ECE 574 – Cluster Computing
Lecture 23

Vince Weaver

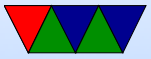
http://web.eece.maine.edu/~vweaver

vincent.weaver@maine.edu

16 April 2019

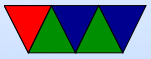
Announcements

- HW#10 was posted
- Don't forget project topics Thursday

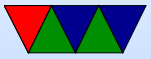


Virtualization

- Lets you run multiple operating system images, giving each the illusion that they are running on distinct hardware.
- The OSes are context-switched between, much as processes are context-switched under an OS
- When running inside a fully virtualized system, code should not be able to tell it's not running on bare metal
- The OSes are isolated and one crashing should not affect

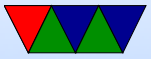


any of others.

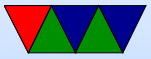


Why virtualize?

- Server consolidation – if you have multiple servers, each using 10% of CPU, can put them on one big server
- Security – can give each critical task its own full OS instance, so if something goes wrong it won't affect the others (this is harder to do with processes on an OS)
- Multiple OSes – can run multiple OSes (Windows, Linux, Etc) on same machine at same time
- Ease of deployment – can make OS snapshots/images

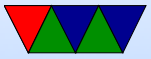


and can quickly bring up and down on other machine without having to install



Downsides of virtualization?

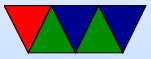
- Like any layer of abstraction: Overhead
- Slow, slow, slow



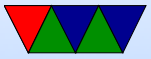
Terms

TODO: draw a diagram

- Guest – the operating system running inside a virtual system
- Host – the operating system running on bare metal (may be a hypervisor instead)
- VM (virtual machine) not virtual memory – the software/hardware that provides the virtual hardware interface

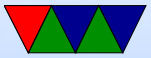


- Hypervisor – the software that controls bringing up and controlling the guest operating systems



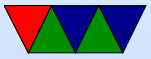
Are you ever running on real hardware?

- Some modern machines all you ever get to run on is the VM
- VM (some power machines, ps3, never run on raw hardware)
- Nested VM
- SMM mode (system maintenance mode)



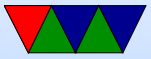
Full Simulation

- Emulate the entire CPU and all hardware in software
- Full system simulators, such as Qemu
- What's the downside of this? (slow, slow, slow)



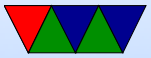
Full Virtualization

- "Virtualize the CPU"
  Run instructions as normal, but anything that gives away it is virtual must trap to the hypervisor.
- Trap on access to hardware and simulate (with Qemu or similar)
- KVM
- VMware



KVM

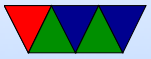
- Kernel-based virtual machine
- Hardware-assisted virtualization
- Requires CPU with hardware virtualization extensions
- Kernel acts as hypervisor
- Provides /dev/kvm
  - Sets up address space
  - Provide boot firmware
  - Display hardware



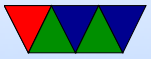
Popek and Goldberg virtualization requirements

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

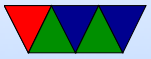


- efficiency (performance): most instructions must execute without intervention



Hardware Virtualization Extensions (CPU)

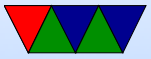
- IBM System/370 in 1972
- x86 chips by default were not, leak too much info.
- Intel VT-x and AMD-V



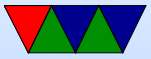
x86 virtualization

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.

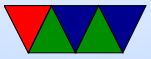
- De-privledging: any attempt to read privileged info traps and can be intercepted
- Shadow structures: need copies of things that can't be intercepted at CPU level, like page-tables. Need to trap on access to these. True vs hidden page faults.



- x86 issues (assume protected mode)
  - visible privileged state (see privilege 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.
- Intel VT-x and AMD-V
  - 2006
  - Adds virtual machine code block

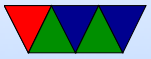


- Intel: extended page tables (nested page tables)
- VMCS shadowing: allow nested VMs



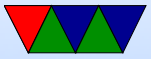
Paravirtualization

- 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

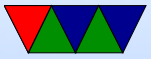


Containers

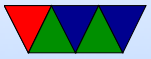
- ;Login article
- Look like you have own copy of OS, but just walled off more thoroughly than normal Unix process. More lightweight than VM



“The Cloud”

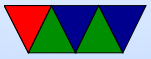


Containers



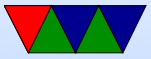
Docker

- Software can be installed on Linux to allow running containers
- Lightweight virtualization, runs on top of normal Linux but uses containers to isolate from other instances
- Uses cgroups, namespaces, union filesystems
- Unlike full virtualization, does not require another copy of the OS
- Also a way of packaging
- Docker swarm – clusters? more like an automatic failover



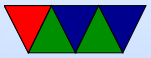
type situation?

