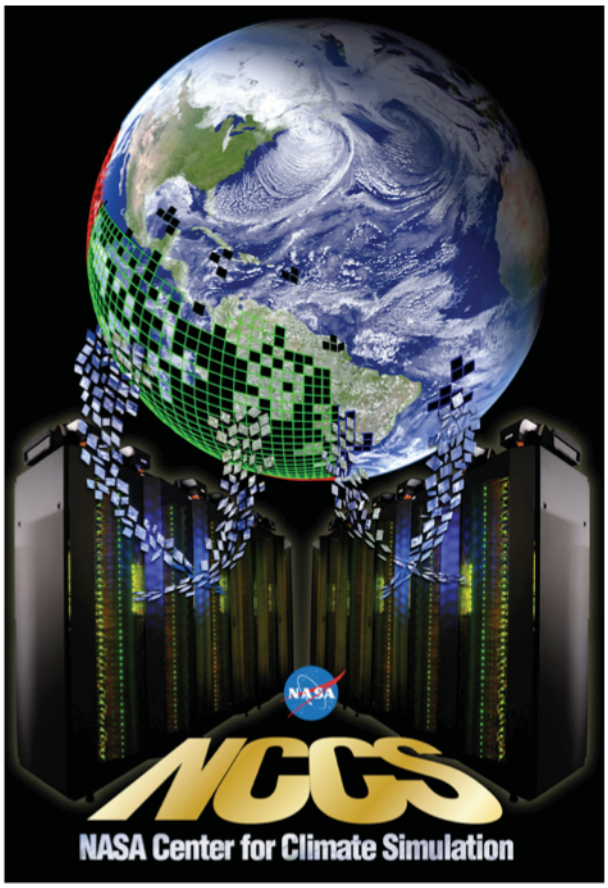

NASA Center for Climate Simulation (NCCS)
Advanced Technology AT5
Virtualized Infiniband Report

01 May 2013

Hoot Thompson (NASA GSFC), Ben Bledsoe (NASA GSFC), Mark Wagner (Red Hat),
Dr. John Shakshober (Red Hat), Alex Neefus (Mellanox), Russ Fromkin (Intel)

Advanced Technology AT5
Virtualized Infiniband Report
Table of Contents

List of Figures

1 Introduction

The NCCS is part of the Computational and Information Sciences and Technology Office (CISTO) of Goddard Space Flight Center's (GSFC) Sciences and Exploration Directorate. The NCCS's mission is to enable scientists to increase their understanding of the Earth, the solar system, and the universe by supplying state-of-the-art high performance computing (HPC) solutions. To accomplish this mission, the NCCS (https://www.nccs.nasa.gov) provides high performance compute engines, mass storage, and network solutions to meet the specialized needs of the Earth and space science user communities.

2 Background

The NCCS, as part of its charter, continuously investigates avenues to expand its processing footprint either through the acquisition of on-site resources or leveraging capabilities available on a burst or as-needed basis. Cloud and related technologies present a new alternative to addressing potentially elastic computational demands and simultaneously providing a degree of flexibility over fixed architectures. Today cloud computing has become a widely accepted paradigm for datacenter design justified on the basis cost savings stemming from moving applications to a cloud based service. These cost savings come from two factors. First operational cost savings like power, space and reduced management requirements. Secondly through lower capital expenditures realized by reducing the overall amount of hardware required to host an enterprises application. This is done either by running more applications on a physical machine using virtualization or by moving applications to a lower cost public cloud solution.

HPC in the cloud offers benefits perhaps not meaningful to the traditional datacenter. In addition to providing a mechanism for managing processing demand surges, cloud-based HPC presents the option for:

- Special/temporary debug queues
- Customized run-time environments
- Code validation against older system images

As previously noted, references to the cloud imply servers running in virtual space. For cloud-based HPC to be practical, it must approximate the performance of a bare metal instantiation. This dictates a full-up HPC cluster with potentially hundreds or thousands of computational nodes, high-speed interconnects and fast shared storage all existing in virtual space with minimum degradation in comparison to its bare-metal counterpart.

This paper focuses on virtualized Infiniband as being the enabler for HPC in the cloud and correspondingly, single root I/O virtualization or SR-IOV as being the technology that makes it feasible. Most modern HPC clusters rely on Infiniband for node-to-node connections because of its high-bandwidth, low latency characteristics. Therefore getting it right in the cloud is key. But significant work is also being done in the software domain to lessen and/or almost eliminate any degradation of running in virtualized space. This paper explores virtualization by exercising an eight node test cluster first with benchmarks on the bare metal systems and then in an identically sized virtualized cluster. The work to date has been a collaborative initiative involving NCCS personnel with contribution from engineers at Red Hat, Mellanox and Intel.

3 Virtualization Technology

Before getting into the test setup and the benchmarks, key hardware and software virtualization technologies need to be introduced.

3.1 Hardware Host Virtualization

Two hardware based technologies that directly impact I/O performance are Single Root I/O Virtualization (SR-IOV) and Virtualization Technology for Directed I/O (VT-d). These technologies are inherent in Intel's modern processor chip families.

3.1.1 SR-IOV

Early techniques for network virtualization worked by doing network emulation. Fundamentally, the method was to create a software representation of a network and to allow the hypervisor operating system to handle the queuing and message delivery. This technique is known as paravirtualization. Paravirtualization works well because it allows a virtual machine (VM) to be completely agnostic to the type of hardware below.

While paravirtualization allows for very easy virtual network creation and migration it adds a heavy weight software emulation layer that adversely impacts the performance of the network IO. This is most noticeable when measuring message latency and CPU cycles per message. For instance, moving a packet through a paravirtulization framework can add a factor of 10x or more to the overall latency. To avoid this, most hypervisors offer a PCI pass-through option that gives one virtual machine direct access to the PCI network device. However, other VMs cannot access this device concurrently.

In this paper we will explore the viability of a third approach for high performance network device virtualization using a technique called SR-IOV. SR-IOV or Single Root IO Virtualization is a method of IO device partitioning done using PCIe semantics. Each device partition is called a virtual function and enumerated on the PCIe bus. The virtual functions can then be used by VMs using the direct pass-through method provided by the hypervisor. Mellanox has fully embraced this technology in its current and future products. For a thorough discussion of SR-IOV refer to the PCI-SIG SR-IOV Primer: An Introduction to SR-IOV Technology [1].

3.1.2 VT-d

To create VMs (or guests) a virtual machine monitor (VMM) or hypervisor acts as a host and has full control of the platform hardware. The VMM presents guest software (the operating system and application software) with an abstraction of the physical machine and is able to retain selective control of processor resources, physical memory, interrupt management, and data I/O [2].

A VMM supports virtualization of I/O requests from guest software. This is done in software using either of two well known models: Emulation of devices or Paravirtualization. A general reliability and protection requirement for these or any I/O-device virtualization (IOV) models is the ability to isolate and contain device accesses to only those resources that are assigned to the device by the VMM.

Intel VT-d is the latest part of the Intel Virtualization Technology hardware architecture. VT-d helps the VMM better utilize hardware by improving application compatibility and reliability, and providing additional levels of manageability, security, isolation, and I/O performance. By using the VT-d hardware assistance built into Intel's chipsets the VMM can achieve higher levels of performance, availability, reliability, security, and trust.

VT-d provides VMM software with the following capabilities:
- Improve reliability and security through device isolation using hardware assisted remapping
- Improve I/O performance and availability by direct assignment of devices

3.2 Software Virtualization

Testing for this paper was based exclusively on servers loaded with Red Enterprise Linux 6 (RHEL6). Understanding RHEL6's performance proposition requires a basic awareness of an aspect of computer hardware architecture referred to as Non Uniform Memory Architecture (NUMA). The non-uniformity portion of the NUMA acronym refers to the fact that it is faster to access memory directly attached to a CPU (a "local *access")* in comparison to accessing memory affiliated with another CPU (a "remote *access").* There are dramatic performance benefits realized from correctly scheduling compute tasks on processors having local memory accesses. A primary function of an operating system is to efficiently manage the allocation of computer resources, including CPU and memory access – optimizing the resource allocation to align compute tasks with local memory access is referred to as NUMA awareness. NUMA awareness is greatly complicated by the fact that workloads are dynamic. On running systems new tasks are constantly being created, memory is dynamically allocated and freed. Tasks migrate among free processors to utilize idle components.

The virtualization hypervisor layer in RHEL6 is called KVM (Kernel Virtual Machine). The Linux community has collectively rallied around development of KVM – primarily because its architectural approach is tightly integrated with the remainder of the Linux kernel. In fact, the KVM codebase is significantly smaller than alternative virtualization implementations because its design approach utilizes, rather than replaces, existing kernel functionality. A perfect example is the fact that KVM virtual machines are implemented as conventional Linux processes – which are then scheduled using the NUMA-aware Completely Fair Scheduler (CFS). As a result all of the optimization benefits of the CFS scheduler accrue directly to virtualized guests. Similarly, KVM-based virtualized guests immediately benefit from all of the bare-metal optimizations.

In addition to inheriting the bare-metal NUMA scalability optimizations, there are a number of KVM optimizations in Red Hat Enterprise Linux 6 specifically targeted at virtualization. The objective of these optimizations is to minimize the overhead of virtualization, with the goal being to approach the raw performance of a bare-metal system. A key focus area for such optimizations is the I/O path, primarily in storage and network access. These I/O optimizations enable intensive workloads such as database and file serving to be well suited for utilizing the flexibility of virtualized deployment. The following sections describe two of the key virtualization enhancements in RHEL6 tailored for NUMA hardware, CPU pinning and transparent hugepages.

3.2.1 CPU Pinning

RHEL6 permits binding virtualized guests to run on specific CPUs on the same socket. This fine-

grained control optimizes cache locality, as well as obviates inter-socket communication and remote memory access. These configuration interfaces are provided through the management software stack to allow easy and flexible allocation.

3.2.2 Transparent Hugepages

Transparent hugepages automatically performs memory allocation requests for large chunks of memory (via 2MB chunk size) in a NUMA aware manner. Allocating in large chunks dramatically reduces the software book-keeping operations needed within the virtual memory management layer. Virtual guest instances are represented in KVM using large memory structures that directly benefit from the transparent hugepages due to NUMA awareness and reduced lock contention. On virtual machines, the reduction in translation lookaside buffer (TLB) memory management operations yields a performance speedup of up to 20%.

4 Test Configuration

For comparative analysis of bare metal versus virtualized compute environments, a test cluster was constructed using two Dell PowerEdge C6100 systems each with four, dual socket nodes. The result was eight physical nodes or 96 total Westmere cores. The nodes were interconnected via a QDR Infiniband fabric using a Mellanox SX6036 FDR switch (reference Figure 1).

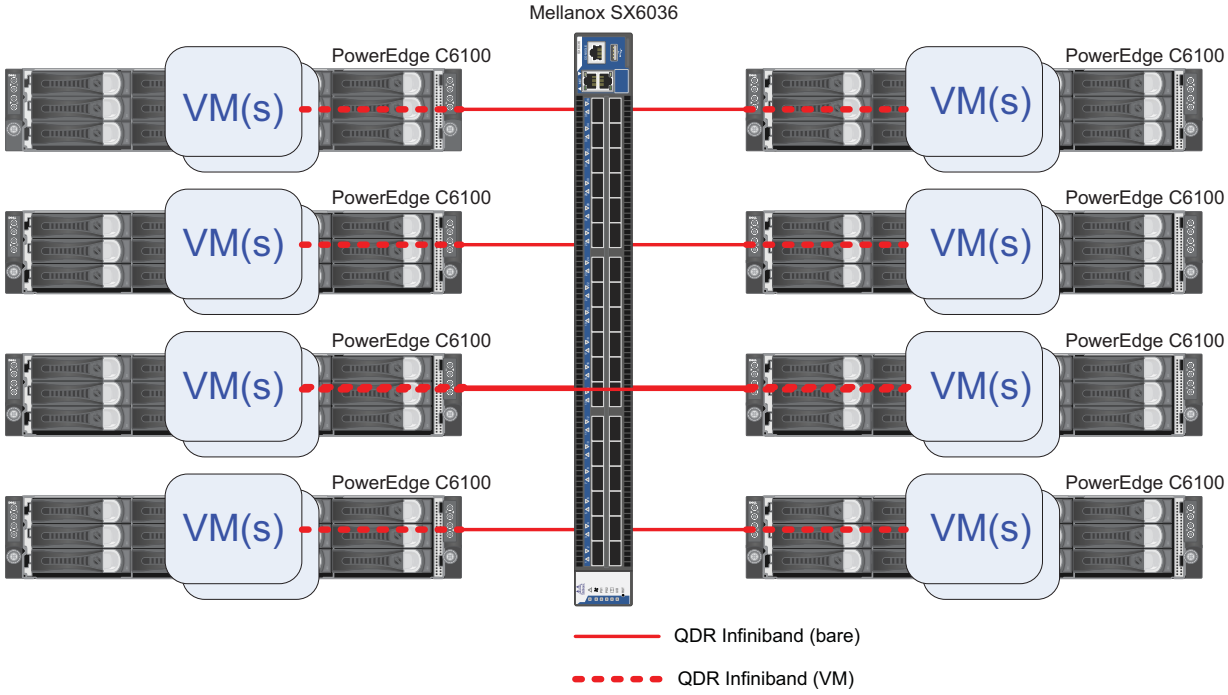

Figure 1 - Test cluster architecture

The following table provides hardware details of the C6100 nodes.

