Integration Arm SPE in Perf for Memory Profiling

Leo Yan
Linaro Support and Solutions Engineering

Linaro Connect
VIRTUAL • SPRING 2021

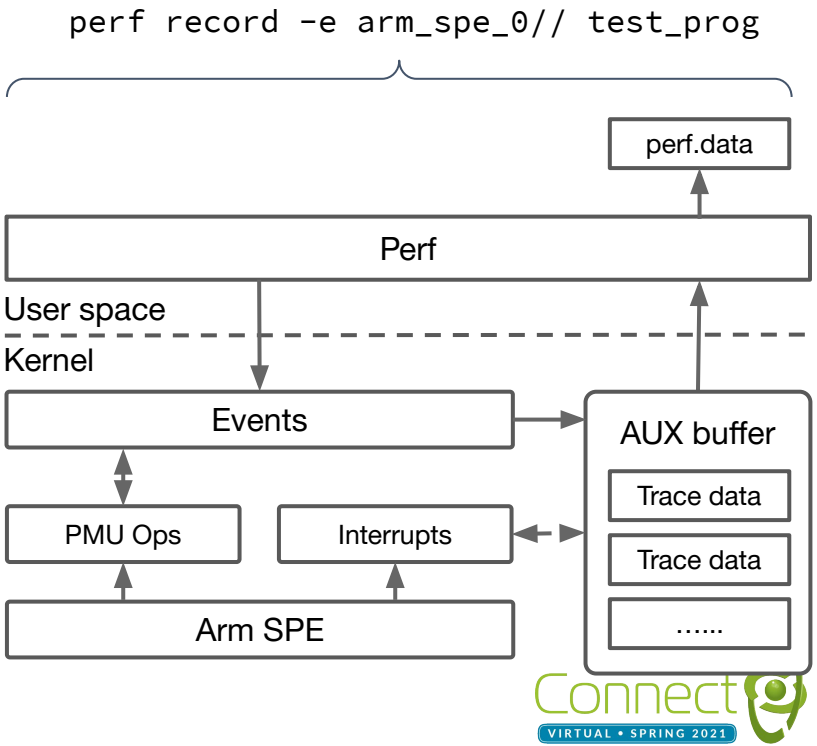
Introduction

Arm **Statistical** Profiling Extensions (SPE) is defined as part of Armv8-a architecture (starts from v8.2), which provides hardware based statistical sampling for CPUs.

SPE records operations (memory, exception, SVE, etc) and gathers associated information for the operation, like PC value, data address, event type, timestamp, etc. To avoid prominent overload caused by tracing, SPE uses statistical approach (e.g. random interval) and filter (like latency).

This session gives introduction for Linux supports Arm SPE with Perf tool.

Using Arm SPE with perf tool



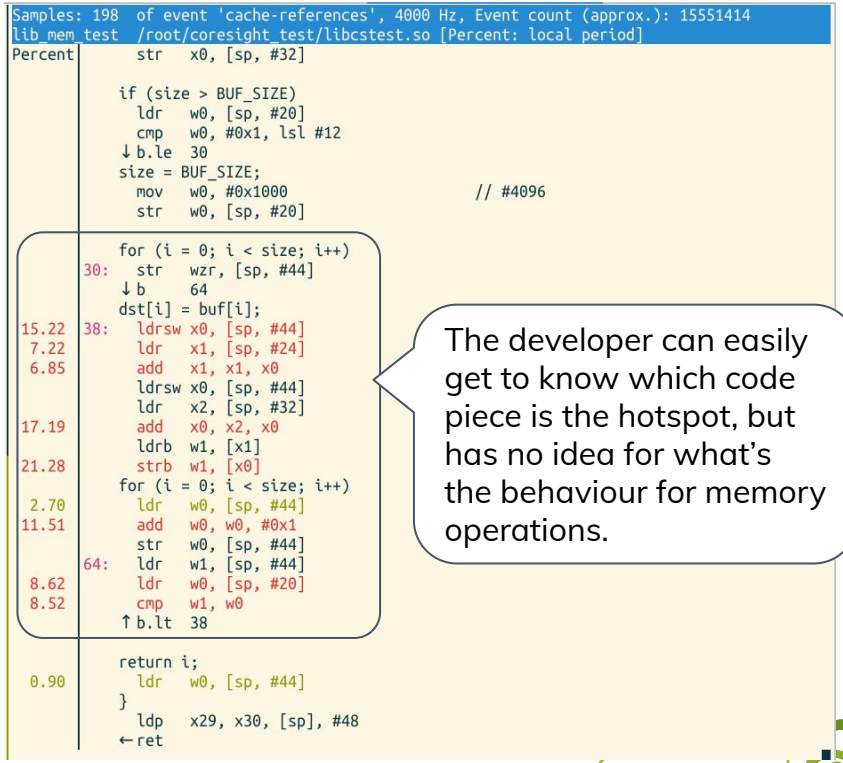
Agenda

- Why we need Arm SPE?
- Arm SPE hardware mechanism
- Integration Arm SPE with perf

What is missed from the standard PMU events?

