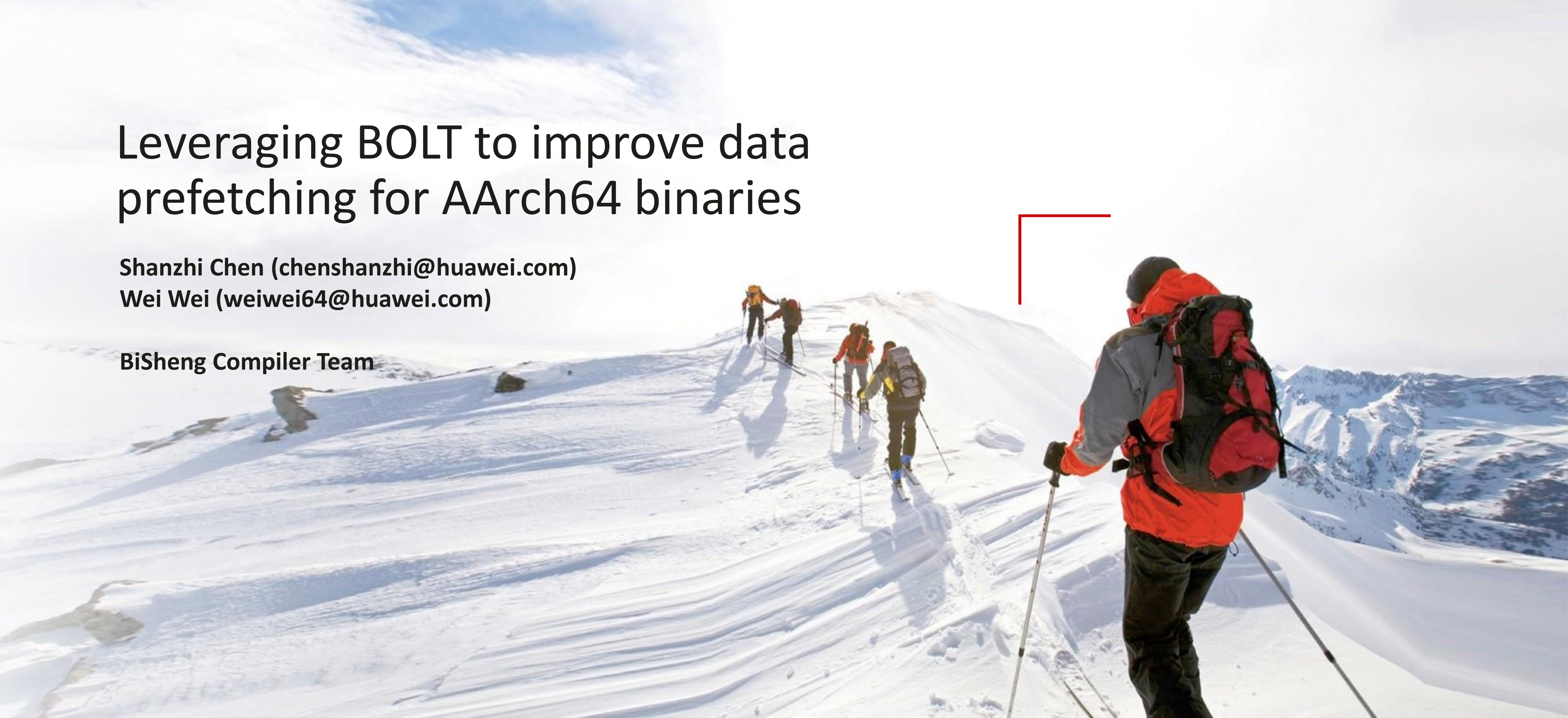
Leveraging BOLT to improve data prefetching for AArch64 binaries

Shanzhi Chen (chenshanzhi@huawei.com)
Wei Wei (weiwei64@huawei.com)

BiSheng Compiler Team



HUAWEI

A Quick Glance at BOLT

- **B**inary **O**ptimization and **L**ayout **T**ool

- A post-Link optimizer
  > Rewrite binary directly
  > Rely on relocations saved by linker (-Wl,--emit-relocs)

- A profile-guided optimizer
  > Instrumentation profile
  > Sample-based profile via LBR/BRBE, SPE, or PMU

