Performance Evaluation of HPC Benchmarks on VMware's ESXi Server

Qasim Ali, Vladimir Kiriansky, Josh Simons and Puneet Zaroo
{qali, vkiriansky, simons, puneetz}@vmware.com

VMware

Abstract. A major obstacle to virtualizing HPC workloads is a concern about the performance loss due to virtualization. We will demonstrate that new features significantly enhance the performance and scalability of virtualized HPC workloads on VMware's virtualization platform. Specifically, we will discuss VMware's ESXi Server performance for virtual machines with up to 64 virtual CPUs as well as support for exposing virtual NUMA topology to guest operating systems, enabling the operating system and applications to make intelligent NUMA aware decisions about memory allocation and process/thread placement. NUMA support is especially important for large VMs which necessarily span host NUMA nodes on all modern hardware. We will show how the virtual NUMA topology is chosen to closely match physical host topology, while preserving the now expected virtualization benefits of portability and load balancing. We show that the benefit of exposing the virtual NUMA topology can lead to performance gains of up to 167%. Overall, we will show close to native performance on applications from SPEC MPI V2.0 and SPEC OMP V3.2 benchmarks virtualized on our prototype VMware's ESXi Server.

Keywords: Non Uniform Memory Architecture (NUMA), ESXi, High Performance Computing (HPC), virtual NUMA (vNUMA), virtualization

1 Introduction

Interest in system virtualization technologies for HPC applications is increasing [12, 14, 5–7, 4, 9]. While much of this interest stems from a desire to exploit cloud computing approaches, virtualization offers additional values for HPC [11, 6, 10]. These include both proactive and reactive application resiliency; dynamic resource management for scheduling efficiency and power management; multi-tenant security; and operational flexibilities.

Despite these potential values, adoption of virtualization for HPC will be determined in large part by the performance achievable on relevant workloads in virtualized environments. And that performance will be determined primarily by two factors: the hardware support for virtualization and the capabilities of the virtual infrastructure that provides the virtual machine (VM) abstraction to the guest operating system instance and its applications.

While there are many aspects of the VM abstraction that contribute to the delivered performance of an application, this paper focuses on two such capabilities of particular importance to HPC. The first is support for VMs with many virtual CPUs which is required to allow thread-parallel OpenMP and other similar codes including hybrid MPI/OpenMP applications to take full advantage of the underlying cores in modern multi-core systems. The second is support for exposing the NUMA topology of the underlying hardware to the guest operating system so that it, along with any NUMA-aware runtime libraries, can optimally co-locate compute and data where possible.

The paper presents a brief overview of ESXi server, VMware's commercial hypervisor, with an emphasis on its scalability properties. We then describe how ESXi exposes a virtual NUMA topology to guest operating systems, and finally present experimental results using SPEC OMP and SPEC MPI as representative workloads.

2 ESXi Server Architecture

VMware ESXi Server is VMware's bare-metal hypervisor which runs directly on physical hardware. It multiplexes physical hardware among multiple VMs and works in conjunction with the virtual machine monitor (VMM), an instance of which runs per VM. By managing hardware resources directly, ESXi Server achieves high performance by reducing virtualization overhead [13].

In about a decade of existence, VMware's server virtualization platform has undergone many advancements to increase its scalability and performance. The scalability improvements have led to support for larger physical and virtual processor counts and memory sizes, as well as higher VM consolidation rates. These scalability increases have been enabled by advancements in both the virtual machine monitor and ESXi hypervisor. Some key features which have enabled these advances are architectural changes to support large virtual processor counts (large SMP VMs), support for 64 bit x86 architecture and efficient use of hardware virtualization support [2], along with advanced physical CPU, memory and IO management [13]. Supporting large SMP VMs required careful data structure design, coscheduling improvements [3], fine-grained locking, and best software engineering practices to enable support for 64 virtual CPUs in a VM with minimal virtualization overhead. In the next section, we discuss the presentation of a virtual NUMA topology to guest operating systems running inside VMs, which proved crucial to the performance of virtualized HPC workloads.

