ECE 571 – Advanced Microprocessor-Based Design Lecture 21

Vince Weaver

http://www.eece.maine.edu/∼vweaver

vincent.weaver@maine.edu

25 November 2014

Project Reminder

- Remember to work on your related work.

Virtualization

Different levels of abstraction.

- Simulation
- Full-virtualization
- Paravirtualization
- Containers

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 maintanenence 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

KVM

- Requires CPU with hardware virtualization extensions
- Kernel acts as hypervisor

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 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 and segmentation.
  - De-privledging: any attempt to read privledged 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 privledged state (see privlede mode when read CS register; CPL (privlede level) lower 2 bits) Lack of traps when privledged instructions run at user-level. popf (pop flags) changes both ALU and system flags (IF, enable interrupts). When run non-privledged ignores this, doesn't trap.

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

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

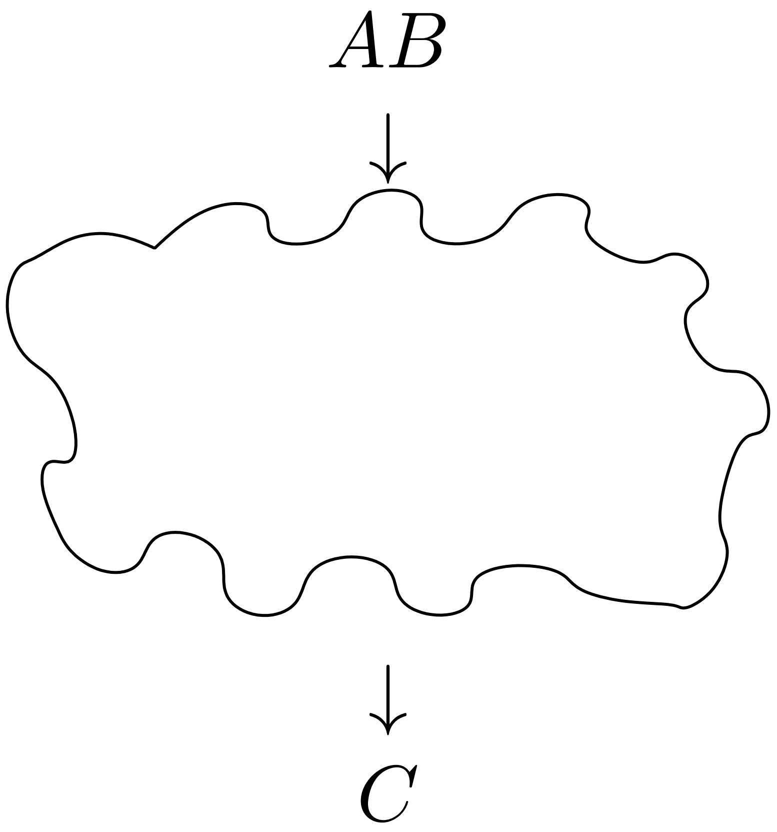
$$AB$$

$$\downarrow$$

$$\downarrow$$

$$C$$

Cloud-based HPC



$$\begin{tikzcd}AB \arrow[d] \\ \boxed{\quad} \arrow[d] \\ C\end{tikzcd}$$

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

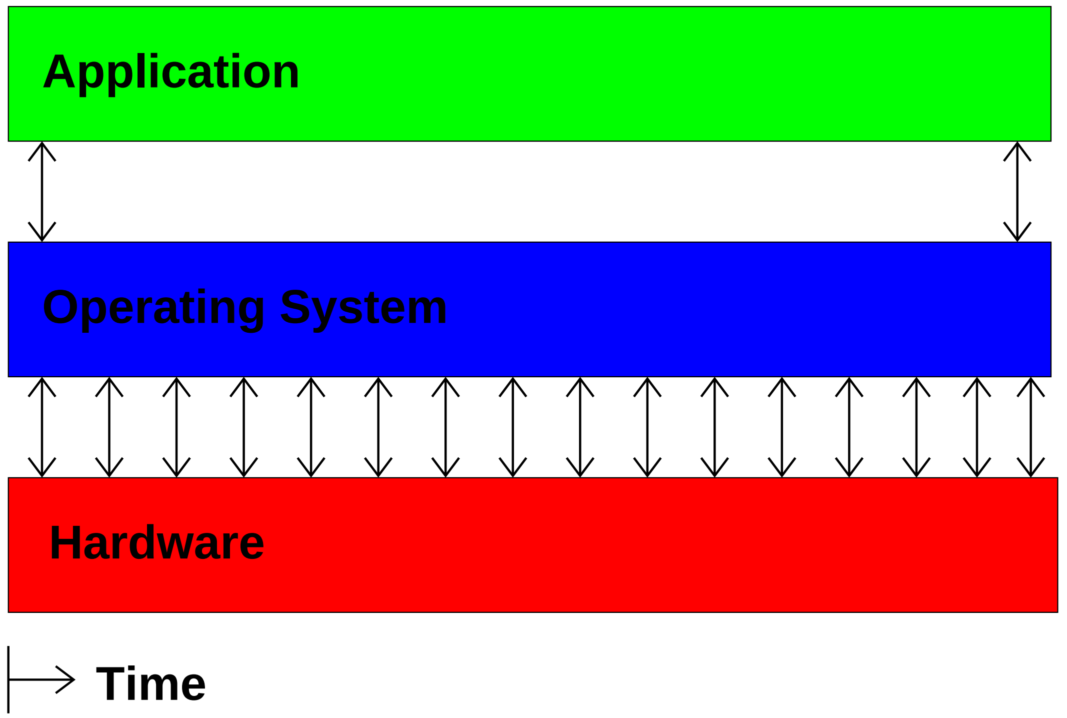
Measuring Performance in the Cloud

First let's just measure runtime

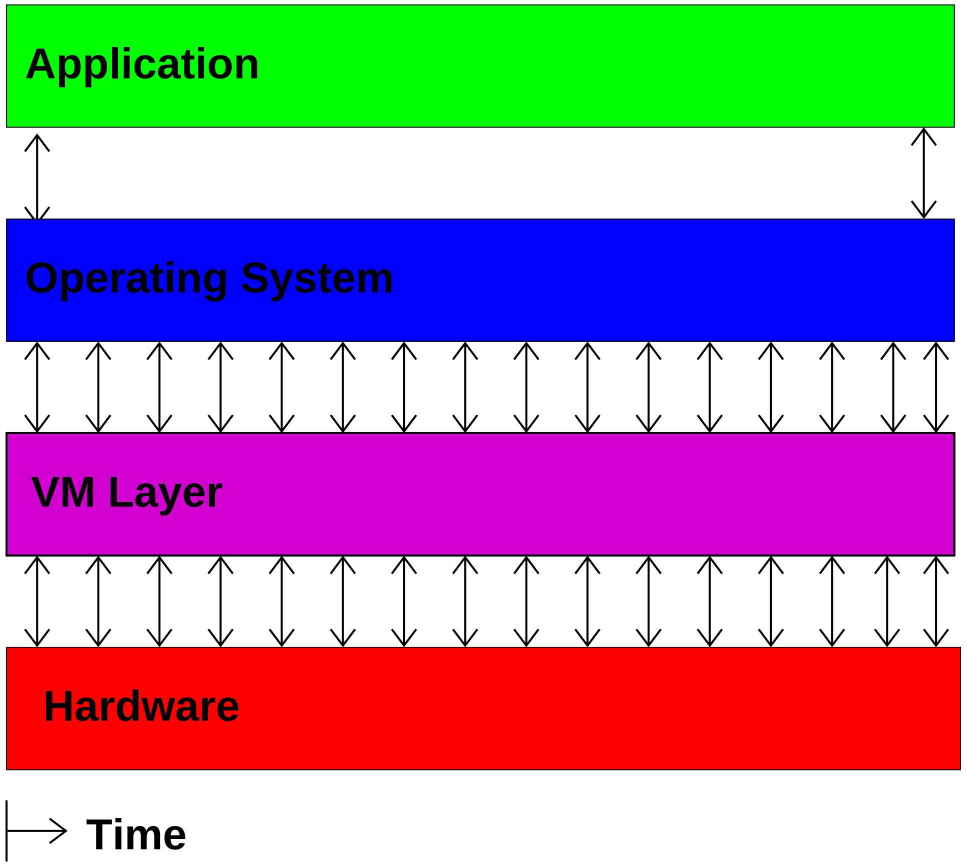
This is difficult because in virtualized environments

