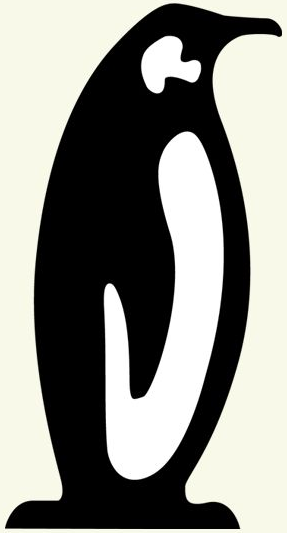


Linux Plumbers Conference

Richmond, Virginia | November 13-15, 2023

Linux
Plumbers
Conference | Richmond, VA | Nov. 13-15, 2023

Linux Perf Tool Metrics

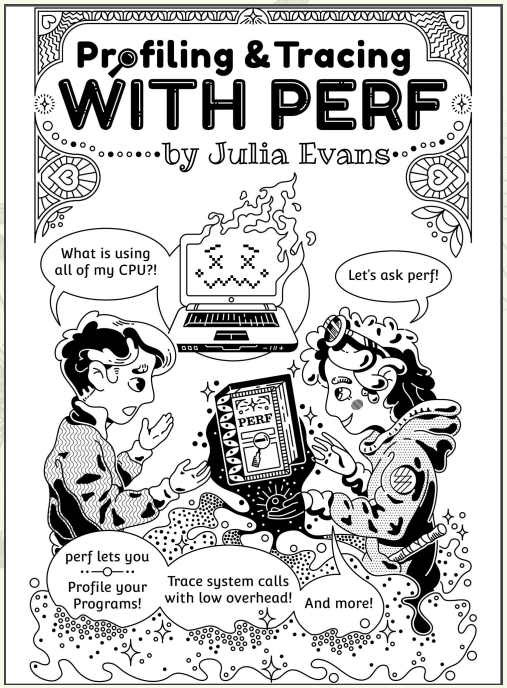
Ian Rogers (Google) Weilin Wang (Intel)

