在虚拟化场景中构建基于硬件的性能监控服务

Like Xu
腾讯云虚拟化开源团队

CLK 2021

Agenda

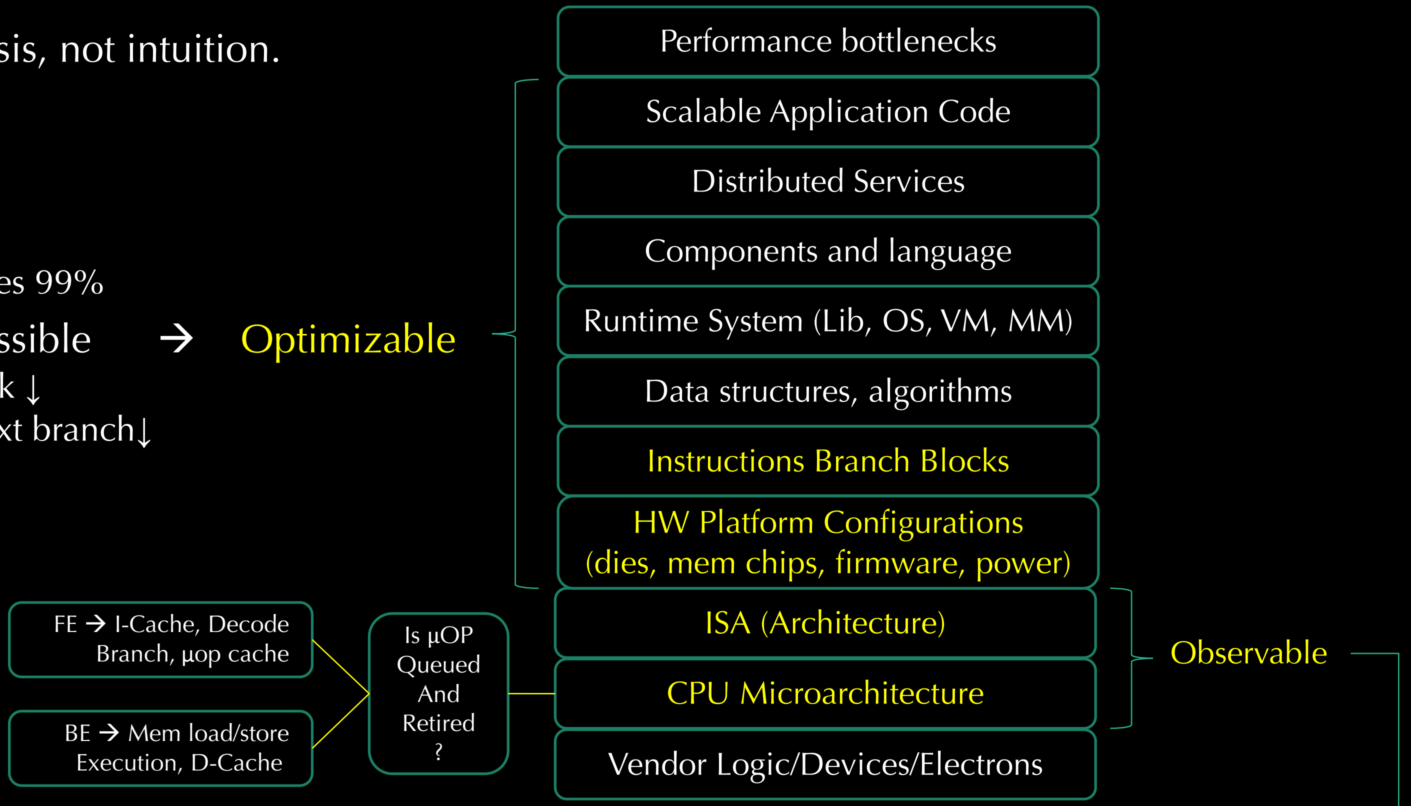
1. A basic mindset of CPU performance analysis
2. Perf profiling modes and its subcommands roadmap
3. Current publicly available hardware capabilities on x86
    1. The PMC workflow and some devilish details
    2. Virtualizing PMC and the basic KVM framework
    3. Virtualizing Branches Sampling Facilities
    4. Virtualizing Instructions Trace Facilities
4. Challenges in the ongoing hybrid scenarios
5. (Virtualized) PMU Use Cases at Tencent Cloud

1. A basic mindset of CPU performance analysis

- Optimization is driven by careful performance analysis, not intuition.
  - Full stack, short board, joggle, spikes, long tail
  - Telemetry/APM agent → Please fix other issues first
- How can we further accelerate our code ?
  - HW utilization and saturation, and also errors.
  - even if the CPU occupancy (not real utilization) reaches 99%
- Increase as many Instructions per cycle (**IPC**) ↑ as possible → Optimizable
  - elapsed time ↓, retired instructions per functional block ↓
  - real walked cycles (not crystal clock/tsc) before the next branch↓
  - cache-references ↑ + cache-misses ↓
    - cache (I, D, TLB) ↑ and memory latency ↓ hierarchy
    - branch-instructions ↓ + branch-misses ↓
    - frontend stalled cycles ↓
  - utilization of execution units ↑
    - backend stalled cycles ↓
    - execution units and types (Int, FP, Scalar, Vector) ↑
  - More CPU HW details
    - Out of order Window, Scheduler Entries …
    - Register Files, Allocation Queue …
    - Smarter speculation/prefetch … algorithms
    - Cstate change, uncore iio/imc …
    - check https://perfmon-events.intel.com/
- Workload Classification :: Top-down Microarchitecture Analysis (not for CLK)
  - http://www.cs.technion.ac.il/~erangi/TMA_using_Linux_perf__Ahmad_Yasin.pdf
  - keep performance in mind as you make early design and architectural decisions