Time Loses All Meaning

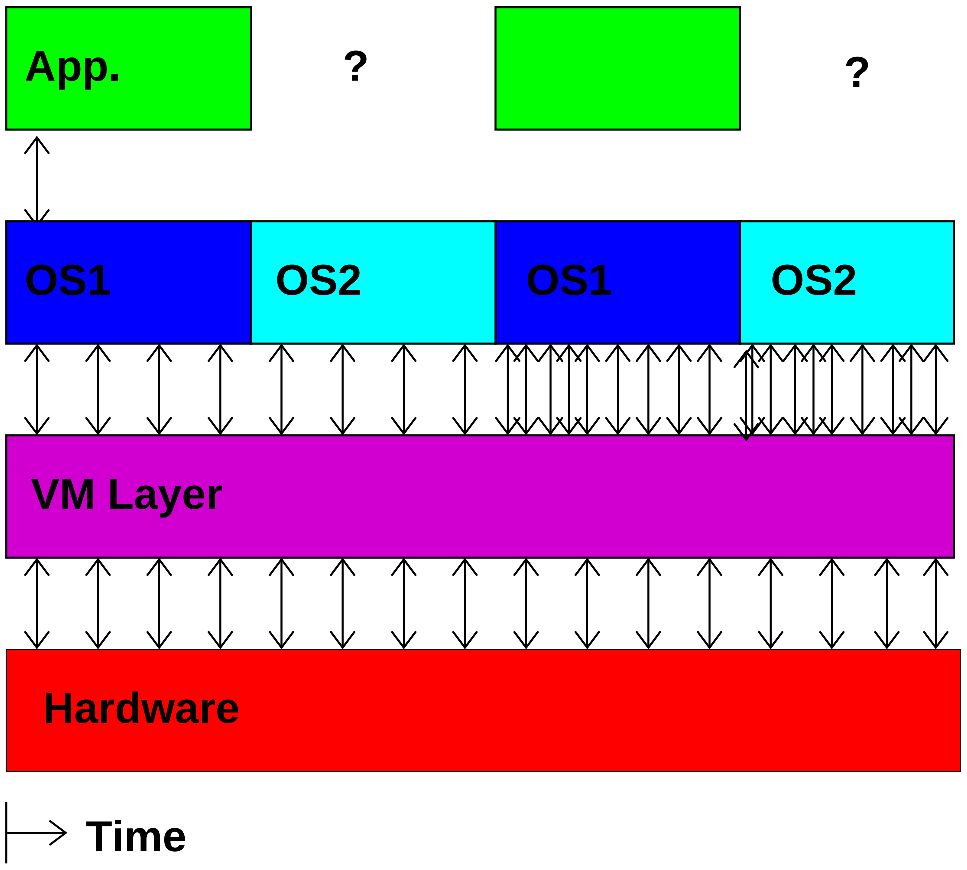
Simplified Model of Time Measurement



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

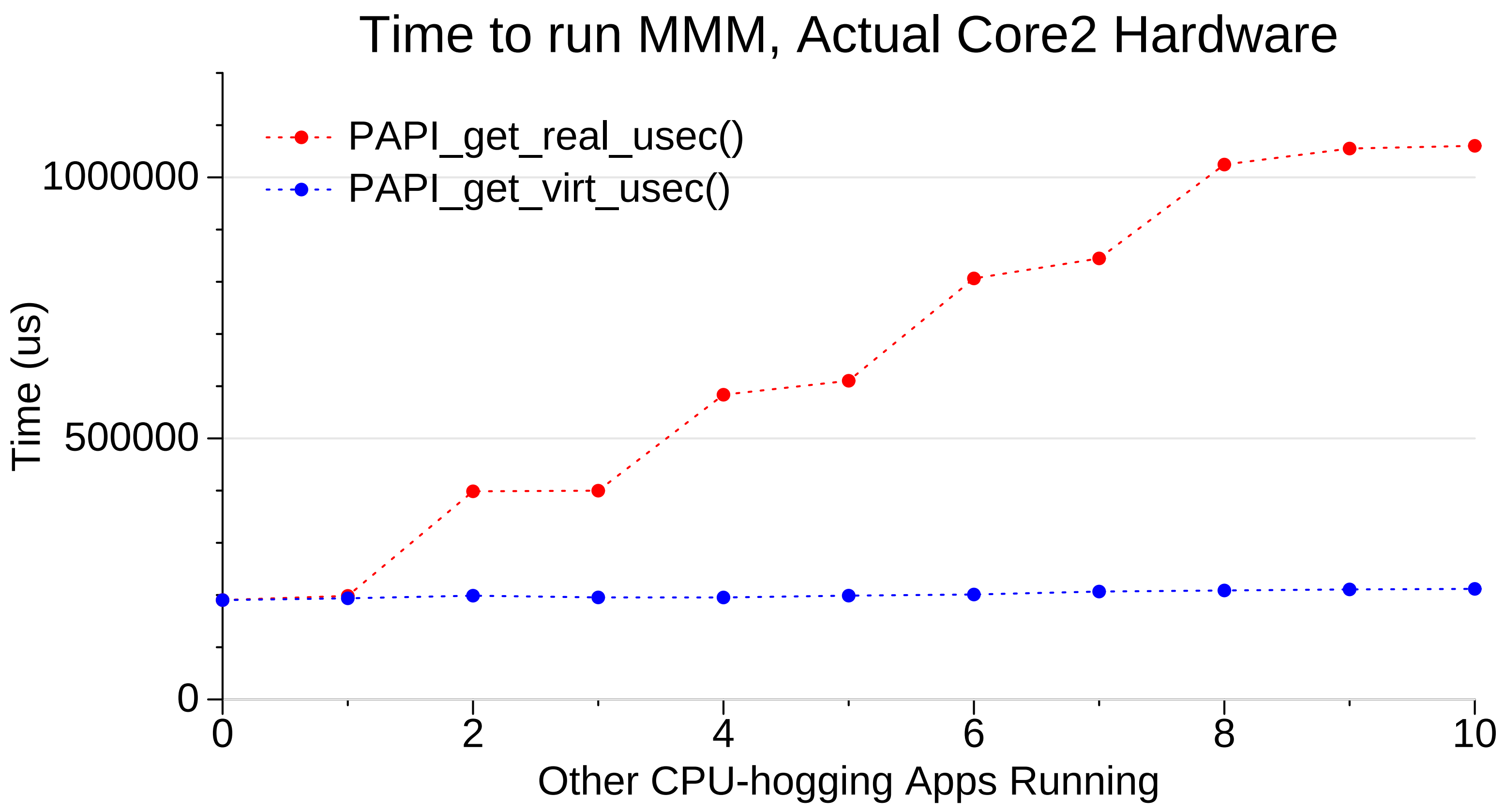
- Time is an important component of most performance measurements
- The concept of “time” gets fluid once virtualization is involved
- Ideally you want wallclock time; this is hard to get within a VM guest

