Characterizing the Performance of Parallel Applications on Multi-Socket Virtual Machines

Costin Iancu
Khaled Ibrahim, Steven Hofmeyr

Lawrence Berkeley National Laboratory

Discovery 2015 – HPC and Cloud Computing Workshop
June 22 2011

Motivation

❖ Virtualization is an enabling technology

- Resource consolidation
- Fault tolerance & isolation

❖ Virtualization Performance Expectations

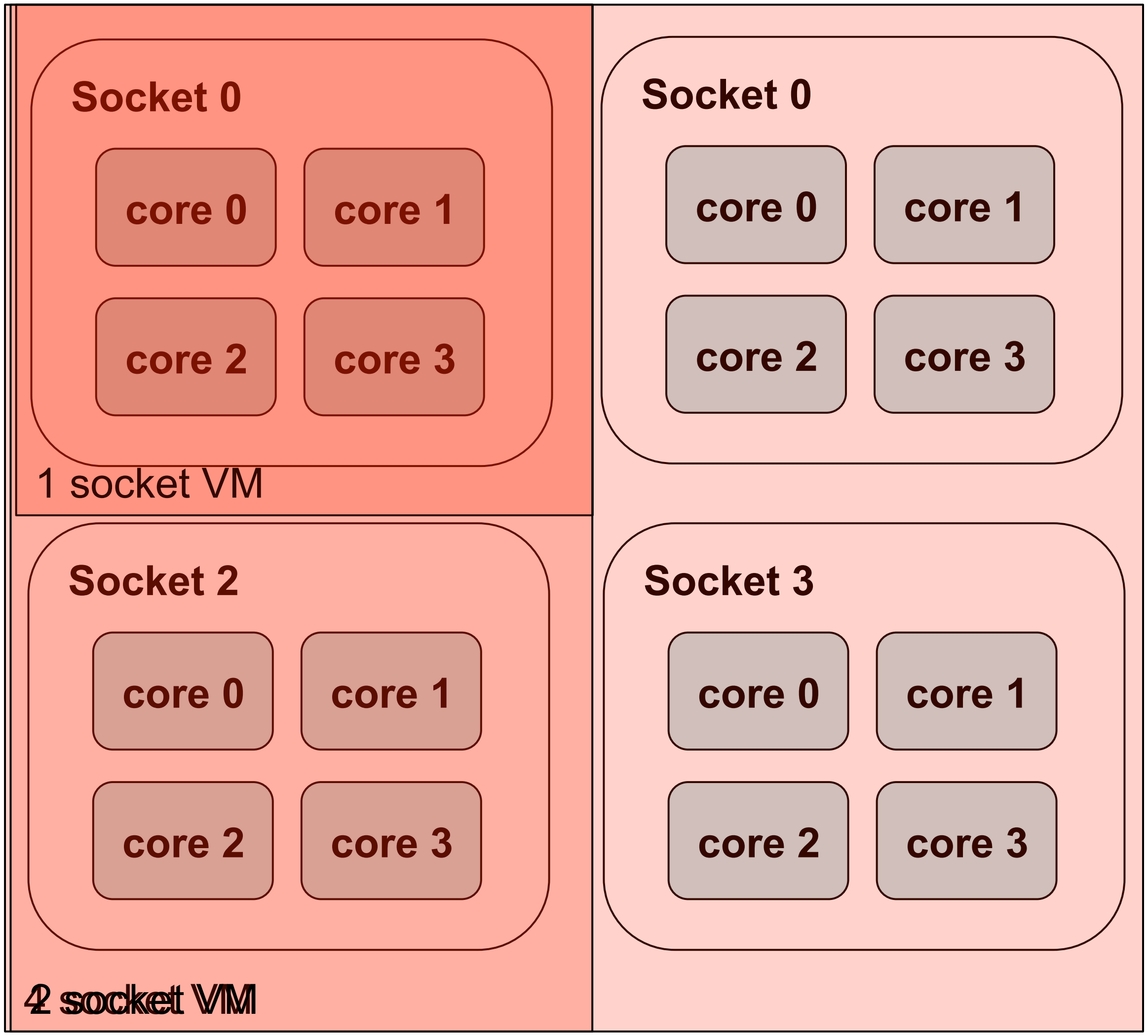
- Performance overhead is low (3-5% of raw)
- Current design and performance tuning techniques good enough!

❖ HPC Workloads

- Persistently use a large fraction of the system memory
- Data locality determines performance – NUMA support
- Sensitive to network bandwidth and latency – I/O support
- Use shared and/or distributed memory programming models – configuration/ software support
- Most HPC studies are single socket or on dual core systems

Virtualization Overhead

- **Three configurations**
  - 1 socket VM
  - 2 socket VM
  - 4 socket VM
- **Two architectures**
  - UMA
  - NUMA
- **Three programming models**
  - MPI
  - UPC
  - OpenMP



Socket 0
core 0
core 1
core 2
core 3
1 socket VM

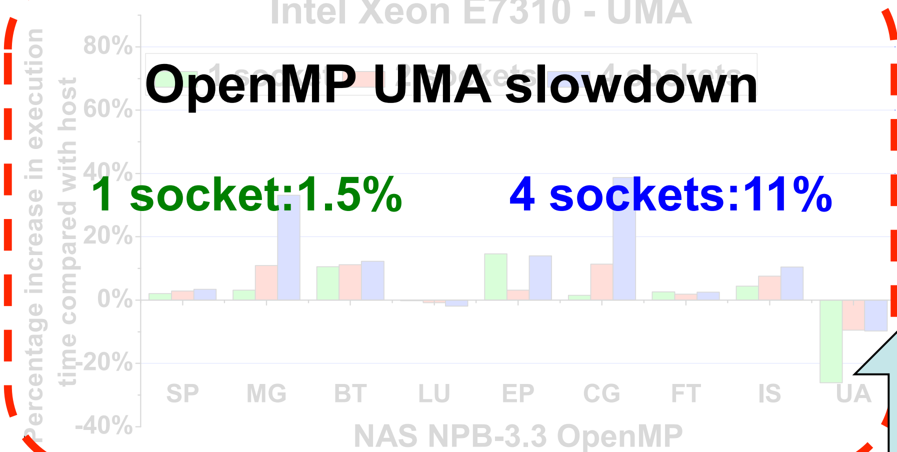
Socket 0
core 0
core 1
core 2
core 3

Socket 2
core 0
core 1
core 2
core 3

Socket 3
core 0
core 1
core 2
core 3

2 socket VM
4 socket VM

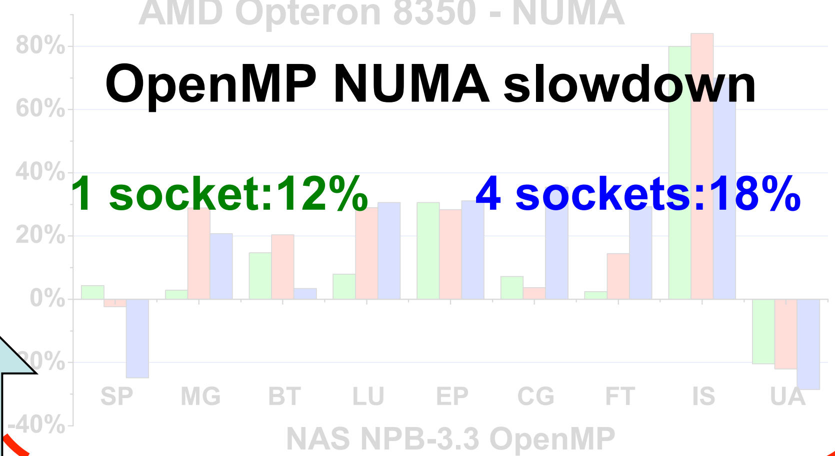
Performance on KVM



OpenMP UMA slowdown

1 socket:1.5%

4 sockets:11%

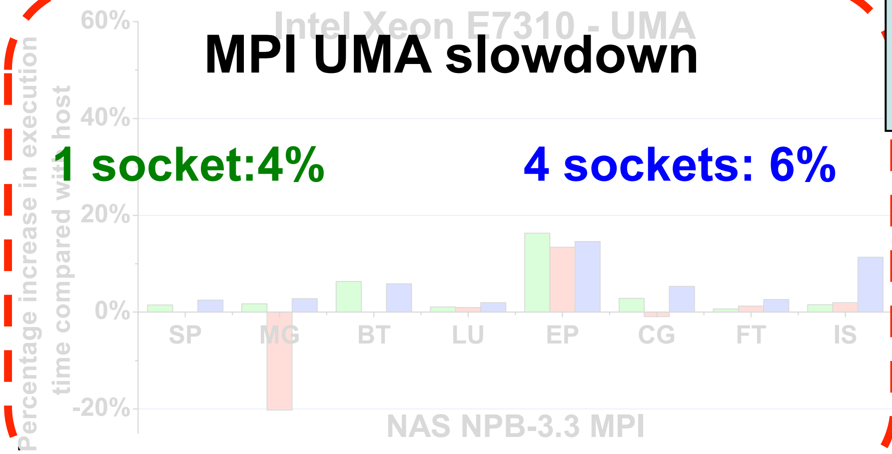


OpenMP NUMA slowdown

1 socket:12%

4 sockets:18%

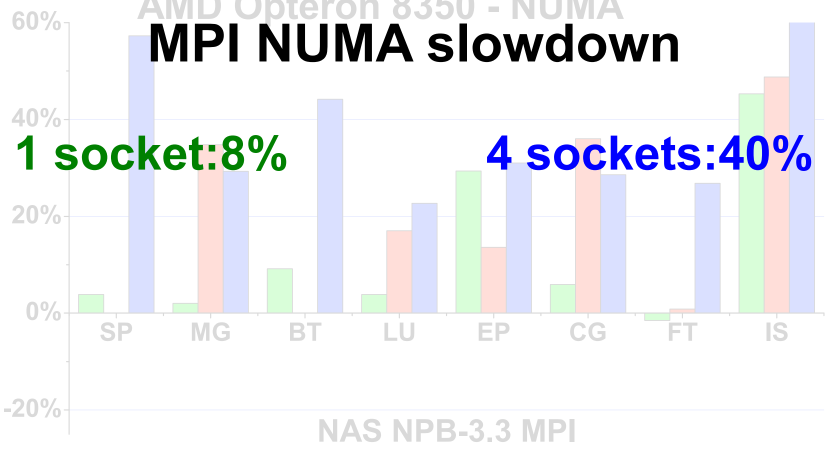
worse



MPI UMA slowdown

1 socket:4%

4 sockets: 6%



MPI NUMA slowdown

1 socket:8%

4 sockets:40%

NPB Performance Trends

- **Single socket performance is OK (KVM and Xen, matches performance expectations)**
- **Multi-socket UMA performance is OK ~ 10%**
- **High performance degradation when VMs span multiple NUMA domains:**
  - KVM on average 40%
  - Xen on average 233%
- **MPI seems to be slightly more affected than OpenMP**

Main Topics

1. Reasons for performance degradation on multi-socket NUMA
2. Interaction between programming models and Virtualization
3. Techniques to improve NUMA support

Experimental Setup

- **Virtualization technology full H/W support for memory and I/O**
  - **KVM/QEMU 0.13.0**
  - **Xen 4.0**

- **NUMA support**
  - Xen 4.0 - NUMA support is the default setting for the hypervisor
  - Qemu-kvm allows NUMA emulation on the guest.

