Performance Evaluation of OpenMP Applications on Virtualized Multicore Machines

Jie Tao[1], Karl Fürlinger[2], and Holger Marten[1]

[1]Steinbuch Center for Computing
Karlsruhe Institute of Technology, Germany
{jie.tao,holger.marten}@kit.edu
[2] Department of Computer Science
Ludwig-Maximilians-Universität (LMU) München, Germany
fuerling@nm.ifi.lmu.de

Abstract. Virtualization technology has been applied to a variety of areas including server consolidation, High Performance Computing, as well as Grid and Cloud computing. Due to the fact that applications do not run directly on the hardware of a host machine, virtualization generally causes a performance loss for both sequential and parallel applications.

This paper studies the performance issues of the OpenMP execution on virtualized multicore systems. The goal of this study is to quantify the performance deficit of virtualization of OpenMP applications and further to detect the reason of the performance loss. The results of the investigation are expected to guide the optimization of virtualization technologies as well as the applications.

1 Introduction

Virtualization is a technology that allows a physical host to run different operating systems on the same hardware. Besides OS customization, a virtualized machine has the advantage of performance isolation, fault tolerance, easy management, hardware independence, on-demand resource creation, and legacy software support. Due to these features, virtualization has been widely used to consolidate servers, provision on-demand resources in the computing Clouds, mitigate the problems of system scalability and management in High Performance Computing, and provide customized environments for various Virtual Organizations and users of computing Grids.

For server consolidation and Cloud computing the performance of a virtualized system is currently not a major concern because server consolidation uses the virtualization technology for running multiple isolated servers on a single hardware and Cloud computing relies on virtualization to provide on-demand, customized resources. Grid computing and HPC, however, aim at building computing infrastructures to efficiently run large scientific applications and performance is hence a key issue in these fields. Furthermore, Cloud users will increasingly care about performance in the future when the concept of HPC as a Service, as proposed by the OpenCirrus [22] project, comes into widespread use. For resource providers to optimize the underlying infrastructures and for users to fully exploit the computational capacity of the resources, it is necessary to understand the performance feature of applications on virtualized architectures.

B.M. Chapman et al. (Eds.): IWOMP 2011, LNCS 6665, pp. 138–150, 2011.

Computer architecture, on the other hand, is entering a new evolution era. The design of microprocessors is moving from single core to multicore (or manycore). The reasons for this trend are primarily related to the physical limits of semi-conductor based electronics and heat dissipation on the chip. With Moore's law intact for the near and midterm future, the solution is straightforward: spend the growing transistor budget on a number of simpler and more energy efficient processing cores. Over the last few years, almost every processor vendor developed multicore solutions. Processors with 64 cores are emerging. The increase of cores at an exponential rate (doubling every 18 to 24 months) means that hundreds of cores will be integrated on a single silicon die in several years. It is clear that multicore will be the single processing node that builds the future computing systems.

Available virtualization technologies, however, were not specifically developed for multicores. How to adapt the existing techniques to the next generation computer architectures must be an open question to the developers of virtualization techniques. For application developers the challenge is how to adapt the computational algorithms and the parallelization mechanisms to the virtualized environment in order to achieve the optimal runtime performance.

This work aims at giving these developers an initial view of the performance characteristics of a virtualized multicore machine. For this, we studied a set of OpenMP applications from standard benchmarks and compared the parallel performance on virtual machines with the performance of native executions. We then used a performance tool to analyze the runtime execution behavior. The results show the reasons of performance loss, the bottlenecks, and the potential optimization.

The remainder of the paper is organized as follows. Section 2 gives some background knowledge on virtualization and introduces related work in the area of performance analysis on virtual machines. Section 3 shows the performance of several OpenMP applications running on a virtualized 8-core machine. This is followed by the performance analysis based on available tools in Section 4. The paper concludes in Section 5 with a brief summary and several future directions.

2 Background and Related Work

This section describes the basic concept of virtualization and introduces similar research work in the area of performance evaluation on virtual machines.

2.1 The Virtualization Technology

System virtualization was pioneered on IBM mainframes [4] in the 1970s, with the goal of running different applications on the same hardware. The main motivation for virtualization in that time was to increase the utilization of expensive computing resources. In the 1990s, microcomputers were widely adopted to build client-server and peer-to-peer systems. These new computing environments brought with them several problems including security, increased administration complexity, and power consumption. As a solution, virtualization was applied and became thereafter a hot topic in this field.

Recently, Cloud computing deploys virtual machines to provide on-demand infrastructures, promoting virtualization to the second most important technology after multicore, according to a recent study [12].

