Hardware Performance Monitoring Landscape

Stéphane Eranian

ProTools 2019

SuperComputing 2019

November 2019

Denver, CO

Agenda

- PMU hardware features
- Linux support
- Google usage
- Challenges

Intel SkylakeX core PMU

- 3 fixed counters, 4 generic counters (8 w/ HT off), all 48-bit wide
  - Global control and overflow status, intr on overflow, NMI
- Precise sampling: PEBS: eliminates interrupt-based sampling IP-skid
  - Supported by subset of at-retirement events
  - Sample recorded by ucode to virtual memory buffer
  - 1 intr per buffer full: **significant** overhead reduction = **increased** sampling frequency
  - Captures: IP, machine state, data addr for ld/st
- Topdown bottleneck decomposition support
- 32-deep Last Branch Record (LBR), 2x vs. Haswell/Broadwell
  - Captures consecutive taken branches (src/dst, prediction, **cycles since last taken branch captured**)
  - Can freeze on PMU interrupt, filter on branch types and priv levels, call stack mode

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B

Intel Skylake Timed LBR: Triad example

```
Triad: A[i] = B[i] + c * C[i]
Each vector is 256MB
```

```
408e10:movsd  (%rdx),%xmm2
408e14:movsd  (%rcx),%xmm1
408e18:add    $0x8,%rdx
408e1c:add    $0x8,%rcx
408e20:add    $0x8,%rsi
408e24:mulsd  %xmm0,%xmm1
408e28:addsd  %xmm2,%xmm1
408e2c:movsd  %xmm1,-0x8(%rsi)
408e31:cmp    %rax,%rdx
408e34:jne    408e10
```

```
$ perf record -b -e br_inst_retired.any:upp -c 1000002
$ perf script -F brstack
```

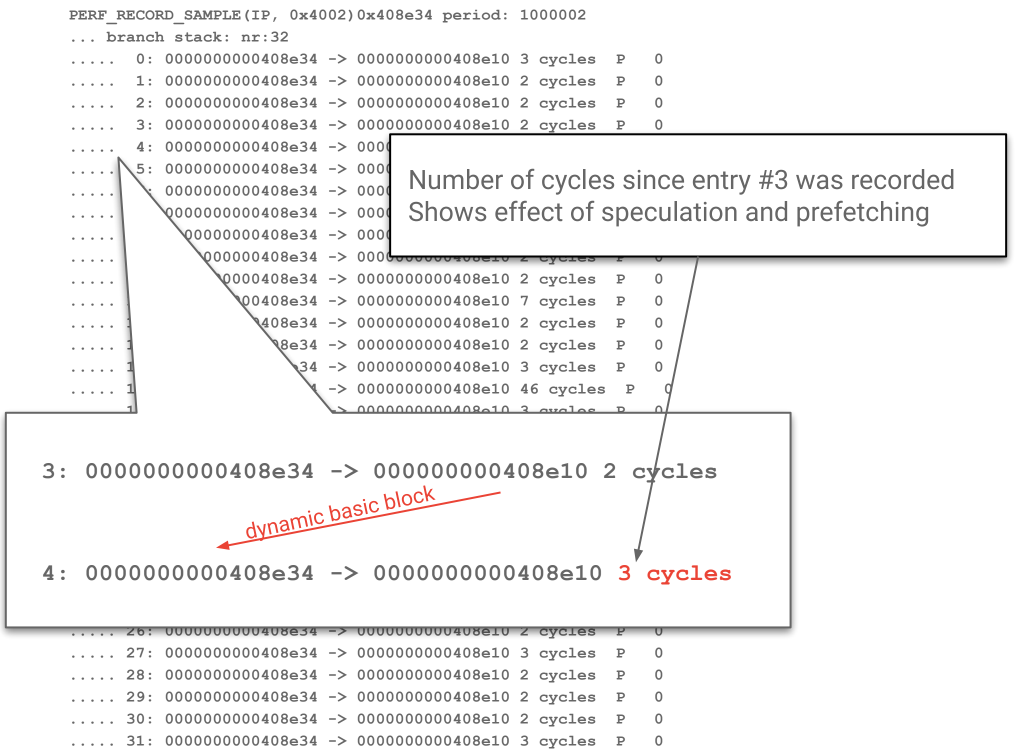
```
PERF_RECORD_SAMPLE(IP, 0x4002)0x408e34 period: 1000002
... branch stack: nr:32
.....  0: 0000000000408e34 -> 0000000000408e10 3 cycles  P   0
.....  1: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
.....  2: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
.....  3: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
.....  4: 0000000000408e34 -> 0000000000408e10 3 cycles  P   0
.....  5: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
.....  6: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
.....  7: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
.....  8: 0000000000408e34 -> 0000000000408e10 3 cycles  P   0
.....  9: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
..... 10: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
..... 11: 0000000000408e34 -> 0000000000408e10 7 cycles  P   0
..... 12: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
..... 13: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
..... 14: 0000000000408e34 -> 0000000000408e10 3 cycles  P   0
..... 15: 0000000000408e34 -> 0000000000408e10 46 cycles  P   0
..... 16: 0000000000408e34 -> 0000000000408e10 3 cycles  P   0
..... 17: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
..... 18: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
..... 19: 0000000000408e34 -> 0000000000408e10 3 cycles  P   0
..... 20: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
..... 21: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
..... 22: 0000000000408e34 -> 0000000000408e10 3 cycles  P   0
..... 23: 0000000000408e34 -> 0000000000408e10 252 cycles  P   0
..... 24: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
..... 25: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
..... 26: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
..... 27: 0000000000408e34 -> 0000000000408e10 3 cycles  P   0
..... 28: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
..... 29: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
..... 30: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
..... 31: 0000000000408e34 -> 0000000000408e10 3 cycles  P   0
```

Intel Skylake Timed LBR: Triad example

```
Triad: A[i] = B[i] + c * C[i]
Each vector is 256MB
```

```
408e10:movsd  (%rdx),%xmm2
408e14:movsd  (%rcx),%xmm1
408e18:add    $0x8,%rdx
408e1c:add    $0x8,%rcx
408e20:add    $0x8,%rsi
408e24:mulsd  %xmm0,%xmm1
408e28:addsd  %xmm2,%xmm1
408e2c:movsd  %xmm1,-0x8(%rsi)
408e31:cmp    %rax,%rdx
408e34:jne    408e10
```

```
$ perf record -b -e br_inst_retired.any:upp -c 1000002
$ perf script -F brstack
```



```
PERF_RECORD_SAMPLE(IP, 0x4002)0x408e34 period: 1000002
... branch stack: nr:32
.....  0: 0000000000408e34 -> 0000000000408e10 3 cycles  P   0
.....  1: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
.....  2: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
.....  3: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
.....  4: 0000000000408e34 -> 0000
.....  5: 0000000000408e34 -> 0000
.....     0000000000408e34 -> 0000
.....     000000000408e34 -> 0000
.....      00000000408e34 -> 0000
.....      0000000408e34 -> 0000
.....       000000408e34 -> 0000000000408e10 2 cycles  P   0
.....        00408e34 -> 0000000000408e10 7 cycles  P   0
.....  1      0408e34 -> 0000000000408e10 2 cycles  P   0
.....  1       8e34 -> 0000000000408e10 2 cycles  P   0
.....  1        34 -> 0000000000408e10 3 cycles  P   0
.....  1          -> 0000000000408e10 46 cycles  P   0
.....  26: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
.....  27: 0000000000408e34 -> 0000000000408e10 3 cycles  P   0
.....  28: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
.....  29: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
.....  30: 0000000000408e34 -> 0000000000408e10 2 cycles  P   0
.....  31: 0000000000408e34 -> 0000000000408e10 3 cycles  P   0
```

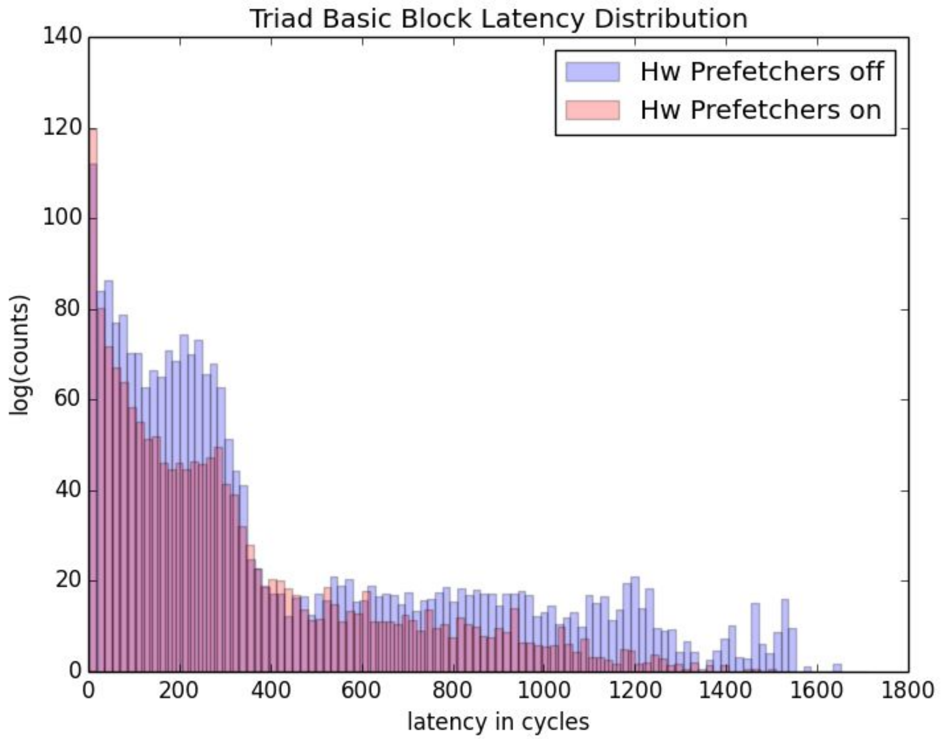
Number of cycles since entry #3 was recorded
Shows effect of speculation and prefetching

```
3: 0000000000408e34 -> 0000000000408e10 2 cycles

4: 0000000000408e34 -> 0000000000408e10 3 cycles
```

