The Price of Safety: Evaluating IOMMU Performance

Muli Ben-Yehuda[1] Jimi Xenidis[2] Michal Ostrowski[2] Karl Rister[3]
Alexis Bruemmer[3] Leendert Van Doorn[4]

[1]muli@il.ibm.com
[2]{jimix,mostrows}@watson.ibm.com
[3]{krister,alexisb}@us.ibm.com
[4]Leendert.vanDoorn@amd.com

Table of Contents

- The “what” and “why” of IOMMUs.
- How much does it cost?
- What can we do about it?

Virtual Machine IO

- Virtual machines use a variety of models for IO.
- The two prevalent models are:
  - Emulation.
  - Para-virtualized drivers.
- I'm actually going to talk about the third: direct hardware access.
- I should probably mention that I'm focusing on the x86 space. . . elsewhere we do things differently. But not that differently.

Emulation and Para-virtualized Drivers

- Emulation
  - Fully-virtualized guest OS — guest is not aware of hypervisor.
  - VMware, Xen HVM and KVM.
- Para-virtualized drivers
  - Special “hypervisor aware” drivers.
  - Guest (or at least its drivers) knows it is running on top of a hypervisor.
  - VMware, Xen frontend and backend drivers and KVM’s paravirt drivers.

The Drawbacks of Virtual IO

- Requires new drivers, no support for oddball devices.
- Performance, performance, performance.

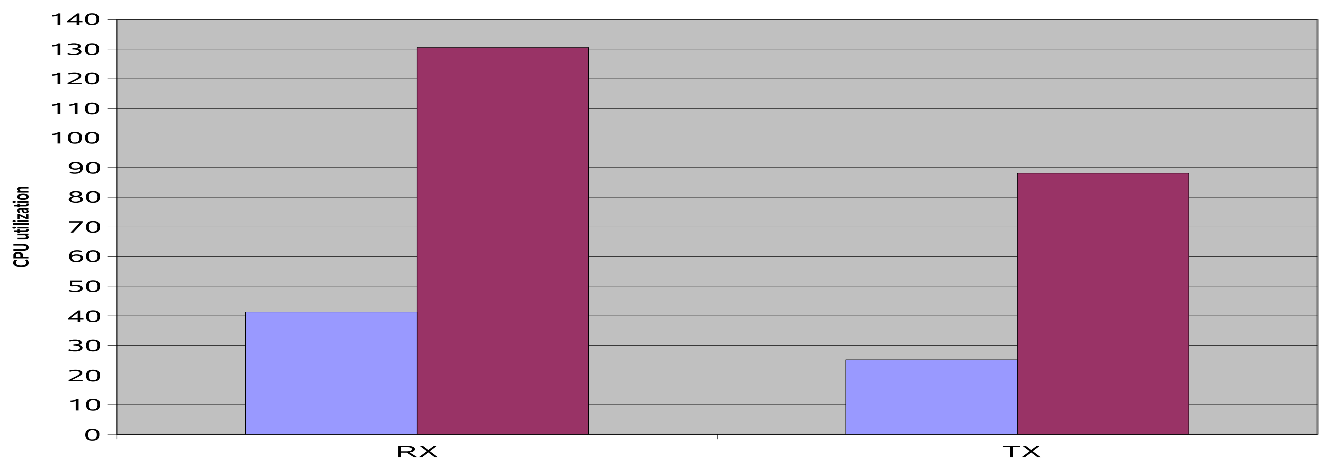


Figure 1: Xen PV drivers CPU utilization vs Linux

"Xen Network IO Perf. Analysis", Santos et al., XenSummit 2007.

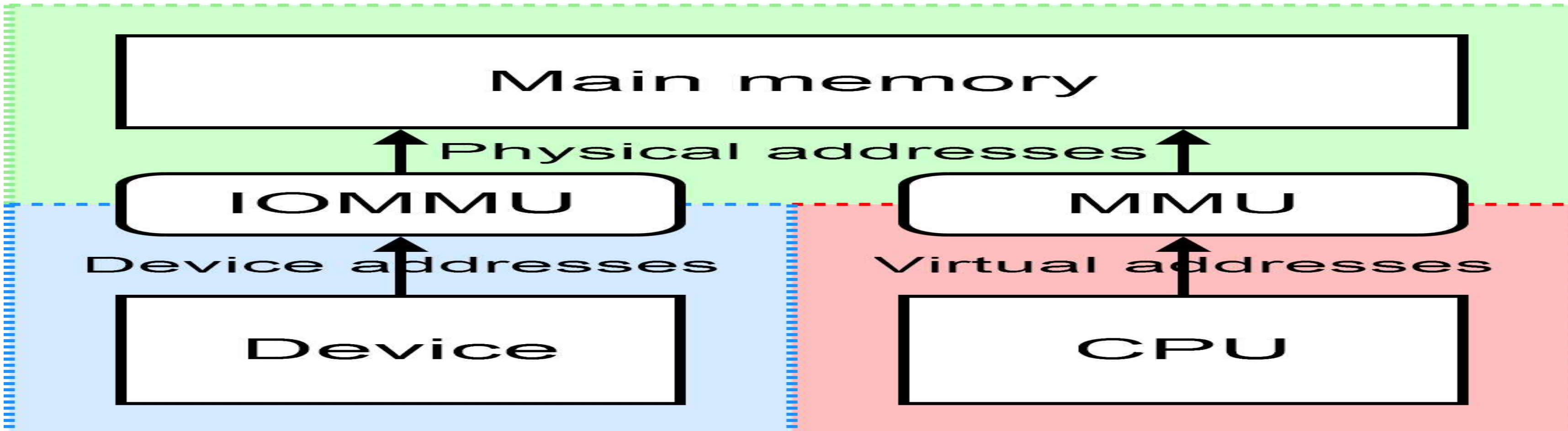
Direct Hardware Access

- Give virtual machine direct access to a hardware device.
- Without any software intermediaries between the virtual machine and the device.
- Examples:
  - Self-virtualizing adapters (including Infiniband).
  - Legacy adapters.