Configuration	Host (Bare Metal)
Node	Dell PowerEdge C6100
Processor Type	Westmere
Processor Number	X5660

Processor Speed	2.8GHz
Sockets	2
Cores per Socket	6
Memory	24GB
Infiniband HCA	Mellanox MT26428 QDR

The following sections describe basic configuration details of the hosts and the VMs. For reference, the hosts were numbered rh64-1 through rh64-8 and the VMs rh64-1-vm1 through rh64-8-vm1. All eight hosts were setup identically with the exception of rh64-1 which as will be noted later provided shared directories to the other seven nodes. Eight identical VMs were built, one VM per host, which when active consumed essentially all the physical resources of the corresponding hosts.

4.1 Hypervisor/Host Setup

1. Enabled SR-IOV in BIOS

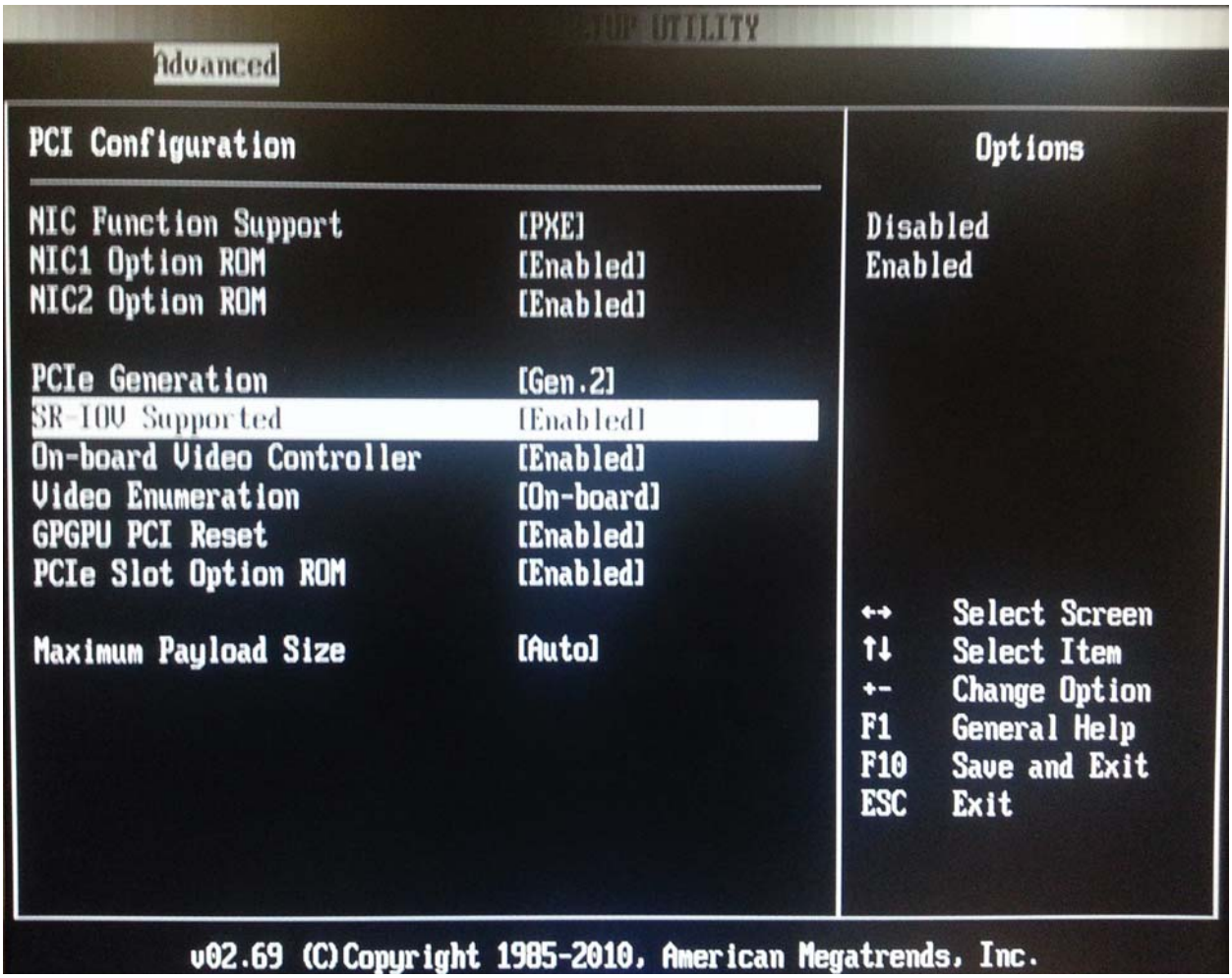

Figure 2 - SR-IOV Enable

2. Enabled Intel Virtualization Technology in BIOS

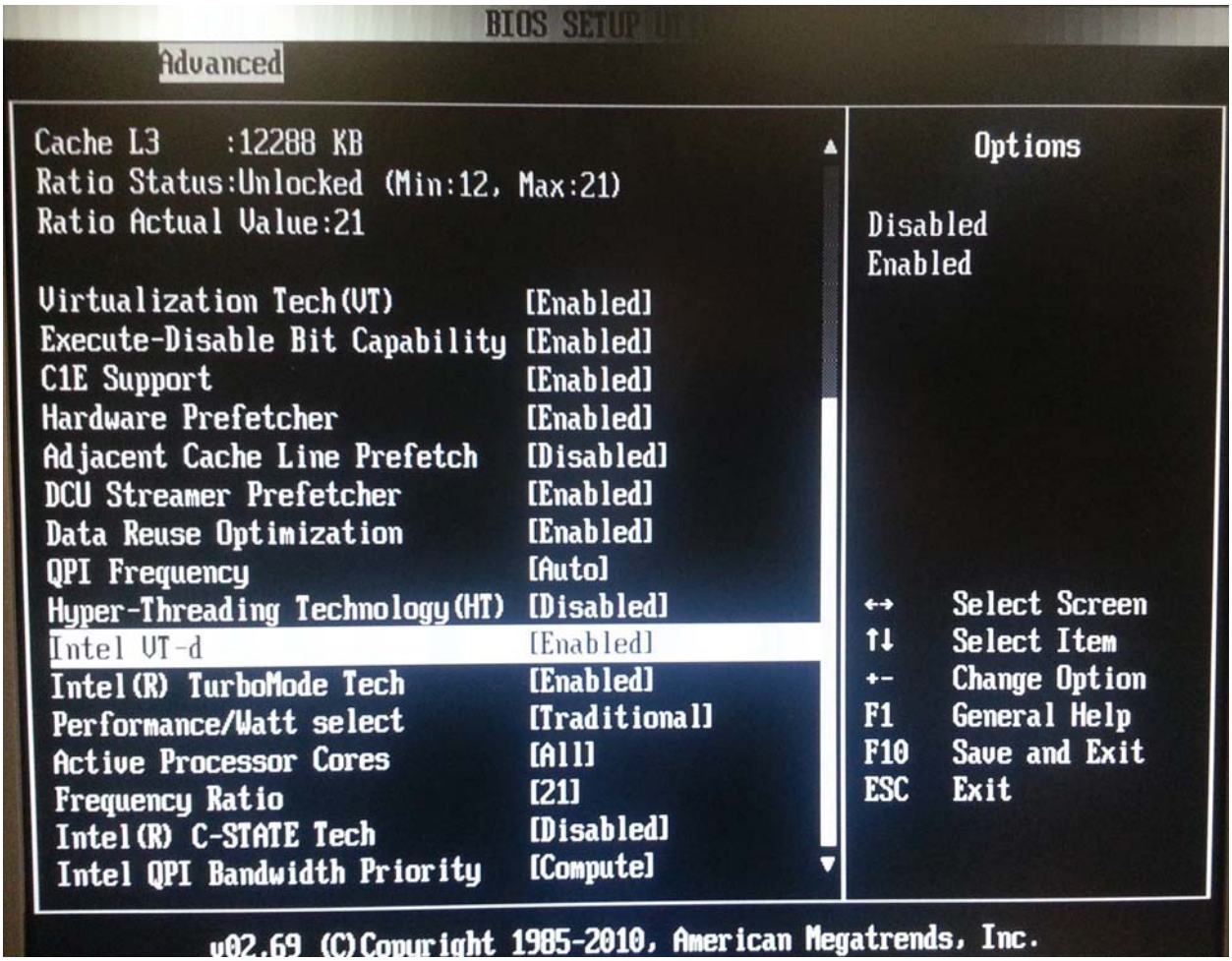

Figure 3 - Intel VT-d enable

3. Installed RHEL6.4 operating system.
4. Disabled firewalls.
5. Installed Infiniband support. Note that distribution provided Infiniband modules support SR-IOV natively. Configured Infiniband interfaces for ip-over-ib and set all hosts to 10.0.0.x subnet.
6. Set memory limits in /etc/security/limits.conf

```
*       soft    memlock unlimited
*       hard    memlock unlimited
```

7. Enabled Intel Input/Output Memory Management Unit by adding the following to the end of the active kernel line in /boot/grub/menu.lst

```
kernel /vmlinuz-2.6.32-358.2.1.el6.x86_64 ro …… intel_iommu=on
```

8. Modified Mellanox HCA firmware by adding the following to the .ini file and updating the firmware to version 2.10.2000

```
[HCA] num_pfs = 1
total_vfs = 8
sriov_en = true
```

9. Added /etc/modprobe.d mlx4_core.conf with following contents

```
options mlx4_core num_vfs=8
```

At this point the node after rebooting showed the following in the *lspci* output indicating availability of Infiniband virtual functions:

```
05:00.0 InfiniBand: Mellanox Technologies MT26428 [ConnectX VPI PCIe 2.0 5GT/s - IB QDR / 10GigE]
(rev b0)
05:00.1 InfiniBand: Mellanox Technologies MT25400 Family [ConnectX-2 Virtual Function] (rev b0)
05:00.2 InfiniBand: Mellanox Technologies MT25400 Family [ConnectX-2 Virtual Function] (rev b0)
05:00.3 InfiniBand: Mellanox Technologies MT25400 Family [ConnectX-2 Virtual Function] (rev b0)
05:00.4 InfiniBand: Mellanox Technologies MT25400 Family [ConnectX-2 Virtual Function] (rev b0)
05:00.5 InfiniBand: Mellanox Technologies MT25400 Family [ConnectX-2 Virtual Function] (rev b0)
05:00.6 InfiniBand: Mellanox Technologies MT25400 Family [ConnectX-2 Virtual Function] (rev b0)
05:00.7 InfiniBand: Mellanox Technologies MT25400 Family [ConnectX-2 Virtual Function] (rev b0)
05:01.0 InfiniBand: Mellanox Technologies MT25400 Family [ConnectX-2 Virtual Function] (rev b0)
```

10. Added /etc/modprobe.d kvm_iommu_map_guest.conf with following contents:

```
options kvm allow_unsafe_assigned_interrupts=1
```

11. Created the following directories on rh64-1 and shared across remaining hosts rh64-2 through rh64-8:

```
/home
/usr/local/other
```

12. Installed Intel compilers and MVAPICH2 into:

```
/usr/local/other/utilities
```

13. Setup passwordless access across rh64-1 through rh64-8
14. Configured /etc/hosts on rh64-1 through rh64-8

15. For VM testing, mounted hugepages on each host. Note that mounting hugepages should done when VMs are stopped and it requires a restart of /etc/init.d/libvirtd. Note hugepages was always unmounted during host level testing.

```
huge.sh
# as root, note: value of hugepages is in 2 MB regions ... 11000 == 22GB
# setup_huge.sh
umount /dev/hugepages
echo 0 > /proc/sys/vm/nr_hugepages
cat /proc/meminfo | grep Huge
echo 3 > /proc/sys/vm/drop_caches
sync
echo 11000 > /proc/sys/vm/nr_hugepages
cat /proc/meminfo | grep Huge
mount -t hugetlbfs hugetlbfs /dev/hugepages
```

The following table summarizes the host software configuration:

Host Software	Version
Operating System	Red Hat 6.4 (Santiago)
Kernel	2.6.32-358.2.1
Intel C Compiler (icc)	13.1.1
Intel Fortran Compiler (ifort)	13.1.1
MVAPICH2	2-1.8.1

4.1.1 Virtual Machine/Guest

Using virt-manager, guests were built on all the hosts per the following highlights meaning this is not a step-by-step set of instructions but rather a guide:

