The Effect of Multi-core on HPC Applications in Virtualized Systems

Jaeung Han[1], Jeongseob Ahn[1], Changdae Kim[1],
Youngjin Kwon[1], Young-ri Choi[2], and Jaehyuk Huh[1]

[1] Computer Science, KAIST, Daejeon, Korea
[2] Korea Institute of Science and Technology Information (KISTI), Daejeon, Korea

Abstract. In this paper, we evaluate the overheads of virtualization in commercial multicore architectures with shared memory and MPI-based applications. We find that the non-uniformity of memory latencies affects the performance of virtualized systems significantly. Due to the lack of support for non-uniform memory access (NUMA) in the Xen hypervisor, shared memory applications suffer from a significant performance degradation by virtualization. MPI-based applications show more resilience on sub-optimal NUMA memory allocation and virtual machine (VM) scheduling. However, using multiple VMs on a physical system for the same instance of MPI applications may adversely affect the overall performance, by increasing I/O operations through the domain 0 VM. As the number of cores increases on a chip, the cache hierarchy and external memory will become more asymmetric. As such non-uniformity in memory systems increases, NUMA and cache awareness in VM scheduling will be critical for shared memory applications.

1 Introduction

Virtualization has become popular to improve system utilization by consolidating multiple servers into a physical system. In addition to the improved utilization, other benefits of virtualization, such as flexible resource management, fault isolation, and support for different operating systems, have led to the increase of interest in the virtualization of computing clusters for high performance computing (HPC). Public cloud computing services, such as Amazon EC2 [1], also accelerated the adoption of virtualization for HPC applications. However, the characteristics of compute-intensive HPC applications are quite different from those of I/O-intensive server applications. To adopt virtualization for HPC applications, thorough analysis of their performance characteristics in virtualized systems is necessary. Furthermore, the fast increase of core counts in multicore architectures, combined with virtualization techniques, affects the performance of HPC applications significantly.

In multicore architectures, the effects of complicated memory hierarchies, such as non-uniform memory access (NUMA), have become significant for HPC applications. Virtualization hides the underlying non-uniformity in memory access, and thus a guest operating system may not be able to make optimal scheduling decisions.

M.R. Guarracino et al. (Eds.): Euro-Par 2010 Workshops, LNCS 6586, pp. 615–623, 2011.

In this paper, we investigate the overheads of virtualization on HPC applications running on multicore systems with uniform and non-uniform memory access latencies. Using the Xen hypervisor, we evaluate both a shared-memory multi-threaded benchmark, PARSEC [4], and a MPI-based benchmark, NAS Parallel Benchmark (NPB) [3] in various configurations.

The experimental results show that for shared memory applications, the performance overheads by virtualization are minor with uniform memory latency. However, in non-uniform memory access architecture, the current Xen hypervisor [6] adds a significant overhead for shared-memory applications and small overhead for MPI applications. However, for MPI applications, the granularity of VMs, the number of virtual CPUs (vCPUs) per VM, is important.

2 Methodology

2.1 Target Multicore Architectures

We use two different types of commercial multicore systems to evaluate HPC applications with virtualization. The first system is a single-socket system with a 12-core AMD Opteron 6168 processor (`single-socket`), which is a multi-chip module with two dies packaged together. Each die has six cores. Each core has separate 64KB instruction and data caches, and 512KB L2 cache. Six cores in a die share a 6MB L3 cache. The twelve cores in the system have almost uniform memory latencies to any memory modules.

The second system (`dual-socket`) uses two Intel Nehalem E5530 processors, which have four cores in each processor. Each core has separate 32KB instruction and data caches, and a 256KB private L2 cache. Four cores in a processor share an 8MB L3 cache. In the dual-socket system, two quad-core processors are connected by QPI interconnections. With the QPI interconnections, each processor has its own DRAM memory banks. An important characteristic of the system is non-uniform memory access (NUMA).

2.2 Methodology

To evaluate the effects of virtualization, we use the Xen hypervisor (version 3.4.2) [6]. We compare the performance of two selected benchmarks on virtualized configurations to that on non-virtualized (native) configurations. The guest operating system in the virtualized configurations is a Linux (kernel version 2.6.31.13) modified to support the para-virtualization mode of the Xen hypervisor. For the operating system in the native configurations, the same version of the Linux kernel is used.

