2009 Sixth IFIP International Conference on Network and Parallel Computing

Performance Evaluation of Parallel Programming in Virtual Machine Environment

Cong Xu, Yuebin Bai, Cheng Luo
School of Computer Science
Beihang University, Beijing 100191, P.R.China
xucong2007@cse.buaa.edu.cn

Abstract

As multi-core processors become increasingly mainstream, architects have likewise become more interested in how best to make use of the computing capacity of the CPU, for instance, through multiple simultaneous threads or processes of execution with OpenMP or MPI. At the same time, the increasingly mature and prevailing virtualization technique in server consolidation and HPC promotes the emergence of a large number of virtual SMP servers. Therefore, whether the parallel program can run in the virtual machine environment efficiently or not is a topic of concern.

In this paper, we investigate the performance of three typical parallel programming paradigms, including OpenMP, MPI, and Hybrid of OpenMP and MPI in the popular, open-source, Xen virtualization system. The results show that the performance of the traditional parallel program in Xen VMs is close to it in native, non-virtualized environment, if there is little communication or synchronization between threads or processes. In most cases, without excessive IO access, we can get an ideal speedup in a SMP VM or virtual cluster, which is close to linearity when the total virtual CPUs (vCPUs) number is not larger than the number of Physical CPUs (pCPUs). And the pure MPI implementation shows the best scalability and stability in virtual machine environment compared with the other two paradigms.

1. Introduction

In the computing space, virtualization is defined as an environment in which multiple Operating Systems (OS) run on a single physical machine. Each OS runs in its own partition, or Virtual Machine (VM). This is implemented by inserting an additional software layer between the hardware and the OS, called the Virtual Machine Monitor (VMM). The VMM schedules the guest OSs and manages the hardware resources in much the same way that an OS manages the execution of applications.

Nowadays, virtualization technologies like Xen hypervisor [3], VMWare's ESX server [9] and KVM [1] are becoming a prevalent solution for resource consolidation, power reduction, and to deal with bursty application behaviors. Amazon's Elastic Compute Cloud (EC2) [2], for instance, uses virtualization to offer datacenter resources (e.g., clusters or blade servers) to applications run by different customers, safely providing different kinds of services to diverse codes running on the same underlying hardware (e.g., trading systems jointly with software used for financial analysis and forecasting). Virtualization has also shown to be an effective vehicle for dealing with machine failures, to improve application portability, and to help debug complex application codes, even to build a virtual cluster for HPC [4, 6, 7, and 8].

Both in business consolidation and HPC clusters, with the popularity of multi-core processors, parallel programming seems more and more significant. Developers began to change the code structure into what can be parallel executed, to meet the change of processors' structure. So, today, the concurrency performance is another crucial indicator for Virtual Machine. We found that, with some special configuration, virtual Machine (VM) can not only utilize thread-level parallelism through OpenMP or POSIX, but also implement the inter-VM parallelism through MPI or PVM. We even can deploy a hybrid parallel programming paradigm (combined with OpenMP and MPI) on a virtual SMP cluster in one multi-core processor server with Virtual Machine Monitor (VMM). However, very few HPC and large-scale parallel applications are currently running in a virtualized environment due to the performance overhead of virtualization. This paper aims at

978-0-7695-3837-2/09 $26.00 © 2009 IEEE
DOI 10.1109/NPC.2009.22

evaluating the performance of different parallel application in virtual machine environment, mainly in one node with a multi-core processor, not the performance of HPC in the cluster. In another words, I hope to find the overhead of VM in parallel programming in multi-core platform. For example, usually the total number of virtual CPUs (vCPUs) in the virtualized system is larger than the number of physical CPUs (pCPUs), and the schedule module in the VMM maps vCPUs of virtual machines into pCPUs in a time-share manner. So when the workload in the VM is the concurrent application such as multithreaded programs or parallel programs with the synchronization operation or much communication, these existing vCPU scheduling methods may deteriorate the performance.