Architecture HW events

Model Specified HW events


Performance bottlenecks
Scalable Application Code
Distributed Services
Components and language
Runtime System (Lib, OS, VM, MM)
Data structures, algorithms
Instructions Branch Blocks
HW Platform Configurations (dies, mem chips, firmware, power)
ISA (Architecture)
CPU Microarchitecture
Vendor Logic/Devices/Electrons
Optimizable
Observable
FE → I-Cache, Decode Branch, µop cache
BE → Mem load/store Execution, D-Cache
Is µOP Queued And Retired ?

Monitoring interesting event with counter (PMC) →

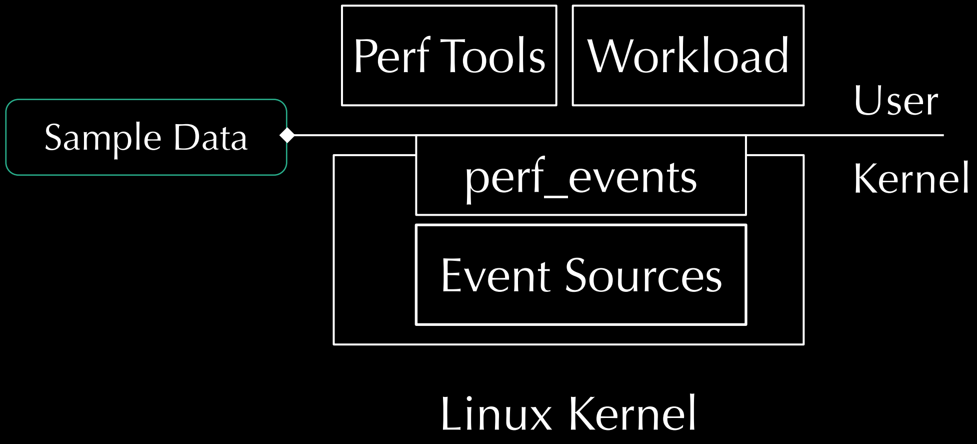
How the instructions flow reaches this event →

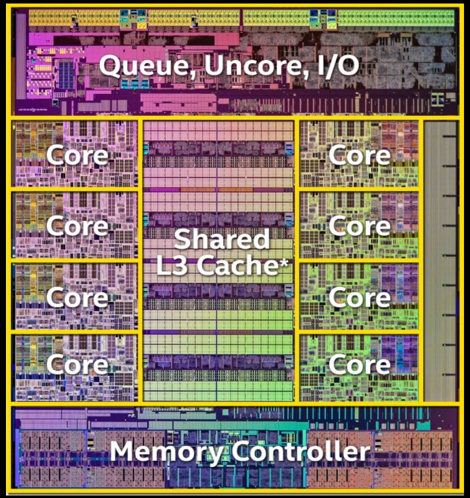
Monitoring hardware with hardware (PMU) →