- Written in go
- Difference from virtualization?
  - Doesn't need full disk image (large)
  - Doesn't need large reserved memory range
  - Diagram
    (Host-Hypervisor)-(GuestOS/Libs/App)
    (Host-Docker)-(Libs/apps)



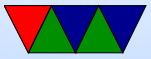
Kubernetes

- How to pronounce? Word is Greek for captain
- Originally from google? Lighter version of project borg?
- Pods full of containers that can communicate locally, to communicate remotely through an IP?
- Pods work together, use DNS and can share load
- Can run on top of Docker (but doesn't have to)
- Also written in go
- Master node, worker nodes



Kubernetes vs Docker

- https://containerjournal.com/2019/01/14/kuberne



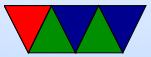
Traditional HPC

$$AB$$

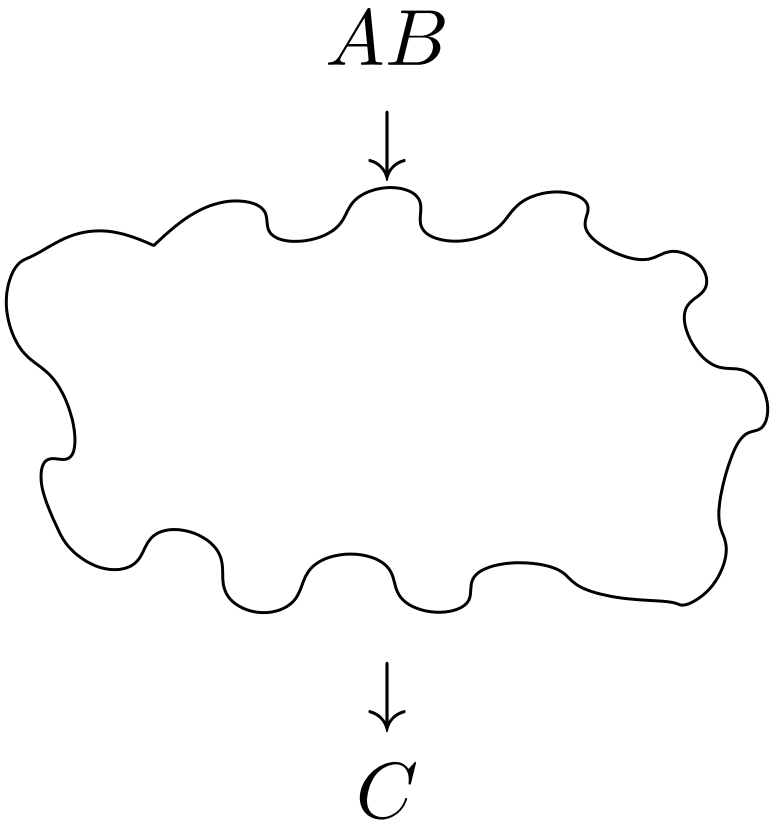
$$\downarrow$$

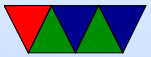
$$\downarrow$$

$$C$$



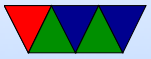
Cloud-based HPC





Cloud Tradeoffs

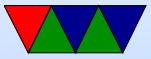
Pros	Cons
• No AC bill	• Unexpected outages
• No electricity bill	• Data held hostage
• No need to spend $$$ on infrastructure	• Infrastructure not designed for HPC



