ECE 571 – Advanced Microprocessor-Based Design Lecture 23

Vince Weaver

http://web.eece.maine.edu/ vweaver

vincent.weaver@maine.edu

26 November 2019

Announcements

- Project Status – remember to include related work
- Related: never pay for an academic paper
  - No one involved with the paper ever sees that money
  - Also the UMaine library pays a lot of money to subscribe to the papers, so access them through the library website
- Homeworks – will be graded

Virtualization

- Running multiple copies of operating system on one machine
- Often designed to be transparent – operating systems do not realize they are not running on bare metal

Virtualization – Why do it?

- Server consolidation – take many lightly loaded servers, combine on one machine (reduce maintenance costs)
- Security/Sandboxing – hackers can hack an OS but not whole machine
- Reliability – one OS image goes down, doesn't take rest with it
- Virtual memory images can be easily copied and brought up on various systems (hypervisor hides hardware differences)

Virtualization – Downsides

Virtualization Types

- Slower/overhead
- Security – if someone breaks out and into the VM can compromise them all
- Security – timing attacks
- Reliability – one machine dying can take out many OS images

Different levels of abstraction.

- Simulation – perfectly emulate hardware/CPU – slow

- Full-virtualization – fully simulate hardware, OS generally does not realize is being virtualized
- Paravirtualization – full hardware not simulated, virtual I/O interfaces provided
  guest oses have to be aware of this (but less hardware emulation slowdown)
- Containers – operating system userspace separation (processes see independent OS setup, filesystem, etc) but still talking to one OS image via syscalls

Terms

- Guest
- Host
- VM (virtual machine)
- Hypervisor

Are you running on real hardware?

- VM (some power machines, ps3, never run on raw hardware)
- Nested VM
- SMM mode (system maintenance mode)

Simulation

- Simulation

Full Virtualization

- Virtualize the CPU, some sort of simulation of hardware
- Trap on access to hardware and simulate (with Qemu or similar)
- KVM
- VMware

Popek and Goldberg virtualization requirements

Formal requirements for virtualizable third generation architectures, Communications of the ACM, 1974.

- equivalence (fidelity): a program running under a VM should behave identical to running on bare metal monitor (VMM) should
- resource control (safety): the VM must control all resources

- efficiency (performance): most instructions must execute without intervention

Hardware Virtualization Extensions (CPU)

- IBM System/370 in 1972
- x86 chips by default were not, leak too much info.

Intel VT-x and AMD-V

- See *A Comparison of Software and Hardware Techniques for x86 Virtualization by Adams and Agesen, ASPLOS 2006.*
- VMware managed full virt on 32-bit x86 using dynamic binary instrumentation (to handle trapping privileged instructions) and segmentation (to handle memory)
- De-privledging: any attempt to read privileged info traps and can be intercepted
- Shadow structures: need copies of things that can't be

intercepted at CPU level, like page-tables. Need to trap on access to these. True vs hidden page faults.

- x86 issues (assume protected mode)
  - visible privileged state (see privileged mode when read CS register; CPL (privilege level) lower 2 bits)
  - Lack of traps when privileged instructions run at user-level.
  - popf (pop flags) changes both ALU and system flags (IF, enable interrupts). When run non-privileged ignores this, doesn't trap.
- x86-64 mostly removed segmentation (At least at first)

so old ways wouldn't work

- Intel VT-x and AMD-V
  - Adds virtual machine code block
  - Intel: extended page tables (nested page tables)
  - VMCS shadowing: allow nested VMs

Second Level Address Translation (SLAT)

- HW alternate to SW managed shadow page tables
- Virtual memory host serves as Phys memory of guest, so need to translate pages twice on every page fault, etc

Other things

- KSM – kernel same-page mapping, de-duplicate (consolidate) identical pages in system to save RAM
- Balloon driver, to balance RAM across VMs only as needed
- GPU virtualization
- IOMMU
- Interrupt virtualization

KVM

- Requires CPU with hardware virtualization extensions
- Kernel acts as hypervisor
- `/dev/kvm` interface
  - Set up VM and add memory, provide firmware
  - Set up I/O traps and handlers
  - Map video display
- Hardware and I/O emulation often handled by Qemu