This paper contributes experimental insights and measurements to better understand the effects of resource sharing on the performance of parallel programming. Specifically, for one or multiple virtual machines running on a multi-core platform, we evaluated the performance of three mainstream parallel programming paradigms. Stated more precisely, using a standard x86-based quad-core computer and the Xen hypervisor, we evaluate the efficiency of OpenMP, MPI and the hybrid paradigms respectively. The purpose is (1) to understand the performance implication and overheads of supporting multiple VMs on virtualized multi-core platforms; (2) to explore the performance implication of different parallelism programming paradigm in virtual machine environment.

The rest of the paper is organized as follows. In section 2 we illustrate the architecture of Xen VMM and analyze its problem for parallel programming. In Section 3 we describe our testing environment and methodology. Section 4 discusses the experimental results gathered from various benchmark. Section 5 touches upon related work. Finally, in Section 6 we summarize and conclude.

2. Problem and Analysis of Parallel programming in VM

In this section, we firstly describe the general virtual machine architecture of Xen VMM, and then give a typical vCPU scheduling scenario to describe the vCPU scheduling problem for some parallel programs in the VM envrionment.

2.1 Overview of the Xen Virtual Machine Monitor

Xen is a popular high performance VMM originally developed at the University of Cambridge. It uses paravirtualization [11] (can also support HVM with hardware virtualization such as Intel-VT [12] or AMD-V [13]), which requires that the host operating systems be explicitly ported to the Xen architecture, but brings higher performance. However, Xen does not require changes to the application binary interface (ABI), so existing user applications can run without any modification.

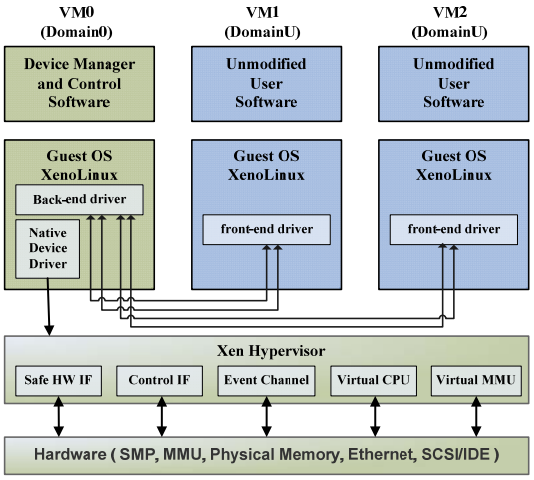

Figure 1. The structure of the Xen hypervisor, hosting three xenoLinux operating systems

Figure 1 illustrates the structure of a physical machine running Xen. The Xen hypervisor (the VMM) is at the lowest level and has direct access to the hardware. The hypervisor is running in the most privileged processor level. Above the hypervisor are the Xen domains (VMs). Guest OSes running in guest domains (User Domain or DomainU, DomU) are prevented from directly executing privileged processor instructions. A special domain called Domain0 (or Dom0), which is created at boot time, is allowed to access the control interface provided by the hypervisor and performs the tasks to create, terminate or migrate other guest domains through the control interfaces.

In Xen, domains communicate with each other through shared pages and event channels, which provide an asynchronous notification mechanism between domains. A "send" operation on one side of the event channel will cause an event to be received by the destination domain, which may in turn cause an interrupt. If a domain wants to send data to another, the typical scheme is for a source domain to grant access to local memory pages to the destination domain and then send a notification event. Then, these shared pages are used to transfer data.

Due to the special I/O mechanism of Xen, as the increase of total communication size in Xen, the burden in domain 0 is more and more heavy. This is one of the main bottlenecks of Xen for parallel programs, especially for the communication-sensitive programs.

2.2 Problem Scenario for parallel program

There are four processors in a SMP virtual machine, on which a thread of a multithreaded program is running respectively, and the unit time of CPU scheduling is a slot. The weight of the VM is 3/10, and there is a synchronization operation between threads at the end of each step, and the length of the step is equal to the length of the slot. This scenario can be abstracted from the multithreaded program or the parallel program.