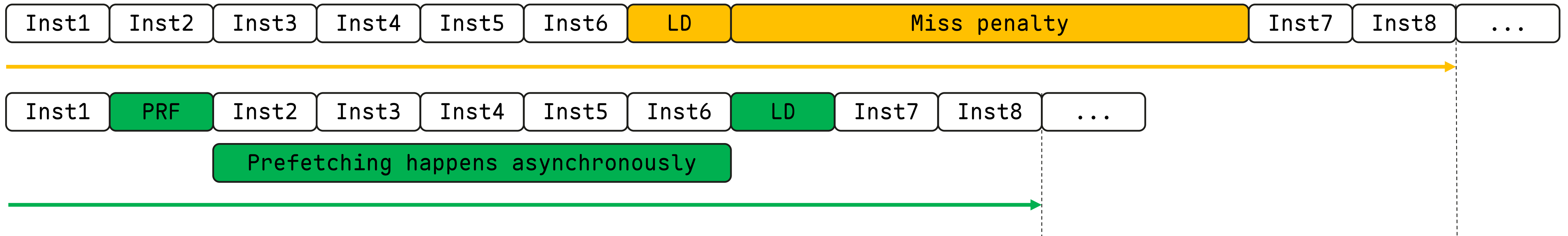
- Many optimizations on **code layout**
  > --reorder-blocks, --reorder-functions, --reorder-data
  > --split-functions, --split-all-cold, --split-eh
  > --icp, --icf, --plt, --jump-tables

- Passes listed in the original BOLT paper:

Pass Name	Description
1. strip-rep-ret	Strip `repz` from `repz retq` instructions used for legacy AMD processors
2. icf	Identical code folding
3. icp	Indirect call promotion
4. peepholes	Simple peephole optimizations
5. inline-small	Inline small functions
6. simplify-ro-loads	Fetch constant data in .rodata whose address is known statically and mutate a load into a mov
7. icf	Identical code folding (second run)
8. plt	Remove indirection from PLT calls
9. reorder-bbs	Reorder basic blocks and split hot/cold blocks into separate sections (layout optimization)
10. peepholes	Simple peephole optimizations (second run)
11. uce	Eliminate unreachable basic blocks
12. fixup-branches	Fix basic block terminator instructions to match the CFG and the current layout (redone by reorder-bbs)
13. reorder-functions	Apply HFSort [25] to reorder functions (layout optimization)
14. sctc	Simplify conditional tail calls
15. frame-opts	Removes unnecessary caller-saved register spilling
16. shrink-wrapping	Moves callee-saved register spills closer to where they are needed, if profiling data shows it is better to do so

A Quick Glance at Prefetching

- Purpose of Prefetching: **Hide the latency of a memory access**



- Hardware Prefetching
  > Hardware prefetcher implementations are often **proprietary and undisclosed**.

  > Analyze **memory access patterns** and prefetch at runtime.
- Software Prefetching
  > Processor instructions: **PRFM, PRFUM, RPRFM**

  > Compiler built-in functions: **__builtin_prefetch(const void *addr, int rw=0, int locality=3);**

  > Compiler optimizations: **LoopDataPrefetchPass**
- Instruction Prefetching / Data Prefetching
- 3 Main Metrics: Coverage / Accuracy / Timeliness

HUAWEI

Background

- Why BOLT?
  > The source code may not be available for performance analysis.
  > The compiler toolchain may not be changed for performance tuning.
  > BOLT could be used to rewrite a binary.
  > Adding a binary rewriting pass in BOLT is relatively straightforward.

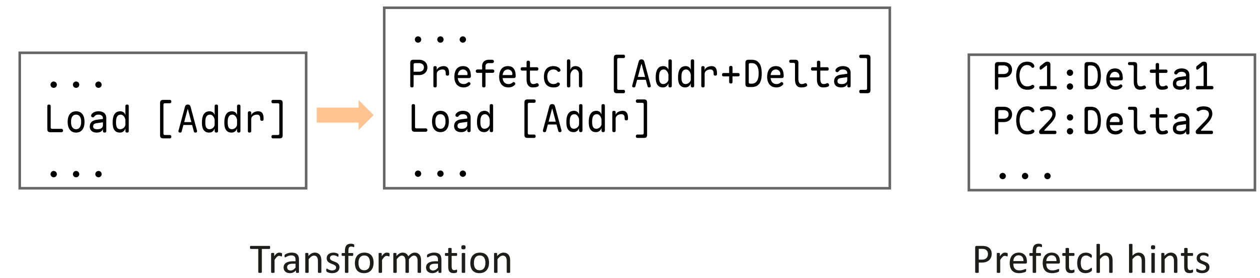
- Why Data Prefetching?
  > Top-Down Microarchitecture Analysis indicates a high memory bound and a low front-end bound in our target application.
  > Minor changes on prefetch instructions could lead to significant performance improvements (**3+%** in our case).
  > Data prefetching is a valuable optimization and a challenging engineering problem.