- **Benchmarks NAS Parallel benchmarks (3.3)**
  - MPI
  - OpenMP
  - UPC (Unified Parallel C)
- **Architectures-** Linux (Kernel 2.6.32.8)
  - 4X4 UMA : Tigerton Xeon(R) CPU E7310
  - 4X4 NUMA: AMD Opteron(tm) Processor 8350
  - 2X4 NUMA: Intel Xeon E5530 (Nehalem EP).

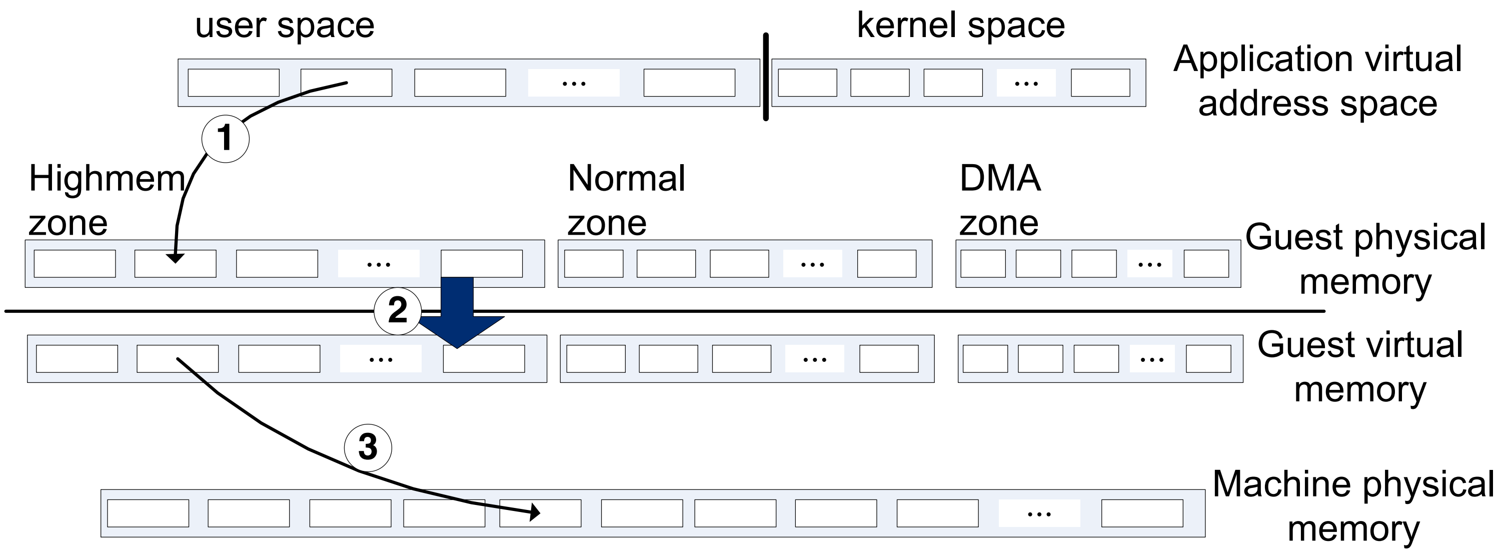
NUMA Support

- **Vendor provided (Xen, KVM, VMWare, OpenBox, etc)**
  - Hypervisor manages NUMA locality of pages.
  - Guests are typically architecture neutral.
- **NUMA Page allocation**
  - On-demand: KVM, VMWare.
  - Pre-allocation: Xen (problematic for NUMA)
  - Two level translation (Xen, VMWare), three level (KVM)
- **Xen (The other open-source)**
  - 233% average slowdown (compared with 40% for KVM).
- **VMWare – restricted info**
  - Limited vcpus
  - Guest is not NUMA aware
- **Vendors advocate node confinement (1 VM per NUMA Domain).**

Achieving Locality on NUMA

- **Enabling NUMA, pinning and page granularity do not provide good multi-socket NUMA performance.**
- **Page granularity might affect performance**
  - Minor effect in our experiments.
- **Node confinement (1 VM per NUMA Domain).**
  - Implicitly assumes first-touch allocation
  - Requires pinning VMs and workloads, etc
  - Multi-socket?!
- **Is current support enough?**

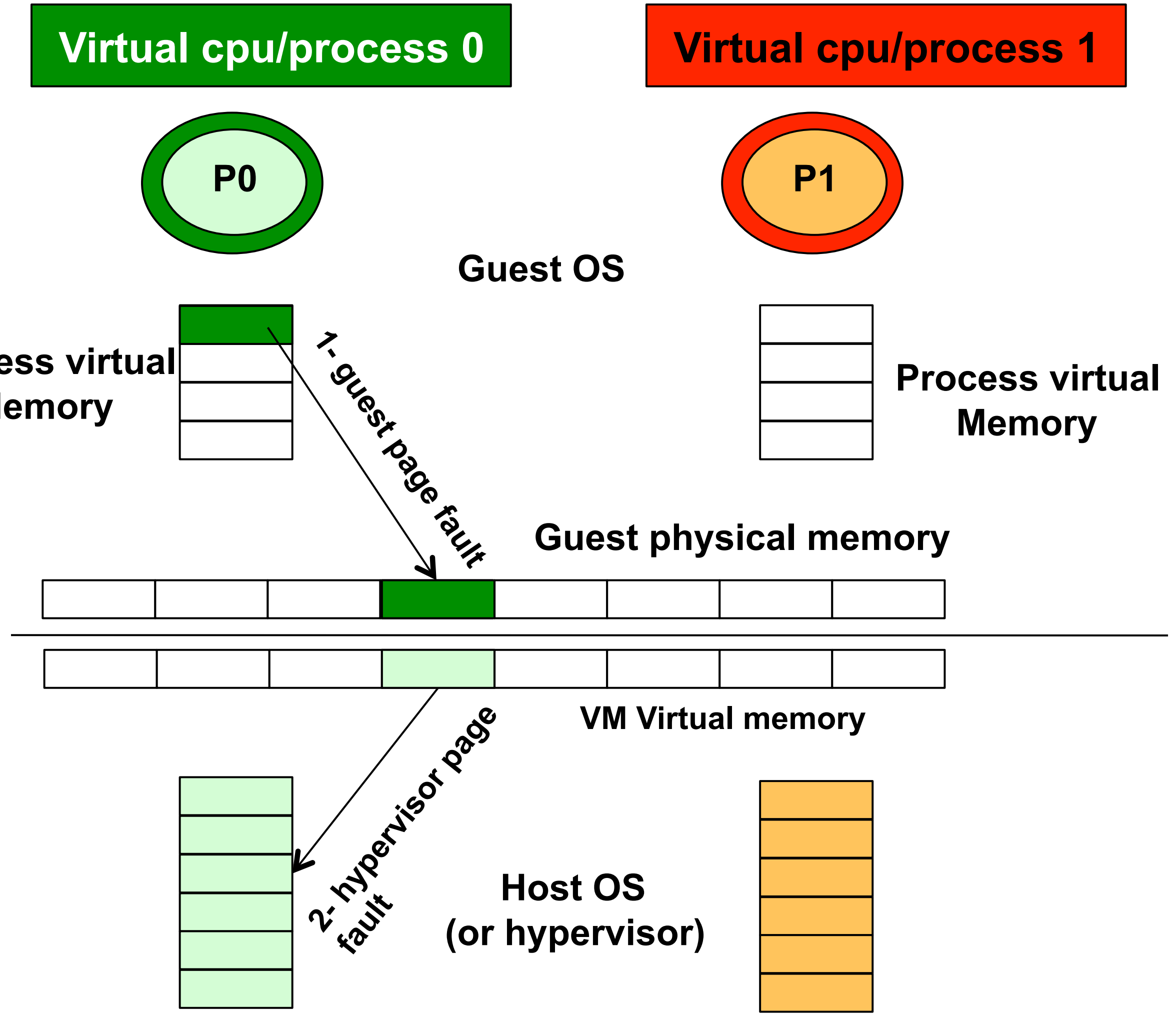
Page Translation (QEMU/KVM)



- Three stage translation
  - 2 Dynamic (runtime) and one static (launch time)

Application Starts

- Cold touch involves two page faults
  - Guest fault (NUMA oblivious)
  - Hypervisor fault (NUMA aware)



