Self-Virtualized I/O: High Performance, Scalable I/O
Virtualization in Multi-core Systems
Himanshu Raj
Karsten Schwan
CERCS
Georgia Institute of Technology
Motivation

• Virtualization
  – Key technology for Resource Sharing with Strong Isolation Guarantees among multiple Guest domains
  – Hypervisors/VMMs
    • Virtualize architectural resources
    • Manage virtualized resources
  – Focus: Device Virtualization
    • Most commonly virtualized by *Driver Domains*
      – Use host CPU cycles for virtualization
      – Requires multiple domain schedules
    • Physical device used for I/O only
    • Alternative approach: Self-Virtualized devices
      – *Free up host CPU, power efficient*
Self-Virtualized Devices

• Self-Virtualized Devices
  – Smart Peripherals that Virtualize themselves
    • With minimal support from hypervisor
  – Provide API for management of virtual devices
  – Provide virtual devices that can be used directly from guest domains

• Example - Self-Virtualized NIC
  – Using IXP2400 NP based Radisys enp2611 board
    • 1 600 MHz XScale Core, 8 600 MHz Micro-Engines (RISC cores)
    • PM3386 Gb ethernet device
    • 21555 NT PCI-PCI bridge, 66MHz/64bit PCI bus
  – Provides Virtual Interfaces (VIFs)
Self-Virtualized NIC

Host

Controller Domain

Guest 0

VIF0

Guest 1

VIF1

Hypervisor

PCI Communication

Network

Mgmt

VIFs

IXP2400 NP
Virtual Interface

• Configuration
  – Two Queues – one for send, one for receive
  – PCI implications -> Send queue in IXP SDRAM, Receive queue in host memory
    • Send – PIO by host
    • Receive – PIO by IXP micro-engines
      • DMA is WIP
  – Signals via PCI interrupt
    • Single interrupt
    • Generated via 21555's doorbell register
    • De-multiplex based on VIF<->bit association by hypervisor
      • Side core processing on multi-cores
Self-Virtualized NIC – Functionalities

• Virtual Interface Management
  – Jointly done by host CPU and XScale core
  – API for
    • Creating new VIFs
    • Removing existing VIFs
    • Configuration changes

• Network I/O
  – Jointly done by host CPU and IXP micro-engines
Experimental Results

• Host Configuration
  – Dell PE2650 server, 2-way hyper-threaded Xeon 2.8GHz
  – 2GB RAM
  – Xen 3.0-unstable
  – 2.6.16 Linux para-virtualized kernel for Dom0 and DomU

• ENP2611 Configuration
  – 256 MB SDRAM, 8MB SRAM
  – 2.4.18 Linux kernel
Experimental Results – Latency

The diagram shows experimental results for latency with different total numbers of concurrent guest domains. The x-axis represents the total number of concurrent guest domains, ranging from 1 to 32. The y-axis represents latency in milliseconds, ranging from 0 to 0.5. Two different types of NIC VIFs are compared: SV-NIC VIFs (green bars) and HV-NIC VIFs (yellow bars). The error bars indicate the variability of the latency measurements.
Experimental Results – Throughput

The bar chart shows the aggregate throughput (Mbps) for different total numbers of concurrent guest domains. The chart compares SV-NIC VIFs (green bars) and HV-NIC VIFs (yellow bars). As the number of concurrent guest domains increases, the throughput for SV-NIC VIFs remains consistently higher than that for HV-NIC VIFs.
Experimental Results Summary

• Lower Latency
• Easy Exploitation of Hardware Parallelism for Higher Throughput
• Scalability
  – End-to-End performance
  – Micro-Benchmarks
Need for I/O Translations

- Limited accessibility of host RAM from IXP
  - 64 MB in current configuration (2 GB maximum)
- Current implementation: S/W IOMMU (bounce buffers)
  - Receive ring consisting of bounce buffers
  - Extra copy required by guest
- H/W IOMMU will obviate extra copy
  - Will require an interface to modify the receive ring securely (via hypercall)
    - Or runtime check by self-virtualized device
  - VT-D's Device Exclusion Vector support allows exclusive use of a physical device securely
    - How to share a device among multiple domains?
Other Architectural Considerations

• Virtual Interrupt Space
  – Must be increased for better scalability

• Tightly coupled Heterogeneous Multi-core systems are a better vehicle to implement self-virtualized devices
  – PCI interconnect limitations
    • Maybe obviated in newer systems
Ongoing Work

- **Logical Devices**
  - Export functionality specific to applications
- **QoS aware Virtual Network Interfaces**
  - Resources per VIF
  - Scheduling of micro-engines for handling multiple VIFs
- **VMM + OS Bypass for High Performance inter-application communication**
VIF Creation

1. VIF create

5. SQ, RQ, IRQ

6. Map RQ pages
7. ioremap SQ pages
8. register for IRQ

Host

Controller Domain

Management Driver

4. Grant (SQ, RQ, IRQ, guest)

Hypervisor

3. Chan ID, SQ, RQ

2. VIF channel create

ENP2611

Management Driver

VIF

Guest Domain

VIF Driver

...
Micro-Benchmarks - Ingress

Pkt size = 64 bytes
Micro-Benchmarks - Egress

![Bar chart showing performance metrics for different scenarios.](image-url)
<table>
<thead>
<tr>
<th>Num VIFS</th>
<th>Interrupt Virtualization Cost</th>
<th>Send</th>
<th>Recv</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.99uS</td>
<td>9.42uS</td>
<td>14.47uS</td>
</tr>
<tr>
<td>8</td>
<td>3.24uS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>11.57uS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Micro-Benchmarks - Throughput

![Graph showing throughput for PCI write and PCI read operations]

- **PCI write**: High throughput for both NP and host.
- **PCI read**: Lower throughput compared to PCI write.

Throughput (Mbps) vs. Operation Type:
Network I/O

• **Send**
  – Guest domain copies a packet on send queue of the VIF
  – Micro-engines poll the send queue and send packet out on physical network device

• **Receive**
  – Micro-engines receive a packet from physical device
  – The packet is classified based on MAC address and is associated with a VIF
  – The packet is copied in the receive queue of VIF
  – A signal is sent to the corresponding guest domain, *if needed*
Virtual Interface

• Configuration
  – Two Queues – one for send, one for receive
  – PCI implications -> Send queue in IXP SDRAM, Receive queue in host memory
    · Send – PIO by host
    · Receive – PIO by IXP micro-engines
      · DMA is WIP
    · Alternative design for Netronome based system
  – Signals via PCI interrupt
    · Single interrupt
    · Generated via 21555's doorbell register
    · De-multiplex based on VIF<->bit association by hypervisor
      · Side core processing on multi-cores
Network I/O

• Send managed by one micro-engine
  – One context per VIF
  – Simple RR scheduling of VIFs and contexts

• Receive managed by one micro-engine
  – All contexts work as pool of threads
Outline

• **Motivation**

• **Self-Virtualized Devices**
  – Abstraction
  – Implementation
  – Functionalities

• Experimental Results

• Architectural Considerations

• Ongoing Work