If profile with the PMU events *cache-references* or *cache-misses*, the developer can get to know which code piece is the hotspot for memory accessing, but still has no idea which memory region accessing causes performance issue.

Arm PMU events doesn't provide any info for the memory accessing affiliated info, like cache level, remote accessing, TLB, etc, so developers have no chance to optimize memory accessing.



How to profile memory on x86?

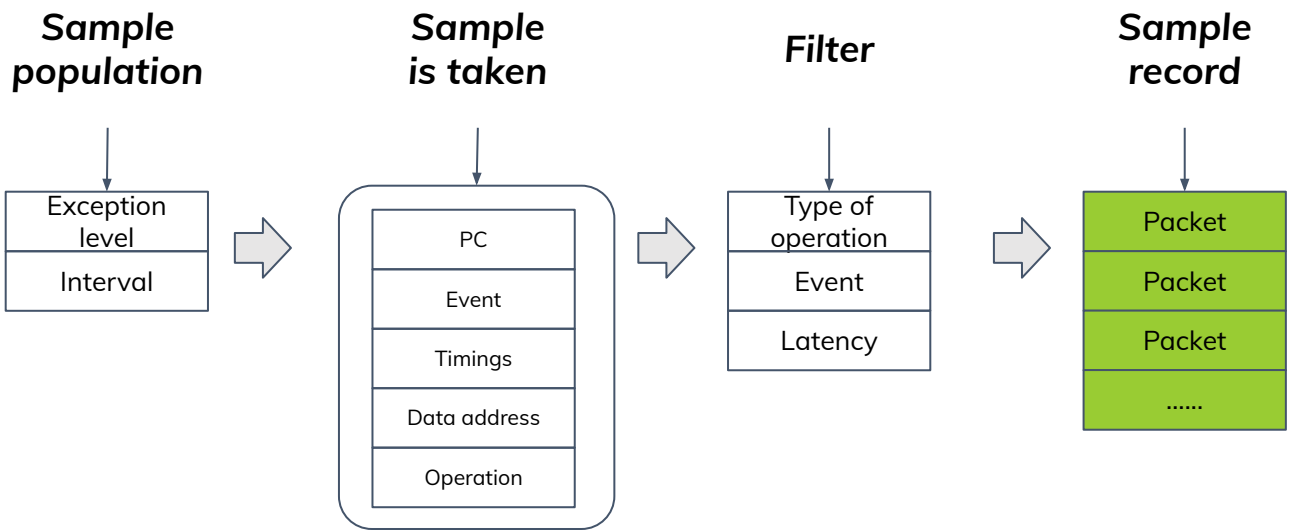
```
# ls /sys/devices/cpu/events/mem*
/sys/devices/cpu/events/mem-loads  /sys/devices/cpu/events/mem-stores

# perf mem record -t load,store -- false_sharing.exe 2
949 mticks, reader_thd (thread 3), on node 0 (cpu 2).
991 mticks, reader_thd (thread 2), on node 0 (cpu 1).
1111 mticks, lock_th (thread 1), on node 0 (cpu 3)
1120 mticks, lock_th (thread 0), on node
[ perf record: Woken up 1 times to write
[ perf record: Captured and wrote 0.

# perf mem report
```

But memory events are not supported by Arm CPUs. So this is one reason we want to enable Arm SPE for memory profiling on Arm platforms.

