Linux perf_events updates

Stephane Eranian
Scalable Tools Workshop 2018
Solitude, UT
Agenda

- Quick updates on perf_events
- Cache QoS
- Code heat maps
- Branch mispredictions
Perf_events news

- **PERF_SAMPLE_PHYS_ADDR (v4.14)**
  - Extract data physical address, useful for cache simulation

- **Spectre-v1 (v4.17) protection**
  - Code impact: eliminate array indexing in speculative code

- **Intel Skylake uncore IIO PMU free running counters PMUs (v4.17)**
  - PCIe bandwidth, utilization, IIO clockticks
  - Ex: `perf stat -a -I 1000 -e uncore_iio_free_running_0/bw_in_port0/

- **PMU capabilities exported in sysfs (v4.14)**
  - `% grep . /sys/bus/event_source/devices/cpu/caps/*`
    - `/sys/bus/event_source/devices/cpu/caps/branches:32`
    - `/sys/bus/event_source/devices/cpu/caps/max_precise:3`
    - `/sys/bus/event_source/devices/cpu/caps/pmu_name:skylake`

- Event scheduling improvements
- Lots of bugs fixes
Cache QoS introduction

- Monitor cache and memory subsystem utilization per process
  - Track offenders

- Enforce cache and memory subsystem utilization restrictions per process
  - Isolate offenders

- Useful in shared runtime environments (Cloud, Web)
  - Machine is shared by workloads with different memory usage or SLO

- Hardware support introduced with Intel Haswell EP
Cache QoS: cache occupancy

- **Monitor cache usage (CMT) - Intel Haswell EP**
  - Track **new cache line allocations** (loads, rfo, hw/sw prefetch)
  - Supported in L3 (L2 on some processors)
  - Tag process with RMID
  - 4 RMID per core on Haswell, 8 on later processors
  - RMID saved/restored on context switch

- **Cache partitioning (CAT) - Intel Haswell EP**
  - Enforce cache utilization restrictions
  - Can split Code vs. Data (CDP) on Broadwell EP and later
  - 4 CLOSID on Haswell, 16 on later processors (8 with CDP enabled)
  - Restriction enforced per way (not all combinations possible)
  - CLOSID saved/restored on context switch
Cache QoS: memory bandwidth

- **Monitor memory bandwidth usage (MBM) - Intel Broadwell EP**
  - Count cache lines read and written back from LLC
  - Count total vs. local socket traffic
  - Tag process with RMID
  - 8 RMID per core
  - RMID saved/restored on context switch

- **Memory bandwidth allocations (MBA) - Intel Skylake X**
  - Restrict by percentage of total peak bandwidth per core
  - Tag process with CLOSID
  - 8 CLOSID
  - CLOSID saved/restored on context switch
  - Control enforced per core (not per socket), both hyperthread impacted
  - Control applied between L2 -> L3 => L3 hits are also slowed down
Cache Qos: Linux support

- New interface introduced in Linux v4.9 (Intel contribution)
  - Using a new filesystem: resctrl
  - No cgroup, perf_events, perf tool connections anymore
  - Old interface deprecated

- Retain principles of cgroup fs
  - A global default resource group (root of fs)
  - One subdir per resource group
  - Tasks are moved into resource groups with: `echo tid > tasks`
  - Monitoring data read via specific file entry

- Enforcement via a programmable text-based schemata
  ```
  # cat schemata
  L3:0=1ff;1=1ff
  MB:0=100;1=100
  ```
Cache QoS: example

- **Cache Monitoring:** Read cache occupancy on socket0 for `my_grp`:
  
  ```bash
  $ mkdir /sys/fs/resctrl/my_grp
  $ echo $$ >/sys/fs/resctrl/my_grp/tasks (move my shell into my_grp)
  $ do_some_work
  $ cat /sys/fs/resctrl/my_grp/mon_data/mon_L3_00/llc_occpuancy
  ```

- **Bandwidth Monitoring:** Read local memory bandwidth (read/write)
  
