Linux Perf Tool Metrics

Ian Rogers (Google) Weilin Wang (Intel)
Getting started

Linux

Perf Tool

Metrics
Getting started

Linux

Perf Tool

Metrics
Getting started

Linux

Perf Tool

Metrics
Getting started

Linux

Perf Tool

Metrics
Why metrics?

Events are good but have interesting properties:

- What are the units of a counter? Bytes, cache lines, cycles, instructions, different clocks. Are speculative instructions counted?
- Perf will aggregate the same event across multiple PMUs (e.g. memory controllers) and events can be scaled.

Metrics allow for multiple different counters to be combined across different PMUs, incorporating things like time and outputting with human readable units.
Metric Groups

- Memory Controller
  - Event
- Last level cache
  - Event
- CPU
  - Event
- Interconnect
  - Event
How events are encoded

```bash
$ ls /sys/bus/event_source/devices/cpu/events
branch-instructions  cpu-cycles  slots
branch-misses       instructions topdown-bad-spec
bus-cycles          mem-loads   topdown-be-bound
cache-misses        mem-stores  topdown-fe-bound
cache-references    ref-cycles  topdown-retiring
```
$ perf list --details
...
Metric Groups:

Backend: [Grouping from Top-down Microarchitecture Analysis Metrics spreadsheet]
  tma_core_bound
    [This metric represents fraction of slots where Core non-memory issues were of a bottleneck]
    [max(0, tma_backend_bound - tma_memory_bound)]
    [tma_core_bound > 0.1 & tma_backend_bound > 0.2]
  tma_info_core_ilp
    [Instruction-Level-Parallelism (average number of uops executed when there is execution) per-core]
    [UOPS_EXECUTED.THREAD / (UOPS_EXECUTED.CORE_CYCLES_GE_1 / 2 if #SMT_on else UOPS_EXECUTED.CORE_CYCLES_GE_1)]
  tma_info_memory_l2mpki
    [L2 cache true misses per kilo instruction for retired demand loads]
    [1e3 * MEM_LOAD RETIRED.L2_MISS / INST RETIRED.ANY]
$ perf list --details
...
Metric Groups:

Backend: [Grouping from Top-down Microarchitecture Analysis spreadsheet]
  tma_core_bound
    [This metric represents fraction of slots where Core non-memory issues were of a bottleneck]
    [max(0, tma_backend_bound - tma_memory_bound)]
    [tma_core_bound > 0.1 & tma_backend_bound > 0.2]
  tma_info_core_ilp
    [Instruction-Level-Parallelism (average number of uops executed when there is execution) per-core]
    [UOPS_EXECUTED.THREAD / (UOPS_EXECUTED.CORE_CYCLES_GE_1 / 2 if #SMT_on else UOPS_EXECUTED.CORE_CYCLES_GE_1)]
  tma_info_memory_l2mpki
    [L2 cache true misses per kilo instruction for retired demand loads]
    [1e3 * MEM_LOAD RETIRED.L2_MISS / INST RETIRED.ANY]
...
$ perf list --details
...
Metric Groups:

Backend: [Grouping from Top-down Microarchitecture Analysis Metric spreadsheet]
   tma_core_bound
     [This metric represents fraction of slots where Core were of a bottleneck]
     [max(0, tma_backend_bound - tma_memory_bound)]
     [tma_core_bound > 0.1 & tma_backend_bound > 0.2]
   tma_info_core_ilp
     [Instruction-Level-Parallelism (average number of uops executed when there is execution) per-core]
     [UOPS_EXECUTED.THREAD / (UOPS_EXECUTED.CORE_CYCLES_GE_1 / 2 if #SMT_on else UOPS_EXECUTED.CORE_CYCLES_GE_1)]
   tma_info_memory_l2mpki
     [L2 cache true misses per kilo instruction for retired demand loads]
     [1e3 * MEM_LOAD RETIRED.L2_MISS / INST RETIRED.ANY]
...
The `tma_core_bound` metric

\[
\max(0, \text{tma_backend_bound} - \text{tma_memory_bound})
\]

\[
\frac{(\text{CYCLE_ACTIVITY.STALLS_MEM_ANY} + \text{EXE_ACTIVITY.BOUND_ON_STORES})}{(\text{CYCLE_ACTIVITY.STALLS_TOTAL} + (\text{EXE_ACTIVITY.1_PORTS_UTIL} + \text{tma_retiring} \times \text{EXE_ACTIVITY.2_PORTS_UTIL}) + \text{EXE_ACTIVITY.BOUND_ON_STORES}) \times \text{tma_backend_bound}}
\]