Virtual cpu/process 0

Virtual cpu/process 1

P0

P1

Guest OS

Process virtual Memory

1- guest page fault

Process virtual Memory

Guest physical memory

VM Virtual memory

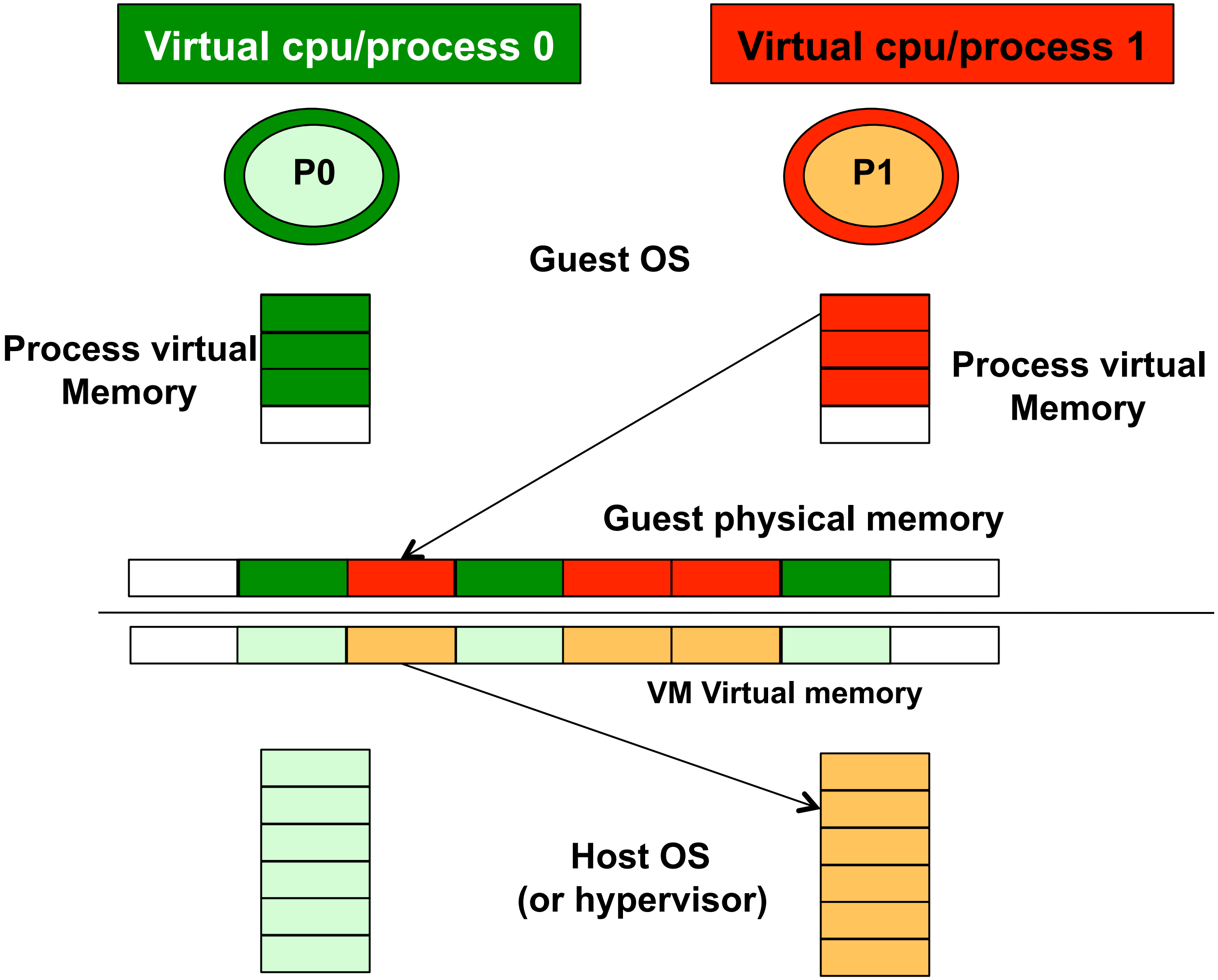
2- hypervisor page fault

Host OS (or hypervisor)

Two phase translation mechanism for application for the first touch of a guest page

Multiple Page Fault Outcome

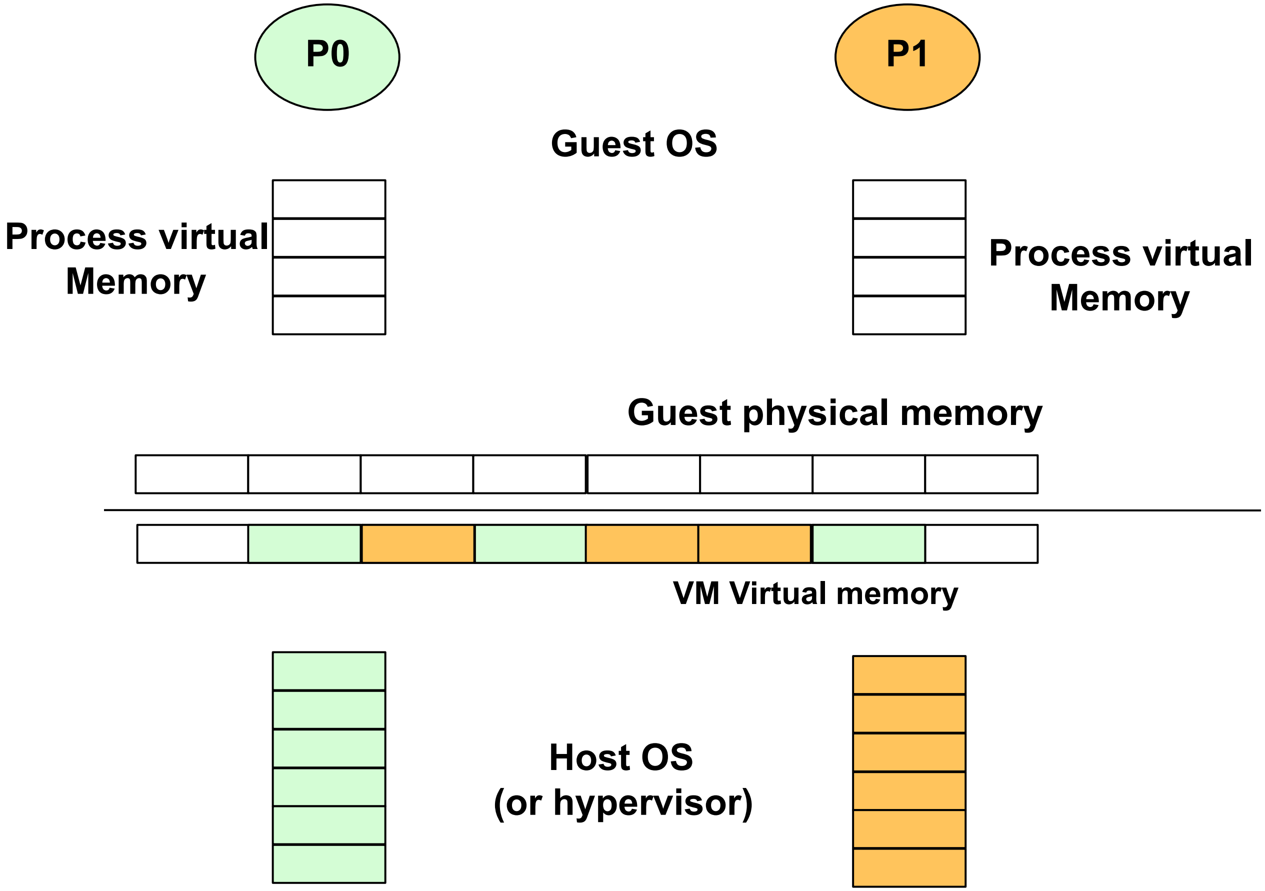
- Correct NUMA affinity is managed by hypervisor.



Two phase translation mechanism for application for the first touch of a page

Application Terminates

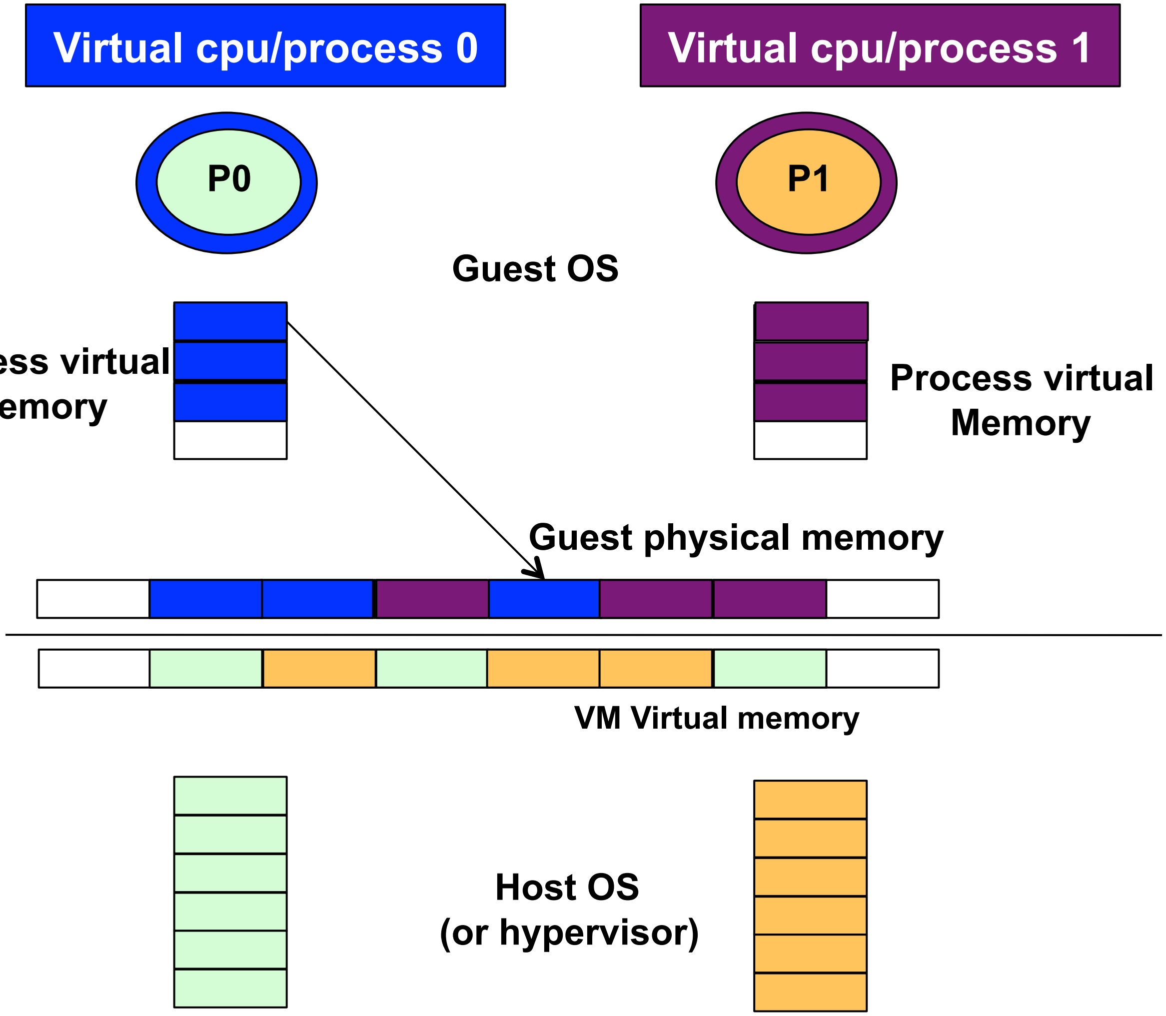
- Memory mappings in hypervisor are persistent.



System image after application termination.

New Application is Launched

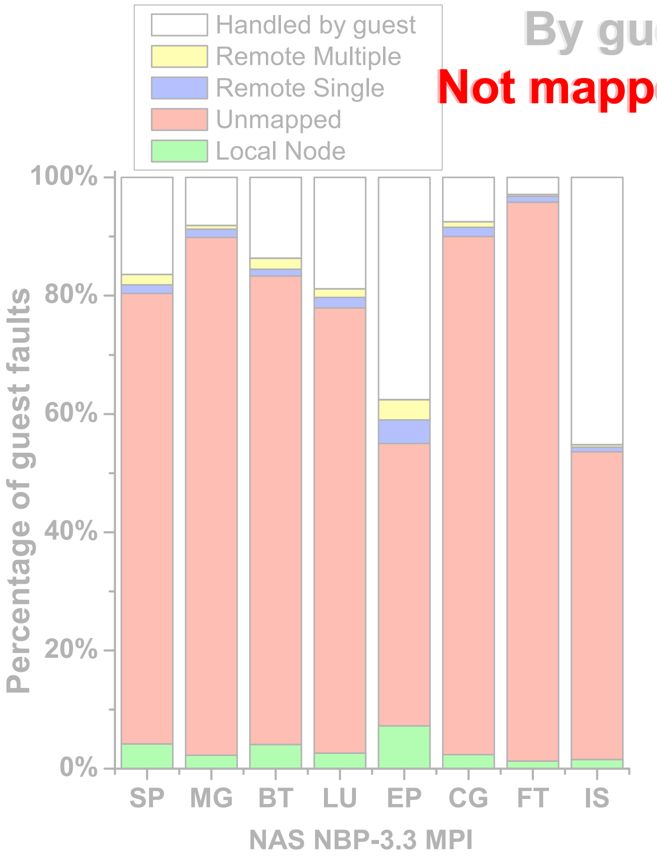
- Hypervisor mapping is recycled and locality is not guaranteed.