3 Virtual NUMA support

Current generation operating systems, runtime libraries and applications are expected to be NUMA-aware for improved performance, e.g. they would allocate memory and schedule execution threads to take advantage of the NUMA topology. To support this within a virtual environment, we introduce the concept of a *virtual NUMA topology* which can be exposed to a guest operating system. This

abstraction is then mapped by the ESXi NUMA scheduler to an intermediate level of abstraction *physical NUMA node*, which is in turn dynamically bound to specific *machine NUMA node* resources on the host system. This hierarchy of (virtual, physical, machine) is analogous to that used to implement virtual memory in a virtualized environment. It should be noted that there is not necessarily a one-to-one correspondence between virtual and physical NUMA nodes since multiple physical NUMA nodes may be required to provision the required CPUs and interleaved memory for a virtual NUMA node. Similarly, multiple physical NUMA nodes may be scheduled on the same machine NUMA node, and even over-commit available CPU and memory resources.

The number of virtual NUMA nodes, the number of virtual CPUs (vCPUs) and the amount of memory associated with the virtual nodes normally remain fixed for the lifetime of a VM. We also support hot-plug add of CPU and memory resources, though not all OSes support hot-plug remove or dynamic changes to NUMA topology. Our virtual BIOS exposes the ACPI Static Resource Affinity Table (SRAT) [1] and ACPI PXM methods to guest OSes to reveal which memory regions and vCPUs belong to which virtual NUMA node. Minimally NUMA aware OSes only distinguish between local vs remote allocation; most modern ones take into account the minimal inter node latencies (or number of hops); yet more advanced OSes need to track maximum link bandwidths, and ultimately total system bandwidth for optimal scheduling. There are no standard facilities for exposing the actual link topology connecting NUMA nodes, e.g. Intel QPI or AMD HyperTransport frequency, number of lanes, lane widths, and for partially connected systems the routes and congestion policies that determine the maximum total interconnect bandwidth. We don't expose ACPI System Locality Information Table (SLIT) [1] information which would only provide fixed latency information. Most modern guest OSes measure during initialization the unidirectional access costs for a vCPU from one virtual NUMA node accessing memory on another virtual NUMA node.

There are several scenarios in which the NUMA topology deduced by the guest OS may become inaccurate over the lifetime of the virtual machine and the remote memory access latency and bandwidth may change. First, the VM might get powered on, suspended and resumed, or live migrated to hosts with different machine NUMA topology than the original system. If different number of physical nodes may be needed, e.g. if originally 4 nodes were used and now 8 nodes are needed, memory interleaving across pairs of physical nodes will be necessary. The physical NUMA node abstraction accommodates even a VM started with 8 vCPUs per NUMA node after it is migrated to a node with 6 PCPUs per NUMA node with best efforts to minimize the performance impact. Second, even if the same number of physical nodes are needed they may use a different NUMA topology. For example, when no subset of 4 machine nodes on an 8-socket host is fully connected, a 4-node VM would need more hops for some remote accesses, instead of symmetric accesses on a fully connected 4-socket host.

It is also possible for the ESXi NUMA scheduler to choose different actual NUMA nodes for placement based on the system workload. Physical NUMA

nodes belonging to the same VM which are smaller than the machine NUMA node may be either consolidated or spread over multiple machine nodes. For example, a single VM workload may benefit from spreading over four NUMA nodes to gain higher memory bandwidth and cache capacity, but two such VMs should each be using two non-overlapping NUMA nodes and links.

Trading off maximum single VM performance versus optimal total system performance and overall fairness across VMs can add additional constraints. While the optimal choices are specific to the VM workload, the best physical NUMA topology and placement on machine nodes depends on the overall load on the system and each NUMA node load. Other VMs' CPU utilization, cache capacity, and memory bandwidth consumption may impact the scheduler choices as well. We expose manually configurable policies that affect both the fixed virtual NUMA topology, initial physical NUMA topology placement, and load balancing.

