# Improving perf_events measurement correctness

Maria Dimakopoulou
Optimization Team

CERN PMU Workshop 2013

# What Is It About?

- at-retirement memory events may corrupt events on the sibling counter with HyperThreading enabled on Intel processors

```
0xd0 : MEM_UOPS_RETIRED.*
0xd1 : MEM_LOAD_UOPS_RETIRED.*
0xd2 : MEM_LOAD_UOPS_LLC_HIT_RETIRED.*
0xd3 : MEM_LOAD_UOPS_LLC_MISS_RETIRED.*
```

- Example: SNB, CPU0,1 siblings

```
# perf stat -a -C0  -e r81d0 sleep 10   (r81d0: MEM_UOPS_RETIRED:ALL_LOADS)
# perf stat -a -C1  -e r20cc sleep 1    (r20cc: ROB_MISC:LBR_INSERTS )
       10,022,279 r20cc    (LBR unused: should be zero)
```

- Silent & random measurement corruption

- Errata: SandyBridge (BJ122), IvyBridge (BV98), Haswell (HSD29)

# Severity

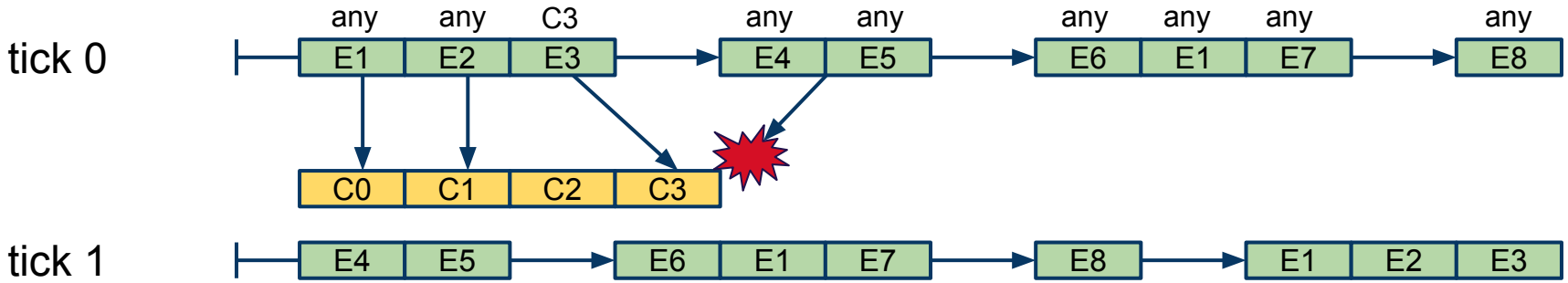
- Corrupting events are commonly used
  - to study cache behavior
- Multiplexing increases risk
  - occurs asynchronously on each CPU
- Error maximized with high frequency vs. low frequency events
  - `mem_load_uops_retired vs. mem_load_uops_llc_miss_retired:remote_dram`
  - `mem_load_uops_retired vs. mispredicted_branch_retired`
  - `...`

# Solutions

- No Intel firmware fix available
- Only measure one logical CPU per physical core
  - coarse-grained exclusion
- Current Kernel Fix: black-list corrupting events (IvyBridge for now)
  - at-retirement memory events can never be measured
- Our approach: USX Protocol (Cache Coherence Style Protocol)
  - fine-grained exclusion based on the sibling thread's state
  - force mutual exclusion for counters with corrupting events
  - allow sharing for counters with non-corrupting events

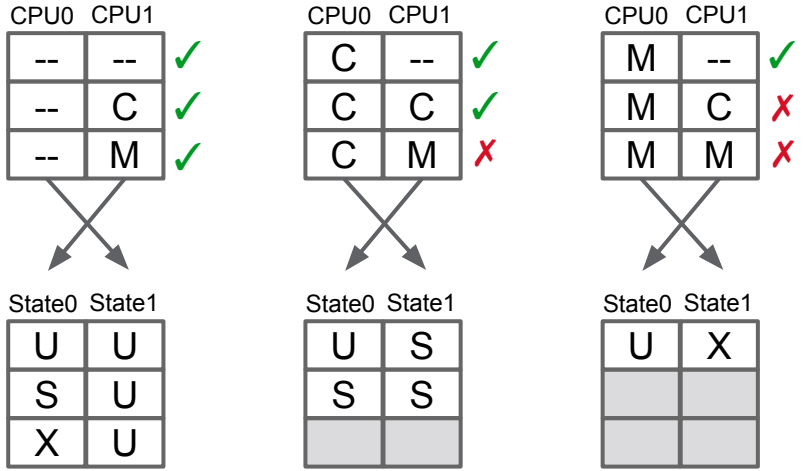
# perf_events Scheduler Overview

- Kernel-level scheduling of event groups
  - greedy 1st match algorithm, stops at first error
- Static constraints on events are hardcoded in kernel
- Multiplexing if necessary
  - Round-Robin of event group list for fairness
  - default rate is each timer tick