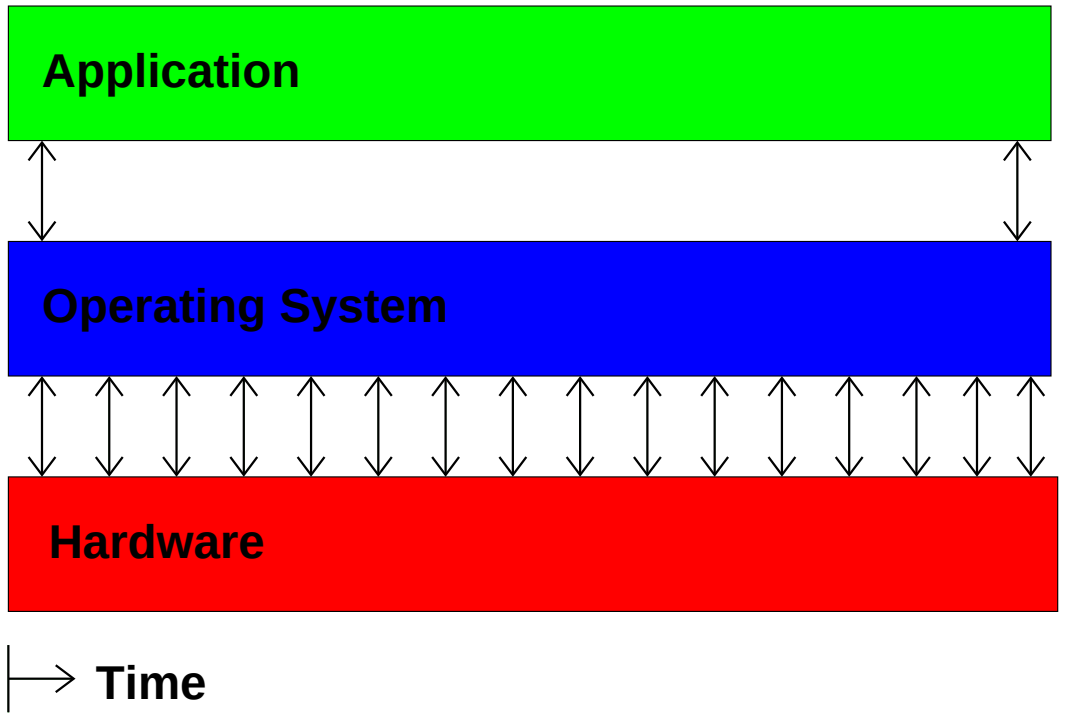
Measuring Performance in the Cloud

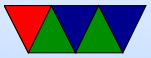
First let's just measure runtime

This is difficult because in virtualized environments

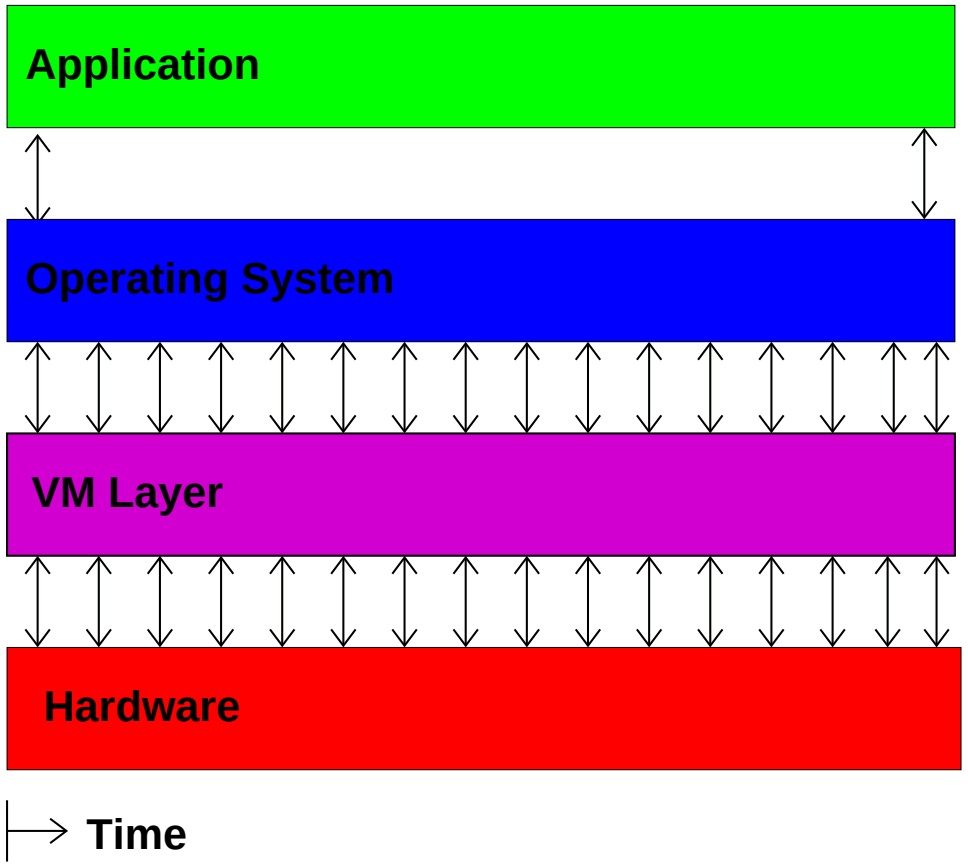


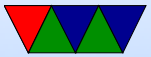
Simplified Model of Time Measurement



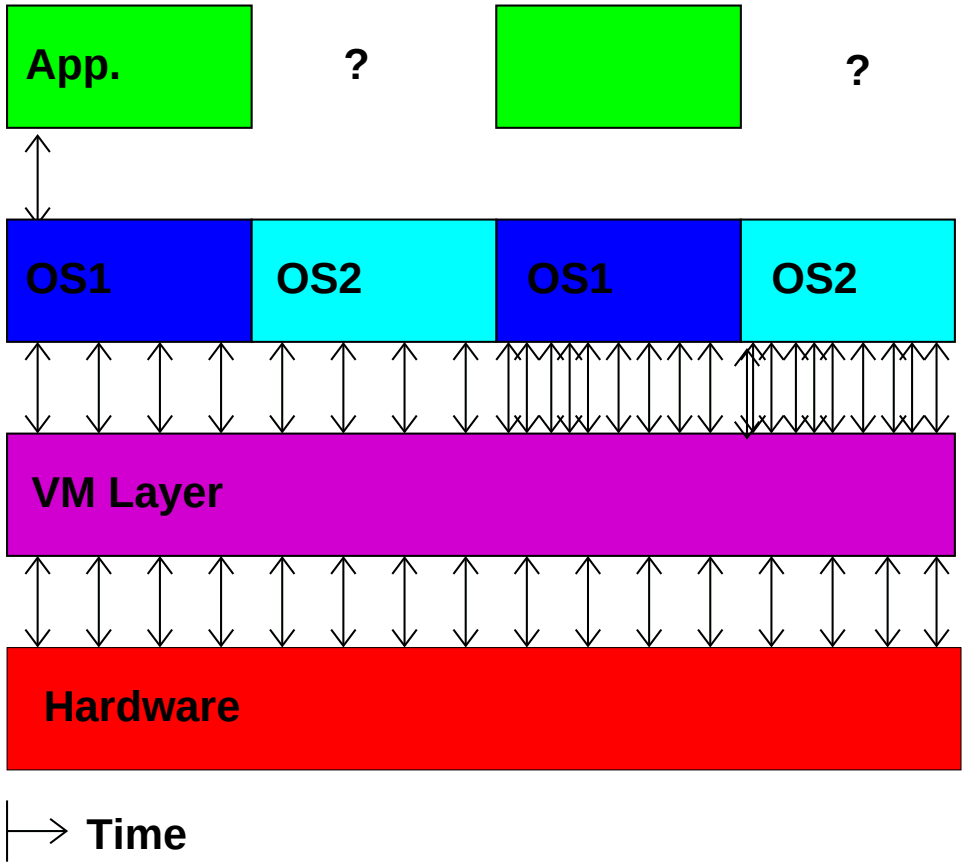


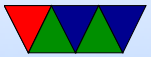
Then the VM gets involved





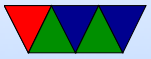
Then you have multiple VMs





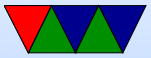
So What Can We Do?

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



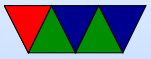
Measuring Time Externally

- 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



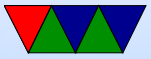
Measuring Time From Within Guest

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



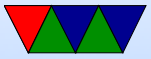
Cloud Performance Measurement

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



Performance API (PAPI)

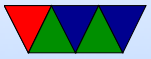
- Widely-used, Cross-platform, Open-Source Performance Measurement Library
  - $\Rightarrow$ Linux, AIX, FreeBSD, Solaris
  - $\Rightarrow$ x86, Power, ARM, MIPS
  - $\Rightarrow$ BlueGene P/Q, Cray
- Use directly or via high-level tools (TAU, Perfsuite, Vampir, Scalasca, HPCToolkit)



PAPI-V

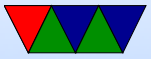
Virtualization-aware PAPI, or "PAPI-V" extends PAPI to be useful in cloud environments.

