Perfctr-Xen: A Framework for Performance Counter Virtualization

Ruslan Nikolaev, Godmar Back
Department of Computer Science
Virginia Tech, Blacksburg, VA 24061

Background and Motivation

- IaaS (Infrastructure as a Service) systems are gaining significant leverage among developers. These systems widely use virtual machine monitors such as Type-I hypervisors: Xen, KVM, VMWare ESX (Figure 1)
- Developers working in these systems need powerful tools for performance evaluation. Commonly used performance analysis tools (e.g., PAPI) cannot be used because existing VMM and guests do not provide necessary per-thread virtualization support for hardware event counters. There are only a few existent solutions
- Widely used XenoProf is an extension of OProfile system-wide profiler. It does not provide per-domain abstraction of hardware counter facilities (supports only 1 domain at a time)
- Xen's VPMU driver works only with hardware assisted virtualization and currently supports very limited number of architecture generations. Each CPU generation code for performance counting is completely incompatible with the other one, and VMM must contain a great deal of architecture-specific code. In addition, Xen hypervisor is commonly used in paravirtualized mode which does not use hardware assistance

Library or Framework	Type	Monitoring	Direct access	Interface used
Perf_events	Low level	Per thread	Yes	ioctl, mmap, sysctl, prctl
Perfctr	Low level	Per thread	Yes	ioctl, mmap, dev
Perfmon	High and low level	Per thread	No	syscalls, mmap, signals
PAPI	High level	Per thread	Yes (w/ perfctr)	perfctr, perf events, perfmon
OProfile	Profiler	System wide	N/A	oprofilefs
TAU PerfExplorer	Profiler	Per thread	N/A	PAPI
HPCToolkit	Profiler	Per thread	N/A	PAPI

Table 1. Libraries and frameworks for performance measurement

- To support per-thread monitoring in virtualized environments, the hardware event counters must be virtualized. It may be challenging because of the mutual blindness of the hypervisor and guest threads. Both inter-domain and intra-domain context switches must be taken into account as shown in Figure 2
- Table 1 presents commonly used libraries and frameworks (in non-virtualized environments) and their corresponding characteristics. These libraries work with hardware event counters, and many of them provide per-thread virtualization (i.e. a thread can obtain its logical event count from the global hardware event count). Notice that some libraries provide "Direct Access" which means that a thread can obtain its per-thread event count directly without using expensive system calls

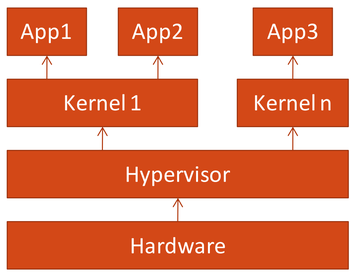

Figure 1. Type-I hypervisor architecture

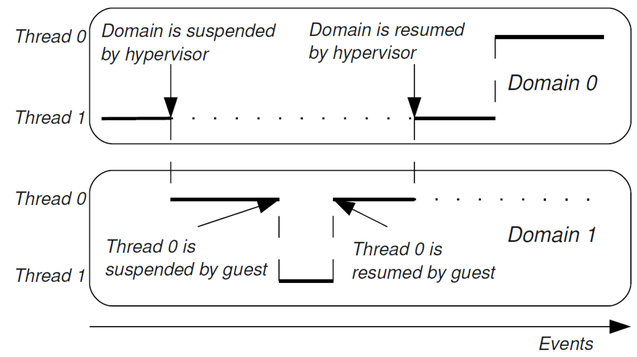

Figure 2. Inter-domain and Intra-domain context switches must be taken into account for proper virtualization

- We want to provide an efficient and universal framework that can work in any mode of virtualization. It must be compatible with commonly used toolkits (in non-virtualized environments) such as PAPI, HPCToolkit, TAU PerfExplorer, so that they can be used in virtualized environements. Performance Monitoring Unit (PMU) is extremely architecture-specific. We want to reuse as much architecture-dependent code as possible, so that new architectures can be added easily to our framework from the existent codebase

Implementation

- Our framework is based on perfctr library (which is widely used for non-virtualized environments) and is fully compatible with many well-known toolkits such as PAPI and HPCToolkit
- All above-mentioned toolkits can run unmodified as shown in Figure 3. Modifications are only needed for the low-level performance counter layer as shown with (*)
- Guest's and hypervisor's states are maintained as shown in Figure 4