Next Page : Perf - Linux Kernel official performance analyzing tool ←

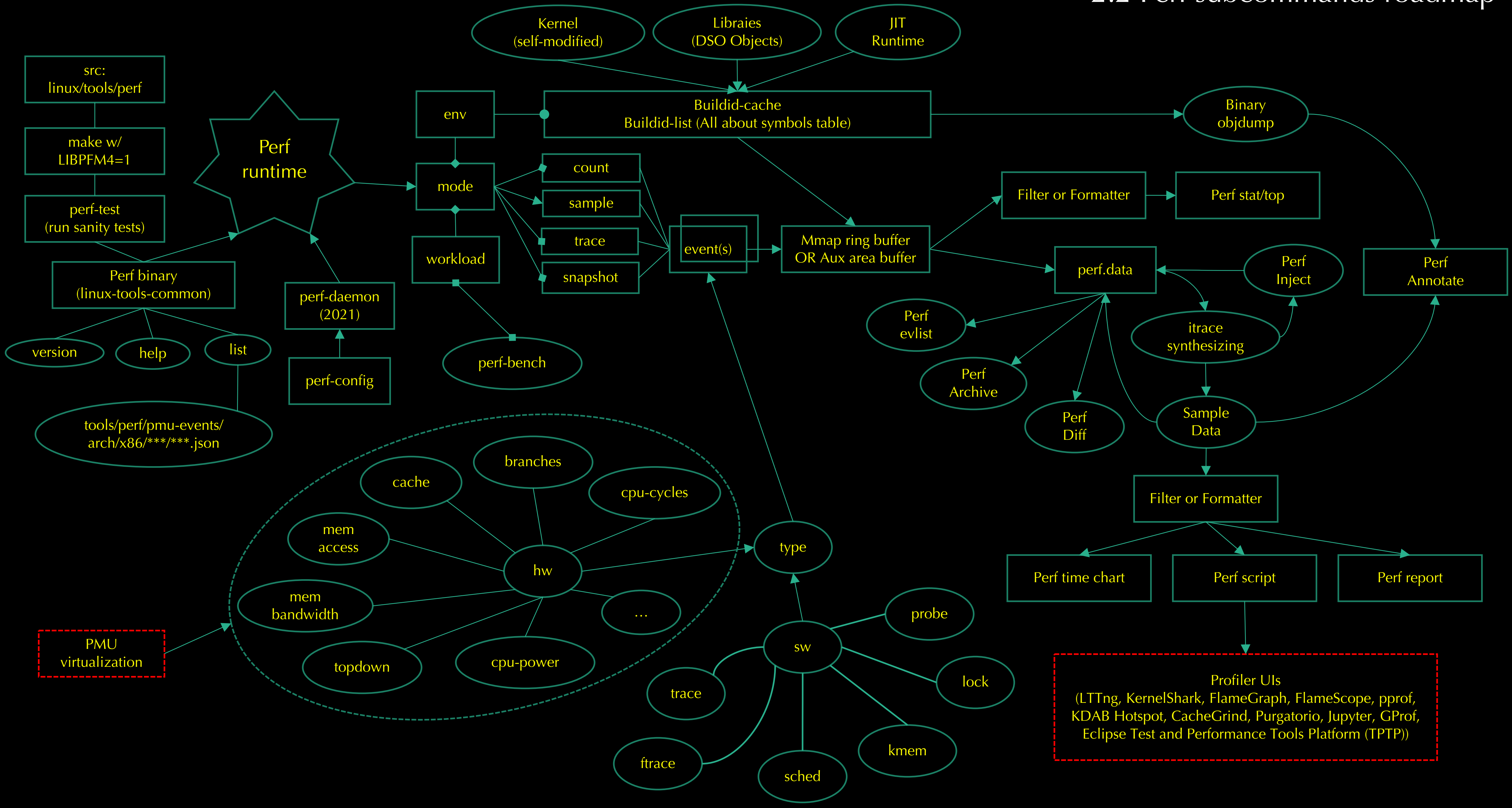
2.1 Perf profiling modes

- Profiling Target
  - system wide, per-user/numa/cpu/thread/
  - Namespaces/cgroup/dso(dynamic shared object (DSO))/
  - command/guest, exclude-perf
- Profiling Lifecycle :: Count -> Sampling -> Snapshot -> Trace -> Debug -> Count
- Count mode
  - `$ perf stat --interval-clear -I 1000 --metrics IPC -C 0`
  - `$ perf stat --topdown --td-level=2 --no-metric-only`
- Periodic Sampling mode --> `$ perf record …`
  - based on the occurrence of a particular event such as a timer or interrupt
  - Record w/ -F, -c,
  - `$ perf top -e 'cache-misses,cycles' -v --force -F 49 --realtime=1 -u root \`
    `-s overhead,comm --call-graph lbr --no-children --percentage relative --show-nr-samples`
- Full-trace mode
  - Get all execution records
  - Software trace
    - `$ perf trace/ftrace/probe/kmem/lock/sched/kvm …`
  - Hardware trace (more on next slide)
    - Intel branch trace store/Processor Trace/LBR call-stack mode
- Snapshot mode :: let trace run and overwrite older data in the buffer
  - Run ring buffer, stop trace on event of interest, save only tail of trace
  - `$ perf record -v -e intel_pt//u -S ./loopy 1000000000 &`
  - `$ kill -USR2 11435` # Recording AUX area tracing snapshot
- Mixed mode (such as, count + sample)
  - `$ perf stat record …`





2.2 Perf subcommands roadmap



2.3 Sample Data Fields

```
$ perf script -F comm,tid,pid,cpu,time,event,ip,dso,sym,symoff,srcline,period,insnlen,insn,misc
```

```
  CPU 115/KVM 466675/466796 [076] K     106883.295520:      35940   cycles:  ffffffff81e31d40 vmx_vmexit+0x0 (/lib/modules/5.15.0-rc2/build/vmlinux)
vmenter.S:79 ilen: 2 insn: eb 30
```