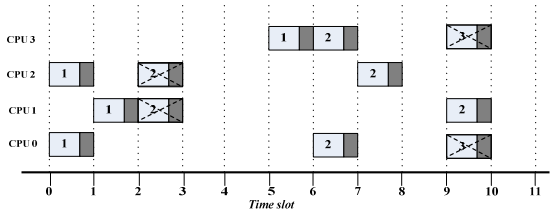

Figure 2. The scenario of Credit schedule in VM

There exist three kinds of scheduling strategies in Xen, BVT, SEDF and Credit [10]. The last one is the default scheduler in Xen. For Credit scheduler, it is that each virtual CPU is asynchronously assigned to the physical CPU in order to maximize the throughput, while guaranteeing the CPU fairness according to the weight. However, this strategy will deteriorate the performance when the workload is a concurrent application. A possible scheduling sequence of the multithreaded program by this scheduling strategy is shown as Figure 2, the multithreaded application only completes the 2 steps in the length of the 10 slots, while there are 4 slots of CPU time to be wasted for the synchronization. And the more synchronization, the more performance loss for the parallel programs.

3. Testing Environment and Methodology

Our experimental hardware platform is a Dell OPTIPLEX 755 server, with an Intel quad-core processor at 2.4GHZ. Each core has 32KB private data and instruction L1 cache; every two core shared 4MB L2 cache. The server has 4GB of RAM and a 250 GB SCSI hard disk with DMA enabled.

We perform our experiments by repeatedly executing the benchmarks and collecting the performance data. We use K-best measurement schema proposed by Randal E. Bryant and David R. O' Hallaron to collect the result from benchmarks, with K = 10, ε = 2%, M = 100. More information about the K-best measurement method and the formulation we use can be found in [5].

3.1. Host OS and Guest OS

In my experiment, the server is running the OpenSUSE-11.0 (paravirtualized 2.6.25 SMP kernel) in Dom0 with the Xen 3.2 hypervisor. We use the credit scheduler which is set default in Xen3.0 or latter version to schedule the VCPUs of VM. The guest OS in DomU are also OpenSUSE-11.0 (2.6.25 SMP kernel) with all unnecessary services removed. Each DomU is allocated 2 VCPU, 256MB of RAM, 8GB Disk and use bridge network interface to interconnect with Dom0 and other DomUs.

3.2 Benchmark

We overview the benchmarks that we use in this empirical investigation in Table 1.The benchmarks set consists of the widely used complicated and simple parallel applications. We employ the same benchmark binaries for all operating system configurations.

Benchmark Category	Code Name	Problem Size	What it measures?
micro-benchmark	Stream		Memory bandwidths
macro-benchmark	LU in NPB3.3-OMP	Class A	Total time (s) and millions of operations per second (Mop)
	BT in NPB3.3-OMP	Class A	
	LU-MZ in NPB3.3-MZ	Class A	
	BT-MZ in NPB3.3-MZ	Class A	
	BT in NPB3.3-MPI	Class A	
	LU in NPB3.3-MPI	Class A	
	CPI	2^9	

Table 1. Benchmark Overview

The STREAM benchmark [18] is a simple synthetic benchmark program that measures sustainable memory bandwidth and the corresponding computation rate for simple vector kernels.

We use a subset of the NAS Parallel Benchmarks as macro-benchmark for testing. NPB consists of a

variety of tests which can be run on a number of datasets of different sizes. There are 6 classes of problem size, S (used only for testing), W, and A-D, which are strictly increasing in size. We chose to use problem class A for our tests. Hereon, we only make a brief introduction of the subset of NPB that we used. More detailed descriptions of the various algorithms used in NPB are given in [14].

BT: Block Tri-diagonal. It is a solution of block tri-diagonal equations with a (5 x 5) block size, which mainly aims to test the balance of the computation with non-continuous point-to-point long memory access message-oriented communications, but not sensitive to communication delay very much.

LU: Lower-Upper Gauss-Seidel. This is a sample application, designed to test a wider variety of hardware features than the above benchmarks. Its communication message size is only about 40 Bytes.

What's more, we use another simple testing program CPI from Argonne National Laboratory, using Regular Polygon Approximation to compute the value of Pi, with MPI version respectively, intending to test the overhead of virtual machine on parallel programs. It has little communication between different processes.

4. Benchmark results and analysis

In this section we analyze the result collected from different benchmark application, explore the overhead of virtual machine environment, and evaluate the parallelism performance and the efficiency of OpenMP, MPI and Hybrid of them in Xen VM environment.

