Perfctr-Xen: A framework for Performance Counter Virtualization

Ruslan Nikolaev and Godmar Back
Virginia Polytechnic Institute
Blacksburg
Overview

- IaaS widely use virtual machine monitors
  - Type 1 hypervisors: Xen, KVM, ESX ...
- 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
- Our contribution: Perfctr-Xen:
  - Framework for performance counter virtualization
  - Software-compatible with widely used perfctr library
  - Techniques for collaboration of guest and hypervisor
- Experimental validation
Existing Performance Counter Virtualization Solutions

- XenoProf
  - Extension of Oprofile system-wide profiler
  - Does not provide per-domain abstraction of hardware counter facilities (supports only 1 domain at a time)

- VPMU driver
  - Treats PMU registers like ordinary registers (saved/restored by VMM)
  - Requires use of hardware assisted virtualization mode; support for limited number of architecture generations since VMM must contain architecture-specific code
  - Not compatible with all architectures
Perfctr-Xen

- Perfctr (Native)

- Profilers
  - PerfExplorer, HPCToolkit, etc.

- High-level performance counters
  - PAPI

- Low-level performance counters
  - Perfctr Lib

- Kernel
  - Perfctr Driver
Perfctr-Xen

- Perfctr (Native)

  PerfExplorer, HPCToolkit, etc.

  PAPI

  Perfctr Lib

  Perfctr Driver

  Kernel

- Perfctr-Xen

  Profilers

  High-level performance counters

  Low-level performance counters

  Guest Kernel

  Xen Hypervisor
Perfctr-Xen

- Perfctr (Native)
  - PerfExplorer, HPCToolkit, etc.
  - PAPI
  - Perfctr Lib
  - Perfctr Guest Driver
  - Profilers
    - High-level performance counters
  - Low-level performance counters

- Perfctr-Xen
  - Guest Kernel
  - Xen Hypervisor
**Perfctr-Xen**

- **Perfctr (Native)**
  - PerfExplorer, HPCToolkit, etc.
  - PAPI
  - Perfctr Lib
  - Perfctr Driver

- **Perfctr-Xen**
  - Profilers
  - High-level performance counters
  - Low-level performance counters
  - Perfctr Guest Driver
  - Guest Kernel
  - Xen Hypervisor
**Perfctr-Xen**

- **Perfctr (Native)**
  - PerfExplorer, HPCToolkit, etc.
  - PAPI
  - Perfctr Lib
  - Perfctr-Xen Driver

- **Perfctr-Xen**
  - Profilers
    - High-level performance counters
  - Low-level performance counters
  - Perfctr Guest Driver
  - Guest Kernel
  - Xen Hypervisor
Perfctr-Xen

- **Perfctr (Native)**
  - PerfExplorer, HPCToolkit, etc.
  - PAPI
  - Perfctr Lib
  - Perfctr Driver

- **Perfctr-Xen**
  - Profilers
  - High-level performance counters
  - Low-level performance counters
  - Perfctr Guest Driver
  - Guest Kernel
  - Perfctr-Xen Driver
  - Xen Hypervisor
Perfctr-Xen

- Perfctr (Native)
  - Perfctr Lib
  - PAPI
  - PerfExplorer, HPCToolkit, etc.

- Profilers
- High-level performance counters
- Low-level performance counters

- Perfctr Driver
- Guest Kernel
- Xen Hypervisor
- Perfctr Guest Driver

- Perfctr-Xen
Perfctr-Xen

- Perfctr (Native)
  - PerfExplorer, HPCToolkit, etc.
  - PAPI
  - Perfctr Lib
  - Perfctr Driver
  - Kernel

- Profilers
  - High-level performance counters
  - Low-level performance counters

- Perfctr-Xen
  - Perfctr Lib
  - Perfctr Guest Driver
  - Guest Kernel
  - Perfctr-Xen Driver
  - Xen Hypervisor
Perfctr-Xen

- Perfctr (Native)
  - PerfExplorer, HPCToolkit, etc.
  - PAPI
  - Perfctr Lib
  - Perfctr Driver

- Profilers
  - High-level performance counters
  - Low-level performance counters

- Perfctr-Xen
  - PerfExplorer, HPCToolkit, etc.
  - PAPI
  - Perfctr Lib
  - Perfctr Guest Driver
  - Guest Kernel
  - Perfctr-Xen Driver
  - Xen Hypervisor

Software Compatible
Per-thread PMU Virtualization

Logical per-thread value includes only events incurred during the thread execution
Per-thread PMU Virtualization

Logical per-thread value includes only events incurred during the thread execution
Logical per-thread value includes only events incurred during the thread execution.
Per-thread PMU Virtualization

Logical per-thread value includes only events incurred during the thread execution.
Per-thread PMU Virtualization

Logical per-thread value includes only events incurred during the thread execution
Per-thread PMU Virtualization

Logical per-thread value includes only events incurred during the thread execution