If the memory bandwidth demands of the VM are very high, then using multiple memory controllers will be more beneficial and thus spreading over multiple machine NUMA nodes would be favored. If there are high levels of inter-thread communication, sharing the same last level cache will be preferred and thus consolidation over fewer machine nodes will be favored. Finally, a VM may be configured to favor using SMT hyperthreads instead of full cores - appropriate when the lower latency of local memory may outweigh the disadvantage of not using the full resources of a core. This benefits workloads with higher level of inter-thread communication or external I/O, where cores will otherwise be underutilized. So, for example, a VM with 16 vCPUs, on a host with four 8-core (16 SMT) sockets will perform best with either one, two, or four NUMA nodes, depending on the workload.

The above complexities and potential divergence over time of guest OS determined topology are most pronounced for heterogeneous workloads consolidated on hosts in heterogeneous clusters. Maximum performance of a single VM running on hosts with identical machine NUMA topology is an equally important scenario. In our experimental evaluation we discuss performance of VMs with maximum vCPU count with matching virtual, physical and machine topology. We expect these to be typical configurations for virtualized HPC environments which generally will not over-subscribe hardware resources and which employ homogeneous clusters.

4 Experimental Evaluation

We evaluated our prototype ESXi Server using four applications from the SPEC MPI V2.0 suite and seven applications from SPEC OMP V3.2 suite. SPEC MPI workloads are meant to be evaluated on distributed memory machines but can be useful for performance evaluation on shared memory machines as well. SPEC OMP is designed for use on shared-memory machines and is well-suited to evaluate the performance of large SMP virtual machines for HPC. All the experimental results reported here were gathered on 2.27 GHz Intel(R)

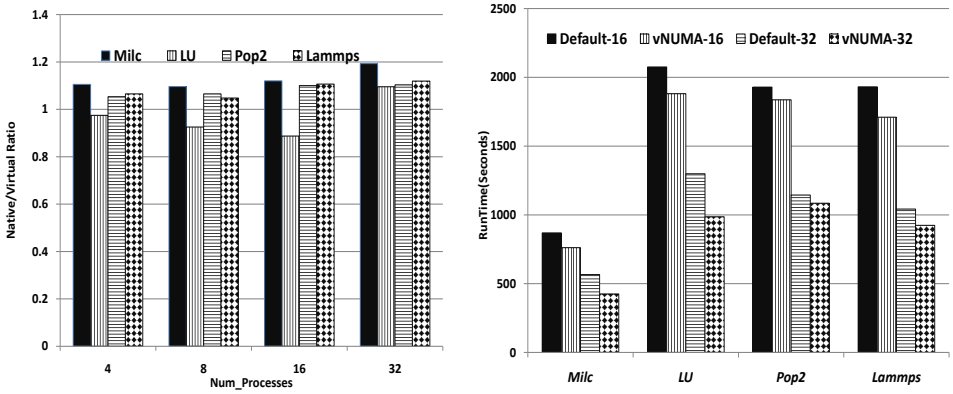
Xeon(R) CPU X7560 processors based on the Nehalem-EX architecture. All the benchmarks were compiled using gcc-4.3 and gfortran. We present three types of results in this paper:

- Gains obtained due to vNUMA
- Native to virtual runtime ratio showing virtual performance
- Native and virtual scaling

Our virtual NUMA(vNUMA) results start at 16 vCPUs since this represents the two-socket case on our test system. Native to virtual ratio is obtained by dividing the completion time of an application in the native case by the completion time of the virtual run. A ratio greater than 1 indicates that a VM runs faster than the native. Scaling results are reported starting at four vCPUs / processes / threads because some MPI tests failed to run using fewer than four processes. All of the SPEC benchmark applications were run for two iterations, which is not SPEC-compliant. Run-to-run variation was within 1-2%. The numbers reported here are the average of the two runs.