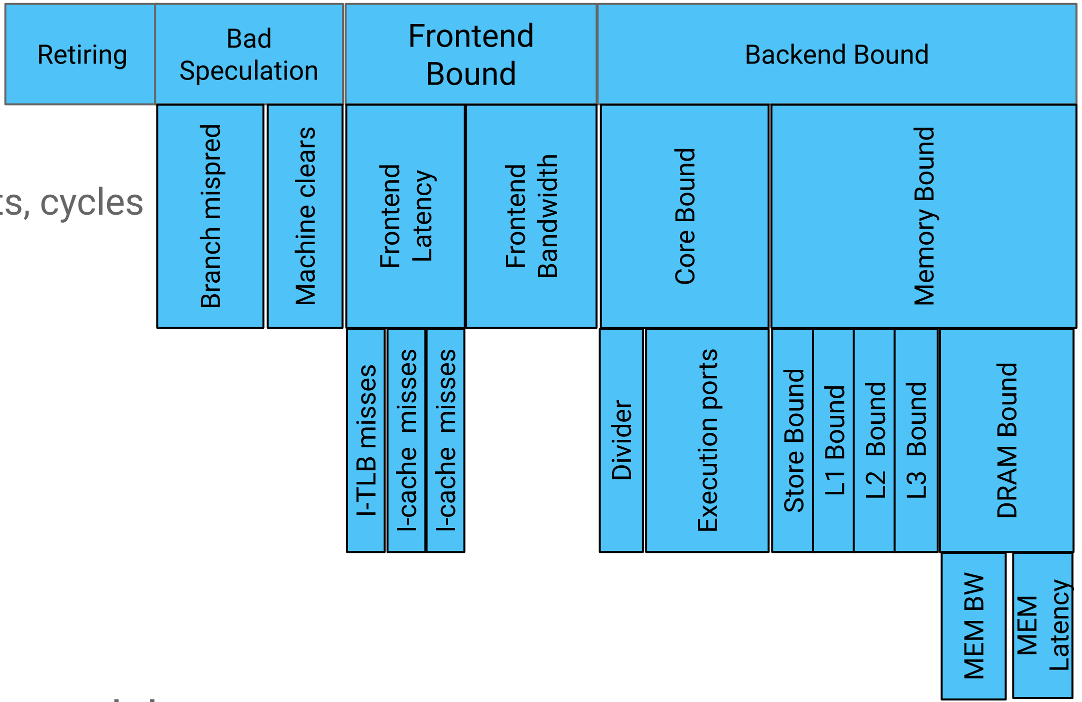
Intel Skylake Timed LBR: Triad example

- Measuring the effects of hardware prefetchers on core loop execution



Intel Topdown analysis

- Characterize hw bottlenecks
  - **NOT** a cycle decomposition
  - Does **not** tell how each cycle is used
  - Not all metrics use same unit, e.g., slots, cycles
- Hierarchical decomposition
  - Uses lots of PMU events
  - Toplevel: 5 events
- Counting mode analysis
  - Steers developer towards problem
- Sampling to locate bottlenecks
- Topdown specs available as XLS spreadsheet
  - Contains PMU events and formulas to compute each metric in the tree



Google

[Topdown spreadsheet]
[A Top-Down method for performance analysis and counters architecture, Ahmad Yasin, ISPASS14]

Intel Topdown example: Triad example

1

```
#=====================================================================
#               |                                             topdown
#               |---------------------------------------------------
#               |                     unit: Slots
#               |---------------------------------------------------
#               |     Frontend           Bad      Backend    Retiring
#               |        Bound   Speculation        Bound
#=====================================================================
 HW prefetch on          0.47%         0.11%       72.32%      27.10%
 HW prefetch off         0.28%         0.08%       88.67%      10.97%
```

```
Triad: A[i] = B[i] + c * C[i]
Each vector is 256MB
```

Google

Intel Topdown example: Triad example

1

```
#====================================================================
#                |                                           topdown
#                |---------------------------------------------------
#                |                       unit: Slots
#                |---------------------------------------------------
#                |     Frontend          Bad      Backend   Retiring
#                |        Bound  Speculation        Bound
#====================================================================
 HW prefetch on           0.47%        0.11%       72.32%     27.10%
 HW prefetch off          0.28%        0.08%       88.67%     10.97%
```

2

```
#=================================
#                |     topdown_be
#                |----------------
#                |   unit: Slots
#                |----------------
#                |  Memory    Core
#                |   Bound   Bound
#=================================
 HW prefetch on     66.43%   5.96%
 HW prefetch off    81.95%   6.71%
```

```
Triad: A[i] = B[i] + c * C[i]
Each vector is 256MB
```

Google

Intel Topdown example: Triad example

1

```
#===================================================================
#                |                                          topdown
#                |--------------------------------------------------
#                |                   unit: Slots
#                |--------------------------------------------------
#                |    Frontend          Bad       Backend   Retiring
#                |       Bound  Speculation         Bound
#===================================================================
 HW prefetch on          0.47%        0.11%        72.32%     27.10%
 HW prefetch off         0.28%        0.08%        88.67%     10.97%
```

2

```
#================================
#                |   topdown_be
#                |---------------
#                |  unit: Slots
#                |---------------
#                | Memory    Core
#                |  Bound   Bound
#================================
 HW prefetch on    66.43%   5.96%
 HW prefetch off   81.95%   6.71%
```

3

```
#===================================================
#               |                      topdown_be_mem
#               |-----------------------------------
#               |          unit: Stalls
#               |-----------------------------------
#               |    L1     L2     L3   DRAM  Store
#               | Bound  Bound  Bound  Bound  Bound
#===================================================
 HW prefetch on   6.67% 12.47%  1.03% 40.81%  3.31%
 HW prefetch off  0.18%  0.00%  2.78% 75.87%  1.11%
```

```
Triad: A[i] = B[i] + c * C[i]
Each vector is 256MB
```

Google

Intel Topdown example: Triad example

1

```
#=================================================================
#               |                                          topdown
#               |-------------------------------------------------
#               |                      unit: Slots
#               |-------------------------------------------------
#               |    Frontend          Bad     Backend    Retiring
#               |       Bound  Speculation       Bound
#=================================================================
 HW prefetch on         0.47%        0.11%      72.32%      27.10%
 HW prefetch off        0.28%        0.08%      88.67%      10.97%
```

2

```
#===============================
#              |     topdown_be
#              |----------------
#              |   unit: Slots
#              |----------------
#              |  Memory    Core
#              |   Bound   Bound
#===============================
 HW prefetch on    66.43%   5.96%
 HW prefetch off   81.95%   6.71%
```

3

```
#=================================================
#               |                 topdown_be_mem
#               |---------------------------------
#               |         unit: Stalls
#               |---------------------------------
#               |    L1     L2     L3  DRAM  Store
#               | Bound  Bound  Bound Bound  Bound
#=================================================
 HW prefetch on    6.67% 12.47%  1.03% 40.81%  3.31%
 HW prefetch off   0.18%  0.00%  2.78% 75.87%  1.11%
```

4

```
#======================================
#               |   topdown_be_mem_dram
#               |-----------------------
#               |     unit: Clocks
#               |-----------------------
#               |         MEM        MEM
#               |   Bandwidth    Latency
#======================================
 HW prefetch on        65.35%     13.43%
 HW prefetch off       51.28%     26.78%
```

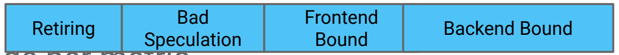
```
Triad: A[i] = B[i] + c * C[i]
Each vector is 256MB
```

Google

Steering towards largest bottleneck

Intel Icelake core PMU

- 8 generic counters, 2x Skylake!
  - But event constraints are back (`PMC0-PMC3` support more events)
- New fixed counter: `TOPDOWN.SLOTS`
  - Counts issue slots per thread = `cycles * machine_width`
- New fixed counter : `PERF_METRICS` for Topdown
  - High level derived metrics: 4 top level metrics in one MSR, 8-bit percentage per metric
  - Reduces pressure from 5 counters to 1 (+SLOTS)
  - Topdown possible **per thread** (vs. only per core on Skylake)



- Extended PEBS
  - **All counters** support PEBS (vs. only 4 generic counters on Skylake)
  - **All events** can use the buffer and recoding of machine state
  - Non at-retirement events have smaller skid, but not skidless
- Adaptive PEBS
  - PEBS record is configurable by register groups vs. fixed size on previous generation (192 bytes)
  - Can record `XMM0-XMM7 and full LBR`