Paravirtualization

- Hypervisor creates a special API that the guest OS uses (operating system must be modified)
- Can be faster (talk directly to hypervisor, no need to emulate hardware)
- Xen – uses stripped down Linux as hypervisor?
- Need specially compiled kernel that knows about hypervisor interfaces

Containers

- ;Login article
- Look like you have own copy of OS, but just walled off more thoroughly than normal Unix process. More lightweight than VM

Traditional HPC

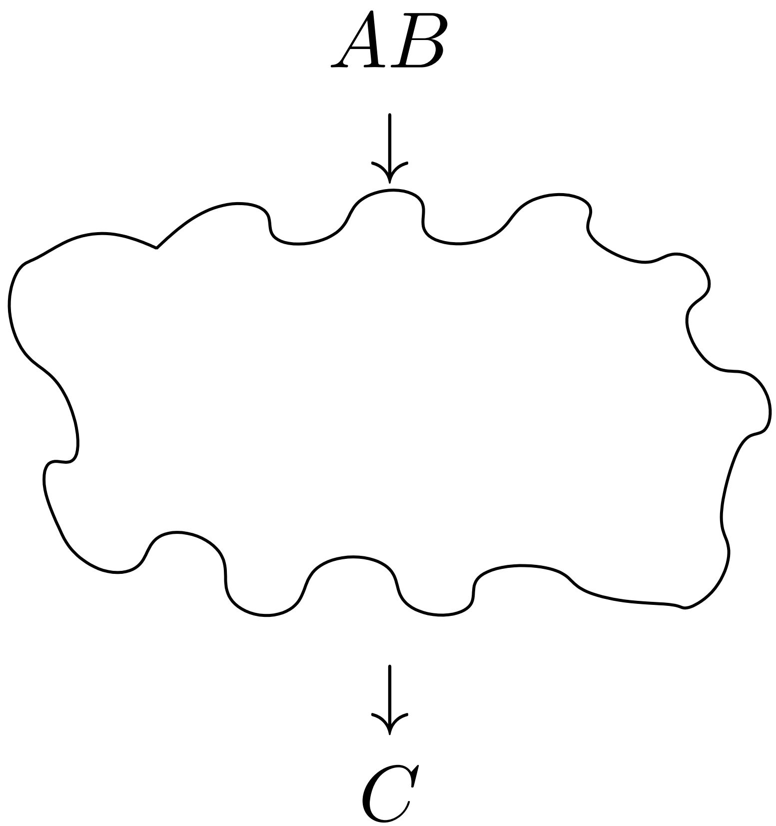
$$AB$$

$$\downarrow$$

$$\downarrow$$

$$C$$

Cloud-based HPC



Cloud Tradeoffs

Pros

- No AC bill
- No electricity bill
- No need to spend $$$ on infrastructure

Cons

- Unexpected outages
- Data held hostage
- Infrastructure not designed for HPC

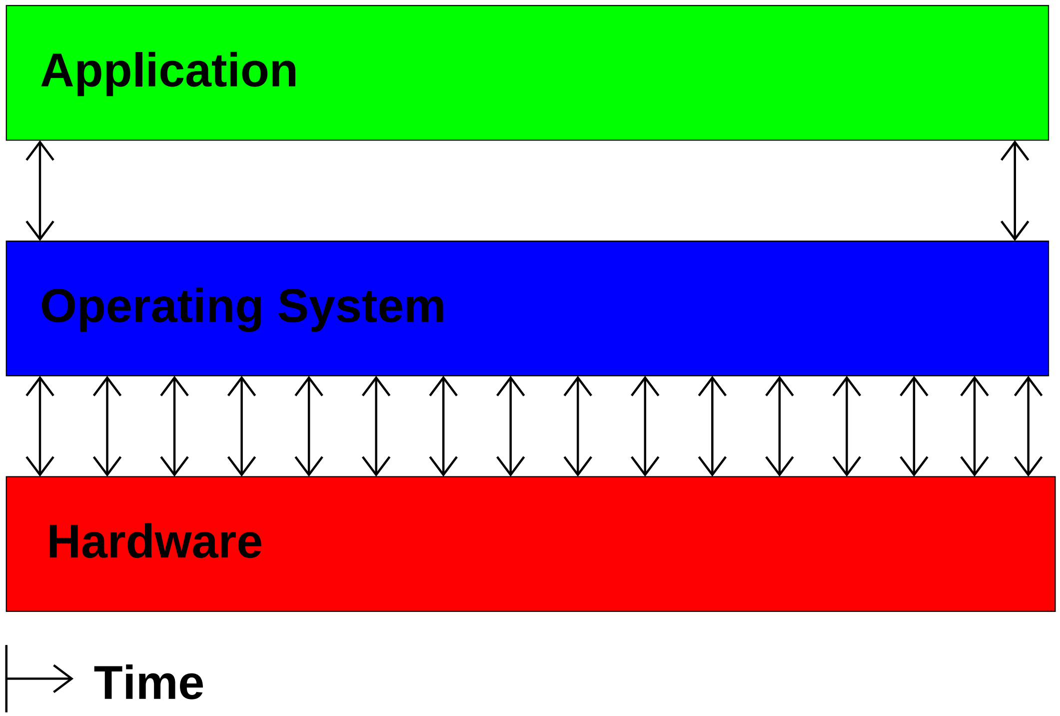
Measuring Performance in the Cloud

First let's just measure runtime

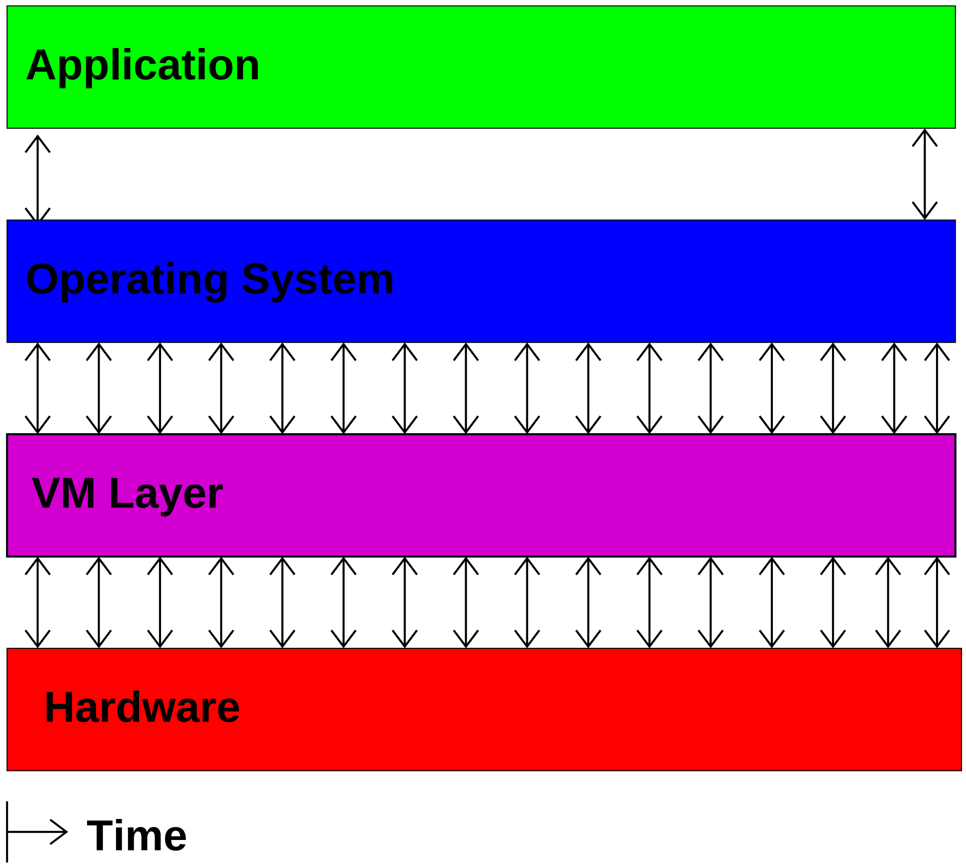
This is difficult because in virtualized environments