PAPI Timing Interface

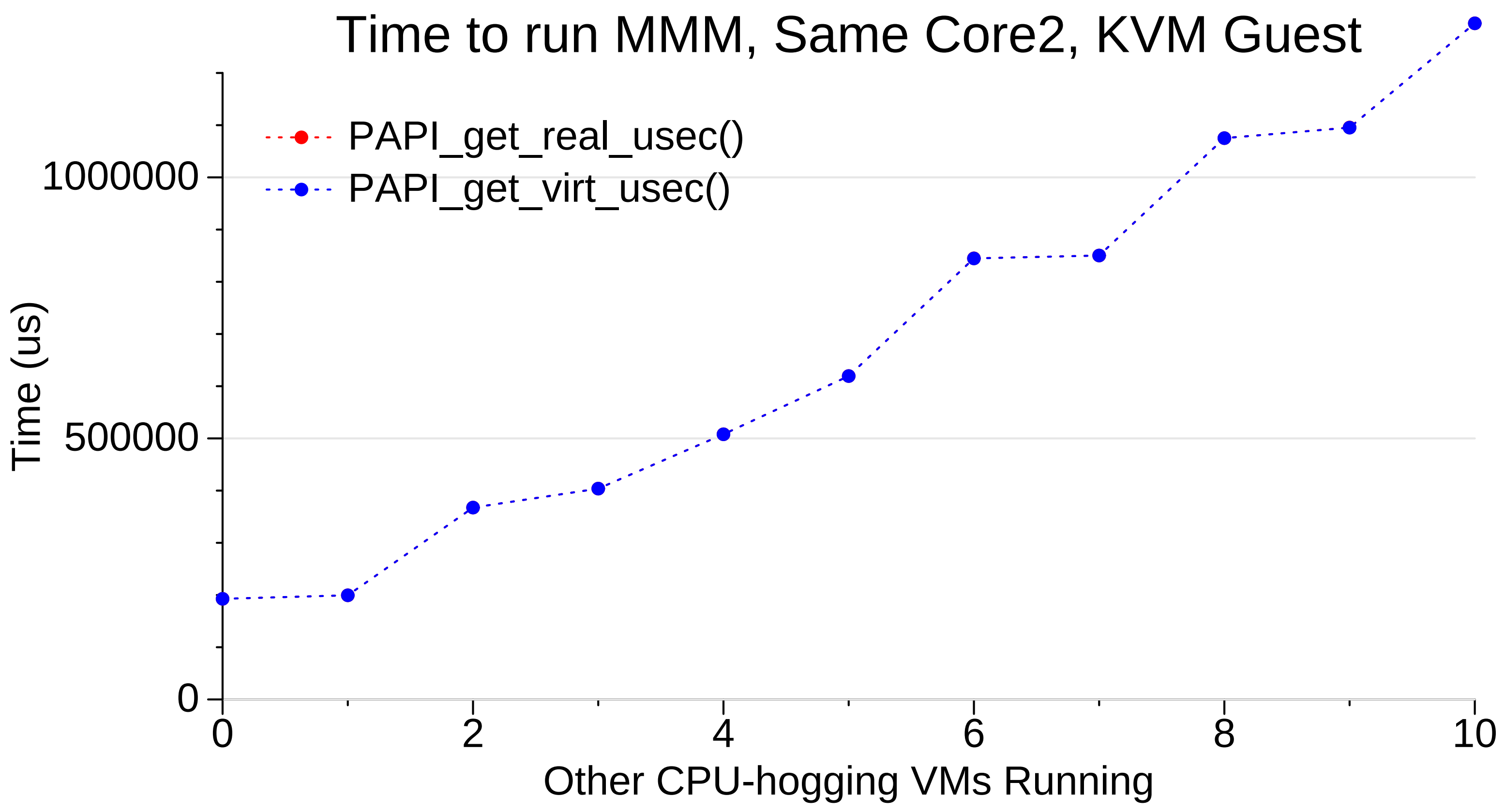
On Linux the timing functions use the POSIX timer interface

