Linux perf_events status update

Stephane Eranian
Google

Petascale Tools Workshop 2014

Agenda

- new features, updates
- upcoming features
- use case
- Q&A

Miscellaneous progress

- Intel official event tables available online now!
  - https://download.01.org/perfmon/
  - Andi Kleen’s patches to use symbolic event names with perf

- IBM Power 8 branch stack sampling patches under LKML review
  - similar to Intel LBR sampling capabilities
  - seamless integration under perf_events branch stack abstraction

- Intel Haswell LBR call-stack patches under LKML review
  - LBR push/pop to collect call stack statistically (last 16 calls)
  - better call stack unwinding support: no framepointer, no dwarf

- Ability to sample interrupted machine state under LKML review
  - and includes the PEBS machine state in precise mode

- Intel IvyTown uncore PMU support since Linux 3.12

perf: monitoring power consumption (RAPL)

- Intel Running Average Power Limit (RAPL) counters
  - power limiting, energy consumption in Joules
  - available in SNB*, IVB*, HSW*
  - consumption also reported by `turbostat` tool

- Integration in perf_events with Linux 3.14
  - new separate uncore PMU: power
  - system-wide mode counting only
  - package-level consumption only
  - new events: `power/energy-cores/`, `power/energy-pkg/`, `power/energy-dram/`, `power/energy-gpu/`

```
# perf stat -a -e power/energy-cores/,power/energy-pkg/ -I 1000 sleep 10
#           time             counts   unit events
     1.000119482               7.72 Joules power/energy-cores/
     1.000119482              12.67 Joules power/energy-pkg/
```

perf: measuring memory bandwidth on client CPU

- Intel X86 client processors only (SNB/IVB/HSW)
  - using integrated memory controller (IMC)
  - PCI space, free running counters

- Integration in perf_events with Linux 3.15
  - separate uncore PMU: `uncore_imc`
  - system-wide, counting mode only
  - two events: `uncore_imc/data_reads/, uncore_imc/data_writes/`
  - counting full cache-line accesses only

```
# perf stat -a -e uncore_imc/data_reads/,uncore_imc/data_writes/ -I 1000 sleep 2
#           time             counts unit events
     1.000181288           13442.16 MiB  uncore_imc/data_reads/
     1.000181288            4469.58 MiB  uncore_imc/data_writes/
     2.000418548           13442.89 MiB  uncore_imc/data_reads/
     2.000418548            4469.79 MiB  uncore_imc/data_writes/
```

Hyperthreading counter corruption bug

2013 slide

- Measuring memory events may corrupt events on sibling thread

```
MEM_LOAD_UOPS_RETIRED.*, MEM_UOPS_RETIRED.*
MEM_LOAD_UOPS_LLC_HIT_RETIRED.*
MEM_LOAD_UOPS_LLC_MISS_RETIRED.*
Example:
THREAD0: counter0=MEM_LOAD_UOPS_RETIRED:L3_MISS
THREAD1: counter0 may be corrupted regardless of measured event
```

- Impacted CPUs: SNB*, IVB*, HSW*

- No workaround in firmware
  - disable HT or measure only one thread/core (but clashes with NMI watchdog)

- Linux 3.11
  - blacklisting events on IVB even if HT is off (may add SNB, HSW soon)

- Google working on modifications to event scheduler
  - enforce mutual exclusion on sibling counters when corrupting events used

HT bug: Google workaround eliminates corruption

- Posted kernel patch series to eliminate corruption
  - still under LKML review
  - developed by M. Dimakopoulou (Google intern in Paris)

- Enforce mutual exclusion between HT at counter granularity
  - uses cache-coherency style protocol: Shared, Exclusive, Unused
  - leverages built-in event scheduler
  - adds dynamic event constraints based on sibling thread state

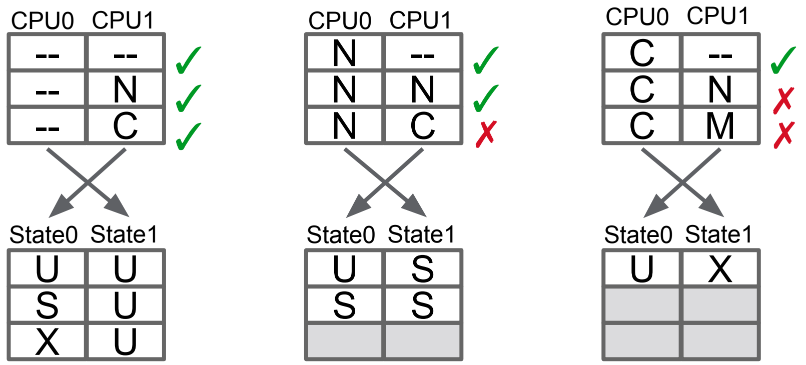
- No modifications to user tools or machine config