We use two benchmarks representing different uses of HPC clusters: PARSEC [4] is a shared-memory multi-threaded benchmark with a single physical machine. We use the native input set for PARSEC. As a MPI-based benchmark, we evaluate the NAS parallel benchmark (NPB) [3]. To evaluate the overheads of MPI communications, we connected two systems by a 1gigabit Ethernet switch, and used the MPICH 1.2 library [2]. For NPB, we use the class C input set.

2.3 Virtual Machine Scheduling

In the Xen hypervisor, the unit of scheduling is a virtual CPU (vCPU). Each VM may have multiple vCPUs, emulating a multiprocessor system. The Xen hypervisor assigns credits to each vCPU periodically to guarantee fairness among vCPUs. Since vCPUs are scheduled independently, there is no guarantee that the vCPUs from a single VM are scheduled together. The Xen hypervisor maintains queues for each physical core, but vCPUs may migrate to all the physical cores freely unless they are pinned to specific cores. In the default setting, the scheduler will try to maximize the overall throughput by not wasting any CPU cycles. Whenever a core becomes idle, it will attempt to steal active vCPUs waiting in the queues of other cores.

In the target dual-socket system, relocating a thread across the processor boundary may cause two effects: shared L3 cache and NUMA effects. When a thread migrates from a processor to the other processor, it can no longer access the cached data in the L3 cache in the old processor directly. The other effect is non-uniform memory access latencies. Depending on which memory modules a thread mostly accesses, the processor where the thread is running may have a significant effect on the overall performance due to non-uniform memory access latencies.

3 Shared Memory Applications: PARSEC

3.1 Performance

Single Socket Results: To isolate the effect of NUMA, we first evaluate the effect of virtualization by using a system with one processor (single-socket). Among 12 cores, we use only 8 cores to be consistent with dual-socket results. Figure 1 presents the execution times of the PARSEC benchmark normalized to those of the native system with the same number threads. In this experiment, the vCPUs are not pinned to physical cores, and thus the Xen scheduler can migrate vCPUs without any restriction to minimize unused CPU cycles. For each application, three bars are shown: one, four, and eight vCPUs. The number of threads in each application is set to the number of vCPUs.

In general, for the single-socket system, the performance overheads by virtualization are insignificant, regardless of the number of vCPUs. The Xen hypervisor

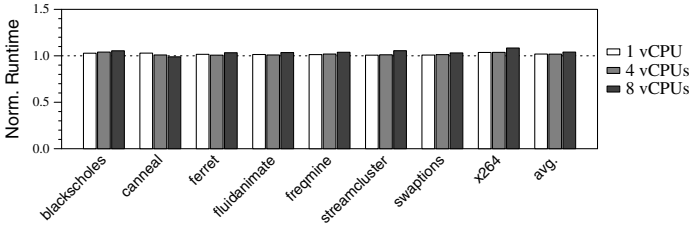

Fig. 1. Single socket (*unpinned* vCPUs): execution times with 1, 4, and 8 vCPUs

supports efficient virtualization for compute-intensive shared-memory applications for the single-socket system with uniform memory access. To further investigate the effect of scheduling, we fix vCPUs to physical cores. Figure 2 presents the execution times normalized to those of the native system, when vCPUs are pinned to physical cores. The results are similar to those with the unpinned configuration. With uniform memory access latencies, mapping between vCPUs and physical cores does not have a significant impact on the performance of the PARSEC applications. Furthermore, the cost of vCPU migration across shared L3 caches is minor, as shown by the almost same performance by the pinned and unpinned configurations.

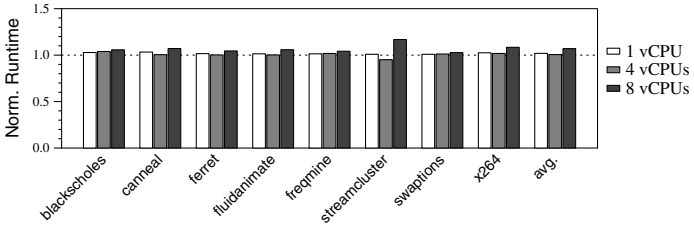

Fig. 2. Single socket (*pinned* vCPUs): execution times with 1, 4, and 8 vCPUs

Dual Socket Results. To include the NUMA effect, we use a dual-socket system in which each socket has four cores. Figure 3 presents the execution times with the dual-socket system normalized to those of the native system. In this experiment, the vCPUs are not pinned to physical cores. Unlike the previous single socket results, the performance degrades significantly. The performance degradation is 12% for 1 vCPU, 16% for 4 vCPUs, and 37% for 8 vCPUs on average, respectively.