The Problem with Direct Access

- Untrusted domain controls a device, without any supervision.
  - That’s where direct hardware access.
- Device is DMA capable (all modern devices are).
  - Which means the domain can program the device to overwrite any memory location.
- . . . including where the hypervisor lives . . . game over.

Safe Direct Hardware Access



- IOMMU—think MMU for IO devices—separate address spaces, protection from malicious devices!
- IOMMUs enable direct hardware access for para-virtualized and fully-virtualized guests.
- IOMMUs are useful on bare-metal, too—protect the kernel from buggy drivers.

Motivation and Assumptions

- IOMMUs will be ubiquitous—Intel, AMD and the PCI-SIG are busy at work.
- Virtual IO has advantages and disadvantages.
- As IOMMUs become ubiquitous, so will direct access—where it makes sense.
- We set out to learn how will IOMMUs affect IO performance, why, and what can we do about it?

IOMMU Design

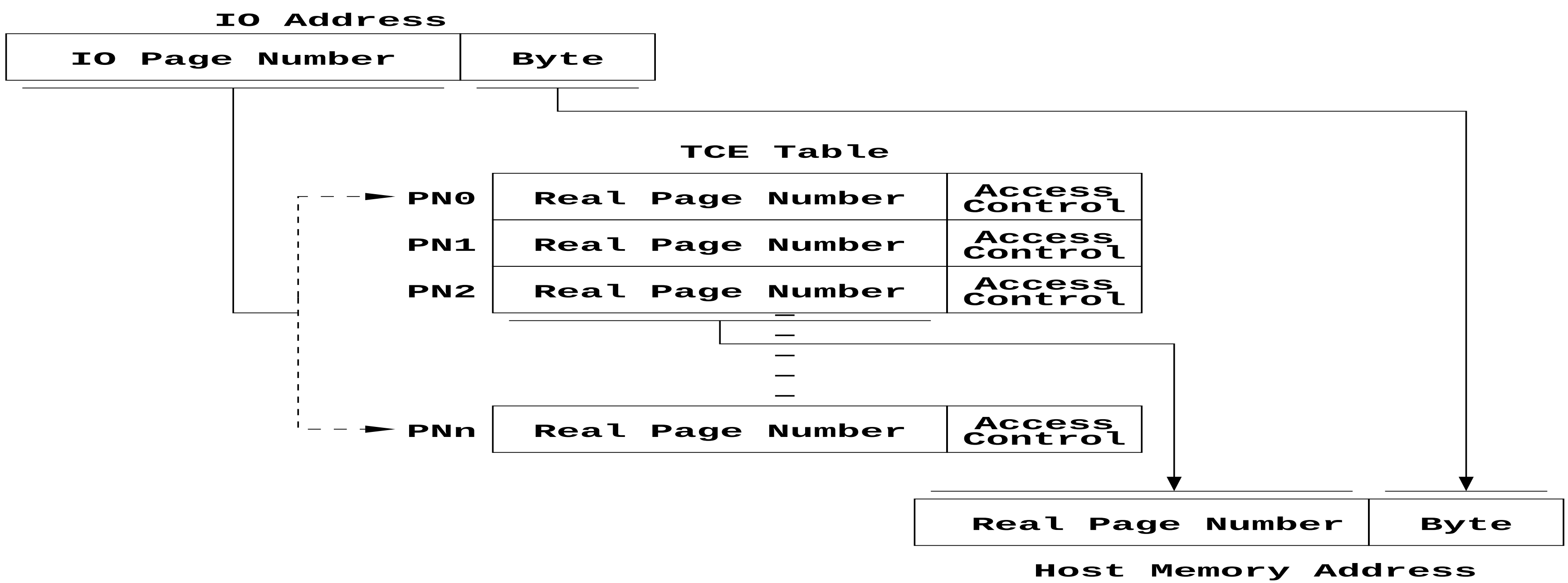
There are many different ways to build IOMMUs, and all of them affect performance:

- IOMMU design, in particular cache size, associativity and invalidation mechanisms.
- IOMMU location, core vs. chip vs. device.
- Hardware $\leftrightarrow$ software interfaces.
- Pure software interfaces (e.g., between user-space and kernel-space or between kernel-space and hypervisor).

The Calgary IOMMU

- IBM (accidentally) has an IOMMU in System x servers, based on the TCE (Translation Control Entry) family of IOMMUs.
- Calgary provides a unique I/O address space up to 4GB in size to all devices behind each PCI Host Bridge (PHB).

Calgary TCE format



Calgary uses the DMA address as an index into its IOTLB. If a translation is not found in the IOTLB, the address is used as an index a system controlled translation table in memory. If the address is not found there either, or doesn't have the right access permissions, the DMA is stopped!

