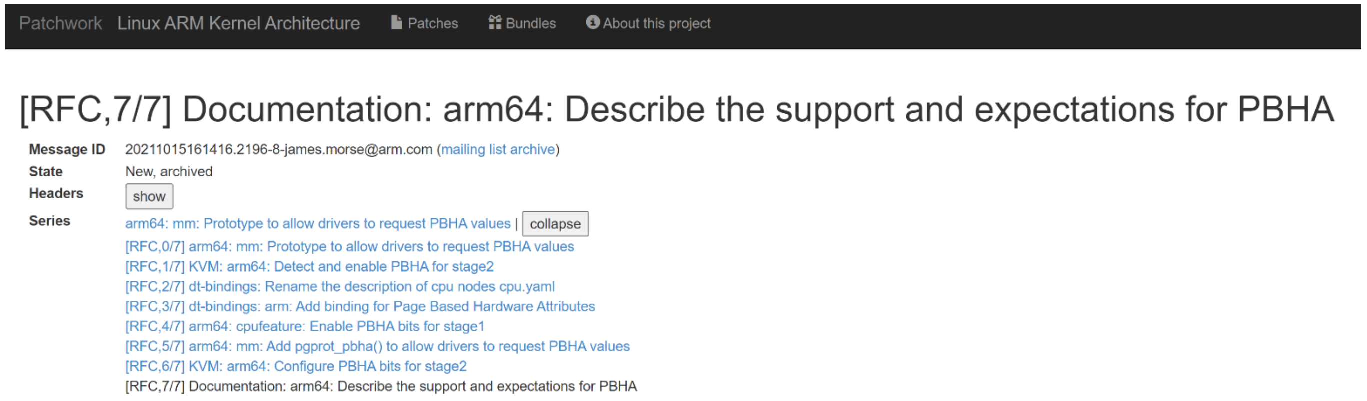
Operating System

- Loader is executed prior to running the application
- Loader reads information in program headers (incl. temperature information)
- Loader creates the Page Tables and corresponding PTEs
- Populate PBHA bits with temperature information

