Google

Linux perf_events status update

Stephane Eranian
Google

Petascale Tools Workshop 2016

Agenda

- New hardware support
- Advanced Skylake PMU features
- Jitted code support
- A little Quizz

New hardware support

- Intel Broadwell server (BDX) uncore in Linux v4.5
  - memory controller, power, qpi, pcie, …
- Intel Skylake client (SKL) in Linux v4.3
  - core PMU, precise frontend
  - power (RAPL) in v4.6
  - memory controller (IMC)
- Intel Xeon DE uncore PMU in Linux v4.3
- ARM64: ThunderX (v4.6),Vulcan (v4.7),Cortex A72 (v4.5), A57 (v4.4), A53 (v4.4)
- Intel Goldmont in Linux v4.7
- Intel Knights Landing (KNL) in v4.7
- Intel MSR driver (TSC, APERF, MPERF) in Linux v4.3

TSC, APERF, MPERF

- Provide a way to add free-running counters support
  - free-running: non-stop, no-interrupt, fixed register
- Patch adds TSC, APERF, MPERF
  - APERF:increments in proportion to actual performance
  - MPERF: increments in proportion to a fixed frequency
  - ratio APERF/MPERF architecturally defined
- New `msr` PMU with new events: `tsc, aperf, mperf`
  - no sampling, no vmm

```
$ perf stat -a -e msr/tsc/,msr/mperf/ -I 1000 sleep 10
#           time             counts unit events
    2.005199136    194,116,392,977      msr/tsc/    □ 49% idle (low power)
    2.005199136     98,771,528,325      msr/mperf/    (tsc - mperf)/tsc
```

SkyLake new features

- Last Branch Buffer (LBR) has 32 entries (2x Haswell)
- Timed LBR : basic block cycle duration
  - capture cycle duration between consecutive branches
  - LBR record: 3x `uint64_t` now (50% increase)
- TSC is captured by PEBS
- PEBS precise Front-End sampling
  - sample where I-TLB, I-CACHE misses occur

Timed LBR

```
$ perf record -b -e cpu/event=0xc0,umask=0x1/upp triad
$ perf report --no-branch-stack
```

histogram on interrupted IP not branch entries

Haswell

```
14.65 |401838:   movupd (%rbx,%r8,1),%xmm1
10.41 |40183e:   add    $0x1,%r9d
16.12 |401842:   mulpd  %xmm2,%xmm1
20.85 |401846:   addpd  (%r12,%r8,1),%xmm1
11.15 |40184c:   movups %xmm1,(%rax,%r8,1)
16.49 |401851:   add    $0x10,%r8
 0.00 |401855:   cmp    %r10d,%r9d
10.32 |401858: ↑ jb     401838
```

Skylake

```
10.26  0.50      |401838:   movupd (%rbx,%r8,1),%xmm1
29.94  0.50      |40183e:   add    $0x1,%r9d
15.97  0.50      |401842:   mulpd  %xmm2,%xmm1
14.96  0.50      |401846:   addpd  (%r12,%r8,1),%xmm1
 8.90  0.50      |40184c:   movups %xmm1,(%rax,%r8,1)
13.63  0.50      |401851:   add    $0x10,%r8
       0.50      |401855:   cmp    %r10d,%r9d
 6.35  0.50   16 |401858:   jb     401838
```

Average block latency (core cycles)

IPC = num_insn / block_latency = 8 / 16

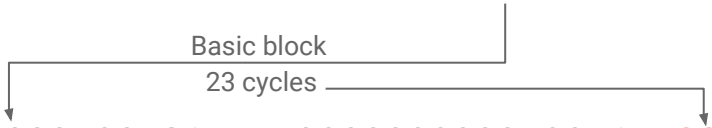
Time LBR raw data

- Best method:
  - Perf script -F brstack (Linux v4.4) is recommended for easier parsing

```
$ perf script -F ip,brstack
400f21 0x400f31/0x400f1a/P/-/-/2  0x400f31/0x400f1a/P/-/-/2  0x400f31/0x400f1a/P/-/-/2
```

  - Perf report -D

```
... branch stack: nr:32
.....  0: 0000000000400f31 -> 0000000000400f1a 10 cycles  P   0
.....  1: 0000000000400f31 -> 0000000000400f1a 23 cycles  P   0
.....  2: 0000000000400f31 -> 0000000000400f1a 2 cycles  P   0
.....  3: 0000000000400f31 -> 0000000000400f1a 3 cycles  P   0
```