1. Launched virt-manager on each of the hosts rh64-1 through rh64-8 and built VMs using RHEL6.4 iso.

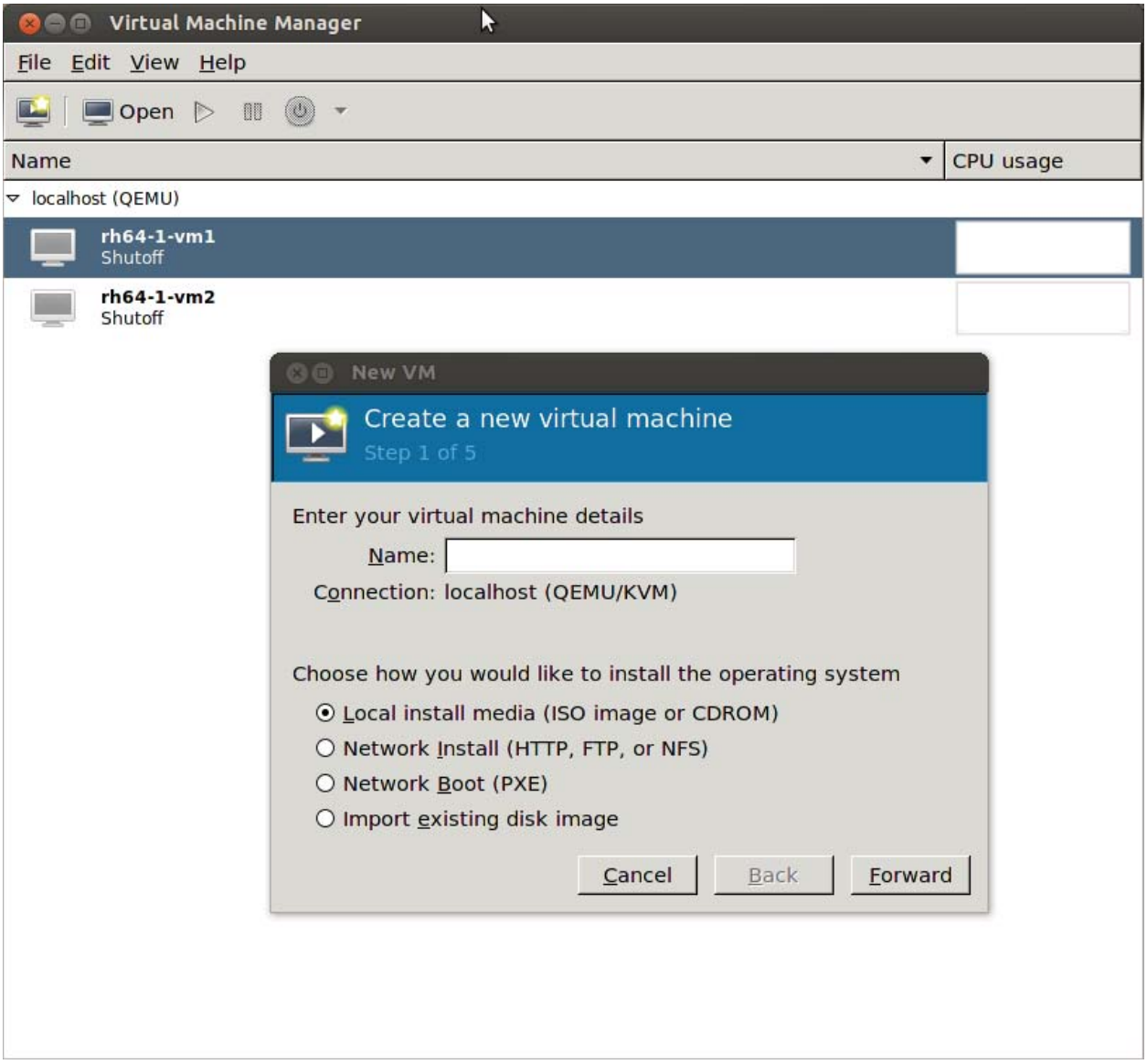

Figure 4 - Virt-Manager VM build screen

Loaded operating system mirrored that of the host.

Item	Version
Operating System	Red Hat 6.4 (Santiago)
Kernel	2.6.32-358.2.1

2. Disabled firewalls.
3. Once rebooted, loaded Infiniband support modules. Configured Infiniband interfaces for ip-over-ib and set all VMs to 10.0.0.x subnet.
4. Set memory limits in /etc/security/limits.conf

```
*       soft    memlock unlimited
*       hard    memlock unlimited
```

5. Configured shared directories /home and /usr/local/other and /etc/hosts to operate like their host counterparts.
6. Set number of processor to 12 to match host available CPUs/cores and used “Copy host CPU configuration” to mimic Westmere functionality within the VMs.

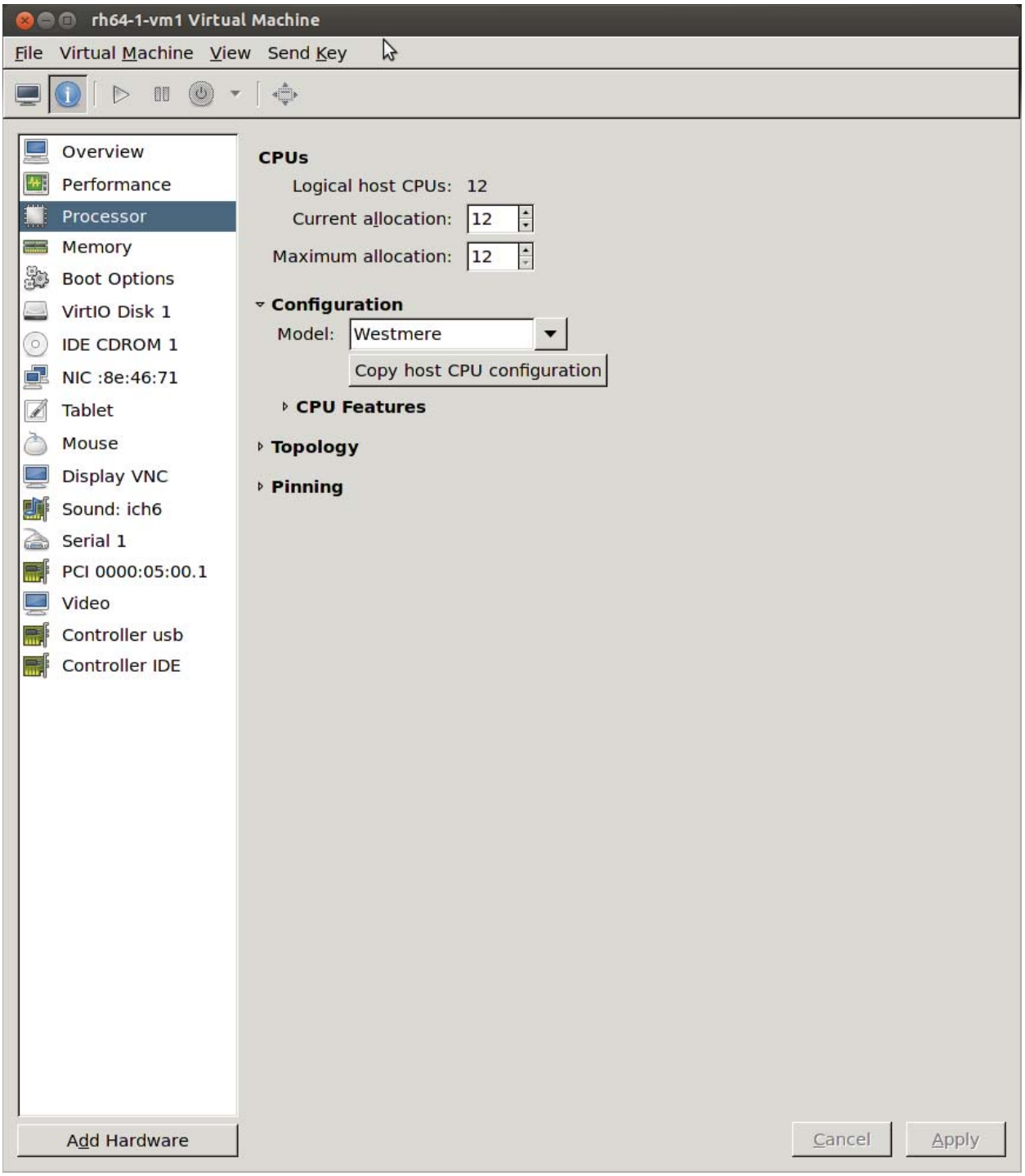

Figure 5 - Virt-manager CPU configuration

7. Allocated 20GB (out of 24GB) of physical memory to VMs.

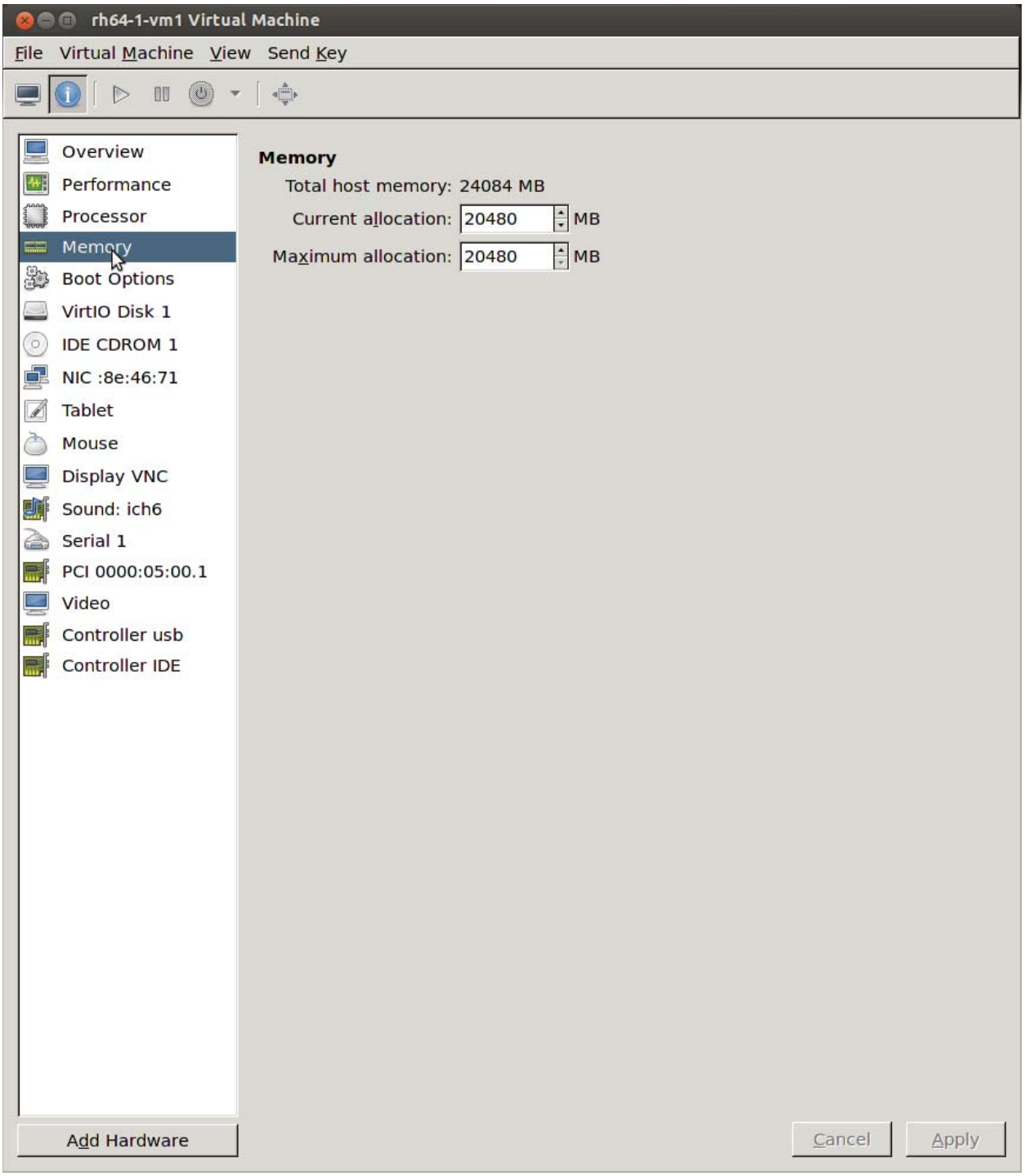

Figure 6 - Virt-manager memory configuration

8. Mapped one of the Infiniband VFs to the VMs.

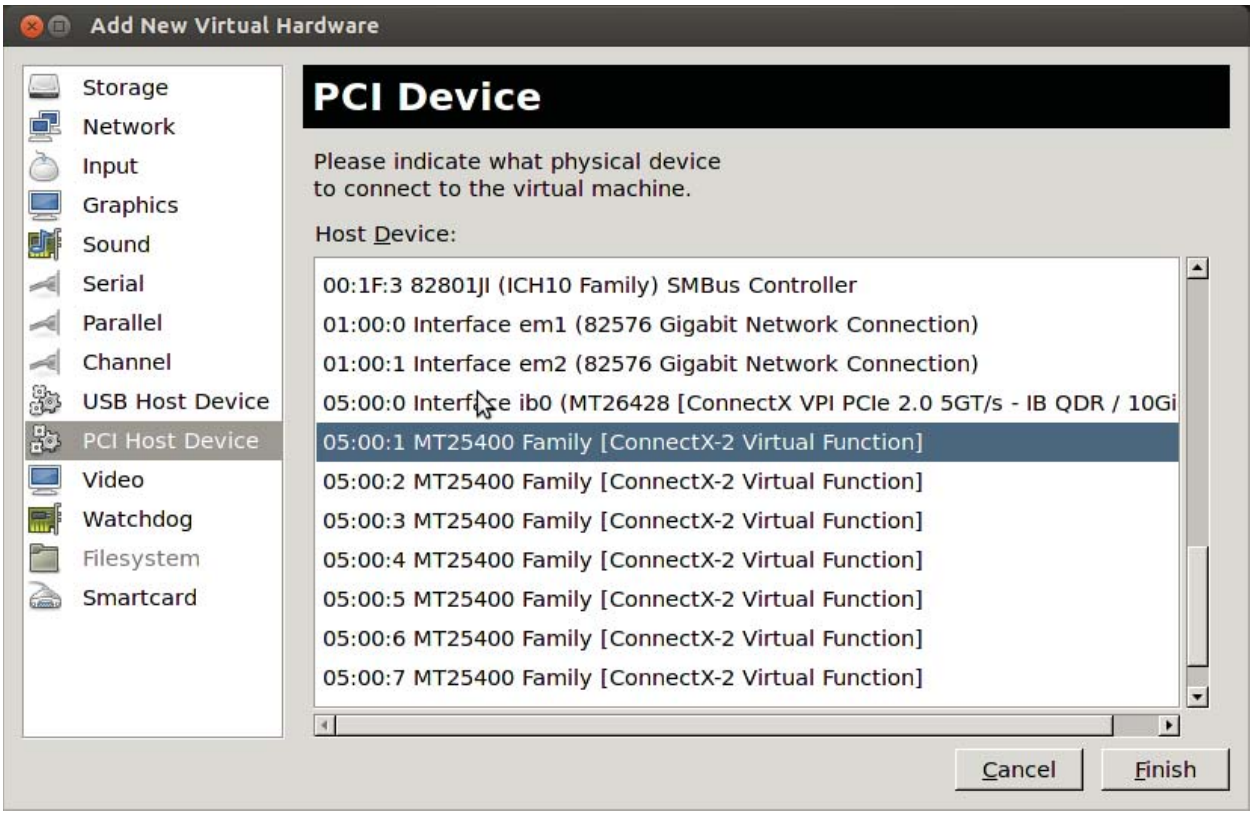

Figure 7 - Virt-manager PCI Host Device mapping

9. Modified the VM xml file using virsh edit <vm name> to use hugespages.

```
<memoryBacking>
    <hugepages/>
  </memoryBacking>
```