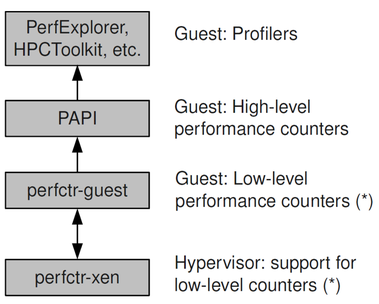

Figure 3. Software layers in perctr-xen

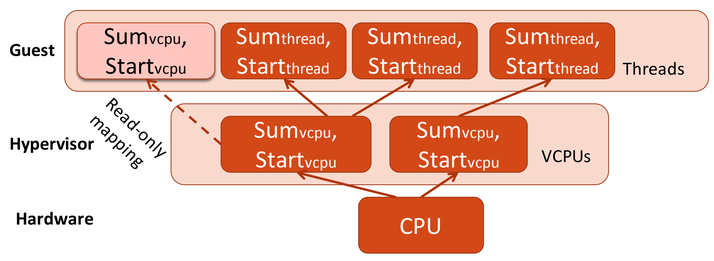

Figure 4. Guest's and hypervisor's state values

Accumulative mode (a-mode) counters

- Logical per-thread value

$$Log_{thread}(t) = Sum_{thread} + (Phys^*(t) - Start^*_{thread})$$

- Adjusted physical value for possible VCPU preemption

$$Phys^*(t) = Sum_{vcpu} + (Phys(t) - Start_{vcpu})$$

- Compensation for the hypercall

$$Start^*_{thread} = Phys^*(t_c)$$

- Hypercall necessary for the new configuration only

Interrupt mode (i-mode) counters

- We use save-and-restore mechanism to preserve proper interrupt capabilities
- Hypervisor delivers overflow interrupts to guest via VIRQ_PERFCTR virtual interrupts. Upon receipt, guest kernel signals user thread
- Since virtual interrupts are delivered asynchronously (as soft interrupts), guest must ensure that overflow interrupt is delivered to correct thread by rechecking overflow status. If the thread causing overflow is suspended before virtual interrupt arrives at guest, mark as pending and deliver on the next resume

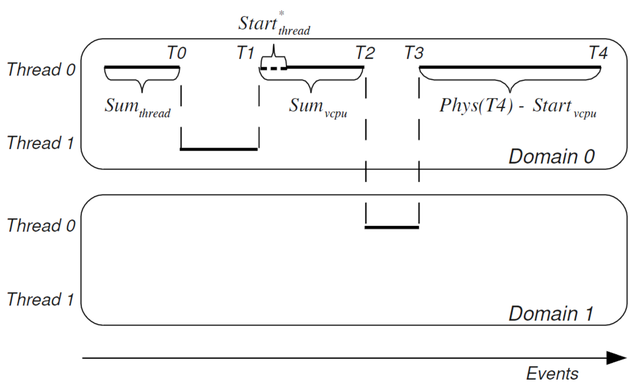

Figure 5. Example scenario for virtualized counters

Figure 5 shows an example scenario. Initially, thread 0 in domain 0 is running.

- At point T0, thread 0 is suspended by the guest kernel and its accumulated event count is recorded in Sum_{thread}
- At T1, thread 0 is resumed. The hypervisor sets $Sum_{vcpu} = 0$; upon return from the hypercall, the guest records $Start_{thread}$
- At point T2, the domain is suspended; the hypervisor records the number of events elapsed in Sum_{vcpu} and later resumes the domain at point T3. At this point the hypervisor samples $Start_{vcpu}$ as Phys(T3)
- The logical value computed at time T4 reflects the sum of the three segments during which the thread was active, while excluding those time periods during which the thread or domain was suspended