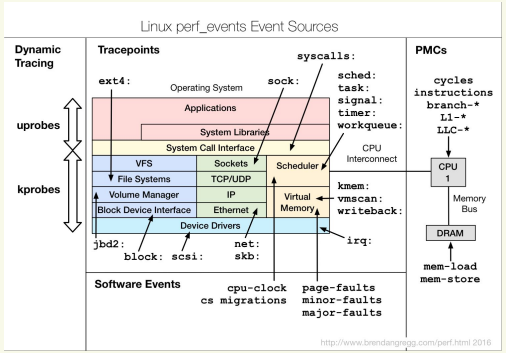
Getting started

Linux

Perf Tool

Metrics



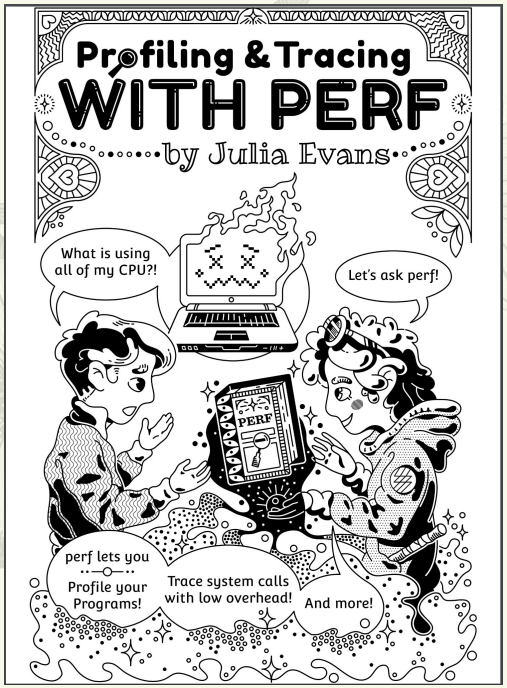


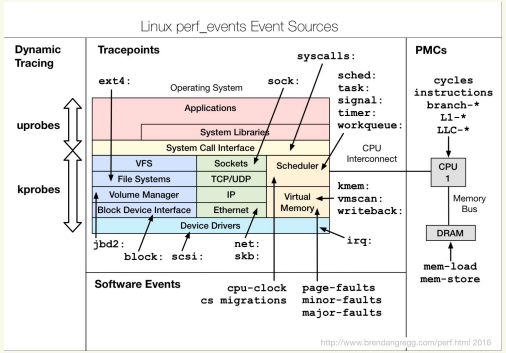
Getting started

Linux

Perf Tool

Metrics



