Cooperative Virtual Machine Scheduling on Multi-core Multi-threading Systems — A Feasibility Study

Dulcardo Arteaga, Ming Zhao, Chen Liu, Pollawat Thanarungroj, Lichen Weng
School of Computing and Information Sciences
Department of Electrical and Computer Engineering
Florida International University
{darte003, mzhao, cliu, pthan001, lweng001}@fiu.edu

ABSTRACT

Virtual machines (VMs) and multi-core multi-threading micro-processors (MMMP) are two emerging technologies in software and hardware, respectively, and they are expected to become pervasive on computer systems in the near future. However, the nature of resource sharing on an MMMP introduces contention among VMs which are scheduled onto the cores and the threads that share the processor computation resources and caches. Such contention can lead to performance degradation of individual VMs as well as the overall system throughput, if not carefully managed. This paper proposes to address this problem through cooperative VM scheduling that takes processor input to schedule VMs across processors and cores in a way that minimizes the contention on processor resources and maximizes the total throughput of the VMs. As a first step towards this goal, this paper presents an experiment-based feasibility study for the proposed approach and focuses on the effectiveness of process contention aware VM scheduling. The results confirm that when VMs are scheduled in a way that mitigates their contention on the shared cache, the cache miss rates from the VMs are reduced substantially, and so do the runtimes of the benchmarks.

1. INTRODUCTION

With the rapid growth of computational power on compute servers and the fast maturing of x86 virtualization technologies, Virtual Machines (VM) are becoming increasingly important in supporting efficient and flexible application and resource provisioning. Modern VM technologies (e.g. [1][2][3]) allow a single physical server to be carved into multiple virtual resource containers, each delivering a powerful, secure, and isolated execution environment for applications. In addition to providing access to resources, such environments can be customized to encapsulate the entire software and hardware platform needed by the applications and support their seamless deployments.

In the meantime, Multi-core Multi-threading MicroProcessors (MMMPs) are recognized as the new paradigm for continuing the rapid growth of processing power in computing systems. Instead of solely increasing the clock frequency for better performance, it is parallelism that finally enlightens the future of microprocessor design. The emergence of MMMPs is enabling increasingly powerful computer systems. Today multi-core multi-threading systems are already pervasive, whereas the advent of many-core many-threading systems is anticipated to happen in the near future.

However, the multi-core multi-threading architecture introduces resource sharing at both the core-level and the thread-level in a processor. The cores on the same processor commonly share the last level cache (LLC), while the threads on the same core also share computation resources and private caches. In the Simultaneous Multithreading (SMT) architecture, the execution resources in the same core are fully shared by the concurrently executing threads. Nevertheless, resource distribution among cores and threads determines not only the individual core/thread performance, but also the overall system throughput [4]. Without well-defined resource management scheme, monopoly on the shared resources may happen in an MMMP and thus lead to performance degradation [5].

Consequently, when considering the scheduling of multiple VMs onto the different cores of a processor and the different threads of each core, they may compete for the same processor resources when they are of the similar behavior in phases of execution, while other resources that are not mainly consumed by these VMs are relatively idle. Nonetheless, this kind of idle resources may be heavily demanded in other cores or other processors of the system. As a result, the distributed resources are not fully utilized, and the workload performance suffers from the scheduling policy, rather than limited processor resources in the system.

This paper proposes to address this problem through cooperative VM scheduling that allows software-level VM scheduler and hardware-level thread scheduler to communicate and cooperate in order to minimize the contention on processor resources among concurrent VMs and maximizes the total throughput of the VMs. Such cross-layer cooperation in VM scheduling is two-fold. On one hand, the software VM scheduler can use the hardware-provided information to model the behaviors of VMs on their use of shared processor resources and schedule them in a way that minimizes their contention. On the other hand, the software-level scheduling information can be fed back into the hardware-level scheduler in order to assist the latter's thread scheduling decision and to achieve global system optimization.

As a first step towards this goal, this paper presents an experiment-based feasibility study for the proposed approach. It focuses on validating the necessity of scheduling policy based on VM processor resource demands for better utilization and less competition. The results from an Intel Core 2 Quad processor based platform confirm that when VMs are scheduled in a way that mitigates their contention on shared cache, the cache miss rates from the VMs are reduced substantially, and so do the runtimes of the benchmarks. But the results from a hyperthreaded Intel Core i7 Quad processor based platform do not show obvious contention between VMs running on simultaneous threads, which require further investigation.