\[
\frac{\text{topdown-be-bound}}{(\text{topdown-fe-bound} + \text{topdown-bad-spec} + \text{topdown-retiring} + \text{topdown-be-bound}) + 5 \times \text{cpu@INT_MISC.RECOVERY_CYCLES,cmask=1,edge@} / \text{tma_info_thread_slots}}
\]

\[
\frac{\text{topdown\-retiring}}{(\text{topdown\-fe\-bound} + \text{topdown\-bad\-spec} + \text{topdown\-retiring} + \text{topdown\-be\-bound}) + 0 \times \text{tma_info_thread_slots}}
\]
Where do the events and metrics come from?

Per architecture event json

- create_perf_json.py
- pmu-events.c
- Perf json from other architectures

Server metrics

- Github hosted generator
- LKML
- Linux build

TMA metrics spreadsheet

- csv
- https://github.com/intel/perfmon
Top-down Microarchitecture Analysis (TMA)

TMA methodology

- Identifying performance bottlenecks in out-of-order cores
- No requiring deep knowledge of the microarchitecture details
- Available in Intel client and server platforms

TMA in Linux Perf Tool

- Use `perf stat -M` to drill down

General TMA Hierarchy for Out-of-Order Microarchitecture

From: Intel® 64 and IA-32 Architectures Optimization Reference Manual

1. Intel® 64 and IA-32 Architectures Optimization Reference Manual, Appendix B.1
Example: TMA Level Breakdown with Linux Perf Tool

```
perf stat -M TopdownL1
```

```
$ perf stat -M tma_backend_bound_group -a -c1 stress-ng --matrix 1 --taskset 1 -t 5s
```

**TMA Level 1**
- 44.7% Retiring
- 0% Bad Speculation
- 6.3% Frontend Bound
- 49.0% Backend Bound

**TMA Level 2 Backend Bound Group**
- 43.4% Core Bound
- 3.9% Memory Bound

**General TMA Hierarchy**
From: Intel® 64 and IA-32 Architectures Optimization Reference Manual
Example: TMA Level Breakdown with Linux Perf Tool

```
perf stat -M tma_core_bound_group
```

```
perf stat -M tma_ports_utilization_group
```

General TMA Hierarchy

From: Intel® 64 and IA-32 Architectures Optimization Reference Manual
$ perf stat true

Performance counter stats for 'true':

1.08 msec task-clock # 0.089 CPUs utilized
  1 context-switches # 926.027 /sec
  0 cpu-migrations # 0.000 /sec
  52 page-faults # 48.153 K/sec
1,245,404 cycles # 1.153 GHz
1,339,902 instructions # 1.08 insn per cycle
  269,832 branches # 249.872 M/sec
  7,143 branch-misses # 2.65% of all branches

TopdownL1
  # 24.6 % tma_backend_bound
  # 9.6 % tma_bad_speculation
  # 41.9 % tma_frontend_bound
  # 23.9 % tma_retiring

0.012078534 seconds time elapsed

0.000000000 seconds user
0.003140000 seconds sys
Optionality of metric thresholds

Metric thresholds are themselves metrics. This means more events may be present when a threshold is computed which may cause event multiplexing.

To avoid multiplexing metric thresholds are computed:
- whenever all events are present,
- when a metric is explicitly requested except when –metric-no-threshold is passed.
Counters, metrics and their thresholds indicate performance issues but samples show where in your code things are happening. Use “Sample with” from `perf list` to get the event to use with `perf record`.

```
$ perf list -v
...
 tma_ports_utilized_1
 [This metric represents fraction of cycles where the CPU executed total of 1 uop per cycle on all execution ports (Logical Processor cycles since ICL, Physical Core cycles otherwise). This can be due to heavy data-dependency among software instructions; or over oversubscribing a particular hardware resource. In some other cases with high 1_Port_Utilized and L1_Bound; this metric can point to L1 data-cache latency bottleneck that may not necessarily manifest with complete execution starvation (due to the short L1 latency e.g. walking a linked list) - looking at the assembly can be helpful. Sample with: EXE_ACTIVITY.1_PORTS_UTIL. Related metrics: tma_l1_bound]
...
Counters, metrics and their thresholds indicate performance issues but samples show where in your code things are happening. Use “Sample with” from `perf list` to get the event to use with `perf record`.