Page reuse results in host only page fault

Page Faults in KVM

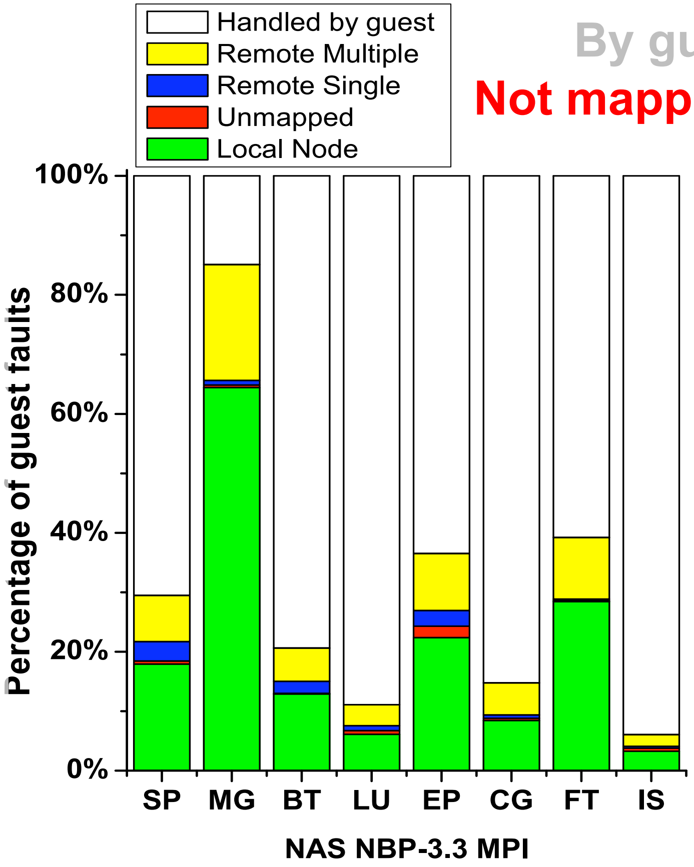
Cold VM



By guest 18%

Not mapped 75%

Warm VM



By guest 70%

Not mapped 0.6%

NUMA Support in KVM

- Hypervisor can provide locality
- Page faults are filtered by guests – do not reach hypervisor

Main Topics

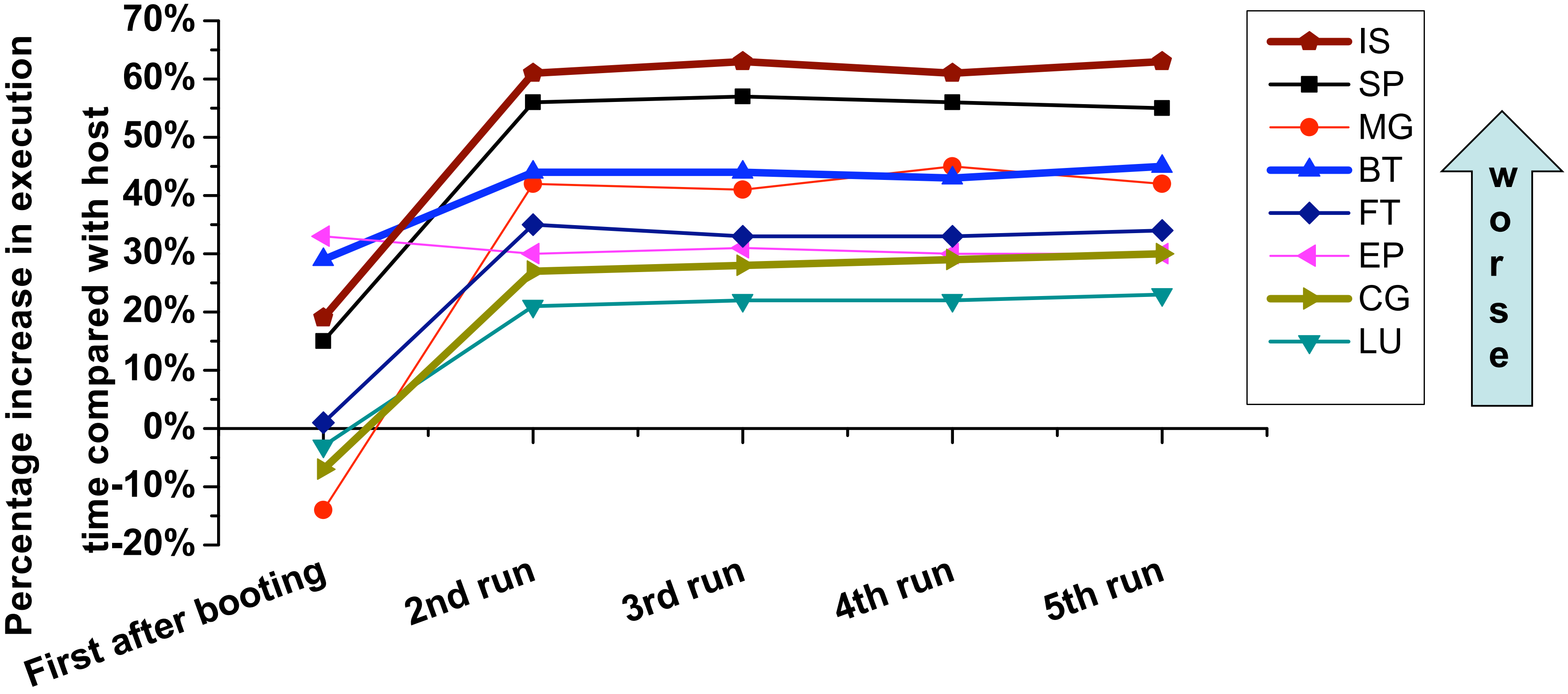
1. Reasons for performance degradation on multi-socket NUMA

2. **Interaction between programming models and Virtualization**

3. Techniques to improve NUMA support

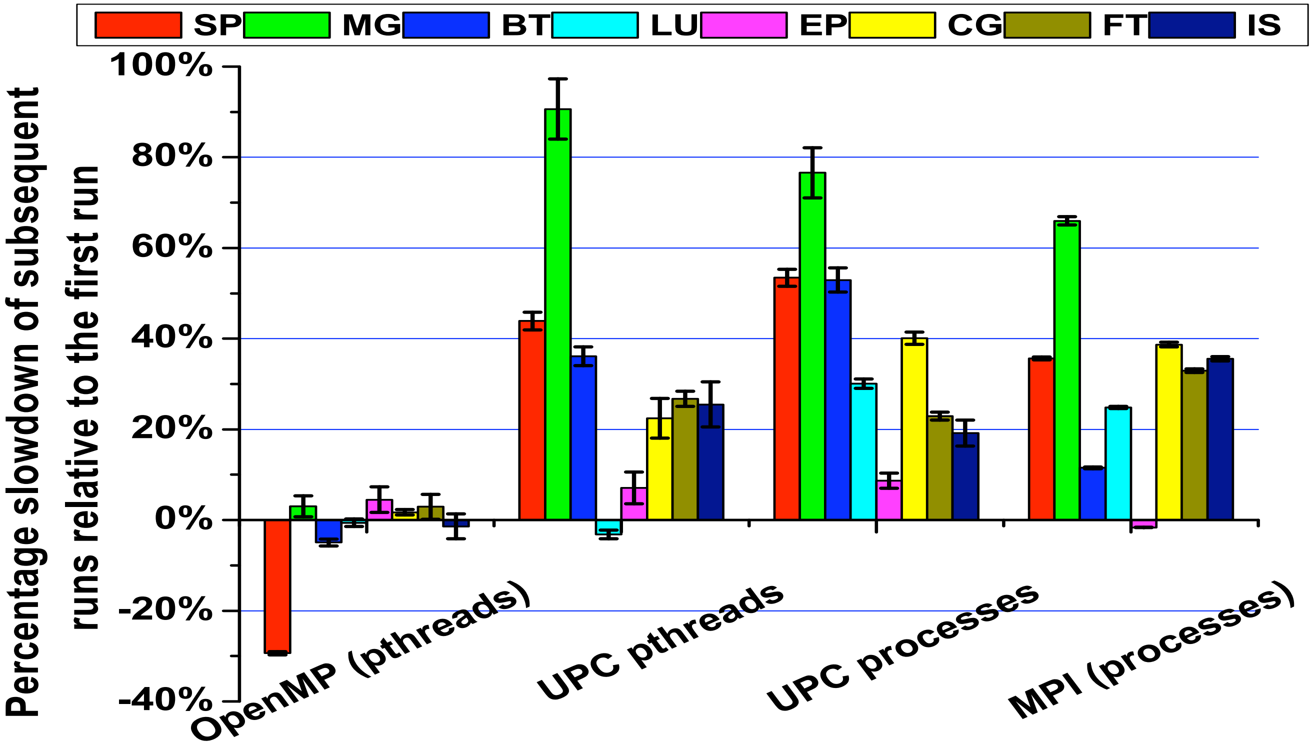
MPI Behavior

Warm VMs provide lower performance!



First run avg. slowdown: 9%, second run avg. slowdown: 40%

Distributed vs Shared Memory



- **Shared Memory (OpenMP)**
  - No locality. Remote data are fetched each time they are needed.
- **Distributed Memory (MPI and UPC)**
  - Implicit/explicit locality. Copy data locally before referencing them.

Main Topics

1. Reasons for performance degradation on multi-socket NUMA

2. Interaction between programming models and Virtualization

3. **Techniques to improve NUMA support**

Improving NUMA Support

- **How to improve locality?**
  - Hypervisor?
  - Guest?
  - Application? Shell? Runtime?
- **Expose NUMA architecture to the guest**
  - “Enlightenment” proposal for Xen
- **Modify memory management**
  - Page migration – hypervisor
  - Fault propagation – guests, hypervisor
  - Configuration/services
- **Use node confinement (partitioning)**
  - Transparent/configuration
  - With support – hypervisor, runtime

Exposing NUMA

- **Expose NUMA architecture to the guest**
  - How to over-commit memory?
  - Can we handle non-contiguous NUMA nodes?
  - How to flexibly manage memory of the VMs (reclamation, for instance)?
  - How to resize memory?
  - How to migrate VM to a non-compatible destination?
  - Can the hypervisor commit to guarantee page node allocation?