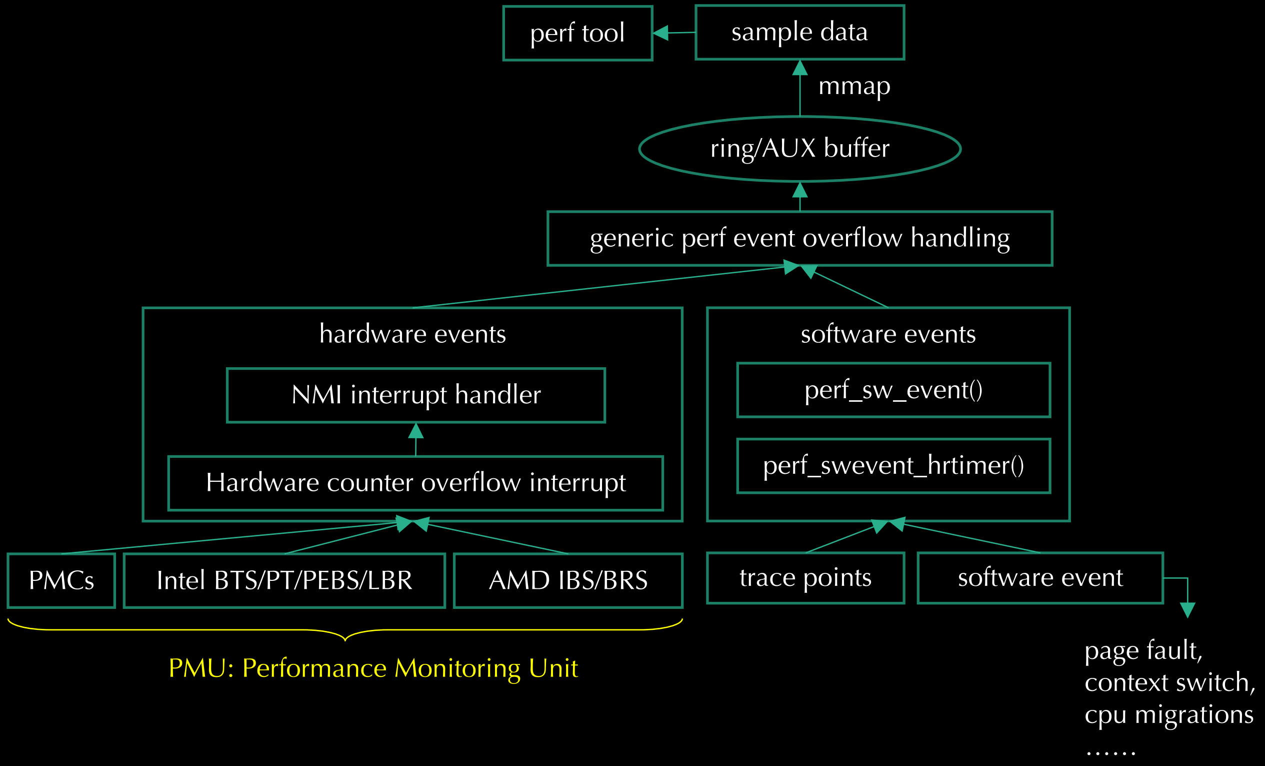
```
                                                        comm,tid,pid,cpu,time,event,ip,dso,sym,symoff,srcline,period,insnlen,insn,misc
     branches:   tr strt                                      flags ,ipc,callindent # -e intel_pt//, --itrace=cr -F +callindent
     branches:   call        IPC: 0.07 (124/1637)
     branches:   tr strt     IPC: 0.03 (17/475)                                                   addr # sample address, --data
     branches:   tr strt                                                                  tod # --clockid CLOCK_MONOTONIC
                                    ABI:2    AX:0xff    BX:0xff11003f772137e0
 lock_acquire                       ABI:2    AX:0xff    BX:0xff11003f772137e0                                iregs, # --intr-regs
    rcu_read_lock_sched_held        ABI:2    AX:0xff    BX:0xff11003f772137e0                                uregs, # --user-regs
       __fentry__                                      2M       data_page_size, # -e "mem-loads,mem-stores" --data-page-size
       __mem_cgroup_uncharge_list                      1024M    code_page_size  # -e "mem-loads,mem-stores" --code-page-size
          uncharge_page                                4K                  phys_addr # -e "mem-loads,mem-stores" --phys-data
             __fentry__
             release_pages                   ffffffff811f8267      10e5d0e9a                                   brstack # -b
                free_unref_page_list         ffffffff811d3003      3f77fedc08                        brstacksym,brstackoff # -b
                   __fentry__                ffffffff811d3031      3f7f7ed18c                                  brstackinsn # -b
                   tlb_flush_mmu
                      tlb_flush_mmu      perf_event_exec+308:
                      tlb_finish_mmu     ffffffff81359254       insn: 48 8b 7c 24 20
                      __vma_adjust       ffffffff81359259       insn: e8 e2 fc ae 00            # PRED 9 cycles [260] 0.22 IPC
```