4.1 SPEC MPI Results

The SPEC MPI applications were evaluated on a Dell PowerEdge R910, which has four sockets and a total of 256 GB memory. RHEL 5.4 was used as a native and guest OS with kernel version 2.6.18-164. All VMs were configured with 128 GB of virtual RAM. We used Open MPI v1.4 [8] and the SPEC MPI medium dataset for our tests.



(a) Native to Virtual Runtime Ratio (b) Default vs vNUMA Runtime

Fig. 1. SPEC MPI Virtualization performance, and Interleaving vs vNUMA with vCPU count = Number Of Physical Cores = 32

We conducted two sets of experiments. In the first set, the number of vCPUs is equal to the number of physical cores in the system (32 in this case) and the

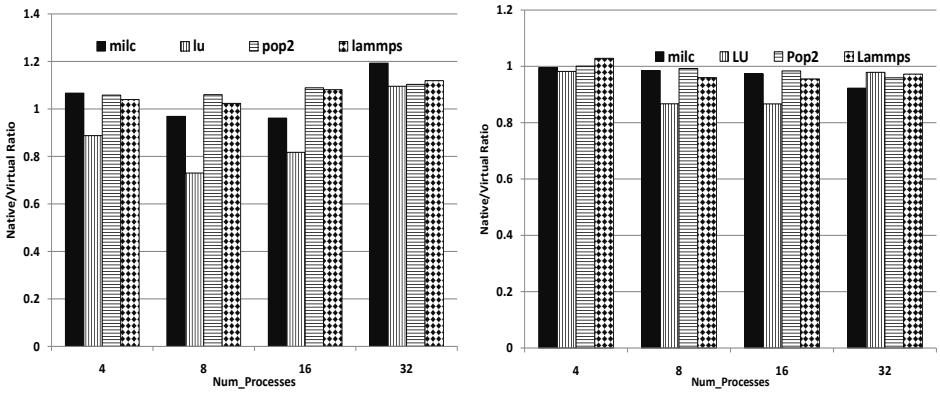
number of processes spawned is the same as in the case of native. This configuration is similar to the native setup in the sense that the guest OS scheduler would schedule as it would on a native system with the minor difference that in the native case it would schedule on 64 logical threads, while in the virtual case, it would schedule on 32 vCPUs and the 32 vCPUs would then be scheduled by the ESXi scheduler on the 64 logical threads. We do not expose hyper-threading information to the guest and for this workload we need full cores for each vCPU. While this configuration has the best performance, we discuss alternative VM size configurations later in our evaluation. In the second set of experiments, the number of vCPUs configured for a VM is set equal to the number of processes spawned in each SPEC MPI run – we size the VM equal to the size of the MPI job.

Figure 1(a) shows the native to virtual runtime ratio for four applications from the SPEC MPI V2.0 suite. The ratio for most of the applications is close to one which indicates that virtualization is adding little or no overhead in this case. In certain cases we see up to 20% better virtual performance than native as in the case of the milc application run with 32 processes. We observed that native Linux scheduler gives better performance if HT is OFF versus when it is ON (32-process SPEC MPI applications on the 64 logical threads). We also observed that many SPEC MPI applications on a native system with 64 processes on a 64 logical core system like the Nehalem-EX system were not gaining much over the performance of the 32-process run. Hence we sized the VM with 32 vCPUs (because there were 32 physical cores in the system). Also typically in HPC applications, processes/threads are spawned based on number of physical cores rather than logical cores.

Figure 1(b) shows the performance gain obtained due to exposing to the guest OS a virtual NUMA topology matching the machine topology. In this figure, Default-16 means that vNUMA is disabled (memory is interleaved over the physical NUMA nodes) and the suffix 16 means that sixteen MPI processes were spawned within the MPI application, whereas vNUMA-16 means that vNUMA is enabled (virtual NUMA nodes match physical NUMA nodes). The gain due to vNUMA for milc were 14% and 33% for the 16 and 32 vCPU cases, respectively. Similarly for LU, the gains are 11% and 32% for 16 and 32vCPU VMs. Noticeable gains were also observed for the pop2 and lammps applications.