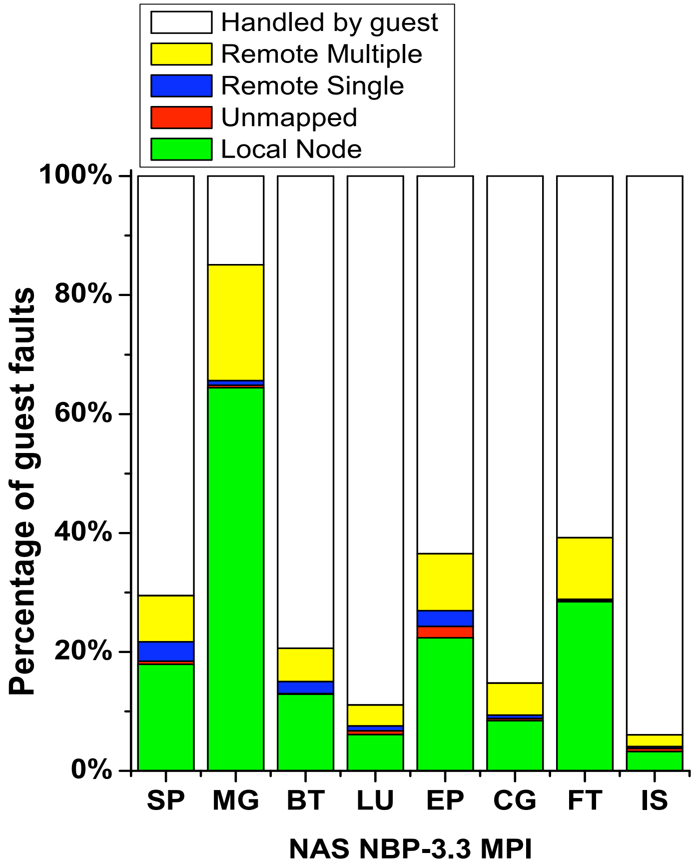
Void virtualization benefits!

Modified Memory Management

- Page Migration: fix locality for badly mapped pages.
  - **1% remote single**

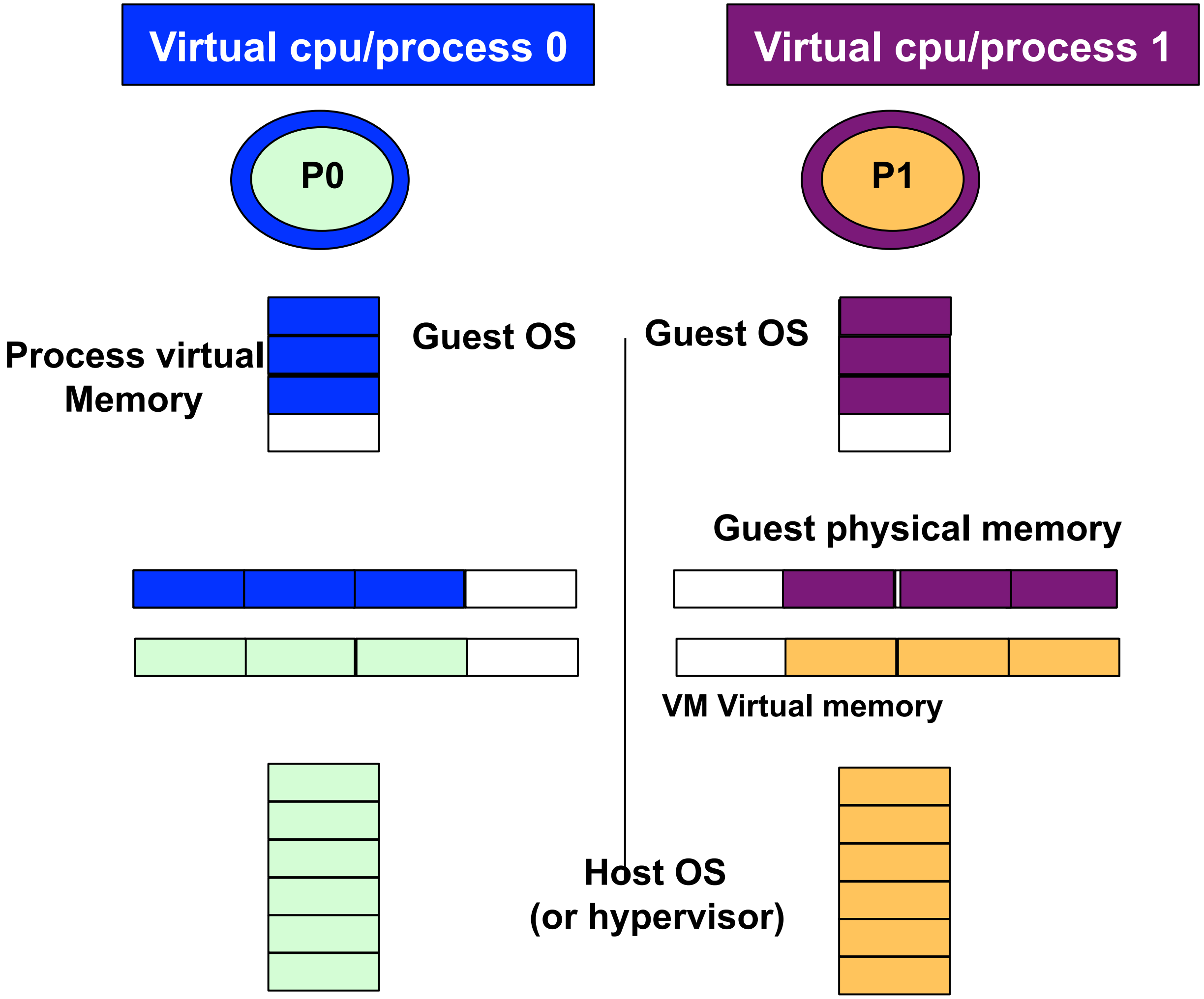
- Most faults handled by the guest **(70%)**
- Propagating **faults** requires changes to all guest Oses
  - Fast allocation
  - Slow reclamation

Page faults propagated to the hypervisor



VM Node Confinement (Partitioning)

- **Vendors advocate node confinement**
- **Performance:**
  - Resource Contention
  - Inter-VM communication

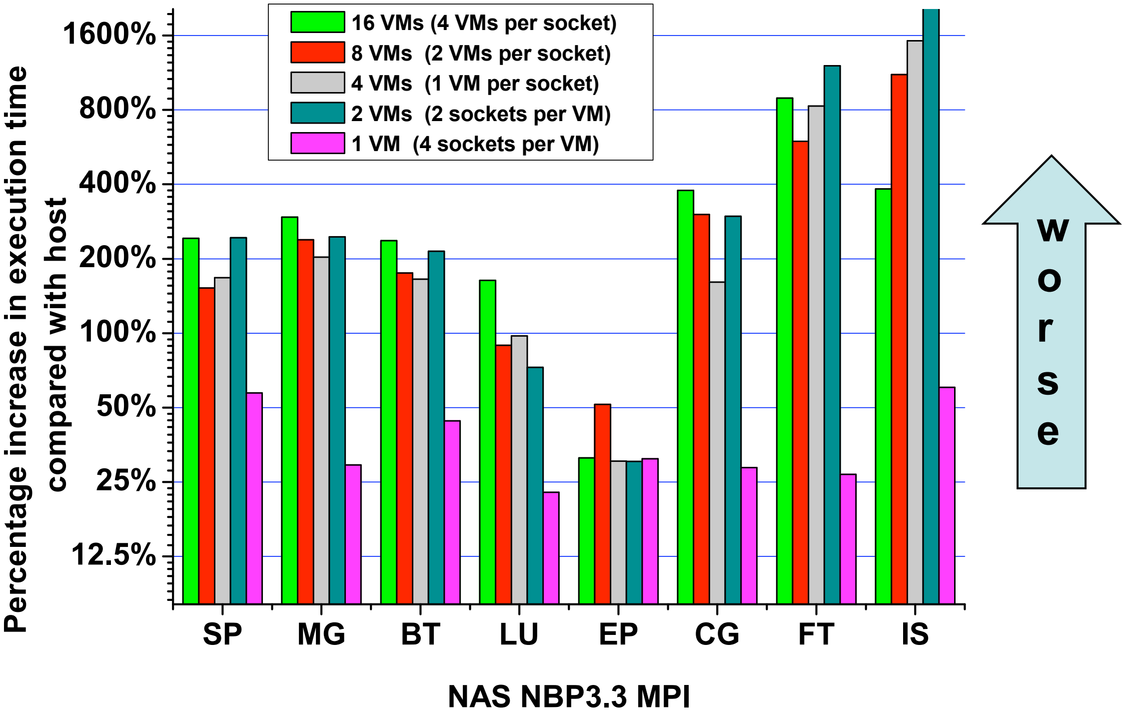


Page reuse results in host only page fault

Partitioning for HPC

1 VM per node 40% slowdown

1 VM per NUMA domain is 400% slowdown

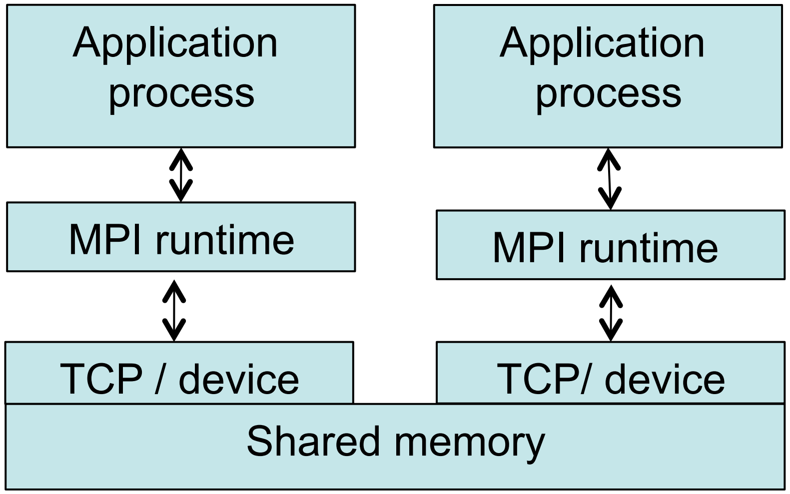


- Up to **16x** performance degradation, mostly more than **2x**.
- HPC workloads depends on efficient inter-VM communication.