- How?
  > Add an experimental tool: `llvm-bolt-prefetch`

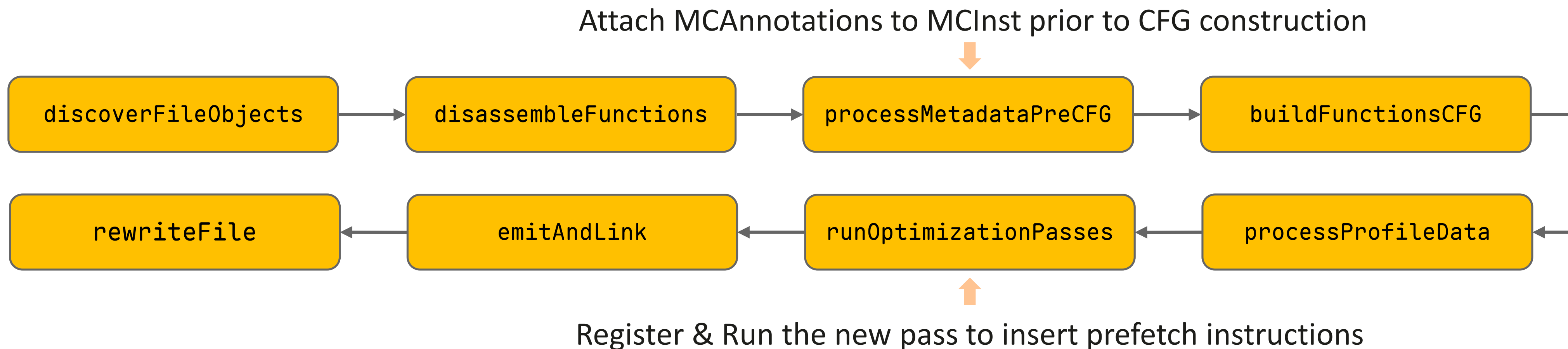
HUAWEI

Adding a Simple Prefetching Pass (LoadDataPrefetchPass)

- Basic Idea
  - > Insert **Prefetch [Addr+Delta]** before **Load [Addr]**.
  - > **Delta** is a constant value (positive/negative), indicating prefetch distance (an offset in byte).
  - > Simply use a text file containing <PC>:<Delta> pairs as prefetch hints.



- Changes in the Rewriting Pipeline



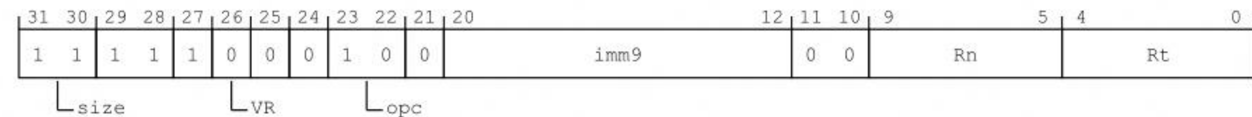
HUAWEI

Inserting AArch64 Prefetching Instructions for Different Loads

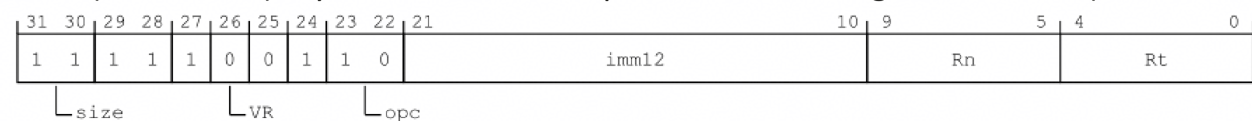
- This new pass requires a bit more work than I expected.
  ① Some prefetching address patterns can fit into one prefetch instruction on AArch64, others cannot.
  ② To move an immediate into a register, one or more instructions may be needed on AArch64. (#189304)
  ③ Various Load/Store/Prefetch addressing modes on AArch64 introduce additional complexity.

```
; For Delta=255
prfum   pldl1keep, [x0, #255]
ldurb   w9, [x0]

; For Delta=256
prfm   pldl1keep, [x0, #256]
ldurb   w9, [x0]

; For Delta=257 and x10 is usable
mov     x10, #257
prfm    pldl1keep, [x0, x10]
ldurb   w9, [x0]
```
①