10. Modified the VM xml file using virsh edit <vm name> to pin virtual CPUs to physical CPUs.

```
  <vcpu placement='static'>12</vcpu>
  <cputune>
    <vcpupin vcpu='0' cpuset='0'/>
    <vcpupin vcpu='1' cpuset='1'/>
    <vcpupin vcpu='2' cpuset='2'/>
    <vcpupin vcpu='3' cpuset='3'/>
    <vcpupin vcpu='4' cpuset='4'/>
    <vcpupin vcpu='5' cpuset='5'/>
    <vcpupin vcpu='6' cpuset='6'/>
    <vcpupin vcpu='7' cpuset='7'/>
    <vcpupin vcpu='8' cpuset='8'/>
    <vcpupin vcpu='9' cpuset='9'/>
    <vcpupin vcpu='10' cpuset='10'/>
    <vcpupin vcpu='11' cpuset='11'/>
  </cputune>
```

11. Modified the VM xml file using virsh edit <vm name> to set NUMA memory parameters.

```
<numa>
      <cell cpus='0-5' memory='10485760'/>
      <cell cpus='6-11' memory='10485760'/>
    </numa>
```

5 Benchmarks

Multiple benchmarks were performed on the cluster contrasting host level performance versus a cluster constructed using one VM per host. Note that during host level tests, VM were shutdown. When VMs were up, they were fully tuned with hugepages, CPU pinning and NUMA memory changes in place in the VM xml definition files. The following table captures the benchmarks that were run and their version numbers.

Item	Version
Stream	5.10
OSU Micro-benchmarks	2-1.8.1
Linpack	11.0.3
NAS PB	3.3.1

5.1 Stream

The STREAM benchmark is a simple synthetic benchmark program that measures sustainable memory bandwidth (in MB/s) and the corresponding computation rate for simple vector kernels [4]. The following graph compares host memory performance to stock or unturned VM memory performance and lastly to tuned VM memory performance. Tuned infers that hugepages, CPU pinning and NUMA memory control were all in place. A threaded version of the benchmark was compiled with the Intel compiler and was executed on a single node – Host and VM independently – by increasing the number of threads starting at one going to twelve, the total core count in a given node. With tuning in place, VM memory bandwidth surpassed that of the Host likely due to more NUMA localization.

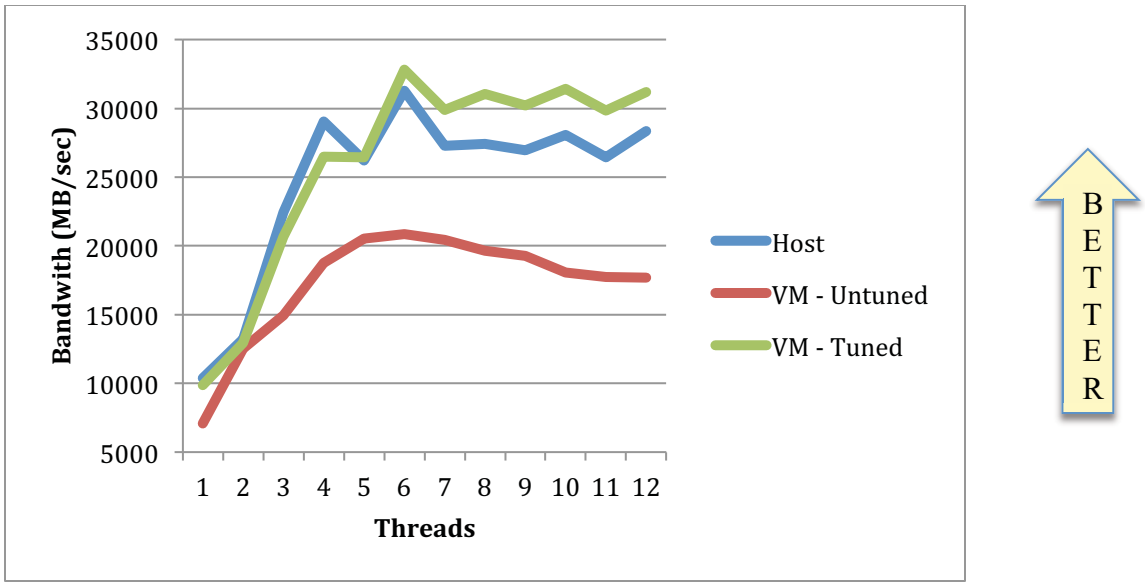

Figure 8 - Stream memory bandwidth benchmark

5.2 OSU Micro-Benchmarks

The OSU Micro-Benchmarks were compiled as part of the MVAPICH2 package using the Intel compiler. Bandwidth and latency of Host-to-Host Infiniband connections were compared to VM-to-VM Infiniband connections.

MVAPICH2 configured and installed per the following:

```
./configure prefix=/usr/local/other/utilities/mvapich2
```

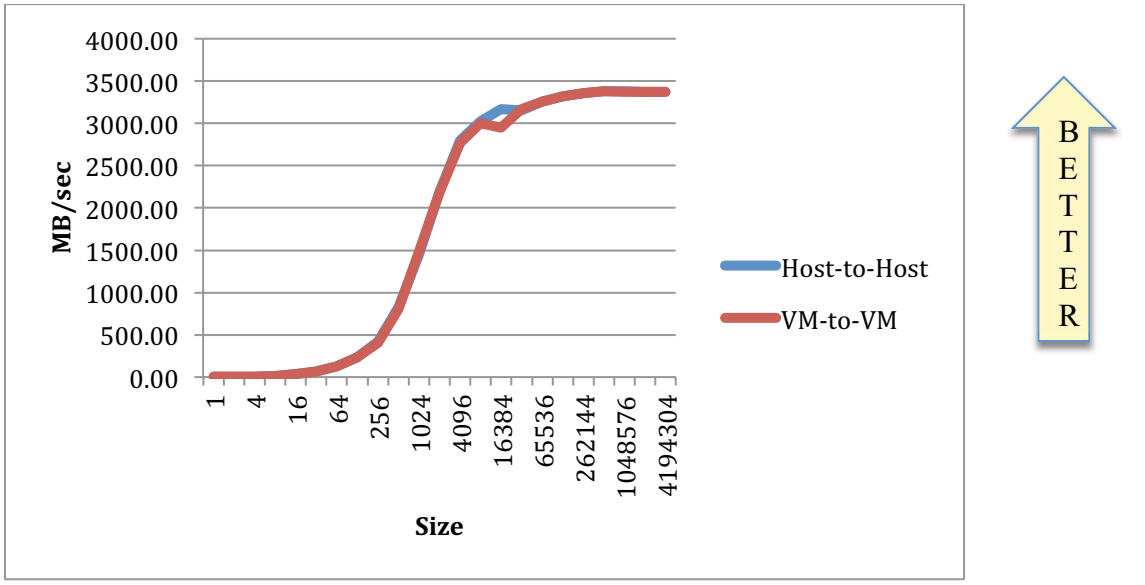

Figure 9 - OSU Micro-benchmark bandwidth

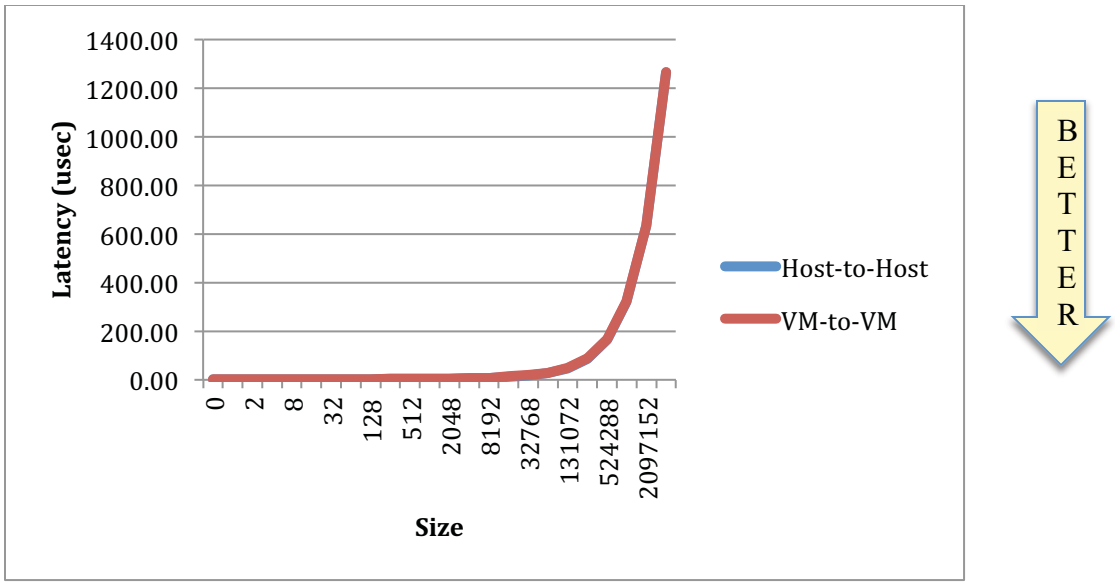

Figure 10 - OSU Micro-benchmark latency

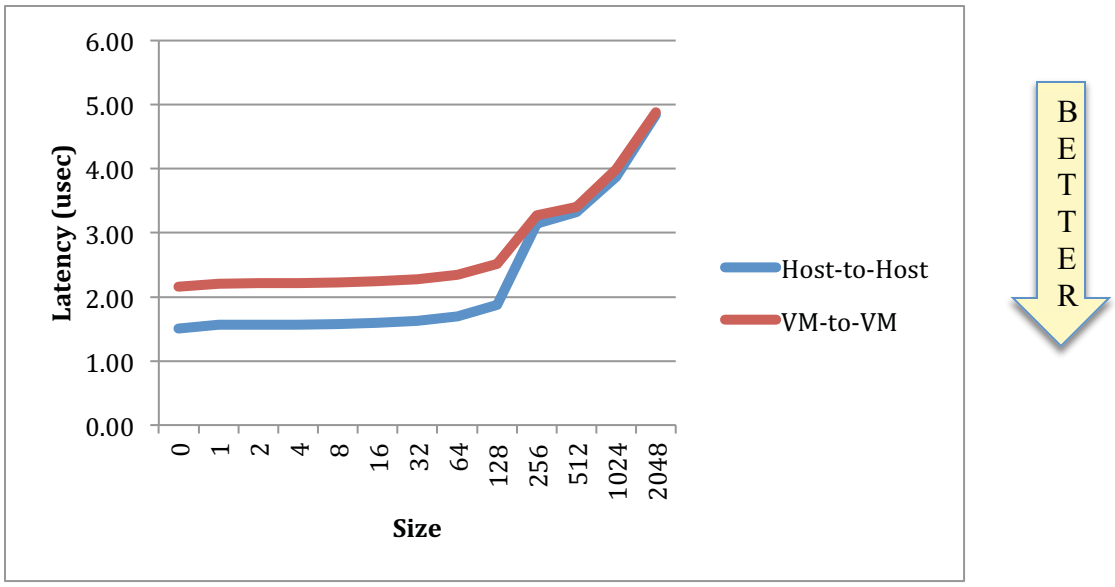

Figure 11 - OSU Micro-benchmark latency - small sizes

VM-to-VM bandwidth and latency essentially mimicked that of Host-to-Host with the minor exception of small transfer size latency. At below 256 bytes there was roughly a 40% increase in latency of the VM-to-VM numbers which theoretically could impact high message count, small packet size environments.

5.3 LINPACK

LINPACK is a collection of Fortran subroutines that analyze and solve linear equations and linear least-squares problems. The package solves linear systems whose matrices are general, banded, symmetric indefinite, symmetric positive definite, triangular, and tridiagonal square. In addition, the package computes the QR and singular value decompositions of rectangular matrices and applies them to least-squares problems. LINPACK uses column-oriented

algorithms to increase efficiency by preserving locality of reference [6]. The precompiled version of the Intel Math Kernel Library [7] was used.

MVAPICH2 configured per the following for the mpi versions of the tests:

```
./configure --with-rdma=gen2 --enable-fast --with-pm=mpd,hydra --
prefix=/usr/local/other/utilities/mvapich2
```

5.3.1 Single Node

The single node tests involved running the openmp version of LINPACK in contrast to the hybrid mpi-threaded version with sizes ranging from N=1000 to N=45000. In the case of the hybrid version (launched using mvapich2), the block size (NB) was set to 192 and the products of the P/Q values were set to equal the mpi count.