2.4 More lesser known tips on perf usages
(a.k.a, perf tips Brendan Gregg won't tell you)

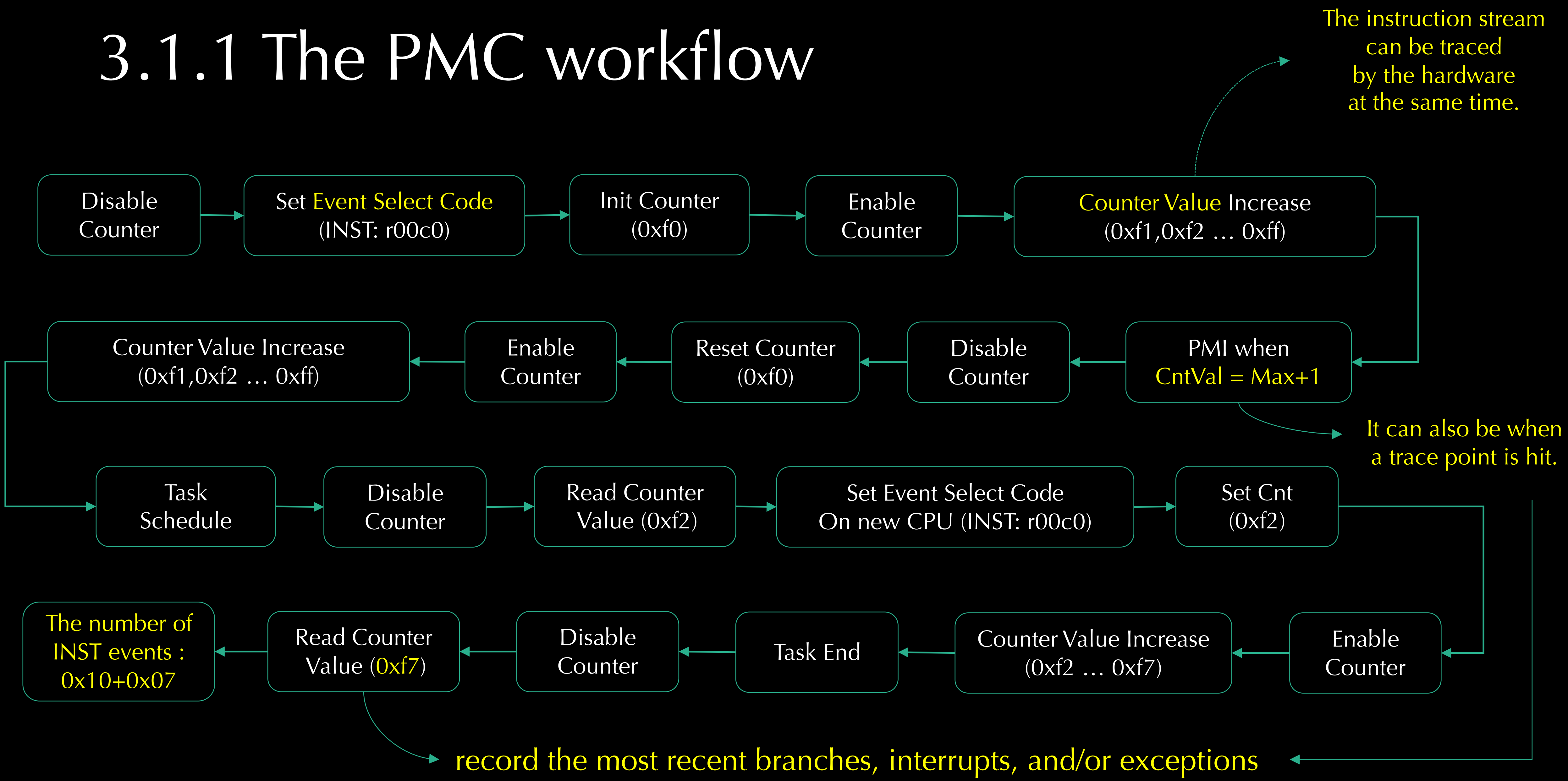
- How to build Perf in the best way
- How to run sanity tests for perf
- How to list all events
  - $ perf list --desc --long-desc --details (--raw-dump)
- Event switch-on/off tracing mode
- How to use uncore hardware events
  - $ perf iostat '0000:16,0000:20' -I 1000
- How to detect cache line false sharing w/ perf-c2c
- What is perf buildid & buildid-cache
- Many ways to program events
- Profile using event groups sampling
- What is advanced metric
  - $ perf list metric
- How to save copies of /proc/kcore, /proc/kallsyms and /proc/modules
- How to control mmap buffer behavior and the size of output perf.data
  - --no-buffering, --mmap-pages, --mmap-flush, --overwrite
- How to dump event raw sample/trace
- How to check each trace/sample
- How to synthesize samples from traces
- How to understood perf-annotate
- How to run perf session on background
- Tips to reduce perf record overhead
  - Multi AIO trace writing
  - Set affinity mask of trace reading thread
  - Produce compressed trace
  - --proc-map-timeout
- Perf built-in benchmark for kernel
- How to use perf-trace/perf-ftrace/perf-kmem/perf-lock/perf-kmem/perf-sched/perf-config …
- No time to cover all this time but the slide is already available at *https://github.com/xuliker/kde/raw/master/likexu_slides/Deep%20dive%20into%20Linux%20perf%20tool.pdf*

3. Current publicly available hardware capabilities on x86

- Why do we need hardware assistance ?
  - Achieving low cost and high resolution profiling is an art.
  - Do not require recompilation, restart, or
    embedded instrumentation (though support it if present).
  - Run fast, non-intrusively,
    to minimize overhead and gross disruption
  - Measure really granular things
- Performance Monitoring Counters (Available)
  - Architecture events
  - Specific Topdown/PERF_METRICS counter on Intel
- Intel features (Upstream ed-or-ing)
  - Last Branch Recording ((Arch, Timed) LBR)
  - Processor Event Based Sampling ((Extended, Adaptative) PEBS)
  - Branch Trace Store (BTS)
  - Processor Trace (PT)
- AMD features (Under Development)
  - Instruction-Based Sampling (IBS)
  - Branch Sampling Feature (BRS)
  - Lightweight Profiling (LWP, for user space)
- More and more PMU hw features will be available.
  - Some inspiration for RISC-V design