Timed LBR & OO-Execution

```
10.26   0.50          |401838:   movupd (%rbx,%r8,1),%xmm1
29.94   0.50          |40183e:   add    $0x1,%r9d
15.97   0.50          |401842:   mulpd  %xmm2,%xmm1
14.96   0.50          |401846:   addpd  (%r12,%r8,1),%xmm1
 8.90   0.50          |40184c:   movups %xmm1,(%rax,%r8,1)
13.63   0.50          |401851:   add    $0x10,%r8
        0.50          |401855:   cmp    %r10d,%r9d
 6.35   0.50     16   |401858:   jb     401838
```

- Iterations executed out-of-order
- Iterations retire a couple cycles after each other
- Latency totally hidden by OO

```
... branch stack: nr:32
.....  0: 0000000000401858 -> 0000000000401838 4 cycles
.....  1: 0000000000401858 -> 0000000000401838 3 cycles
.....  2: 0000000000401858 -> 0000000000401838 245 cycles
.....  3: 0000000000401858 -> 0000000000401838 2 cycles
.....  4: 0000000000401858 -> 0000000000401838 6 cycles
.....  5: 0000000000401858 -> 0000000000401838 3 cycles
.....  6: 0000000000401858 -> 0000000000401838 14 cycles
.....  7: 0000000000401858 -> 0000000000401838 2 cycles
.....  8: 0000000000401858 -> 0000000000401838 6 cycles
.....  9: 0000000000401858 -> 0000000000401838 3 cycles
..... 10: 0000000000401858 -> 0000000000401838 31 cycles
..... 11: 0000000000401858 -> 0000000000401838 2 cycles
..... 12: 0000000000401858 -> 0000000000401838 2 cycles
..... 13: 0000000000401858 -> 0000000000401838 1 cycles
..... 14: 0000000000401858 -> 0000000000401838 2 cycles
..... 15: 0000000000401858 -> 0000000000401838 2 cycles
..... 16: 0000000000401858 -> 0000000000401838 2 cycles
..... 17: 0000000000401858 -> 0000000000401838 1 cycles
..... 18: 0000000000401858 -> 0000000000401838 2 cycles
..... 19: 0000000000401858 -> 0000000000401838 2 cycles
..... 20: 0000000000401858 -> 0000000000401838 4 cycles
..... 21: 0000000000401858 -> 0000000000401838 3 cycles
..... 22: 0000000000401858 -> 0000000000401838 28 cycles
..... 23: 0000000000401858 -> 0000000000401838 2 cycles
..... 24: 0000000000401858 -> 0000000000401838 2 cycles
..... 25: 0000000000401858 -> 0000000000401838 2 cycles
..... 26: 0000000000401858 -> 0000000000401838 1 cycles
..... 27: 0000000000401858 -> 0000000000401838 2 cycles
..... 28: 0000000000401858 -> 0000000000401838 2 cycles
..... 29: 0000000000401858 -> 0000000000401838 4 cycles
..... 30: 0000000000401858 -> 0000000000401838 160 cycles
..... 31: 0000000000401858 -> 0000000000401838 2 cycles
```

Skylake Precise Front-End

- Ability to get precise IP attribution for front-end events via PEBS
  - Locate icache, itlb misses, frontend bubbles
- New FRONTEND_RETIRED event (code=0xc6,umask=0x1)
  - Extra MSR to encode actual frontend event

DSB_MISS	0x11	Decode stream buffer miss
L1I_MISS	0x12	L1I miss primary miss
L2_MISS	0x13	L2 code primary miss
ITLB_MISS	0x14	L1 ITLB primary miss
STLB_MISS	0x15	L2 TLB primary miss

```
$ perf record -e cpu/event=0xc6,umask=0x1,frontend=0x12/pp ….
```

Skylake precise frontend example: ITLB

- Test program : 1 function per page, 16 nop insn per func

```
static void func0(void) __attribute__((aligned(4096)));
static void func0(void){ asm volatile(".fill 16, 1, 0x90");}
static void func1(void) __attribute__((aligned(4096)));
static void func1(void){ asm volatile(".fill 16, 1, 0x90");}
...
```

```
$ perf record -e cpu/event=0xc6,umask=0x1,frontend=0x12/pp
```

```
Samples: 15K of event 'cpu/event=0xc6,umask=0x1,frontend=0x14/upp', Event count: 173206460
Overhead  Command          Shared Object      Symbol
   0.30%  func512-i16-ali  func512-i16-align  [.] func499
   0.30%  func512-i16-ali  func512-i16-align  [.] func184
   0.30%  func512-i16-ali  func512-i16-align  [.] func345
   0.29%  func512-i16-ali  func512-i16-align  [.] func231
   0.29%  func512-i16-ali  func512-i16-align  [.] func170
   0.28%  func512-i16-ali  func512-i16-align  [.] func22
   0.28%  func512-i16-ali  func512-i16-align  [.] func173
```

