Integration Arm SPE in Perf for Memory Profiling

Leo Yan
Linaro Support and Solutions Engineering
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

```
perf record -e arm_spe_0// test_prog
```

```
perf.data
```

```
AUX buffer
```

```
Trace data
```

```
......
```

```
Arm SPE
```
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.

The developer can easily get to know which code piece is the hotspot, but has no idea for what’s the behaviour for memory operations.
How to profile memory on x86?

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

```bash
# perf mem record -t load,store -- false_sharing.exe
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 0.
[ perf record: Woken up 1 times to write ]
[ perf record: Captured and wrote 0.763 MB perf.data (10645 samples) ]

# 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

Sample population
- Exception level
- Interval

Sample is taken
- PC
- Event
- Timings
- Data address
- Operation

Filter
- Type of operation
- Event
- Latency

Sample record
- Packet
- Packet
- Packet
- ......
Arm SPE Packets

$ ./perf report -D -i perf.data

[...]

- 00000148: b0 30 bb 3d 0a ec b8 ff c0
- 00000151: 99 06 00
- 00000154: 98 76 00
- 00000157: 52 1e 06
- 0000015a: 49 00
- 0000015c: b2 e0 a1 b4 c4 27 20 ff 00
- 00000165: 9a 01 00
- 00000168: 9e 6f 00
- 0000016b: 00
- 0000016c: 65 0f 33 00 00
- 00000171: 00 00 00 00 00 00
- 00000177: 71 09 a9 e4 75 50 00 00 00

[...]
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:

```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

**perf.data with SPE trace data**

- header
- SPE trace data

**Decoding**

- packet
- packet
- packet
- packet

**Synthesize memory samples**

- ID
- PID
- addr
- phys_addr
- data_src

*perf mem report*
Synthesization data source field

```c
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_L1D_ACCESS | ARM_SPE_L1D_MISS)) {
321     data_src.mem_lvl = PERF_MEM_LVL_L1;
322     if (record->type & ARM_SPE_L1D_MISS)
323     data_src.mem_lvl |= PERF_MEM_LVL_MISS;
324     else
325     data_src.mem_lvl |= PERF_MEM_LVL_HIT;
326     } else if (record->type & (ARM_SPE_L1D_ACCESS | ARM_SPE_L1D_MISS)) {
327     data_src.mem_lvl = PERF_MEM_LVL_L1;
328     if (record->type & ARM_SPE_L1D_MISS)
329     data_src.mem_lvl |= PERF_MEM_LVL_MISS;
330     else
331     data_src.mem_lvl |= PERF_MEM_LVL_HIT;
332     }
333     
334     if (record->type & ARM_SPE_REMOTE_ACCESS)
335     data_src.mem_lvl |= PERF_MEM_LVL_MEM_CCE;
336     
337     if (record->type & (ARM_SPE_TLB_ACCESS | ARM_SPE_TLB_MISS)) {
338     data_src.dtlb = PERF_MEM_TLB_MISS;
339     if (record->type & ARM_SPE_TLB_MISS)
340     data_src.dtlb |= PERF_MEM_TLB_MISS;
341     else
342     data_src.dtlb |= PERF_MEM_TLB_HIT;
343     }
344     
345     return data_src.val;
346 }
```
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

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

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

Press ‘d’ to display cache line details.

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

Arm SPE doesn’t support HITM!
Experiment: “perf c2c” with option “-d all”

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

## Shared Data Cache Line Table

<table>
<thead>
<tr>
<th>Index</th>
<th>Address</th>
<th>Node</th>
<th>PA cnt</th>
<th>L1 Hit</th>
<th>Load Hit</th>
<th>Total Loads</th>
<th>Total Stores</th>
<th>L1 Hit</th>
<th>L1 Miss</th>
<th>Core Load Hit</th>
<th>LLC Load Hit</th>
<th>RMT Load Hit</th>
<th>Load Dram</th>
<th>L1</th>
<th>L2</th>
<th>LcL</th>
<th>Rmt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0xaaaaa771f80</td>
<td>N/A</td>
<td>0</td>
<td>26.34%</td>
<td>39113</td>
<td>39113</td>
<td>39113</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0xaaaaa771f80</td>
<td>N/A</td>
<td>0</td>
<td>24.77%</td>
<td>36750</td>
<td>36750</td>
<td>36750</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0xaaaaa761480</td>
<td>N/A</td>
<td>0</td>
<td>9.67%</td>
<td>13462</td>
<td>13462</td>
<td>13462</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0xaaaaa7722080</td>
<td>N/A</td>
<td>0</td>
<td>7.43%</td>
<td>11016</td>
<td>11016</td>
<td>11016</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0xffff952f0980</td>
<td>N/A</td>
<td>0</td>
<td>5.85%</td>
<td>8686</td>
<td>8686</td>
<td>8686</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0xffff93fefe980</td>
<td>N/A</td>
<td>0</td>
<td>5.15%</td>
<td>7640</td>
<td>7640</td>
<td>7640</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0xaaaaa772080</td>
<td>N/A</td>
<td>0</td>
<td>4.61%</td>
<td>6837</td>
<td>6837</td>
<td>6837</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0xffff93f9d896</td>
<td>N/A</td>
<td>0</td>
<td>4.55%</td>
<td>6757</td>
<td>6757</td>
<td>6757</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0xffffab847980</td>
<td>N/A</td>
<td>0</td>
<td>4.38%</td>
<td>6593</td>
<td>6593</td>
<td>6593</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0xaaaaa772080</td>
<td>N/A</td>
<td>0</td>
<td>2.93%</td>
<td>4351</td>
<td>4351</td>
<td>4351</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0xaaaaa772080</td>
<td>N/A</td>
<td>0</td>
<td>1.80%</td>
<td>2663</td>
<td>2663</td>
<td>2663</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0xffff98a898980</td>
<td>N/A</td>
<td>0</td>
<td>0.96%</td>
<td>1428</td>
<td>1428</td>
<td>1428</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0xffff98a99980</td>
<td>N/A</td>
<td>0</td>
<td>0.96%</td>
<td>1428</td>
<td>1428</td>
<td>1428</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0xffffab89a980</td>
<td>N/A</td>
<td>0</td>
<td>0.49%</td>
<td>723</td>
<td>723</td>
<td>723</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0xffffab9aa980</td>
<td>N/A</td>
<td>0</td>
<td>0.41%</td>
<td>607</td>
<td>607</td>
<td>607</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Experiment: “perf c2c” with option “-d all” - cont.

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

Shared Cache Line Distribution Pareto Table

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/](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/](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>