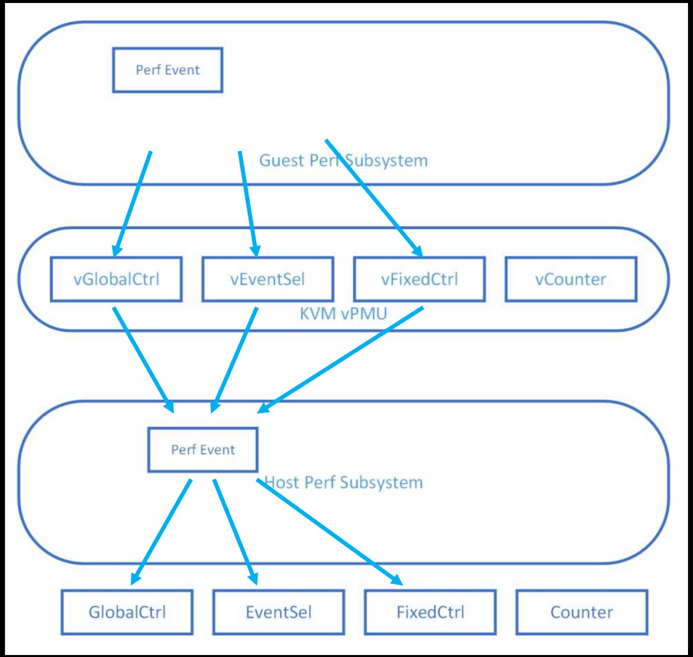


3.1.1 The PMC workflow


The instruction stream can be traced by the hardware at the same time.
Disable Counter
Set Event Select Code (INST: r00c0)
Init Counter (0xf0)
Enable Counter
Counter Value Increase (0xf1,0xf2 … 0xff)
PMI when CntVal = Max+1
It can also be when a trace point is hit.
Disable Counter
Reset Counter (0xf0)
Enable Counter
Counter Value Increase (0xf1,0xf2 … 0xff)
Task Schedule
Disable Counter
Read Counter Value (0xf2)
Set Event Select Code On new CPU (INST: r00c0)
Set Cnt (0xf2)
Enable Counter
Counter Value Increase (0xf2 … 0xf7)
Task End
Disable Counter
Read Counter Value (0xf7)
The number of INST events : 0x10+0x07
record the most recent branches, interrupts, and/or exceptions

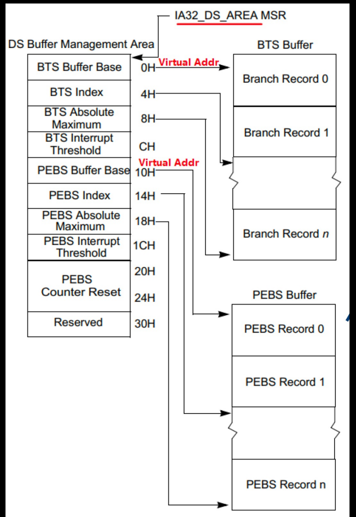
3.2 Virtualizing PMC and the basic KVM framework

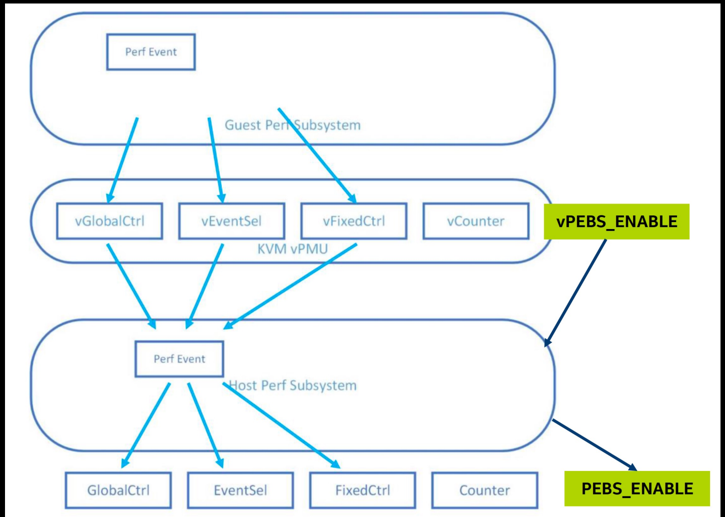
- $ perf record -e cpu/event=0xa8,umask=0x1,cmask=0x1/ ...
- Emulating guest MSR-based counter resource
  - Event Code = Umask + EvtSel (per logical core)
  - RD/WR Event counter value
- Emulating guest counter enable
  - and disable behavior
- Emulating PMI for guest
- Emulating RDPMC for guest
  - Read Event Counter Value
- How to count for guess instruction events ?
- The scheduling of counters for guests emulation and host itself.



