



# Extending I/O scalability in Xen

Dong Eddie, Zhang Xiantao, Xu Dongxiao, Yang Xiaowei

Xen Summit Asia 2009



# Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel may make changes to specifications and product descriptions at any time, without notice.

All products, computer systems, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

Intel is a trademark of Intel Corporation in the U.S. and other countries.

\*Other names and brands may be claimed as the property of others.

Copyright © 2009, Intel Corporation. All rights are protected.



# Agenda

- Scalability challenges of I/O virtualization
- VNIF optimizations
- VT-d overhead reduction
- SR-IOV
- Per CPU vector

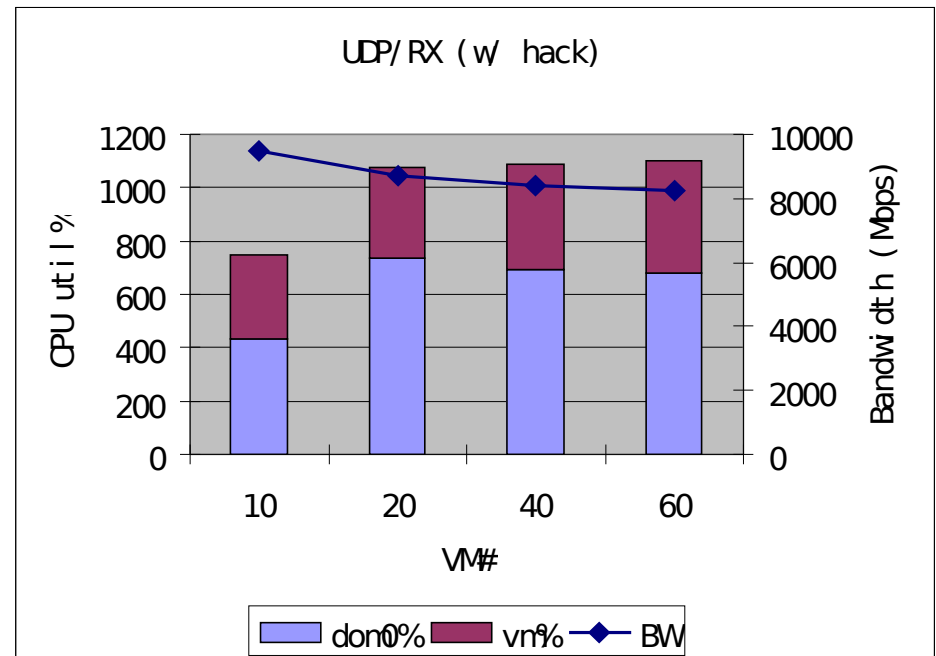
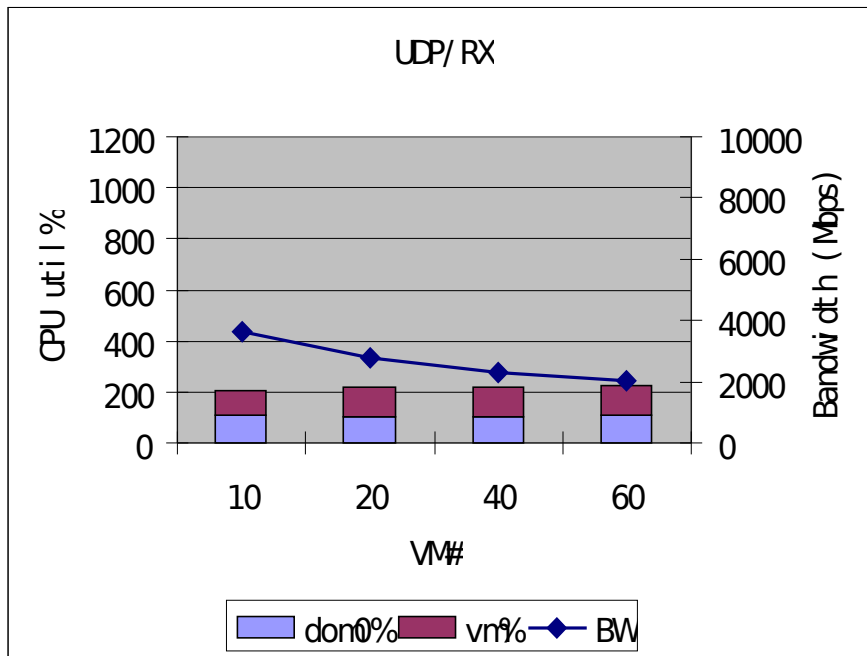
# Scalability challenges of I/O virtualization