Precise front-end assembly view

- Haswell: no precise ITLB event: either ITLB_MISSES.STLB_HIT or MISS_CAUSES_A_WALK

Haswell

```
        |        static void func511(void)
        |        {
        |          asm volatile(".fill 16, 1, 0x90");
75.76   |5f7000:   nop
24.24   |5f7001:   nop
        |5f7002:   nop
        |5f7003:   nop
        |5f7004:   nop
        |5f7005:   nop
        |5f7006:   nop
        |5f7007:   nop
        |5f7008:   nop
        |5f7009:   nop
        |5f700a:   nop
        |5f700b:   nop
        |5f700c:   nop
        |5f700d:   nop
        |5f700e:   nop
        |5f700f:   nop
        |5f7010:   retq
```

Skylake

```
        |        static void func511(void)
        |        {
        |          asm volatile(".fill 16, 1, 0x90");
100.00  |5f7000:   nop
        |5f7001:   nop
        |5f7002:   nop
        |5f7003:   nop
        |5f7004:   nop
        |5f7005:   nop
        |5f7006:   nop
        |5f7007:   nop
        |5f7008:   nop
        |5f7009:   nop
        |5f700a:   nop
        |5f700b:   nop
        |5f700c:   nop
        |5f700d:   nop
        |5f700e:   nop
        |5f700f:   nop
        |5f7010:   retq
```

PEBS multi-entry mode

- Linux uses PEBS in single-entry mode
  - Captures only one sample before interrupting kernel
  - Necessary to add meta-data, such as PID/TID
  - Necessary to adjust sampling period
- PEBS supports N-entry mode
  - Much lower overhead : interrupt after N samples
- Kernel automatically enables multi-entry mode if
  - Fixed period
  - No timestamp (except Skylake)
  - PEBS buffer flushed on context-switches
  - Supported on all processors with PEBS
  - No LBR (no branch sampling)

PEBS multi-entry valid mode (pre-Skylake)

Multi-entry mode	Command line
No	`$ perf record -e cpu/event=0xc0,umask=1/upp … (timestamp is default mode)`
No	`$ perf record  -T -e cpu/event=0xc0,umask=1/upp …`
Yes	`$ perf record --no-timestamp -c 10000003 -e cpu/event=0xc0,umask=1/upp …`
Yes	`$ perf record -a --no-timestamp -c 10000003 -e cpu/event=0xc0,umask=1/upp …`

Skylake PEBS captures TSC

- Each PEBS sample timestamped with TSC
  - Can diff wallclock time between samples
- Must use perf_event_attr.clockid = 0 (= trace_lock = default)
- Real value in multi-pebs entry mode with timestamps

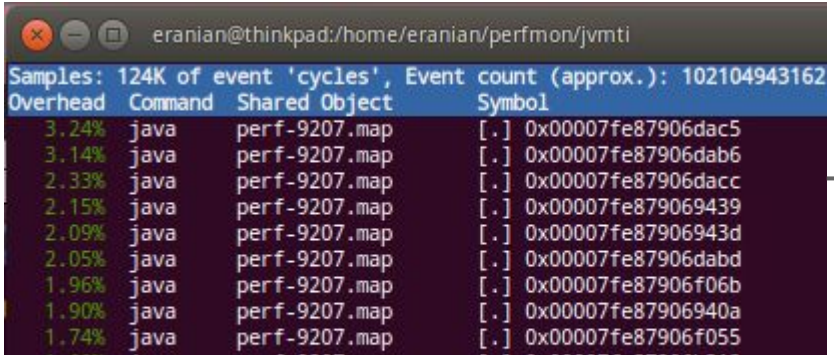
```
$ perf record -c 1000003 -e cpu/event=0xc0,umask=1/upp …
```

- How to extract?
  - perf script -F ip,time, ...