4.1. Simple parallel program

Firstly, we compare the performance of a simple testing parallel program, written in C language, from Argonne National Laboratory in the virtual machine with it in the native computer. This test aims at evaluating the performance of parallel program with little communication and synchronization between threads or processes.

The tests are conducted in one or multiple VMs (DomU) with the same configuration. The compiler is gcc4.3, and MPI environment is MPICH2, a widely used MPI implementation. In order to compare the thread parallelism in DomU with native machine, we change the OpenMP thread number through setting the OMP_NUM_THREADS environment variable in native machine. But in DomU we only change the vCPU number without setting OMP_NUM_THREADS environment variable, making the OpenMP thread number equal to the vCPU number. And the vCPU number is not limited by the pCPU number in native machine.

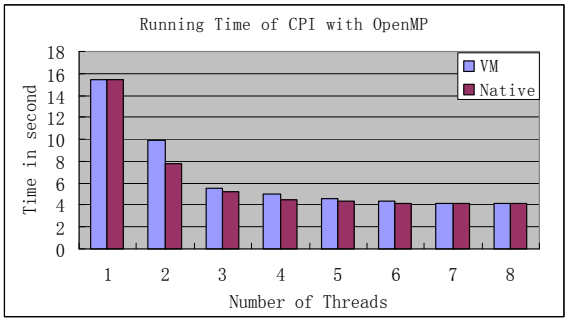

Figure 3. Running time of CPI with OpenMP

The graph in Figure 3 shows that the efficiency of OpenMP nearly equals both in virtual machine and native machine. OpenMP can still take the advantages of SMP multi-processor, no matter whether it is virtual SMP system or not, using multi-threaded to improve the computing speed of parallel program.

We use the Oprofile, a system-wide profiler tool for Linux system, to analyze the CPI code. We find that the proportion of the program which must be serial is less than 0.01%. So, according to Amdahl's law as following:

$$Speedup = \frac{1}{S + (1-S)/P} \quad , S < 0.0001, P = 4$$

The most high speedup value we can get is 4 if not consider about other overhead in system.

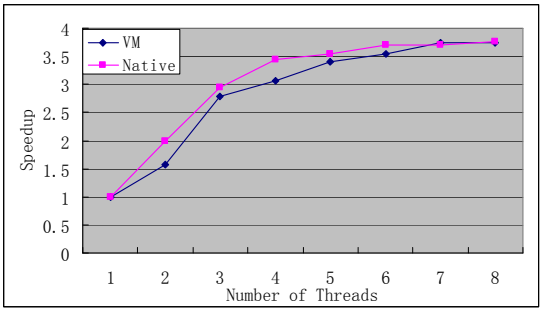

Figure 4. Speedup of OpenMP

The Figure 4 shows that CPI program using OpenMP get an ideal speedup value in DomU. The speedup value in DomU is slightly smaller than in native machine. Even the speedup value is still increasing when the thread number exceeds the number of pCPU of native machine, although the specification of Xen tells that it will cause significant performance decline with more VCPU than PCPU. In my experiment the speedup value finally reaches the

upper limit near to 3.7 when the thread number is equal to 7 and 8. After all, only 4 threads can truly concurrently execute in native machine.

We also evaluate the CPI with Message Passing Library (MPI) implementation. The MPI environment we use is MPICH2, a portable implementation of MPI from Argonne National Laboratory. We respectively make the test in single VM, in a virtual SMP cluster with virtual nodes in one physical node communicating through ssh with each other, and in native. We firstly present the results from the testing in one VM, a virtual cluster with 4 VM nodes, and native machine. In the tested VM, we change the VCPU number from 1 to 8, and set the MPI process number to be equal to the VCPU number every time.

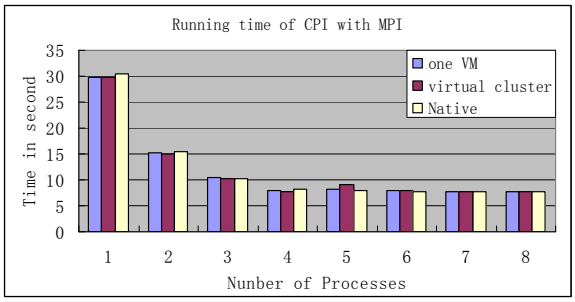

Figure 5. Running Time of CPI with MPICH2 in one VM, Virtual Cluster with 4 virtual nodes and Native Machine (in the case that running CPI in one VM, MPI process number = vCPU number)

Figure 5 shows that the running time of the program in single VM, virtual cluster, and native machine have almost no difference. And when the Process number is higher than 4, that is the PCPU number, the program's running time has not change any more in all three type of environment. But the final running time of lower limit is apparently lower than it without parallel executing.