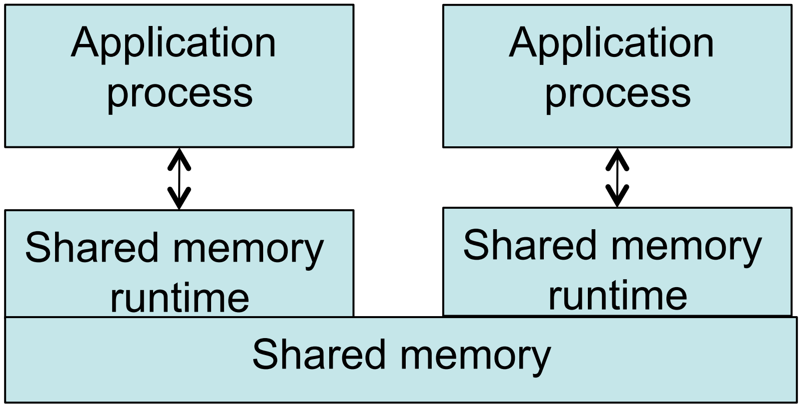
Virtualized I/O

- Earlier proposals implement communication stack over-shared memory
  - Zhang et al [Middleware'07] IP over shared memory.
  - Huang et al. [SC'07] introduce IVC stack
  - ***virtio*** essentially does the same.
- The bottleneck is in using the software stack.
- Instead, we implement inter-VM communication natively on top of shared memory.

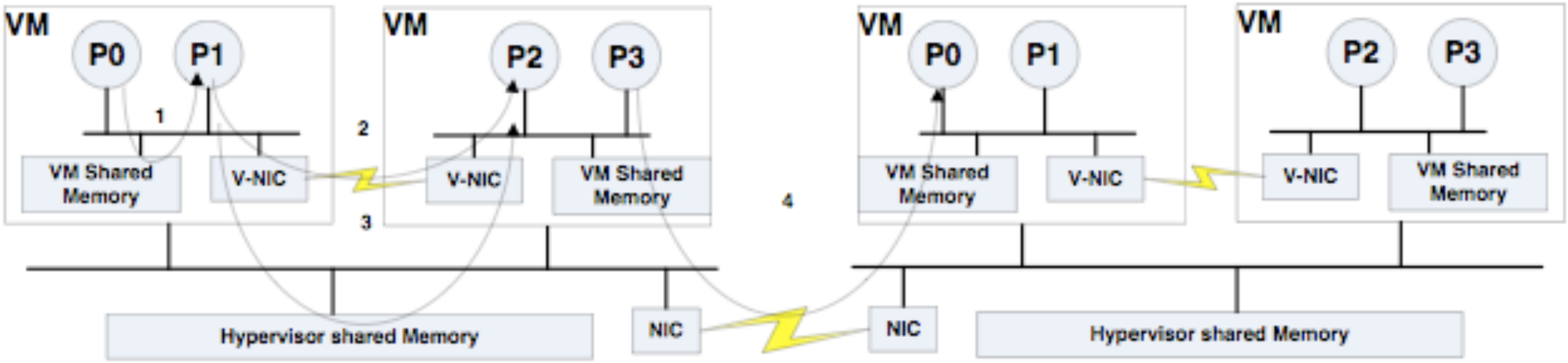
Typical communication across nodes



Typical communication within a node



Inter-VM Communication for OpenMPI



1- MPI communication within a node using shared memory

2- MPI communication between VM using virtual NIC.

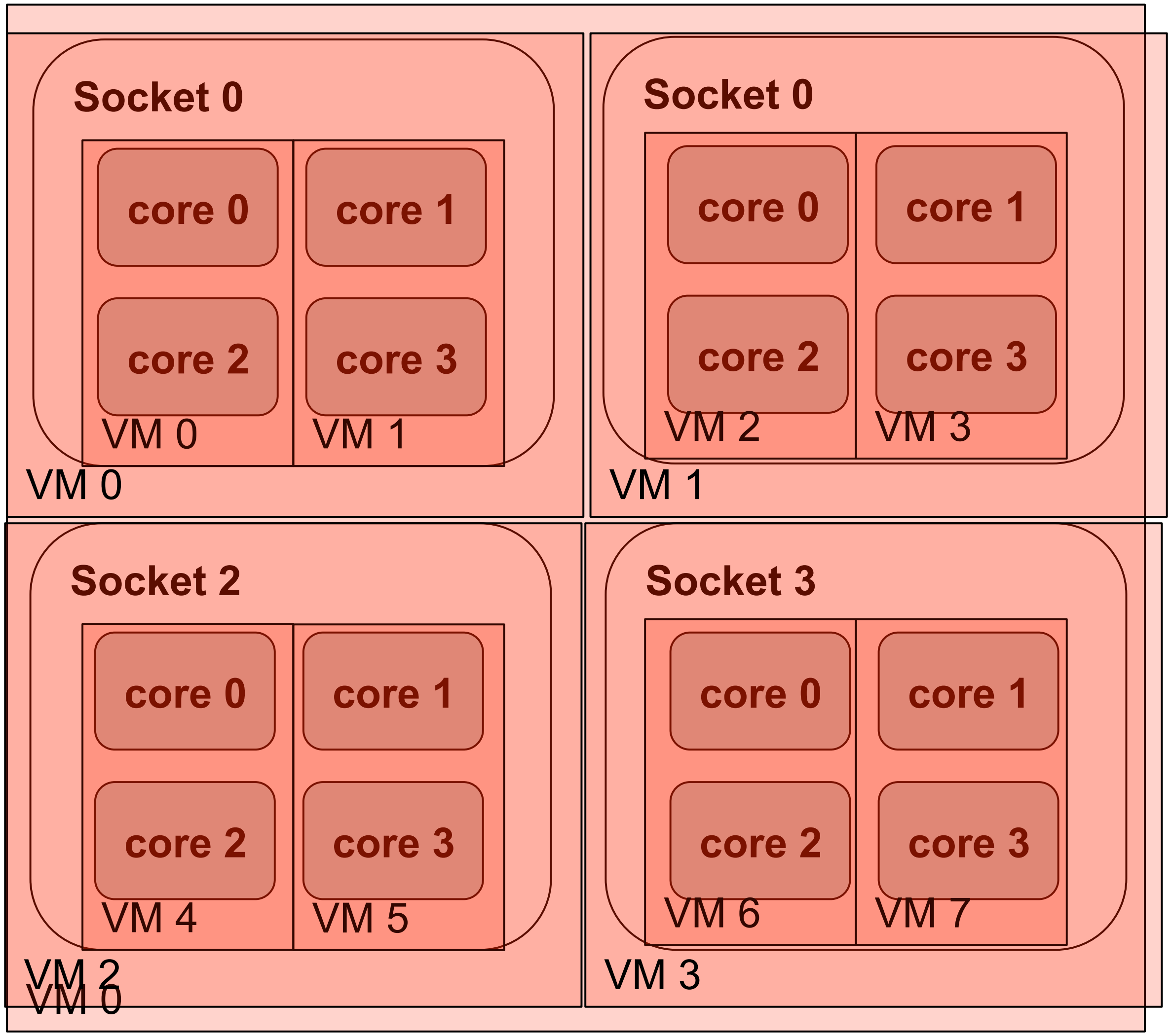
4- Communication between VMs across nodes using normal NIC interface.

3- communication between VMs using Virtual shared memory BTL.

- Shared memory exposed to guest as PCI device memory (ivshmem driver)
- Three new components handle the shared memory between different VMs
  - VM Shared memory communication component.
  - VM memory pool communication component.
  - VM collective communication component.
- New selection mechanism for communication component.
- Similar mechanism is implemented for UPC

VM Partitioning Schemes

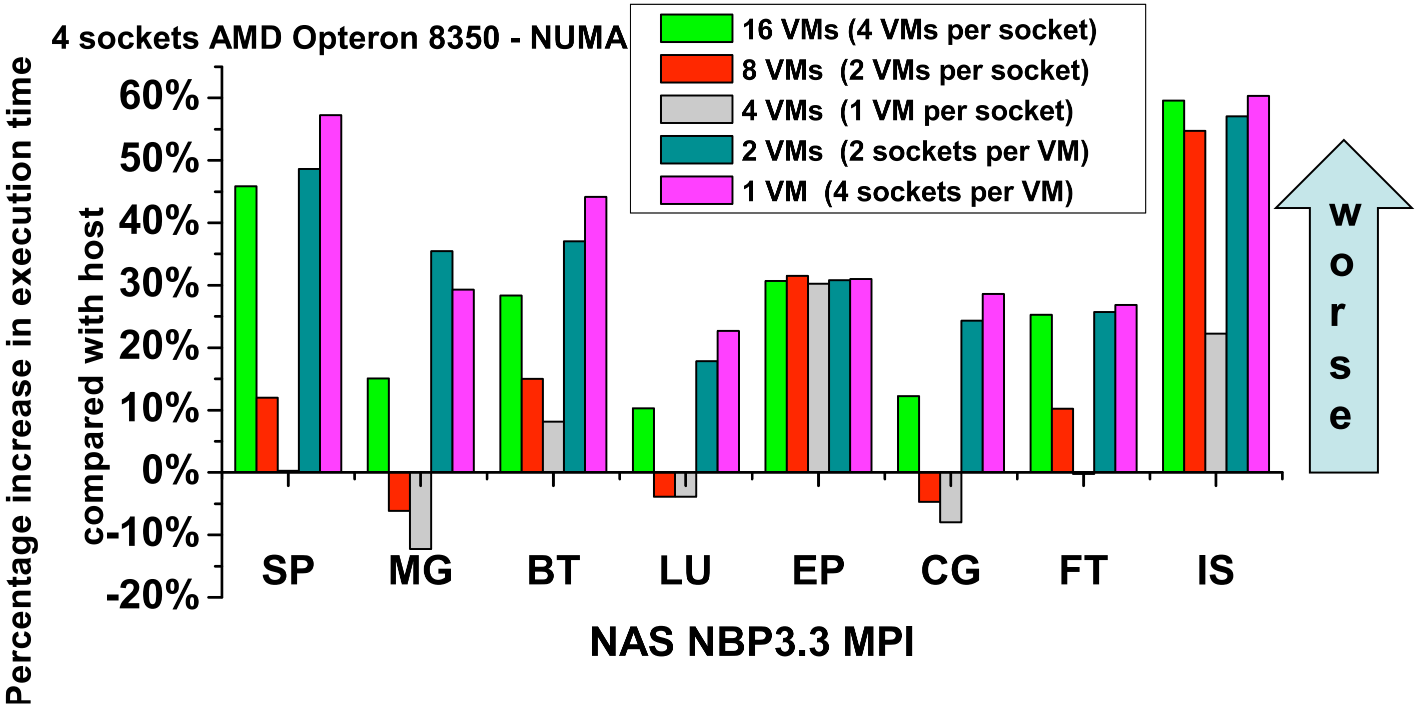
- Partitioning strategy
  - 1 VM (4 socket per node)
  - 4 VM (1 socket per node)
  - 8 VM (2VM per socket)
  - …