- Para-virtualized device driver
  - Software bottleneck; high CPU utilization
- Direct I/O
  - Interrupt overhead; device# limit
- SR-IOV
  - Driver not optimal
- Limitation inside VMM

# VNIF optimization

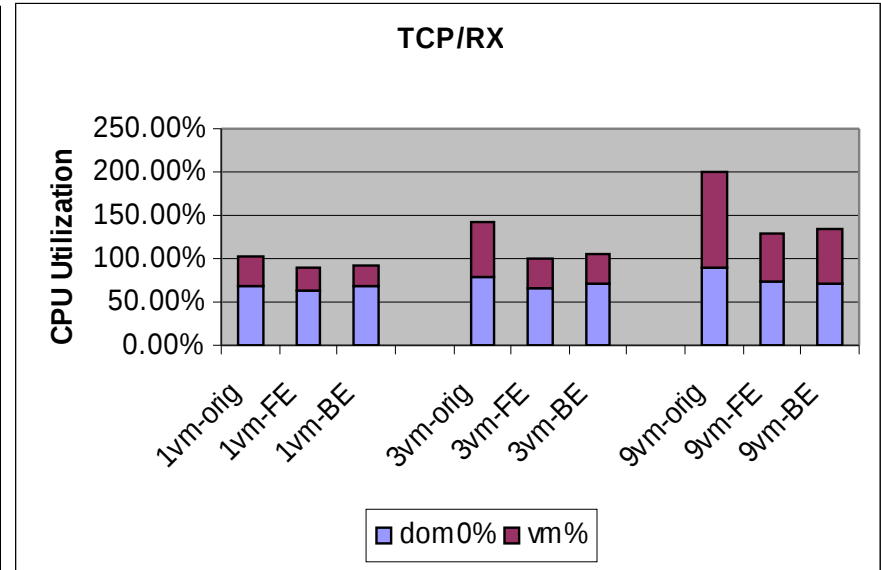
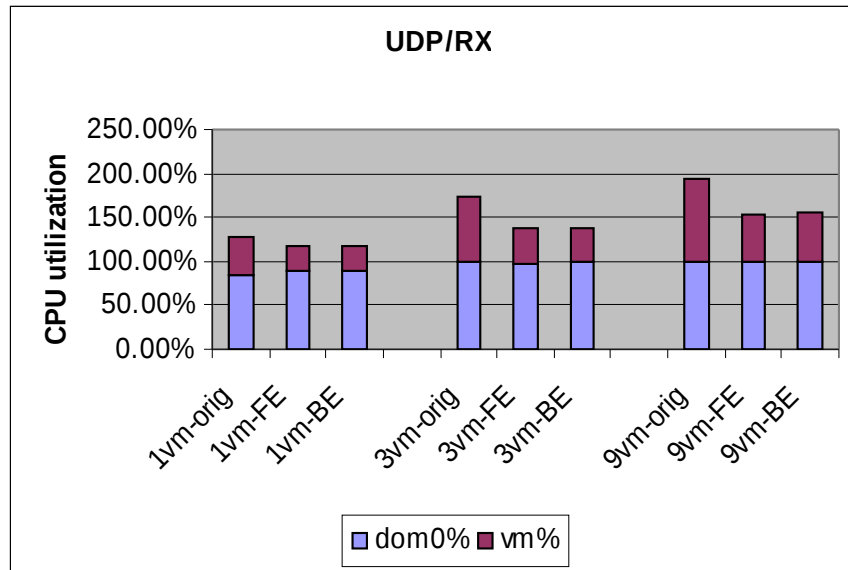


# Improving scalability in 10G network



- **Issue:** the total throughput is limited by the bottleneck of netback driver in dom0 - only one TX/RX tasklet serves all netfront interfaces.
- **Hack:** duplicate 10 netback drivers, each serves VNIFs of domX (X=domid%10).
- **Benefit:** reach near 10G throughput, with the cost of high CPU util%
- **Ongoing:** multiple TX/RX tasklets