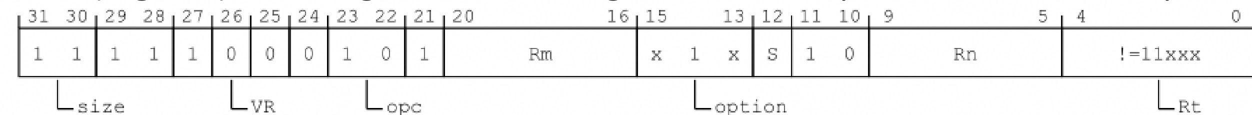
PRFUM: imm9 is used to encode a signed byte offset in the range -256 to 255



PRFM (immediate): byte offset is a multiple of 8 in the range 0 to 32760 (i.e. imm12 * 8)



PRFM (register): base register + offset register with an optional extend/shift specifier




```
; For Delta=0x10002, x10 is usable
movz    x10, #2
movk    x10, #1, lsl #16
prfm    pldl1keep, [x0, x10]
ldurb   w9, [x0]
```
②

```
; For Delta=0x10002 and x10 is usable
movz    x10, #2
movk    x10, #1, lsl #16
add     x10, x10, w1, uxtw #2
prfm    pldl1keep, [x0, x10]
ldr     w9, [x0, w1, uxtw #2]
```
③

HUAWEI

Using Dataflow Analysis & Liveness Analysis in BOLT

- It might be necessary to use a register to store the prefetching address on AArch64.
- BOLT is already equipped with a **dataflow-analysis framework** and **liveness analysis** support.
- Dataflow equation:

$$\mathrm{out}[n] = \bigcup_{s \in \mathrm{succ}[n]} \mathrm{in}[s]$$

$$\mathrm{in}[n] = \mathrm{use}[n] \bigcup (\mathrm{out}[n] - \mathrm{def}[n])$$

```
template <typename Derived, typename StateTy, bool Backward = false,
          typename StatePrinterTy = StatePrinter<StateTy>>
class DataflowAnalysis {
  ...
  void run(); // Public entry point of the entire dataflow analaysis
};

class LivenessAnalysis : public DataflowAnalysis<LivenessAnalysis, BitVector,
                                                 true, RegStatePrinter> {
  ...
  void run() { Parent::run(); }
  void doConfluence(BitVector &StateOut, const BitVector &StateIn) {
    StateOut |= StateIn;      // out[n] = union{in[s]} where s in succ[n]
  }
  BitVector computeNext(const MCInst &Point, const BitVector &Cur) {
    BitVector Next = Cur;
    BitVector Written = ...   // Registers written by current MCInst
    Written.flip();
    Next &= Written;
    BitVector Used = ...      // Registers used by current MCInst
    Next |= Used;
    return Next;              // in[n] = use[n] U (out[n] - def[n])
  }
  MCPhysReg scavengeRegAfter(ProgramPoint P) const;
};
```

- AArch64 liveness analysis in BOLT was broken when we started this proof-of-concept feature. We added some partial support internally.
- Glad to see a better version (#183298) was provided upstream recently! Glad to participate in the discussion and review!

ARM SPE (Statistical Profiling Extension)

- It could avoid **skid** in PMU sampling (caused by the delay from PMC overflow to the interrupt handler).
- It could precisely identify the **delinquent loads** (top-N loads with the highest cache-miss rate).
- Example: **perf record -e LLC-load-misses:uppp,arm_spe//u -- <benchmark>**