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



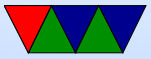
Virtual System Info

- 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

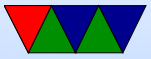
- Time is an important component of most performance measurements
- The concept of “time” gets fluid once virtualization is involved
- Ideally you want wallclock time; this is hard to get within a VM guest



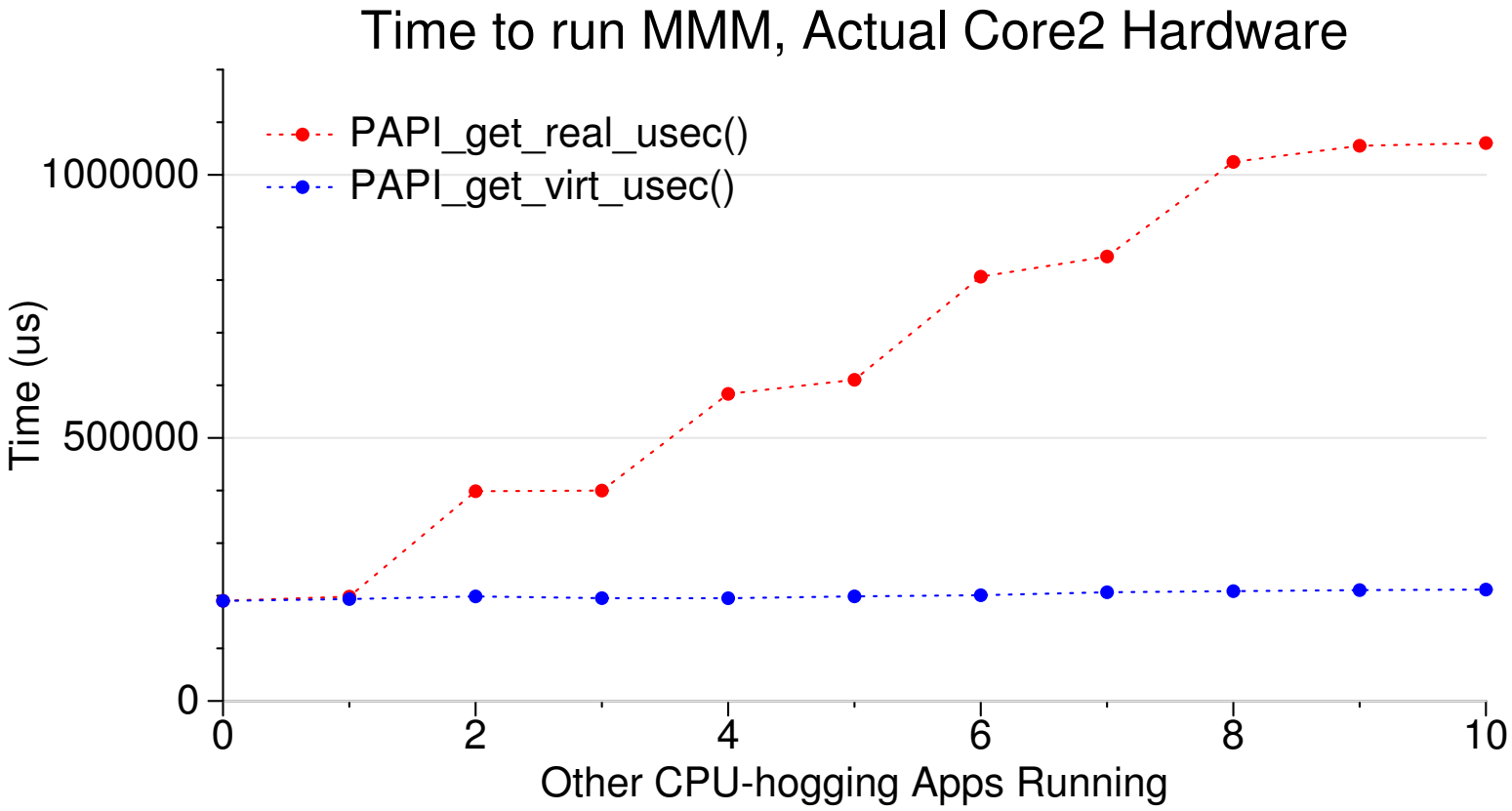
PAPI Timing Interface

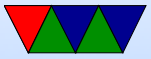
On Linux the timing functions use the POSIX timer interface