Similar to the CPI program with OpenMP implementation, the proportion of this code can not be made parallel is less than 0.01% as well. The ideal speedup value for CPI with MPI is near 4. Figure 6 shows that the running effect in VM is very efficient which is close to the effect in native machine and its upper limit value is also close to 4. What's more, from the graph, we can know that using MPI can also get an almost linear speedup when the process number is less than the PCPU number in VM like in native machine.

To sum up, virtual machine environment only take a little impact on the simple computing-sensitive parallel application with little communication between threads or processes. This is mainly due to the paravirtualization technology adopted in Xen, which is

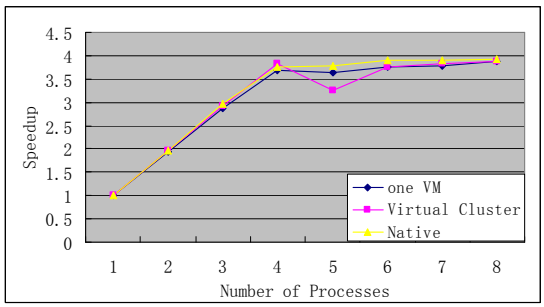

Figure 6 Speedup of MPI
(MPI process number = vcpu number of the VM, when running CPI in one VM)

detailed in [11]. With paravirtualization, running parallel program on much more vCPU than pCPU in single or multiple VMs, is effective without the noticeable performance degradation, despite the fact that vCPU needs some extra overhead to map to physical cups when executing a thread or process. However, in concurrent program (such as scientific computing, etc.), several threads (or processes) not only execute parallel, but also need for mutual synchronization. So, in some case, the asynchronous vCPU scheduling, used in most VMM, will result in performance decrease for concurrent programs [20].

4.2. Complex parallel program

In this section, we explore the performance of three typical parallel programming paradigms through running a set of benchmarks from NASA Parallel Benchmark (NPB), which are more complex than CPI and with more communication and synchronization.

Firstly, we evaluate the performance of OpenMP implementation in NBP. We run the two application, BT and LU with OpenMP implementation, respectively in native and VM, and we set the environment variable OMP_NUM_THREADS which decide the active threads number in the application from 1 to 7.

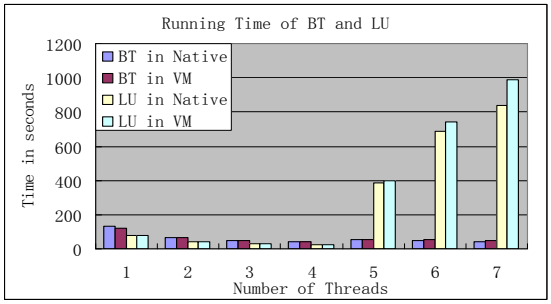

Figure 7 OpenMP test with NPB-OMP

From Figure 7 and Figure 8 we can see that, the running time of BT in VM is approximate to in native, no matter whether the number of active thread is less than the pCPU or not. And the performance is not decrease very much when the thread number is higher than pCPU. However, the performance of LU decrease dramatically when the active thread number is over 4, the pCPU number. And the performance in VM decreases a little faster than in native. The reason for that is LU use block communication which will cause more synchronization between threads or processes. If the active thread number is larger than pCPU number, it will generate a large of block-waiting for scheduling. And the vCPU of VM is scheduled by VMM like the thread scheduled by OS. Like shown in Figure 9, the threads in VM need two mappings to get the time slice of pCPU in native to execute, and this will cause more overhead. What's more, the Credit scheduler will also deteriorate the performance, because with the Credit scheduler the vCPUs can not be mapped to pCPUs simultaneously. So if the total vCPU is more than the pCPU, it may extend the block-waiting time of the threads or processes in parallel programs.