The rest of this paper is organized as follows: Section 2 introduces the background and related work; Section 3 presents the proposed approach; Section 4 discusses the experimental study; and Section 5 concludes this paper.

2. BACKGROUND AND RELATED WORK

Virtual Machine (VM) is a powerful layer of abstraction for application and resource provisioning [6]. The VMs considered in this project are system-level VMs (e.g., [1][2][3]), which virtualize an entire physical host's resources, including CPU, memory, and I/O devices and present virtual resources to the guest OSes and applications. System virtualization is implemented by the layer of software called virtual machine monitor (VMM, a.k.a., hypervisor), which is responsible of multiplexing physical resources among the VMs. Full-virtualized VMs [1][3] present the same hardware interface to guest OSes as the physical machines and support unmodified OSes. Paravirtualized VMs [2] present a modified hardware interface optimized for reducing virtualization overhead but require the guests OSes to be modified as well. In addition, hardware support for virtualization is also emerging in new processors [7][8], which can be used by VMMs to further improve the efficiency of VMs.

Multi-core Multi-threading MicroProcessors (MMMPs) are recognized as the new paradigm for continuing the rapid growth of processing power in computing systems. Transistor count on a chip increased rapidly in the past several decades, urged by the famous Moore's Law [9]. The pressure, however, now falls onto the parallelism in a processor [10]. As a result, the multi-core architecture was employed to explore the Job Level Parallelism. Furthermore, the parallelism is well developed within multi-threading architecture. Especially, in the SMT architecture, the Horizontal waste and Vertical waste are minimized [11]. Motivated by these emerging VM and MMMP technologies in computer systems, this paper studies the scheduling of VMs on multi-core multi-threading platforms when they compete for shared processor resources.

Application Processor Resource Demand Modeling is important because applications have diverse needs of processor resources depending on whether their execution time is spent on processing or memory access [12]. Zhu *et al.* proposed to use Cycle Per Instruction (CPI) portions to statistically measure application resource demands, specifically in the format of CPI_{proc} and CPI_{mem} [13]. The application mainly consumes computation resources when most of its average CPI is spent on processing (CPI_{proc}), or composed of small CPI values spent on memory access (CPI_{mem}). On the contrary, the larger the CPI_{mem} is, the more memory resources the application requests. Therefore, they also argue that performance of applications with large CPI_{mem} values are memory-bound, while those with small CPI_{mem} values are computation-bound. The same categorizing result about SPEC CPU2000 benchmarks [14] is also given by Cazorla et al. in [14], using average Level 2 Cache Miss Rate. This paper follows this approach to classify the processor resource demands of VMs when they host different types of applications.

VM-level Resource Scheduling supports dynamic allocation of shared physical machine resources, including CPU cycles, memory capacity, and I/O bandwidth to concurrent VMs according to their demands. Various methods have been studied in the literature to model the resource usage behaviors of VMs in order to support on demand resource scheduling. For example, Xu *et al.* [25] studied fuzzy-logic based modeling for allocating CPU cycles across VMs hosting CPU-intensive applications; The CRAVE project employs simple regression analysis to predict the performance impact of memory allocation to VMs [16]; The VCONF project has studied using reinforcement learning to automatically tune the CPU and memory configurations of a VM in order to achieve good performance for its hosted application [18]; Kund *et al.* employs artificial neural networks (ANN) to build performance models that consider both CPU and memory allocation to VMs and I/O contention between VMs [19]. This paper tries to model the behaviors of VMs on the use of shared processor cache resources and the impact of contention on such resources.

Processor-level Resource Scheduling is important for a multithreading processor to achieve the optimum resource distribution among threads, which leads to desired performance improvement. Raash *et al.* [20] studied various system resource partitioning mechanisms on SMT processors and concluded that the true power of SMT lies in its ability to issue and execute instructions from different threads at every clock cycle. ICOUNT policy [21] prioritizes the threads based on the number of instructions in the front-end stages from each thread to decide from which thread to fetch instructions. DCRA [22] was proposed in an attempt to dynamically allocate the resources among threads by dividing the execution of each thread into different phases, using instruction and cache miss counts as indicators. Hill-Climbing [23] dynamically allocates the resources based on the current performance of each thread and feedback into the resource-allocation engine. ARPA [24] is proposed as an adaptive resource partitioning algorithm that dynamically assigns pipeline resources to the threads according to thread behavior changes.