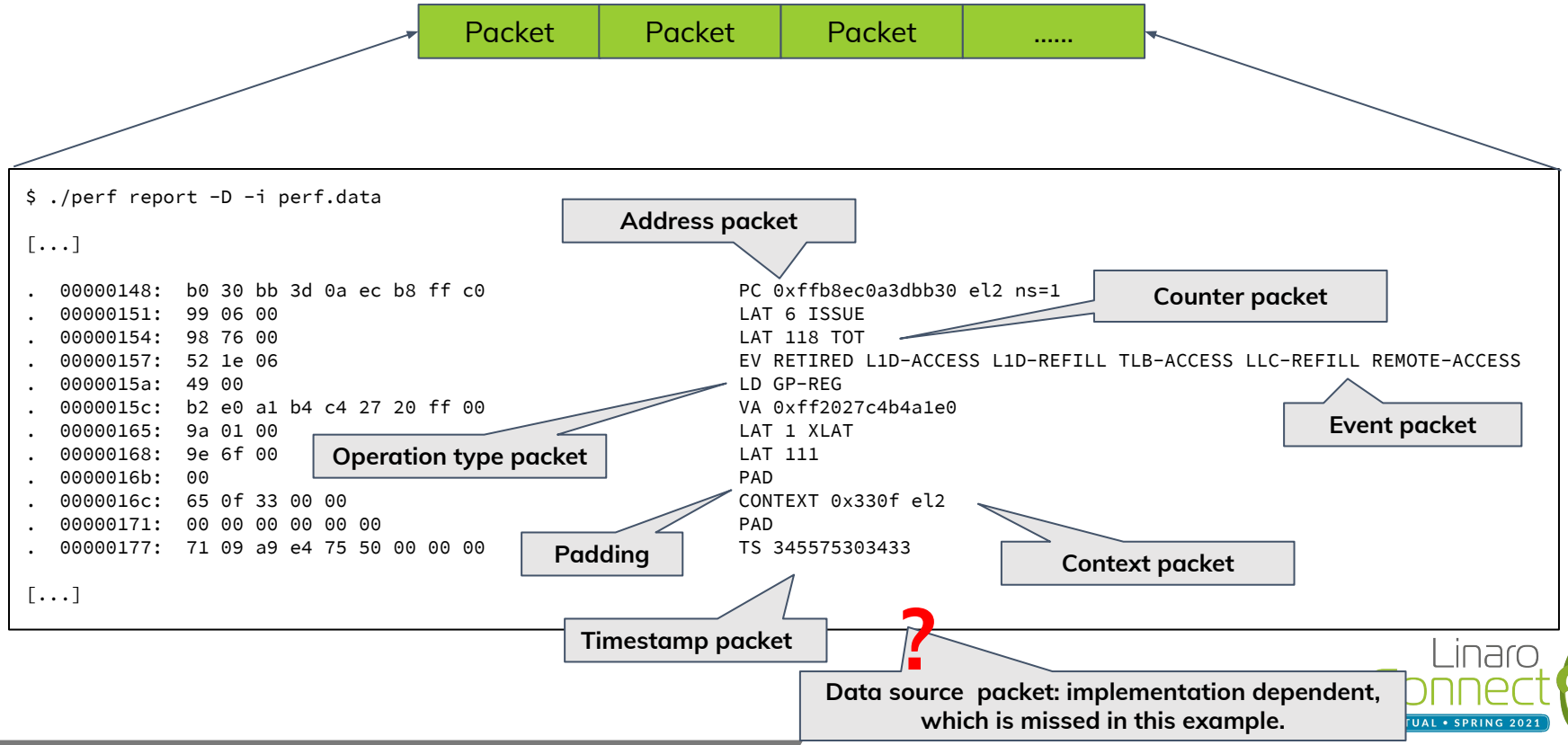
Agenda

- Why we need Arm SPE?
- Arm SPE hardware mechanism
- Integration Arm SPE with perf

Four stages hardware tracing in Arm SPE



Arm SPE Packets



Agenda

- Why we need Arm SPE?
- Arm SPE hardware mechanism
- Integration Arm SPE with perf