In the 70s, virtualization was mainly used to run different applications on a single hardware. The 90s widened the application areas and saw virtualization in various scenarios where isolation, security, easy system management, and on-demand resources were required [9].

Today, virtualization is changing the way of computing and services. Virtualization is widely used for server consolidation [29]. In this use case, different servers, like Web, application, and database servers, run on the same physical hardware but with separate virtual machines. Each virtual machine supports the operating system and application environment of a single server. The servers run safely on the shared hardware and can be migrated transparently, increasing server utilization, reliability, and availability while reducing the overall number of physical systems and related recurring costs.

Virtualization can be used, and is even a simpler and better solution, in the HPC field for on-line maintenance, fault tolerance, performance isolation, productivity, security, reliability, availability, and more specifically for running applications with their customized operating systems [18]. Nevertheless, virtualization is currently not widely adopted by HPC users due to the performance loss.

Grid computing uses virtualization to achieve interoperability and interoperation between different grid infrastructures [25], to gain administrative flexibility [8], to protect sensitive resources [35], and also to benefit from the traditional advantages of the virtualization technology [25]. We have used virtual machines to build grid workflow systems, e-science infrastructures, and on-demand virtual environments for running legacy codes [32,34].

Virtualization is adopted in Cloud computing as a key technology. Existing cloud infrastructures including the Amazon EC2 [1], OpenNebula [26], Eucalyptus [21], Nimbus [16], and our Cumulus [33], all use the virtual machine technology to provide Infrastructure as a Service (IaaS).

Early approaches simply used binary transformation to run different code on an existing system. Comprehensive machine virtualization started in the 1990s with a virtualization layer between the hardware and the guest operating systems. This is called as Virtual Machine Monitor (VMM) or hypervisor [24]. The main task of a VMM is to virtualize the memory, the processor, and the devices including the network. Additionally, a VMM is responsible for resource allocation, Virtual Machine (VM) scheduling, and routing of I/O requests. CPU virtualization mainly deals with sensitive instructions of the OS, such as privileged instructions and exceptions, which cannot be executed directly because with a virtualization layer inserted the guest OS now runs at a lower privileged level. One solution is para-virtualization that deploys hypercalls to communicate with the hypervisor. The OS kernel has to be slightly modified to replace the sensitive instructions with hypercalls. The other approach is full virtualization which translates the sensitive instructions to a new sequence of instructions for the virtualized hardware without changing the guest OS. Memory virtualization aims at mapping the guest physical memory to the actual machine memory and concerns mainly the

virtualization of the translation lookaside buffer (TLB). Device I/O virtualization provides each virtual machine with a virtual device, which is either a simple device abstraction or an emulation of the real hardware.

Xen [2], VMware [30], and KVM [17] are three well known and widely used VMMs [5,28,20]. The Xen hypervisor is an open source development and is widely used for research purposes. KVM is also an open source product. It adds the virtualization capacities directly in the Linux kernel, achieving the thinnest hypervisor of only a few hundred thousand lines of code. Therefore, KVM is being increasingly deployed, even though it requires the virtualization extensions in the hardware of modern microprocessors (e.g. Intel VT and AMD-V). VMware is a commercial product and used mainly for server consolidation.

2.2 Related Work

Over the last years, researchers have evaluated the performance of running both sequential and parallel programs on virtual machines. For the former the users of the ATLAS experiments have measured up to a 14% runtime overhead of VMware ESX with compute-intensive simulation applications in High Energy Physics [13]. The Xen developers also measured the performance loss on three VMMs: Xen, VMware and User Mode Linux. The results showed that all VMMs introduced a significant slowdown with database and web applications. [2].

For parallel execution, performance evaluation has been conducted with MPI and MapReduce applications. Evangelinos and Hill [7] built a virtual cluster on top of the Amazon EC2 and used the cluster to test various MPI implementations including OpenMPI, GridMPI, LAM and MPICH-2. The results showed a quite poor latency and bandwidth. In all cases, the message latency is more than double of that measured on a physical, gigabit based cluster, and the asymptotic bandwidth is only a half. Similarly, Ekanayake and Fox measured the MPI runtimes on a virtualized multicore node of a private Cloud [6] and reported a slowdown of 10% to 40%. Another test on the Amazon EC2 [31] showed an even worse performance. Jackson et al. [15] analyzed HPC applications on Amazon EC2 and found a substantial slowdown compared to dedicated clusters and HPC systems depending on the communication characteristics.