# Notification frequency reduction

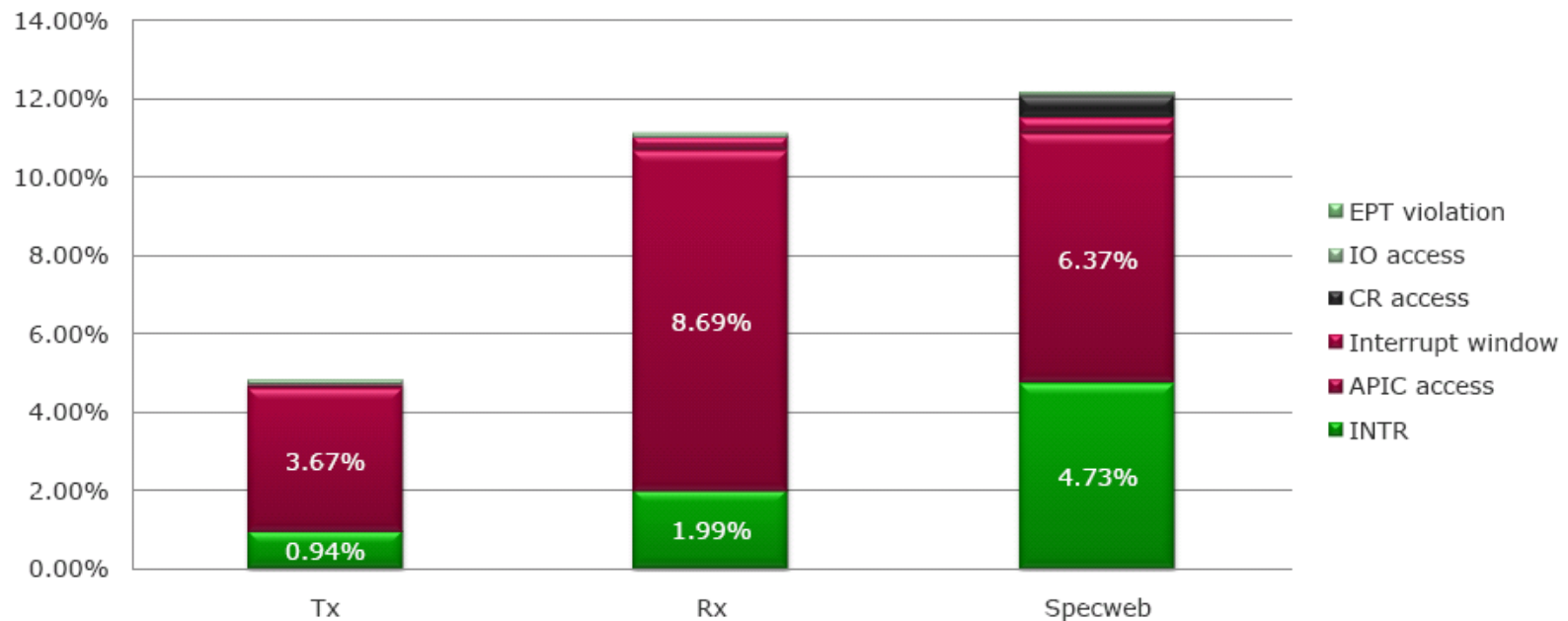


- **Issue:** Evtchn freq  $\approx$  physical NIC's intr freq \* n (n: VNIF# sharing the NIC).  
But per our test, 1K HZ evtchn freq can sustain 1G throughput (TCP/UDP) already.
- **Solution:** add evtchn freq controlling policy inside 1) netback or 2) netfront, and interface to userspace.
- **Benefit:** guests' CPU util% reduction.

# VT-d overhead reduction



# Network virtualization overhead with Direct I/O - 10G NIC



\* '%' here means system wide CPU utilization normalized to 100%; 4 CPUs are used in the test.

- **Interrupt frequency is very high in 10G network:** 4K/8K HZ for each TX/RX queue; 8 TX/RX queues in total;
- APIC access VMExit causes the most overhead, within which EOI access occupies 90%/50% for TX/RX case.
- Interrupt (including ext. intr and IPI) VMExit causes the 2<sup>nd</sup> most overhead, within which IPI occupies >50%.