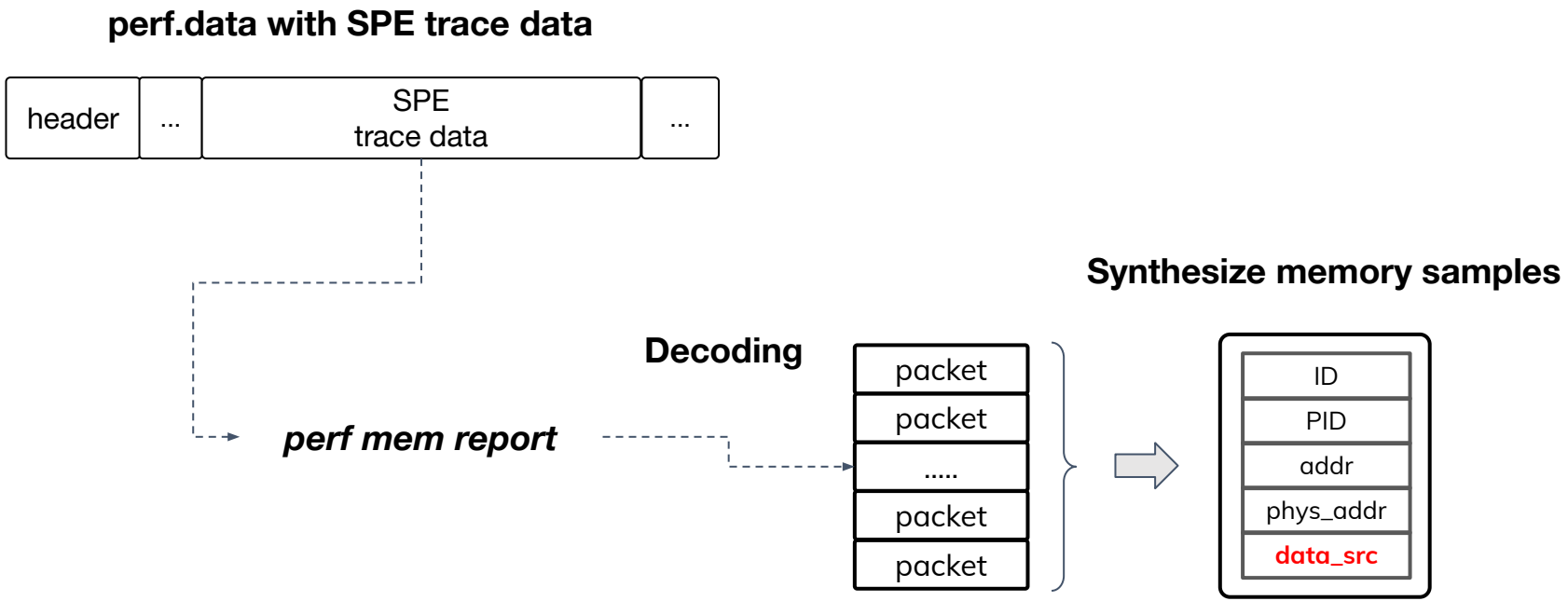
Enabling Perf memory events for Arm SPE

File tools/perf/arch/arm64/util/mem-events.c:

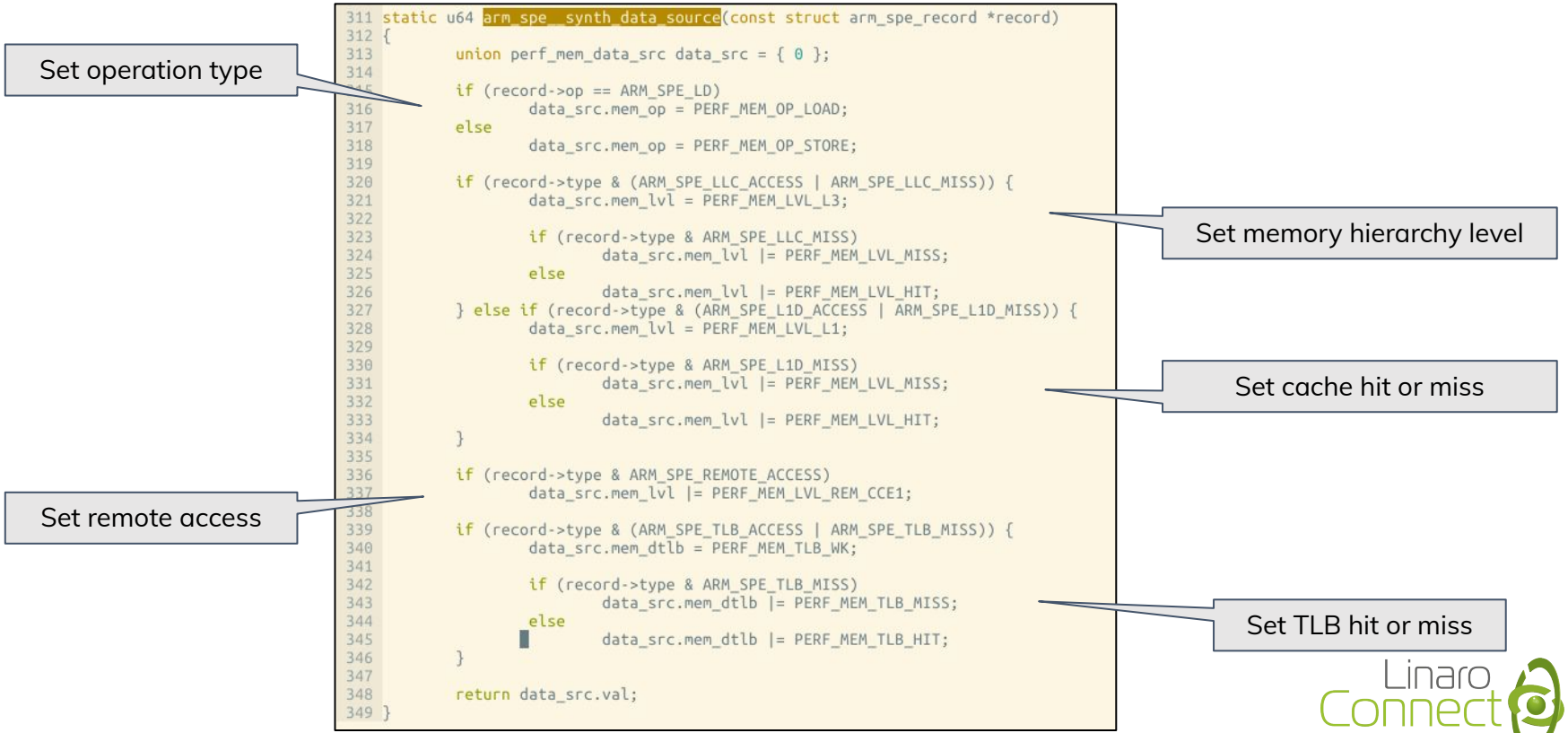
```
static struct perf_mem_event perf_mem_events[PERF_MEM_EVENTS__MAX] = {
        E("spe-load",   "arm_spe_0/ts_enable=1,load_filter=1,store_filter=0,min_latency=%u/",   "arm_spe_0"),
        E("spe-store",  "arm_spe_0/ts_enable=1,load_filter=0,store_filter=1/",                  "arm_spe_0"),
        E("spe-ldst",   "arm_spe_0/ts_enable=1,load_filter=1,store_filter=1,min_latency=%u/",   "arm_spe_0"),
};
```

```
# perf mem record -t load -- false_sharing.exe 2

# perf mem record -t store -- false_sharing.exe 2

# perf mem record -t load,store -- false_sharing.exe 2

# perf mem record -- false_sharing.exe 2   // This command is equivalent to '-t load,store'
```

