Accurate cycles measurement

Offline Mazen Ezzeddine over 4 years ago

Dear experts,

I am currently trying to measure the cycles required to context switch between two linux processes and the cycles required to world-switch between two linux VMs running above a thin bare-metal hypervisor. For this purpose, I am using the PMCCNTR (cycle counter register) after enabling/disabling it in the PMCR and set it to increment once every 64 cycles.

- Does the value of the PMCCNTR reset when switching from the user mode to the kernel mode or even to the hypervisor or it keeps its value when migrating between different provilegeslevels until it is reset by the user. How about when switching between different same-privilege contexts?

- To get accurate and reliable results , I have to to disable interrupts when measurements are made, disable preemption(in the case of linux processes), and insert serializing instruction barriers at the front-end and back-end of the profiled code. What else shall I pay attention to?

- When calculating the total time how does the Linux kernel optimizations such as the DVFS influence the result? Do I need to turn off the DVFS optimization? Any suggestion.

- The PMCCNTR is a 32-bit register so it can actually count 4294967296 * 64=276690969984, is this a concern (e.g. overflow), is it advisable to use the 64-bit architected counter/timer instead? Any advanced tutorial on the usage of the 64-bit architected counter for accurate profiling. Does the 64-bit system architected counter run at frequency of the processor, how can I get the procesor cycles from the architected counter which to my knoweldge run a lower frequency?

Thank you so much.

Feedback

Offline Martin Weidmann over 4 years ago

> *- Does the value of the PMCCNTR reset when switching from the user mode to the kernel mode or even to the hypervisor or it keeps its value when migrating between different provilegeslevels until it is reset by the user. How about when switching between different same-privilege contexts?*

The processor won't auto disable/reset PMU counters when switching modes.

However, you might need to watch out for it being done in software. Applications (User mode/EL0) don't typically have direct access to the PMU. If they use the PMU it's normally via an API such as perf or oprofile. In which case you'd expect these libraries to manage/hide the affect of task switching - which might include save/restoring PMU configs and counts.

Another thing to watch out for filtering. If you processor (you didn't mention which you're using) implements PMUv2, then events can be filtered by privilege level. For example, "filter" any events that occur in PL2/EL2 to avoid execution in the Hypervisor interfering with the counts seen by a guest OS.

> *- To get accurate and reliable results , I have to to disable interrupts when measurements are made, disable preemption(in the case of linux processes), and insert serializing instruction barriers at the front-end and back-end of the profiled code. What else shall I pay attention to?*

It partly depends on what you want to measure. By adding additional barriers/disabling interrupts you are no longer measuring what would otherwise have happened. Which might be fine.

There is also the question of things like cache/TLB misses. You'll expect to see variations in how long a task switch would take depending on the state of the caches and TLBs. Which will be affected by what's run recently. So, for example, you might see a switch between two threads of the same process (or two vCPUs of the same VM) take less time than a switch between two unrelated threads (or vCPUs). As the TLBs are more likely to hit.

> *- When calculating the total time how does the Linux kernel optimizations such as the DVFS influence the result? Do I need to turn off the DVFS optimization? Any suggestion.*

Again, it kind of depends on what you care about. The PMU cycle counter counts cycles (yes, I know - I'm stating the obvious) not time.

The frequency can impact the cycle counts. For example, if the memory system is not subject to DVFS, a cache line fille will (typically) require fewer core cycles as the core clock is ramped down.

If what you want "time" you might be better off using the System Timer (which is based off a fixed frequency) or an external timer. If you care about cycles, you might need to do several measurements with different settings.

> *- The PMCCNTR is a 32-bit register so it can actually count 4294967296 * 64=276690969984, is this a concern (e.g. overflow), is it advisable to use the 64-bit architected counter/timer instead? Any advanced tutorial on the usage of the 64-bit architected counter for accurate profiling. Does the 64-bit system architected counter run at frequency of the processor, how can I get the procesor cycles from the architected counter which to my knoweldge run a lower frequency?*

2^32 is still a pretty big count! My quick calculation puts the 0 to overflow time at about 4 seconds assuming 1GHz.

Feedback