3.2 Virtualizing PEBS and the upstream story

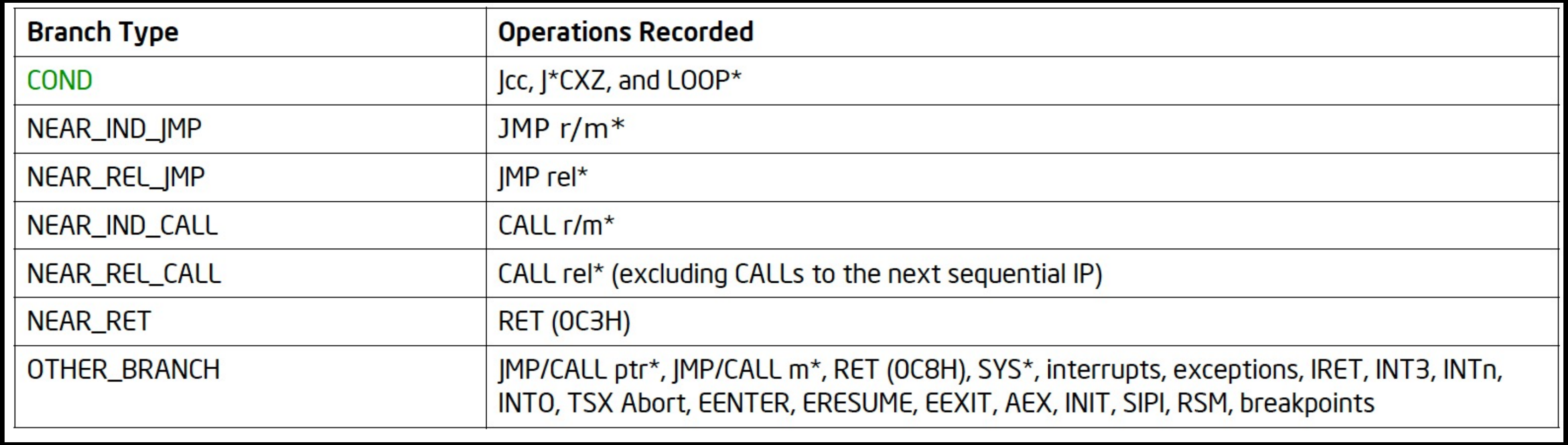
- $ perf mem --data-page-size record -e instructions:P …
- After the counter overflows,
  - the processor copies the current state of the general-purpose and EFLAGS registers and instruction pointer into a record in the precise event records buffer.
- The debug store (DS) buffer
- When the precise event records buffer is nearly full, an interrupt is generated





3.3 Virtualizing Branches Sampling Facilities

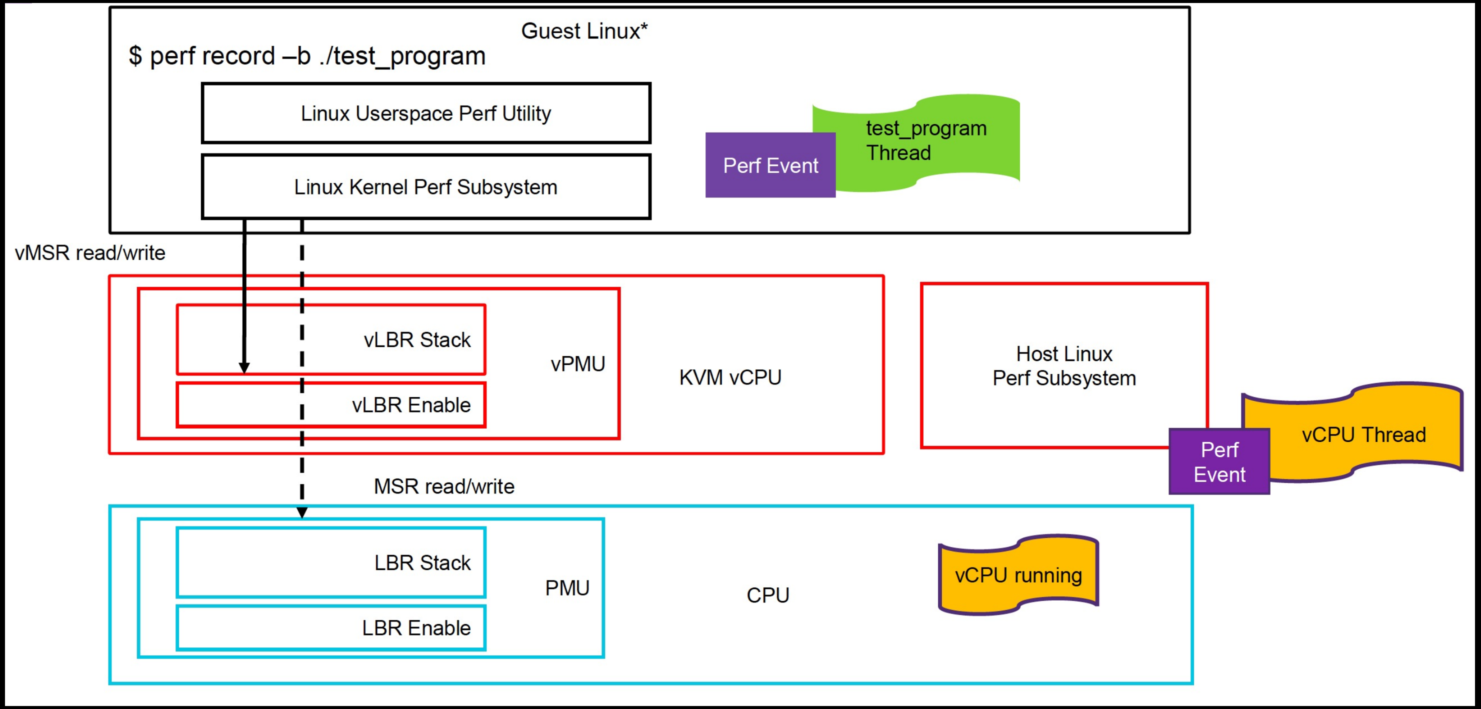
- $ perf record -g ... OR perf record --call-graph lbr ...
- logging taken branches and control flow transfers within registers.
  - Intel LBR vs AMD BRS (Zen 3,19h)
    - Diff on lbr_depth, enable after counter overflow, support call_stack mode, hardware branch type filtering, fixed sampling period
  - enable autoFDO-style optimization by compilers
  - doesn't support LER (Last Event Record) due to security leakage