- `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

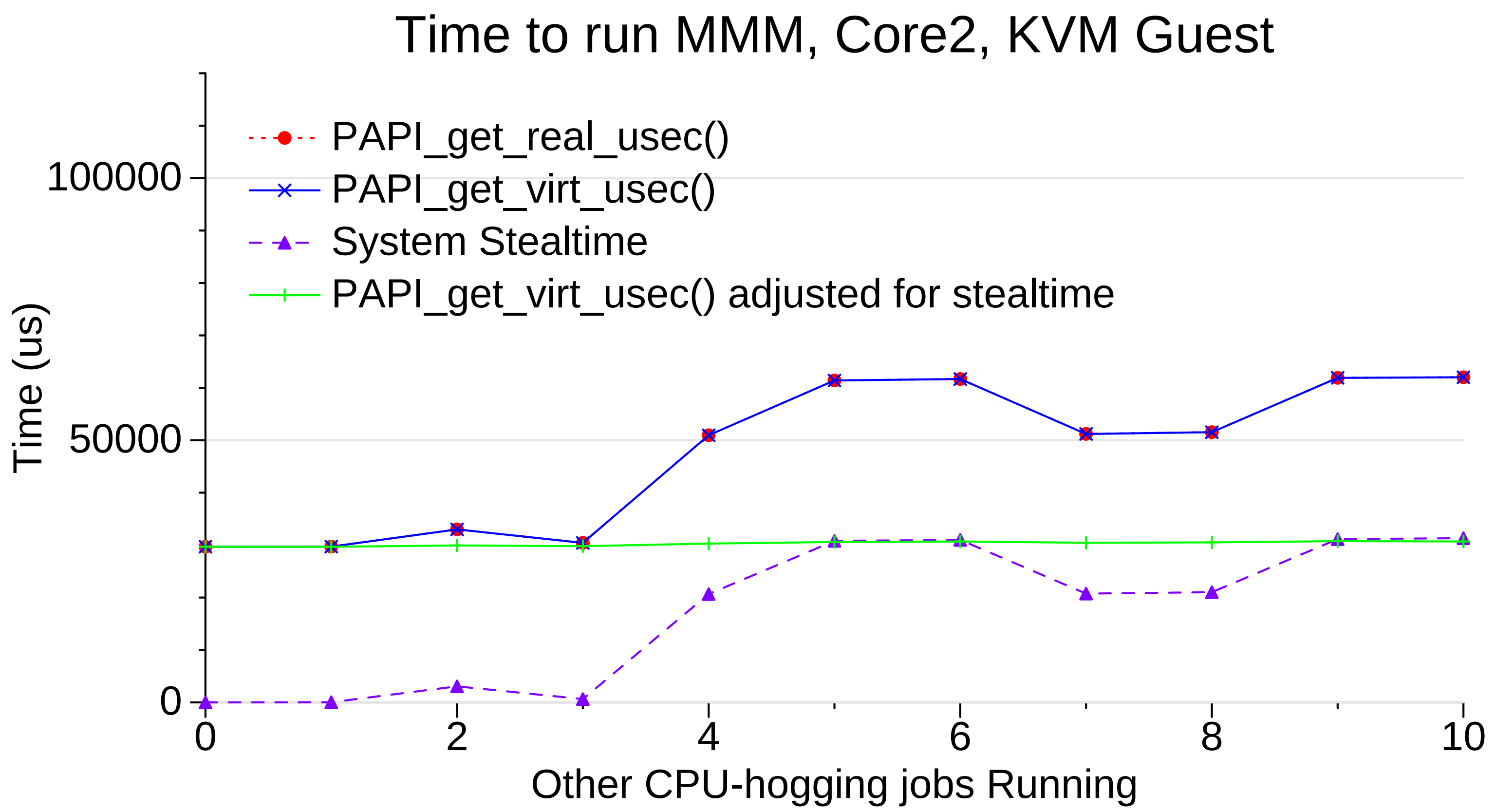


Stealtime

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