Calgary Exploitation

We exploited the Calgary IOMMU to give Xen virtual machines safe direct hardware access.

- The hardware has never been validated.
- So first we did the bringup on bare-metal Linux.
- Then, we did a Xen prototype.
- In between, we made sure we get reasonable performance. . .
- . . . and supported new hardware (CalIOC2 — PCIe version of Calgary).

I'm going to tell you how it performed. . . in a bit.

Linux IOMMU Support

- Linux has a standard API for dealing with DMA memory which all well written drivers are already using, the DMA-API.
- First we cleaned up the x86-64 DMA-API implementation to support more than nommu, swiotlb and gart cleanly — the dma-ops patch.
- Then we did Calgary bringup on bare metal Linux.
- And implemented the DMA-API for Calgary on the server formerly known as “xSeries x366.”
- Despite the hardware having never been validated, it actually works.
- We had to work around a few oddities creatively - cue funny story about TCE shoot-downs.

Linux IOMMU Support continued

- Calgary support merged in 2.6.18-rc1, we're the maintainers.
- CalIOC2 (PCIe version of Calgary) to debut in 2.6.23.
- This provides an isolation capable IOMMU on System x servers - get your DMA handling wrong and the DMA will be stopped by the IOMMU with an informative message rather than corrupting memory!
- Some open issues: direct userspace access (i.e., X), graceful handling and recovery of driver errors, better integration with swiotlb, NUMA support, etc, etc. . .

dmesg in action

```
PCI-DMA: Using Calgary IOMMU
Calgary: enabling translation on PHB 0
Calgary: errant DMAs will now be prevented on this bus.
Calgary: enabling translation on PHB 1
Calgary: errant DMAs will now be prevented on this bus.
Calgary: enabling translation on PHB 2
Calgary: errant DMAs will now be prevented on this bus.
```

Xen IOMMU Support

- Main goal: using Calgary to provide direct access to devices from multiple driver domains.
- Almost there — dom0 running with Calgary enabled.
- Working on getting another driver domain running with Calgary enabled as well.
- dom0 detects Calgary in the machine — notifies hypervisor which initializes Calgary support.
- Hypervisor has a common IOMMU layer, to support Intel VT-d and AMD's IOMMU.

Xen IOMMU Support continued

- New privileged hypercalls: iommu detected, create and destroy IO space. IO spaces are identified by PCI BDF (or parts of BDF).
- Linux xen-iommu DMA-API implementation makes map and unmap hypercalls: map and unmap translation entry in IO space.

The Straight-forward Implementation

- Map Linux DMA API calls to TCE (translation control entries—think MMU PTEs) map / unmap calls.
- Straight-forward implementation.
- With the best isolation properties! Only entries in active use are mapped—minimizes window of exposure.
- Unfortunately, map / unmap hypercalls are expensive.
- . . . even on bare metal calling into the DMA API too many times hurts.
- Xen multicalls don't help.

Performance Results

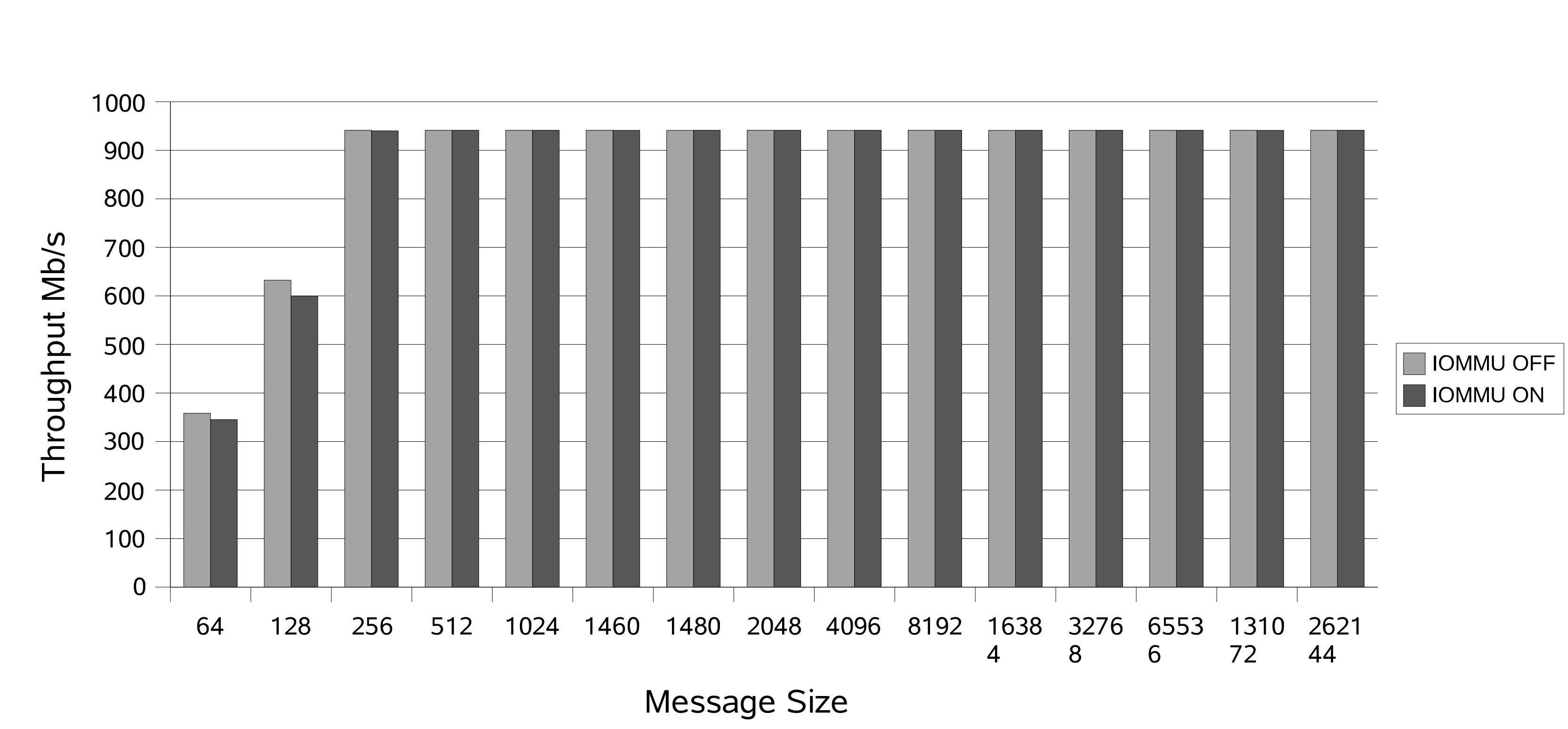
- Network (netperf) and disk IO (ffsb) tests.
- Two IOMMUs, Calgary and DART, on x86-64 and PPC, respectively.
- We only present the Calgary network results (but DART, and disk IO, were comparable).
- Always compare a given scenario:
  - With the IOMMU enabled.
  - With the IOMMU disabled.
- Not comparing with virtual IO.