```bash
$ perf list -v
...
tma_ports_utilized_1
   [This metric represents fraction of cycles where the CPU executed total of 1 uop per cycle on all execution ports (Logical Processor cycles since ICL, Physical Core cycles otherwise). This can be due to heavy data-dependency among software instructions; or over oversubscribing a particular hardware resource. In some other cases with high 1_Port_Utilized and L1_Bound; this metric can point to L1 data-cache latency bottleneck that may not necessarily manifest with complete execution starvation (due to the short L1 latency e.g. walking a linked list) - looking at the assembly can be helpful. Sample with: EXE_ACTIVITY.1_PORTS_UTIL. Related metrics: tma_l1_bound]
...
$ perf record -e EXE_ACTIVITY.1_PORTS_UTIL ...
```
Key part of TMA metrics is a measure of slots, number of functional units multiplied by cycles, pre-Icelake there was no counter for this. Hyperthreading complicated the slots calculation and counters were added measuring when 1 or both hyperthreads were active. EBS mode scaled metrics pre-Icelake accordingly, but was buggy unless in system-wide mode (ie. when no scaling was necessary). Because of the bugginess, the metrics are not enabled by default on pre-Icelake. TopdownL1 and other metrics are available pre-Icelake but some caution should be observed when measuring benchmarks as EBS mode will be implicitly used.
$ perf stat -a sleep 1

Performance counter stats for 'system wide':

24,081.38 msec cpu-clock # 23.984 CPUs utilized
391 context-switches # 16.237 /sec
25 cpu-migrations # 1.038 /sec
68 page-faults # 2.824 /sec
129,900,175 cpu_atom/cycles/ # 0.005 GHz (54.18%)
16,045,550 cpu_core/cycles/ # 0.001 GHz
19,513,883 cpu_atom/instructions/ # 0.15 insn per cycle (63.34%)
8,909,751 cpu_core/instructions/ # 0.07 insn per cycle
3,904,849 cpu_atom/branches/ # 162.152 K/sec (63.33%)
1,870,930 cpu_core/branches/ # 77.692 K/sec
662,455 cpu_atom/branch-misses/ # 16.96% of all branches (63.34%)
98,623 cpu_core/branch-misses/ # 2.53% of all branches

TopdownL1 (cpu_core)

# 30.3 % tma_backend_bound
# 8.4 % tma_bad_speculation
# 49.6 % tma_frontend_bound
# 11.7 % tma_retiring

TopdownL1 (cpu_atom)

# 20.8 % tma_bad_speculation (63.35%)
# 37.7 % tma_frontend_bound (63.71%)
# 35.4 % tma_backend_bound
# 35.4 % tma_backend_bound_aux (64.11%)
# 5.5 % tma_retiring (64.15%)

1.004077587 seconds time elapsed
$ perf stat -a sleep 1

Performance counter stats for 'system wide':

<table>
<thead>
<tr>
<th>Counter</th>
<th>Value</th>
<th>% of utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>24,081.38 msec cpu-clock</td>
<td>23.984 CPUs utilized</td>
<td></td>
</tr>
<tr>
<td>391 context-switches</td>
<td>16.237 /sec</td>
<td></td>
</tr>
<tr>
<td>25 cpu-migrations</td>
<td>1.038 /sec</td>
<td></td>
</tr>
<tr>
<td>68 page-faults</td>
<td>2.824 /sec</td>
<td></td>
</tr>
<tr>
<td>129,900,175 cpu_atom/cycles/</td>
<td>0.005 GHz</td>
<td></td>
</tr>
<tr>
<td>16,045,550 cpu_core/cycles/</td>
<td>0.001 GHz</td>
<td></td>
</tr>
<tr>
<td>19,513,883 cpu_atom/instructions/</td>
<td>0.15 insn per cycle</td>
<td></td>
</tr>
<tr>
<td>8,909,751 cpu_core/instructions/</td>
<td>0.07 insn per cycle</td>
<td></td>
</tr>
<tr>
<td>3,904,849 cpu_atom/branches/</td>
<td>1.152 K/sec</td>
<td></td>
</tr>
<tr>
<td>1,870,930 cpu_core/branches/</td>
<td>0.692 K/sec</td>
<td></td>
</tr>
<tr>
<td>662,455 cpu_atom/branch-misses/</td>
<td>6.96% of all branches</td>
<td></td>
</tr>
<tr>
<td>98,623 cpu_core/branch-misses/</td>
<td>5.53% of all branches</td>
<td></td>
</tr>
</tbody>
</table>

TopdownL1 (cpu_core)

- 8.4% tma_bad_speculation
- 49.6% tma_frontend_bound
- 11.7% tma_retiring

TopdownL1 (cpu_atom)

- 20.8% tma_bad_speculation
- 37.7% tma_frontend_bound
- 35.4% tma_backend_bound
- 35.4% tma_backend_bound_aux
- 5.5% tma_retiring

1.004077587 seconds time elapsed