- `PAPI_get_real_usec();`
  $\Rightarrow$`clock_gettime(CLOCK_REALTIME);`
- `PAPI_get_virtual_usec();`
  $\Rightarrow$`clock_gettime(CLOCK_THREAD_CPUTIME_ID);`

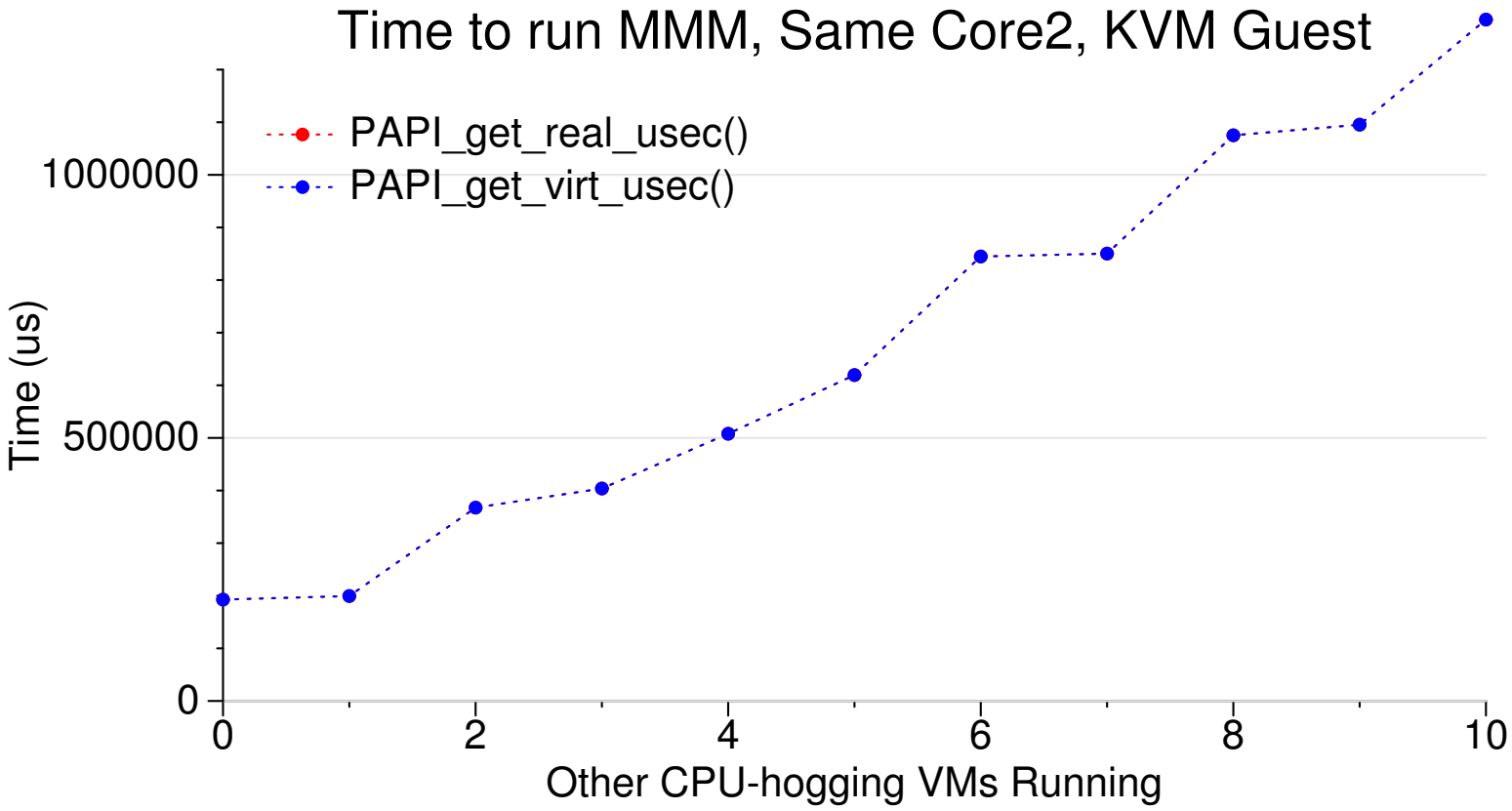


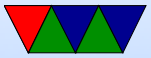
Timing Behavior on Bare Metal