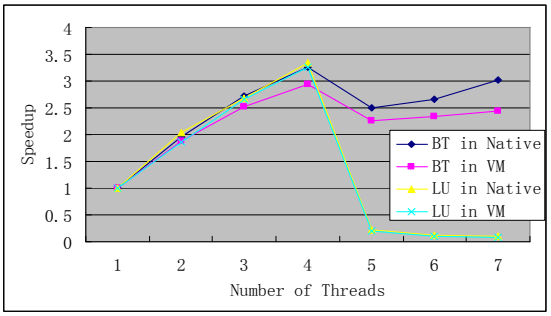

Figure 8 Speedup for OpenMP paradigm in NPB-OMP

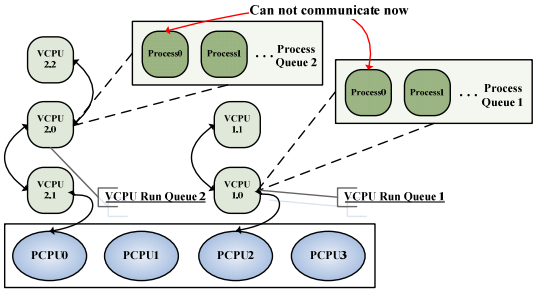

Figure 9 VCPU Schedule

All in all, the performance of OpenMP in VM is near the performance in native. Especially, it can get a linear speedup in VM with OpenMP, when the vCPUs are less than pCPUs. But it performs not very well for some applications sensitive to communication delay, as the vCPU number increasing up to over PCPU number.

Then, we evaluate the performance of MPI and Hybrid paradigm with NPB. We create 4 VMs with SMP in the server and use them to form a virtual cluster. We run BT-MZ and LU-MZ with MPI and Hybrid of MPI and OpenMP respectively in the virtual cluster. For MPI test, we set the process number to 1, 2, 4, 8, and 16 respectively, due to the request of NPB and the limit of my environment. For Hybrid paradigm, we set the process number as the MPI test, and set the OMP_NUM_THREADS environment variable to 2 in every VM node, enabling each process to be executed with 2 threads in every SMP node.

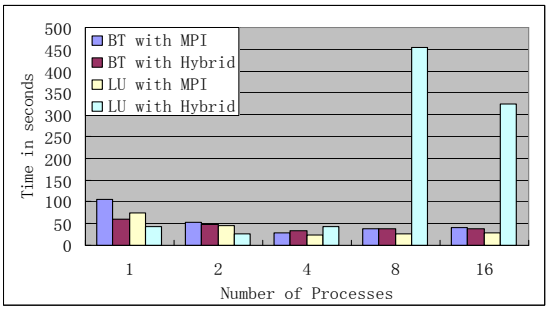

Figure 10 Timing comparison of MPI and Hybrid MPI + OpenMP version of BT-MZ, LU-MZ for the Class A

From Figure 10 and Figure11, we can apparently see that MPI paradigm in our virtual Cluster performs best, even for the LU application which is sensitive to communication delay. And the hybrid OpenMP + MPI which can combine the advantage of MPI and OpenMP

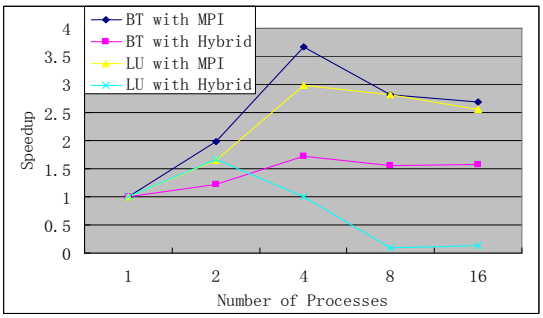

Figure 11 Speedup for MPI and Hybrid Paradigm

in physical SMP cluster [16, 17] does not perform very well, especially for LU, because of the limitation of OpenMP in virtual machine just like what Figure 6 reflects. As running time as concerned, hybrid paradigm costs less time than MPI, since it can take the advantage of the SMP nodes when the process number is 1 or 2. But the running time of LU increases dramatically after the process number is over 2, because the OpenMP need too much time for

application sensitive to communication delay in virtual machine with more vCPU than pCPU.