- All events can be measured safely

- Current limitations (work-in-progress):
  - no re-integration of leaked counts (can be huge > 3x)
  - PMU starvation: some events never scheduled because of other HT

HT bug: XSU protocol

- Events
  - Non-Corrupting (N)
  - Corrupting (C)
- Counter States
  - Xclusive (X)
  - Shared (S)
  - Unused (U)



- Principles
  - event scheduling on one HT affects the state of the other HT
  - **C** events → allowed on counters only with **U** state
  - **N** events → allowed on counters only with **U or S** state

upcoming features

perf tool: profiling jitted code

- Many runtimes use jit-in-time (JIT) compilation
  - openJDK Java, V8, DART, ….
- `perf report` very limited support for symbolizing jitted code
  - runtime emits /tmp/perf-PID.map file: `addr, size, symbol`
  - no support for assembly view
  - no support for jit code cache reuse

perf tool: current situation with OpenJDK Java

```
$ perf record java jnt/scimark/commandline

# Samples: 125K of event 'cycles'
# Event count (approx.): 102160463028
#
#   Ovh  Cmd   ShObj                Symbol
# .....  ...   ...............      ...............
  2.16%  java  perf-17584.map       [.] 0x00007fed17fdb9fd
  2.13%  java  perf-17584.map       [.] 0x00007fed17fdb9f9
  2.00%  java  perf-17584.map       [.] 0x00007fed17fdf3ab
  1.98%  java  perf-17584.map       [.] 0x00007fed17fdb9ca
  1.76%  java  perf-17584.map       [.] 0x00007fed17fdf395
  1.68%  java  perf-17584.map       [.] 0x00007fed17fddfed
  1.51%  java  perf-17584.map       [.] 0x00007fed17fd7dfe
  1.49%  java  perf-17584.map       [.] 0x00007fed17fde058
  1.45%  java  perf-17584.map       [.] 0x00007fed17fde029
     …
  0.01%  java  libjvm.so            [.] PhaseLive::compute(unsigned int)
  0.01%  java  perf-17584.map       [.] 0x00007fed17f94a3c
```

perf-PID.map is not emitted by runtime, no symbolization

perf tool: Google adding full jitted code support

- Cooperation from runtime mandatory
  - must emit function mappings
  - must emit assembly code
  - must emit source line information
  - emitted info must be timestamped to correlate with samples
  - emitted file format must be runtime and arch agnostic

- Timestamps synchronized with perf_events timestamps
  - perf_events uses `sched_clock()` which is not exposed to users
  - using POSIX dynamic clocks to expose a `sched_clock()` to user

- No modification to `perf_events` kernel subsystem

- Minimize changes to `perf` tool
  - no changes to `report` and `annotate` commands

- Similar approach used by OProfile

perf tool: full jit code support example

```
$ perf record java -agentpath:libjvmti.so jnt/scimark/commandline
$ perf inject -i perf.data -o perf.data.j -j ~/.debug/jit/XXqw/jit-1815.dump
$ perf report -i perf.data.j
```

```
# Samples: 124K of event 'cycles'
# Event count (approx.): 101762443128
#
#    Ovh Cmd  ShObj       Symbol
#  ..... ...  ..........  ........
#
  23.38% java j-1815-245 void class jnt.scimark2.SparseCompRow.matmult(double[], double[], int[],double[])
  18.96% java j-1815-231 void class jnt.scimark2.FFT.transform_internal(double[], int)
  17.99% java j-1815-241 void class jnt.scimark2.SOR.execute(double, double[][], int)
  17.94% java j-1815-250 int class jnt.scimark2.LU.factor(double[][], int[])
  17.89% java j-1815-243 double class jnt.scimark2.MonteCarlo.integrate(int)
   2.03% java j-1815-230 void class jnt.scimark2.FFT.bitreverse(double[])
   0.27% java j-1815-251 double class jnt.scimark2.kernel.measureLU(int, double, class jnt.scimark2.Random)
   0.22% java j-1815-18  Interpreter
   0.22% java j-1815-248 void class jnt.scimark2.kernel.CopyMatrix(double[][], double[][])
```