$ perf stat -a sleep 1

Performance counter stats for 'system wide':

24,081.38 msec cpu-clock                        #   23.984 CPUs utilized
391      context-switches                 #   16.237 /sec
25      cpu-migrations                   #    1.038 /sec
68      page-faults                      #    2.824 /sec
129,900,175      cpu_atom/cycles/                 #    0.005 GHz                         (54.18%)
16,045,550      cpu_core/cycles/                 #    0.001 GHz                         (54.18%)
19,513,883      cpu_atom/instructions/           #    0.15  insn per cycle              (63.34%)
8,909,751      cpu_core/instructions/           #    0.07  insn per cycle              (63.34%)
3,904,849      cpu_atom/branches/               #  162.152 K/sec                       (63.34%)
1,870,930      cpu_core/branches/               #   77.692 K/sec                       (63.34%)
662,455      cpu_atom/branch-misses/            #  16.96% of all branches             (63.34%)
98,623      cpu_core/branch-misses/             #  2.53% of all branches              (63.34%)

TopdownL1 (cpu_core)                 
#  30.3 % tma_backend_bound
#   8.4 % tma_bad_speculation
#  49.6 % tma_frontend_bound
#  11.7 % tma_retiring

TopdownL1 (cpu_atom)
#  20.8 % tma_bad_speculation
#  37.7 % tma_frontend_bound
#  35.4 % tma_backend_bound
#  35.4 % tma_backend_bound_aux
#   5.5 % tma_retiring

1.004077587 seconds time elapsed

Support for hybrid processors

Multiplexing on Atom due to insufficient counters for both topdown and branch events
$ perf test -v validation

107: perf metrics value validation:
--- start ---
...
Workload: perf bench futex hash -r 2 -s
Total metrics collected: 200
Non-negative metric count: 200
Total Test Count: 100
Passed Test Count: 100
Test validation finished. Final report:
[
  {
    "Workload": "perf bench futex hash -r 2 -s",
    "Report": {
      "Metric Validation Statistics": {
        "Total Rule Count": 100,
        "Passed Rule Count": 100
      },
      "Tests in Category": {
        "PositiveValueTest": {
          "Total Tests": 200,
          "Passed Tests": 200,
          "Failed Tests": []
        },
        "RelationshipTest": {
          "Total Tests": 5,
          "Passed Tests": 5,
          "Failed Tests": []
        },
        "SingleMetricTest": {
          "Total Tests": 95,
          "Passed Tests": 95,
          "Failed Tests": []
        }
      },
      "Errors": []
    }
  }
]
test child finished with 0
---- end ----
perf metrics value validation: Ok
Ongoing technical challenges
Event grouping and hardware counters

**Invalid Grouping:**

- Metric1: Event1, Event2, Event3, Event4
- Metric2: Event3, Event4, Event5
- Metric3: Event1, Event5

**Counters:**

- C1
- C2
- C3

- Event1
- Counter: 1,2,3

- Event2
- Counter: 1,2,3

- Event3

**Functional but Inefficient Grouping:**

- Metric1: Event1, Event2, Event3, Event4
- Metric2: Event3, Event4, Event5
- Metric3: Event1, Event5

**Counters:**

- C1
- C2
- C3

- Event1
- Counter: 1,2,3

- Event2
- Counter: 1,2,3

- Event3

**Functional and Better Grouping:**

- Metric1: Event1, Event2, Event3, Event4
- Metric2: Event3, Event4, Event5
- Metric3: Event1, Event5

**Counters:**

- C1
- C2
- C3

- Event1
- Counter: 1,2,3

- Event2
- Counter: 1,2,3

- Event3

- Event1
- Counter: 1,2,3

- Event5
Event grouping and hardware counters

Metric1: Event1, Event2, Event3, Event4
Metric2: Event3, Event4, Event5
Metric3: Event1, Event5

Counters:
- **C1**
- **C2**
- **C3**

Event Grouping:

**Invalid Grouping:**

- Group 1: Event1, Event2, Event3, Event4
- Group 2: Event3, Event4, Event5
- Group 3: Event1, Event5

**Functional but Inefficient Grouping:**

- Group 1: Event1, Event2, Event3
- Group 2: Event4
- Group 3: Event3, Event4, Event5
- Group 4: Event1, Event5

**Functional and Better Grouping:**

- Group 1: Event1, Event2, Event3