Processor Contention Aware Application/VM Scheduling: Several related projects have studied the impact of contention on shared processor resources, particularly shared caches, to the performance of applications or VMs and the scheduling of applications/VMs under such contention. Q-Clouds [27] showed that the interference on the shared last level cache (LLC) has a significant impact on a memory-bound VM and proposed a feedback control based approach to tune resource allocation for mitigating this impact. Koller *et al.* [26] proposed a generalized effective reuse set size model to characterize an application's memory usage behaviors including the impact of cache contention. Knauerhase *et al.* [28] suggested that the OS should be able to dynamically migrate a thread onto a different core, so as to achieve the ideal scheduling of pairing the cache-heavy and cache-light threads on the same core. Weng *et al.* studied the use of cycles per instruction spent on memory (CPI_{mem}) to express an application needs and proposed mix-scheduling of threads with different CPI_{mem} diversity on a multithreading processor in order to reduce the cache miss rate [29]. This paper follows this approach to study the mix-scheduling of VMs with different processor usage behaviors in order to reduce the contention on shared processor resources between concurrent VMs.

3. COOPERATIVE VM SCHEDULING ON MMMP SYSTEMS

As discussed in the previous section, on an MMMP-based VM system, there exist two levels of resource scheduling, the software-level VM scheduler that allocates the processors and cores across concurrent VMs, and the hardware-level thread scheduler that allocates each core's resources across VMs mapped to the same core. However, these two levels of schedulers are traditonally isolated: the software-level scheduler is unaware of the contention occurred on the processor resources, while the

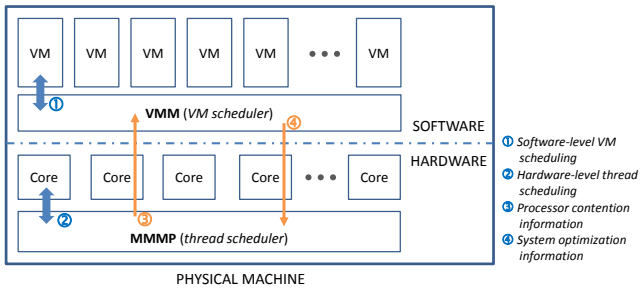

Figure 1. The architecture of cooperative VM scheduling

hardware-level scheduler is constrained by its optimizatio capability. This section presents our proposed cooperative VM scheduling architecture (Figure 1) designed to break this isolation and enable cross-layer scheduling cooperation, in an effort to fully utilize the distributed resources on an MMMP platform with minimum competition and maximum throughput.

3.1 Processor Contention Aware VM Scheduling

Given the fact that processor resources are distributed among cores in an MMMP, the scheduling of VMs on such a system should overcome the resource isolation among different cores, so that they can fully utilize the distributed processor resources. On the other hand, because processor resources can be also shared by different cores and different threads on the same core, the scheduling should also map VMs to the cores in a way that minimizes competition for the shared resources. Therefore, following the approach presented in our prior work for application scheduling [29], we propose the *mix-scheduling* policy to keep VM diversity in terms of their demand of processor resources in every set of cores and threads that share the resources.

Specifically, the shared processor resources that need to be considered on a MMMP platform include both the shared LLC between different cores on the same processor and the shared private cache and computation resources (such as instruction queue, reorder buffer, register files) between different threads on the same core. To realize the proposed mix-scheduling policy, the software-level VM scheduler needs to understand each VM's behaviors in using such shared processor resources. The information necessary for understanding such behaviors has to be provided by the processor, because these shared resources are generally directly managed by hardware. To capture the behavior of a VM on the shared private cache and LLC, the scheduler can leverage the performance counters that most modern processors support for collecting the cache miss statistics. To capture a VM's use of shared computation resources, additional support would be necessary from processors.

Taking the miss rate on a shared LLC as an example, the mix-scheduling policy would evenly mix VMs of various miss rate across cores so that the variance of miss rates among the cores and/or the threads that share the LLC are maximized. As the direct result of the proposed policy, the difference of miss rates between competing VMs is expected to be significant. Hence, the VMs with smaller miss rates consume mainly the computation resources, while the throughput of other VMs with larger miss rates depends on memory resources. Consequently, VMs in the resource sharing cores and/or threads require different resources, rather than compete for the same resource severely. Moreover, from the perspective of the entire system, it means to utilize more resources globally if its VMs are able to access various resources in the cores, and thus it results in better overall performance.