Scenarios

Where is the netperf server running?

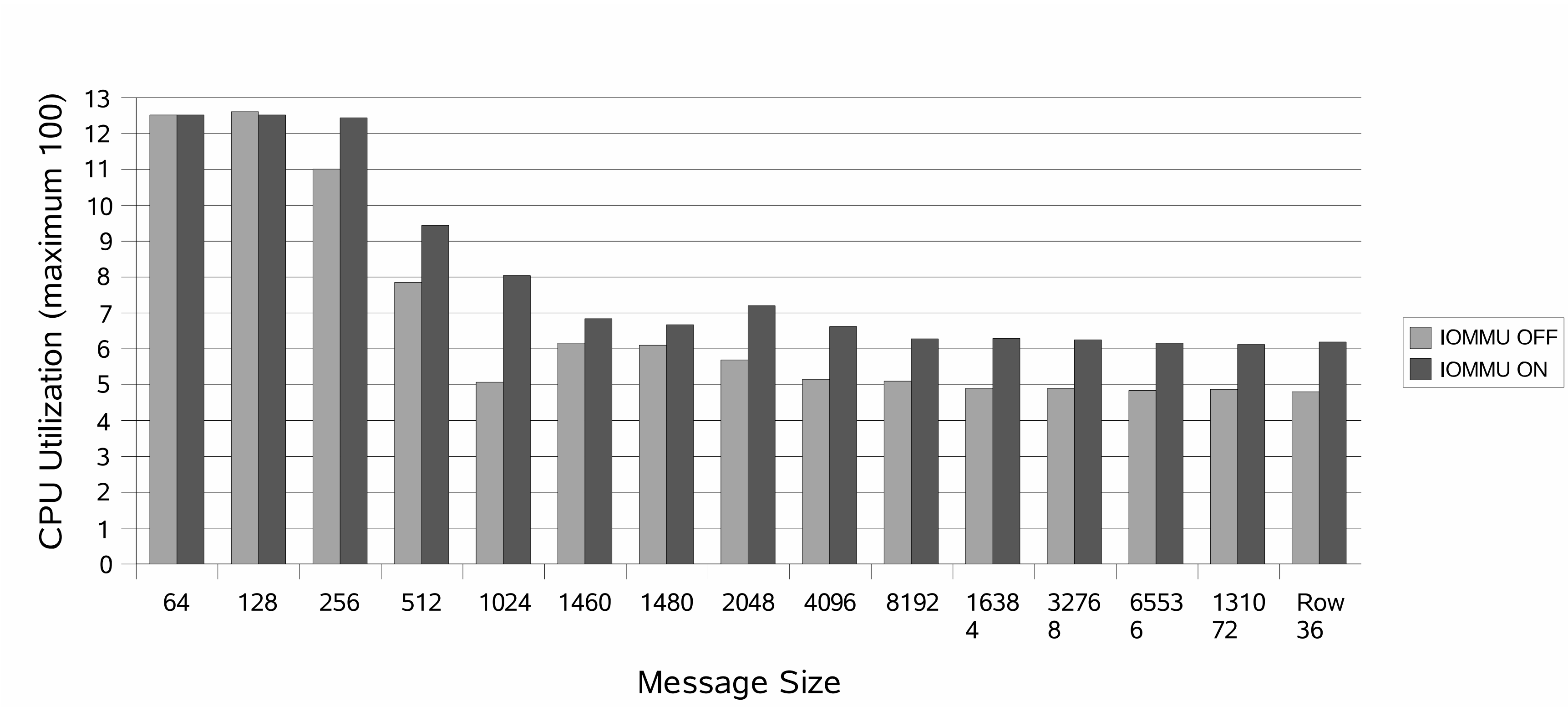
- On a bare-metal kernel.
- In Xen dom0:
  - dom0 driving the IOMMU.
  - How does the IOMMU perform for a “direct hardware access” domain?
- In Xen domU:
  - Still dom0 driving the IOMMU.
  - domU using virtual IO (netfront or blkfront).
  - How does the IOMMU perform for “‘driver domains”?