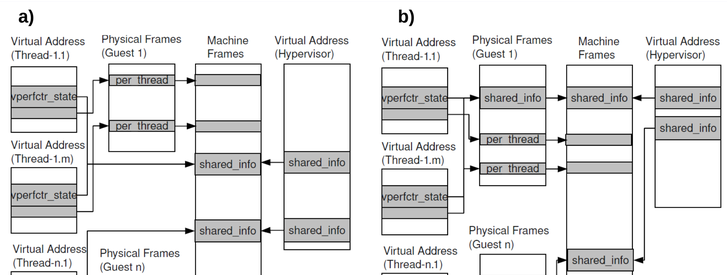

Figure 6. Page mapping (a) in paravirtualized mode; (b) in HVM mode

Software Engineering Considerations

- In paravirtualized mode, shown in Figure 6 (a), the shared_info structure does not appear as physical memory to the guest kernel. It is allocated by the hypervisor in machine memory and appears at a fixed virtual address in the guest kernel's address space. The guest kernel uses a Xen Guest API (xen_remap_domain mfn_range) to create an additional mapping to these machine frames
- In hardware-assisted mode, shown in Figure 6 (b), the guest can allocate the shared_info in any of its physical page frames. The chosen physical address is communicated from the guest kernel to the hypervisor. Linux's standard mapping API (vm_insert_page) can be used to add mappings into the user threads' address spaces
- We expose the per-VCPU data structures of all VCPUs to every user thread. We also added an additional field smp_id to the per-thread structure to record the VCPU on which the thread is resumed. The user-level library uses this field as an index to access the correct per-VCPU structure
- Thread or domain migrations may occur while accessing counters. We implemented an optimistic approach in which we check if the values of the $Start_{thread}$ and $Start_{vcpu}$ fields corresponding to the TSC counter changed between before and after the attempted access. Such a change indicates a domain and/or thread migration, in which case we retry the access until we succeed
- For the guest kernel driver, we replaced the functions that assumed direct access to the hardware with the appropriate hypercalls
- For the hypervisor driver, we needed to provide glue code so that it could function within the Xen hypervisor rather than the Linux kernel for which it was designed. It is written in the form of preprocessor macros and inlined functions contained in a separate header file, allowing us to avoid changes to most of the perfctr code

Results

- We used native execution environment as the baseline
- Exercised multiple VCPU/PCPU scenarios
- We ran our framework using different virtualization modes: paravirtualization and hardware-assisted virtualization (HVM)
- A specially developed microbenchmark verified correctness and accuracy of a-mode counters (Figure 9)
- PAPI built-in test verified correctness and accuracy of i-mode counters (Figures 7, 8)
- We executed macrobenchmarks from SPEC CPU 2006 to test our framework with real-life applications (Figures 10-13)
- To verify profiling, we have chosen 429.mcf benchmark and used HPCToolkit (Table 2)

a)

Function	1:D0	1:D1	2:D0	2:D1
main	1.01	1.01	0.98	1.01
global_opt	1.01	1.02	0.98	1.02
price_out_impl	1.02	1.04	0.99	1.03
primal_net_simplex	1	1	0.97	1
primal_bea_mpp	1	0.98	0.97	1
replace_weaker_arc	0.99	1.05	1.04	1.09
refresh_potential	1.02	1.28	1.03	1.04
update_tree	0.86	0.75	0.84	0.86
primal_iminus	1.53	1.47	1.23	1.27
insert_new_arc	0.99	0.87	0.95	0.96
flow_cost	0.97	0.97	0.97	0.97
dual_feasible	1.03	1	1	1
suspend_impl	1.07	1	1	1.07

b)

Function	1:D0	1:D1	2:D0	2:D1
main	0.98	0.99	0.95	0.98
global_opt	0.98	1	0.95	0.98
price_out_impl	0.98	1.01	0.95	0.99
primal_net_simplex	0.98	0.98	0.94	0.96
primal_bea_mpp	0.99	0.99	0.97	0.98
replace_weaker_arc	0.9	0.97	0.88	0.94
refresh_potential	0.96	0.98	0.88	0.9
update_tree	0.96	0.97	0.9	0.93
primal_iminus	1.56	1.56	1.36	1.51
insert_new_arc	1.2	1.13	1.02	1.13
flow_cost	0.94	0.93	0.94	0.94
dual_feasible	0.95	0.95	0.95	0.93
suspend_impl	0.91	0.9	0.9	0.89

Table 2. HPCToolkit profiling results for 429.mcf, ratio virtualized/native (a) Branch instructions, (b) Total cycles

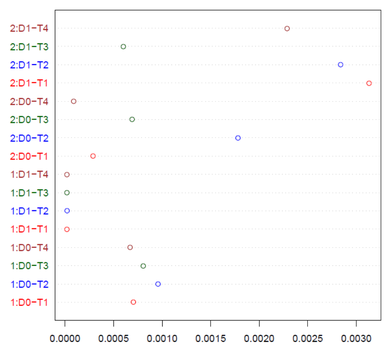

Figure 7. PAPI overflow: Relative error for i-mode counters (FP instructions)

1. Dom0 and Dom1 run on separate PCPUs.
2. Dom0 and Dom1 run on a shared PCPU.

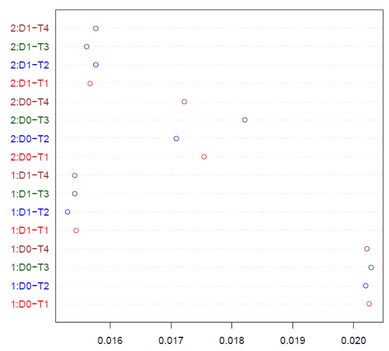

Figure 8. PAPI overflow: Relative error for i-mode counters (total cycles)