Tikotekar et al. [27] used a virtualized cluster to analyze the performance of two HPC applications with the goal of observing the virtualization overhead. Based on a system profiler, performance data such as wall clock time, TLB miss, and cache miss were collected. The results show that the overall performance penalty does not differ much between the two applications, while the concrete overhead profiles of individual applications are not similar. Ranadive et al. [23] also studied the performance of HPC applications on virtualized clusters. The purpose of this study is to understand the overhead of supporting multiple VMs on a host machine as well as the inter and intra-VM communication patterns.

Ibrahim et al. [14] conducted several experiments to measure the performance of the Hadoop MapReduce implementation on VMs. Performance is measured using a physical cluster and a virtual cluster with different data size and cluster scale. The physical

cluster performs better in terms of data transfer from and to the file system Hadoop is based on. The performance gap increases considerably as the data size increases or the system scale is enlarged. For execution time, again the physical cluster works much better, where running an application on a cluster of VMs takes double of the time required by the physical cluster in the same scale.

Overall, the above experiments all reported that the performance on virtual machines is lower than the physical system. This is true for both sequential and message-passing parallel applications. Nevertheless, there exist experimental results showing that using single VMs to run multiple programs is in some cases better than the native execution [19]. How about multi-threading applications, like OpenMP?

The answer to this question is still not available because the OpenMP performance has not been evaluated on VMs. On the other hand, application developers of different domains, such as High Energy Physics (HEP), have seen the multicore trend and are investigating multi-threading solutions to accelerate their applications running on the Grid which is increasingly adopting virtual worker nodes. The test results of this paper can serve as the base of such solutions.

3 OpenMP Performance on Virtualized Multicore

The test environment is equipped with an 8-core AMD processor with hardware virtualization support. The hypervisor running on this machine is the open source Xen. For the experiments, we created two virtual machines, including a full virtualization VM called VM-full and a para-virtualization VM called VM-para. Some parameters of the VMs are depicted in Table 1.

Table 1. System configuration of the virtual machines for testing

hypervisor	Xen
Operating System	Debian 2.6.26
Compiler	gcc 4.3.2
#cores	1-8
memory	4 GB
CPU	Quad-Core AMD Opteron(tm) Processor 2376

The applications for the test are chosen from the SPEC and NAS OpenMP benchmark suites. The applications are compiled with the *gcc* compiler which supports OpenMP since version 4.2. The NAS applications are compiled with a data size of class A, while the SPEC applications are executed using the reference data.

Figure 1 depicts the test results with four SPEC applications. As expected, *applu*, *equake*, and *swim* run faster on the host machine than on the virtual machines. The same behavior can be observed with *wupwise*, except the case of running the application using only one core. For this experiment, the application runs slightly faster on the para-virtualized machine. This may be contributed by the memory and device virtualization.

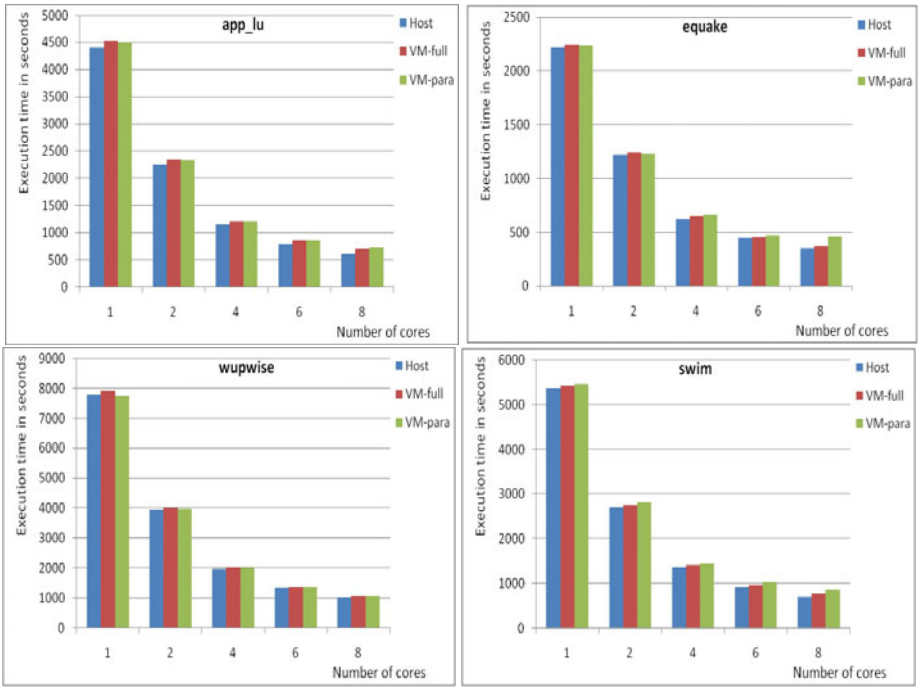

Fig. 1. Execution time of SPEC applications running on the host machine and virtual machines

Now we examine the execution behavior with the NAS applications. The experimental results are demonstrated in Figure 2. As shown in the figure, the performance with the NAS applications is interesting, where each application behaves differently.