Bare-metal Network Throughput



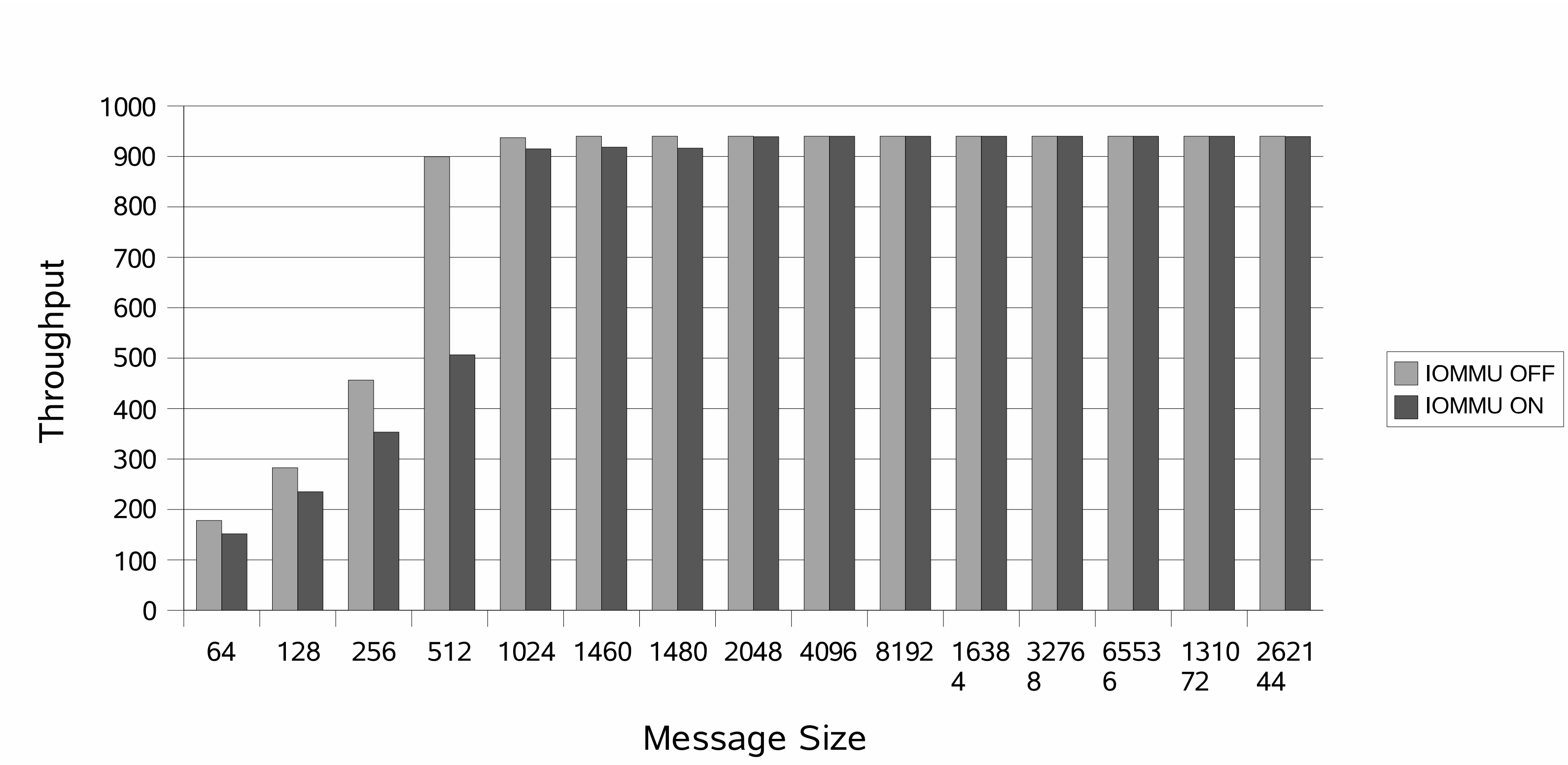
On bare-metal throughput is barely affected.