Time Loses All Meaning

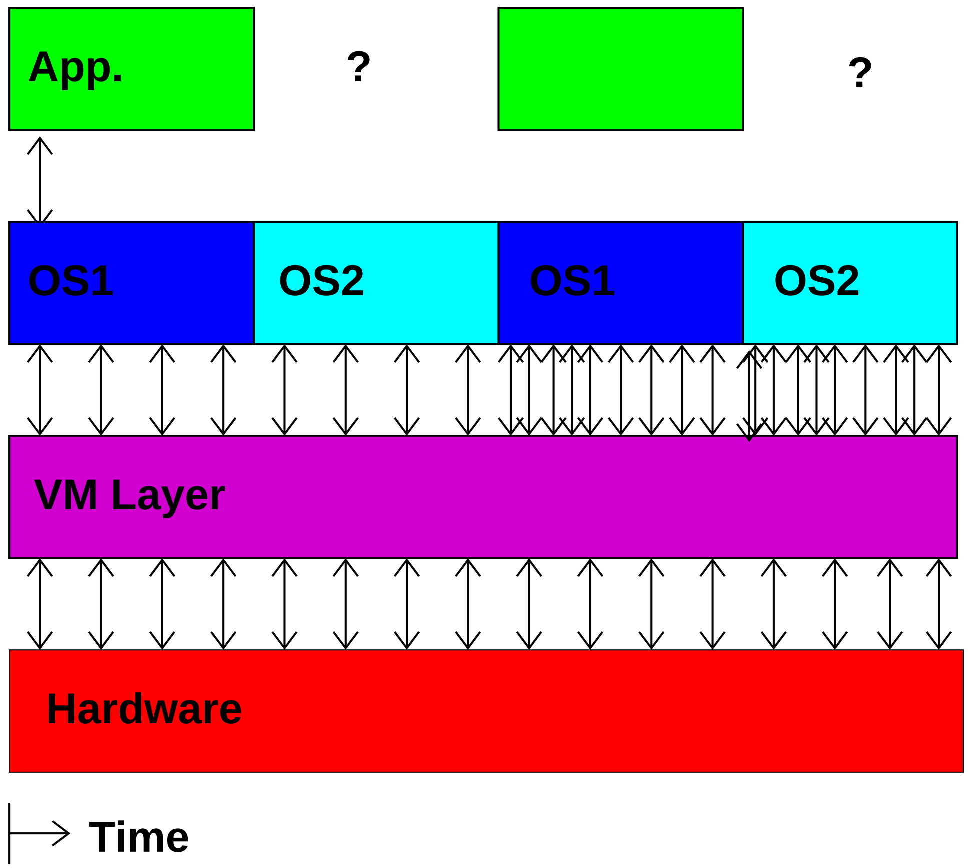
Simplified Model of Time Measurement



Then the VM gets involved



Then you have multiple VMs



So What Can We Do?

Hope we have exclusive access and measure wall-clock time.

Measuring Time Externally

- Ideally have local hardware access, root, and hooks into the VM system
- Otherwise, you can sit there with a watch
- Danciu et al. send UDP packet to remote server
- Most of these are not possible in a true “cloud” setup

Measuring Time From Within Guest

- Use `gettimeofday()` or `clock_gettime()`
- This might be the only interface we have
- How bad can it be?

Cloud Performance Measurement

With High Performance Computing moving to the cloud, virtualization-aware performance measurement tools are a necessity.

Performance API (PAPI)

- Widely-used, Cross-platform, Open-Source Performance Measurement Library

  ⇒ Linux, AIX, FreeBSD, Solaris

  ⇒ x86, Power, ARM, MIPS

  ⇒ BlueGene P/Q, Cray

- Use directly or via high-level tools (TAU, Perfsuite, Vampir, Scalasca, HPCToolkit)

PAPI-V

Virtualization-aware PAPI, or "PAPI-V" extends PAPI to be useful in cloud environments.