perf tool: jit code assembly view

```
$ perf annotate -i perf.data.j
void class jnt.scimark2.SparseCompRow.matmult(double[], double[], int[], int[], double[], int)
  Ovh%
  . . .
  2,64 |13e:   cmp    %ecx,%r10d
  1,84 |141: ┌──jge    1d2 <Ljnt/scimark2/SparseCompRow;matmult([D[D[I[I[DI)V+0x1d2>
       |147: │  data32 xchg %ax,%ax
  2,55 |14a: │  mov    0x10(%r8,%r10,4),%ebp
  0,00 |14f: │  cmp    %esi,%ebp
  1,81 |151: │  jae    22d <Ljnt/scimark2/SparseCompRow;matmult([D[D[I[I[DI)V+0x22d>
       |157: │  vmovsd 0x10(%rdx,%r10,8),%xmm1
  2,78 |15e: │  vmulsd 0x10(%r9,%rbp,8),%xmm1,%xmm1
       |165: │  vaddsd %xmm0,%xmm1,%xmm0
  2,50 |169: │  movslq %r10d,%r14
  1,97 |16c: │  mov    0x14(%r8,%r14,4),%ebp
  2,07 |171: │  cmp    %esi,%ebp
  0,04 |173: │  jae    224 <Ljnt/scimark2/SparseCompRow;matmult([D[D[I[I[DI)V+0x224>
  1,58 |179: │  vmovsd 0x18(%rdx,%r14,8),%xmm1
  0,90 |180: │  vmulsd 0x10(%r9,%rbp,8),%xmm1,%xmm1
       |187: │  mov    0x18(%r8,%r14,4),%ebp
```

perf tool: cache line access analysis

- perf c2c: profile load/store, analyze accesses patterns
  - developed by Redhat
- using abstract load/store sampling feature of perf_events
  - leverages Intel SNB/IVB/HSW load latency, precise store sampling
- Very helpful to detect:
  - cache line false sharing
  - bad NUMA locality
- under LKML review

```
$ perf c2c record -a sleep 10
$ perf c2c report
```

perf c2c: demo

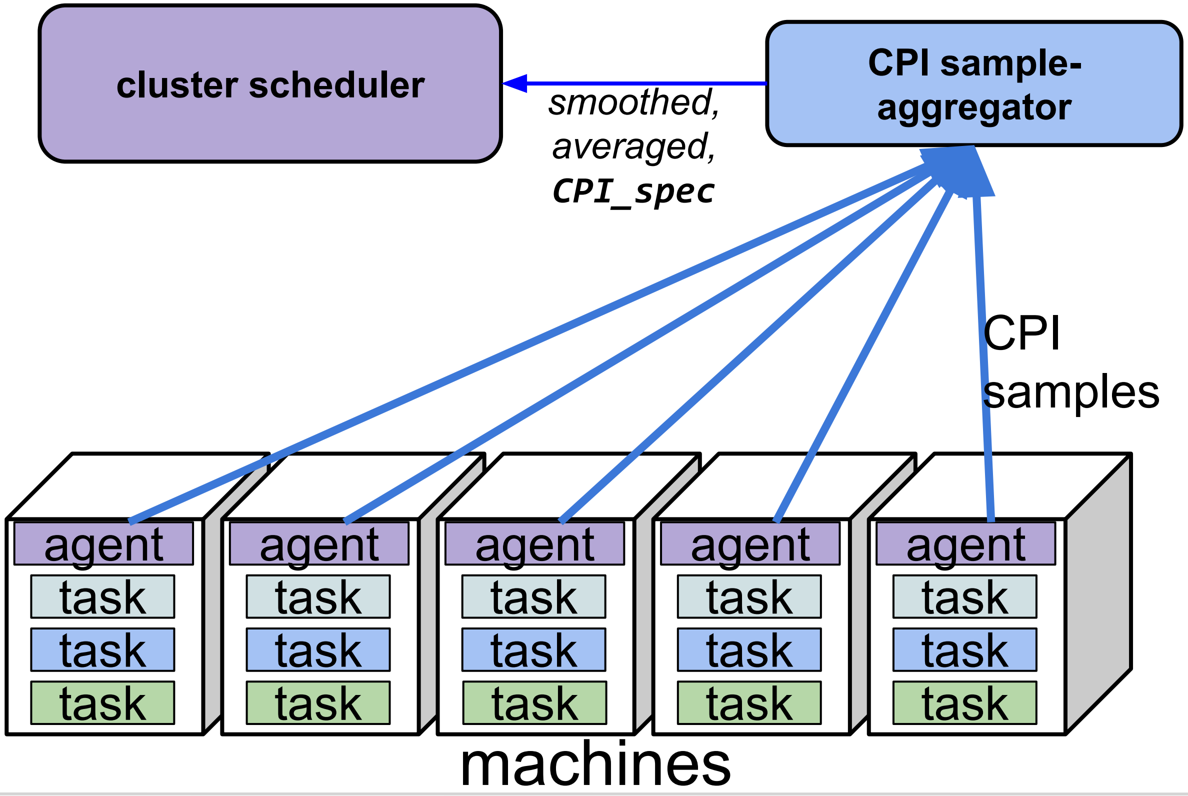
How does Google use all of this?