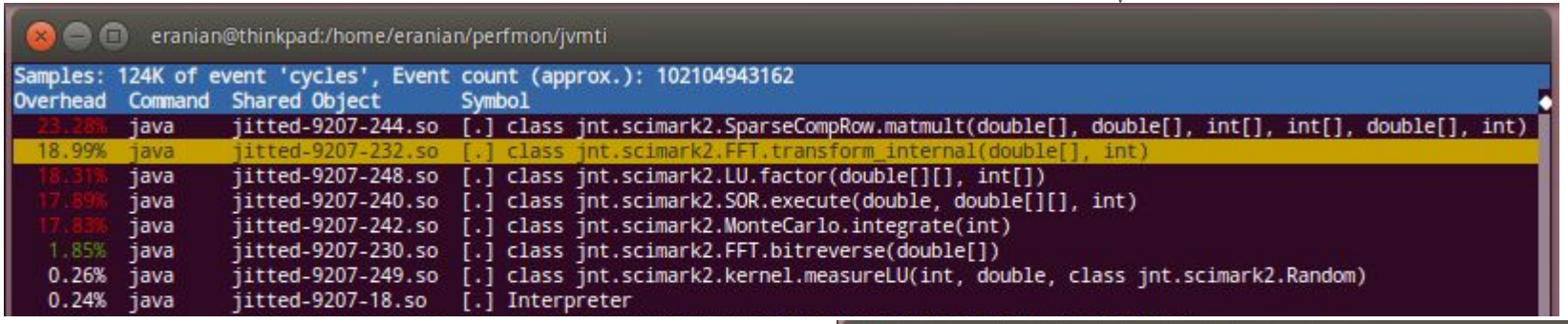
Perf jit support in Linux v4.6

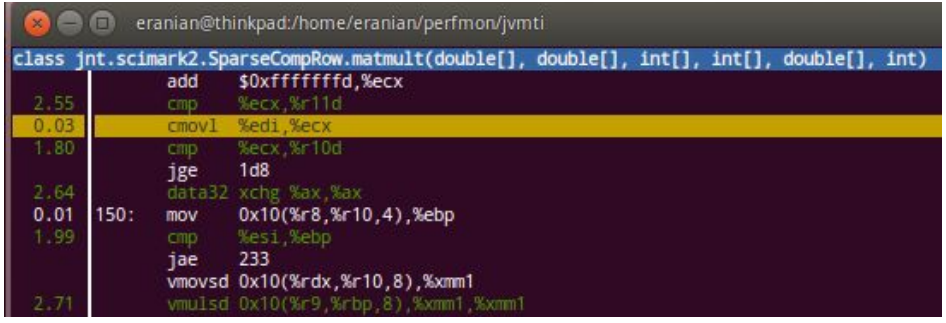
- Provides symbolization for jitted code
- Provides assembly and source levels profiles for jitted code
- Works with Java, V8, DART and possibly others
  - For java: perf comes with a JVMTI agent to use with the runtime
  - Supports code movements (re-jitting)

```
$ perf record -k mono -e cycles java -agentpath:/usr/lib/libjvmti.so my_class
$ perf inject -j -i perf.data -o perf.data.jitted
$ perf report -i perf.data.jitted
```



With the support





Broadwell & Skylake IPC quizz

```
401f10: cmp  %ebp,%r12d                    counts unit events
401f13: je   401f80                 3,068,459,643 cycles:u
401f15: add  $0x1,%rbx              9,167,665,598 uops_issued:any:u
401f19: test $0x3fff,%ebx           9,155,367,765 uops_retired:all:u
401f1f: jne  401f10                15,245,169,267 inst_retired:any_p:u
```

- IPC = 4.97 sustained! Whoaaou!
- But how is that possible?
- The machine can only issue up to 4 uops from the FE
- We know instructions always retire in order

Broadwell & Skylake IPC quizz

- All uops retired, no bad speculation.
- How many functional units used?

```
#=============================================================
#                                                       ports
#    Port0    Port1    Port2    Port3    Port4    Port5    Port6    Port7
#A/FPU/Jmp    A/FPU Ld/StAd Ld/StAd  StData    A/FPU    A/Jmp     StAd
#=============================================================
    99.63%   65.59%    0.01%    0.02%    0.01%   33.81%   99.23%    0.01%
```

- We see 4 units for a total of 300% total cycles = 3 units/cycle
- 5 instructions => 3 units => 2 branch, 2 ALU => macro fusion
- 5 instructions => 3 uops issued by frontend, expand to 5 insn at retirement

Broadwell & Skylake IPC quizz

- Topdown analysis confirms

```
#=================================
#                            topdown
# ---------------------------------
#          unit: issue slots
# ---------------------------------
# FrontEnd     Bad     Uops BackEnd
#  Bound      Spec Retiring   Bound
#=================================
    24.91%    0.22%   74.48%   0.38%
```

3 uops out of 4 (sustained) = 5 instructions

25% capacity left = issuing 3 out of 4 uops from FE

Conclusions

- SkyLake PMU introduces very powerful features
- Linux v4.4 has full Skylake support
- Linux tools are getting better