In order to better understand the trade-offs in performance, we modified the virtual experiments so that the number of vCPUs configured for a VM is equal to the number of processes spawned in SPEC MPI applications. Figure 2(a) shows the results with the number of vCPUs equal to number of processes.

In Figure 2(a), milc and LU actually lose performance for 4, 8 and 16 rank runs when compared to results in Figure 1(a). This is because the scheduling decisions in native and virtual environments were different. For example, in the native four-process case the Linux scheduler spreads the workload across multiple sockets, which increases the effective cache size and delivers better performance for applications that benefit from larger cache sizes. ESXi instead will schedule the four virtual processors on a single socket for better consolidation and to



(a) vCPU count = Num_processes (b) vCPU count = 32 with HT OFF

Fig. 2. SPEC MPI Virtualization Performance with vCPUs = processes, and with hyper-threading off

increase cache sharing effects. That is the reason why LU's native to VM ratio is in between 0.7 and 0.9 for 4, 8 and 16vCPU runs. In the light of these results, the best practice would be to size your VMs depending on your application's cache footprint.

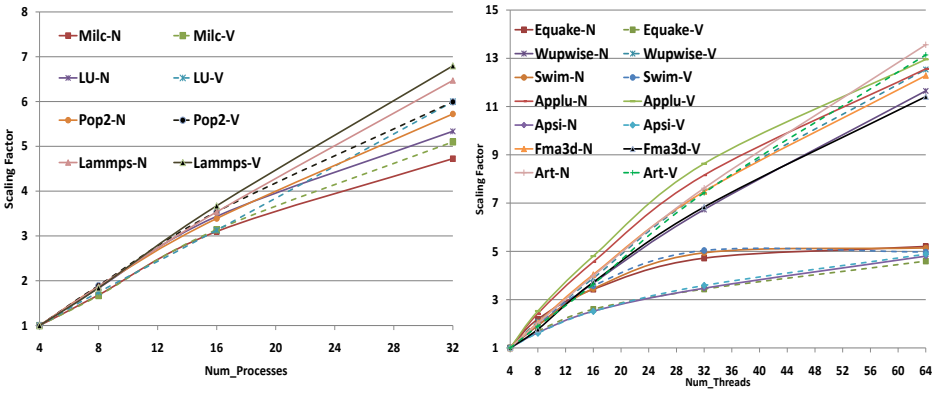
Both Figure 1(a) and Figure 2(a) show that native suffers performance degradation with HT. To test whether HT is responsible for this degradation, we disabled HT from the system BIOS. Figure 2(b) shows the native to virtual ratio with HT OFF in both the VM and the native case with vCPU count equal to 32. The ratio is less than one for almost all cases (expect lammps which is slightly higher). Given the fact that both native and virtual configurations represented in Figure 2(b) have same amount of cache, we conclude that the ESXi scheduler is more optimal than the native scheduler in this case.

Figure 3(a) shows native and virtual scaling of the four applications from the SPEC MPI V2.0 suite. The suffix "-N" means native run and "-V" means virtual run. In this figure, HT is ON, which explains why virtual numbers are sometimes better than native, e.g., the 32-process data point.

In the light of the above results, for the virtual runs, HT ON and number of vCPUs being equal to the physical cores in the system is the best configuration. We used this configuration for SPEC OMP V3.2 benchmarks in the next section.