- **Intradomain Switch**
  - Thread 0
  - Thread 1
  - Domain 0

- **Interdomain Switch**
  - Thread 0
  - Thread 1
  - Domain 1
Per-thread PMU Virtualization

Logical per-thread value includes only events incurred during the thread execution
Perfctr Library

Modes of operation

- A-mode: an event count in some region of a program
- I-mode: an interrupt after a certain number of events has occurred
Perfctr: A-mode counters

Thread 0

Thread 1
Perfctr: A-mode counters

$Sum_{thread}$ records accumulated event count
Perfctr: A-mode counters

$\sum_{thread}$ records accumulated event count
Kernel records physical value to $\text{Start}_{thread} = \text{Phys}(t_1)$
Perfctr: A-mode counters

$Sum_{thread}$ records accumulated event count
Kernel records physical value to $Start_{thread} = \text{Phys}(t_1)$
Perfctr: A-mode counters

$Sum_{thread}$ records accumulated event count

Kernel records physical value to $Start_{thread} = \text{Phys}(t_1)$

Thread samples physical value $\text{Phys}(t_2)$, computes Logical value $Log_{thread} = Sum_{thread} + (\text{Phys}(t_2) - Start_{thread})$

Thread 0

$Sum_{thread}$

Thread 1

$Phys(t_2) - Start_{thread}$
Perfctr: A-mode counters

\( \text{Sum}_{\text{thread}} \) records accumulated event count

Kernel records physical value to \( \text{Start}_{\text{thread}} = \text{Phys}(t_1) \)

Thread samples physical value \( \text{Phys}(t_2) \), computes

Logical value \( \text{Log}_{\text{thread}} = \text{Sum}_{\text{thread}} + (\text{Phys}(t_2) - \text{Start}_{\text{thread}}) \)
Perfctr: A-mode counters

$Sum_{thread}$ records accumulated event count

Kernel records physical value to $Start_{thread} = \text{Phys}(t_1)$

Thread samples physical value $\text{Phys}(t_2)$, computes Logical value $Log_{thread} = Sum_{thread} + (\text{Phys}(t_2) - Start_{thread})$

Kernel increments $Sum_{thread} = Sum_{thread} + (\text{Phys}(t_3) - Start_{thread})$

Thread 0

$t_1$

$t_2$

$t_3$

$Phys(t_3) - Start_{thread}$

Thread 1
Perfctr: I-mode counters

- PMU registers trigger interrupt on zero-overflow
- Physical register initialized to negated sample period
- Requires that physical value be saved & restored on each context switch
- Compute logical accumulated value similar to a-mode
Perfctr-Xen: A-mode counters

- Requires cooperation of guest kernel and hypervisor:
  - Guest: maintains per-thread state: $\text{Sum}_{\text{thread}}$, $\text{Start}_{\text{thread}}$
  - Hypervisor: a per-VCPU (Virtual CPU) state: $\text{Sum}_{\text{vcpu}}$, $\text{Start}_{\text{vcpu}}$
- Guest kernel makes per-VCPU state available user threads
Perfctr-Xen: A-mode counters

Thread 0

Thread 1

Domain 0

Thread 0

Thread 0

Domain 1

Thread 1
Perfctr-Xen: A-mode counters

![Diagram of A-mode counters for Thread 0 and Thread 1 across Domain 0 and Domain 1 with a sum of thread counts labeled as $\text{Sum}_{\text{thread}}$]
Perfctr-Xen: A-mode counters
Perfctr-Xen: A-mode counters

Thread 0

Thread 1

Sum\text{thread}

Sum\text{vcpu}

Domain 0

Thread 0

Thread 1

Domain 1
Perfctr-Xen: A-mode counters

Start_{vcpu} = Phys(t_1)
Perfctr-Xen: A-mode counters

\( \text{Start}_{vcpu} = \text{Phys}(t_1) \)

Thread 0
- \( \text{Sum}_{thread} \)
- \( \text{Sum}_{vcpu} \)
- \( \text{Phys}(t_2) - \text{Start}_{vcpu} \)

Thread 1

Thread 0

Thread 1

Domain 0

Domain 1
Perfctr-Xen: A-mode counters
Perfctr-Xen: A-mode counters

Thread 0

Thread 1

Domain 0

Thread 0

Thread 1

Domain 1
Perfctr-Xen: A-mode counters
Perfctr-Xen: A-mode counters

Hypervisor: \( \text{Sum}_{\text{vcpu}} = 0, \text{Start}_{\text{vcpu}} = \text{Phys}(t_1) \)
Perfctr-Xen: A-mode counters

Hypervisor: $\text{Sum}_{\text{vcpu}} = 0$, $\text{Start}_{\text{vcpu}} = \text{Phys}(t_1)$
Perfctr-Xen: A-mode counters

