Advanced Operating System: Hardware Performance Counters (HWPMC)

Dr Robert N. M. Watson

2020-2021

Hardware performance counters are a processor facility that gathers statistics about *architectural* and *micro-architectural* performance properties of code execution and data access:

Architectural features are those exposed explicitly via the documented instruction set and device interfaces – e.g., the number of instructions executed, or the number of load instructions executed.

Micro-architectural features have to do with the programmer-transparent implementation details, such as pipelining, superscalar execution, caches, and so on – e.g., the number of L2 cache misses taken.

The scope for *programmer transparency* (e.g., what is included in the architecture vs. micro-architecture) varies by Instruction-Set Architecture (ISA): whereas MIPS exposes certain pipelining effects to the programmer (e.g., branch-delay slots), ARM and x86 minimise the visible exposure other than performance impact. MIPS and ARM both require explicit cache management by the operating system during I/O operations and code loading, whereas x86 also masks those behaviours.

Although this distinction between architectural and microarchitectural events seems clear, in fact life is more tricky. In some microarchitectures (especially superscalar ones), tracking committed instructions can affect area and frequency – e.g., by requiring the full program counter (PC) or instruction encoding to be plumbed through the implementation where the design might not otherwise require it. As a result, counters for architectural events (e.g., loads and stores) might in fact be for speculated rather than committed instructions. When the processor is predicting accurately, committed instruction count will dominate incorrectly inspected instruction count, leading the two to be largely the same. However, there may be edge cases in which this is not true. For our purposes, we may reasonably consider them interchangeable.

How counters are used

Performance counters can be used in two ways: *counting*, in which instances of a particular architectural or micro-architectural event are counted during program execution; and *sampling*, in which $1/n$ instances of the event will trigger a hardware trap that allows, for example, a stack trace to be taken (similar to historic timer-driven profiling techniques). In these labs, we will use PMC only in counting mode.

PMC support may be integrated into the operating system in a variety of ways. Typically, this is done by an additional tracing and profiling framework: in FreeBSD, HWPMC; in Linux, OProfile and related tools. It is also possible to integrate PMC support with DTrace, as has been done in Solaris, but not yet in FreeBSD. On FreeBSD, HWPMC provides a programming API that allows applications to measure their own micro-architectural impacts. We have integrated explicit PMC support into the IPC benchmark using these APIs, allowing it to count events such as memory accesses and cache misses at various points in the cache hierarchy.

The Arm Cortex A72, used in the RPi4, can track up to six sources at a time. Our benchmark will automatically track instructions retired (those performed architecturally) and CPU cycles. It will also track four other counters that vary depending on the command-line parameters you use.

In these labs, we will focus almost exclusively on memory-related counters, rather than looking at other micro-architectural performance events such as branch prediction. This is because our IPC benchmark results will be most strongly affected by memory footprint of our buffers and IPC primitives.

FreeBSD also includes tools to sample PMC behaviour by process or systemically, capturing stack traces via sampling, and mapping them back to program symbols or annotated source code. You may wish to also use these

tools to help explain performance behaviour (i.e., not just that L2 cache misses were dominant at a particular buffer size, but also that the majority of cache misses were taken in a particular part of the kernel), but that is not required. If you wish to use these tools, please see the FreeBSD `pmcstat(8)` man page for details on capturing counter data for whole-program and whole-system analysis. Note that, although on the hardware side PMC may have no measurable probe effect, the software framework around PMC (i.e., to virtualise counters across multiple processes) can introduce substantial probe effect, which we must be aware of when using these counters for performance analysis.

Performance counters in the IPC benchmark

The IPC benchmark has a `-P` argument that requests use of performance counters to analyse the IPC loop. Performance counters are configured in "process mode", meaning that they track user and kernel events associated with a process and its descendents, so should include events from all three of our benchmark modes including their execution in kernel, but not other system events. Where events occur asynchronously in a kernel thread not explicitly associated with the user process, those events will not be counted (e.g., kernel work performed by a timer on behalf of a user process).

Performance-counter arguments

Performance-counter support are enabled using the `-P` flag, which accepts one argument identifying the set of counters to track during execution. Due to the 6-counter limit in the Cortex A72, it is not possible to count all the potential events of interest at the same time. As such, some care will be required to take multiple samples and consider counter readings as members of a distribution. The following counter modes are supported:

arch Track architectural events – i.e., those relating to instructions: loads, stores, exception returns, and function returns. Note that in the microarchitecture used in these labs, some of these counters are for speculated rather than committed instructions.

dcache Track data-cache events: level-1 data and level-2 cache hits and misses.

instr Track level-1 instruction-cache hits and misses, as well as speculated branch-predictor hits and misses.

tlbmem Track ITLB and DTLB misses, memory access instructions, and bus accesses.

Specific performance counters

These counter descriptions are drawn from the *Arm® Architecture Reference Manual - Armv8, for Armv8-A architecture profile* (section D7.11) and *ARM® Cortex®-A72 MPCore Processor Technical Reference Manual (Revision: r0p3)* (section 11.8).

BR_MIS_PRED The counter counts each correction to the predicted program flow that occurs because of a misprediction from, or no prediction from, the branch prediction resources and that relates to instructions that the branch prediction resources are capable of predicting.

BR_PRED The counter counts every branch or other change in the program flow that the branch prediction resources are capable of predicting.

BR_RETURN_SPEC The counter counts every branch speculatively executed - Procedure return.

BUS_ACCESS The counter counts Memory-read operations and Memory-write operations that access outside of the boundary of the PE and its closely-coupled caches

CPU_CYCLES The counter increments on every cycle.

EXC_RETURN The counter increments for each executed exception return instruction.

INST_RETIRED The counter increments for every architecturally executed instruction.

L1D_CACHE The counter counts each Memory-read operation or Memory-write operation that causes a cache access to at least the Level 1 data or unified cache.

L1D_CACHE_REFILL The counter counts each access counted by L1D_CACHE that causes a demand refill of at least the Level 1 data or unified cache from outside the Level 1 cache.

L1D_TLB_REFILL The counter counts each Attributable Memory-read operation or Attributable Memory-write operation that causes a TLB refill of at least the Level 1 data or unified TLB.

L1I_CACHE The counter counts Attributable instruction memory accesses that access at least the Level 1 instruction or unified cache.

L1I_CACHE_REFILL The counter counts each access counted by L1I_CACHE that causes a demand refill of any of the Level 1 caches outside the Level 1 caches of this PE.

L1I_TLB_REFILL The counter counts Attributable instruction memory accesses that cause a TLB refill of at least the Level 1 instruction TLB.

L2D_CACHE The counter counts each Memory-read operation or Memory-write operation that causes a cache access to at least the Level 2 data or unified cache.

L2D_CACHE_REFILL The counter counts each access counted by L2D_CACHE that causes a refill of a demand refill of any of the Level 1 or Level 2 caches from outside the Level 1 and Level 2 caches of the PE.

LD_SPEC The counter counts each operation speculatively executed - Load instructions.

MEM_ACCESS The counter counts Memory-read operations and Memory-write operations that the PE made. The counter increments whether the access results in an access to a Level 1 data or unified cache, a Level 2 data or unified cache, or neither of these.

ST_SPEC The counter counts each operation speculatively executed - Store instructions.