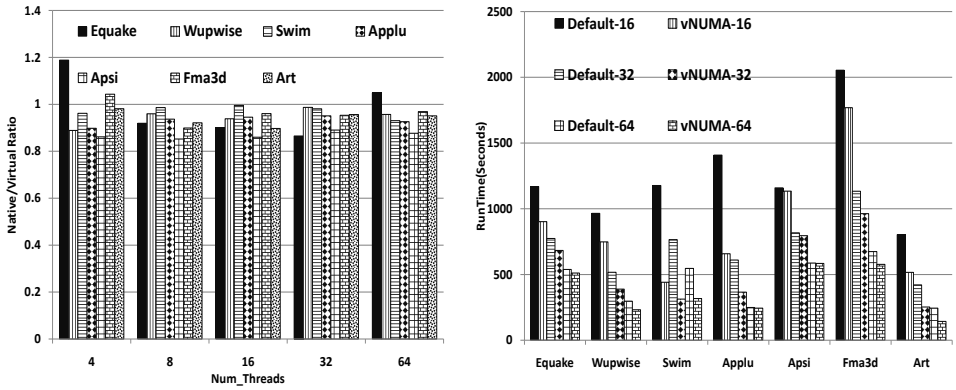
4.2 SPEC OMP Results

SPEC OMP applications were evaluated on an HP ProLiant DL980 G7, with eight sockets and 512 GB RAM. RHEL 6.0 was used as a native and guest OS with kernel version 2.6.32-71.el6. All VMs were configured with 128 GB of virtual RAM. The large data sets were used for all SPEC OMP experiments. Figure 4(a) shows the virtualization performance for SPEC OMP V3.2 benchmarks. The native to virtual runtime ratio was close to one for all applications expect for



(a) SPEC MPI V2.0 Applications (b) SPEC OMP V3.2 Applications

Fig. 3. Native and Virtual Scaling with vCPUs = No. Of Physical Cores.



(a) Native to Virtual Runtime Ratio (b) Default vs vNUMA Runtime

Fig. 4. SPEC OMP Virtualization performance, and Interleaving vs vNUMA with vCPU count = Number Of Physical Cores = 64

equake (four process and 64 process run) and fma3d (4 process run) where the ratio was greater than one. As was explained earlier, this is because native scheduling with HT ON is sub-optimal.

Figure 4(b) shows performance gains due to vNUMA for seven different applications from the SPEC OMP V3.2 suite.

Strikingly, swim with 16 threads and virtual NUMA performs better than 32 or even 64 threads without virtual NUMA topology! This shows the memory bandwidth demands of this benchmark are much more critical than the thread count. With 16 threads the gains on swim from vNUMA are 167%, but with 64 threads the benefit is down to 72%, probably because of saturated memory bandwidth.

Art starts with 56% vNUMA gains at 16 threads, and the benefits increase to 71% at 64 threads. Most likely memory bandwidth becomes a bigger bottleneck at higher thread counts and bandwidth is still available.

Fma3d and Wupwise have a relatively constant gain from vNUMA at different thread counts respectively 17% and 30% where both peak at 32 threads. Given that the increased memory bandwidth demand of higher number of threads doesn't affect performance, these applications are probably latency-sensitive but not bandwidth-sensitive. Our hypothesis is that instead of 12.5% locality when using 8 sockets, the lower latency for accessing up to 100% local memory is the main reason for these gains.

Applu and Equake gain significantly at 16-threads respectively 114% and 30%, but at 64-threads the vNUMA gains drop to 2% and 5%. This implies that initially memory bandwidth was the biggest bottleneck but new bottlenecks emerge at higher thread counts. Little gain in Apsi was observed probably because of a smaller memory bandwidth demand.

Overall at 64 threads two applications get more than 70% gains from vNUMA, and the geometric mean of the performance gains over all applications is 25%. All applications achieve their best performance at 64 threads but don't scale equally well.

Figure 3(b) shows native and virtual scaling for the seven applications. All applications show similar trends for virtual and native scaling. Some virtual and scaling data points were not close due to the difference in HT scheduling. One important point to note here is that Apsi, Swim and Equake are not scaling as nicely as other applications beyond 16 threads. Evaluating a multi-VM scenario on a single host as well on a cluster of hosts will be covered in future work.