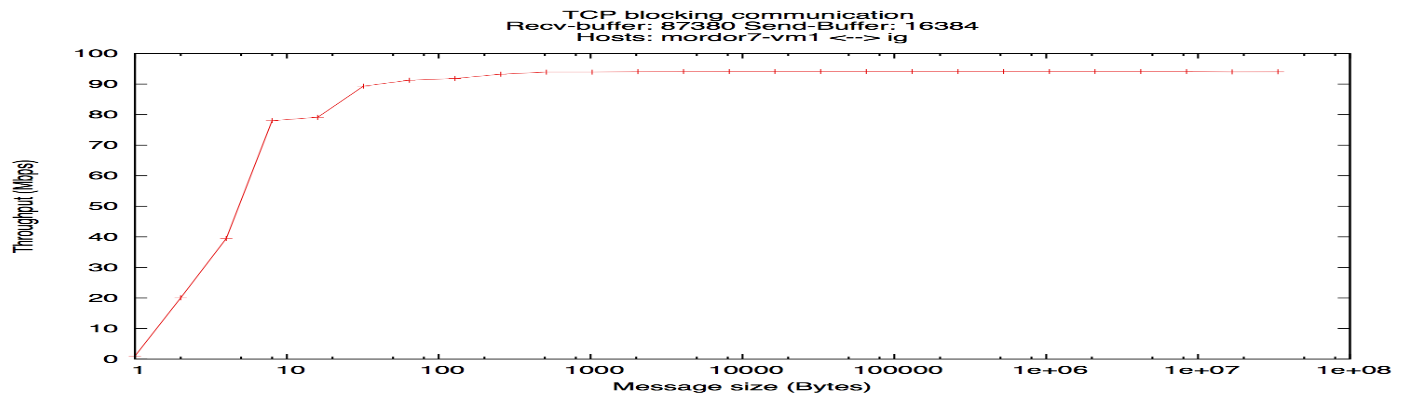


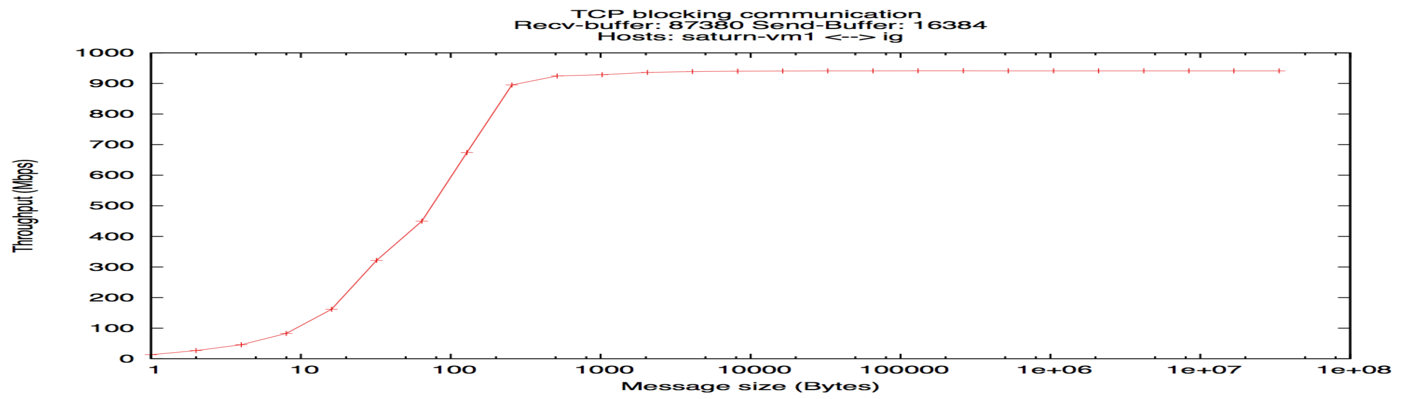
Network Components

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