Google

[Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B](#)

Intel Icelake PMU: `PERF_METRICS` example

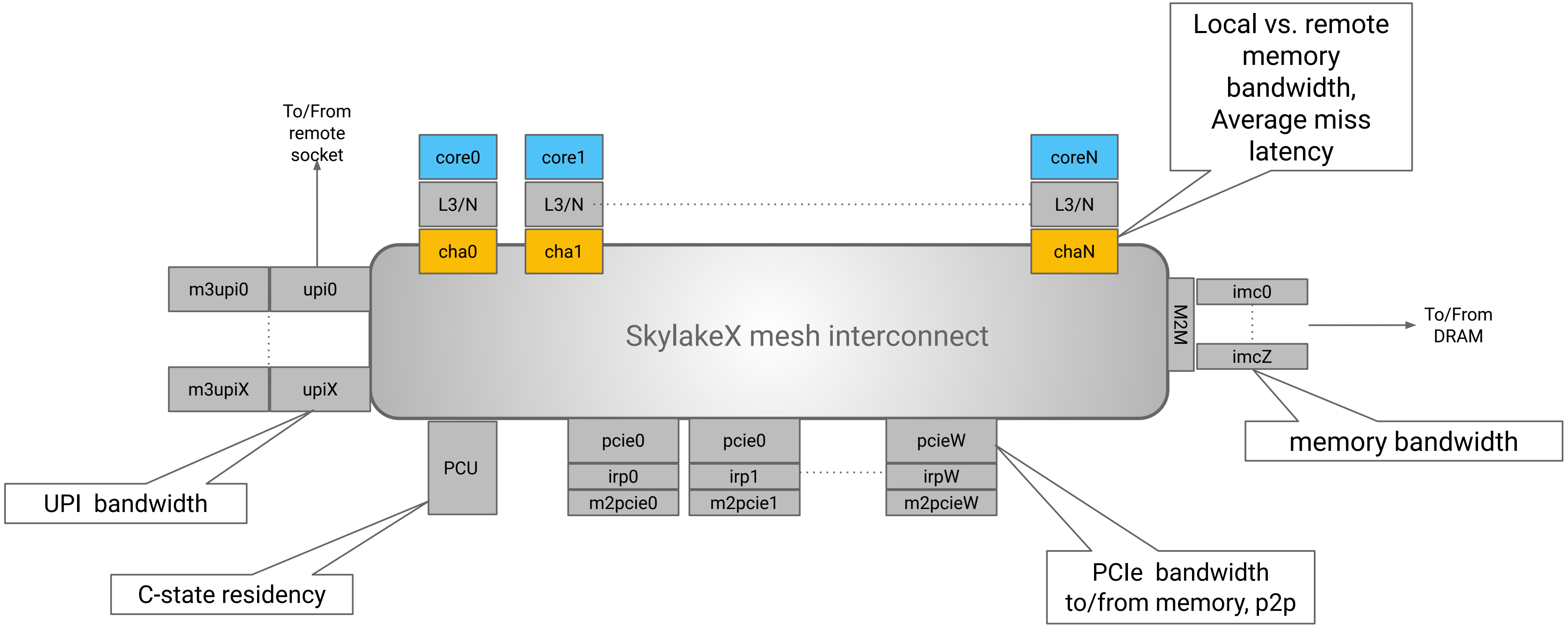
```
$ perf stat -I 1000 --topdown -a
#           time             counts unit events
     1.000373951      8,460,978,609      topdown-retiring # 22.9% retiring
     1.000373951      3,445,383,303      topdown-bad-spec #  9.3% bad speculation
     1.000373951     15,886,483,355      topdown-fe-bound # 43.0% frontend bound
     1.000373951      9,163,488,720      topdown-be-bound # 24.8% backend bound
```

Multiplying `topdown.slots` to scale counts

Actual Topdown breakdown

Google

Intel SkylakeX uncore PMUs



Intel® Xeon® Processor Scalable Memory Family Uncore Performance Monitoring Reference Manual

Google

Intel SkylakeX CacheQoS

- Intel RDT technology introduced in server SKUs with HaswellX
  - 2 monitoring and 2 enforcement components in the L3 cache
- L3 occupancy and partitioning
  - CMT: monitoring allocations/thread via Resource Management ID (4 RMID/core) tag
  - CAT: partitioning per Class of Service ID (16 CLOSID), config bitmask, e.g, 10-bit mask, □th cache
  - CDP: variant of CAT where partitions are split between code and data
  - RMID/CLOSID saved/restored on context switch
- Memory bandwidth
  - MBM: bw usage/thread via RMID, includes reads/writes, local vs. total memory bw
  - MBA: limit bandwidth usage per CLOSID(8), set % of limit in 10% increments in [10-90]
- Very useful because tracking and enforcement can be done per thread

Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B, Chapter 17

AMD Zen2 core PMU

- 6 48-bit generic counters/thread
  - interrupt on overflow support, including non-maskable interrupt (NMI)
  - No event counter constraints
- Max incr. 15/cycle: problem for some events such as `FLOPS`
  - `MERGE` event: fuse to consecutive counters
  - Kernel patch proposed by AMD on LKML to allow fusing, currently no support
- Extremely long PMU interrupt latency
  - Causes problems with skid, has kernel race conditions especially in frequency sampling mode
  - AMD mitigation (v5.2): poll for <=50μs on PMU disable if a counter has overflowed (bit 47 clear)
- No global counter controls, no overflow status
  - All counters fully independently controlled

Processor Programming Reference (PPR) for Family 17h Model 31h, Revision B0 Processors

Google

AMD Zen2 core events

- Not much changes since Fam15h
- 6 event categories
  - FPU: floating points, incl. FLOPS
  - LS: load/store, dtlb, prefetch
  - IC/BP: icache, itlb, branches
  - DE: decoders
  - EX: instructions
  - L2: L2 cache
- Only `CYCLES_NOT_IN_HALT` event to count core cycles
  - No references cycles event, only `MPERF` MSR (no sampling, system-wide only)
- No topdown-style bottleneck decomposition
  - Few stalls events

Processor Programming Reference (PPR) for Family 17h Model 31h, Revision B0 Processors

AMD Zen2 IBS

- AMD precise sampling feature unchanged since Fam15h
- Samples μ-ops at random
  - Interrupts when tagged μ-op on correct path retires, otherwise retry another μ-op
  - Period expressed in **either cycles or μ-op** (with hw randomization)
  - Returns: precise IP, lots of info about μ-op depending on type
  - **Latency** of μ-op: tag to retirement (total exec), completion to retirement (retirement delay)
  - Branch: source, destination, prediction, direction, type
  - Load/Store: IP, phys addr, data virt addr, data phys addr, tlb, data source
- No filtering
  - Cannot use to target specific condition, e.g., L3 misses
- No buffering
  - One interrupt per sampled μ-op

Processor Programming Reference (PPR) for Family 17h Model 31h, Revision B0 Processors

AMD Zen2 IBS_OP on Triad: perfect!

```
$ perf record -a -C 0 -c 825000 -e rc1:pp  sleep 10 (sampling in μ-op domain)
          11.28 │50:    movsd  (%rax),%xmm1
          11.11 │       movsd  (%rcx),%xmm0
          11.13 │       add    $0x8,%rax
          11.13 │       add    $0x8,%rcx
          10.95 │       add    $0x8,%rdx
          11.01 │       mulsd  %xmm2,%xmm0
          11.22 │       addsd  %xmm1,%xmm0
          11.08 │       movsd  %xmm0,-0x8(%rdx)
          11.08 │       cmp    %rsi,%rax  ⎫ macro-fused
                │       jne    50         ⎭ insn
```

9 insns
Each 1/9th samples
11.1%

- Good: If want to understand basic block execution count
- Bad : if I want to understand where the load cache misses are
  - only 22% of samples are relevant here (the 2 loads in red)

Google

AMD Zen2 LBR

- No changes from Fam15h
- 1-deep Last Branch Record
  - Captures 1 source/destination
  - Controlled by `DBG_CTL_MSR`
- No connection to core PMU: no freeze on PMI

[Processor Programming Reference (PPR) for Family 17h Model 31h, Revision B0 Processors]

Google

AMD Zen2 uncore PMUs

- L3 PMU[1]
  - Shared by 4 cores/8 threads
  - 6 counters/CCX
  - Possibility to set a thread mask (8-bit) and slice mask (4-bit) per counter
  - Events: number of requests or misses, miss latency
  - **No possibility to measure L3 activity from core**
- Data Fabric (DF) PMU[2]
  - Shared by all Core Complexes (CCX) on socket
  - 4 counters/DF
  - Events: none published
- IOMMU PMU[3]

[1] Processor Programming Reference (PPR) for Family 17h Model 31h, Revision B0 Processors
[2] Linux kernel source tree: arch/x86/events/amd/uncore.c
[3] Linux kernel source tree: arch/x86/events/amd/iommu.c