Branch Type	**Operations Recorded**
COND	Jcc, J*CXZ, and LOOP*
NEAR_IND_JMP	JMP r/m*
NEAR_REL_JMP	JMP rel*
NEAR_IND_CALL	CALL r/m*
NEAR_REL_CALL	CALL rel* (excluding CALLs to the next sequential IP)
NEAR_RET	RET (0C3H)
OTHER_BRANCH	JMP/CALL ptr*, JMP/CALL m*, RET (0C8H), SYS*, interrupts, exceptions, IRET, INT3, INTn, INTO, TSX Abort, EENTER, ERESUME, EEXIT, AEX, INIT, SIPI, RSM, breakpoints

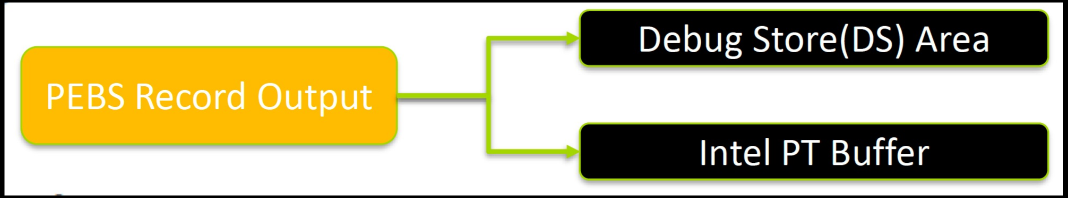
3.3 Virtualizing Branches Sampling Facilities

- Enabled via IA32_LBR_CTL or DEBUGCTRL MSR
  - VMX entry-load/exit-clear control
- What if host/guest branch record filters options are different
- LBR MSRs pass-through
- XSAVES/RESTORES
  - Speed up LBR read/reset
  - For Arch LBR



3.4 Virtualizing Instructions Trace Facilities

- Branch trace store (BTS) → just trace all branches taken
  - $ perf record --per-thread -e intel_bts// …
- Intel PT :: log information about software execution
  - $ perf record -e intel_pt// …
  - Supports control flow tracing
    - determine exact flow of software execution from trace log
  - with minimal impact to system execution (according to Intel)
    - Target <5% performance overhead
  - store both cycle count and timestamp information
  - Gdb 7.10 supports PT for "backwards debugging" (reverse-step)
  - ARM Coresight
- How to virtualizing PT
  - Intel PT VMX improvements will treat PT output addresses as Guest Physical Addresses (GPAs) and translate them using EPT that serves to simplify the process of Intel PT virtualization for using by a guest software.



4. Challenges in the ongoing hybrid scenarios

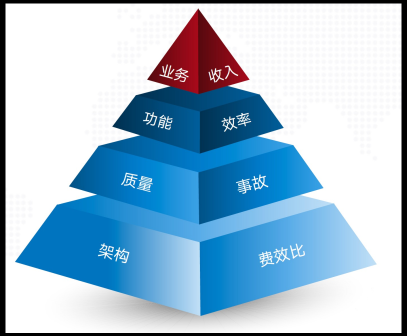
- Performance is a stabilizing feature and of its own right.
    - Performance on the Horizon
        - Zoom In/out
    - We need to satisfy any perf user in any profiling environment.
- Guest user need self-profiling its workload without any risk
    - How to keep both guest and host accurate
    - How to reduce guest and host profiling overhead
- For bare metal, system-wide profiling
    - Kernel space, user space, SMM space, even firmware
    - The tracking framework of the user space is quite colorful
- For CSP
    - Profiling virtual machine(s) and host at the same time
    - Collaboration and integration of guest kernels and hypervisor
    - $ perf kvm --host --guest stat/record …
- For cloud native
    - service layer framework && business customized code
- In the near future
    - TEE (trusted execution environment) code && Non-TEE code
    - Dance with heterogeneous profiling systems facilities
    - eBPF program may need to touch PMU hardware
    - Heisenberg, aka observer effect

5. (Virtualized) PMU Use Cases at Tencent Cloud

- Dedicated Infrastructure Performance Optimization (not SRE) teams
- Performance debugging and code optimization for various languages
    - IPC ↑ forever, hot/critical path, frame pointerless stacktrace/callgraphs
    - Continuous compiler feedback, machine code layout optimizations
    - Garbage collection latency (JVMs, with GC cycles), lockless algorithms, zero-copy optimizer
    - TSX lock elision, deadlock, cache line false sharing
- Performance characterization for cloud workloads
    - NUMA/NVDIMM cold/hot page migration policy
    - Detect whether a virtual machine is live migration friendly
- Running perf as an part of program (libperf, perfmon2-libpfm4, eBPF)
- One of telemetry sources, runtime remotely

    triggering/configuration/instrumentation/collection/visualization

- Subscribe mode for uncore events
    - ask profiling data from various agents
- Performance Isolation Verification (such as RDT/cgroup v2)
- Observing the CPU health of both host and services
    - Add performance regression report to the DevOps pipeline, sub-second level of resolution
    - Security Intrusion (DDoS) Detection
    - Avoid causing the application to violate deadlines/SLA/QoS
- Analyze past, debug present, monitor future
    - Add performance regression to cover all vendor code changes
    - Kernel, Compilers, Retpoline/trampoline, Hotfixes, QEMU, Unikernel, virtio-fs, OCI runtime, block layer



Q & A

Thank you to join me at CLK 2021!

Error correction will be placed here.

Like Xu