To eliminate the effect of vCPU migration, we fix each vCPU to a physical core. Figure 4 presents the normalized execution times (to those of the native system) with the pinned configuration. For the one and four vCPU configurations, the performance degradations reduce to 8% and 9% respectively. However, for the eight vCPU configuration, the performance degradation increases slightly to 40%.

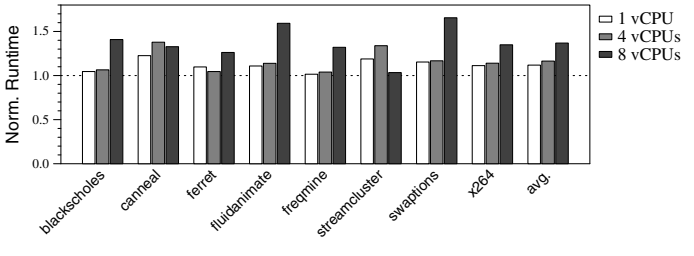

Fig. 3. Dual socket (*unpinned* vCPUs): execution times with 1, 4, and 8 vCPUs

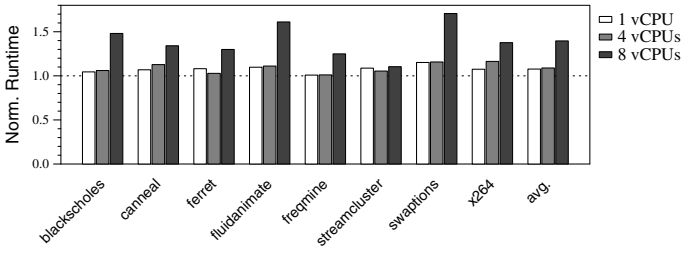

Fig. 4. Dual socket (*pinned* vCPUs): execution times with 1, 4, and 8 vCPUs

When the number of vCPUs is in the range from 1 to 4, pinning vCPUs makes the system use only one socket for the vCPUs, reducing the effect of NUMA and eliminating the cost of vCPU migration across the shared L3 cache boundary. However, for 8 vCPUs, pinning may eliminate the cost of vCPU migration across the shared L3 cache boundary, but it does not mitigate the effect of NUMA. Eight vCPUs must use all the cores in both sockets, but the memory pages of the VM are mostly located in one of the socket.

3.2 Mitigating the NUMA Effect

In this section, we isolate the effect of NUMA to further investigate its performance impact on HPC applications. To explain the benefit of pinning in the dual-socket system (as shown in Figure 4), we evaluate the "worst" and "best" case scheduling for the four vCPU configuration. Considering the NUMA effect, the worst case scheduling is to map all four vCPUs on a socket to which memory pages are not allocated. The best case scheduling is to map all four vCPUs on the same socket to which all the memory pages are located. Figure 5 presents the execution times normalized to those of the native system with the worst and best case scheduling for four vCPUs, as well as the unpinned and pinned configurations.

As shown in Figure 5, the performance with the unpinned configuration is slightly better than that with the worst case range-pinned configuration. The performance with the pinned configuration is similar to that with the best case range-pinned configuration. Pinning vCPUs has a similar effect to the best case

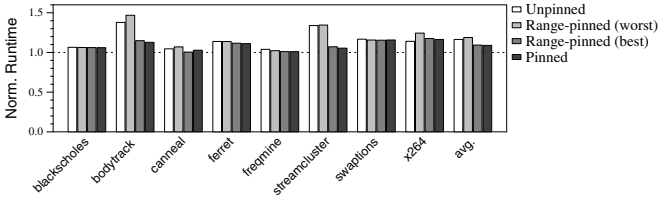

Fig. 5. The worst and best range pinning schemes for 4 vCPUs (dual-socket)

configuration, since in our experiments, all four vCPUs happen to be mapped to the same socket to which their memory pages are located. However, we expect that blindly pinning vCPUs, without considering the memory affinity, will not improve performance consistently.