- Report virtual system info
- Provide enhanced timing info
- Virtualization-related components
- Virtualized Counters

Virtual System Info

- Virtualization vendor obtained via CPUID, reported in `hw_info.virtual_vendor_string`
- Supported by KVM, Xen, VMware, etc.
- Info for user, helps with bug reports

The Timing Problem

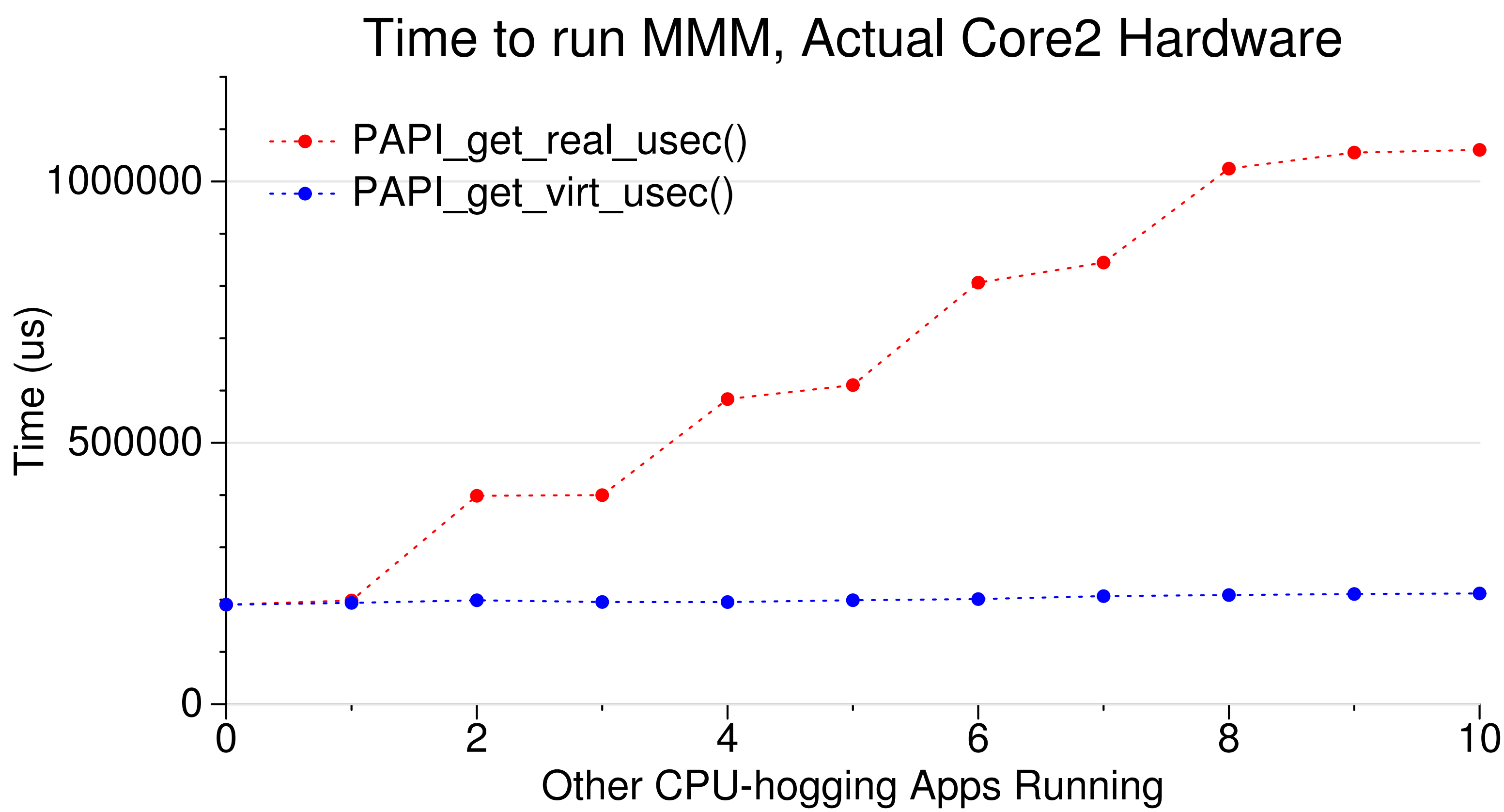
- Time is an important component of most performance measurements
- The concept of "time" gets fluid once virtualization is involved
- Ideally you want wallclock time; this is hard to get within a VM guest