Another example considers VMs heavily involved with floating point operations and others heavily involved with integer operations. In such circumstance, it would be better for the VM scheduler to also make a mix-scheduling decision, e.g., to pair one floating point VM with one integer VM onto the same core in order to maximize the utilization of the execution engine, reduce the contention, and improve system throughput. This certainly cannot be achieved without the hardware-level scheduler being able to track the execution engine utilization for different VMs and feed this information back to the software-level scheduler.

3.2 Software-assisted Thread Scheduling

In MMMP, the processor hardware-level resource scheduling has a significant impact on overall system performance. It must decide how the system resources on each core are divided among multiple threads. For example, how many entries one thread can occupy in Instruction Fetch Queue (IFQ), Instruction Issue Queue (IIQ), ReOrder Buffer (ROB), and Renaming Register, separately. How to divide the issue and commit bandwidth among multiple threads is also a part of the scheme. If there is no control on the resources that can be assigned to one thread in one core, this would cause the uneven distribution of resources among threads and uneven execution of the threads, which also translates into the overall time to execute all threads being extended. However, the optimization that a hardware-level scheduler can perform is constrained by the given VMs mapped by the software-level scheduler to the same core. Therefore, the scheduling optimization made by the software-level VM scheduler should also indirectly benefit the hardware-level thread scheduler in that it allows the latter to better optimize its resource allocation and further improve the performance for the VMs mapped to the same core.

A more direct way for software to assist the hardware-level scheduling is to implement the intelligence necessary for the hardware-level optimization in software. For example, one of the most important topics in threading scheduling is to be able to track the phase change of every thread based on their instruction commit rate and cache miss rate, etc. However, to precisely catch this phase change normally requires complex machine learning algorithms, which is certainly not suitable for hardware implementation. As such, it is conceivable to let the software layer execute this kind of complex algorithms. If we could feed this information back to the hardware-level scheduler, it would certainly benefit its decision making process, especially in terms of making decision from global optimum point of view, as opposed to local optimum.

4. EXPERIMENTAL ANALYSIS

4.1 Setup

A series of experiments were conducted on two different multi-core platforms to investigate the feasibility of the proposed mix-scheduling of VMs. The first platform is a physical machine with Intel Core 2 Quad CPU (Q9400, 2.66GHz) and 4GB RAM. This quad-core processor has two pairs of cores where each pair shares 6MB of L2 cache. The second platform is a physical machine with Intel Core i7 CPU (860, 2.80GHz) and 8GB RAM. This hyperthreaded quad-core processor supports eight simultaneous threads with 8MB of shared L3 cache. These machines are

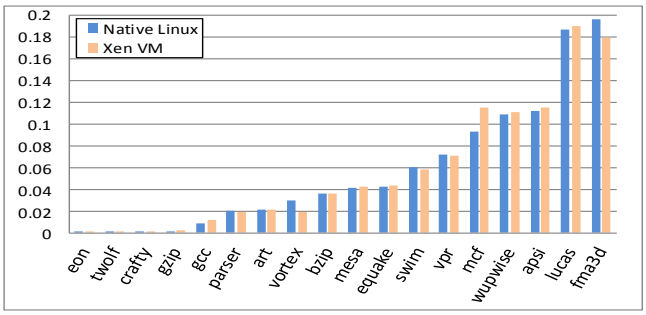

Figure 2. The LLC miss rates of SPEC CPU benchmarks when running on Native Linux vs. on Xen VM

installed with both the native Linux with 2.6.18 kernel and Xen-based VM environment with paravirtualized 2.6.18 kernel.

For this study we used the benchmarks from the SPEC CPU2000 suite [14]. Each benchmark runs on a dedicated VM which is configured with one virtual CPU and 1024MB of memory and pinned to one dedicated physical or logical core during the execution. To collect the cache miss rate, we used Xenoprof [30] to monitor the VM's cache misses online during its benchmark's execution. But the profiling was disabled when we were measuring the benchmark's runtime on the VM. All the results reported in this section are the average values of at least three runs. The standard deviation of all the results is very small and thus not shown here.