CPI^2: monitor CPI

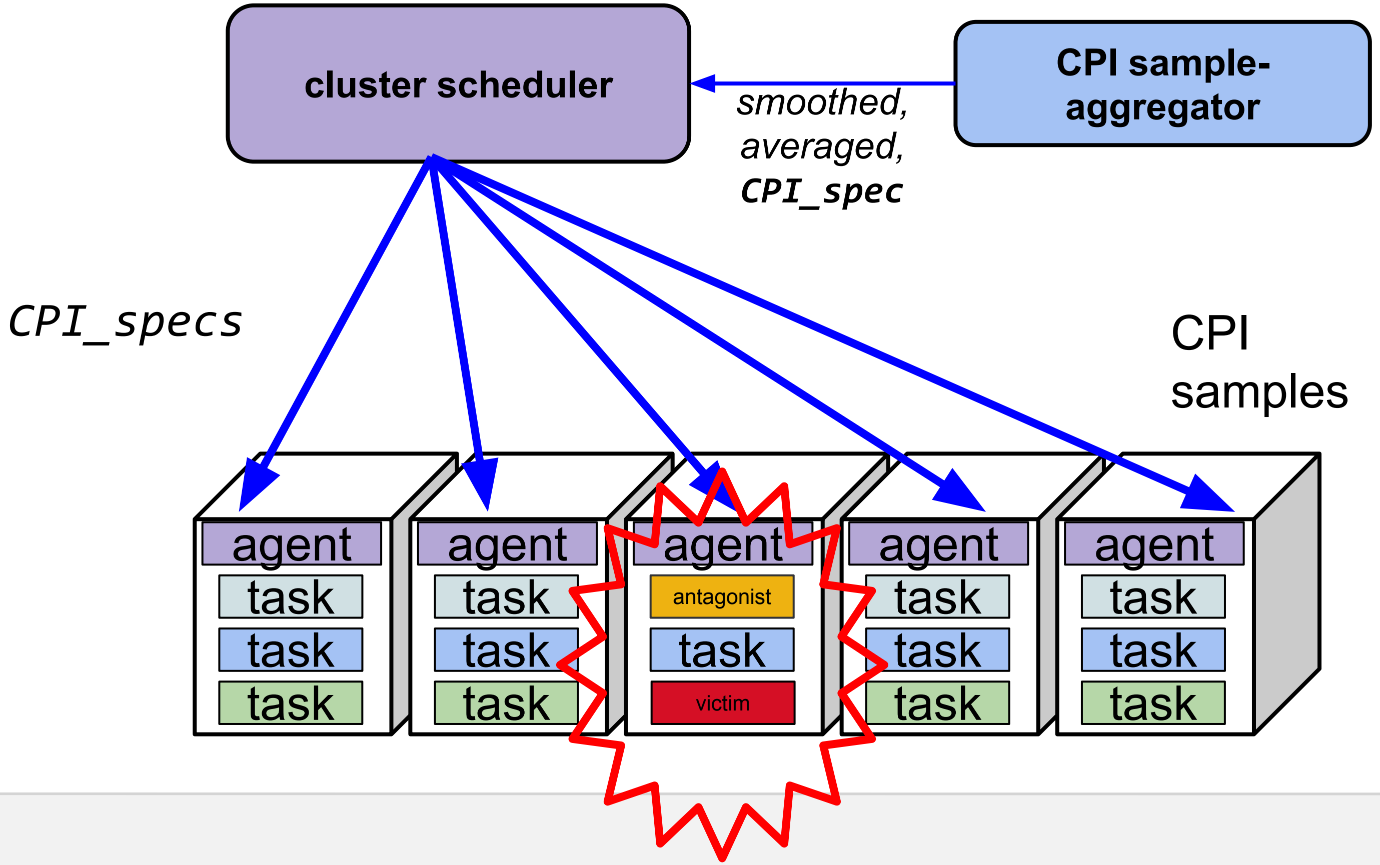
1. monitor Cycles Per Instruction (**CPI**)
2. learn normal and anomalous behaviors
3. identify a likely antagonist, and
4. throttle it to shield victims [optional]