- Group 2: Event3, Event4, Event5
Event grouping and hardware counters

Invalid Grouping:

- Group 1: Event1, Event2, Event3, Event4
- Group 2: Event3, Event4, Event5
- Group 3: Event1, Event5

Functional but Inefficient Grouping:

- Group 1: Event1, Event2, Event3
- Group 2: Event4
- Group 3: Event3, Event4, Event5
- Group 4: Event1, Event5

Functional and Better Grouping:

- Group 1: Event1, Event2, Event3
- Group 2: Event3, Event4, Event5

Counters:

- C1
- C2
- C3

Event1: Counter: 1,2,3
Event2: Counter: 1,2,3
Event3: Counter: 1,2,3
Event4: Counter: 1,2,3
Event5: Counter: 1,2,3
**Event grouping and hardware counters**

### Metric 1
- Event1, Event2, Event3, Event4

### Metric 2
- Event3, Event4, Event5

### Metric 3
- Event1, Event5

#### Counters
- **C1**
  - Event1
  - Counter: 1,2,3
  - Event2
  - Counter: 1,2,3
  - ... 
  - Event5
  - Counter: 1,2,3

#### Invalid Grouping:
- **Group 1**
  - Event1, Event2, Event3, Event4

- **Group 2**
  - Event3, Event4, Event5

- **Group 3**
  - Event1, Event5

#### Functional but Inefficient Grouping:
- **Group 1**
  - Event1, Event2, Event3

- **Group 2**
  - Event4

- **Group 3**
  - Event3, Event4, Event5

- **Group 4**
  - Event1, Event5

#### Functional and Better Grouping:
- **Group 1**
  - Event1, Event2, Event3

- **Group 2**
  - Event3, Event4, Event5
Event grouping and hardware counters

Invalid Grouping:

Metric1: Event1, Event2, Event3, Event4
Metric2: Event3, Event4, Event5
Metric3: Event1, Event5

Counters:

<table>
<thead>
<tr>
<th>C1</th>
<th>C2</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event1</td>
<td>Counter: 1,2,3</td>
<td></td>
</tr>
<tr>
<td>Event2</td>
<td>Counter: 1,2,3</td>
<td></td>
</tr>
<tr>
<td>Event5</td>
<td>Counter: 1,2,3</td>
<td></td>
</tr>
</tbody>
</table>

Functional but Inefficient Grouping:

Group 1: Event1, Event2, Event3
Group 2: Event4
Group 3: Event3, Event4, Event5
Group 4: Event1, Event5

Functional and Better Grouping:

Group 1: Event1, Event2, Event3
Group 2: Event3, Event4, Event5
Event grouping and hardware counters

Metric1: Event1, Event2, Event3, Event4
Metric2: Event3, Event4, Event5
Metric3: Event1, Event5

Counts:
- C1
- C2
- C3

Invalid Grouping:

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Event1</th>
<th>Event2</th>
<th>Event3</th>
<th>Event4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 2</td>
<td>Event3</td>
<td>Event4</td>
<td>Event5</td>
<td></td>
</tr>
<tr>
<td>Group 3</td>
<td>Event1</td>
<td>Event5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Functional but Inefficient Grouping:

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Event1</th>
<th>Event2</th>
<th>Event3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 2</td>
<td>Event4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group 3</td>
<td>Event3</td>
<td>Event4</td>
<td>Event5</td>
</tr>
<tr>
<td>Group 4</td>
<td>Event1</td>
<td>Event5</td>
<td></td>
</tr>
</tbody>
</table>

Functional and Better Grouping:

<table>
<thead>
<tr>
<th>Group 1</th>
<th>Event1</th>
<th>Event2</th>
<th>Event3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 2</td>
<td>Event3</td>
<td>Event4</td>
<td>Event5</td>
</tr>
</tbody>
</table>
Hardware Aware Metric group Event Grouping

The key of FUNCTIONAL grouping is placing events to counters that support the events and avoid oversubscribed group.

Information required to be hardware counter aware:

- Describe all counter restrictions from events in JSON files
- Static counter availability of one platform could be described in JSON files
- Dynamic counter availability needs to be resolved

1. Standardized metrics and events defined in JSON files – Project Valkyrie: GitHub – intel/perfmon
2. Intel PMUs Event Reference: https://perfmon-events.intel.com/
Hardware Aware Metric group Event Grouping Details