- PBHA support ongoing in Linux Kernel





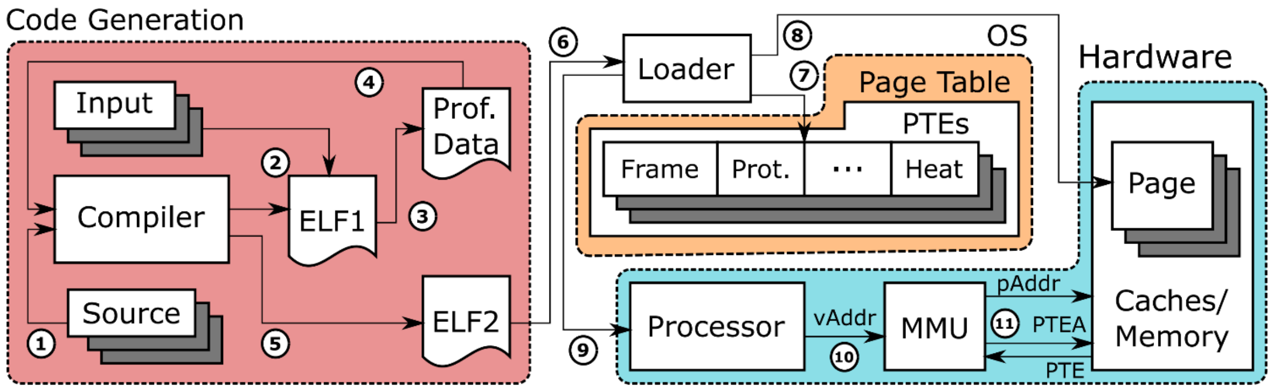


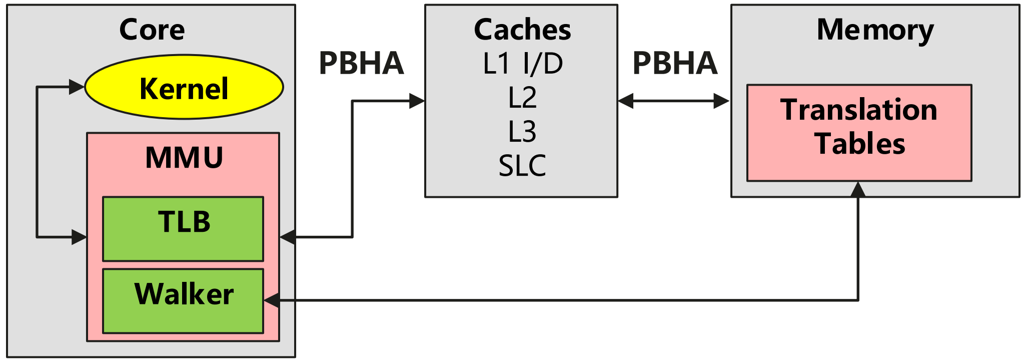
[RFC,7/7] Documentation: arm64: Describe the support and expectations for PBHA

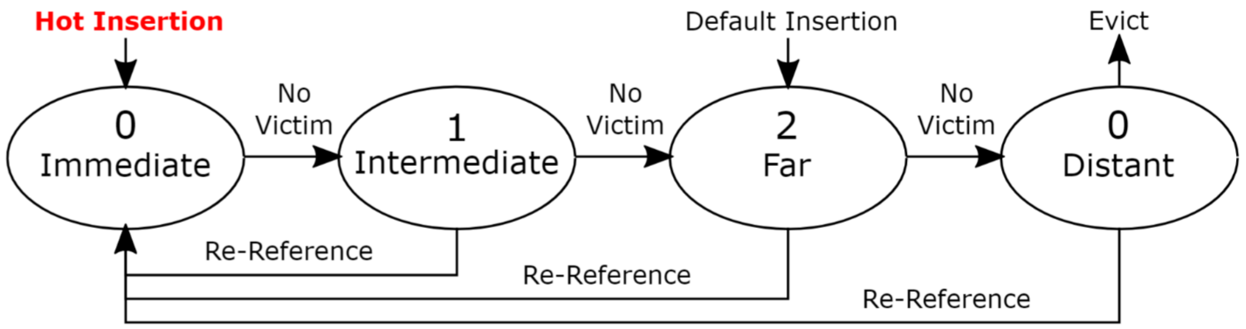
Message ID	20211015161416.2196-8-james.morse@arm.com (mailing list archive)
State	New, archived
Headers	show
Series	arm64: mm: Prototype to allow drivers to request PBHA values | collapse [RFC,0/7] arm64: mm: Prototype to allow drivers to request PBHA values [RFC,1/7] KVM: arm64: Detect and enable PBHA for stage2 [RFC,2/7] dt-bindings: Rename the description of cpu nodes cpu.yaml [RFC,3/7] dt-bindings: arm: Add binding for Page Based Hardware Attributes [RFC,4/7] arm64: cpufeature: Enable PBHA bits for stage1 [RFC,5/7] arm64: mm: Add pgprot_pbha() to allow drivers to request PBHA values [RFC,6/7] KVM: arm64: Configure PBHA bits for stage2 [RFC,7/7] Documentation: arm64: Describe the support and expectations for PBHA