PAPI Timing Interface

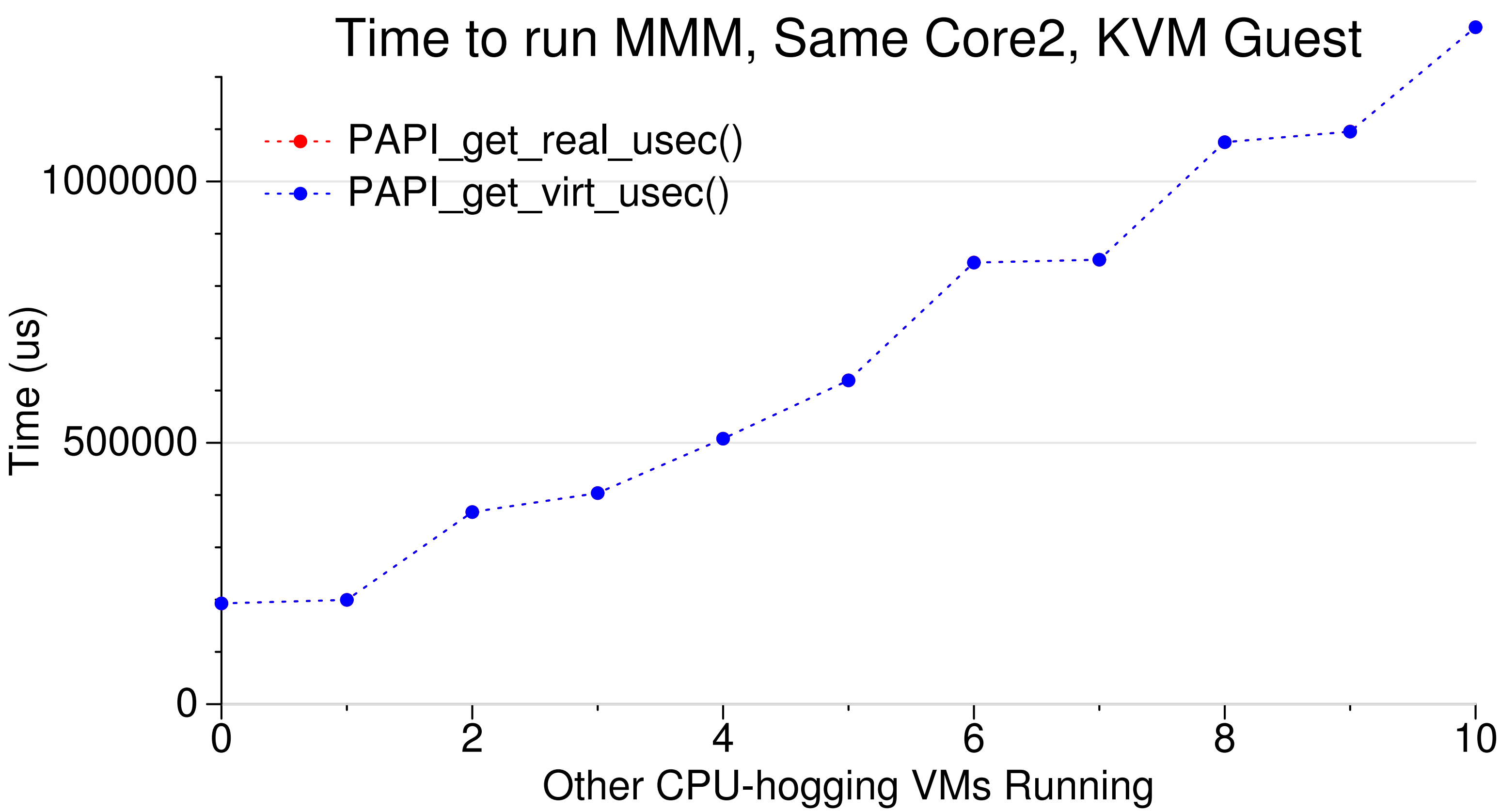
On Linux the timing functions use the POSIX timer interface

- `PAPI_get_real_usec();`
  ⇒`clock_gettime(CLOCK_REALTIME);`
- `PAPI_get_virtual_usec();`
  ⇒`clock_gettime(CLOCK_THREAD_CPUTIME_ID);`