1. "Perf stat metric grouping with hardware information" RFC Patch: https://lore.kernel.org/all/20230925061824.3818631-1-weilin.wang@intel.com

**Load Data From PMU-EVENTS**
- Build hardware counter information: PMU and counter availabilities
- Receive the event list of requested metrics
- Read counter restrictions of each event

**Generate Groups**
- For each event, find a group for the correct PMU that has space
- Fill it into the group based on counter restrictions
- Create a new group if no space available in all the existing groups

**Output Result**
- Generate metric group grouping string

**Event Counter Restrictions for Reference:**
1. Unit – The unit/core where the event is collected on.
2. Counter – The counters in the unit the event could be collected on and availability of the counters.
3. TakenAlone – TAKEN_ALONE event cannot be collected in the same group with any other TAKEN_ALONE events.
4. Filter1 – Events collected in the same group need to have same filter1 value if applicable (SKX/CLX/CPX).
5. Fixed Counter – Do not group events use the same fixed counter in the same group.
6. OCR events – At most two OCR events in one group.
The key of **GOOD** grouping is high counter utilization and good locality of events for metrics

- High counter utilization $\Rightarrow$ Less number of total groups $\Rightarrow$ More time for each group - Improve the overall event and metric accuracy
- Good locality of events $\Rightarrow$ Events that required by one metric in the same or neighboring groups - Improve metric accuracy
- However, these are conflicting conditions in some cases
Timed Processor Event Based Sampling (Timed PEBS)

- It records the number of unhalted core cycles between the retirement of the current instruction and the retirement of the prior instruction
- It significantly increases the accuracy of TMA
- IA32_PERF_CAPABILITIES.PEBS_TIMING_INFO[bit 17]
- Feature available in next generation Intel processors

Timed PEBS in perf tool

- Sampling mode - upstreamed
  - Retire_lat is enabled as a weight of PMU events in perf record
  - `perf record -W -e event_name:P`
- Counting mode - WIP
  - Retire_latency is included in some of the metrics in TMA for processors that support Timed PEBS

### PEBS Basic Info Group

<table>
<thead>
<tr>
<th>Offset</th>
<th>Field Name</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>Record Format</td>
<td>[31:0]</td>
</tr>
<tr>
<td></td>
<td>Retire Latency</td>
<td>[47:32]</td>
</tr>
<tr>
<td></td>
<td>Record Size</td>
<td>[63:48]</td>
</tr>
<tr>
<td>0x8</td>
<td>Instruction Pointer</td>
<td>[63:0]</td>
</tr>
<tr>
<td>0x10</td>
<td>Applicable Counters</td>
<td>[63:0]</td>
</tr>
<tr>
<td>0x18</td>
<td>TSC</td>
<td>[63:0]</td>
</tr>
</tbody>
</table>

From: Intel® Architecture Instruction Set Extensions and Future Features
Enabling counting mode for Timed PEBS

- “Retire Latency” field in the PEBS record requires sampling
- Counting mode solution requires both perf record and perf stat
- Proposed method is to fork perf record within perf stat
- Perf stat process sampling data and capture the retire latency value, calculate and print out the final metric counts
Discussion

• Sampled timings plus counters gives greatest accuracy for metrics but at the cost of using more counters.

• Current hard-coded values are for the worst case.

• Potential to use a variety of hard-coded values based on:
  • Averages: mean, median
  • Timings of similar benchmarks
  • Periodic sampling of the system
  • BPF vs perf record
Questions
Future Work

- Perf toptown
  - Automate the drill down
- Perf record with the “Sample with”
- Support for non-CPU metrics
- ML in metrics, for example, I don’t have instructions but I have branches. As there is usually a fixed ratio of branches to instructions can I swap a counter I don’t have for one I do.
Extra slides