Bare-metal Network CPU Utilization



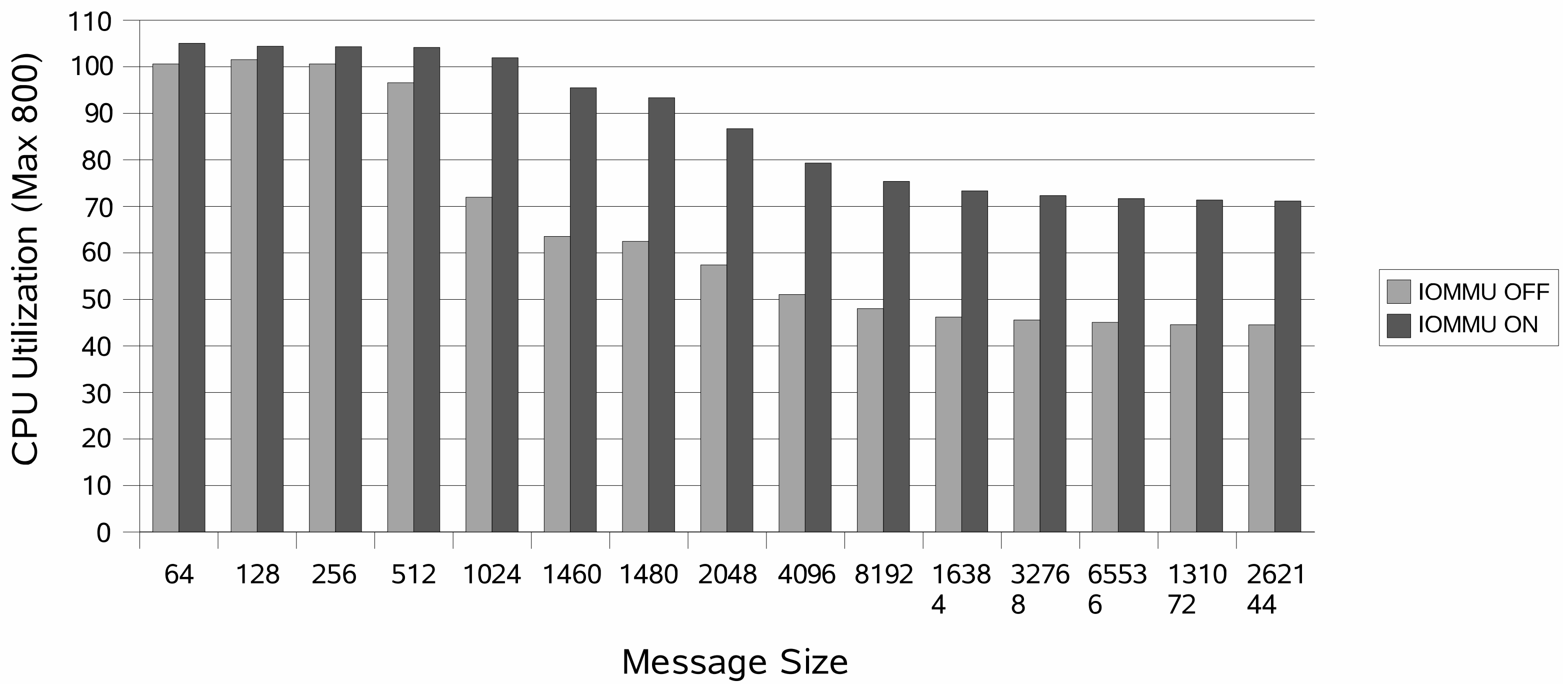
On bare-metal CPU utilization is up to 30% more!

Direct Access Network Throughput



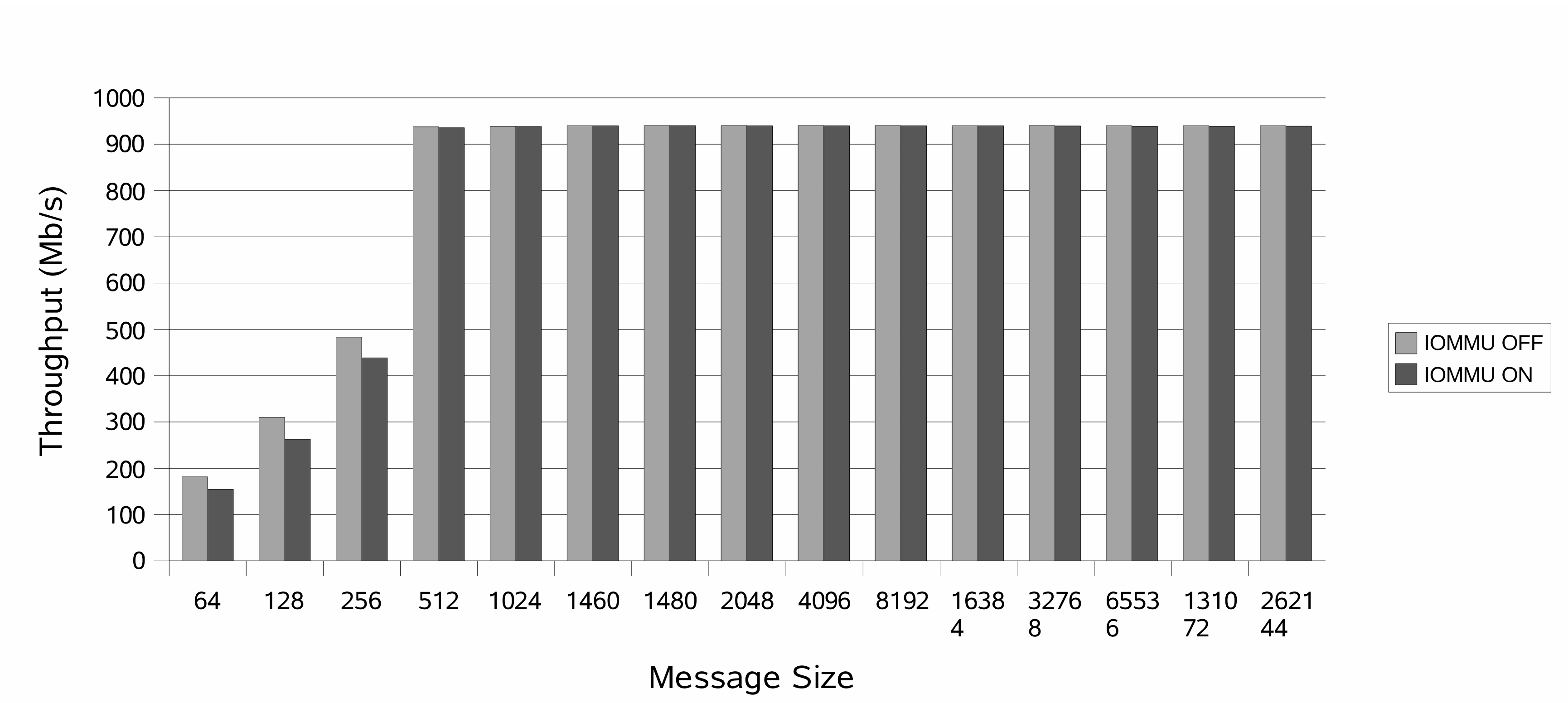
- Msg size < 1024: throughput as much as 45% less.
- Msg size >= 1024: throughput barely affected.