Cache Replacement

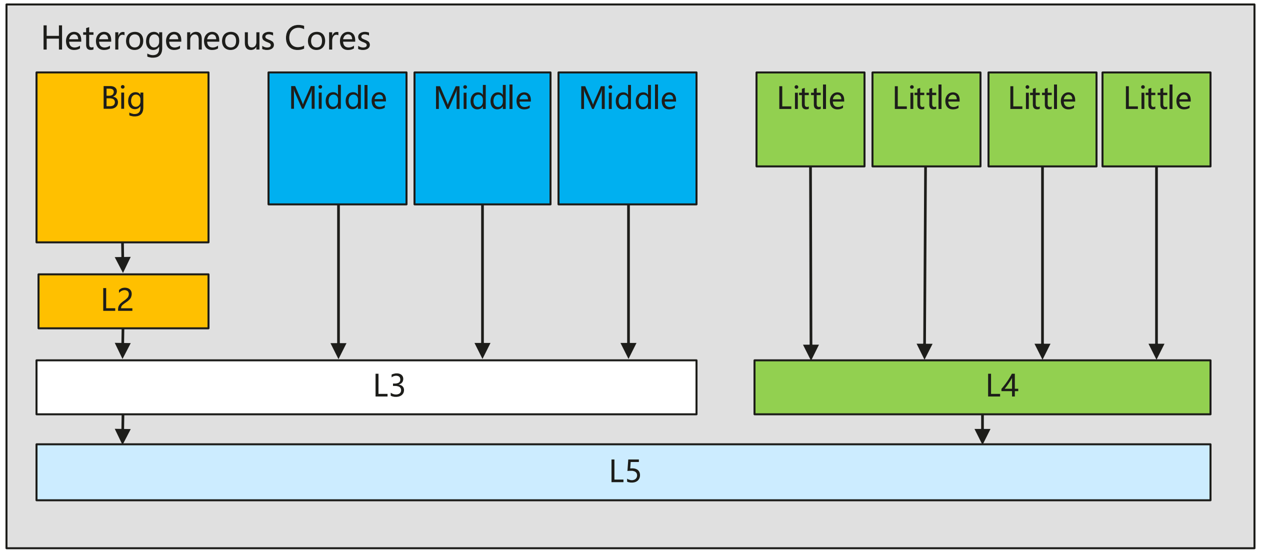
- Pass temperature information from PTE with memory request to caches
- Cache replacement reads temperature information to improve temporal locality of **HOT** code
- Use RRIP replacement policy
- Applied to L2 cache







Evaluation Methodology



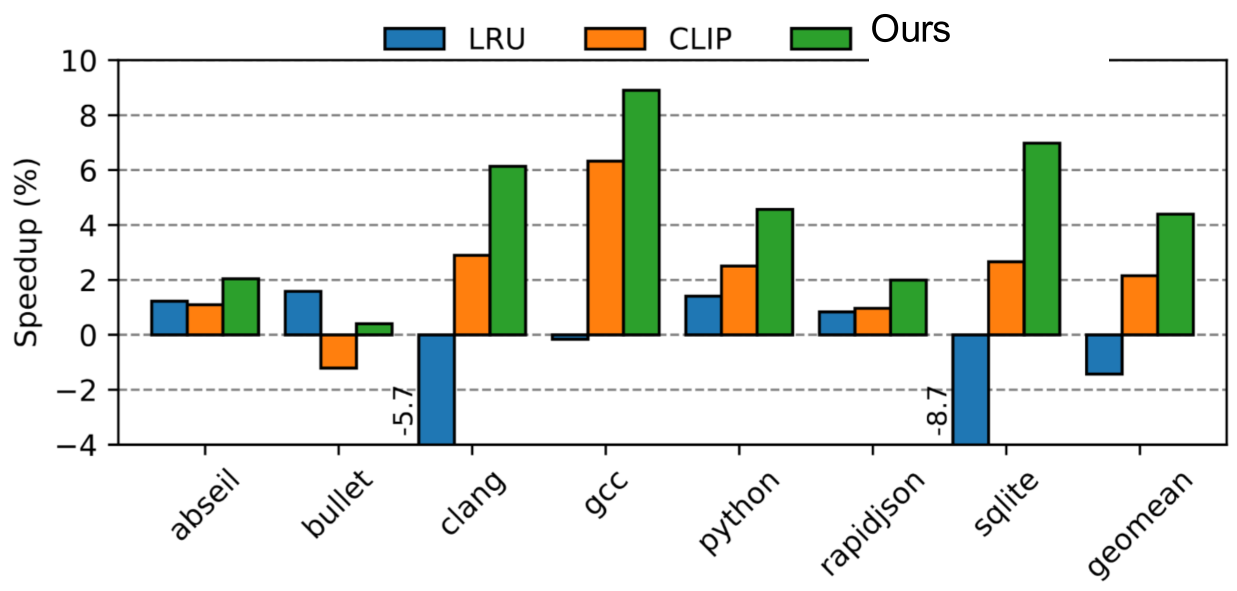
Component	Configuration
Core	6-wide dispatch, 128 entry ROB, 2GHz, pseudo-FDIP prefetching
Branch	1K-entry BTB, 512-entry indirect-BTB, 256-entry loop predictor, 1K-entry global predictor
L1-D	64KB, 4-way, LRU replacement, 1/3 (tag/data)-cycle latency, stride prefetcher
L1-I	64KB, 4-way, LRU replacement, 1/3 (tag/data)-cycle latency, stride prefetcher
Unified Shared L2	512kB (128kB per core), 8-way, TRRIP replacement, 8/12 (tag/data)-cycle latency, inclusive, stride prefetcher
Unified Shared SLC	1MB, 16-way, LRU replacement, 10/30 (tag/data)-cycle latency, exclusive
DRAM	8 chips/DIMM, 4 DIMMs, 7.6 GB/s controller bandwidth, 400-cycle latency