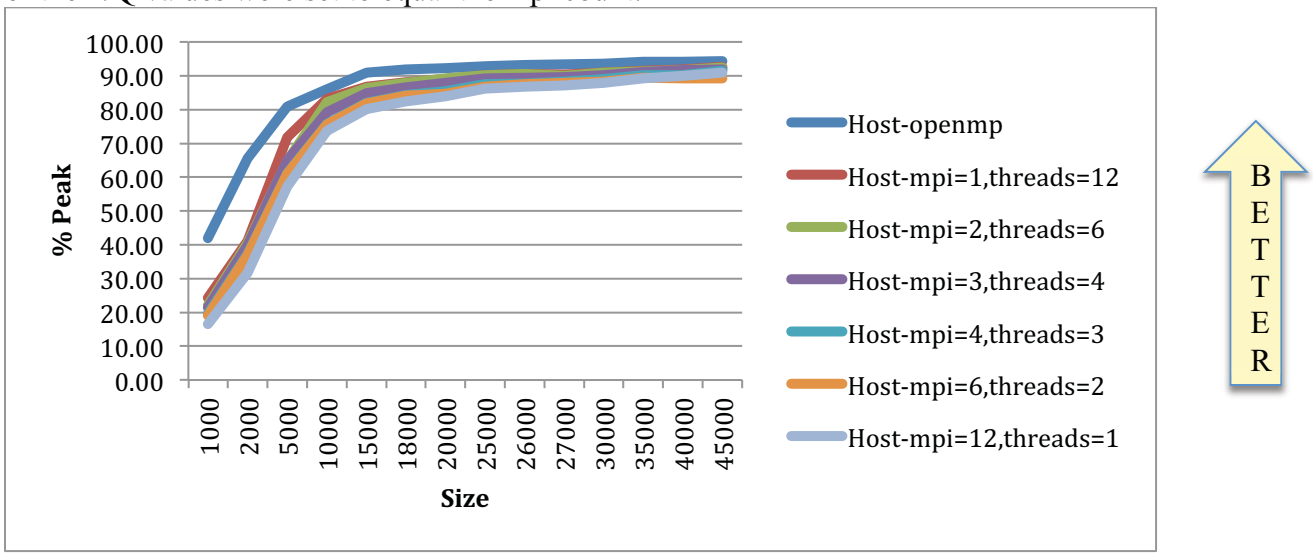

Figure 12 - LINPACK single node Host

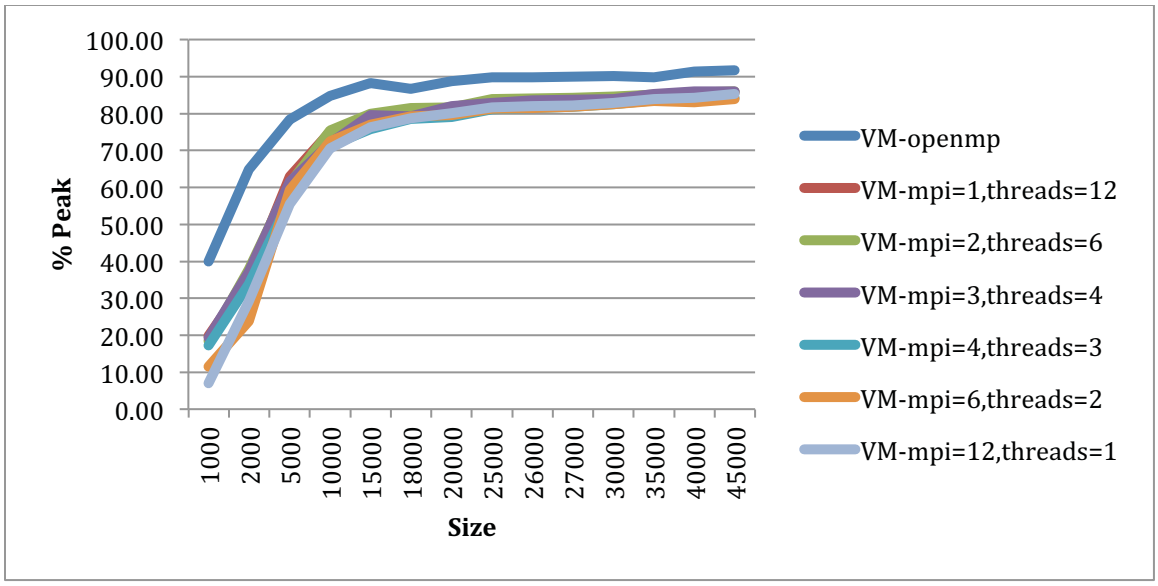

Figure 13 - LINPACK single node VM

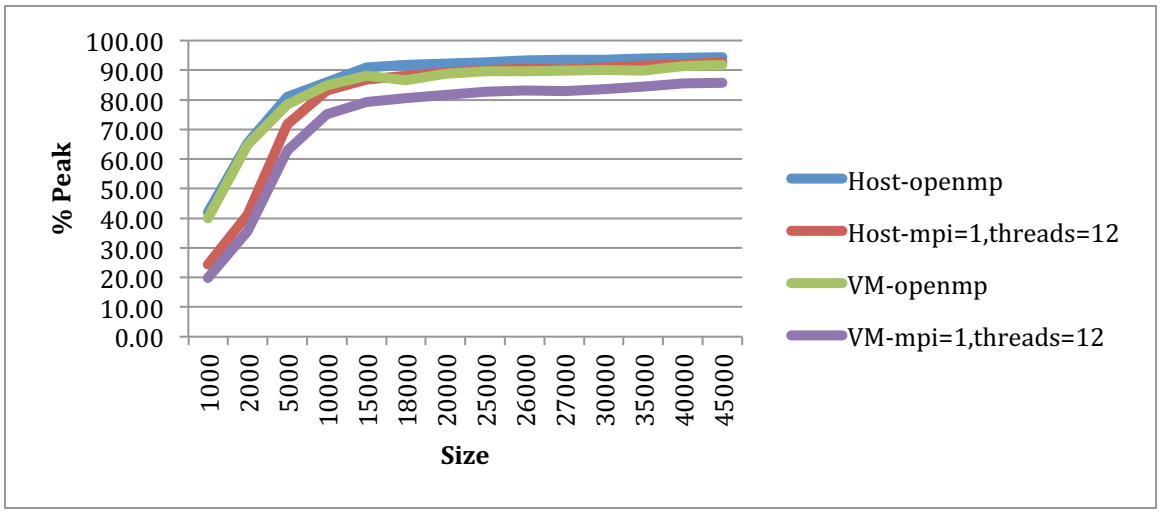

Figure 14 - LINPACK single node Host versus VM

The benchmark output values were compared against a calculated peak value for a dual-socket Westmere system of 134.4 GFlops (12 cores X 4 results per clock X 2.8 GHz). Overall the VM performed well in comparison to the Host with the only significant degradation being the hybrid (mpi/thread) runs that showed between 6% to 9% degradation over their openmp counterpart.

5.3.2 Multi-node

The multi-node tests were executed in a fashion similar to the single node but only involved the hybrid-mpi version of LINPACK. Multi-node means that an eight node cluster was instantiated comprised either of real Hosts or VMs (one per Host). Sizes ranged from N=100,000 to N=150,000 for the real Host based cluster and N=100,000 to N=140,000 for the VM based cluster. The size was reduced in the case of the VM cluster due to earlier exhaustion of available memory (20GB versus 24GB). The block size (NB) was again set to 192 and the products of the P/Q values were adjusted to equal the total mpi count across the eight nodes, real or VMs. The host file utilized knowledge of the cluster size to spread mpi jobs across the nodes evenly as

opposed to loading up the first node before proceeding to the second (rh64-1, rh64-2, etc. as opposed to rh64-1, rh64-1, etc.)

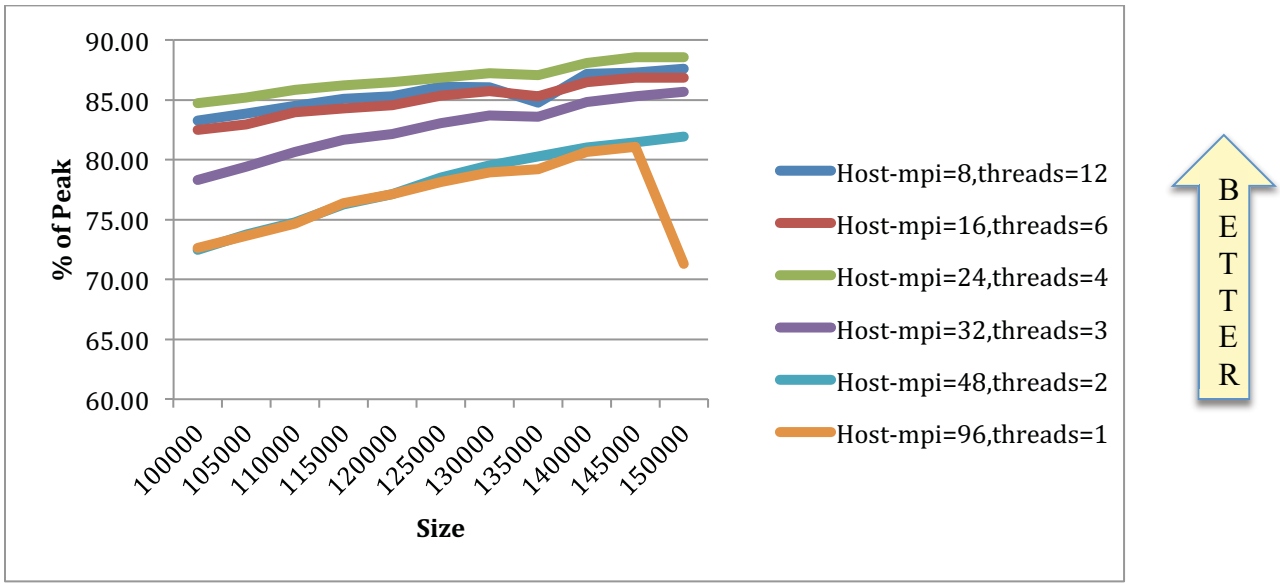

Figure 15 - LINPACK multi-node Host

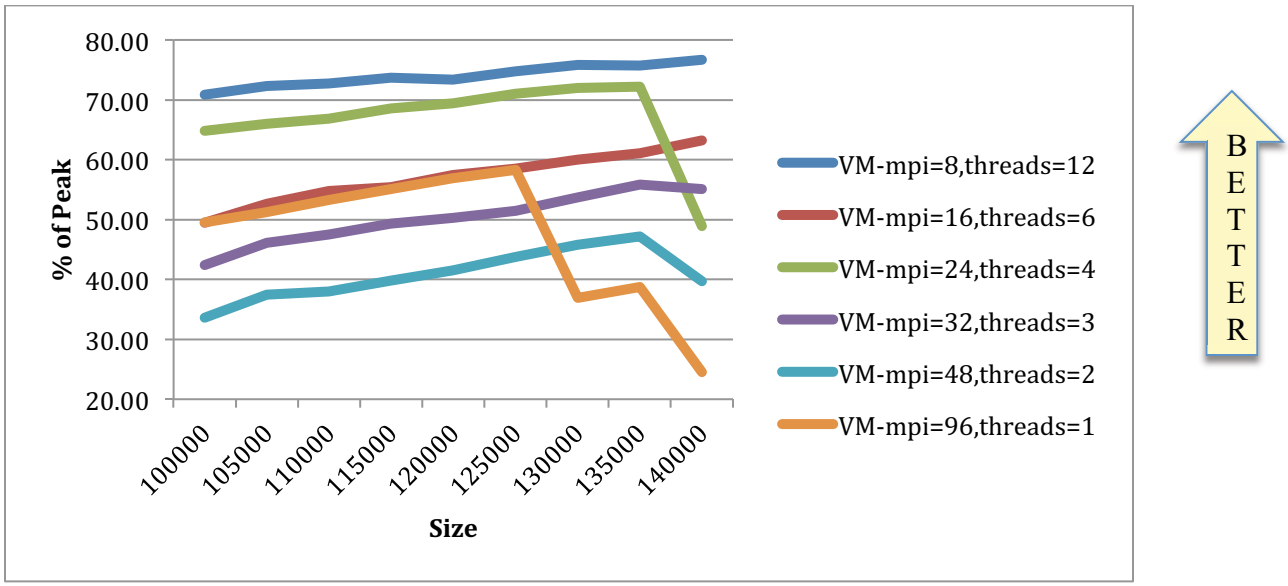

Figure 16 - LINPACK multi-node VM

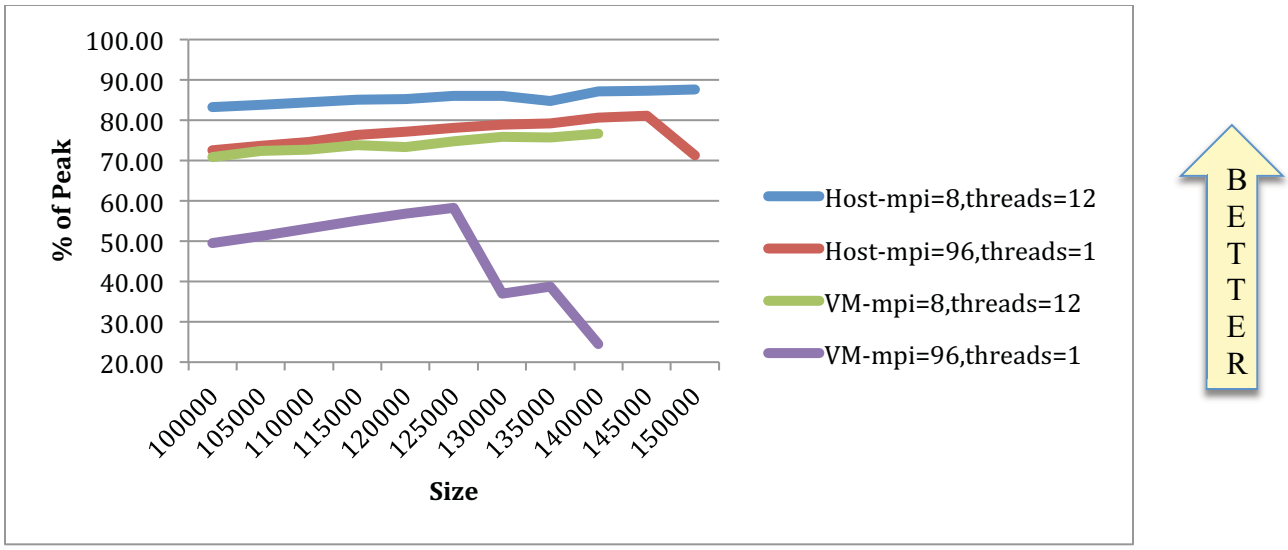

Figure 17 - LINPACK multi-node Host versus VM

The benchmark output values were again compared against a calculated peak value for an eight node, dual-socket Westmere cluster or 8 X 134.4 GFlops or 1075.2 GFlops. One initial observation is that larger mpi counts had a more significant impact on the VM cluster performance as opposed to the real Host cluster. The VM cluster also showed more evidence of running out memory on the larger problem sizes. However, comparing results for the 8 mpi, 12 thread case showed the VM cluster only lagging the real host cluster by about 12%.