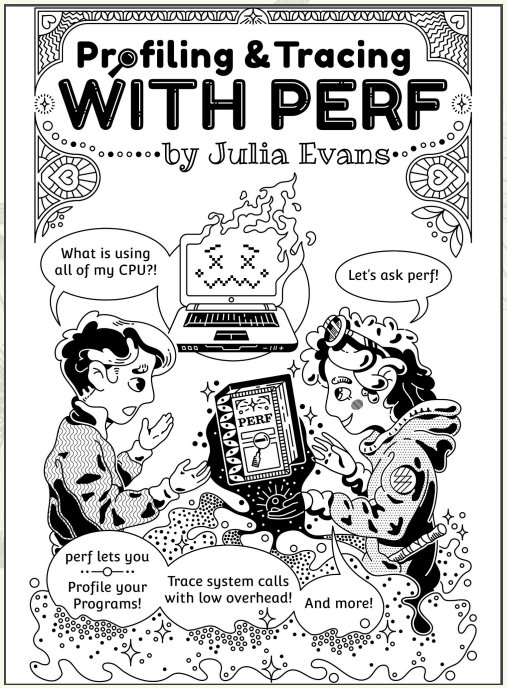


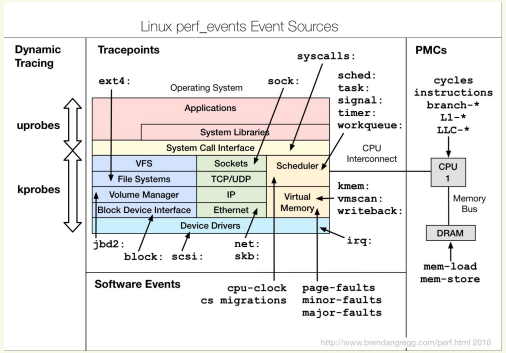
Getting started

Linux

Perf Tool

Metrics





Getting started

Linux

Perf Tool

Metrics

?

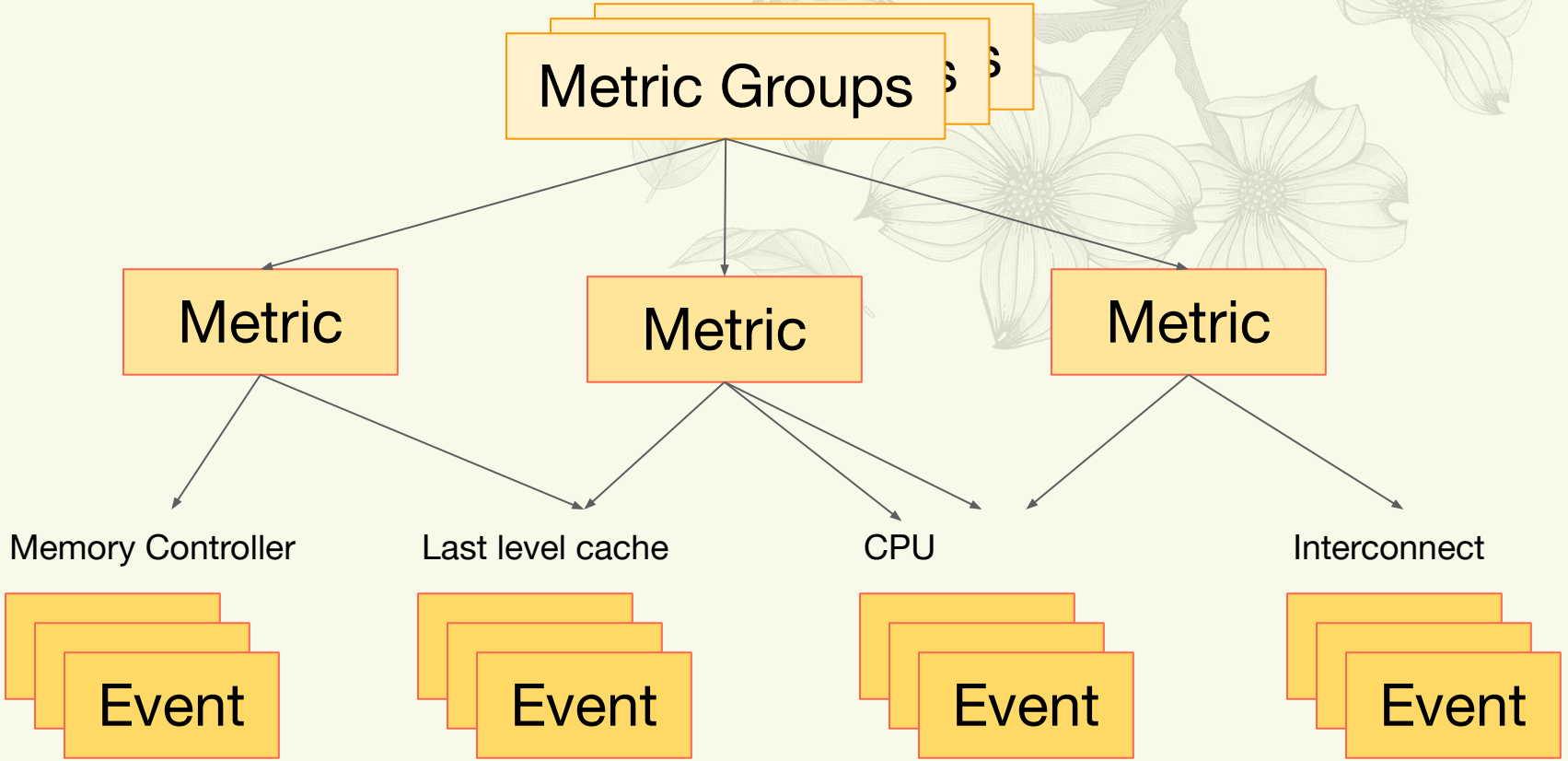
Why metrics?