Partitioning and Inter-VM Shared Memory

One VM per node (1VM)
Slowdown: 40%

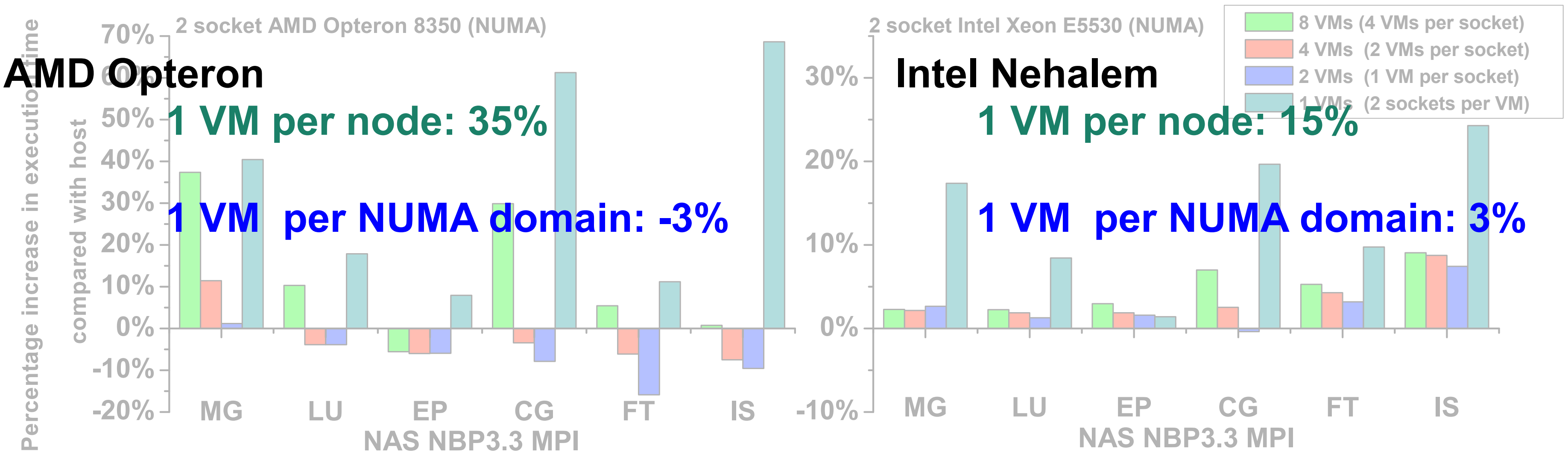
One VM per NUMA domain: (4VM)
Slowdown: 3%



- One VM per socket is usually the best configuration.
- Efficient Inter-VM communication is key to performance.

Partitioning and Inter-VM Shared Memory

- VM spanning sockets is always less efficient than multiple VMs with efficient inter-VM communication.



AMD Opteron

1 VM per node: 35%

1 VM per NUMA domain: -3%

Intel Nehalem

1 VM per node: 15%

1 VM per NUMA domain: 3%

Other Benefits of Partitioning

- **Partitioning and resource contention**
  - Introduces multi-level locking
  - Reduces “system” overhead – e.g. MPI on UMA 6%->3%

- **Partitioning and I/O**
  - KVM software driver – best is 1core per VM
    - MPI overhead: 17% on 32 VMs, 223% on 2 VMs
  - Virtio – best is 8 cores per VM (12%)
    - MPI overhead: 34% on 32VMs, 63% on 2VMs

Conclusions

- **The performance on NUMA machines is severely penalized if a VM span multiple sockets (avg. slowdown: 40% KVM, 223% Xen).**

- **NUMA cannot be handled by hypervisor alone**
  - Lacking (Xen), or hindered by guest (KVM locality leakage).

- **VM partitioning requires efficient inter-VM communication**
  - Better than virtualized IO or communication stack on top of shared memory.
  - Our implementation reduces slowdown to 3% on average.

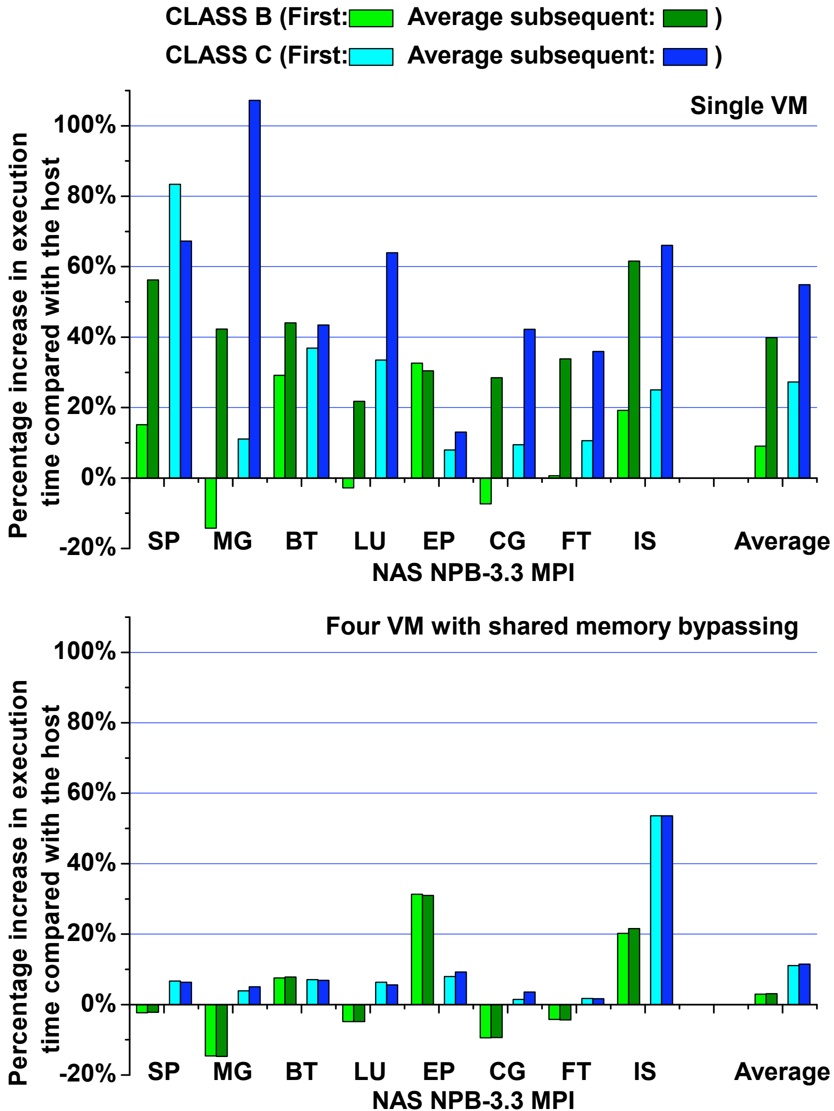
- **Other solutions may be needed for shared memory programming models, for instance OpenMP.**

Questions

Thanks for attending!

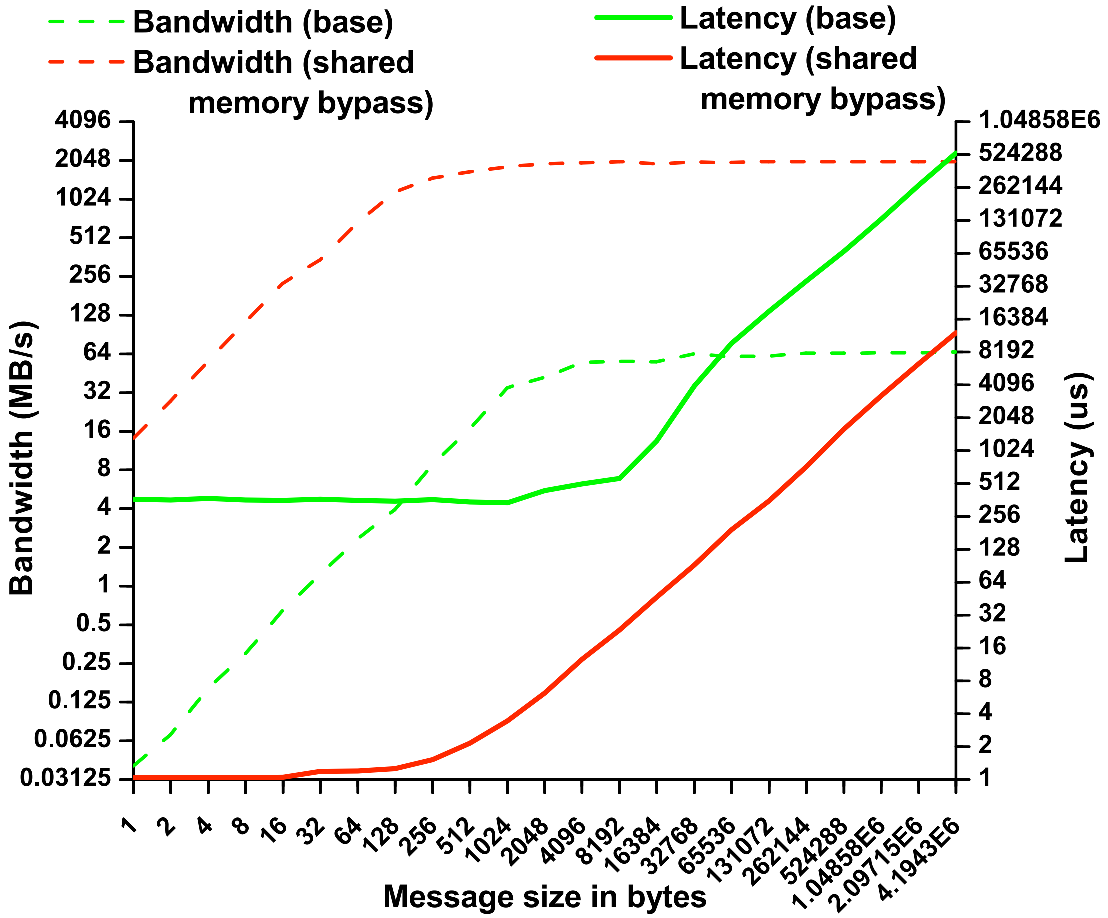
Impact of Dataset

- First run performance becomes less optimal for the large dataset.
- Less data are cached so bad locality is associated with higher cost.
  - Class B: avg. 40% slowdown (up to 61%)
  - Class C: 57% in average (up to 105%)
- With partitioning and efficient communication
  - Class B: avg. 3%
  - Class C: avg. 11%