4.2 Benchmark VM Classification

As discussed in Section 3.1, VMs can be modeled with respect to their behaviors in using various shared processor resources and classified into different categories accordingly. Such modeling and classification can also be performed online in order to capture the different behaviors of a VM's various phases. However, to make this feasibility study more focused, our experiments are simplified in two aspects.

First, we assume that the benchmark VMs have only a single phase and they are modeled offline by running them separately without any contention. The profiling was focused on the VM's LLC miss rate. Second, the offline modeling and classification consider only the VMs' LLC miss rates. Some benchmark VMs perform a lot of operations on a single data fetched from the memory system, so that they have low cache miss rates and mainly consume the computation resources. These benchmarks are capable of providing more Instruction Level Parallelism (ILP) and fall into the *computation-bound* category. On the contrary, some other benchmark VMs deal with a large amount of data during execution, but perform relatively less operations on a single data, so they have high cache missrates. Hence, they belong to the *memory-bound* category.

Figure 2 shows the LLC miss rates of the different SPEC CPU2000 benchmarks when they run on VMs versus when they run on the native Linux. When a benchmark runs on its Xen VM, the measured cache misses include not only the cache misses occurred during the execution of the benchmark on the dedicated core but also those during the execution of the supporting guest OS and Xen VMM services. But, as we can see from the figure, the use of VM to run the benchmarks does not result in a significant difference on the use of shared LLC compared to when these benchmarks are run on the native Linux. This observation infers that the conclusions and solutions developed for application-level scheduling on MMMP systems in the literature can be also applied to the VM-level scheduling. Second, the results in Figure 2 confirm that these benchmark VMs indeed have diverse behaviors in their LLC usage with their miss rates varying from nearly zero, in the cases of *eon*, *twolf*, *crafty*, and *gzip*, to around 20%, in the cases of *lucas* and *fma3d*. Based on these results, it is reasonable to use the 2% cache miss rate as the cutoff line and classify these benchmarks into the computation-bound category (including *eon*, *twolf*, *crafty*, *gzip*, and *gcc*) and the memory-bound (including *parser*, *art*, *vortex*, *bzip*, *mesa*, *equake*, *swim*, *vpr*, *mcf*, *wupwise*, *apsi*, *lucas*, and *fma3d*) category. This classification would not change even if we use the benchmark cache miss rates from the native Linux.

4.3 Mix-Scheduling vs. Mono-Scheduling

According to the above classification, memory-bound category demands memory resources while computation-bound category demands computation resources. Therefore, following the mix-scheduling policy, which VMs should be scheduled according to their resource demands in order to maximize VM cache miss rate variance in the same set of resource-sharing core(s). Hence, the benchmark VMs belonging to different categories should be scheduled onto the same set of core(s). Specifically, a memory-bound VM should be mapped together with a computation-bound VM onto the same set of core(s). In this way, the memory-bound VM relies more on memory resource in the core(s), while the computation-bound VM lives more on computation resource in the core(s), which matches our goal to minimize competition and optimize utilization. On the contrary, the *mono-scheduling* policy schedules benchmark VMs from the same category onto the same set of core(s), i.e., it schedules two memory-bound VMs or two computation-bound VMs onto the same set of core(s).

The first group of experiments that compare mix-scheduling to mono-scheduling was done on the Intel Core 2 Quad CPU based platform. Because the four cores on this processor consist of two LLC-sharing pairs, each pair of cores forms a resource-sharing core set. In the experiments, two VMs were run on a pair of cores with each VM pinned to a different core of the pair. Figure 3 plots the miss rates of different benchmark VM pairs (*VM1*, *VM2*) when running on a pair of LLC-sharing cores in such a way. The data are grouped based on *VM1* and within each group the bars are arranged in an increasing order of the standalone miss rates of *VM2* reported in Figure 2. From the results where *VM1* is a computation-bound VM, we can see that when it runs along with a memory-bound VM the combined LLC miss rates are still low, so such mix-scheduling based pairs can effectively use the process resources without much adverse performance impact. From the results where *VM1* is a memory-bound VM, we can see that when it runs with another memory-bound VM, the combined cache miss rates are substantially higher than when it runs with a computation-bound VM. This also validates the effectiveness of our proposed mix-scheduling policy because it can significantly lower the contention on shared LLC compared to the mono-scheduling. These observations can also be confirmed from the difference in runtimes of *VM1* in different (*VM1*, *VM2*) pairs (Figure 4). Note that this figure has less VM pairs than Figure 3 because it includes only the pairs where *VM1* finishes earlier than *VM2* and thus ensures that the execution of *VM1* is completely under the contention from *VM2*. For example, when benchmark