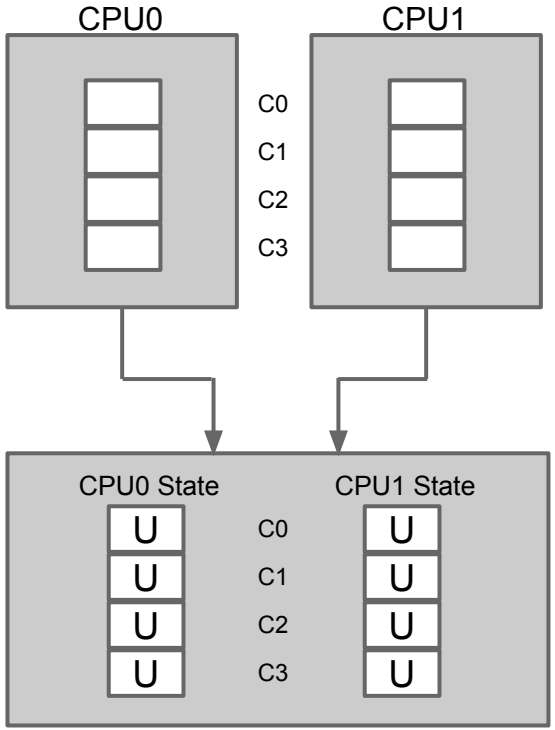


# USX Protocol

- Counter Events
  - **C**ycles = Non-Corrupting
  - **M**emory = Corrupting
- Counter States
  - **U**nused
  - **S**hared
  - **X**clusive
- Principles
  - event scheduling on **one HT's counters** affects the **other's HT's state**
  - **M** events → allowed on counters only with **U** state
  - **C** events → allowed on counters only with **U or S** state

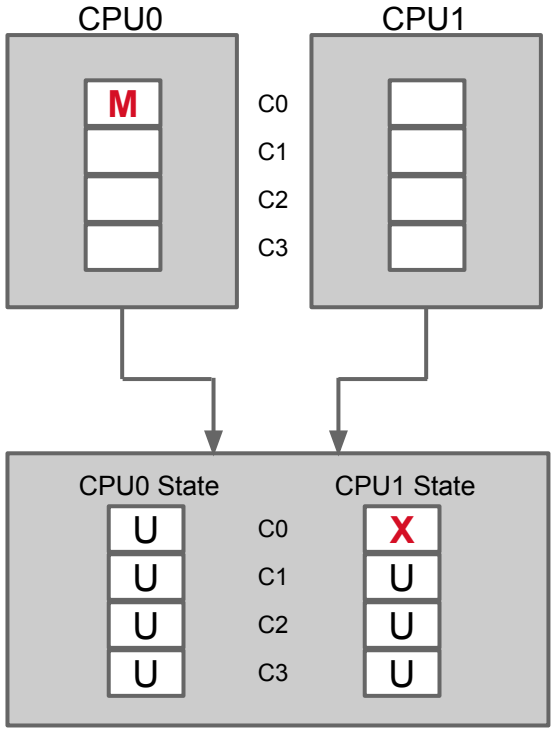


# USX Protocol: Example



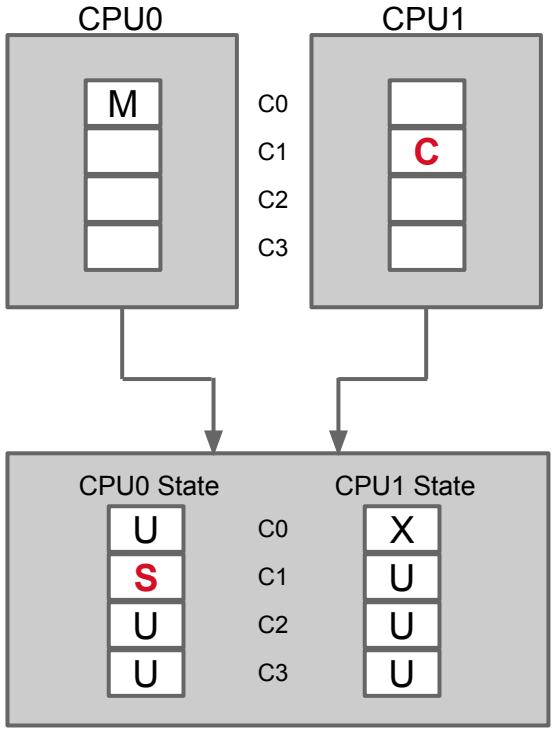

- CPU0, CPU1 hyperthreads
- Event Lists
  - CPU0: M, C, C
  - CPU1: C, M, M
- Initial State