BT shows the expected performance, where the execution on the host machine is faster in all cases. CG and LU perform similarly to the SPEC application *wupwise*, where the para-virtualized machine runs the application faster using one processor core. For EP both virtual machines perform better when the application runs sequentially or in parallel using 8 cores. For the case of 6-cores, the para-virtualized machine is faster. FT also shows the expected behavior with a better performance on the host. A specific feature of FT is that the execution on the para-virtualized machine with 8 cores is extremely slow, with a slowdown of 75% to the host machine, indicating a poor scalability of this application on the para-virtualized machine.

The most interesting application is SP. It can be seen that the performance on all virtual machines is very poor. However, the most interesting aspect is that the execution time increases with the number of cores. As depicted in the last diagram of the figure, it requires 2821 seconds to run SP on the fully virtualized machine using 8 cores. This is 28 times slower than the host run and a 368% slowdown to the case of sequential runs on the same virtual machine. This strange behavior leads us to apply performance tools to find the reasons. The results are described in the following section.

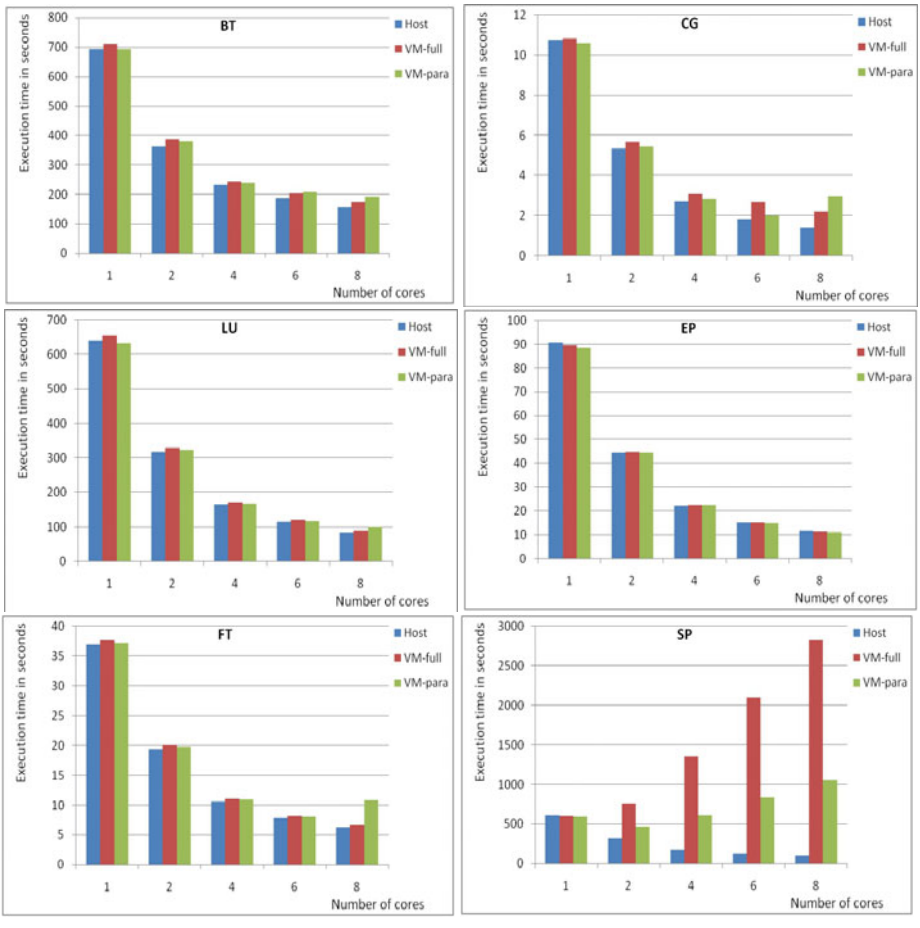

Fig. 2. Execution time of NAS applications running on the host machine and virtual machines

4 Performance Analysis Using ompP

The performance tool we used for the analysis work is ompP [11]. ompP is an OpenMP profiling tool that relies on source-code instrumentation. It delivers per-region and per-thread timing statistics at the end of the program run in a text-based profiling report. ompP also supports the measurement of hardware performance counters using PAPI, again on a per-region and per-thread basis. As an advanced feature, ompP produces an overhead analysis report which quantifies overheads into four categories (load imbalance, synchronization, limited parallelism, and thread management). By analyzing a program with an increasing number of threads one can determine impediments to the scalability of an application, for example, factoring out the contribution of load imbalance in particular worksharing region to the overall scalability problem.