What's more, we can find that the MPI application performs better than the same applications with OpenMP, although the applications may be affected negatively by the VM environment. Taking Lu for example, in the experiment above, it can get more than 1.5 speedup with MPI implementation; nevertheless only get less than 0.5 speedup with OpenMP, when the vCPU number is more than 5. Since the performance of simple parallel programs with little communication or synchronization is very good both with MPI and OpenMP, which has been described in section 4.1, a possible reason for this phenomenon is that the MPI has a better communication ability than OpenMP in VM. Due to that within a node, MPI implementations such as MPICH2 can also use shared memory to finish inter-process communicating like OpenMP does. So we would test the memory access difference of MPI and OpenMP in Xen VM.

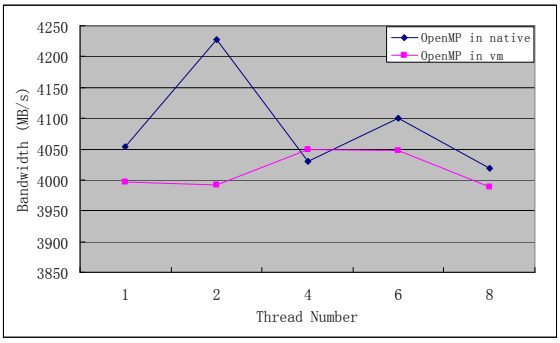

Figure 12 Memory bandwidth for OpenMP

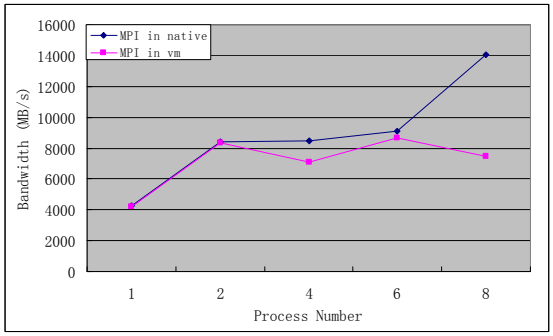

Figure 13 Memory bandwidth for MPI

From Figure 12 and 13, we can clearly see that the memory bandwidth of MPI paradigm is significantly larger than OpenMP. Since OpenMP provides thread parallelism, and threads don't span processes, the memory access of these threads is limited into one process level, although the bandwidth is stable with the thread number increasing. As we all know, the inter-thread or inter-process communication is crucial for parallel programs, especially for the communication-sensitive programs. So the parallel programs with MPI will perform better in VM environment, due to its communication advantage over OpenMP.

5. Related Work

There are many published reports on the comparison of different programming paradigms in physical cluster. We can only name a few of them. Some aspects of hybrid programming on SMP clusters are discussed in [17]. An evaluation of multilevel parallel programs applicable to shared memory computer architectures is given in [16]. In HPC field, [19] provides a performance evaluating of the impact of Xen on MPI and process execution for HPC systems. And, a case for HPC with virtual machine is described in [8]. But these researches all focus on the parallel performance in cluster, however, this paper mainly explore the performance of parallel programs in one node with the virtual machine environment.

For improving the performance of parallel computation-intensive workload in the virtual machine system, especially with the SMP VM, some researches provide the new vCPU scheduler. For example, co-schedule method in [20] trys to make all vCPU be mapped to pCPU simultaneous for the parallel programs with much synchronization. However, the vCPU number in co-schedule is strict.

6. Conclusions and future work

We have run several implementations of the same CFD benchmark code employing different parallelization paradigms on a VM or multiple virtual SMP nodes. I found that it is not true that more vCPUs than pCPUs always make a significantly negative impact on the parallel programs in virtual machine environment, especially when the communication and synchronization between threads or processes is little. And the pure MPI paradigm turned out to be the most efficient. For the parallel programs which are not sensitive to communication delay, the OpenMP and Hybrid paradigm have the close performance to MPI, no matter whether the vCPU number is higher than the pCPU number or not. But for some application sensitive to communication delay, taking LU for example, the performance of OpenMP and Hybrid degrades significantly.