Timing Behavior on Bare Metal



Timing Behavior on Virtualized System

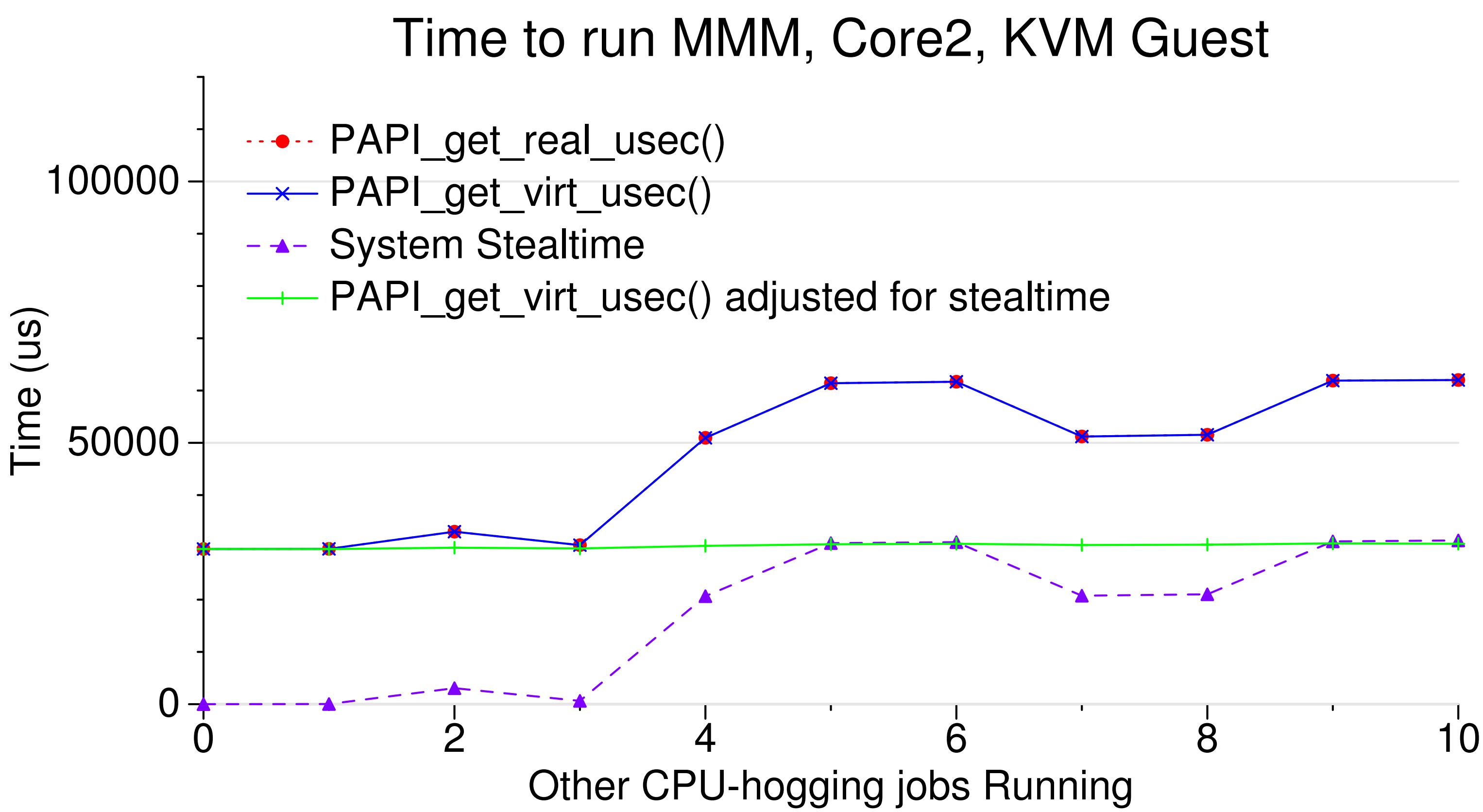


Stealtime

What is needed is a way for accounting for time the VM is scheduled out.