Table 2. Runtime overhead of the NAS OpenMP applications

	Total	Overhead (%)	Synch	Imbal	Limpar	Mgmt
BT-host	1253.71	81.23 (6.48)	0.00	80.87	0.00	0.36
BT-full	1294.55	148.48 (11.47)	0.00	148.47	0.00	0.01
BT-para	1400.50	163.66 (11.65)	0.00	163.64	0.00	0.02
FT-host	72.27	25.62 (35.44)	0.01	1.06	24.43	0.12
FT-full	75.02	25.97 (34.53)	0.01	1.04	24.85	0.07
FT-para	88.67	32.22 (36.34)	0.00	6.45	25.73	0.04
CG-host	14.36	1.55 (8.95)	0.00	0.95	0.19	0.41
CG-full	17.64	4.87 (23.59)	0.00	3.46	1.37	0.04
CG-para	24.05	6.37 (26.49)	0.00	5.27	1.08	0.02
EP-host	92.27	1.08 (1.17)	0.00	0.93	0.00	0.15
EP-full	89.66	1.24 (1.37)	0.00	0.75	0.00	0.49
EP-para	133.76	29.60 (22.13)	0.00	29.32	0.00	0.27
MG-host	27.99	2.11 (7.54)	0.01	1.96	0.07	0.06
MG-full	28.67	2.47 (8.67)	0.00	2.34	0.07	0.06
MG-para	28.40	2.27 (7.99)	0.00	2.23	0.02	0.02
SP-host	4994.76	1652.66 (33.03)	0.11	1651.95	0.00	0.60
SP-full	16466.47	14315.84 (86.89)	1.45	14314.36	0.00	0.03
SP-para	6816.17	5302.04 (77.68)	2.74	5299.29	0.00	0.01

Using ompP we measured the overhead of parallelization and the execution time of individual code regions. We compare the data of SP with other NAS applications in order to understand why SP runs slower with increasing number of processor cores.

Table 2 shows the results delivered by ompP when running the applications with all 8 cores. The table contains seven columns. The first column denotes the applications and the virtualization settings. For each application we collected the performance data on the physical machine and the two virtual machines. The second column is the total execution time which is the sum of the time spent by each thread [1]. The third column shows the measured overhead which is categorized in the following columns into:

Synchronization (Synch): Overheads that arise because threads need to coordinate their activity. An example is the waiting time to enter a critical section or to acquire a lock.

Imbalance (Imbal): Overhead due to different amounts of work performed by threads and subsequent idle waiting time, for example, in work-sharing regions.

Limited Parallelism (Limpar): Overhead resulting from unparallelized or only partly parallelized regions of code. An example is the idle waiting time threads experience while one thread executes a `single` construct.

Thread Management (Mgmt): Time spent by the runtime system for managing the application's threads. That is, time for creation and destruction of threads in parallel regions and overhead incurred in critical sections and locks for signaling the lock or critical section as available.

[1] Note that the total execution time depicted in Table 2 is not the runtime of an application as shown in Figure 2. It is the total time of eight threads. The data shown in the figure and the table were measured in different runs with varied CPU loads. Hence, they may not be identical.

Observing the third column, it is evident that most of the programs suffer from significantly larger overheads on virtual machines than on the physical host. For SP on fully virtualized machine, for example, the overhead is as high as 86.89%. The table shows that the overhead is mainly caused by imbalance. This means that more than 80% of the processor time is not used for calculating SP, but wasted in waiting for other threads.

Table 3. Region overview of the NAS applications

	PARALLEL	LOOP	SINGLE	BARRIER	CRITICAL	MASTER
BT	2	54	0	0	0	2
FT	2	6	5	1	1	1
CG	2	22	12	0	0	2
EP	1	1	0	0	1	1
MG	5	10	4	1	1	1
SP	2	69	0	3	0	2

For a deeper insight into this issue we examined the ompP region overview, which shows the statistics for different OpenMP regions. As depicted in Table 3, SP differs from other applications primarily in the number of LOOPs. Nevertheless, BT also shows a large number in this category. Why does BT achieve speedup but SP not? In order to get the answer, we examine further the individual LOOPs of both programs.