‘LLC-load-misses’ event from PMU sampling ➡

```
Percent
                    stp     x19, x20, [sp, #16]
          88:       ldr     w0, [sp, #124]
                    sxtw    x23, w0
                    mov     x20, x23
                    cmp     w0, w21
                  ↓ b.lt    ac
                  ↓ b       20c
   6.61   a0:       add     x20, x20, #0x1
                    cmp     w21, w20
                  ↓ b.le    150
   0.17   ac:       ldr     x0, [x27, #8]
                    lsl     x22, x20, #2
   1.61             ldr     w0, [x0, x20, lsl #2]
   0.29           ↑ cbz     w0, a0
   5.27             ldr     w0, [x25, x20, lsl #2]
                    tst     w0, #0xfffffffd
                  ↑ b.eq    a0
   2.66             ldr     x0, [x27, #16]
   0.06             ldr     w5, [x0, x22]
                    cmp     w5, #0x0
                  ↑ b.le    a0
   3.34             mov     x19, #0x0                       // #0
                  ↓ b       ec
  10.55   e0:       add     x19, x19, #0x1
   0.21             cmp     w5, w19
   0.13           ↑ b.le    a0
   3.88   ec:       ldr     x0, [x26, x20, lsl #3]
   1.57             ldr     w1, [x0, x19, lsl #2]
   0.16             sxtw    x3, w1
  57.92             ldr     w0, [x24, x3, lsl #2]
   0.75             cmp     w0, #0x1
   0.26           ↑ b.ne    e0
   0.30             mov     w2, #0x30                       // #48
   0.18             str     x3, [sp, #104]
   0.68             ldr     x0, [x27, #24]
                    smull   x28, w1, w2
                    add     x0, x0, x28
```

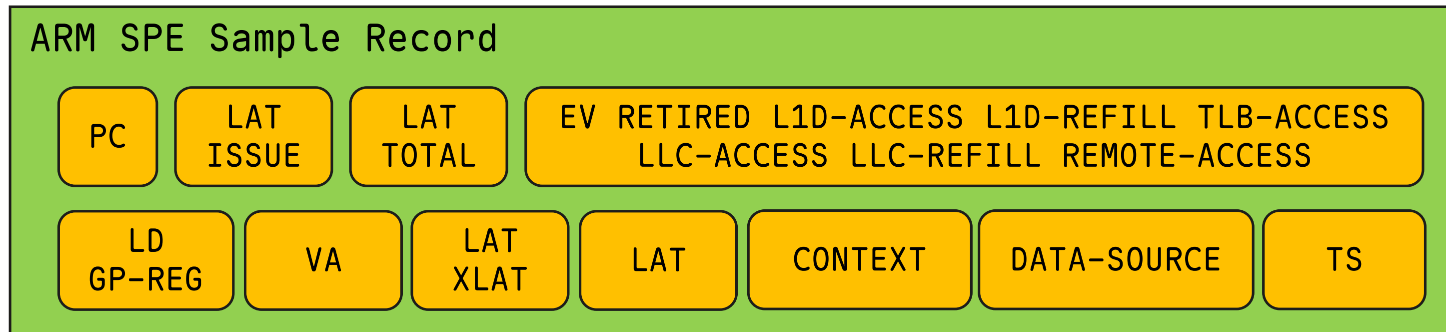
```
Percent
                    stp     x19, x20, [sp, #16]
          88:       ldr     w0, [sp, #124]
                    sxtw    x23, w0
                    mov     x20, x23
                    cmp     w0, w21
                  ↓ b.lt    ac
                  ↓ b       20c
          a0:       add     x20, x20, #0x1
                    cmp     w21, w20
                  ↓ b.le    150
          ac:       ldr     x0, [x27, #8]
                    lsl     x22, x20, #2
  21.01             ldr     w0, [x0, x20, lsl #2]
                  ↑ cbz     w0, a0
   0.05             ldr     w0, [x25, x20, lsl #2]
                    tst     w0, #0xfffffffd
                  ↑ b.eq    a0
                    ldr     x0, [x27, #16]
                    ldr     w5, [x0, x22]
                    cmp     w5, #0x0
                  ↑ b.le    a0
                    mov     x19, #0x0                       // #0
                  ↓ b       ec
          e0:       add     x19, x19, #0x1
                    cmp     w5, w19
                  ↑ b.le    a0
  14.46   ec:       ldr     x0, [x26, x20, lsl #3]
  57.08             ldr     w1, [x0, x19, lsl #2]
                    sxtw    x3, w1
                    ldr     w0, [x24, x3, lsl #2]
                    cmp     w0, #0x1
                  ↑ b.ne    e0
                    mov     w2, #0x30                       // #48
                    str     x3, [sp, #104]
                    ldr     x0, [x27, #24]
                    smull   x28, w1, w2
                    add     x0, x0, x28
```