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.

Collections of metrics



How events are encoded

Event

```
$ 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
```

linux / tools / perf / pmu-events / arch / x86 / meteorlake / **memory.json**

Code | Blame | 356 lines (356 loc) · 16.3 KB | Code 55% faster with GitHub Copilot

```
131         },
132         {
133             "BriefDescription": "Counts the number of memory ordering machine clears due to memory renaming.",
134             "EventCode": "0x09",
135             "EventName": "MEMORY_ORDERING.MRN_NUKE",
136             "SampleAfterValue": "100003",
137             "UMask": "0x2",
138             "Unit": "cpu_core"
139         },
140         {
```

Seeing metric expressions

```
$ 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]
…
```

Linux Plumbers

Seeing metric expressions

```
$ perf list --details
…
Metric Groups:

Backend: [Grouping from Top-down Microarchitecture Analysi
spreadsheet]
  tma_core_bound
       [This metric represents fraction of slots where Core n    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]
…
```

Metric expression

Linux Plumbers

Seeing metric expressions

```
$ perf list --details
…
Metric Groups:

Backend: [Grouping from Top-down Microarchitecture Analysis
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]
…
```

Threshold expression

The tma_core_bound metric

```
max(0, tma_backend_bound - tma_memory_bound)
```

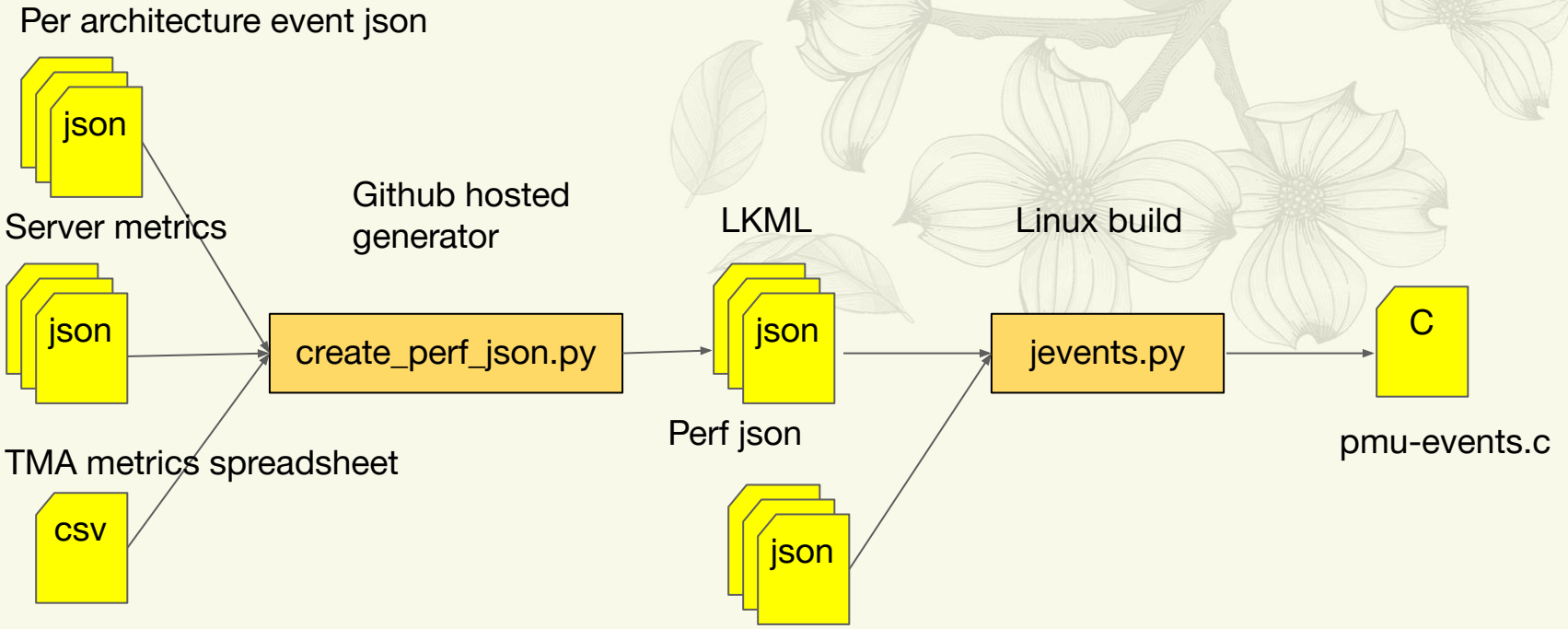
```
(CYCLE_ACTIVITY.STALLS_MEM_ANY + EXE_ACTIVITY.BOUND_ON_STORES) /
  (CYCLE_ACTIVITY.STALLS_TOTAL + (EXE_ACTIVITY.1_PORTS_UTIL +
       tma_retiring * EXE_ACTIVITY.2_PORTS_UTIL) +
    EXE_ACTIVITY.BOUND_ON_STORES) * tma_backend_bound
```

```
topdown-be-bound / (topdown-fe-bound + topdown-bad-spec +
      topdown-retiring + topdown-be-bound) + 5 *
   cpu@INT_MISC.RECOVERY_CYCLES,cmask=1,edge@ /
                tma_info_thread_slots
```

```
TOPDOWN.SLOTS
```

```
topdown\-retiring / (topdown\-fe\-bound + topdown\-bad\-spec +
topdown\-retiring + topdown\-be\-bound) + 0 * tma_info_thread_slots
```

Where do the events and metrics come from?



https://github.com/intel/perfmon

Perf json from other architectures

Top-down Microarchitecture Analysis (TMA)