However, for the eight vCPU configuration, it is not possible to find the best case scheduling, since eight vCPUs must be mapped to 8 cores in two sockets. To reduce the effect of NUMA, we modified the Xen scheduler slightly such that it attempts to schedule vCPUs to the right socket. In the PARSEC applications, all the eight vCPUs are not always used, since available parallelism dynamically changes. If less than eight vCPUs are used, active vCPUs are scheduled as much as possible to the socket in which their memory pages reside. However, we do not make any physical core idle, if there are active vCPUs not scheduled to any core. Thus, if no core in the right socket is available, a vCPU will be scheduled to the other socket. This rudimentary optimization, called `NUMA-first`, provides a significant improvement in performance. Figure 6 presents the normalized execution times with the unpinned, pinned, and NUMA-first configurations. With the NUMA-first scheduling, the average performance degradation is reduced to 18% from 37% of the unpinned configuration. The NUMA-aware scheduling requires further investigation to make it adaptable to more complex cases than our configurations.

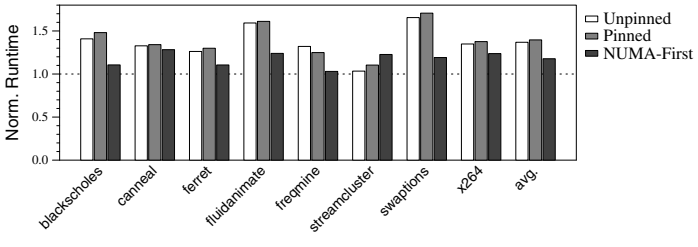

Fig. 6. *NUMA-first* optimization for 8 vCPUs (dual-socket)

4 MPI-Based Applications: NPB

In this section, we evaluate the performance overheads of virtualization with the MPI-based NPB. For the experiments in this section, the half of the total MPI processes are running in a system, and the other half are running in the other system.

Unlike shared memory applications, MPI-based applications can run with various numbers of virtual machines per system. For 16 MPI processes, in each system, 8 MPI processes can use a VM with 8 vCPUs, 2 VMs with 4 vCPUs, 4 VMs with 2 vCPUs, or 8 VMs with 1 vCPU. The VM granularity, or the number of vCPUs per VM, may have some impact on the cost of communication among MPI processes. MPI communications among processes in a VM are done only within the guest operating system. MPI communications among the VMs in a

system do not access the network hardware, but the communications must pass through the hypervisor and the domain0 VM. MPI communications among the VMs in different systems must access the network hardware, hypervisor, and the domain0 VM.

Figure 7 presents the execution times with different numbers of VMs for 16 MPI processes without pinning. Firstly, for all applications, using the largest VM (8 vCPUs per VM) is better than using multiple VMs in a system. It is because MPI communications within a VM have lower overheads than those across VMs. Secondly, for each application, if the best VM granularity (8 vCPUs per VM) is used, the performance overheads on MPI-applications by virtualization are much lower than those on shared memory applications. Even though all the 8 cores are used for each system, the average execution time is only 11% higher than that of the native system. Due to the I/O activities, NUMA effect does not dominate the overall performance.

Figure 8 presents the execution times with vCPUs pinned to physical cores. Pinning vCPUs does not improve the NPB performance for the best VM granularity, with a similar 11% average increase of execution times. However, pinning improves performance for any VM granularity other than 8 vCPU per VM.