Timing Behavior on Virtualized System

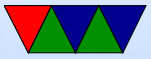




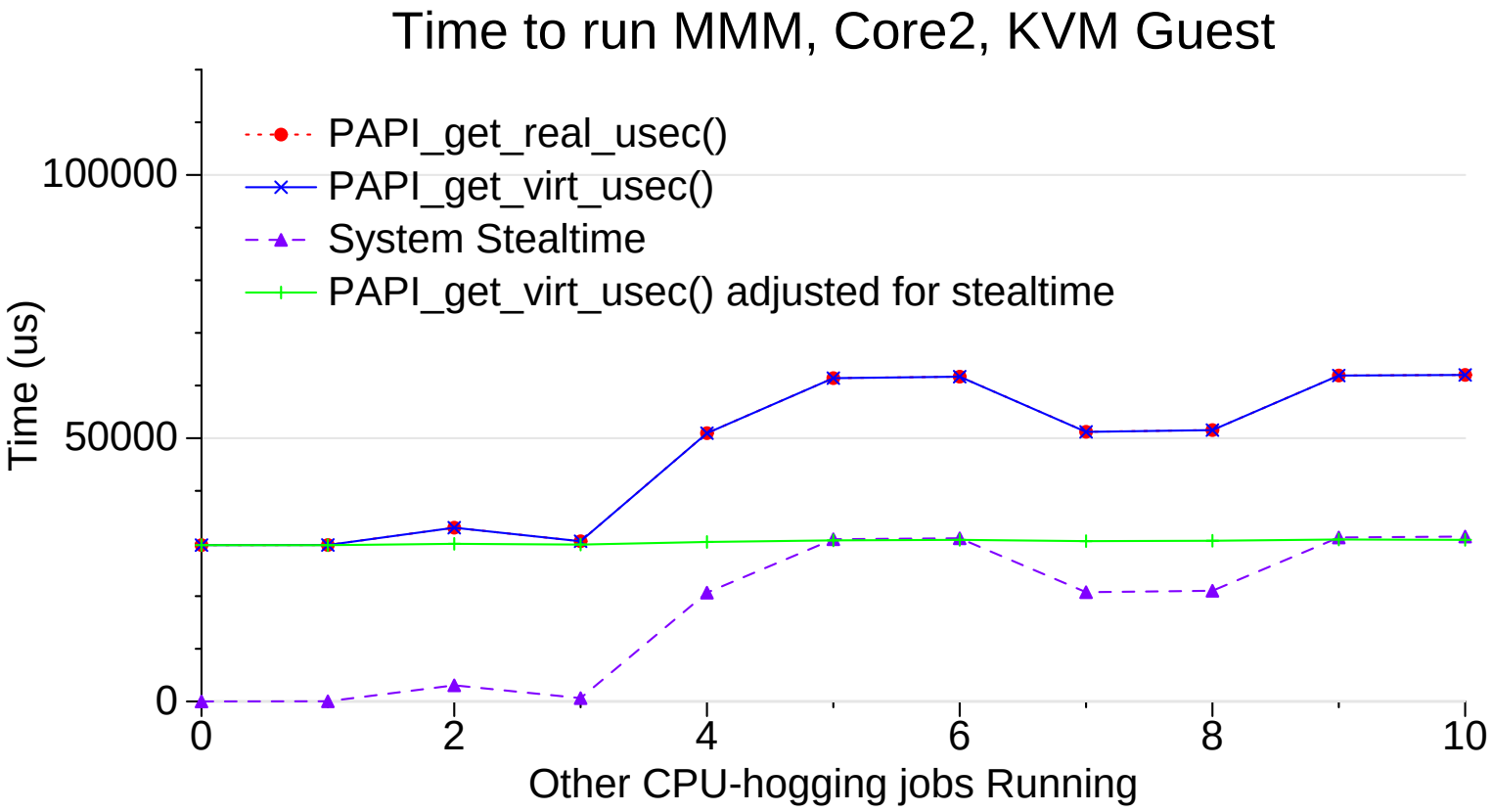
Stealtime

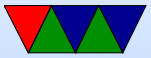
What is needed is a way for accounting for time the VM is scheduled out.