⬅ **‘llc-miss’** event from ARM SPE sampling

HUAWEI

Prefetch Distance Estimation: Initial Approach

- ARM SPE provides various types of profiling information beyond just load PC addresses.
- This information could be leveraged to construct a cost-model for prefetch distance estimation.




```
enum arm_spe_pkt_type {
        ARM_SPE_BAD,
        ARM_SPE_PAD,
        ARM_SPE_END,
        ARM_SPE_TIMESTAMP,
        ARM_SPE_ADDRESS,
        ARM_SPE_COUNTER,
        ARM_SPE_CONTEXT,
        ARM_SPE_OP_TYPE,
        ARM_SPE_EVENTS,
        ARM_SPE_DATA_SOURCE,
};
```

> VA: The virtual address accessed by a sampled data memory accessing operation.

> EV: Event packets indicate events generated by the sampled operation.

> DATA-SOURCE: An implementation-defined value to indicate the loaded data source.

> ...

- We obtained some results, but encountered challenges:

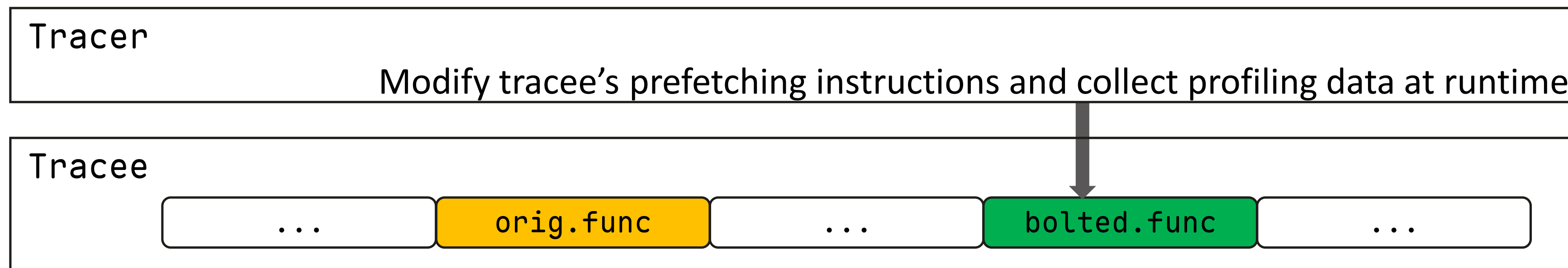
> Limited **event_filter** support before **FEAT_SPEv1p4** makes perf.data very large and hard to analyze.

> Difficult to estimate reasonable prefetch distance from memory access address patterns in discrete sample data.

HUAWEI

Prefetch Distance Estimation: Alternative Solution (WIP)

- Runtime prefetch distance adjustment with simultaneous effect observation
- llvm-bolt as “tracer”, target application as “tracee”




```
ptrace() allows the tracer to observe and control the tracee’s execution
  PTRACE_ATTACH, PTRACE_DETACH, PTRACE_CONT
  PTRACE_GETREGS, PTRACE_GETFPREGS
  PTRACE_SETREGS, PTRACE_SETFPREGS
  PTRACE_PEEKTEXT, PTRACE_PEEKDATA
  PTRACE_POKETEXT, PTRACE_POKEDATA

mprotect() changes the access protections for the calling process's memory pages
  PROT_READ     The memory can be read.
  PROT_WRITE    The memory can be modified.
  PROT_EXEC     The memory can be executed.
```

HUAWEI

Overview & Future Plans

- Overview of the `llvm-bolt-prefetch` tool:
  > Prefetch instruction insertion based on prefetch hints

  > Prefetch distance estimation from ARM SPE profiling information

  > Runtime prefetch distance adjustment with automatic profile collection

- Future plans:
  > Complete & open source the `llvm-bolt-prefetch` tool (track status here)

  > Enhance ARM SPE analysis for prefetch distance estimation

  > Explore more effective data prefetching solutions

  > ...

HUAWEI

Thank you.

把数字世界带入每个人、每个家庭、
每个组织，构建万物互联的智能世界。

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

Copyright©2018 Huawei Technologies Co., Ltd.
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. Huawei may change the information at any time without notice.