Hypervisor: $\text{Sum}_{vcpu} = 0$, $\text{Start}_{vcpu} = \text{Phys}(t_1)$
Guest: $\text{Start}_{thread}^* = \text{Sum}_{vcpu} + (\text{Phys}(t_2) - \text{Start}_{vcpu})$

Hypercall:
- Activate configuration
- $\text{Sum}_{vcpu} \leftarrow 0$
- $\text{Start}_{vcpu} \leftarrow \text{Phys}(t_1)$
Perfctr-Xen: A-mode counters

Hypervisor: $\text{Sum}_{vcpu} = 0$, $\text{Start}_{vcpu} = \text{Phys}(t_1)$

Guest: $\text{Start*}_{thread} = \text{Sum}_{vcpu} + (\text{Phys}(t_2) - \text{Start}_{vcpu})$

Hypervisor: $\text{Sum}_{vcpu} = \text{Sum}_{vcpu} + (\text{Phys}(t_3) - \text{Start}_{vcpu})$

Hypercall:
- Activate configuration
- $\text{Sum}_{vcpu} \leftarrow 0$
- $\text{Start}_{vcpu} \leftarrow \text{Phys}(t_1)$
Perfctr-Xen: A-mode counters

Hypervisor: $\text{Sum}_{\text{vcpu}} = 0,$ $\text{Start}_{\text{vcpu}} = \text{Phys}(t_1)$
Guest: $\text{Start}^{*}_{\text{thread}} = \text{Sum}_{\text{vcpu}} + (\text{Phys}(t_2) - \text{Start}_{\text{vcpu}})$
Hypervisor: $\text{Sum}_{\text{vcpu}} = \text{Sum}_{\text{vcpu}} + (\text{Phys}(t_3) - \text{Start}_{\text{vcpu}})$
Hypervisor: $\text{Start}_{\text{vcpu}} = \text{Phys}(t_4)$
Perfctr-Xen: A-mode counters

Hypervisor: $\text{Sum}_{\text{vcpu}} = 0$, $\text{Start}_{\text{vcpu}} = \text{Phys}(t_1)$

Guest: $\text{Start}_{\text{thread}} = \text{Sum}_{\text{vcpu}} + (\text{Phys}(t_2) - \text{Start}_{\text{vcpu}})$

Hypervisor: $\text{Sum}_{\text{vcpu}} = \text{Sum}_{\text{vcpu}} + (\text{Phys}(t_3) - \text{Start}_{\text{vcpu}})$

Hypervisor: $\text{Start}_{\text{vcpu}} = \text{Phys}(t_4)$

$\text{Log}_{\text{thread}} = \text{Sum}_{\text{thread}} + \text{Sum}_{\text{vcpu}} + (\text{Phys}(t_5) - \text{Start}_{\text{vcpu}}) - \text{Start}_{\text{thread}}$
Perfctr-Xen: I-mode counters

- Suspension hypercall to increment $\text{Sum}_{vcpu}$ and sample $\text{Start}_{vcpu}$
- Resumption hypercall to restore per-VCPU values

$$\log_{thread} = \text{Sum}_{vcpu} + (\text{Phys}(t) - \text{Start}_{vcpu})$$
Perfctr-Xen: Interrupt delivery

- Hypervisor delivers overflow interrupts to guest via VIRQ_PERFCTR virtual interrupts
- Upon receipt, guest kernel signals user thread
- Virtual interrupts are delivered asynchronously (as soft interrupts)
- Guest must ensure that overflow interrupt is delivered to correct thread by rechecking overflow status
  - If thread causing overflow is suspended before virtual interrupt arrives at guest, mark as pending and deliver on next resume
Experimental Results

- Baseline: native execution
- Exercise multiple VCPU/PCPU scenarios
- Exercise multiple virtualization modes
  - Paravirtualization
  - Hardware-assisted virtualization (HVM)
  - Hybrid mode (HVM + guest enhancement)
- Correctness of implementation and accuracy of results
  - Microbenchmarks for a-mode, PAPI test for i-mode
  - Macrobenchmarks: SPEC CPU 2006
  - Verify Profiling (HPCToolkit)
Microbenchmarks

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
SPEC CPU2006: L2 Cache Misses

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
SPEC CPU2006: L2 Cache References

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
Related Work

- Performance counter support for VMM
  - XenoProf [Menon 2005]
  - Counter Virtualization for KVM [Du 2010, 2011]
  - VTSS++ system [Bratanov 2009]
- Performance counters in non-virtualized systems
  - perf_counter, Perfmon [Eranian 2006], Intel VTune, AMD Code Analyst
- Higher-level libraries:
  - PAPI [Browne 1999]
Conclusion

• PerfCtr-Xen
  • Efficient and accurate per-thread virtualization of hardware event counters
  • Supports all commonly used virtualization modes
  • Plug-in Compatibility with PAPI, HPCToolkit, etc.
  • Techniques extend to other Type I hypervisors and low-level virtualization libraries

• Available at http://people.cs.vt.edu/~rnikola/ (LGPL license)