Synthesization memory samples

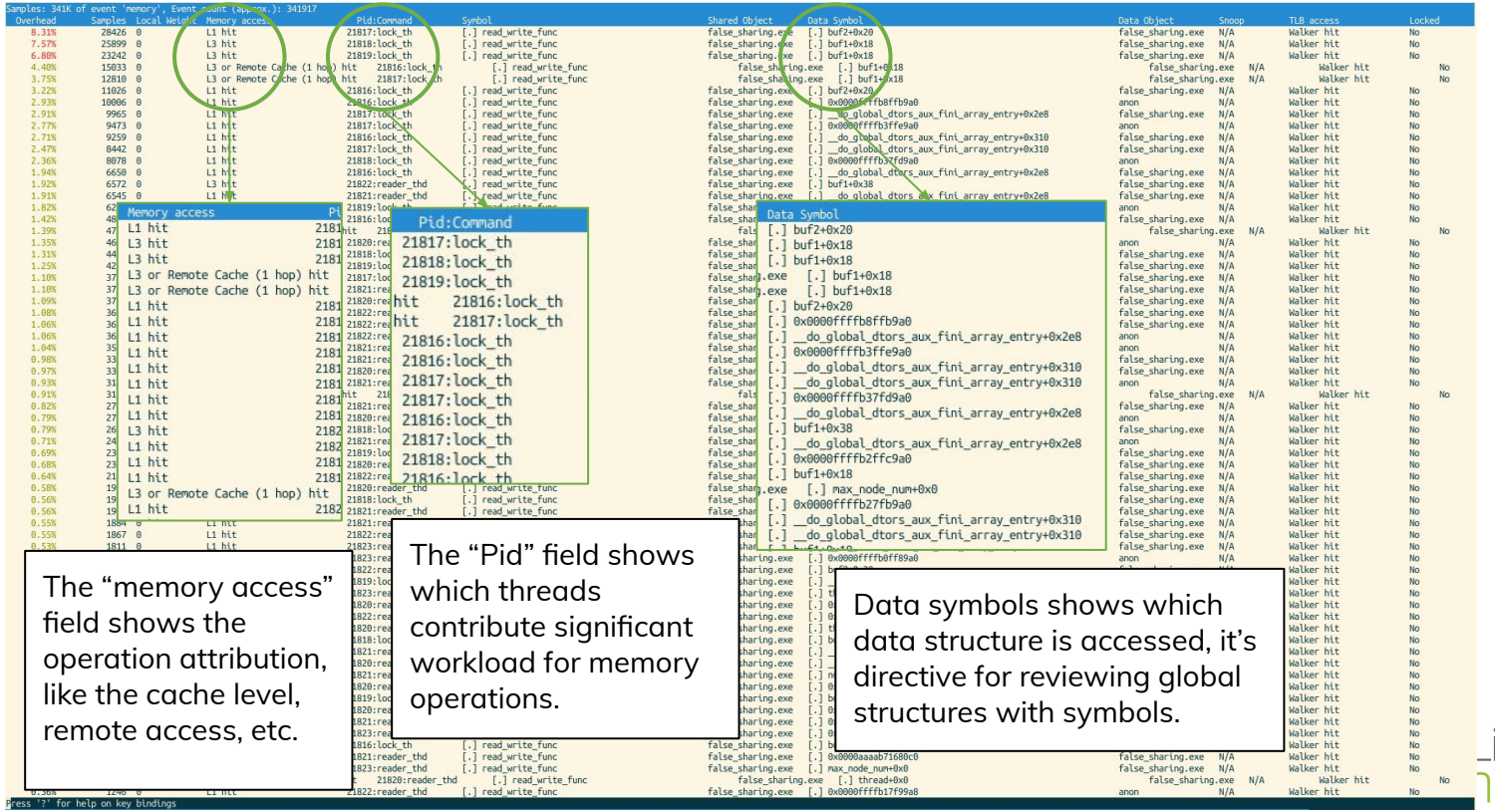


Synthesization data source field




```
311 static u64 arm_spe__synth_data_source(const struct arm_spe_record *record)
312 {
313         union perf_mem_data_src data_src = { 0 };
314
315         if (record->op == ARM_SPE_LD)
316                 data_src.mem_op = PERF_MEM_OP_LOAD;
317         else
318                 data_src.mem_op = PERF_MEM_OP_STORE;
319
320         if (record->type & (ARM_SPE_LLC_ACCESS | ARM_SPE_LLC_MISS)) {
321                 data_src.mem_lvl = PERF_MEM_LVL_L3;
322
323                 if (record->type & ARM_SPE_LLC_MISS)
324                         data_src.mem_lvl |= PERF_MEM_LVL_MISS;
325                 else
326                         data_src.mem_lvl |= PERF_MEM_LVL_HIT;
327         } else if (record->type & (ARM_SPE_L1D_ACCESS | ARM_SPE_L1D_MISS)) {
328                 data_src.mem_lvl = PERF_MEM_LVL_L1;
329
330                 if (record->type & ARM_SPE_L1D_MISS)
331                         data_src.mem_lvl |= PERF_MEM_LVL_MISS;
332                 else
333                         data_src.mem_lvl |= PERF_MEM_LVL_HIT;
334         }
335
336         if (record->type & ARM_SPE_REMOTE_ACCESS)
337                 data_src.mem_lvl |= PERF_MEM_LVL_REM_CCE1;
338
339         if (record->type & (ARM_SPE_TLB_ACCESS | ARM_SPE_TLB_MISS)) {
340                 data_src.mem_dtlb = PERF_MEM_TLB_WK;
341
342                 if (record->type & ARM_SPE_TLB_MISS)
343                         data_src.mem_dtlb |= PERF_MEM_TLB_MISS;
344                 else
345                         data_src.mem_dtlb |= PERF_MEM_TLB_HIT;
346         }
347
348         return data_src.val;
349 }
```

“perf mem report” with memory attributions



The “memory access” field shows the operation attribution, like the cache level, remote access, etc.

The “Pid” field shows which threads contribute significant workload for memory operations.

Data symbols shows which data structure is accessed, it’s directive for reviewing global structures with symbols.

Let's move! - "perf c2c" with HITM tags on x86

```
# perf c2c record -- false_sharing.exe 2
# perf c2c report
```

Shared Data Cache Line Table

```
Shared Data Cache Line Table     (2 entries, sorted on Total HITMs)
           ---------- Cacheline ----------      Tot  ------- Load Hitm -------  Total    Total    Total  ---- Stores ----  ----- Core Load Hit -----  - LLC Load Hit --  - RMT Load Hit --  --- Load Dram ----
Index            Address  Node  PA cnt     Hitm    Total  LclHitm  RmtHitm  records   Loads   Stores   L1Hit   L1Miss      FB      L1      L2  LclHit  LclHitm  RmtHit  RmtHitm     Lcl     Rmt
    0  0x55e076fea100     0    1465     87.42%      528      528        0     7108    3802     3306    2585      721     485    2566      57     165     528       0        0       1       0
    1  0x55e076fea080     0       1     12.58%       76       76        0      654     654        0       0        0     154     376      42       6      76       0        0       0       0
```

Press 'd' to display cache line details.

If the hardware memory event supports HITM tags, it's straightforward to locate which cache line is accessed frequently with its modified copy.