- Experimental data shows CPI correlates well with
  - latency
  - throughput
- CPI is easy to collect with PMU
  - PMU events easily avail: `unhalted_ref_cycles`, `instructions_retired`

CPI^2: data collection architecture

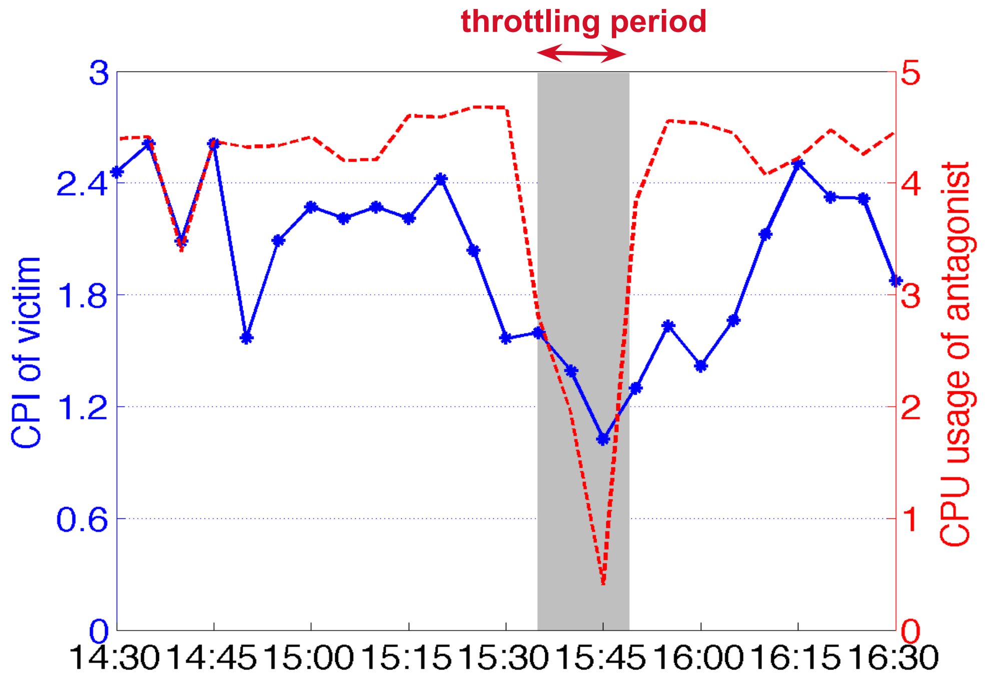


CPI^2: architecture



CPI^2: dealing with antagonist

- Examine time-correlation between victim's CPIs and (suspected) antagonist's CPU usages
- Highest positive correlation signals most likely culprit
- Antagonist throttled via CPU hard capping



CPI^2: results

- deployed to Google's fleet
- thousands of interference events/day
- simple and effective
- 37% CPI reduction
- only using very common PMU events in counting mode

Conclusion: we are almost there!

- All major processor architectures supported
- All key hardware features are supported now
  - Intel X86: core, uncore, PEBS, LBR, ld/st sampling, offcore_rsp, PT
  - Only incremental hardware improvements from now on
- Tool vendors adopting perf_events interface or tool
  - no more custom drivers
- Systematic and continuous profiling implemented
- Need more high-level metric tools
  - core PMU cycles breakdown, leverage uncore PMU

Looking for help!

- full-time software engineer to help with kernel and user tool infrastructure developments

- requirements:
  - experience with Performance Monitoring Unit (PMU) technology
  - experience with Linux kernel development
  - some experience with Javascript development
  - strong interest in the field

References

- HT counter corruption (SandyBridge: BJ122, IvyBridge: BV98, Haswell: HSD29)
  - http://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/3rd-gen-core-desktop-specification-update.pdf
- CPI^2: CPU performance isolation for shared compute clusters
  - Xiao Zhang, Eric Tune, Robert Hagmann, Rohit Jnagal, Vrigo Gokhale, John Wilkes
- Andi Kleen's patches for full symbolic event support in perf
- HT corruption workaround patches
- perf c2c LKML patches