- 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

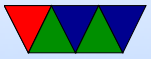




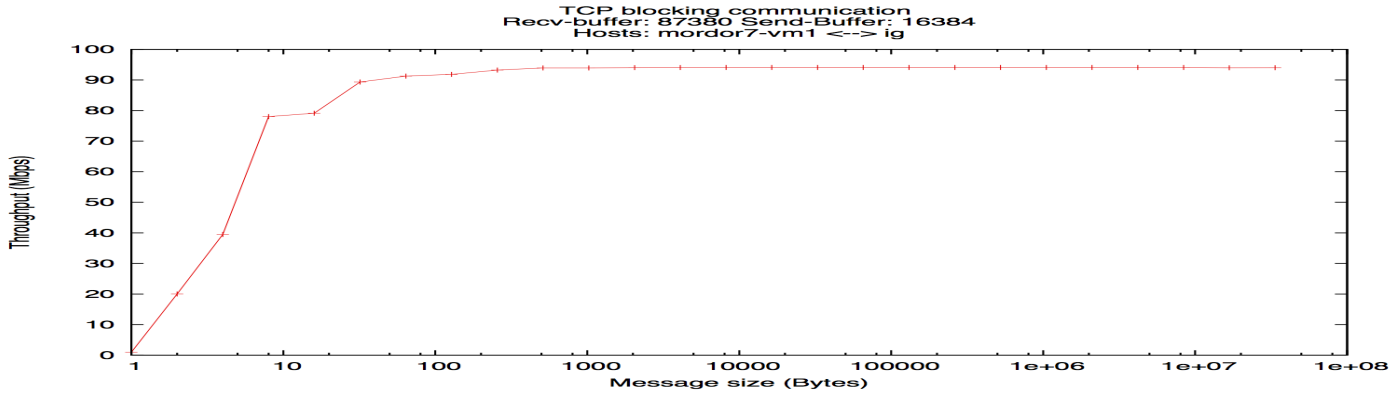
Network Components

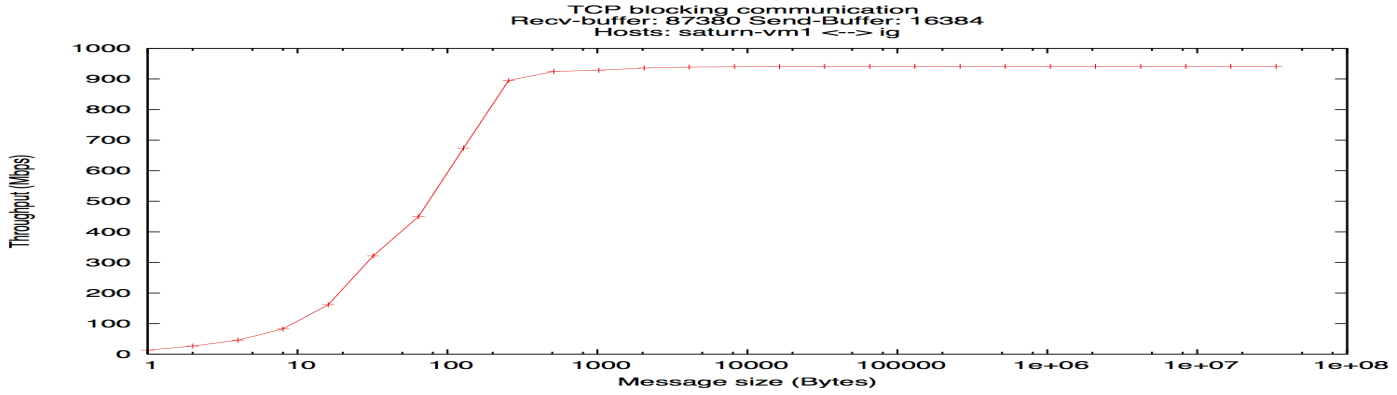
PAPI also has components for measuring Network I/O.

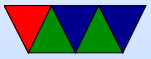
- Generic network component
- Infiniband component
- Myrinet component



Infiniband DirectPath Comparison



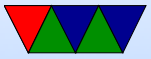




VMware Component

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