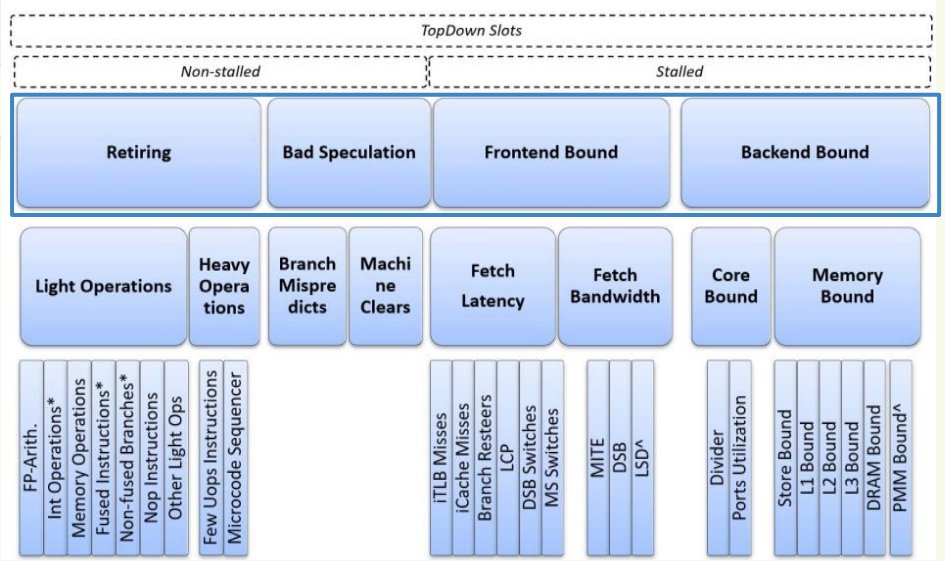
TMA methodology

- 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

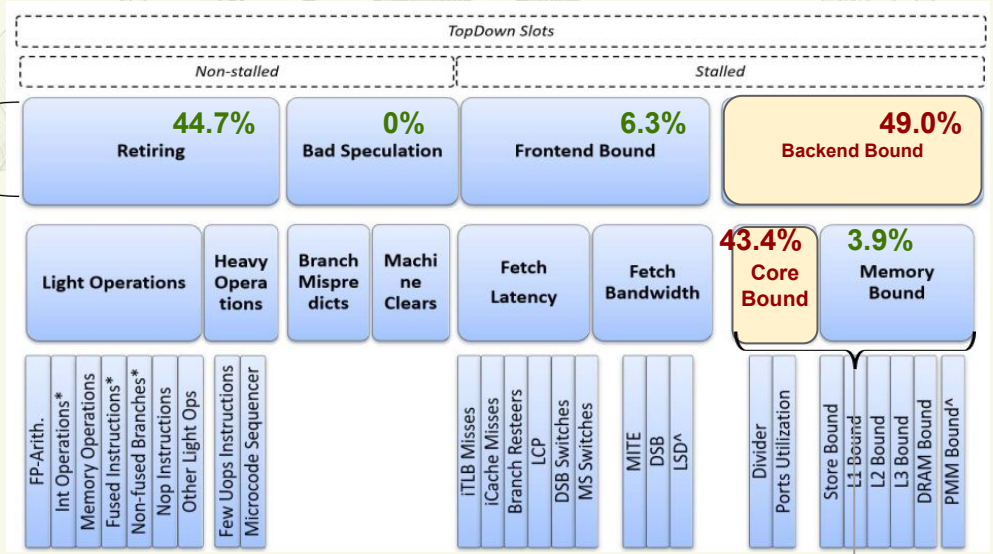
1. *Intel® 64 and IA-32 Architectures Optimization Reference Manual, Appendix B.1*
2. *A. Yasin, A Top-Down method for performance analysis and counters architecture, ISPASS 2014*

Example: TMA Level Breakdown with Linux Perf Tool

perf stat -M ***TopdownL1***

```
$ perf stat -M TopdownL1 -a -C1 stress-ng --matrix 1 --taskset 1 -t 5s
stress-ng: info:  [466696] setting to a 5 second run per stressor
stress-ng: info:  [466696] dispatching hogs: 1 matrix
stress-ng: info:  [466696] successful run completed in 5.00s

 Performance counter stats for 'system wide':

    91,497,997,500      TOPDOWN.SLOTS                 #     49.0 %  tma_backend_bound
                                                      #      0.0 %  tma_bad_speculation
                                                      #      6.3 %  tma_frontend_bound
                                                      #     44.7 %  tma_retiring
    40,904,987,117      topdown-retiring
     5,741,050,823      topdown-fe-bound
    44,851,959,558      topdown-be-bound
     5,741,050,823      topdown-heavy-ops
             3,259      topdown-bad-spec
         1,689,523      INT_MISC.UOP_DROPPING

       5.090903464 seconds time elapsed
```

TMA Level 1



perf stat -M **tma_backend_bound_group**

```
$ perf stat -M tma_backend_bound_group -a -C1 stress-ng --matrix 1 --taskset 1 -t 5s
stress-ng: info:  [501467] setting to a 5 second run per stressor
stress-ng: info:  [501467] dispatching hogs: 1 matrix
stress-ng: info:  [501467] successful run completed in 5.00s

 Performance counter stats for 'system wide':

    91,512,912,720      TOPDOWN.SLOTS                 #     43.4 %  tma_core_bound
                                                      #      3.9 %  tma_memory_bound
    42,347,151,768      topdown-retiring
     6,100,860,848      topdown-fe-bound
     3,588,741,675      topdown-mem-bound
    43,423,774,271      topdown-be-bound
                 0      topdown-bad-spec

       5.091833553 seconds time elapsed
```

TMA Level 2 Backend Bound Group

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_core_bound_group -a -C1 stress-ng --matrix 1 --taskset 1 -t 5s
stress-ng: info:  [841656] setting to a 5 second run per stressor
stress-ng: info:  [841656] dispatching hogs: 1 matrix
stress-ng: info:  [841656] successful run completed in 5.00s

 Performance counter stats for 'system wide':