- Since 2.6.11 Linux can provide this *stealtime* information
- It is system wide, not per-process, which makes auto-adjusting PAPI timing measurements problematic
- PAPI 5.0 provides a stealtime component

Timing Adjusted with Stealtime

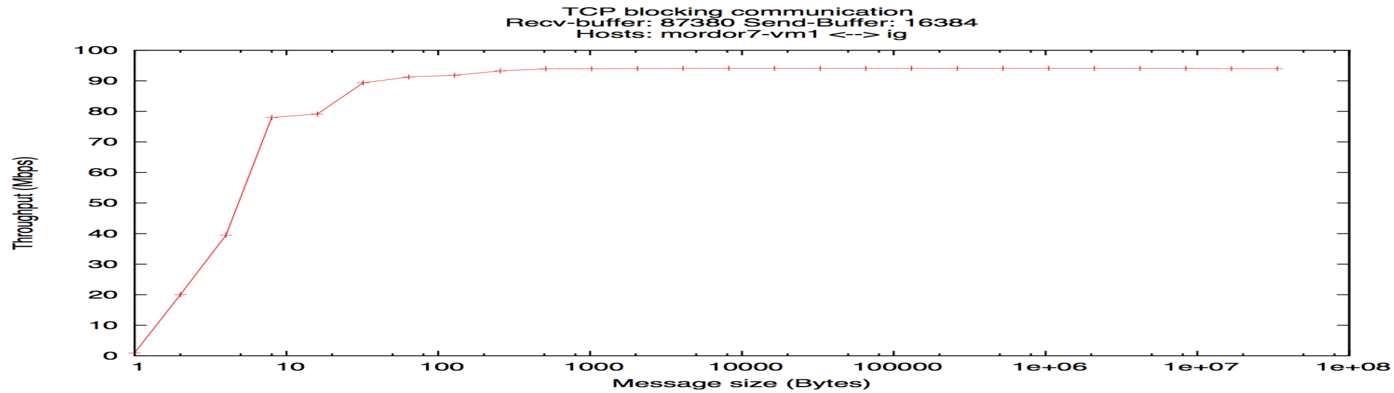


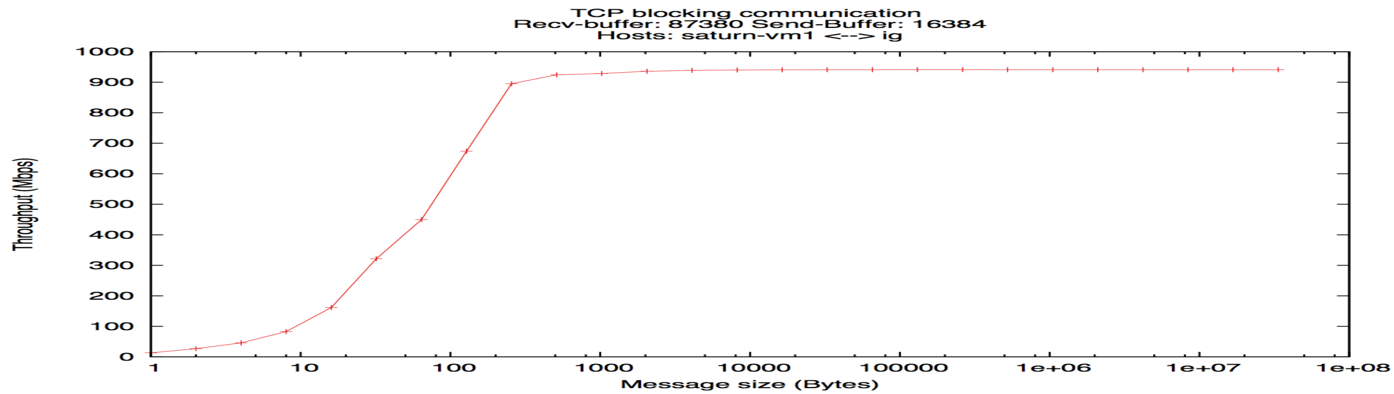
Network Components

PAPI also has components for measuring Network I/O.

- Generic network component
- Infiniband component
- Myrinet component

Infiniband DirectPath Comparison





VMware Component

PAPI supports a component that provides access to VMware-specific interfaces

- pseudo-performance counters – extra timing info via `rdpmc`
- VMware guest SDK (ESX only) – provides various other performance related measurements, including stealtime

Virtualized Performance Counters

The VM host can virtualize performance counter access by trapping access to the MSRs, and saving/restoring values when suspending/resuming VMs.

- KVM supports this as of Linux 3.2 with a sufficiently recent version of the QEMU/KVM tool (with some limitations)
- Xen supports this as of Linux 3.5
- VMware support is underway