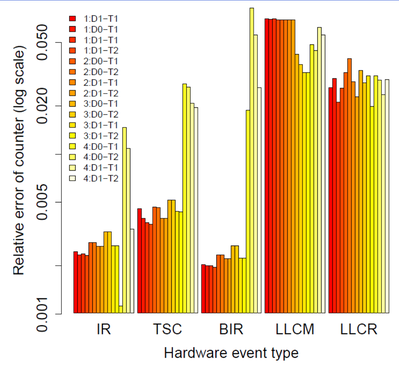

Figure 9.Microbenchmark for a-mode counters

1. Each domain on 2 dedicated PCPUs; each thread on a dedicated VCPU.
2. Each domain on a dedicated PCPU; all threads in a domain on a shared VCPU.
3. All domains on a shared PCPU; all threads on a shared VCPU.
4. Random migration PCPUs and VCPUs

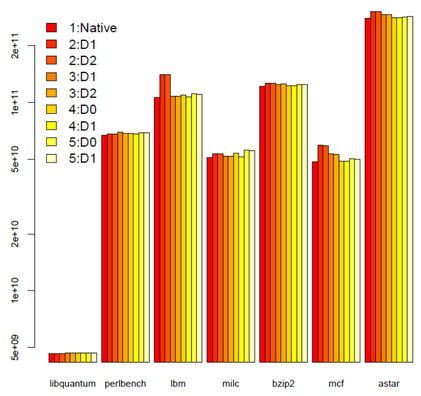

Figure 10. SPEC CPU2006: Relative error for TSC

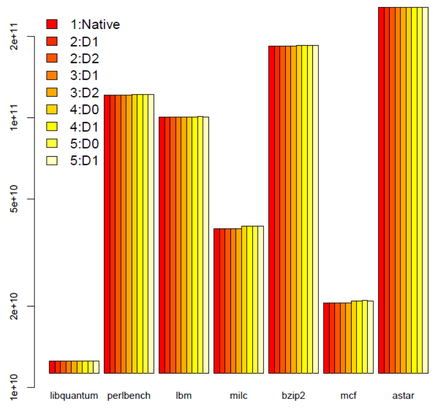

Figure 11. SPEC CPU2006: Relative error for Instructions Retired

1. Native mode
2. Fully-virtualized Dom1 and Dom2, each on a dedicated core
3. Fully-virtualized Dom1 and Dom2 on the same core
4. Paravirtualized Dom0 and Dom1, each on a dedicated core
5. Paravirtualized Dom0 and Dom1 on the same core

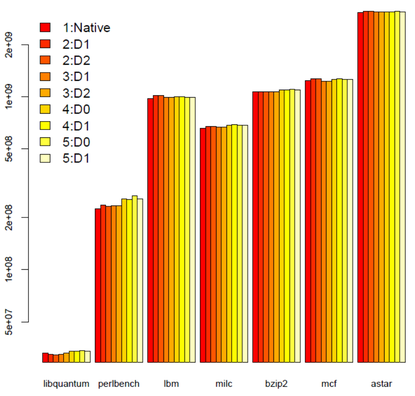

Figure 12. SPEC CPU2006: Relative error for L2 cache references

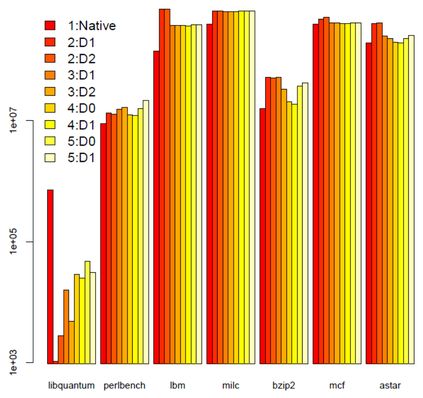

Figure 13. SPEC CPU2006: Relative error for L2 cache misses

Contributions

- We developed a framework for performance counter virtualization which is fully compatible with widely used perfctr library. Our framework is very efficient, accounts for overhead caused by hypercalls
- We have shown the validity of our frameworks using a number of different applications such as SPEC CPU 2006 and HPCToolkit profiler
- Our framework is available under an open source license at http://people.cs.vt.edu/~rnikola

Component	Number of lines	Details
Perfctr	563	VCPU support, hypervisor communication, etc.
Linux	36	shared info management, VIRQ PERFCTR
Xen	3488	perfctr-xen, shared info management, VIRQ PERFCTR

Table 3. Added or Modified code

This work was supported by NSF Award CSR #0720673

The work is published in the Proceedings of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE 2011)