Google

AMD Zen2 CacheQoS

- First implementation of CacheQoS on AMD
- Hardware interface very similar to Intel's RDT
  - RMID (Resource Management ID) to track resource usage
  - COS (Class of Service ID) to enforce class of service restrictions
- L3 Cache support
  - Monitoring: track cache **allocations** per RMID
  - Enforcement: partitioning via cache bitmask(CBM) per COS, CBM bit = 1/n^{th} of cache if LEN=n
- Memory Bandwidth
  - Monitoring: bandwidth per RMID, covers only reads (L3 fills), total vs. local supported
  - Enforcement: bandwidth limits per COS, at ⅛GB/s granularity, covers reads and writes

AMD64 Technology Platform Quality of Service Extensions

Google

Linux support in 5.4

- Intel Icelake
  - Extra core PMU counters
  - Extended PEBS, Adaptive PEBS
  - `PERF_METRICS` still pending approval
- AMD Zen2
  - Core PMU
  - IBS PMU (both IBS OP & FETCH)
  - L3 and DF PMU
- CacheQoS
  - From Intel HaswellX to CascadeLake processors, and some Atom processors (for L2 CAT)
  - AMD Zen2

Linux CacheQoS support: `resctrl`

- Abstract Interface to CacheQoS features: `resctrl` filesystem
  - Supports Intel Xeon and AMD Zen2 QoS
- `resctrl != cgroup`
  - Operates similarly to `cgroup`: move threads (not processes) into `resctrl` group
  - Each `resctrl` group assigned a RMID and CLOSID

```
Example: Read BW from local memory:
     $ cd /sys/fs/resctrl; mkdir grp; cd grp; echo $$ > tasks
     $ taskset -c 0 my_test &
     $ a=$(cat mon_data/mon_L3_00/mbm_local_bytes)
     $ sleep 1
     $ b=$(cat mon_data/mon_L3_00/mbm_local_bytes)
     $ echo "$(((b - a) >> 20)) MiB/s"
```

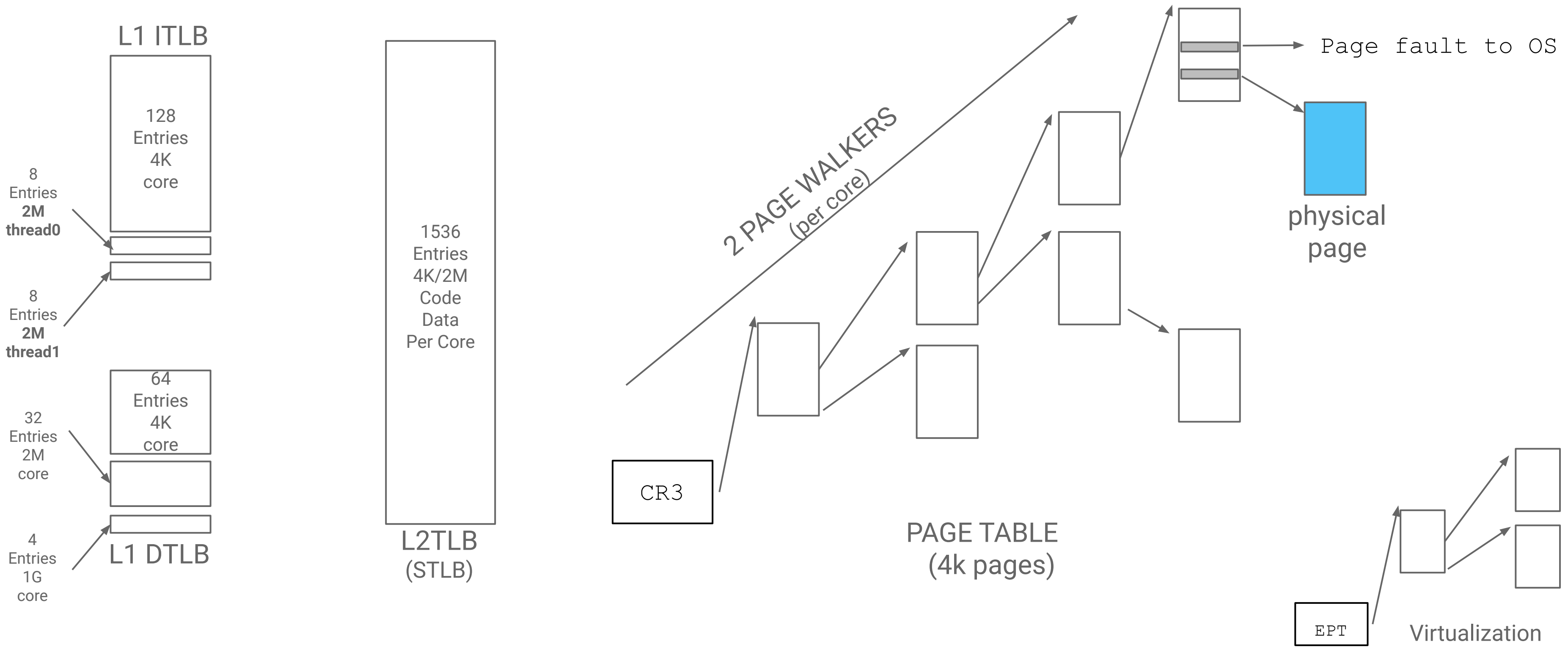
Resctrl UI documentation

Google

PMU events

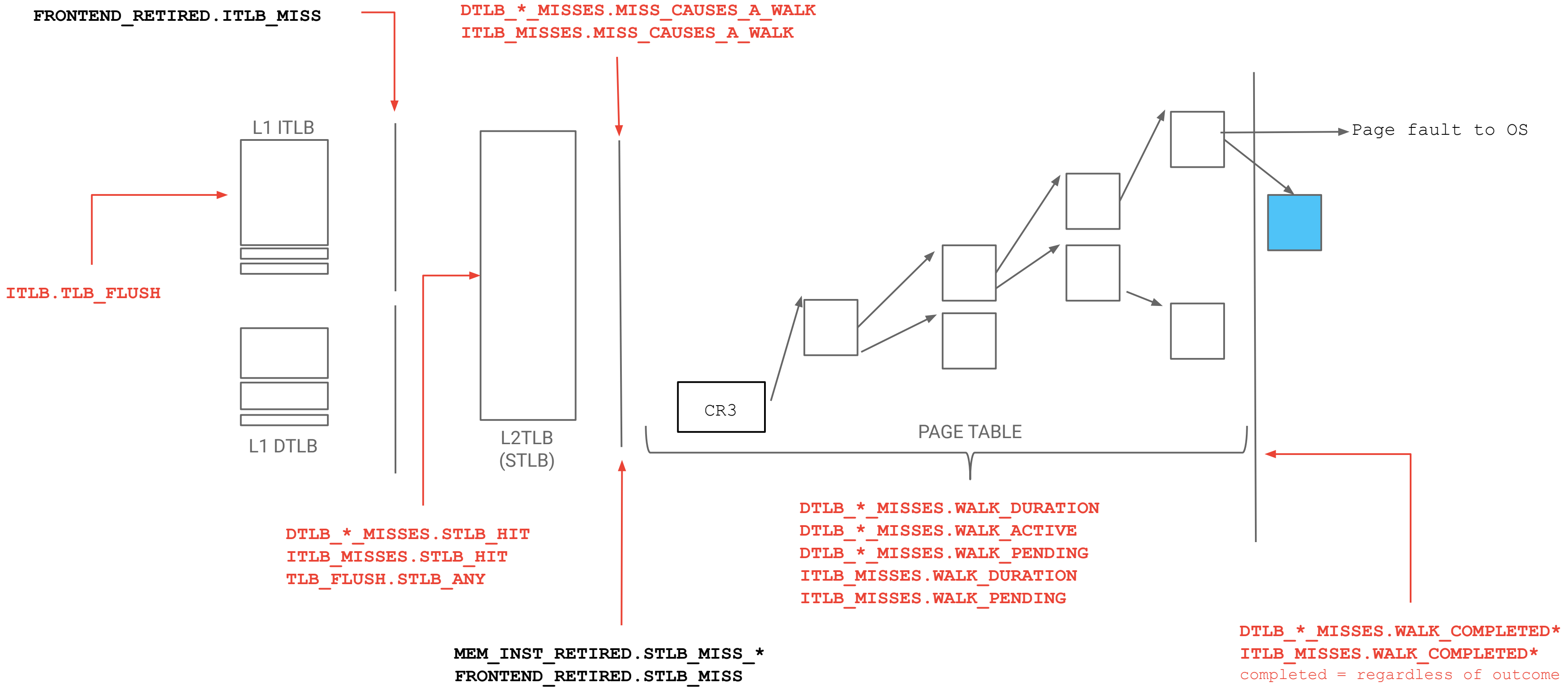
- Micro-architectural events oftentimes difficult to generalize
  - Specific to a micro-architecture, e.g., number of TLB levels, speculative execution
  - May be hard to compare from one CPU generation to another
- What does `cycles` (Linux generic event) event measure?
  - Core cycles? Regardless of SMT
  - Turbo cycles? Increments faster with CPU clock
  - Reference cycles? Increments at constant rate regardless of Turbo
  - Does it count in all C-states?
- What does `cache-misses` (Linux generic event) event measure?
  - At what cache level? For load or stores? For code or data? Speculative accesses?
  - Hardware prefetchers? Software prefetches?
- Always prefer actual PMU events to generic versions
  - Know what you want to measure