# APIC access VMExit optimization - vEOI

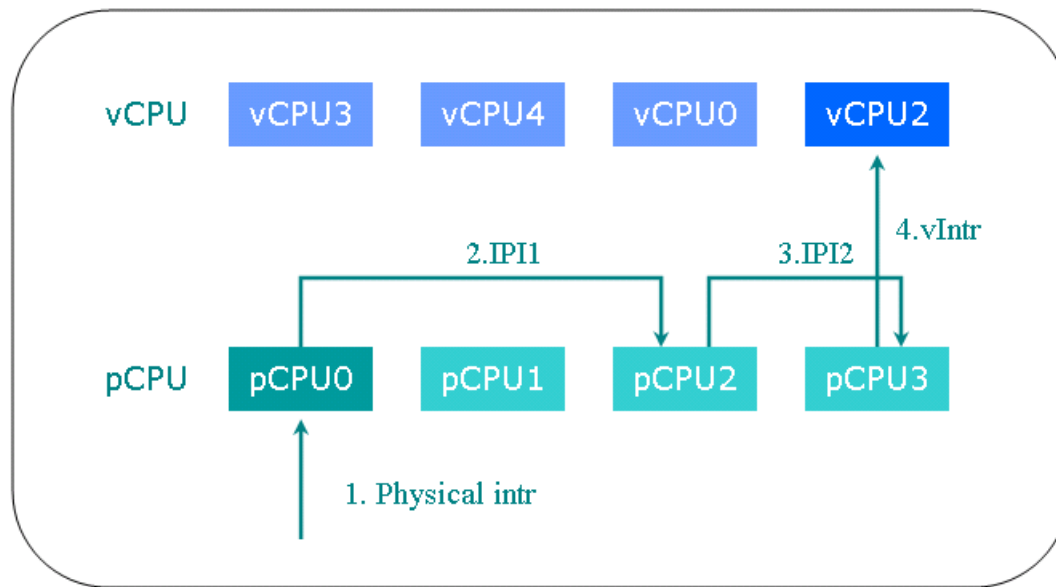


APIC access VMExit handling stages

- EOI's usage
  - S/W writes "0" into EOI for signaling interrupt servicing completion;
  - Upon receiving a EOI, LAPIC clears the highest priority bit in the ISR and dispatches the next priority interrupt.
- Majority OSes (Windows and Linux) only use 'mov' for this purpose, which has no side effect to other CPU states
- **Benefit:** by bypassing instruction fetch/emulation stage, each vEOI's cost decreases from 7.6k to 3.1k cycles, CPU util% can decrease by 15+% ( $12k * 8 * 4.5k / 2.8G$ ) in 10G NIC case.

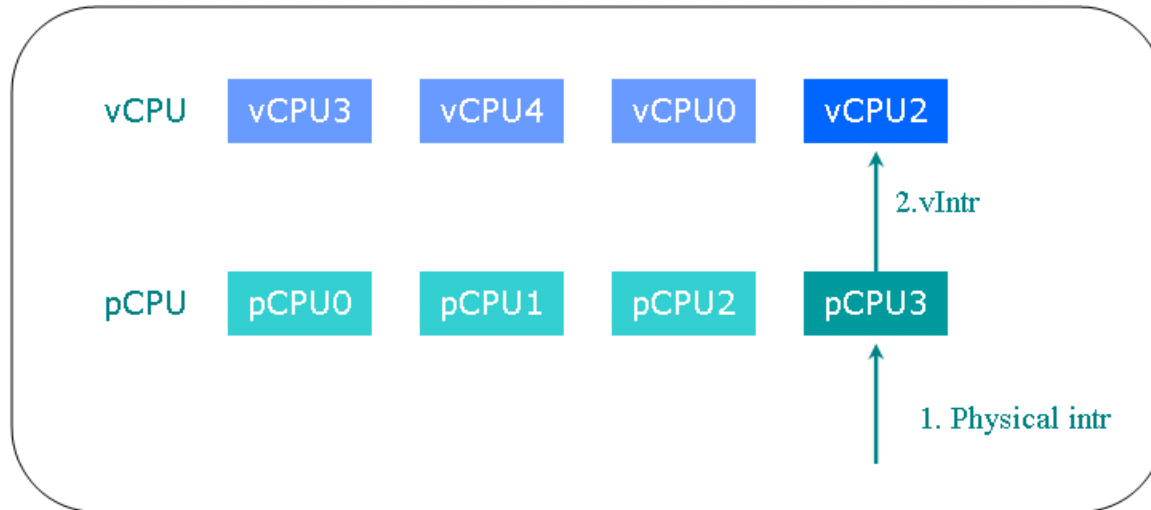
If guests use complex instructions (e.g. stos) for EOI access, there will be side effects to guests, but immune to host.  
Better / complete solution is PV or virtualizing x2APIC (using MSR).