Table 4. ompP report of a LOOP in sp.c (line 898-906) on the para-virtualized machine

TID	execT	execC	bodyT	exitBarT
0	310.60	1541444	11.24	289.41
1	310.50	1541444	11.22	289.35
2	310.44	1541444	11.33	289.12
3	310.26	1541444	11.22	289.14
4	310.85	1541444	11.26	289.68
5	310.82	1541444	11.24	289.62
6	311.10	1541444	11.17	289.99
7	311.14	1541444	10.92	290.48
SUM	2485.71	12331552	89.60	2316.76

Table 4 shows the performance data with a sample parallel LOOP in *sp.c*. The data shows the overall execution time each thread requires to execute the LOOP (execT), the number of each thread entering the LOOP (execC), the times needed for the LOOP body (bodyT), and the time for exiting the implicit BARRIER at the end of the LOOP (exitBarT).

It can be seen that each thread needs more than 310 seconds for executing this LOOP. The time actually used for the LOOP body is only around 11 seconds, which is shown in the fourth column of the table. The remaining time is contributed by the implicit exit BARRIER. There are several LOOPs in SP which behave similarly to the example shown in Table 4. These LOOPs are the cause of high execution overhead of this application.

As mentioned above, BT also contains a large number of LOOPs. However, these code regions (except for two) have a loop execution count of at most 201. Therefore, the overhead is comparatively small and does not limit scaling. With SP, nevertheless, each thread enters the sample LOOP more than 1.5 million times. While after each entering an implicit BARRIER is performed, the overhead is clearly considerable.

Generally, the overhead in a worksharing region, like a LOOP, is caused by the inconsistent execution time of each thread, where some threads complete their work earlier and have to wait for other threads. This situation can be seen with the sample LOOP, depicted in the body execution time – column 4 of Table 4. However, the data measured on the physical machine show such imbalanced scenarios as well, and the situation on the physical machine is even worse. In addition, the fourth column of Table 3 shows a difference of less than 1 second in the execution time of all threads. This indicates that the high overhead is not caused by the threads waiting for other threads. There must be another reason.

The conclusion leads us to study the implementation of BARRIER in *gcc*. The GNU OpenMP implementation [10] uses a function, called *GOMP_barrier* to handle all issues concerned the BARRIER construct of OpenMP with the concrete work done with the subroutine *gomp_barrier_wait*. GNU uses a common approach to achieve the thread synchronization, with a counter combined with a BARRIER. The counter was initialized with the number of the parallel threads that work together for a sharing region. Each thread decreases the counter by one when arriving the BARRIER and then is blocked till the counter value is zero. GNU applies the semaphore synchronization mechanism to grant mutual exclusion with the counter.

The high overhead on the virtual machines may be caused by the decrement of the counter, which is an access to a shared variable in combination with LOCK and UNLOCK operations. The virtualization of the memory on a virtualized multicore can change the access behavior to shared variables. However, for an exact understanding of the problem it is needed to further trace the thread actions. Unfortunately, ompP currently does not provide such information.

Based on the above observation, we optimized the SP source code. The optimization was simply performed with the LOOPs that behave similarly as the sample LOOP. The approach was to move the parallelization from the inner iteration to the outer one, in order to reduce the number of BARRIERs. Surprisingly, this initial optimization achieved a significant performance gain. For example, the execution time on the full virtualized machine with 8 cores reduced from 2821 seconds to 163 seconds. This means that the optimized version of SP runs 17 times faster than the original implementation.

To observe the performance difference of physical and virtual machines in a more systematic manner, we measured the overheads of OpenMP constructs with the OpenMP micro-benchmarks [3].

Figure 3 shows the results with a synchronization construct, REDUCTION in this case, and two loop scheduling schemes: STATIC and DYNAMIC. The left diagram of Figure 3 depicts that the overhead caused by REDUCTION operations goes up linearly with the number of cores on a virtual machine, while the physical machine shows only a slight increase. In addition, the overheads on virtual machines are much larger than the host machine. This kind of behavior is also demonstrated with other