Direct Access Network CPU Utilization



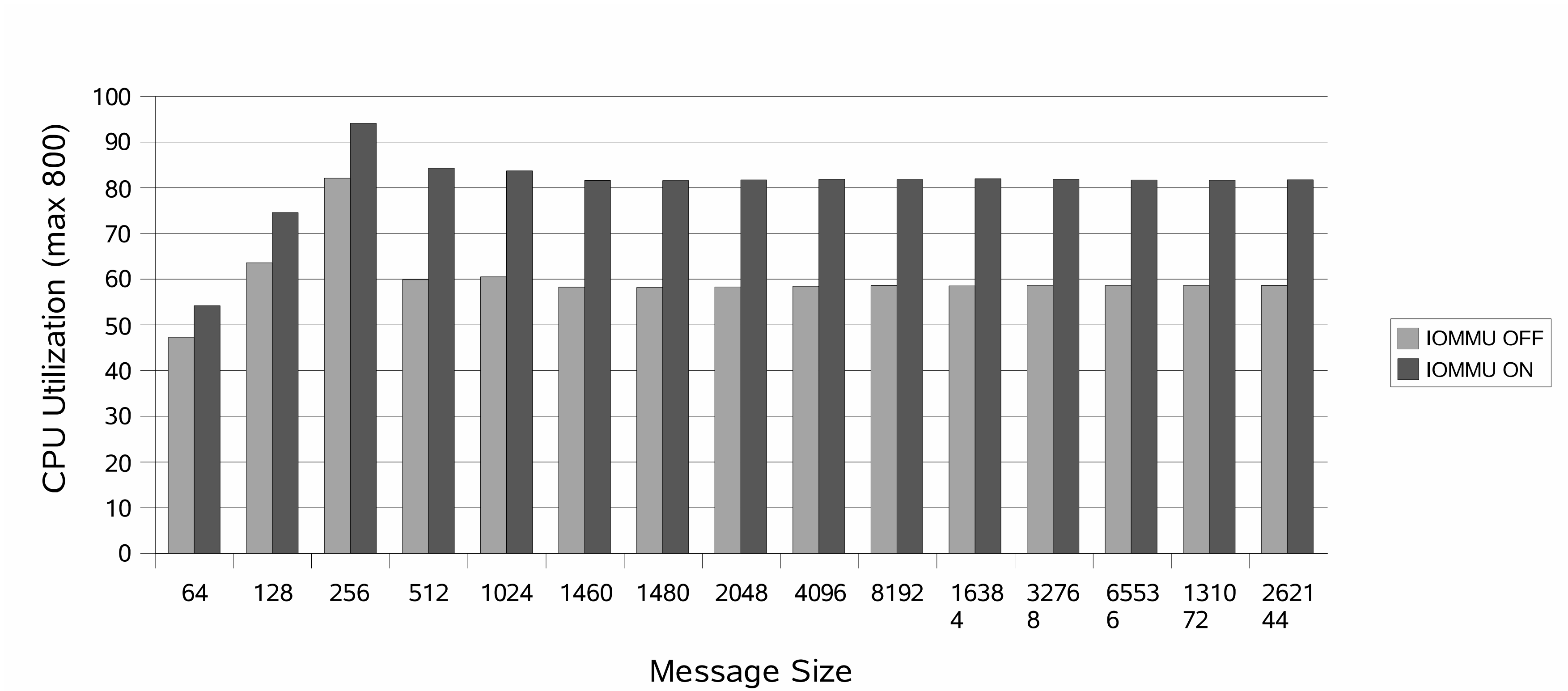
Direct access CPU utilization is up to 40%–60% more!

Driver Domain Network Throughput



- Msg size < 512: throughput as much as 15% less.
- Msg size >= 512: throughput barely affected.

Driver Domain Network CPU Utilization



Driver domain CPU utilization is up to 40% more!

Network Results Summary

Setup	Throughput	CPU Utilization
Bare-metal	line rate	up to 30% more
Direct access (msg size < 1024)	up to 45% less	N/A
Direct access (msg size >= 1024)	mostly the same	up to 40%-60% more
Driver domain (msg size < 512)	up to 15% less	N/A
Driver domain (msg size >= 512)	mostly the same	up to 40% more

IOMMU is expensive (although not prohibitive)...

What can we do about it?