Intel Skylake TLB structures



Intel® 64 and IA-32 Architectures Optimization Reference Manual

Intel Skylake TLB event example

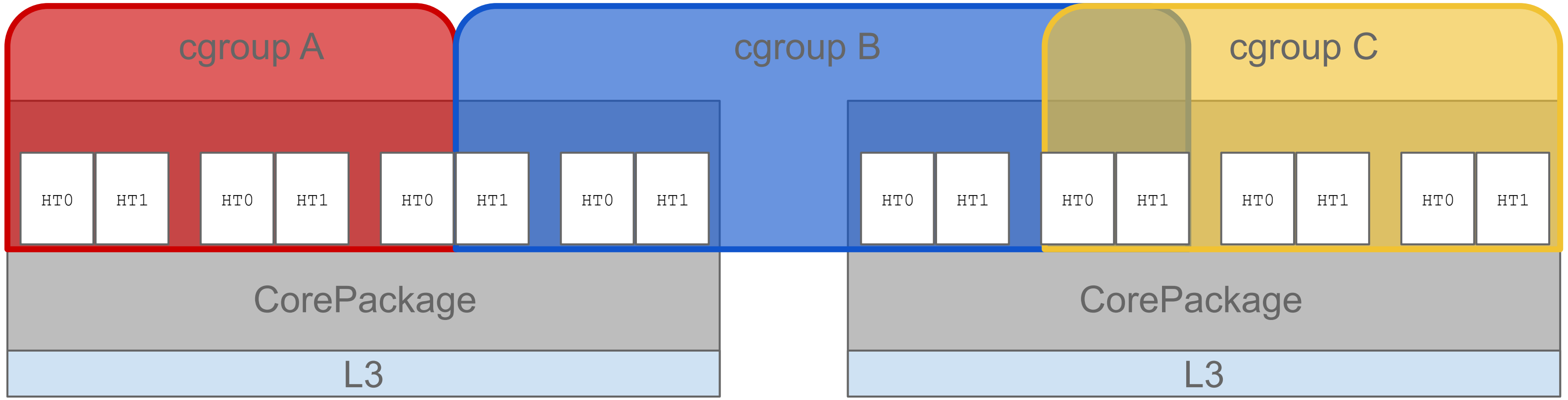

FRONTEND_RETIRED.ITLB_MISS
DTLB_*_MISSES.MISS_CAUSES_A_WALK
ITLB_MISSES.MISS_CAUSES_A_WALK
L1 ITLB
ITLB.TLB_FLUSH
L1 DTLB
L2TLB
(STLB)
Page fault to OS
CR3
PAGE TABLE
DTLB_*_MISSES.STLB_HIT
ITLB_MISSES.STLB_HIT
TLB_FLUSH.STLB_ANY
DTLB_*_MISSES.WALK_DURATION
DTLB_*_MISSES.WALK_ACTIVE
DTLB_*_MISSES.WALK_PENDING
ITLB_MISSES.WALK_DURATION
ITLB_MISSES.WALK_PENDING
MEM_INST_RETIRED.STLB_MISS_*
FRONTEND_RETIRED.STLB_MISS
DTLB_*_MISSES.WALK_COMPLETED*
ITLB_MISSES.WALK_COMPLETED*
completed = regardless of outcome

ALL EVENTS IN RED ARE SPECULATIVE

How is Google using all of this?

Google production environment

- Everything runs in container groups (cgroups)
  - Cgroup: Linux resource encapsulation abstraction: cpuset, memory, …
  - Provides relative isolation
- Jobs are dispatched by a scheduling infrastructure called Borg
- Servers may be dedicated or shared between different jobs



Google PMU data collector

- The `perf` tool is installed on all production servers
  - Can be invoked from remote via a dedicated daemon
- `perf` contains custom extensions
  - Handling of hugepage text
  - High level core and uncore PMU metrics

mem_bw (uncore metrics)

```
#------------------------------------------
#       Socket0      |       Socket1      |
#------------------------------------------
#   RAM Bandwidth    |   RAM Bandwidth    |
#        Wr        Rd|        Wr        Rd|
#      MB/s      MB/s|      MB/s      MB/s|
#------------------------------------------
      16.76     16.70       9.25     11.07
```

Topdown (core metrics)

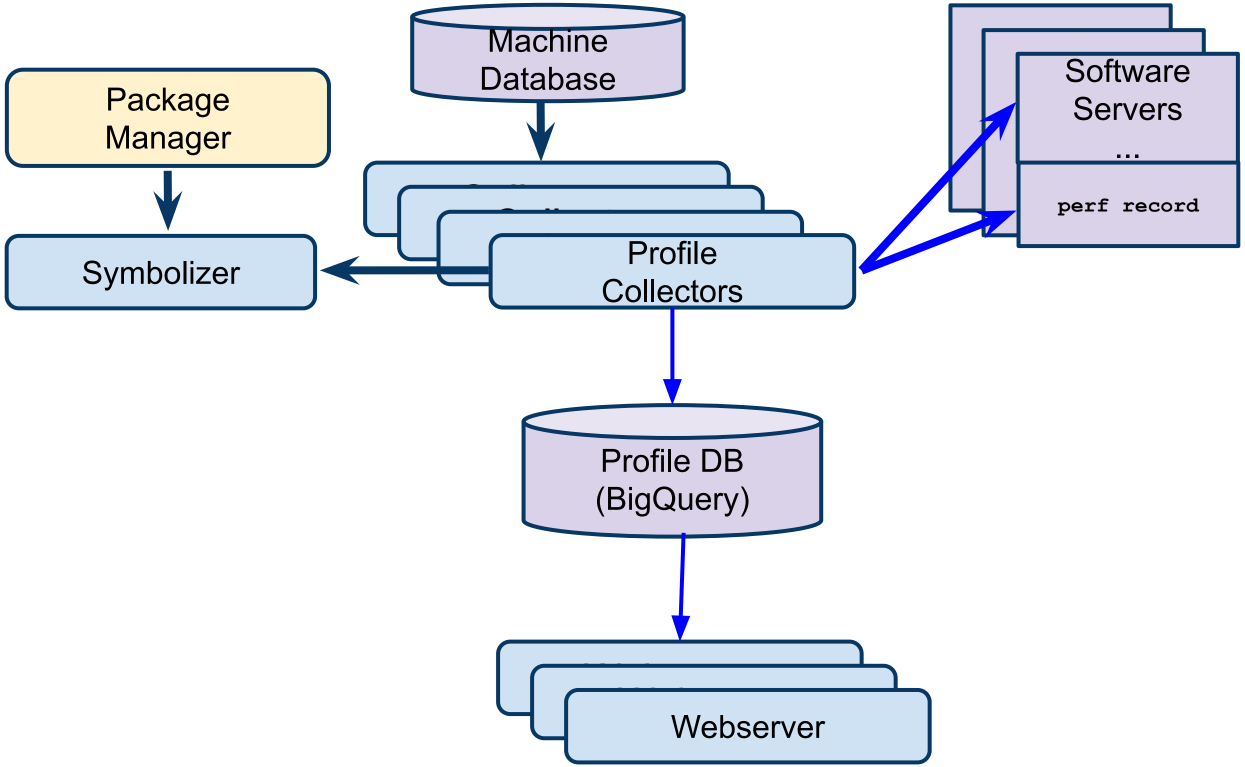
```
#==================================================
#               |                           topdown
#               |----------------------------------
#               |         unit: issue slots
#               |----------------------------------
#               |FrontEnd     Bad   Uops    BackEnd
#               | Bound      Spec Retiring   Bound
#==================================================
     1.038100775   29.05%   7.32%   11.53%   52.10%
```

Google-Wide-Profiler (GWP)

- Fleet wide profiler infrastructure
  - Collects performance monitoring data on XX% of fleet/day for xx secs
- At scale, GWP delivers useful profile despite collecting for a few seconds
- GWP collects PMU and other performance metrics per machine
  - Invokes `perf` record tool in system-wide mode on various PMU events
  - `perf record` invoked in pipe mode to avoid disk I/O. Data processed offline.
- Symbolizations challenge
  - Database of all symbols for all deployed binaries, tagged by BuildID

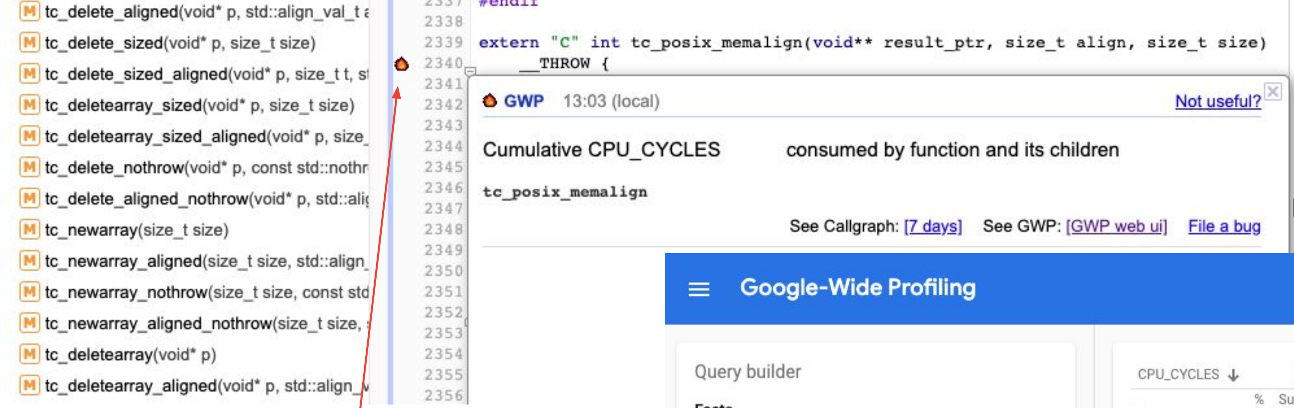
Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers, Gang Ren,Eric Tune, Tipp Moseley, Yixin Shi,Silvius Rus,Robert Hundt, Google