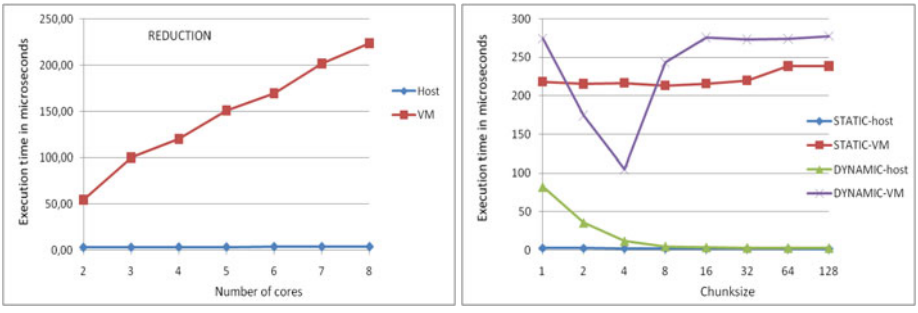

Fig. 3. OpenMP construct overheads on the host machine and the full-virtualized machine

synchronization constructs, including PARALLEL, FOR, PARALLEL FOR, BARRIER, and SINGLE. Two constructs for mutual exclusion synchronization, CRITICAL and LOCK/UNLOCK, behave similarly with a linear overhead increase to the number of cores on virtual machines but slight increase to the physical machine. ATOMIC, however, shows similar values on virtual machines to the host machine.

The behavior of scheduling schemes is more interesting. The data shown on the right side of Figure 3 were measured with eight cores. It can be seen that on the host machine the STATIC scheme works similarly with different chunk sizes while DYNAMIC introduces much less overheads with increasing chunk sizes up to 8. On virtual machines STATIC tends to cause more overheads as the chunk size increases while DYNAMIC shows first a drastic decrease in overheads, then a drastic increase, and finally a slight increase.

Overall, all OpenMP constructs, besides BARRIER, introduce more overheads on virtual machines. Since they are not called so often like BARRIER in the tested applications, their influence on performance is not visible. However, they remain the reason for performance deficit.

5 Conclusion

With increasing adoption of virtual machines in the field of High Performance Computing, benchmarking virtual machines becomes an interesting topic. This work evaluates the OpenMP execution on virtualized multicore machines. The experiments are performed with the SPEC and NAS OpenMP benchmark suites. Besides the expected results, we found a strange behavior with the application SP, where no speedup has been seen when running the program with several cores. We analyzed the execution behavior of this application using the performance tool ompP and detected that the reason for this poor performance lies in the fact that shared regions introduce a high overhead for threads passing the implicit BARRIER. We then made an initial optimization with the result of a considerable performance gain.

In the next step, we will perform experiments with other OpenMP implementations and hypervisors to see whether the described problem is general, and to find more performance bottlenecks on virtualized machines as well. In addition, we will extend ompP

or other performance tools, e.g. system profiling tools, to follow the actions a thread performs after entering and before leaving the BARRIER. Our goal is to detect the concrete reason in order to design common optimization methodologies for OpenMP compilers and for the virtualization techniques.

Acknowledgements

This work was partially supported by the EU project MADAME: Multicore Application Development and Modeling Environment.

References

1. Amazon Web Services. Amazon Elastic Compute Cloud (Amazon EC2), http://aws.amazon.com/ec2/
2. Barham, P., Dragovic, B., Fraser, K.: Xen and the Art of Virtualization. In: Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, pp. 164–144 (2003)
3. Bull, J.M.: Measuring Synchronisation and Scheduling Overheads in OpenMP. In: Proceedings of the First European Workshop on OpenMP, pp. 99–105 (1999)
4. Creasy, R.J.: The Origin of the VM/370 Time-Sharing Systems. IBM Journal of Research and Development, 483–490 (September 1981)
5. Dike, J.: User Mode Linux. Prentice Hall, Englewood Cliffs (2006)
6. Ekanayake, J., Fox, G.: High Performance Parallel Computing with Clouds and Cloud Technologies. In: Proceedings of the first International Conference on Cloud Computing (October 2009)
7. Evangelinos, C., Hill, C.N.: Cloud Computing for parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. In: Proceedings of CCA 2008 (2008)
8. Figueiredo, R., Dinda, P., Fortes, J.: Case for Grid Computing on Virtual Machines. In: Proceedings of the 23rd International Conference on Distributed Computing, pp. 550–559 (May 2003)
9. Figueiredo, R., Dinda, P.A., Fortes, J.: Resource Virtualization Renaissance. Computer 38(5), 28–31 (2005)
10. Free Software Foundation. The GNU OpenMP Implementation, http://gcc.gnu.org/onlinedocs/gcc-4.2.4/libgomp/
11. Fürlinger, K., Gerndt, M.: ompP: A profiling tool for openMP. In: Mueller, M.S., Chapman, B.M., de Supinski, B.R., Malony, A.D., Voss, M. (eds.) IWOMP 2005 and IWOMP 2006. LNCS, vol. 4315, pp. 15–23. Springer, Heidelberg (2008)
12. Gartner Inc. Gartner Identifies Top Ten Disruptive Technologies for (2008 to 2012), http://www.gartner.com/it/page.jsp?id=681107
13. Gilbert, L., Tseng, J., Newman, R.: Performance Implications of Virtualization and Hyper-Threading on High Energy Physics Applications in a Grid Environment. In: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (April 2005)
14. Ibrahim, S., Jin, H., Lu, L.: Evaluating MapReduce on Virtual Machines: The Hadoop Case. In: Proceedings of the First International Conference on Cloud Computing, pp. 519–528 (December 2009)
15. Jackson, K.R., Ramakrishnan, L., Muriki, K., Canon, S., Cholia, S., Shalf, J., Wasserman, H.J., Wright, N.J.: Performance analysis of high performance computing applications on the amazon web services cloud. In: Proceedings of the 2nd International Conference on Cloud Computing Technology and Science, CloudCom 2010 (2010)