In this paper, we only test the parallel programs in paravirtualization system. We plan to make a similar test in virtual machine with the hardware virtualization support, such as Intel-VT, AMD-V technique. Further

more, we also want to test the parallel performance in virtual machine with other vCPU schedule algorisms, SEDF for instance, instead of the Credit used in my experiments of this paper. What's more, the other new scheduler such as co-schedule method will be also considered in the next work.

7. Acknowledge

This work is supported by the National Science Foundation of China under Grant Nos. 90612004 and 60873071. The National High Technology Development 863 Program of China under Grant Nos. 2007AA01Z118 and 2008AA01Z410.

8. References

[1] Himanshu Raj, Karsten Schwan. High Performance and Scalable I/O Virtualization via Self-Virtualized Devices. *HPDC'07*, 2007.

[2] Amazon Elastic Compute Cloud (EC2). aws.amazon.com/ec2.

[3] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the Art of Virtualization. In *SOSP'2003*, 2003.

[4] Lamia Youseff, Rich Wolski, Brent Gorda, and Chandra Krintz. Paravirtualization for hpc systems. *LECTURE NOTES IN COMPUTER SCIENCE*, 2006.

[5] Randal E. Bryant, David R. O'Hallaron. Computer Systems: A Programmer's Perspective. Published by *Prentice Hall*, 2004.

[6] A Gavrilovska, S Kumar, H Raj, K Schwan, V Gupta, R Nathuji, R Niranjan, A Ranadive, P Saraiya. High-Performance Hypervisor Architectures:Virtualization in HPC Systems. *HPCVirt'07*, 2007.

[7] Adit Ranadive, Mukil Kesavan, Ada Gavrilovska, Karsten Schwan. Performance Implications of Virtualizing Multicore Cluster Machines. *HPCVirt'08*, 2008.

[8] W Huang, JX Liu , B Abali, DK Panda. A Case for High Performance Computing with Virtual Machines. *ICS'06*, 2006.

[9] The VMWare ESX Server. http://www.vmware.com/products/esx/.

[10] L. Cherkasova, D. Gupta, and A. Vahdat. Comparison of the Three CPU Schedulers in Xen. *ACM SIGMETRICS Performance Evaluation Review*, 35(2):42–51, 2007.

[11] A. Whitaker, M. Shaw, and S. D. Gribble. Scale and Performance in the Denali Isolation Kernel. *In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, Dec 2002.

[12] INTEL CORPORATION. Intel Virtualization Technology Specification for the IA-32 Intel Architecture, April 2005.

[13] AMD. AMD64 Virtualization Codenamed "Pacifica" Technology:Secure Virtual Machine Architecture Reference Manual, May 2005.

[14] R. V. der Wijngaart, NAS Parallel Benchmarks v. 2.4. NAS Technical Report NAS-02-007, October 2002.

[15] Haoqiang Jin, Barbara Chapman, Lei Huang. Performance evaluation of a multi-zone application in different OpenMP approaches. *International Journal of Parallel Programming*, Volume 36, Issue 3 (June 2008).

[16] Gabriele Jost, Jesús Labarta, and Judit Gimenez. What Multilevel Parallel Programs Do When You Are Not Watching: A Performance Analysis Case StudyComparing MPI/OpenMP, MLP, and Nested OpenMP. *Lecture Notes in Computer Science*, page 29-40, Volume 3349, 2005.

[17] Gabriele Jost, Haoqiang Jin, Dieter an Mey, Ferhat F. Hatay. Comparing the OpenMP, MPI, and Hybrid Programming Paradigms on an SMP Cluster. *The Fifth European Workshop on OpenMP (EWOMP03)*, 2003.

[18] John McCalpin, STREAM: Sustainable Memory Bandwidth in HPC, http://www.cs.virginia.edu/stream.

[19] Lamia Youseff, Rich Wolski, Brent Gorda, and Chandra Krintz. Evaluating the Performance Impact of Xen on MPI and Process Execution For HPC Systems. *In Proceedings of the 2nd International Workshop on Virtualization Technology in Distributed computing*, 2006.

[20] Chuliang Weng, Zhigang Wang, Minglu Li, and Xinda Lu. The Hybrid Scheduling Framework for Virtual Machine Systems. *In Proceedings of the 5th international conference on Virtual execution environments (VEE),* 2009