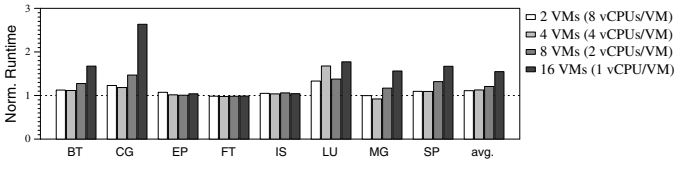

Fig. 7. NPB execution times (*unpinned* vCPUs): varying vCPUs per VM

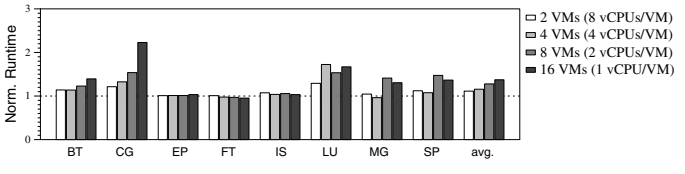

Fig. 8. NPB execution times (*pinned vCPUs*): varying vCPUs per VM

5 Related Work

The effects of virtualization on the performance of applications have been studied in previous work. Due to space limitation, we review some of such work in this section. Huang et al showed that I/O virtualization overhead is the major issue for virtualization, and proposed VMM-bypass I/O to reduce I/O virtualization overhead [9]. In [12], the effects of resource sharing (specially, sharing an Infiniband interconnect) on the performance of HPC applications were studied in a virtualized multicore cluster. In [13], the performance of the compute-bound

benchmark applications was analyzed, and in [11], the performance overheads for network I/O device virtualization were measured. A simulation-driven approach was presented in [7], which analyzes the virtualization overheads of I/O intensive workloads. The performance impact of a consolidated workload, composed of server applications, was evaluated in [5]. In our paper, we focus on how the complex memory hierarchy affects the performance of HPC applications in virtualized systems.

A VM-aware MPI library was developed to reduce the communication overhead for HPC application in [8]. To improve I/O performance, Liao et al presented cache-aware scheduling which co-schedules Dom0 and I/O intensive DomUs to communicate more efficiently via a last level cache, and credit-stealing which steals credits for I/O intensive vCPUs [10].

6 Conclusion

In this paper, we evaluate single and dual socket multicore systems with the Xen hypervisor. For shared memory applications, NUMA awareness is critical for performance in dual-socket systems. As the complexity and non-uniformity in memory systems increase, NUMA and cache awareness in VM scheduling will become critical for them. For MPI-based applications, the NUMA effect is much smaller than that with the shared memory applications. However, the granularity of VMs (the number of vCPUs per VM) becomes critical for the overall performance.

References

1. Amazon EC2, `http://aws.amazon.com/ec2/`
2. MPICH, `http://www.mcs.anl.gov/research/projects/mpich2/`
3. The NAS Parallel Benchmarks, `http://www.nas.nasa.gov/Resources/Software/npb.html/`
4. The Princeton Application Repository for Shared-Memory Computers (PARSEC), `http://parsec.cs.princeton.edu/`
5. Apparao, P., Iyer, R., Zhang, X., Newell, D., Adelmeyer, T.: Characterization & analysis of a server consolidation benchmark. In: VEE 2008: Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, pp. 21–30. ACM, New York (2008)
6. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. In: SOSP 2003: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pp. 164–177. ACM, New York (2003)
7. Chadha, V., Illiikkal, R., Iyer, R., Moses, J., Newell, D., Figueiredo, R.J.: I/o processing in a virtualized platform: a simulation-driven approach. In: VEE 2007: Proceedings of the 3rd International Conference on Virtual Execution Environments, pp. 116–125. ACM, New York (2007)
8. Huang, W., Koop, M.J., Gao, Q., Panda, D.K.: Virtual machine aware communication libraries for high performance computing. In: SC 2007: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, pp. 1–12. ACM, New York (2007)

9. Huang, W., Liu, J., Abali, B., Panda, D.K.: A case for high performance computing with virtual machines. In: ICS 2006: Proceedings of the 20th Annual International Conference on Supercomputing, pp. 125–134. ACM, New York (2006)
10. Liao, G., Guo, D., Bhuyan, L., King, S.R.: Software techniques to improve virtualized i/o performance on multi-core systems. In: ANCS 2008: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, pp. 161–170. ACM, New York (2008)
11. Menon, A., Santos, J.R., Turner, Y., Janakiraman, G.J., Zwaenepoel, W.: Diagnosing performance overheads in the xen virtual machine environment. In: VEE 2005: Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments, pp. 13–23. ACM, New York (2005)
12. Ranadive, A., Kesavan, M., Gavrilovska, A., Schwan, K.: Performance implications of virtualizing multicore cluster machines. In: HPCVirt 2008: Proceedings of the 2nd Workshop on System-Level Virtualization for High Performance Computing, pp. 1–8. ACM, New York (2008)
13. Tikotekar, A., Vallée, G., Naughton, T., Ong, H., Engelmann, C., Scott, S.L.: An analysis of hpc benchmarks in virtual machine environments, pp. 63–71 (2009)