    91,530,013,092      TOPDOWN.SLOTS                                 #      0.1 %  tma_divider
                                                                      #     59.4 %  tma_ports_utilization
    42,355,064,881      topdown-retiring
     5,743,059,644      topdown-fe-bound
     3,589,412,278      topdown-mem-bound
    43,431,888,565      topdown-be-bound
                 0      topdown-bad-spec
        91,601,567      RESOURCE_STALLS.SCOREBOARD
     2,518,236,344      cpu/EXE_ACTIVITY.3_PORTS_UTIL,umask=0x80/
       937,712,789      EXE_ACTIVITY.BOUND_ON_LOADS
        18,868,356      ARITH.DIV_ACTIVE
     3,428,017,600      EXE_ACTIVITY.1_PORTS_UTIL
    15,255,053,374      CPU_CLK_UNHALTED.THREAD
     6,727,671,296      cpu/EXE_ACTIVITY.2_PORTS_UTIL,umask=0xc/
       984,810,088      CYCLE_ACTIVITY.STALLS_TOTAL

       5.093215942 seconds time elapsed
```

TMA Level 3 Core Bound Group

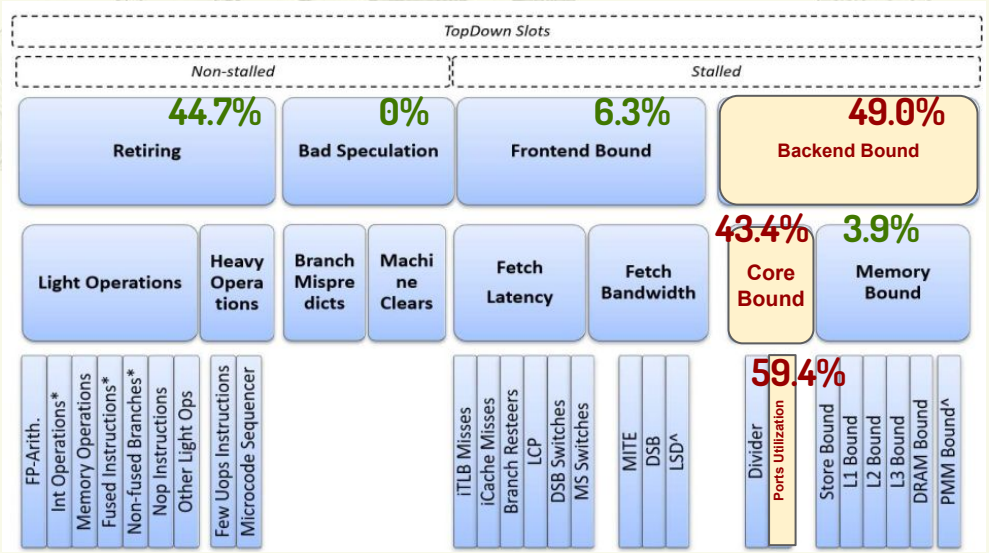
*perf stat -M **tma_ports_utilization_group***

```
$ perf stat -M tma_ports_utilization_group -a -C1 stress-ng --matrix 1 --taskset 1 -t 5s
stress-ng: info:  [854390] setting to a 5 second run per stressor
stress-ng: info:  [854390] dispatching hogs: 1 matrix
stress-ng: info:  [854390] successful run completed in 5.00s

 Performance counter stats for 'system wide':

    91,271,408,118      TOPDOWN.SLOTS                                 #     16.5 %  tma_ports_utilized_0
                                                                      #     23.2 %  tma_ports_utilized_1
                                                                      #     41.9 %  tma_ports_utilized_3m     (33.31%)
    42,593,323,786      topdown-retiring                                                                      (33.31%)
                                                                      ...
     6,671,449,801      cpu/EXE_ACTIVITY.2_PORTS_UTIL,umask=0xc/                                              (33.31%)
     1,127,395,705      CYCLE_ACTIVITY.STALLS_TOTAL                                                           (33.31%)
    91,369,045,193      TOPDOWN.SLOTS                                 #     27.5 %  tma_ports_utilized_2     (33.32%)
    42,997,197,738      topdown-retiring                                                                      (33.32%)
```

23.2% tma_ports_utilized_1

27.5% tma_ports_utilized_2

TMA Level 4 Ports Utilization Group



General TMA Hierarchy

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

Topdown is now present in perf stat default output (for Icelake and newer models)

```
$ 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.