  ```bash
  $ mkdir /sys/fs/resctrl/my_grp
  $ echo $$ >/sys/fs/resctrl/my_grp/tasks
  $ do_some_work &
  $ cat /sys/fs/resctrl/my_grp/mon_data/mon_L3_00/mbm_local_bytes
  ```

- **RMID Limit for CMT**
  - RMID must be recycled
  - Cannot reset a RMID, must wait for line evictions
  - Kernel keeps list of RMID in limbo
Cache QoS: Triad example on Broadwell EP

$ mkdir /sys/fs/resctrl/memtoy; echo $$ >/sys/fs/resctrl/memtoy/tasks
$ triad -i 0 -r 0 -l 3 & (streaming 3 buffers of 256 MiB each)
$ cd /sys/fs/resctrl/memtoy/mon_data/mon_L3_00/
$ cat llc_occupancy
40260672  (40MB BDX L3 cache size)
$ cat mbm_local_bytes; sleep 1; cat mbm_local_bytes
1010472443904
1023447851008
(1023447851008-1010472443904)/(1000*1000)=12975 MB/S
$ perf guncore -M mem (uses CHA to compute local vs. remote traffic)

#-----------------------------------------
#                 Socket0                |
#-----------------------------------------
#     Local     Local    Remote    Remote|
#        Wr        Rd        Wr        Rd|
#      MB/s      MB/s      MB/s      MB/s|
#-----------------------------------------
3198.76   9594.92   0.12    1.09
PMU based code heat map