GWP architecture



GWP usefulness

- Can help answer wealth of questions about how software runs fleet wide
- Classic queries:
  - What is the function consuming most cycles fleetwide? And which applications are calling it?
  - What is the application consuming most cycles fleetwide?
  - What library is most used?
  - Do we have a lot of TLB page walks? On which functions, binaries?
- Can drill down to specific binary version and assembly code
- Queries via a GUI, SQL, and programming
- Small improvements on hot functions make a huge difference at scale



- tc_delete_aligned(void* p, std::align_val_t a
- tc_delete_sized(void* p, size_t size)
- tc_delete_sized_aligned(void* p, size_t t, s
- tc_deletearray_sized(void* p, size_t size)
- tc_deletearray_sized_aligned(void* p, size_
- tc_delete_nothrow(void* p, const std::nothr
- tc_delete_aligned_nothrow(void* p, std::alig
- tc_newarray(size_t size)
- tc_newarray_aligned(size_t size, std::align_
- tc_newarray_nothrow(size_t size, const std
- tc_newarray_aligned_nothrow(size_t size,
- tc_deletearray(void* p)
- tc_deletearray_aligned(void* p, std::align_v

```
#endif

extern "C" int tc_posix_memalign(void** result_ptr, size_t align, size_t size)
    __THROW {
```

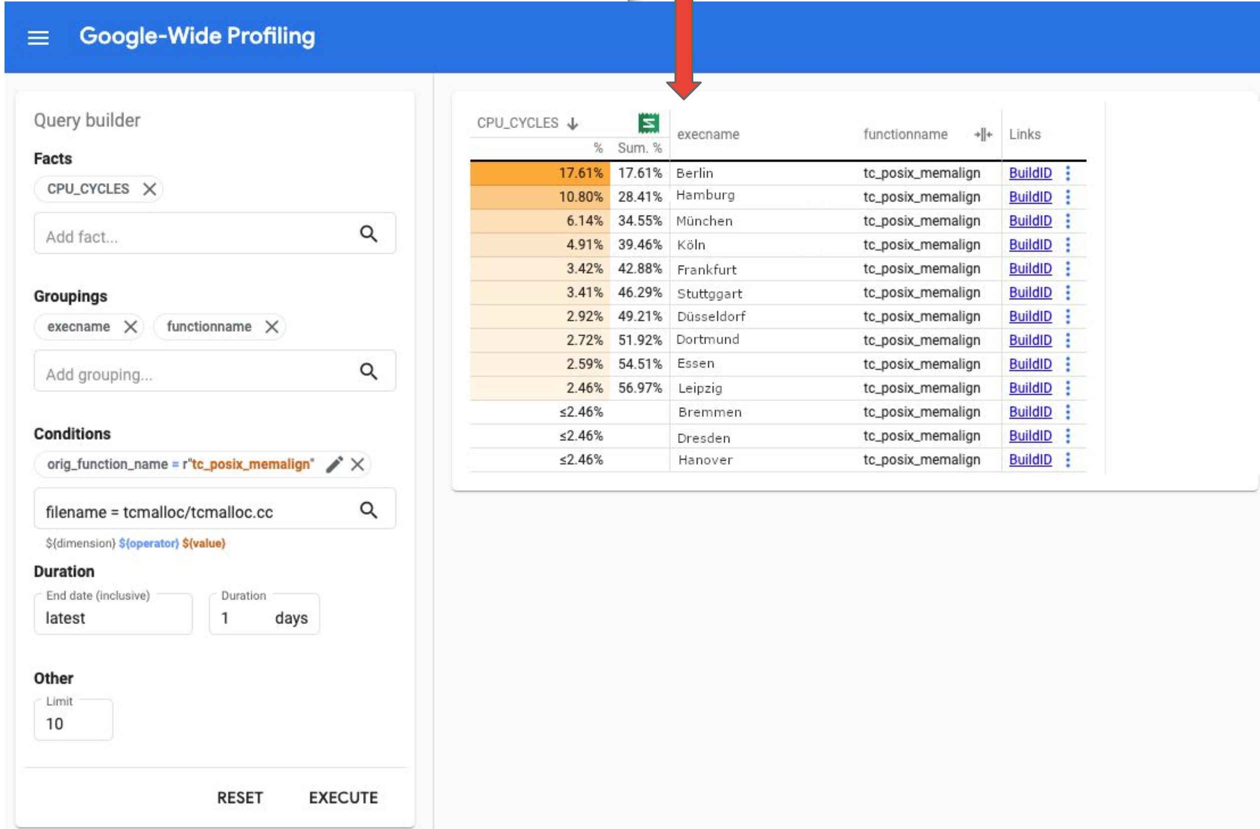
GWP 13:03 (local) Not useful?

Cumulative CPU_CYCLES consumed by function and its children

`tc_posix_memalign`

See Callgraph: [7 days] See GWP: [GWP web ui] File a bug

Function is tagged as hot by GWP data



Google-Wide Profiling

Query builder

Facts

CPU_CYCLES

Add fact...

Groupings

execname functionname

Add grouping...

Conditions

orig_function_name = r"tc_posix_memalign"

filename = tcmalloc/tcmalloc.cc

${dimension} ${operator} ${value}

Duration

End date (inclusive): latest

Duration: 1 days

Other

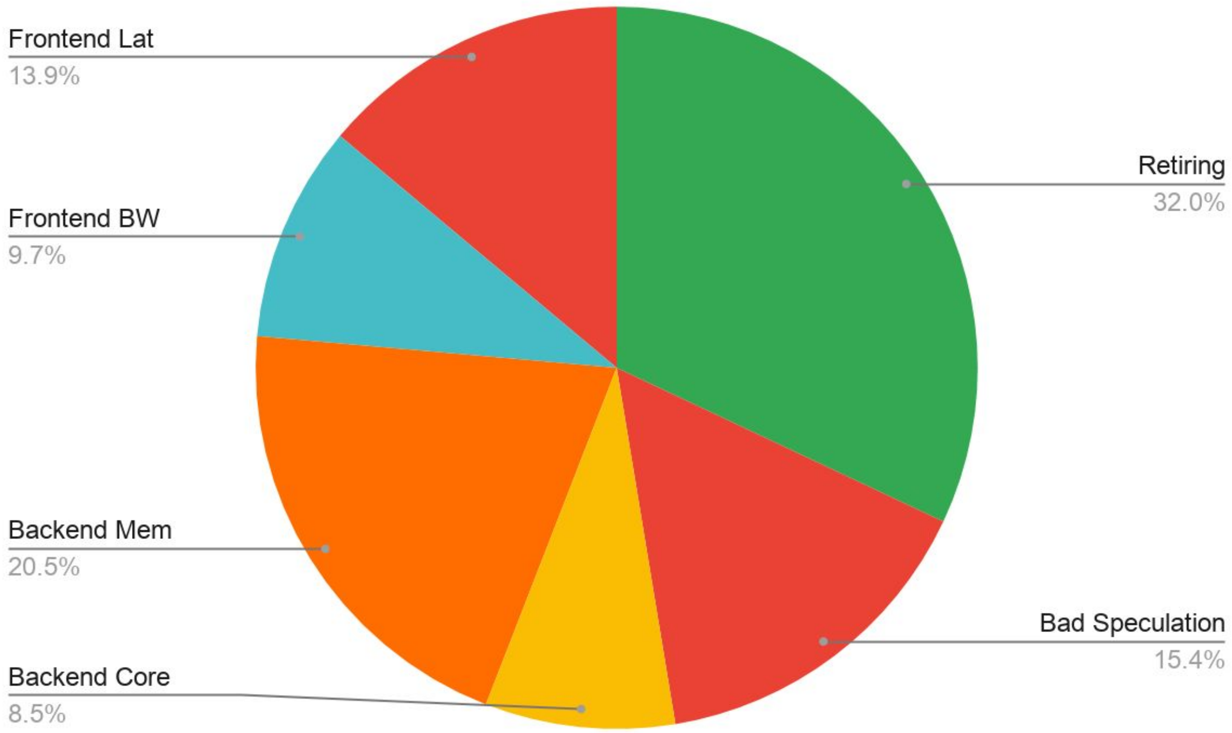
Limit: 10

RESET EXECUTE

CPU_CYCLES %	Sum. %	execname	functionname	Links
17.61%	17.61%	Berlin	tc_posix_memalign	BuildID
10.80%	28.41%	Hamburg	tc_posix_memalign	BuildID
6.14%	34.55%	München	tc_posix_memalign	BuildID
4.91%	39.46%	Köln	tc_posix_memalign	BuildID
3.42%	42.88%	Frankfurt	tc_posix_memalign	BuildID
3.41%	46.29%	Stuttgart	tc_posix_memalign	BuildID
2.92%	49.21%	Düsseldorf	tc_posix_memalign	BuildID
2.72%	51.92%	Dortmund	tc_posix_memalign	BuildID
2.59%	54.51%	Essen	tc_posix_memalign	BuildID
2.46%	56.97%	Leipzig	tc_posix_memalign	BuildID
≤2.46%		Bremmen	tc_posix_memalign	BuildID
≤2.46%		Dresden	tc_posix_memalign	BuildID
≤2.46%		Hanover	tc_posix_memalign	BuildID