5.4 NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPB) are a small set of programs designed to help evaluate the performance of parallel supercomputers. The benchmarks are derived from computational fluid dynamics (CFD) applications and consist of five kernels and three pseudo-applications in the original "pencil-and-paper" specification (NPB 1). Problem sizes in NPB are predefined and indicated as different classes. Reference implementations of NPB are available in commonly-used programming models like MPI and OpenMP (NPB 2 and NPB 3).

The original eight benchmarks specified in NPB 1 mimic the computation and data movement in CFD applications:

- Five kernels
  - IS - Integer Sort, random memory access
  - EP - Embarrassingly Parallel
  - CG - Conjugate Gradient, irregular memory access and communication
  - MG - Multi-Grid on a sequence of meshes, long- and short-distance communication, memory intensive
  - FT - discrete 3D fast Fourier Transform, all-to-all communication
- Three pseudo applications
  - BT - Block Tri-diagonal solver
  - SP - Scalar Penta-diagonal solver
  - LU – Lower-Upper Gauss-Seidel solver [8]

The NPB were compiled for class sizes A, B, C and D but only classes C and D are presented. Class equates to problem size and for this size cluster (eight nodes), classes A and B represent too small a workload for the results to be meaningful. As with the LINPACK tests, the NPB were run on the real Host eight node cluster and an eight node VM cluster. The host file was again round robin spreading the processes evenly across the nodes.

5.4.1 Integer Sort (IS)

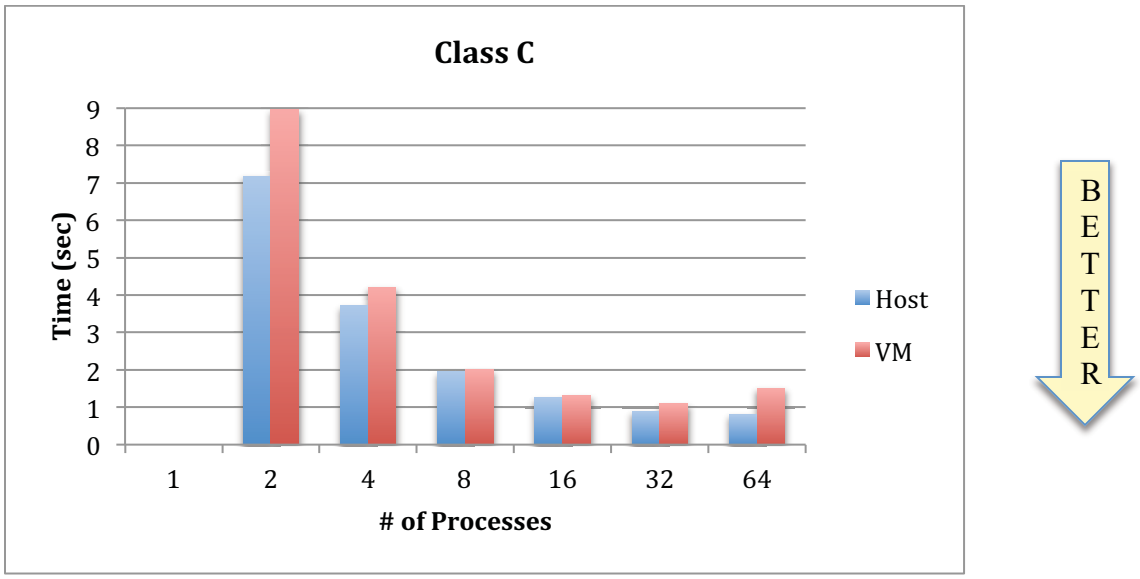

Figure 18 - IS Class C

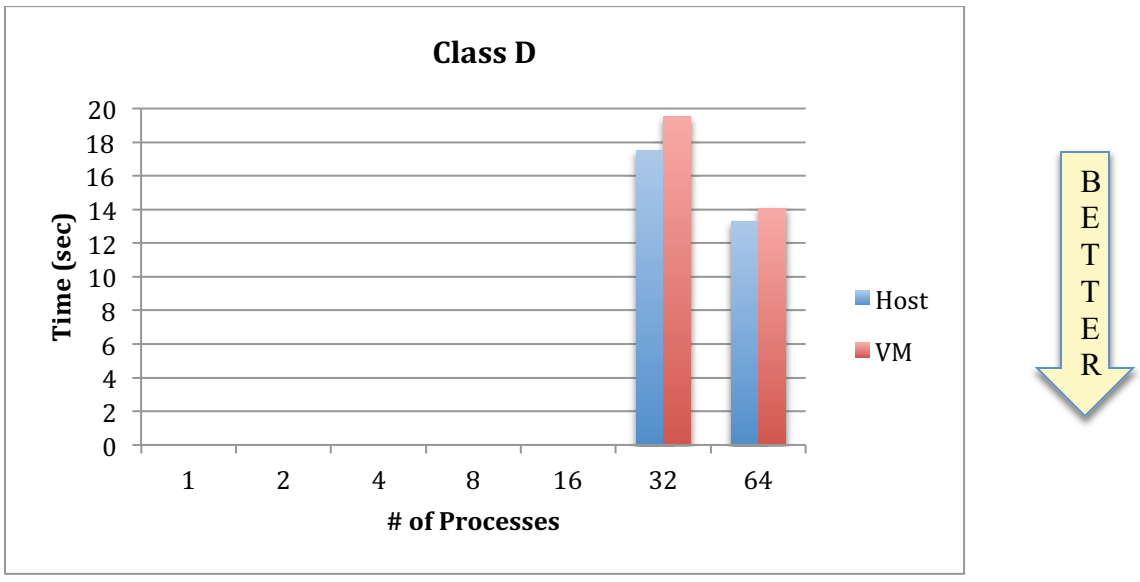

Figure 19 - IS Class D

For the Class D, 64 process case the VM performance was approximately 6% slower in time than the real Host based cluster.

5.4.2 Embarrassingly Parallel (EP)

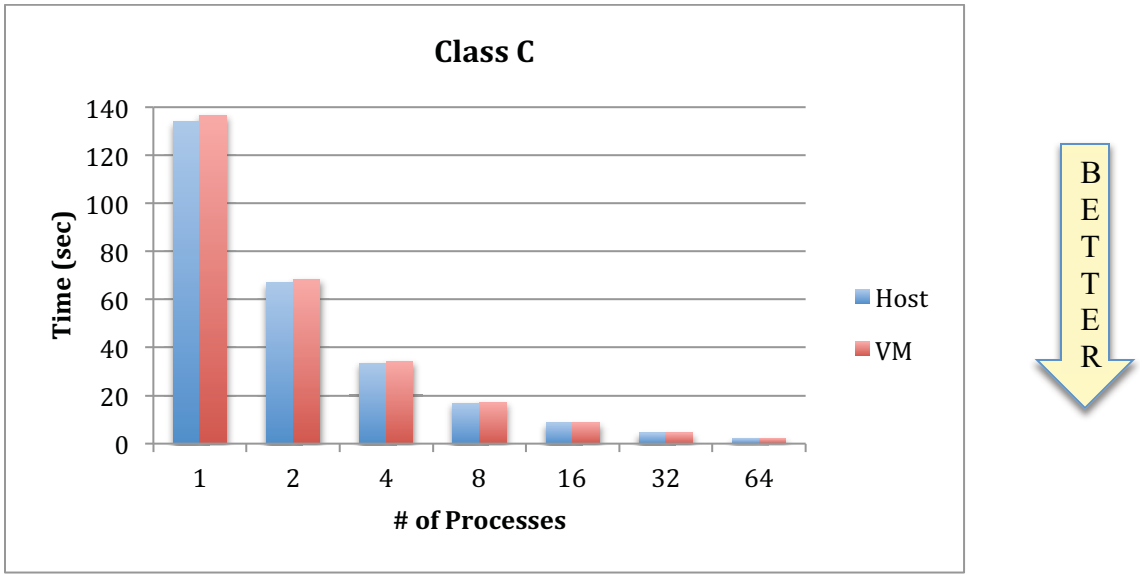

Figure 20 - EP Class C

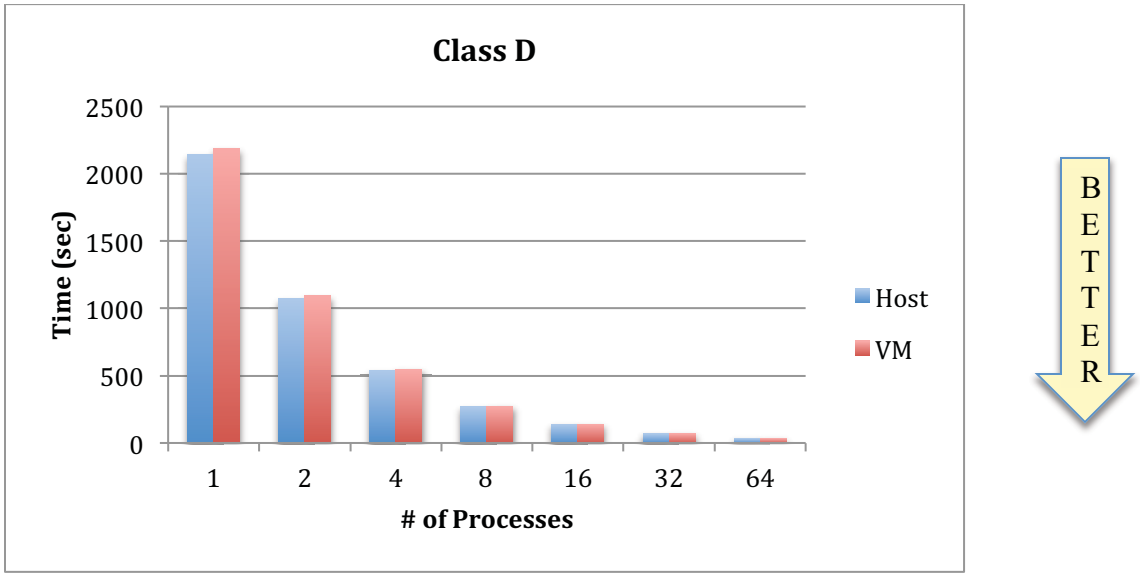

Figure 21 - EP Class D

For the Class D, 64 process case the VM performance was approximately 2% slower in time than the real Host based cluster.