16. Keahey, K., Freeman, T.: Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. In: Proceedings of the First Workshop on Cloud Computing and its Applications (October 2008)
17. KVM. Kernel Based Virtual Machine, http://www.linux-kvm.org/
18. Mergen, M.F., Uhlig, V., Krieger, O., Xenidis, J.: Virtualization for High-performance Computing. ACM SIGOPS Operating Systems Review 40(2), 8–11 (2006)
19. Michelotto, M., Alef, M., Iribarren, A.: A Comparison of HEP Code with SPEC Benchmark on Multicore Worker Nodes. In: Proceedings of the 17th International Conference on Computing in High Energy and Nuclear Physics (March 2009)
20. Microsoft. Hyper-V Server, http://www.microsoft.com/hyper-v-server/en/us/default.aspx
21. Nurmi, D., Wolski, R., Grzegorczyk, C.: The Eucalyptus Open-source Cloud Computing System. In: Proceedings of CCA 2008 (2008)
22. Open Cirrus TM: The HP/Intel/Yahoo! Open Cloud Computing Research Testbed, https://opencirrus.org/
23. Ranadive, A., Kesavan, M., Gavrilovska, A., Schwan, K.: Performance Implications of Virtualizing Multicore Cluster Machines. In: Proceedings of the 2nd Workshop on System-level Virtualization for High Performance Computing, pp. 1–8 (2008)
24. Rosenblum, M., Garfinkel, T.: Virtual Machine Monitors: Current Technology and Future Trends. Computer 38(5), 39–47 (2005)
25. Ruda, M., Denemark, J., Matyska, L.: Scheduling Virtual Grids: The Magrathea System. In: Proceedings of the 3rd International Workshop on Virtualization Technology in Distributed computing (November 2007)
26. Sotomayor, B., Montero, R., Llorente, I., Foster, I.: Capacity Leasing in Cloud Systems using the OpenNebula Engine. In: Proceedings of the First Workshop on Cloud Computing and its Applications (October 2008)
27. Tikotekar, A., Vallee, G., Naughton, T.: An Analysis of HPC Benchmarks in Virtual Machine Environments. In: Proceedings of Euro-Par 2008 Workshops - Parallel Processing. LNCS, vol. 5415, pp. 63–71 (2008)
28. VirtualBox.org. VirtualBox, http://www.virtualbox.org/
29. VMware. Server Consolidation, http://www.vmware.com/solutions/consolidation/
30. VMware Inc. VMware, http://www.vmware.com
31. Walker, E.: Benchmarking Amazon EC2 for High-Performance Scientific Computing. The USENIX Magazine 33(5) (October 2008)
32. Wang, L., Kunze, M., Tao, J.: Performance Evaluation of Virtual Machine-based Grid Workflow System. Concurrency and Computation: Practice & Experience (4), 1759–1771 (2008)
33. Wang, L., Tao, J., Kunze, M.: Scientific Cloud Computing: Early Definition and Experience. In: Proceedings of the 2008 International Conference on High Performance Computing and Communications, pp. 825–830 (September 2008)
34. Wang, L., von Laszewski, G., Kunze, M., Tao, J.: Grid Virtualization Engine: Design, Implementation and Evaluation. IEEE Systems Journal 3(4), 477–488 (2009)
35. Zhao, X., Borders, K., Prakash, A.: Using A Virtual Machine to Protect Sensitive Grid Resources: Research Articles. Concurrency and Computation: Practice & Experience 19(4), 1917–1935 (2007)