# USX Protocol: Example



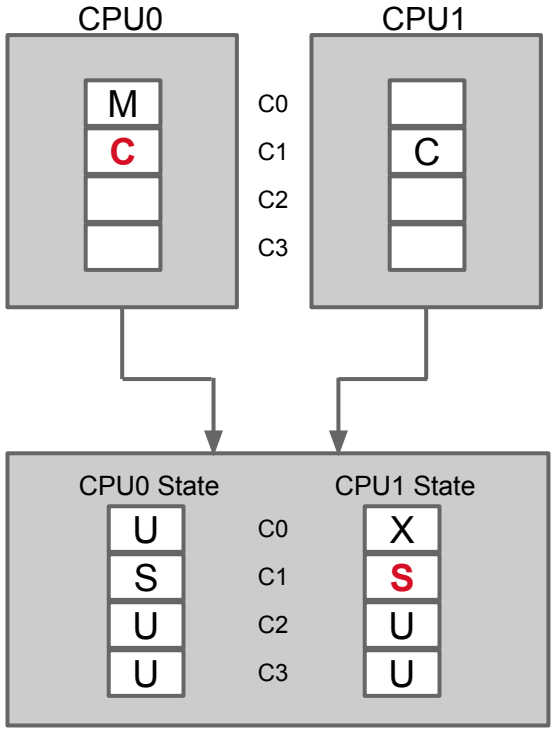

1. Add M event on CPU0
2. M static constraint: 1111 (run on any counter)
3. CPU0 state constraint: 1111
   - all counters unused
4. M dynamic constraint: 1111 & 1111 = 1111
5. Scheduler picks counter0
6. Mark counter0 in CPU1 as Xclusive
   - No events can be scheduled on it

# USX Protocol: Example



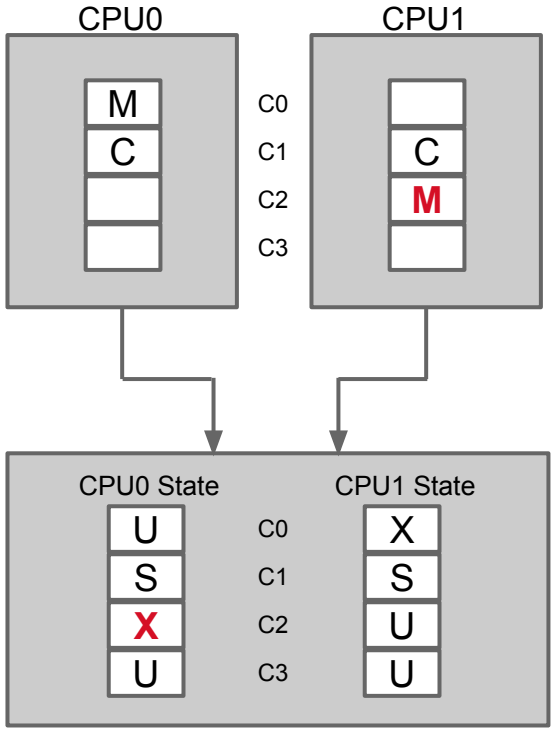

1. Add C event on CPU1
2. C static constraint: 1111 (on any counter)
3. CPU1 state constraint: 1110
   - counter0 marked as X
4. C dynamic constraint: 1111 & 1110 = 1110
5. Scheduler picks counter1
6. Mark counter1 in CPU0 as Shared
   - Only C events can be scheduled on it