5.4.3 Conjugate Gradient (CG)

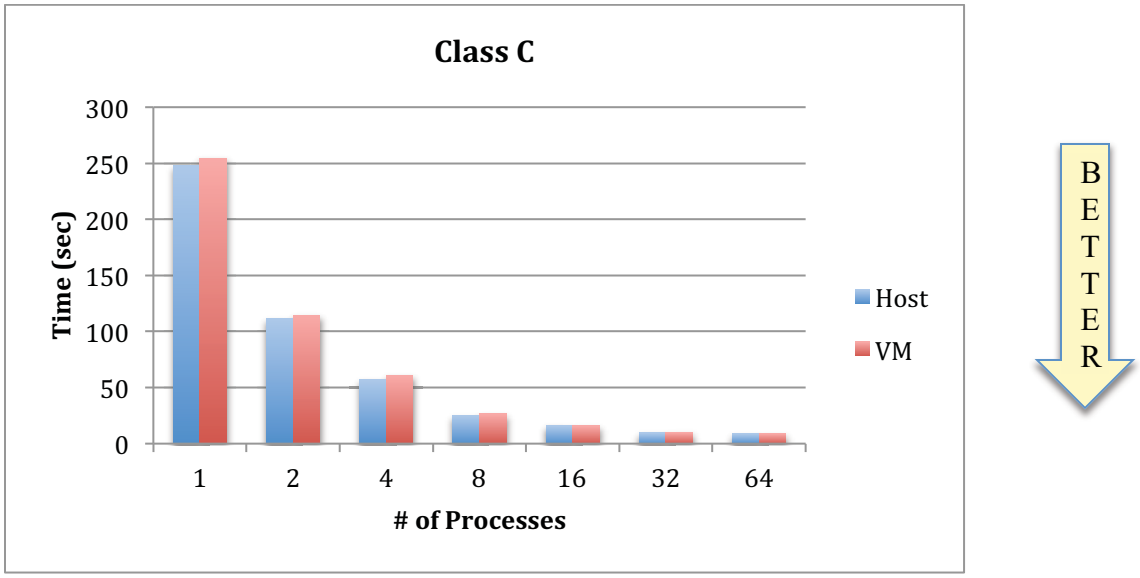

Figure 22 - CG Class C

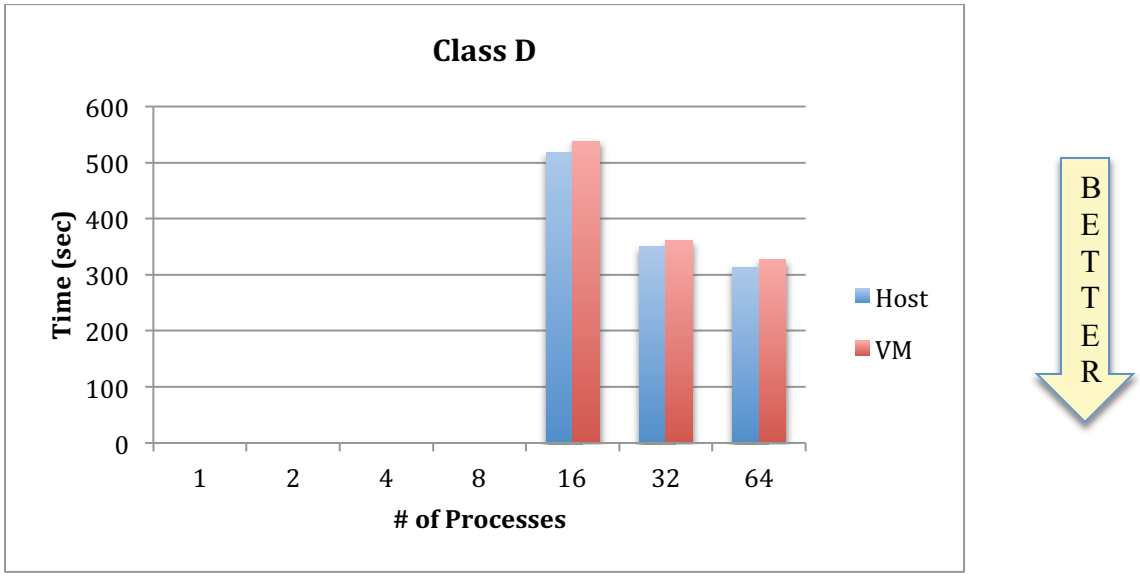

Figure 23 - CG Class D

For the Class D, 64 process case the VM performance was approximately 6% slower in time than the real Host based cluster.

5.4.4 Multi-Grid

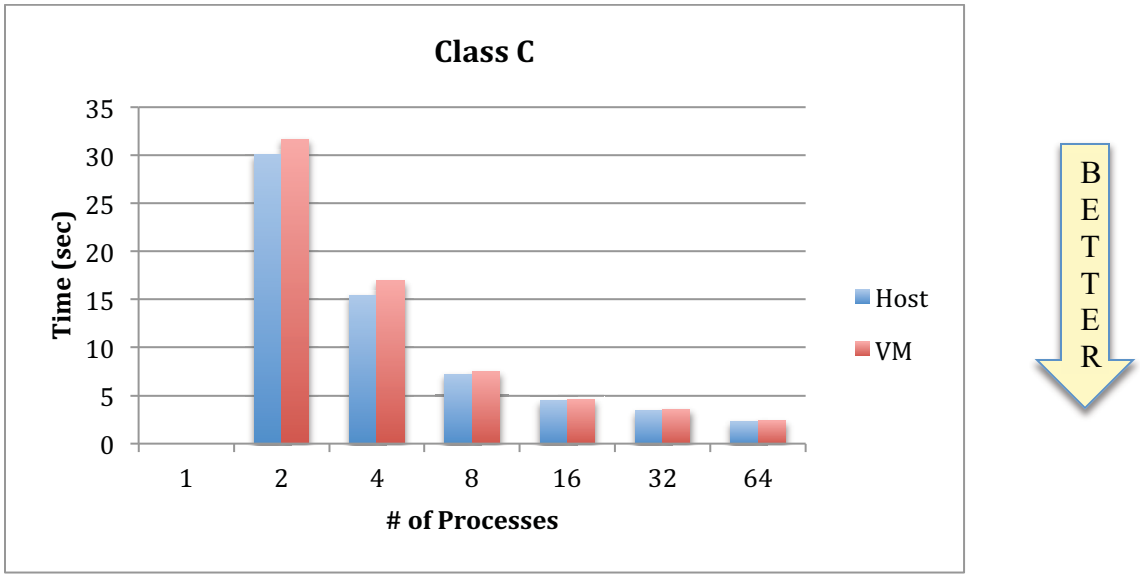

Figure 24 - MG Class C

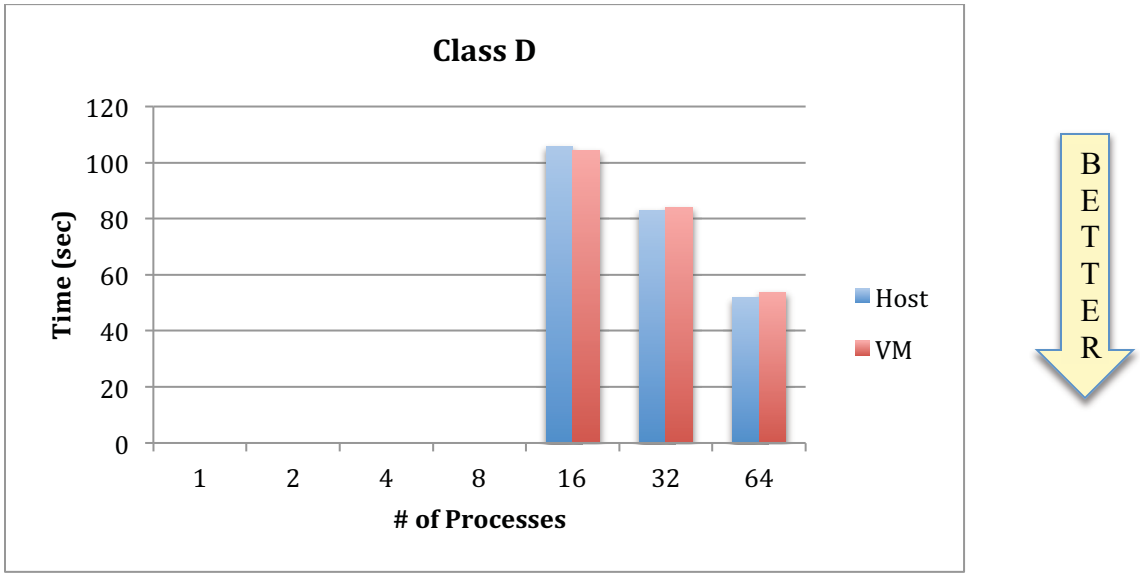

Figure 25 - MG Class D

For the Class D, 64 process case the VM performance was approximately 4% slower in time than the real Host based cluster.

5.4.5 Fourier Transform (FT)

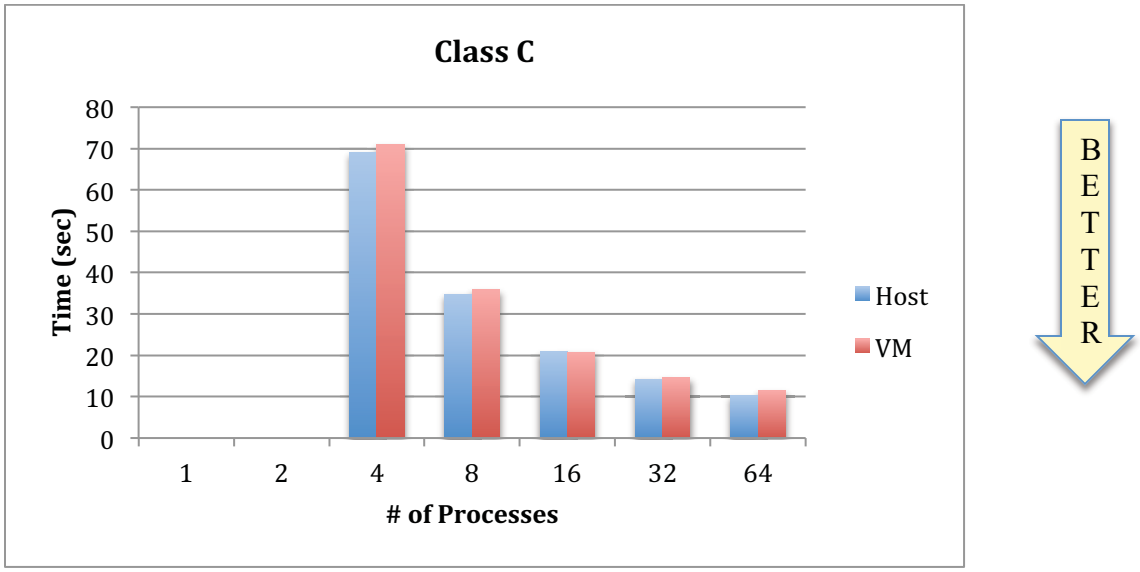

Figure 26 - FT Class C

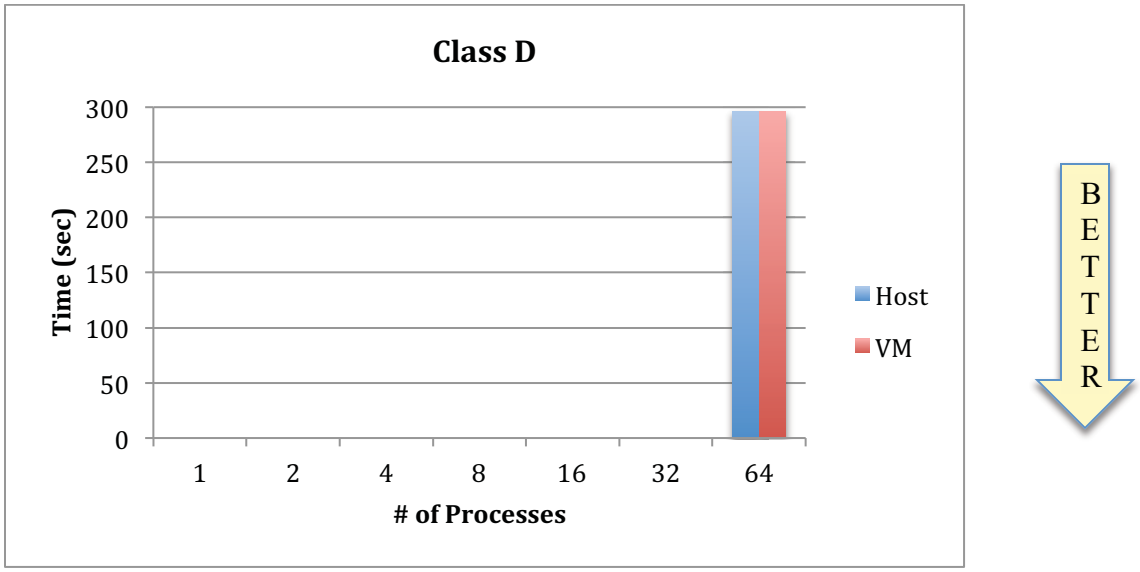

Figure 27 - FT Class D

For the Class D, 64 process case the VM performance and the real Host were essentially equal.