5 Conclusion

We demonstrated that HPC workloads on VMware's virtualization platform achieve close to native performance (in some cases even 20% better than native) with applications from SPEC MPI and SPEC OMP benchmarks. We evaluated the new features in VMware's ESXi Server that significantly enhance the performance and scalability of HPC workloads.

Exposing a virtual NUMA topology that closely matches the physical host topology is a major feature, bridging the gap between native and virtual performance for VMs with large memory and high vCPUs counts. Gains of up to 2.67x were observed.

Virtual machines with up to 64 virtual CPUs now achieve their best performance when a virtual NUMA topology is exposed to guest operating systems. This allows operating systems and applications to make intelligent decisions about memory allocation and process/thread placement. We discussed the features required for a production system that exposes virtual NUMA topology. We preserve the now-expected virtualization benefits of portability and load balancing both within a host and a cluster, yet with with minimal overhead for a single VM.

Acknowledgments We'd like to thank our colleagues Jeffrey Buell, Ishaan Joshi, and Seongbeom Kim for helpful experiments and suggestions, and our monitor and VMkernel group colleagues for making this work possible.

References

1. Advanced Configuration and Power Interface specification, rev 4.0a, 2009. `http://www.acpi.info/spec40.htm`.
2. O. Agesen, A. Garthwaite, J. Sheldon, and P. Subrahmanyam. The evolution of an x86 virtual machine monitor. *Operating Systems Review*, 44(4), 2010.
3. Scalable Infrastructure with the CPU scheduler in VMware ESX 4.1. `http://www.vmware.com/files/pdf/techpaper/VMW_vSphere41_cpu_schedule_ESX.pdf`.
4. A. Gavrilovska and S. K. et al. High-Performance Hypervisor Architectures: Virtualization in HPC Systems. In *In HPCVirt 2007: 1st Workshop on System-level Virtualization for High Performance Computing*, 2007.
5. W. Huang, Q. Gao, J. Liu, and D. K. Panda. High performance virtual machine migration with RDMA over modern interconnects. In *Proceedings of the 2007 IEEE International Conference on Cluster Computing*, CLUSTER '07, pages 11–20, Washington, DC, USA, 2007. IEEE Computer Society.
6. M. F. Mergen, V. Uhlig, O. Krieger, and J. Xenidis. Virtualization for high-performance computing. *SIGOPS Oper. Syst. Rev*, 40:8–11, 2006.
7. A. B. Nagarajan and F. Mueller. Proactive fault tolerance for HPC with Xen Virtualization. In *In Proceedings of the 21st Annual International Conference on Supercomputing (ICS07)*, pages 23–32. ACM Press, 2007.
8. Open MPI: Open Source High Performance Computing, `http://www.open-mpi.org`, 2011.
9. A. Ranadive, M. Kesavan, A. Gavrilovska, and K. Schwan. Performance implications of virtualizing multicore cluster machines. In *Proceedings of the 2nd workshop on System-level virtualization for high performance computing*, HPCVirt '08, pages 1–8, New York, NY, USA, 2008. ACM.
10. D. Rao and K. Schwan. vNUMA-mgr: Managing VM memory on NUMA platforms. In *High Performance Computing (HiPC), 2010 International Conference on*, pages 1 –10, dec. 2010.
11. J. E. Simons and J. Buell. Virtualizing high performance computing. *SIGOPS Oper. Syst. Rev.*, 44:136–145, December 2010.
12. G. Valle, C. Engelmann, S. L. Scott, T. Naughton, and H. Ong. System-Level Virtualization for High Performance Computing, Feb. 13-15, 2008.
13. C. A. Waldspurger. Memory resource management in VMware ESX server. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, pages 181–194, New York, NY, USA, 2002. ACM Press.
14. L. Youseff, K. Seymour, H. You, J. Dongarra, and R. Wolski. The impact of paravirtualized memory hierarchy on linear algebra computational kernels and software. In *Proceedings of the 17th international symposium on High performance distributed computing*, HPDC '08, pages 141–152, New York, NY, USA, 2008. ACM.