- Where is the hot code?
- Why?
  - Improve code layout = Minimize ITLB pressure
  - Use large code pages (2MB or larger)
  - Do not exceed L1 ITLB 2MB entries capacity
  - Demote useless 2MB pages (save physical memory)
- How?
  - Using PMU based sampling
  - Intel: `INST_RETIRED:PREC_DIST` + PEBS or LBR
- Why `INST_RETIRED:PREC_DIST (0x01c0)`?
  - Introduced in Intel Sandy Bridge
  - HW assist to mitigate bias due to shadow of long latency instructions
<table>
<thead>
<tr>
<th>page address</th>
<th>#TLB 2MB</th>
<th>Total %</th>
<th>Cum %</th>
<th>Page size</th>
<th>pages used</th>
<th>util</th>
<th>left</th>
<th>lines used</th>
<th>util</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x138000000</td>
<td>1</td>
<td>72.93%</td>
<td>72.93%</td>
<td>2MB</td>
<td>471</td>
<td>91.99%</td>
<td>41</td>
<td>12952</td>
<td>39.53%</td>
<td>application</td>
</tr>
<tr>
<td>0x13a000000</td>
<td>2</td>
<td>6.87%</td>
<td>79.80%</td>
<td>2MB</td>
<td>109</td>
<td>21.29%</td>
<td>403</td>
<td>2855</td>
<td>8.71%</td>
<td>application</td>
</tr>
<tr>
<td>0x136000000</td>
<td>3</td>
<td>4.34%</td>
<td>84.14%</td>
<td>2MB</td>
<td>160</td>
<td>31.25%</td>
<td>352</td>
<td>4298</td>
<td>13.12%</td>
<td>application</td>
</tr>
<tr>
<td>0x40000000</td>
<td>4</td>
<td>3.55%</td>
<td>87.69%</td>
<td>2MB</td>
<td>507</td>
<td>99.02%</td>
<td>5</td>
<td>15393</td>
<td>46.98%</td>
<td>application</td>
</tr>
<tr>
<td>0xa0000000</td>
<td>5</td>
<td>2.04%</td>
<td>89.73%</td>
<td>2MB</td>
<td>97</td>
<td>18.95%</td>
<td>415</td>
<td>336</td>
<td>1.03%</td>
<td>application</td>
</tr>
<tr>
<td>0x144000000</td>
<td>6</td>
<td>1.89%</td>
<td>91.62%</td>
<td>2MB</td>
<td>3</td>
<td>0.59%</td>
<td>509</td>
<td>36</td>
<td>0.11%</td>
<td>application</td>
</tr>
<tr>
<td>0xa6000000</td>
<td>7</td>
<td>1.61%</td>
<td>93.24%</td>
<td>2MB</td>
<td>4</td>
<td>0.78%</td>
<td>508</td>
<td>49</td>
<td>0.15%</td>
<td>application</td>
</tr>
<tr>
<td>0x60000000</td>
<td>8</td>
<td>1.42%</td>
<td>94.66%</td>
<td>2MB</td>
<td>491</td>
<td>95.90%</td>
<td>21</td>
<td>11983</td>
<td>36.57%</td>
<td>application</td>
</tr>
<tr>
<td>0xffffffff81000000</td>
<td>9</td>
<td>1.31%</td>
<td>95.97%</td>
<td>2MB</td>
<td>373</td>
<td>72.85%</td>
<td>139</td>
<td>5785</td>
<td>17.65%</td>
<td>kernel</td>
</tr>
<tr>
<td>0x7f4631745000</td>
<td>0.83%</td>
<td>96.80%</td>
<td>4KB</td>
<td>1</td>
<td>100.00%</td>
<td>0</td>
<td>10</td>
<td>15.63%</td>
<td>libc.so</td>
<td></td>
</tr>
<tr>
<td>0x7f4631f52000</td>
<td>0.70%</td>
<td>97.50%</td>
<td>4KB</td>
<td>1</td>
<td>100.00%</td>
<td>0</td>
<td>31</td>
<td>48.44%</td>
<td>libm.so</td>
<td></td>
</tr>
<tr>
<td>0x12a000000</td>
<td>10</td>
<td>0.61%</td>
<td>98.11%</td>
<td>2MB</td>
<td>16</td>
<td>3.13%</td>
<td>496</td>
<td>158</td>
<td>0.48%</td>
<td>application</td>
</tr>
<tr>
<td>0x80000000</td>
<td>11</td>
<td>0.46%</td>
<td>98.57%</td>
<td>2MB</td>
<td>323</td>
<td>63.09%</td>
<td>189</td>
<td>4273</td>
<td>13.04%</td>
<td>application</td>
</tr>
<tr>
<td>0x7f4631f4e000</td>
<td>0.22%</td>
<td>98.78%</td>
<td>4KB</td>
<td>1</td>
<td>100.00%</td>
<td>0</td>
<td>6</td>
<td>9.38%</td>
<td>libm.so</td>
<td></td>
</tr>
<tr>
<td>0x7f4631f73f000</td>
<td>0.15%</td>
<td>98.93%</td>
<td>4KB</td>
<td>1</td>
<td>100.00%</td>
<td>0</td>
<td>7</td>
<td>10.94%</td>
<td>libc.so</td>
<td></td>
</tr>
<tr>
<td>0x7f4631f9e000</td>
<td>0.10%</td>
<td>99.03%</td>
<td>4KB</td>
<td>1</td>
<td>100.00%</td>
<td>0</td>
<td>10</td>
<td>15.63%</td>
<td>libm.so</td>
<td></td>
</tr>
</tbody>
</table>
Code heat maps challenges

- Finding page sizes for a code address
  - Many different ways of getting huge pages for text (mremap, THP, tmpfs, ..)
  - Different impact on `/proc/PID/maps`
  - Difficulties for perf tool: must use heuristics

- `/proc/PID/smaps`
  - Shows if VMA is using large pages, but does not tell you the actual address range

- Need `perf_events` support
  - Need new sample format (`perf_event_attr.sample_format`)
  - `PERF_SAMPLE_CODE_PGSZ`: record code page size based on IP
  - `PERF_SAMPLE_DATA_PGSZ`: record data page size based on DLA
  - Must extract information from TLB or vm subsystem, tricky in NMI context
Branch optimizations

- Minimize number of taken branches: FDO or autoFDO
  - Make the code as straight as possible
- Minimize branch misprediction
  - Minimize number of conditional branches
- Minimize branch misprediction cost
  - Mitigate penalty of misprediction
- Topdown branch cost
  - Front-End: Frontend Latency -> Branch Resteers = estimate cycles wasted to fetch correct path
  - Bad Speculation: wasted slots for uops not retiring or lost due to recovery from wrong path
- Large web workload impacted
  - Some large Google apps have 25% wasted uops
PMU support for branches