# USX Protocol: Example



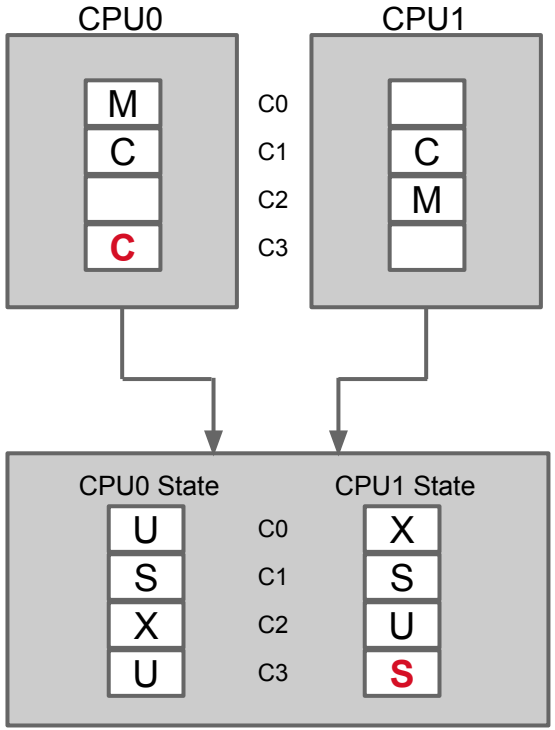

1. Add C event on CPU0
2. C static constraint: 1111 (on any counter)
3. CPU1 state constraint: 1111
   - C events allowed on S counters
4. C dynamic constraint: 1111 & 1111 = 1111
5. Scheduler picks counter1
6. Mark counter1 in CPU1 as Shared
   - Only C events can be scheduled on it

# USX Protocol: Example



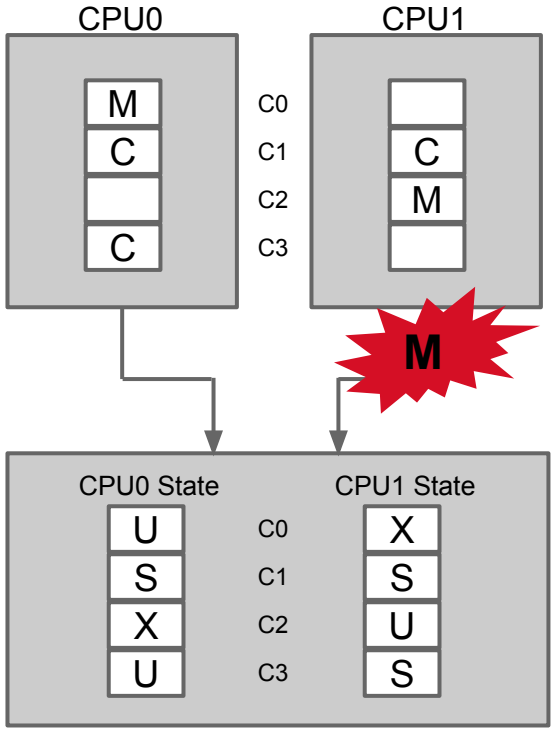

1. Add M event on CPU1
2. M static constraint: 1111 (on any counter)
3. CPU1 state constraint: 1100
4. C dynamic constraint: 1111 & 1100 = 1100
5. Scheduler picks counter2
6. Mark counter2 in CPU0 as Xclusive
   - no events can be scheduled on it

# USX Protocol: Example



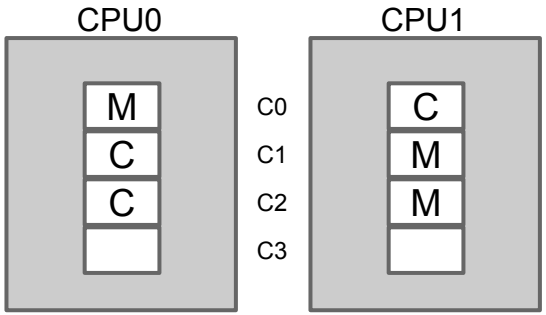

1. Add C event on CPU0
2. M static constraint: 1111 (on any counter)
3. CPU0 state constraint: 1011
4. C dynamic constraint: 1111 & 1011 = 1011
5. Scheduler picks counter3
6. Mark counter3 in CPU1 as Shared
   - only C events can be scheduled on it