Going from counts to samples

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]
…
```

Going from counts to samples

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]
…
$ perf record -e  EXE_ACTIVITY.1_PORTS_UTIL …
```

#EBS_Mode

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.

Support for hybrid processors

```
$ 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
```

Support for hybrid processors

```
$ 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/instru          #    .07  insn per cycle
         3,904,849      cpu_atom/branch          #    .152 K/sec                     (63.33%)
         1,870,930      cpu_core/branch          #    .692 K/sec
           662,455      cpu_atom/branch          #    .96% of all branches           (63.34%)
            98,623      cpu_core/branch          #    .53% of all branches
             TopdownL1 (cpu_core)                #      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
```

Per core type breakdown

Support for hybrid processors

```
$ perf stat -a sleep 1

 Performance counter stats for 'system wide':

         24,081.38 msec cpu-clock                 #   23.984 CPUs utilized
                391      context-swi              #   16.237 /sec
                 25      cpu-migrati              #    1.038 /sec
                 68      page-faults              #    2.824 /sec
        129,900,175      cpu_atom/cy              #    0.005 GHz                         (54.18%)
         16,045,550      cpu_core/cy
         19,513,883      cpu_atom/in                                                     (63.34%)
          8,909,751      cpu_core/in              #    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
```

Multiplexing on Atom due to insufficient counters for both topdown and branch events