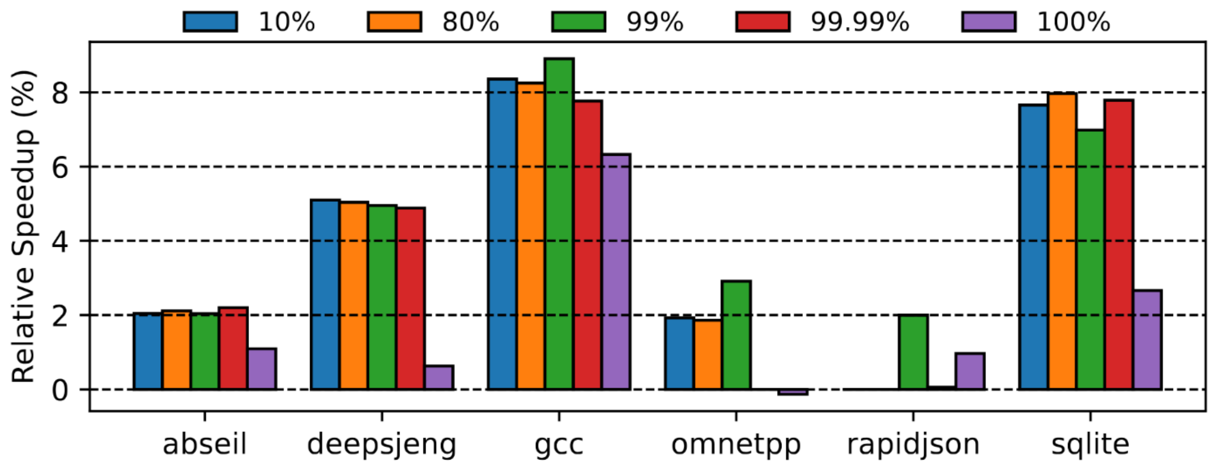
Benchmark	Training	Evaluation	FF
abseil	all[a]	absl_btree_test	1E9
bullet	train[b]	eval[b]	1E9
clamscan	train[b]	eval[b]	1E7
clang	ninja clang-check-c	gcc's ref	1E8
deepsjeng	train[b]	ref[b]	4E9
gcc	train[b]	ref[b]	1E8
omnetpp	train[b]	ref[b]	4E8
python	train[b]	test_statistics	1E8
rapidjson	unittest + perftest	perftest	
sqlite	−shrink-memory − reprepare −heap 10000000 64 −size 50	−shrink-memory − reprepare −heap 10000000 64 −size 5	1E8

[a] All tests in test suite, [b] Use default profiling and evaluation input sets with benchmark

Evaluation

- Normalized to RRIP
- RRIP on average performs better than LRU
- Compare with CLIP (Code Line Preservation HPCA'15)
  - > Prioritize all instruction cache lines
- TRRIP performs better than CLIP due to being more selective about which instruction lines to keep in the cache
- Instruction "hotness" is tunable in the compiler, can select best performing value





Conclusion

- Mobile system libraries show frontend bottlenecks
- PGO shows considerable speedup, but frontend bottleneck still present
  - Improvement comes from improving instruction spatial locality
  - More can be done with temporal locality, since HOT code shows high reuse distances
- Propose a lightweight technique that reuses many existing and established mechanisms
- Pass code temperature information from compiler PGO to hardware cache using ARM’s PBHA
- Improve temporal locality by keeping HOT code in the cache for longer

- Full paper currently in review…

Thank you.

Bring digital to every person, home and organization for a fully connected, intelligent world.

HUAWEI