Network Performance

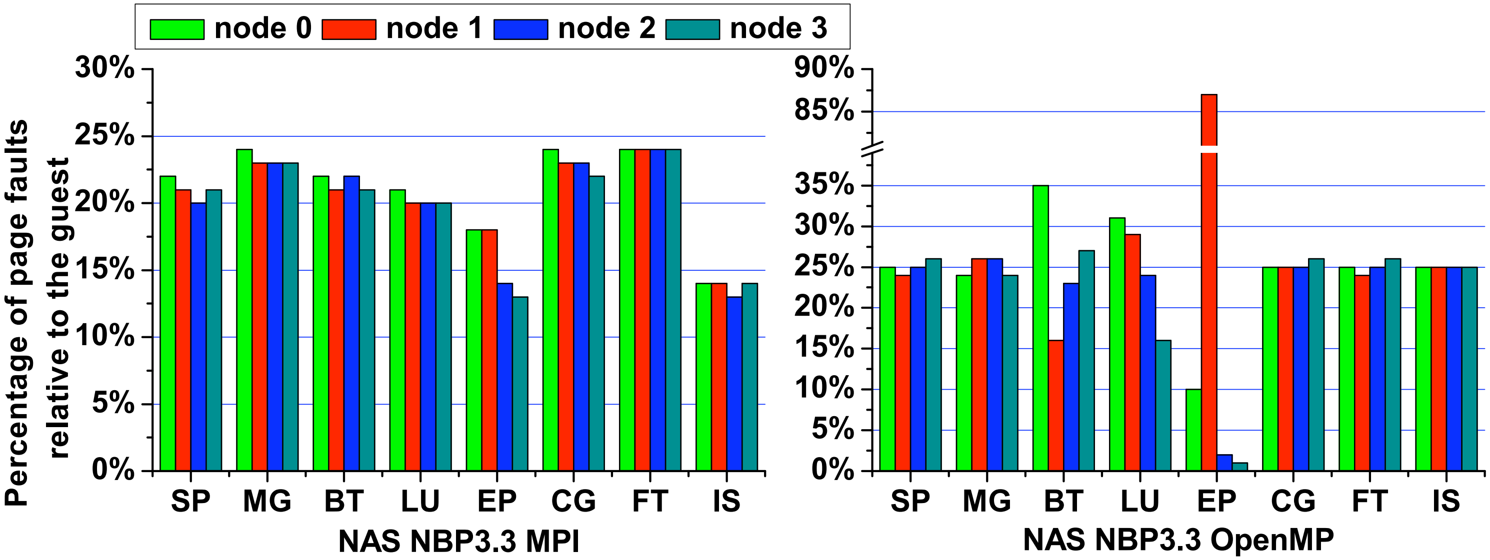
- MPI network performance for TCP network vs shared memory bypassing.



MPI vs. OpenMP

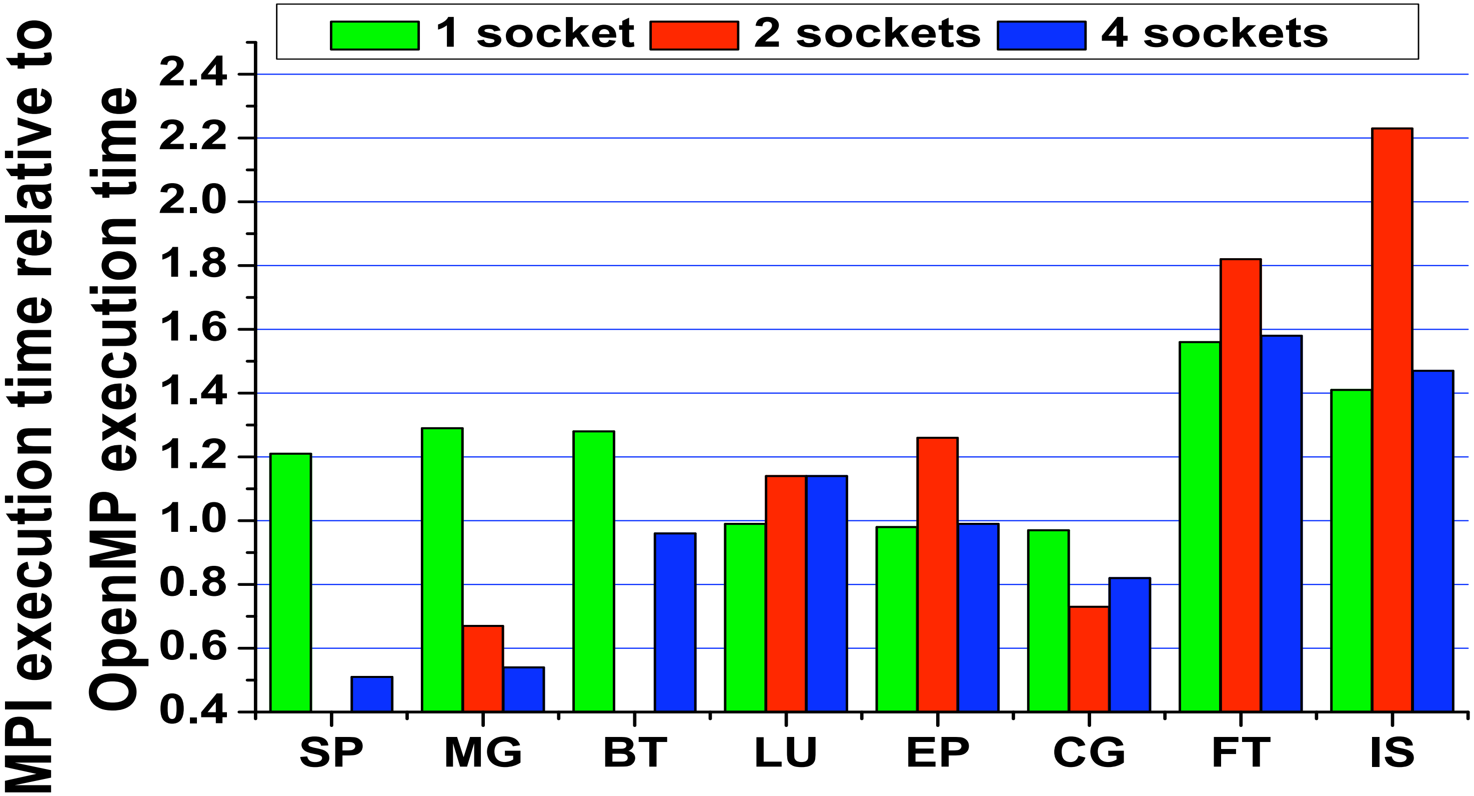
- Benchmark implementations have similar NUMA domain distribution (have well balanced page fault distribution across domains)
- The implementation of the programming model affects behavior:
  - pthread model vs. processes (Higher percentage of faults exposed in the first run for OpenMP.)

- NUMA distribution + implementation of runtime do not explain perforamance differences



MPI vs. OpenMP Performance

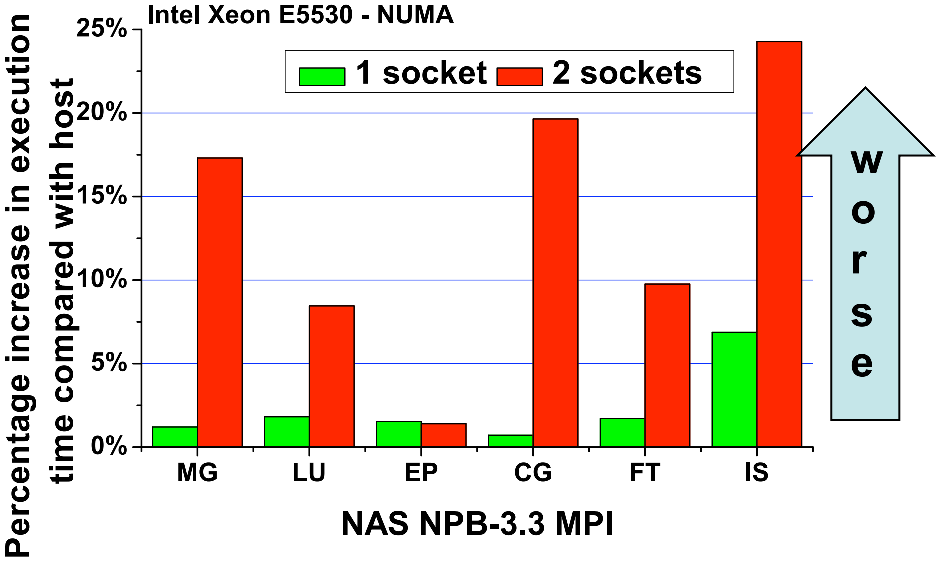
- < 1 ➔ MPI is better



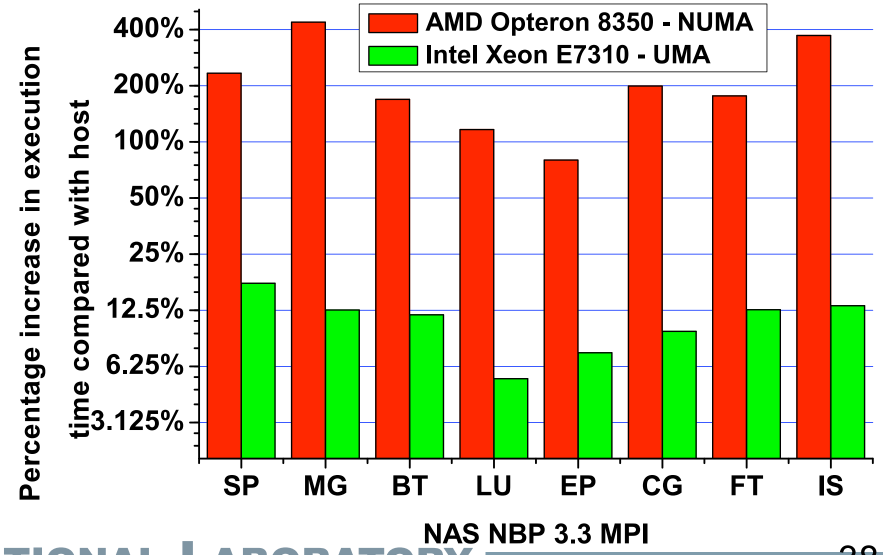
NPB Performance Trends

- Single socket performance is OK (KVM and Xen, matches performance expectations)
- Multi-socket UMA performance is:
- High performance degradation when VMs span multiple NUMA domains:
  - KVM on average 40%
  - Xen on average 233%

- VMWare and HyperV
  - Limited number of vcpu per guest – node confinement
  - Restrictions in reporting performance in addition to lack of source code.



Xen 4.0 – 4 sockets



Inter-VM communication

- Xen – GrantTables
- KVM - Base shared memory is PCI-based IOMEM driver (an extended version of ivshmem) driver.

- Severe restrictions on sizes – MPI works, UPC not
- Breaks migration ? What else?
- Does not work for OpenMP

Communication module support in openMPI

- OpenMPI is based on Open Component architecture.
- Communication is done through communication components that are chosen based on runtime condition.
- Shared memory BTL is higher priority (higher exclusivity) transport layer than all other network (only less than self).
- Each processor tries to find all transport modules (BTLs) that it can use to reach each destination processors. The highest exclusivity BTL win the registration competition.

1- MPI communication within a node uses shared memory (using sm BTL)

2- MPI communication across nodes uses the fastest available network card (using one of the tcp, IB, ... BTLs).