Validation tests

```
$ 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

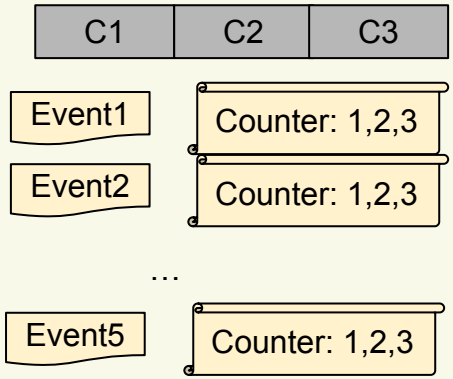
Event grouping and hardware counters

Metric1: Event1, Event2, Event3, Event4

Metric2: Event3, Event4, Event5

Metric3: Event1, Event5

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

Event grouping and hardware counters

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

…

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:

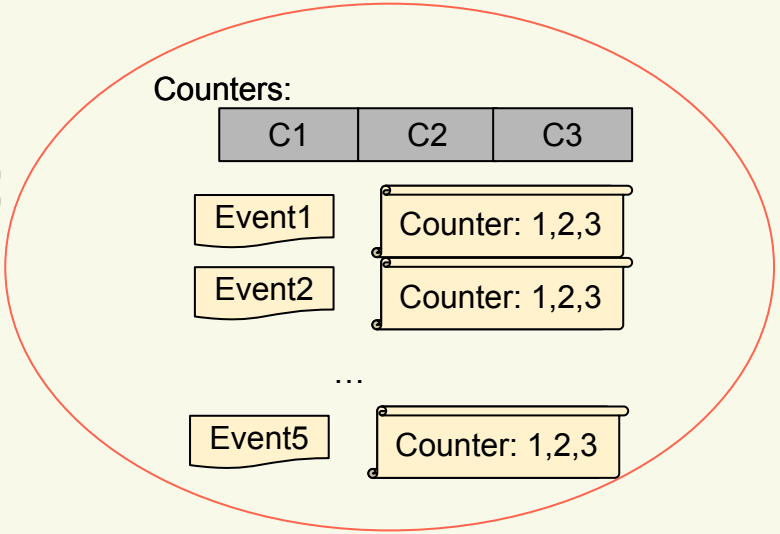
Group 1	Event1	Event2	Event3
Group 2	Event3	Event4	Event5

Event grouping and hardware counters

Metric1: Event1, Event2, Event3, Event4

Metric2: Event3, Event4, Event5

Metric3: Event1, Event5



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

Metric1: Event1, Event2, Event3, Event4

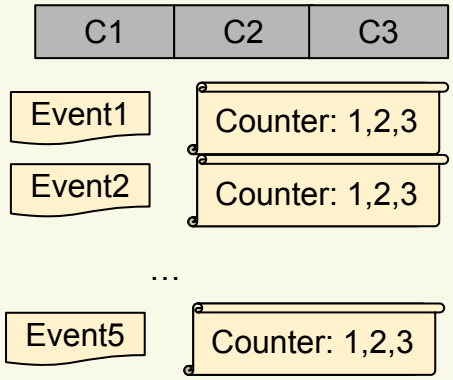
Metric2: Event3, Event4, Event5

Metric3: Event1, Event5

Invalid Grouping:

Group 1	Event1	Event2	Event3	Event4
Group 2	Event3	Event4	Event5	
Group 3	Event1	Event5		

Counters:



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

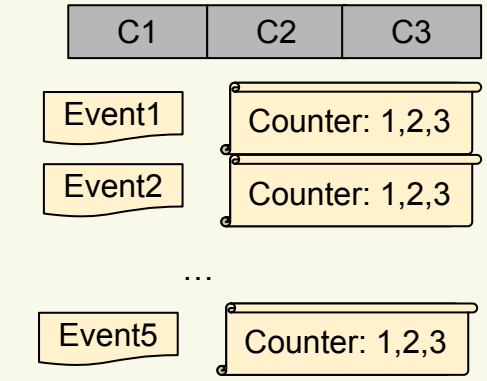
Event grouping and hardware counters

Metric1: Event1, Event2, Event3, Event4

Metric2: Event3, Event4, Event5

Metric3: Event1, Event5

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

Event grouping and hardware counters

Metric1: Event1, Event2, Event3, Event4

Metric2: Event3, Event4, Event5

Metric3: Event1, Event5

Invalid Grouping:

Group 1	Event1	Event2	Event3	Event4
Group 2	Event3	Event4	Event5	
Group 3	Event1	Event5		

Counters:

C1	C2	C3

Event1 — Counter: 1,2,3

Event2 — Counter: 1,2,3

…

Event5 — Counter: 1,2,3

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

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

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 base 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.

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

Discussion

The key of **GOOD** grouping is high counter utilization and good locality of events for metrics

- High counter utilization => Less number of total groups => More time for each group - Improve the overall event and metric accuracy
- Good locality of events => Events that required by one metric in the same or neighboring groups - Improve metric accuracy
- However, these are conflicting conditions in some cases

What is Timed PEBS?

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

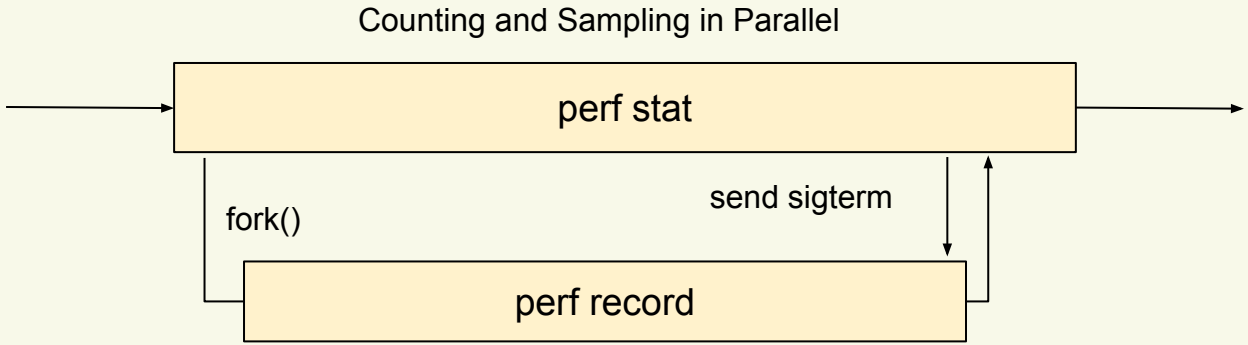
Offset	Field Name	Bits
0x0	Record Format	[31:0]
	Retire Latency	**[47:32]**
	Record Size	[63:48]
0x8	Instruction Pointer	[63:0]
0x10	Applicable Counters	[63:0]
0x18	TSC	[63:0]

From: Intel® Architecture Instruction Set Extensions and Future Features

Counting mode Timed PEBS strategy

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



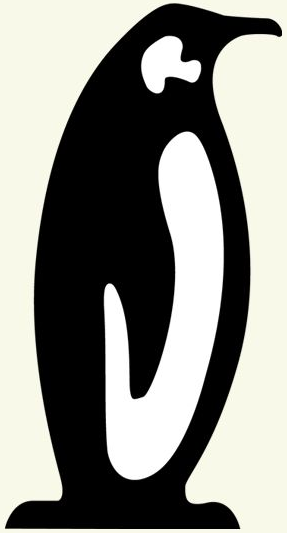
Discussion

- 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 topdown
  - 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.



Linux Plumbers Conference

Richmond, Virginia | November 13-15, 2023

Extra slides