# vIntr delivering - before optimization



0. VT-d device's plntr affinity is set to where vCPU0 is on (pCPU0) at boot time; guest OS sets vCPU2 to receive the vIntr.
1. When the device generates the plntr, it's delivered to pCPU0
2. Xen sends IPI to where vCPU0 is on (pCPU2) for vIntr delivery
3. Before vCPU0 VMentry, Xen delivers vIntr by sending IPI to where vCPU2 is on (pCPU3).
4. On vCPU2 VMentry, vIntr is injected to the guest.

# vIntr delivering - after optimization



0. Most works are done before the plntr happens

\* When guest OS sets vCPU2 to receive the vIntr, Xen sets the corresponding pCPU (pCPU3) to receive the plntr. \* When vCPU2 migrates, the plntr migrates with it.

1. When the device generates the plntr, it's delivered to pCPU3 directly

2. On vCPU2 VMentry, vIntr is injected to the guest.

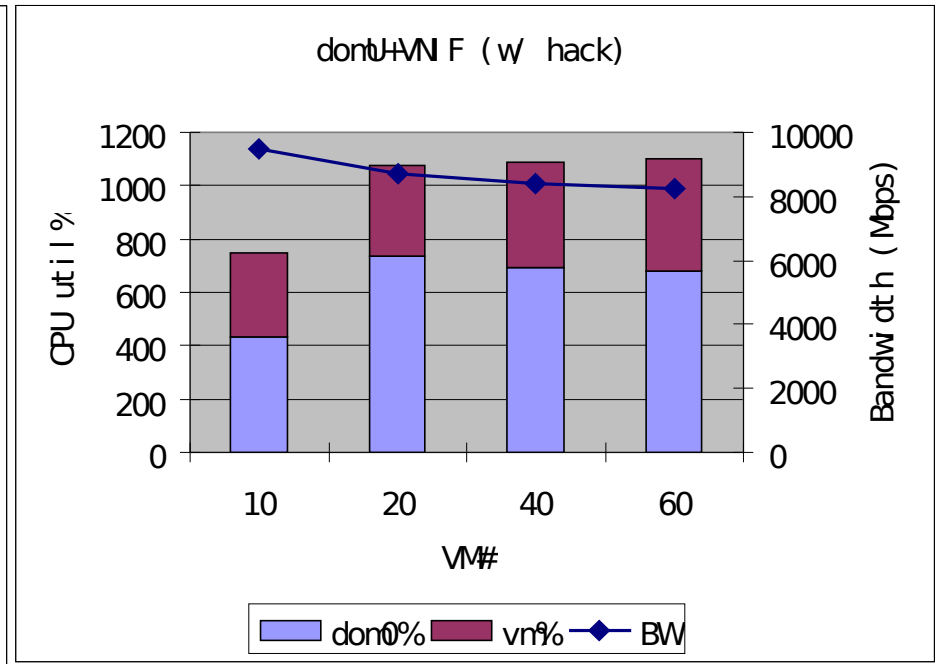
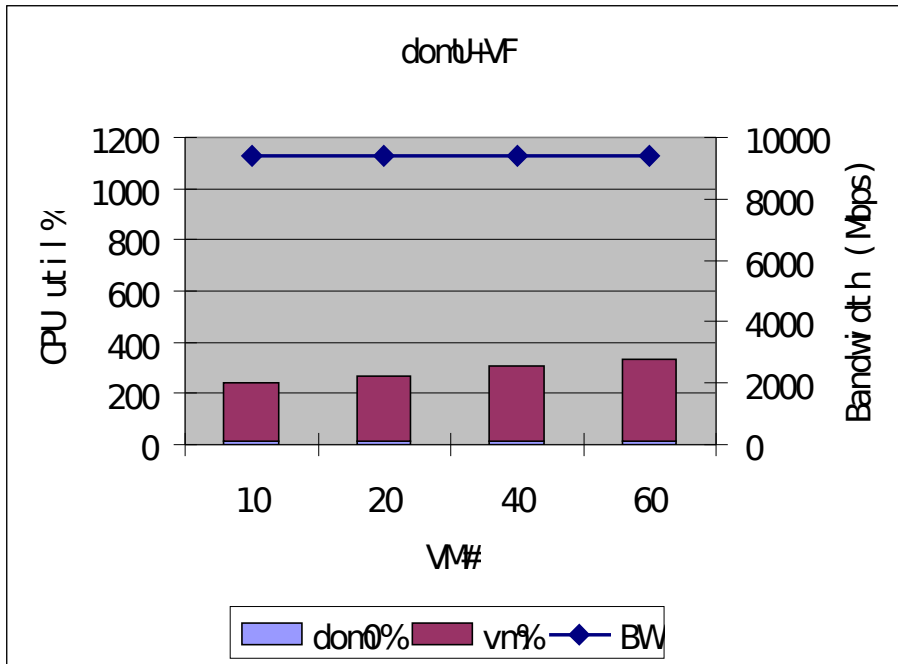
**Benefit:** all IPIs for vIntr delivery (~4%-9% CPU overhead) are removed.