- Conditional mispredictions always on direction (taken vs. not taken)
  - Target is always known as IP-relative offset

- Need misprediction rate **per branch**
  - \texttt{BR\_MISP\_ RETIRED\_COND} : does not tell taken vs. not taken

- Need rate of taken vs. non-taken **per branch**
  - Use \texttt{PEBS + BR\_INST\_RETIRED\_NOT\_TAKEN} but missing \texttt{BR\_INST\_RETIRED\_COND\_TAKEN}
  - Perf_events PEBS issue: return either branch source or target, need both

- New sample format: \texttt{PERF\_SAMPLE\_SKID\_IP} (posted on LKML)
  - With PEBS:
    - \texttt{PERF\_SAMPLE\_IP}: return source (precise\textgreater{}1) or target (precise = 1)
    - \texttt{PERF\_SAMPLE\_SKID\_IP}: return target
  - Without PEBS:
    - \texttt{PERF\_SAMPLE\_SKID\_IP} = \texttt{PERF\_SAMPLE\_IP}
PEBS and branches

- PEBS captures IP at retirement of sampled instruction
  - PEBS.ip: next dynamic instruction
  - PEBS.eventing_ip (Haswell and later): instruction causing event
- PEBS with perf_events
  - precise=1 => PEBS.ip
  - precise=2 or 3 => PEBS.eventing_ip

```
$ perf record -e \
  cpu/br_inst_ret.conditional/pp ...

%smpl
21.5% 400343: test %rax,%rax  ← macro-fusion
42.1% 400346: je 400358
  400348: mov $0x404058,%edi
  ...
400358: leaveq
```

```
$ perf record -e \
  cpu/br_inst_ret.conditional/p ...

%smpl
  400343: test %rax,%rax
  400346: je 400358
  400348: mov $0x404058,%edi
  ...
41.5% 400348: leaveq  ← taken

20.2% 400358: leaveq  ← not taken
```
## Branch mispredictions example

<table>
<thead>
<tr>
<th>Branch Address</th>
<th>Backward</th>
<th>Retired (taken, non-taken, predicted, mispredicted)</th>
<th>Mispredicted Retired</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Address</td>
<td>Samples</td>
<td>Ret%</td>
</tr>
<tr>
<td>0x0000000013951583</td>
<td>1</td>
<td>1799900</td>
<td>5.72%</td>
</tr>
<tr>
<td>0x000000001395115f</td>
<td>0</td>
<td>399392</td>
<td>1.27%</td>
</tr>
<tr>
<td>0x0000000013951317</td>
<td>0</td>
<td>315453</td>
<td>1.00%</td>
</tr>
<tr>
<td>0x0000000013951107</td>
<td>0</td>
<td>1095072</td>
<td>3.48%</td>
</tr>
<tr>
<td>0x0000000013951327</td>
<td>0</td>
<td>256250</td>
<td>0.81%</td>
</tr>
<tr>
<td>0x0000000013951292</td>
<td>0</td>
<td>204519</td>
<td>0.65%</td>
</tr>
<tr>
<td>0x00000000138621d1</td>
<td>1</td>
<td>221428</td>
<td>0.70%</td>
</tr>
<tr>
<td>0x000000001395116f</td>
<td>0</td>
<td>213412</td>
<td>0.68%</td>
</tr>
<tr>
<td>0x000000001395153f</td>
<td>0</td>
<td>301982</td>
<td>0.96%</td>
</tr>
</tbody>
</table>
References

- **Cache QoS**
  - Intel SDM Vol 3b, Chapter 17.19 Intel Resource Director Technology
  - CAT at Scale: Deploying Cache Isolation in a Mixed Workload Environment - Rohit Jnagal & David Lo, Google
  - Heracles: Improving Resource Efficiency at Scale