5.4.6 Block Tri-diagonal solver (BT)

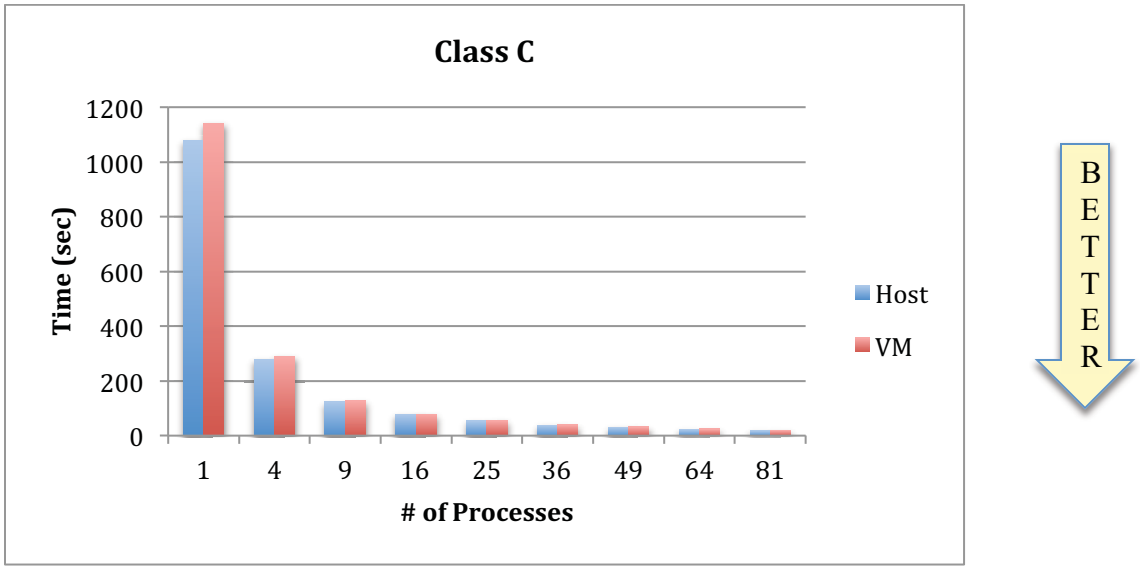

Figure 28 - BT Class C

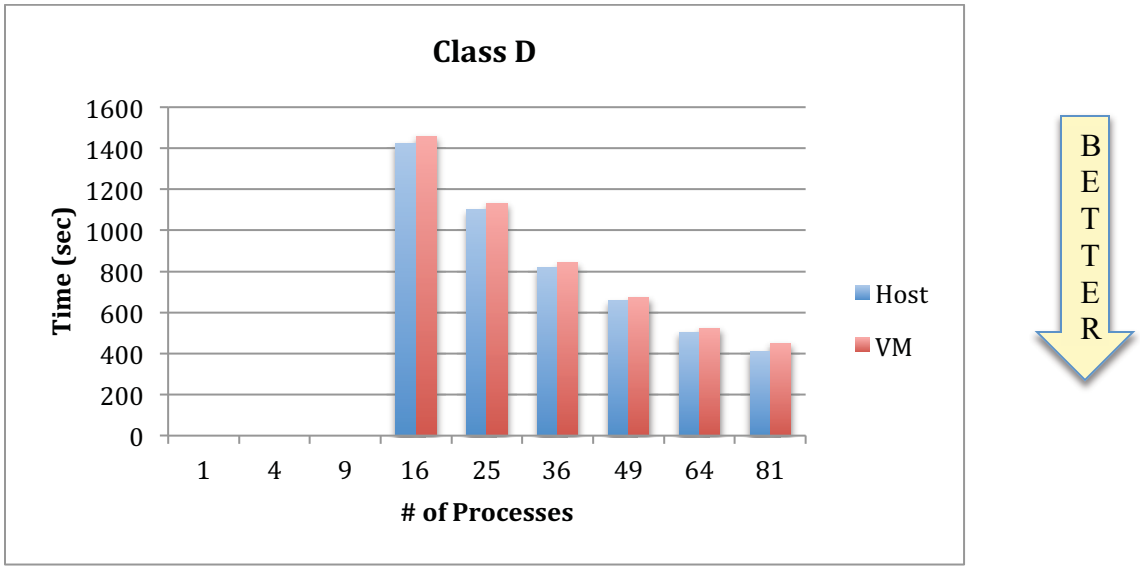

Figure 29 - BT Class D

For the Class D, 81 process case the VM performance was approximately 10% slower in time than the real Host based cluster.

5.4.7 Scalar Penta-diagonal solver (SP)

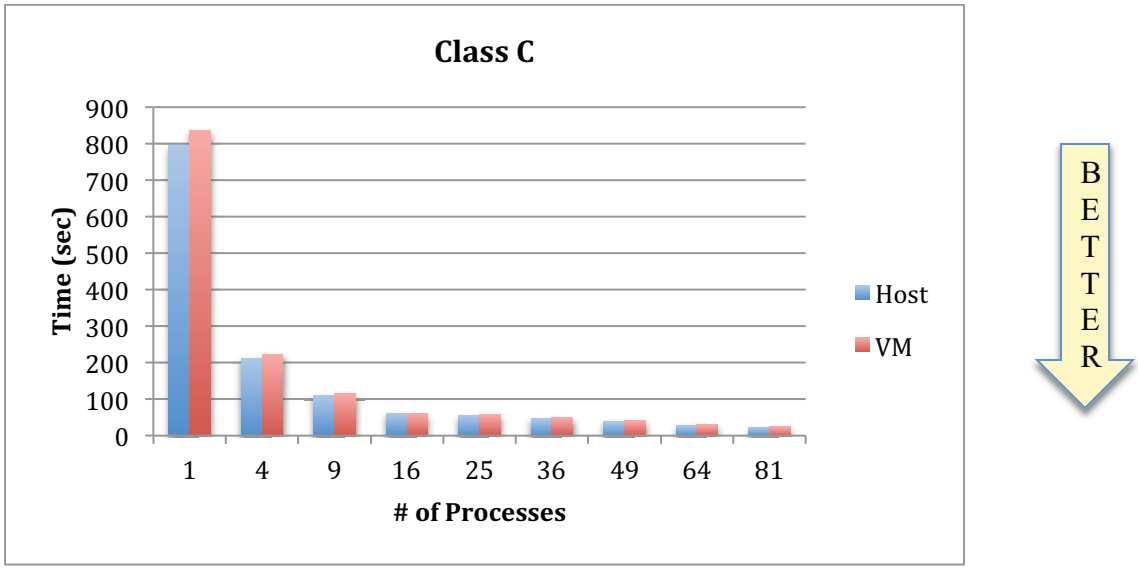

Figure 30 - SP Class C

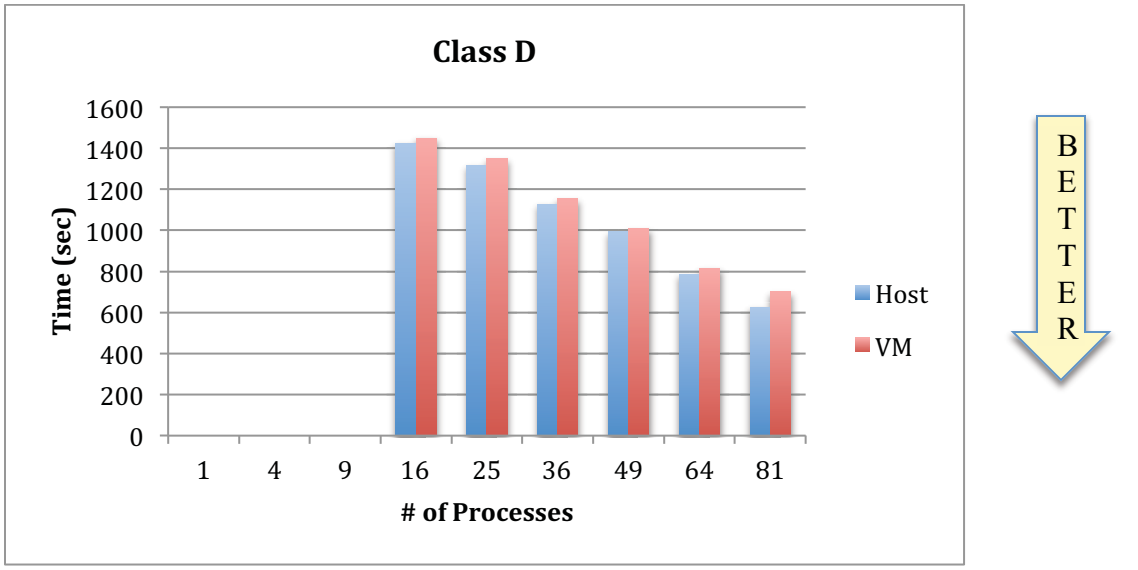

Figure 31 - SP Class D

For the Class D, 81 process case the VM performance was approximately 12% slower in time than the real Host based cluster.

5.4.8 Lower-Upper Gauss-Seidel Solver (LU)

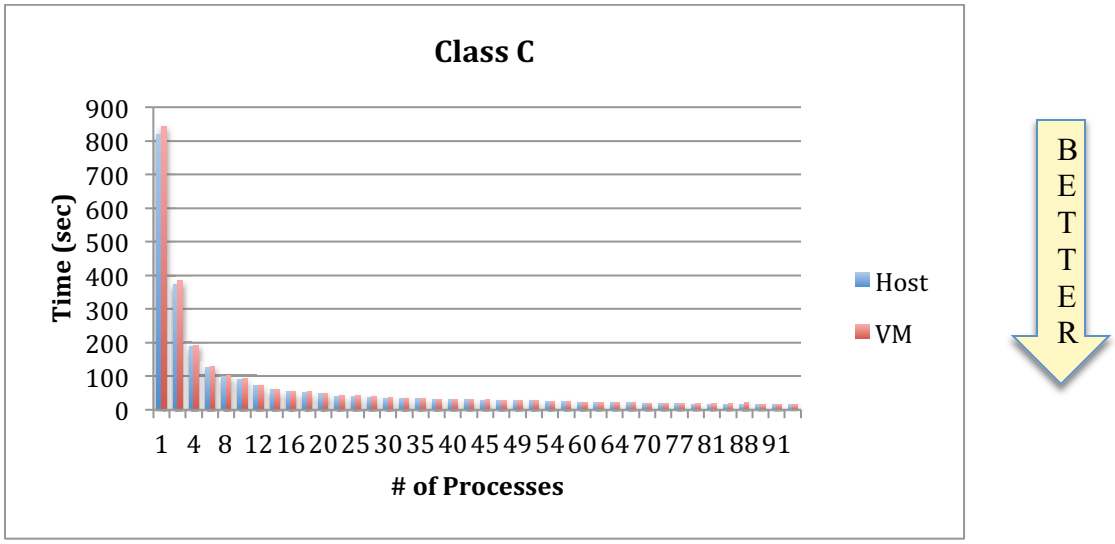

Figure 32 - LU Class C

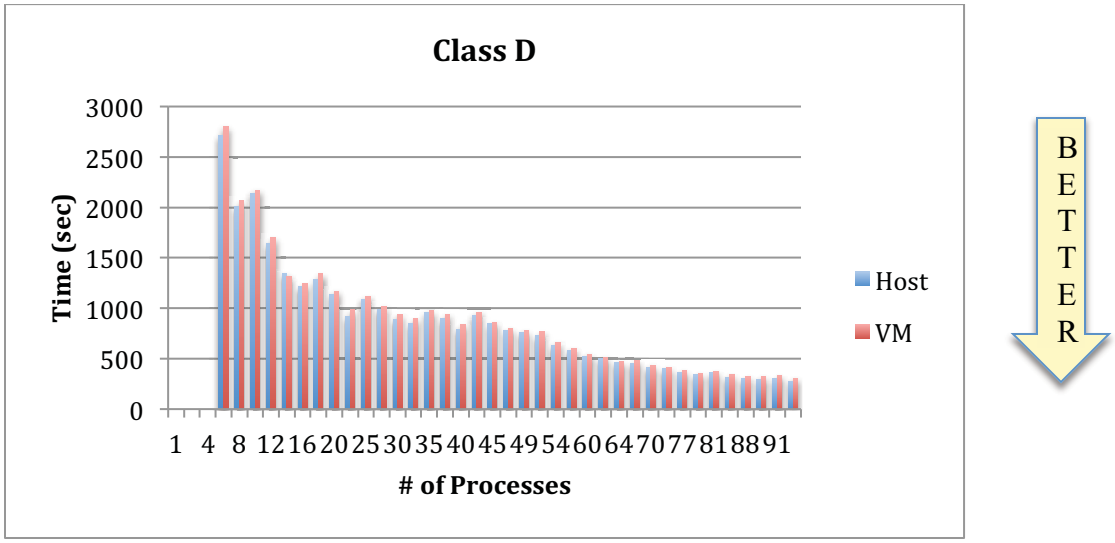

Figure 33 - LU Class D

For the Class D, 91 process case the VM performance was approximately 9% slower in time than the real Host based cluster.

6 Conclusions

The objective of this effort was to assess the viability of HPC in a virtualized environment such as a cloud. The results for the most part go beyond encouraging to almost a conclusive yes at least at the scale of the test system, eight nodes. But that prompts a key question, what happens beyond eight nodes when a cluster is comprised of hundreds if not thousands of nodes? Even within the eight node environment, the impact of multiple mpi processes on VM performance was small but nonetheless significant which may in fact be even more significant at higher node counts. This sets the stage for the next steps:

- Acquire additional C6100 nodes to evaluate higher density scaling.

- Move to a Sandy Bridge based test cluster. The conjecture is that an additional feature, namely x2APIC [9] will lessen if not negate the mpi process impact. X2APIC essentially virtualizes interrupt handling.
- Work with the OpenStack community to motivate the inclusion HPC semantics. Currently setup of such a virtual HPC environment is labor intensive.
- Evaluate advancements being made in other network technologies to determine competing cluster infrastructures should be evaluated.

References

[1] PCI-SIG SR-IOV Primer: An Introduction to SR-IO Technology
http://www.intel.com/content/www/us/en/pci-express/pci-sig-sr-iov-primer-sr-iov-technology-paper.html

[2] Intel® Virtualization Technology for Directed I/O (VT-d): Enhancing Intel platforms for efficient virtualization of I/O devices Submitted by TW Burger on Mon, 03/05/2012 - 23:16
http://software.intel.com/en-us/articles/intel-virtualization-technology-for-directed-io-vt-d-enhancing-intel-platforms-for-efficient-virtualization-of-io-devices

[3] Intel® 64 Architecture x2APIC Specification
http://www.intel.com/content/www/us/en/architecture-and-technology/64-architecture-x2apic-specification.html

[4] STREAM: Sustainable Memory Bandwidth in High Performance Computers
http://www.cs.virginia.edu/stream/

[5] OSU Micro-Benchmarks
http://mvapich2.cse.ohio-state.edu/benchmarks/

[6] LINPACK
http://www.netlib.org/linpack/

[7] Intel® Math Kernel Library – LINPACK Download
http://software.intel.com/en-us/articles/intel-math-kernel-library-linpack-download

[8] NAS Parallel Benchmarks
http://www.nas.nasa.gov/publications/npb.html

[9] Intel® 64 Architecture x2APIC Specification
http://www.intel.com/content/www/us/en/architecture-and-technology/64-architecture-x2apic-specification.html