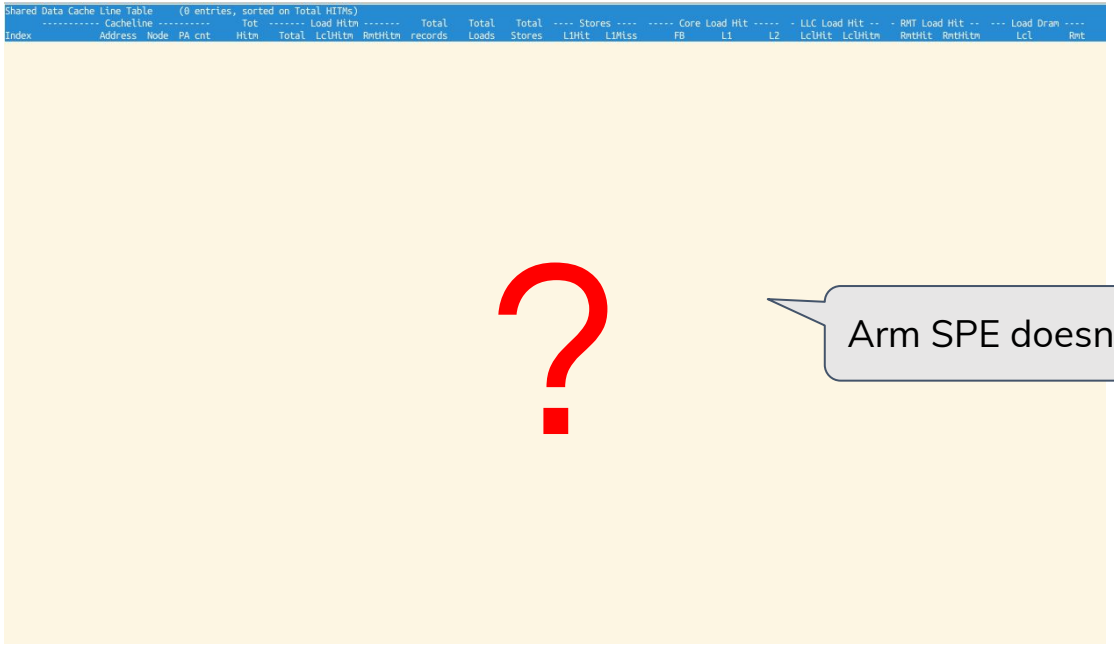
Shared Cache Line Distribution Pareto Table

```
Cacheline 0x55e076fea100
----- HITM -----  -- Store Refs --  ------- CL --------                    ---------- cycles ----------   Total    cpu                                Shared
RmtHitm  LclHitm   L1 Hit  L1 Miss     Off  Node  PA cnt      Code address  rmt hitm  lcl hitm    load   records    cnt             Symbol             Object                Source:Line  Node
  0.00%   68.18%    0.00%    0.00%     0x0     0       1  0x55e076de8c1d         0      2069    1012      1473      3  [.] read_write_func  false_sharing.exe  false_sharing_example.c:146     0
  0.00%   10.23%   95.67%    0.00%     0x0     0       1  0x55e076de8c16         0      1804    1202      3961      3  [.] read_write_func  false_sharing.exe  false_sharing_example.c:145     0
  0.00%    0.00%    4.33%  100.00%     0x0     0       1  0x55e076de8c28         0         0       0       833      3  [.] read_write_func  false_sharing.exe  false_sharing_example.c:146     0
  0.00%   21.59%    0.00%    0.00%    0x20     0       1  0x55e076de8c73         0       158      90       841      2  [.] read_write_func  false_sharing.exe  false_sharing_example.c:155     0
```

In the detailed cache line view, it shows which source lines access the same cache line, and what's the workloads is caused by HITM or store references.

“perf c2c” with Arm SPE

```
# perf c2c record -- false_sharing.exe 2
# perf c2c report
```



Experiment: “perf c2c” with option “-d all”

```
# perf c2c report -d all --coalesce tid,pid,iaddr,dso
```

Shared Data Cache Line Table

Shared Data Cache Line Table (15 entries, sorted on All Load Access)

	Cacheline			Load Hit	Load Hit	Total	Total	Total	Stores		Core Load Hit			LLC Load Hit		RMT Load Hit		Load Dram	
Index	Address	Node	PA cnt	Pct	Total	records	Loads	Stores	L1Hit	L1Miss	FB	L1	L2	LclHit	LclHitm	RmtHit	RmtHitm	Lcl	Rmt
0	0xaaaada771fc0	N/A	0	26.36%	39113	39113	39113	0	0	0	0	39113	0	0	0	0	0	0	0
1	0xaaaada771f80	N/A	0	24.77%	36750	36750	36750	0	0	0	0	36750	0	0	0	0	0	0	0
2	0xaaaada761480	N/A	0	9.07%	13462	13462	13462	0	0	0	0	13462	0	0	0	0	0	0	0
3	0xaaaada772100	N/A	0	7.43%	11016	11016	11016	0	0	0	0	11016	0	0	0	0	0	0	0
4	0xffff92ffc980	N/A	0	5.85%	8686	8686	8686	0	0	0	0	8686	0	0	0	0	0	0	0
5	0xffff93ffe980	N/A	0	5.15%	7640	7640	7640	0	0	0	0	7640	0	0	0	0	0	0	0
6	0xaaaada772000	N/A	0	4.61%	6837	6837	6837	0	0	0	0	6837	0	0	0	0	0	0	0
7	0xffff937fd980	N/A	0	4.55%	6757	6757	6757	0	0	0	0	6757	0	0	0	0	0	0	0
8	0xffffa88a7980	N/A	0	4.38%	6503	6503	6503	0	0	0	0	6503	0	0	0	0	0	0	0
9	0xaaaada772080	N/A	0	2.93%	4351	4351	4351	0	0	0	0	4351	0	0	0	0	0	0	0
10	0xaaaada7720c0	N/A	0	1.80%	2665	2665	2665	0	0	0	0	2665	0	0	0	0	0	0	0
11	0xffffa90a8980	N/A	0	0.96%	1425	1425	1425	0	0	0	0	1425	0	0	0	0	0	0	0
12	0xffffa98a9980	N/A	0	0.96%	1420	1420	1420	0	0	0	0	1420	0	0	0	0	0	0	0
13	0xffffaa8ab980	N/A	0	0.49%	723	723	723	0	0	0	0	723	0	0	0	0	0	0	0
14	0xffffaa0aa980	N/A	0	0.41%	607	607	607	0	0	0	0	607	0	0	0	0	0	0	0