Google

Topdown on parts of Websearch



Google

AsmDB: Understanding and Mitigating Front-End Stalls in Warehouse-Scale Computers,Grant Ayers, Nayana Prasad Nagendra, David I. August, Hyoun Kyu Cho, Svilen Kanev, Christos Kozyrakis,Trivikram Krishnamurthy,Heiner Litz, Tipp Moseley, ISCA'19

AsmDB

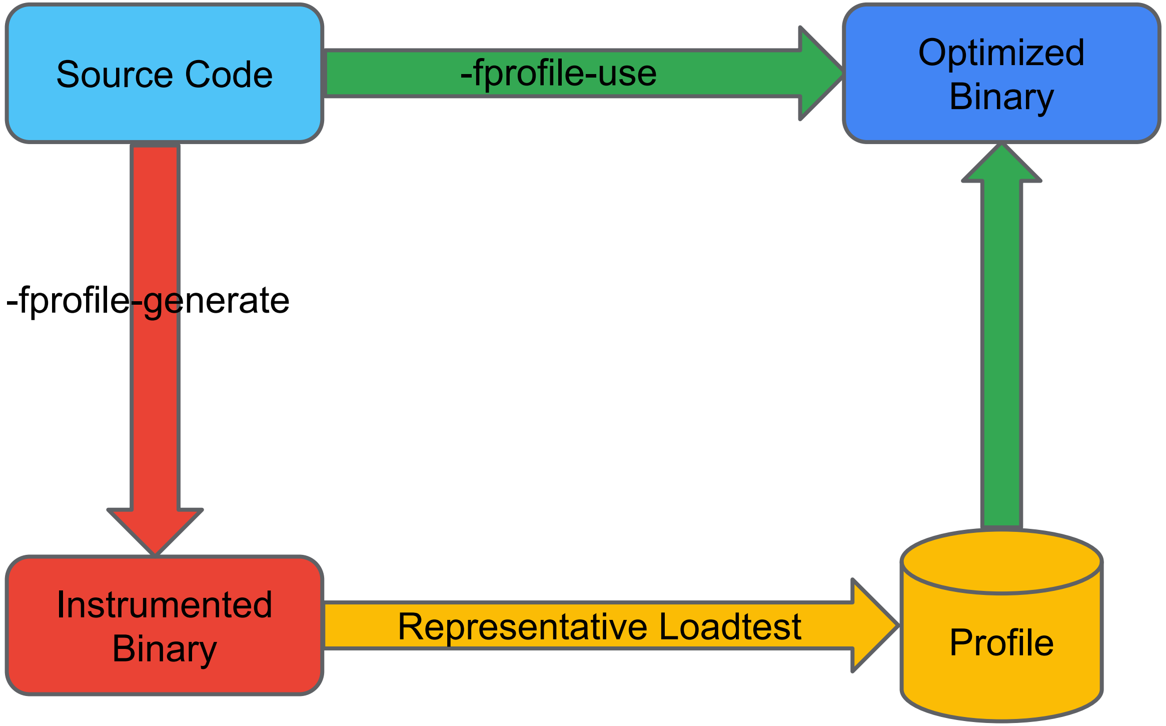
- Goal is to have a database of **nearly all** executed instructions in our data-centers
  - Q: Are we executing x87 insn 'X' ? Q: What is the most common type of load addressing?
  - Ranking of insn
  - DB with one row per insn
- Uses LBR data to identify executed basic blocks
  - Only keep top 1000 binaries by cycles consumed: covers 90% of all cycles
  - Heavy offline post processing: basic block predecessors, identify loops
  - 600GiB/day of data!
- Data used for more advanced analysis
  - Spotting manual optimizations
  - Compiler optimization opportunities
  - Invaluable for code analysis: code working set, + GWP data icache misses and control flow
  - See example advanced analysis in frontend stall paper from Grant Ayers et al

AsmDB: Understanding and Mitigating Front-End Stalls in Warehouse-Scale Computers,Grant Ayers, Nayana Prasad Nagendra, David I. August, Hyoun Kyu Cho, Svilen Kanev, Christos Kozyrakis,Trivikram Krishnamurthy,Heiner Litz, Tipp Moseley, Parthasarathy Ranganathan

Automatic Feedback-Driven Optimization (autoFDO)

- Feedback-directed Optimization (FDO)
  - Use compiler to optimize code by providing execution information for load test run
  - Usually involved 2-pass compilation to instrument(1) and optimize(2)
- Google has lots of applications!
- Not every application has a representative benchmarks
  - Some cannot even create a benchmark (e.g., because hard to get good input data, scale)
  - Behavior may change based on day of the week or the month
- Traditional Feedback Driven Optimization (FDO) is not practical
  - Too much overhead, cannot deploy to production just to collect a profile
- How can we get the benefits of FDO without instrumenting code?

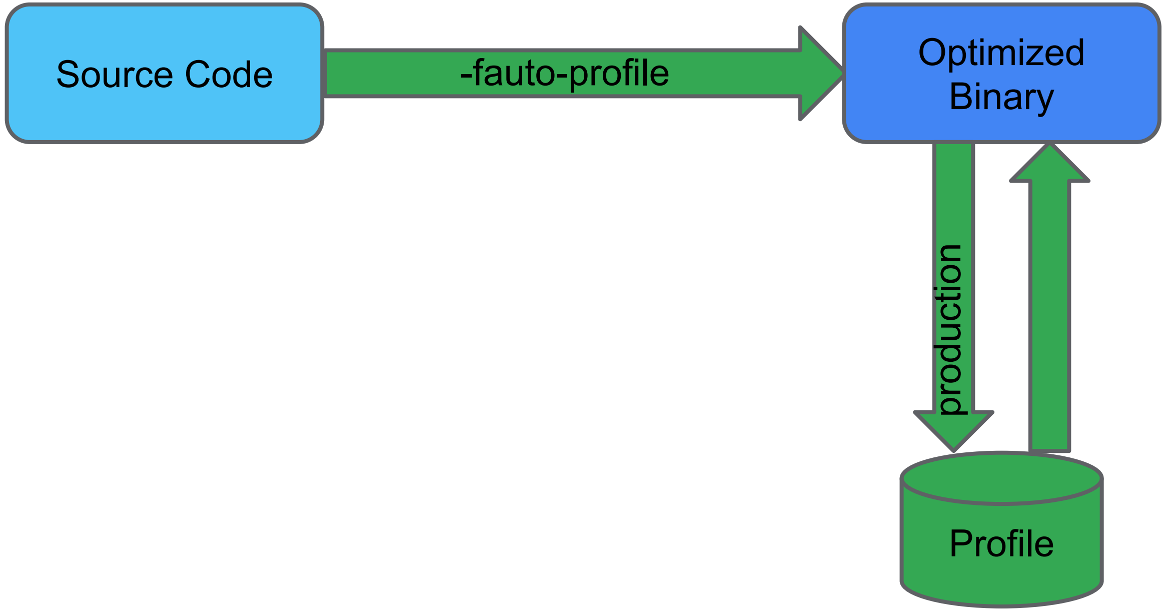
Traditional FDO



1. compile with profiling instrumentation
2. run a load test
    a. instrumentation slowdown
    b. representative input hard, especially with sensitive data and dependent services
3. Recompile and release