**Limit:** for simplicity, our 1<sup>st</sup> patch only handle MSI (no ioapic) and single vIntr destination case, which covers typical usage.

# SR-IOV



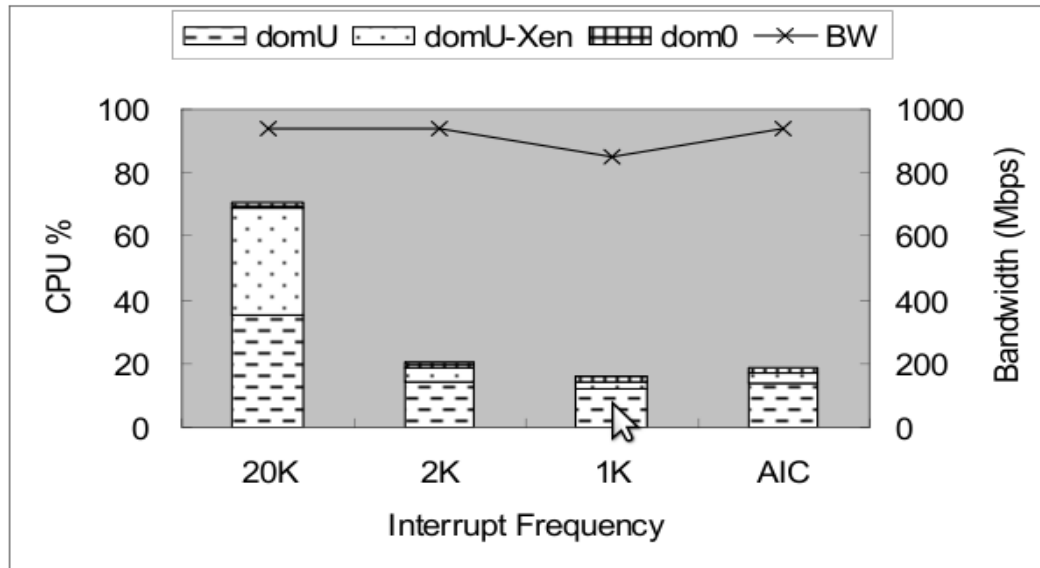
# SRIOV v.s. VNIF scalability



• SRIOV has significant advantage over VNIF solution even the tasklet bottleneck is fixed, in the aspects of

- Less CPU utilization
- Stable latency

# Interrupt coalescing optimization



- Interrupt coalescing is more critical to VM than to native
  - Interrupt handling inside VM has more overhead.
- Interrupt coalescing policy in native is not efficient in SRIOV, as the BW of one VF can be
  - << line speed, e.g. several VFs within one physical port competing at the same time.
  - >> line speed, e.g. inter-VF communication with one port.
- Proposal: adaptive interrupt coalescing (AIC), based on the packet generating rate in the last seconds and buffer size in the system.

# Per CPU vector



# Issues

- Interrupt vector in Xen was global
  - One vector number was shared by all pCPUS
- Only <200 vectors were available for devices in total
  - Vector range is 0-255
  - Lowest 32 vectors reserved by x86 SDM
  - 16 vectors reserved for legacy PIC high priority cases
  - Highest 16 vectors reserved for high priority cases
  - Special case: 0x82
- In SR-IOV case, vectors are easy to be used up
  - E.g. one Niantic NIC can use 384 vectors =  $2(\text{ports}) * 64(\text{VFs+PF}) * 3(\text{TX/RX/mailbox})$ .

# Solution

- Back-port per-CPU vector solution in Linux kernel
- After the change, max vector# =  $256 * \text{pCPU\#}$
- Code changes
  - Precondition: all interrupts to be indexed by 'irq'
  - Vector allocation / management functions, related structures...
  - Interrupt migration need special care: after migration, vector# may be different.
- Evaluation
  - With per-CPU vector, vectors can be assigned to all Niantic VFs successful.