Experiment: “perf c2c” with option “-d all” - cont.

```
# perf c2c report -d all --coalesce tid,pid,iaddr,dso
```

Shared Cache Line Distribution Pareto Table

Cacheline 0xaaaada771fc0

-- Load Refs -- Hit	Load Refs -- Miss	-- Store Refs -- L1 Hit	-- Store Refs -- L1 Miss	------- CL ------- Off	Node	PA cnt	Pid	Tid	Code address	---------- cycles ---------- rmt hitm	lcl hitm	load	Total records	cpu cnt	Symbol	Shared Object	Source:Line	Node
6.26%	0.00%	0.00%	0.00%	0x0	N/A	0	2380	2387:reader_thd	0xaaaada760dfc	0	0	0	2448	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:152	1
4.08%	0.00%	0.00%	0.00%	0x0	N/A	0	2380	2389:reader_thd	0xaaaada760dfc	0	0	0	1594	2	[.] read_write_func	false_sharing.exe	false_sharing_example.c:152	3
2.91%	0.00%	0.00%	0.00%	0x0	N/A	0	2380	2388:reader_thd	0xaaaada760dfc	0	0	0	1138	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:152	2
1.91%	0.00%	0.00%	0.00%	0x0	N/A	0	2380	2386:reader_thd	0xaaaada760dfc	0	0	0	746	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:152	0
10.60%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2388:reader_thd	0xaaaada760e4c	0	0	0	4145	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:155	2
8.28%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2384:lock_th	0xaaaada760ddc	0	0	0	3237	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:146	2
7.32%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2385:lock_th	0xaaaada760ddc	0	0	0	2864	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:146	3
6.30%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2384:lock_th	0xaaaada760dcc	0	0	0	2464	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:146	2
5.71%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2382:lock_th	0xaaaada760ddc	0	0	0	2232	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:146	0
5.51%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2385:lock_th	0xaaaada760dcc	0	0	0	2156	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:146	3
5.41%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2384:lock_th	0xaaaada760db0	0	0	0	2116	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:145	2
5.31%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2385:lock_th	0xaaaada760db0	0	0	0	2077	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:145	3
5.30%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2389:reader_thd	0xaaaada760e78	0	0	0	2073	2	[.] read_write_func	false_sharing.exe	false_sharing_example.c:159	3
5.29%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2383:lock_th	0xaaaada760ddc	0	0	0	2071	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:146	1
4.54%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2386:reader_thd	0xaaaada760e78	0	0	0	1774	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:159	0
4.29%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2387:reader_thd	0xaaaada760ea4	0	0	0	1676	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:163	1
3.09%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2382:lock_th	0xaaaada760dcc	0	0	0	1209	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:146	0
2.93%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2383:lock_th	0xaaaada760dcc	0	0	0	1147	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:146	1
2.65%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2382:lock_th	0xaaaada760db0	0	0	0	1035	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:145	0
2.33%	0.00%	0.00%	0.00%	0x20	N/A	0	2380	2383:lock_th	0xaaaada760db0	0	0	0	911	1	[.] read_write_func	false_sharing.exe	false_sharing_example.c:145	1

For the store samples, since Arm SPE doesn’t give out any memory hierarchy information, like L1 hit/miss or LLC hit/miss, thus the cache line distribution doesn’t show any statistics for store operations.

Recap

- Arm SPE has been enabled with perf tool for below sub commands
  - *perf record / perf report / perf script*
  - *perf mem record / perf mem report*
- Arm SPE is found the memory hierarchy info is missed for store ops
  - *perf c2c* has not yet supported for Arm SPE on the mainline kernel
  - https://lore.kernel.org/patchwork/cover/1353064/
    Only partial patches have been merged for “perf c2c” refactoring; the patches for extension display option “all” are left out.
- Arm SPE PID tracing can only support the root namespace
  - If using the `CONTEXTIDR_EL1/EL2` for PID tracing, it only can support tracing PID in the root namespace and it’s possible to leak info for non-root namespace tracing;
  - So far only support PID tracing for root namespace.
  - https://lore.kernel.org/patchwork/patch/1367664/

Acknowledgement

Al Grant (Arm)

Haojian Zhuang (Linaro)

James Clark (Arm)

Michael Williams (Arm)

Thank you

Accelerating deployment in the Arm Ecosystem

Leo Yan <leo.yan@linaro.org>

Linaro Connect
VIRTUAL • SPRING 2021