Pre-allocating the IO Address Space

- Map the entire guest address space in the IOMMU address space such that the guest pseudo-physical address that maps a given machine frame is equal to the DMA address that maps that machine frame.
- Start-up cost but minimal runtime overhead.
- Isolates the system from the guest.
- But provides no protection inside the guest (guest is oblivious to the IOMMU).
- Precludes (or requires hypervisor involvement for) page flipping, ballooning and anything else that modifies guest P->M translations.
- Size of IO address space may be limited - theoretical 4GB limit on Calgary.

Allocate in Advance; Free When Done

- Don’t use the streaming DMA API (map / unmap).
- Use the persistent allocation DMA API (allocate / free).
- Goes against standard Linux driver practice.
- . . . DMA-API is really designed for platforms with limited number of DMA mappings.
- Alternative is to cache map / unmap calls in the DMA-API itself and save the hypervisor crossing - definitely beneficial for hypervisor scenario but not sure about baremetal.
- Another alternative is to allocate and free in large batches, rather than on a per-buffer basis - add dma_map_multi and dma_unmap_multi and teach drivers and subsystems to batch their DMA mappings.

Other Optimizations

- Deferred TCE cache flush.
- Xen multicalls.
- Never free! The mapping may exist until it gets reused, but if the driver is well-behaved and the mapping does not map anyone else's page. . . who cares?
- Grant table integration: when using PV drivers map and unmap intelligently from the grant table ops rather than from the DMA API. Only applicable for driver domains, not for direct hardware access domains.

Conclusions and Future Work

- IOMMUs are useful and necessary
- . . . but they have non-negligible costs at the moment - up to 60% more CPU utilization
- . . . which we know how to fix!
- What about Intel VT-d and AMD IOMMU?
- Once we get rid of the software inefficiencies, how do we build better IOMMUs?

Availability

- Your favorite kernel.org mirror.
- `http://xenbits.xensource.com/ext/xen-iommu.hg`
- `http://xenbits.xensource.com/ext/linux-iommu.hg`
- *Utilizing IOMMUs for Virtualization in Linux and Xen*, M. Ben-Yehuda, J. Mason, O. Krieger, J. Xenidis, L. Van Doorn, A. Mallick, J. Nakajima, and E. Wahlig, OLS ’06.
- *The Price of Safety: Evaluating IOMMU Performance*, M. Ben-Yehuda, J. Xenidis, M. Ostrowski, K. Rister, A. Bruemmer, L. Van Doorn, to appear at OLS ’07.

Thank You for Listening!

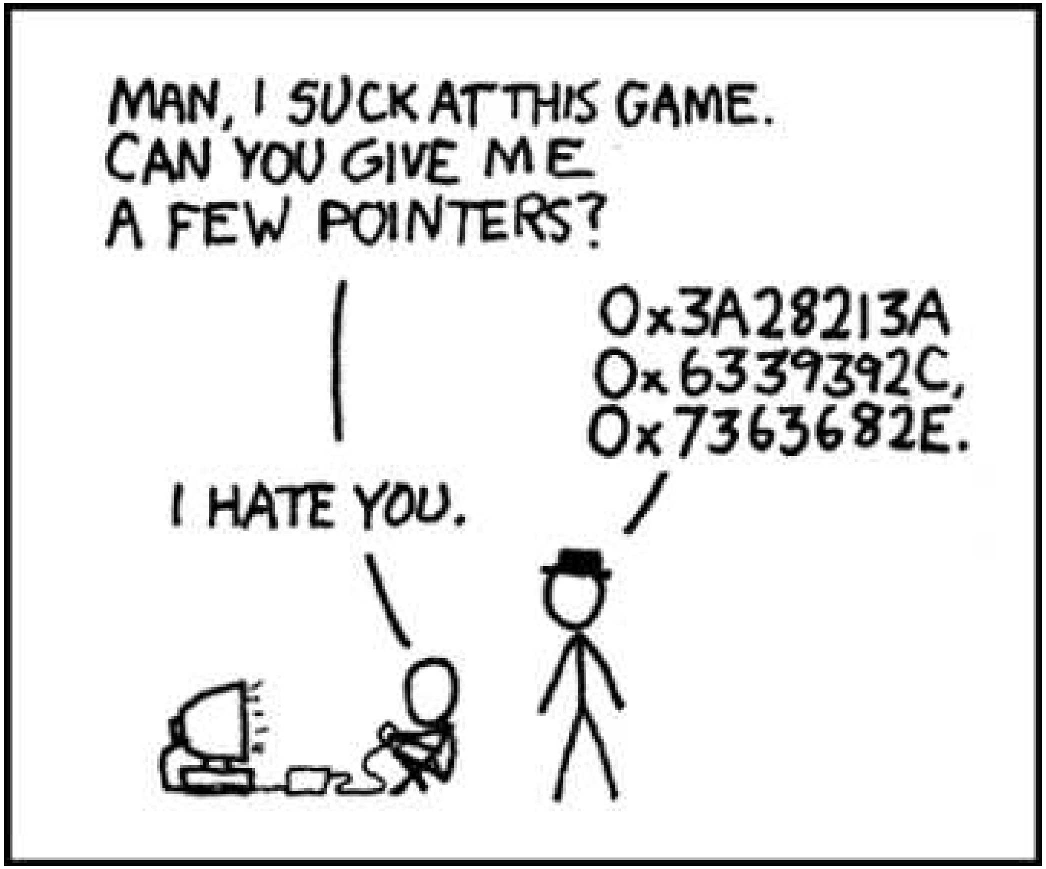


Figure 2: http://xkcd.com/c138.html