# USX Protocol: Example

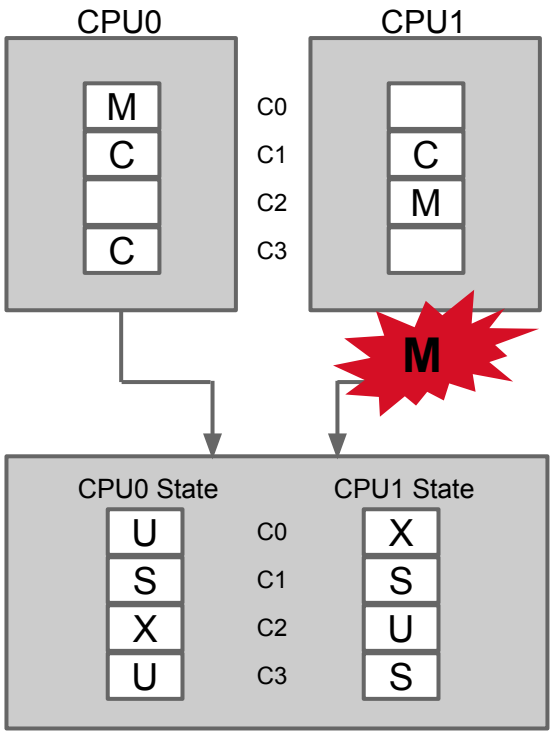



1. Add M event on CPU1
2. M static constraint: 1111 (on any counter)
3. CPU1 state constraint: 0100
4. M dynamic constraint: 1111 & 0100 = 0100
5. Scheduler cannot pick counter2: occupied
   - **Multiplexing!**

# USX Protocol: Example




**Broken Results**




**Correct Results**
**Multiplexing**

# USX Protocol: Example Results

```
# perf stat -a -C0 --pfm-events mem_uops_retired:all_loads,
                               rob_misc_events:lbr_inserts, rob_misc_events:lbr_inserts triad
# perf stat -a -C1 --pfm-events rob_misc_events:lbr_inserts,
                               mem_uops_retired:all_loads,mem_uops_retired:all_loads triad

     6 723 587 814 mem_uops_retired:all_loads    [100,00%]
       30 095 207 rob_misc_events:lbr_inserts    [100,00%]      CPU0
       30 095 207 rob_misc_events:lbr_inserts    [100.00%]
                                                                          Without the change
       29 874 562 rob_misc_events:lbr_inserts    [100,00%]
     1 684 288 987 mem_uops_retired:all_loads    [100,00%]      CPU1
     1 684 289 648 mem_uops_retired:all_loads    [100.00%]
```

---

```
     6 723 554 013 mem_uops_retired:all_loads    [100,00%]
                 0 rob_misc_events:lbr_inserts    [100,00%]      CPU0
                 0 rob_misc_events:lbr_inserts    [100.00%]
                                                                          With the change
                 0 rob_misc_events:lbr_inserts    [66,67%]
     1 686 589 611 mem_uops_retired:all_loads    [66,67%]       CPU1
     1 684 570 798 mem_uops_retired:all_loads    [33,33%]
```

Google

# USX Protocol: Other Results

- Initial example SNB, CPU0,1 siblings

```
# perf stat -a -C0  -e r81d0 sleep 10    (r81d0: MEM_UOPS_RETIRED:ALL_LOADS)
# perf stat -a -C1  -e r20cc sleep 1     (r20cc: ROB_MISC:LBR_INSERTS)
      0 r20cc
```

- Example with overcommitted counters (multiplexing)

```
# perf stat -a --pfm-event rob_misc_events:lbr_inserts, mem_uops_retired:all_loads,...

      5 091 rob_misc_events:lbr_inserts  [79,98%]
4 594 343 504 mem_uops_retired:all_loads  [79,99%]
4 601 444 895 mem_uops_retired:all_loads  [80,02%]
4 593 376 037 mem_uops_retired:all_loads  [80,03%]
4 595 672 820 mem_uops_retired:all_loads  [79,98%]

          0 rob_misc_events:lbr_inserts  [49,19%]
4 118 777 212 mem_uops_retired:all_loads  [41,60%]
4 106 198 318 mem_uops_retired:all_loads  [40,29%]
4 058 269 131 mem_uops_retired:all_loads  [46,27%]
4 200 051 192 mem_uops_retired:all_loads  [41,56%]
```

Without the change

With the change

# Summary

- provided a work-around to unsolved reliability issue on SNB/IVB/HSW
  - no change to the way the workload runs
  - no user-level changes
- all events can now be measured reliably
  - valuable for tools such as Gooda, Perf, GWP
- more **reliability** at the cost of **extra multiplexing**
  - need for an optimal scheduling algorithm (Google Optimization Team)
- kernel patches to be pushed to upstream kernel

# References

- Intel SandyBridge specification update
- Intel IvyBridge specification update
- Intel Haswell specification update
- Gooda Tool
- IA-32 Software Developers Manual (SDM) Vol3b September 2013