Google

AutoFDO



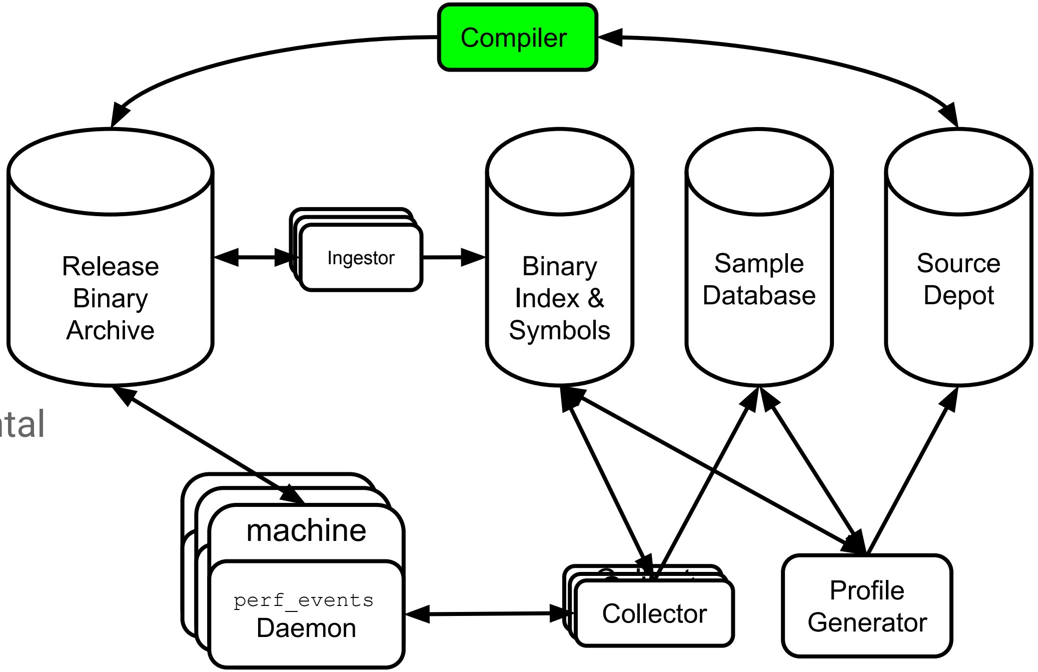
1. compile and release

Available to **ANY** applications: live production, no loadtest needed

Google

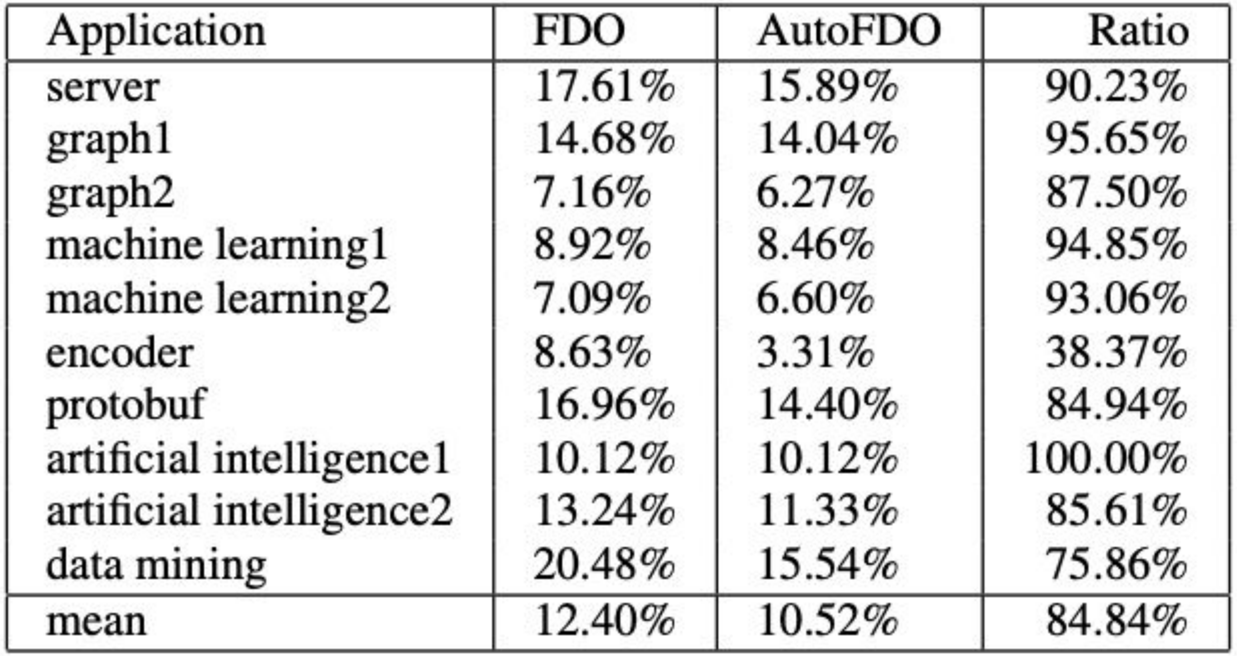
AutoFDO pipeline

- LBR profiles collected via GWP
  - Used for basic block exec count
  - Submitted weekly to database
- Converted to source location
  - func, src offset to func start, disc
- Staleness of profiles
  - Changes for most apps are incremental
- Transformations
  - Straightening of hot paths
  - Indirect call promotions
  - Many more possible…



Google

autoFDO: comparison with FDO



Application	FDO	AutoFDO	Ratio
server	17.61%	15.89%	90.23%
graph1	14.68%	14.04%	95.65%
graph2	7.16%	6.27%	87.50%
machine learning1	8.92%	8.46%	94.85%
machine learning2	7.09%	6.60%	93.06%
encoder	8.63%	3.31%	38.37%
protobuf	16.96%	14.40%	84.94%
artificial intelligence1	10.12%	10.12%	100.00%
artificial intelligence2	13.24%	11.33%	85.61%
data mining	20.48%	15.54%	75.86%
mean	12.40%	10.52%	84.84%

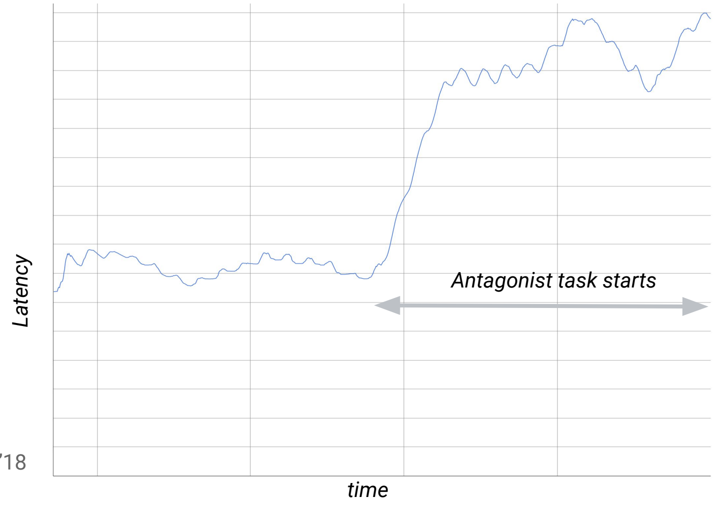
Speedups against -O2 binaries

Most of the benefit of traditional FDO without the recompilation, just a flag on the Makefile

Borg: handling the noisy neighbor problem!

- Bigger machines = shared by more jobs = more resource pressures
  - One job (e.g., ML) can consume a lot of memory bandwidth and cause major slowdown to others
- Must identify antagonistic and limit damage
  - Per-job bandwidth monitoring using PMU (Intel `OFFCORE_RESPONSE`) or CacheQoS MBM (better)
  - Throttle if exceed set threshold using CacheQoS MBA and reduce job bandwidth quota

```
Ex: limit bw to 10% of peak for threads in mygrp
$ cd /sys/fs/resctrl; mkdir mygrp; cd mygrp
$ cat schemata
$ echo MB:0=100;1=100
$ echo "MB:0=10;1=10" > schemata
$ echo $$ >tasks
$ my_test <- runs capped at 10% of peak BW
```



PMU challenges

- Event Validation
  - How to validate all the core and uncore PMU events to remove bugs very early?
- Events vs. counters
  - Want more events to measure more conditions
  - Want more counters to avoid multiplexing
  - But more counters = larger machine state to manage in the kernel
- Better events that count metrics we care about
  - Monitoring vs. debugging, metric events (such as `PERF_METRICS`) are very useful
- Better identification of true costs
  - Hard with massive speculative execution
- Overhead management
  - More complex data = more processing = more overhead
- Assigning blame to jobs
  - Challenging for any offcore micro-architectural feature (such as L3 cache)

Linux PMU support challenges

- Bigger Machines = bigger pressure on the monitoring subsystem
  - AMD Zen2 : 256 CPUs!
- At scale, many rare corner cases become visible quickly
- Serious scalability issues in the `perf_events` interface and implementation
  - File descriptors, memory footprint, algorithmic complexity
- Perf tool scalability issues
  - Parsing of `/proc/PID/maps` very racy and can generate sampe symbolization issues
  - BuildID collection very expensive in CPU and memory footprint
  - Single threaded processing, single output file on 256 CPUs!
- Perf tool robustness
  - Many sanitizer failures
- Patches are being submitted to address these challenges
  - See our Linux Plumbers Conference'19 presentation on this topic here

Perf_events scalability example: file descriptors

- Large number of file descriptors (fds): 1 fd/event/cpu/cgroup
    - 100 cgroups, SkylakeX (112 CPUs), 6 events/cgroup = 112 x 100 x 6 = **67,200** fds
    - 200 cgroups, AMD Zen2 (256 CPUs), 6 events/cgroup = 200 x 256 x 6 = **307,200** fds
- Large number of events per-cpu:
    - 100 cgroups, SkylakeX (112 CPUs), 6 events/cgroup = 100 x 6 = **600** events/CPU
    - 200 cgroups. AMD Zen2 (256 CPUs), 6 events/cgroup = 256 x 6 = **1536** events/CPU

Structure names	Size (bytes)	Intel SkylakeX Total size (bytes)	AMD Rome Total size (bytes)
`struct file`	256	17MB	78MB
`struct perf_event`	1136	76MB	348MB
`Source: Linux-5.3-rc3, pahole`	TOTAL	93MB	427MB
	4KB Pages	22,705	104,400

Final thoughts

- PMU is now a critical part of the any CPU package
- Overall PMU features and quality have improved over the last 10 years
- CacheQoS is another important CPU feature required for data-centers
- Tools have improved but still lag behind hardware too often
- Scalability of tools and kernel monitoring subsystem is a challenge today

THANK YOU