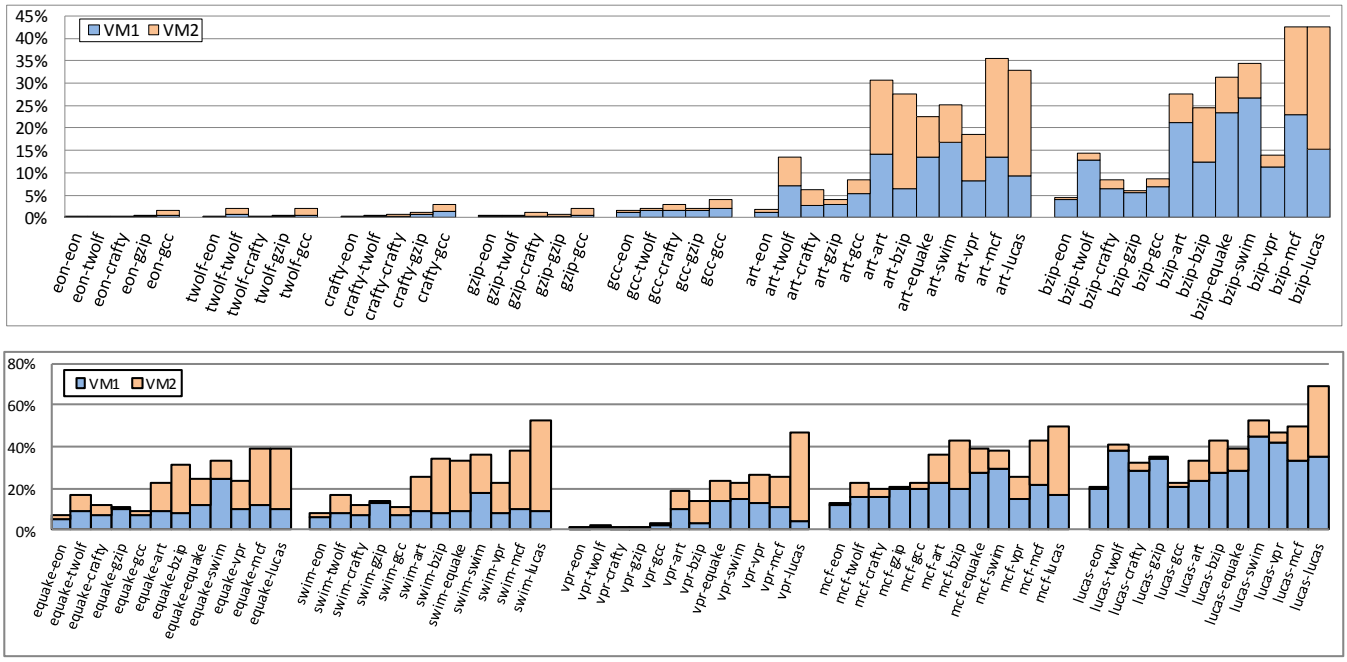

Figure 3. LLC miss rates of different VM pairs on an Intel Core 2 Quad core pair

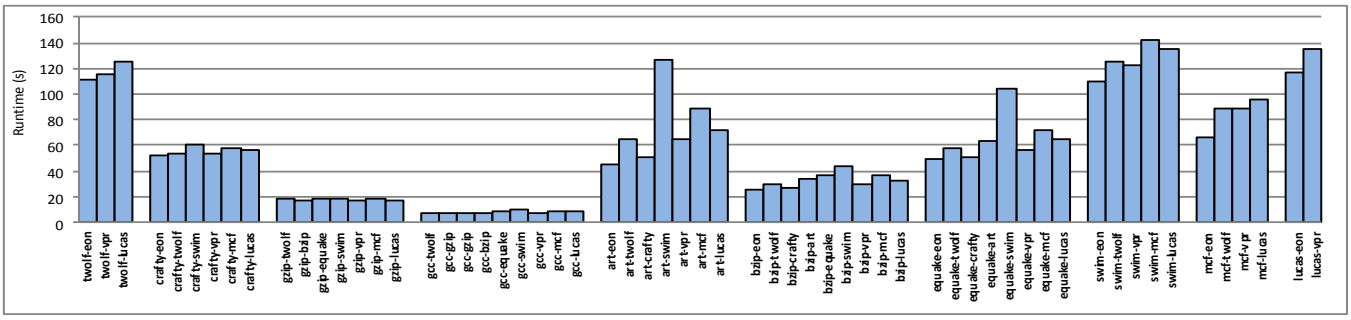

Figure 4. Runtimes of different VM pairs on an Intel Core 2 Quad core pair

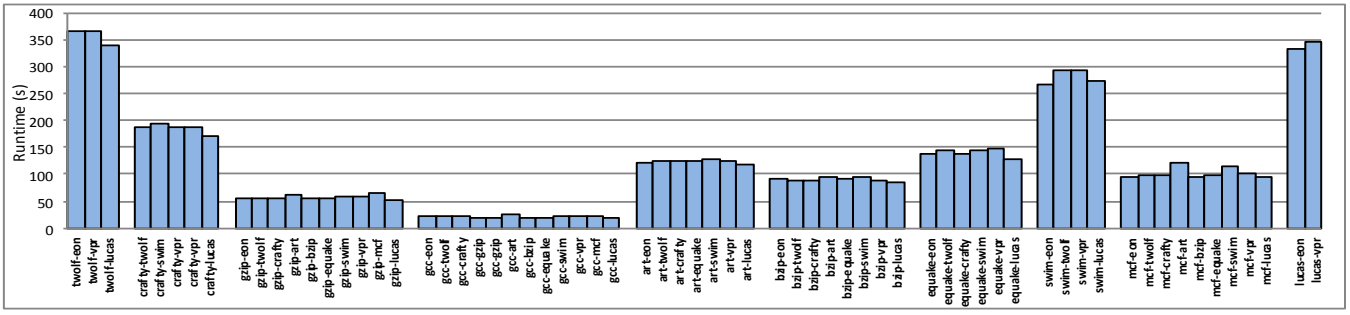

Figure 5. Runtimes of different VM pairs on an Intel Core i7 hyperthreaded core

art runs with *eon*, which has a lower cache miss rate, the runtime is 44s, compared to *art* running with *swim*, which has a higher miss rate, the resulting runtime is 126s.

The second group of experiments was designed similarly as the first group but it was done on the Intel Core i7 based platform. This processor has four cores each with two simultaneous threads. Therefore, when a pair of VMs is executed on two logical cores mapped to the same physical core, they should compete for the shared caches and computation resources. Due the limitation of Xenoprof, we were not able to collect the cache miss rates on Intel Core i7, so only the runtimes of the *VM1* in different (*VM1*, *VM2*) pairs were measured (Figure 5). However, the data reveal a surprising result that the runtime of *VM1* in each pair is not impacted much by the behavior of *VM2* no matter whether in mix-scheduling or mono-scheduling. We are still investigating the reason behind this observation, but we believe that this may be due to the fact that Intel hyperthreading statically partitions the fetch queue between simultaneous threads to mitigate contention as well as the availability of more cache resources in Core i7 as compared to Core 2.

5. CONCLUSION

With the pervasive use of VMs and MMMPs in today's computer systems, it becomes important to understand the contention on shared processor resources between concurrent VMs and mitigate its impact on VM performance and system throughput. This paper proposes a cooperative VM scheduling approach that allows software-level VM scheduler and hardware-level thread scheduler to cooperate and optimize the allocation of MMMP resources to VMs. It presents an experiment-based feasibility study which confirms the effectiveness of processor contention aware VM scheduling. In our future work we will continue this investigation and focus on the VM contention problem of not only shared caches but also shared computation resources on multithreaded processor cores.

REFERENCES

[1] VMware Inc., URL: http://www. vmware. Com.

[2] P. Barham, et al., "Xen and the Art of Virtualization", ACM Symposium on Operating Systems Principles , October 2003.

[3] Kernel Based Virtual Machine, URL: http://www.linux-kvm.org/page/Main_Page.

[4] D.M. Tullsen, et al., "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor", 23rd Annual International Symposium on Computer Architecture, pp. 191–202, 1996.

[5] C. Liu and J. Gaudiot, "The Impact of Resource Sharing Control on The Design Of Multicore Processors", 9th International Conference on Algorithms and Architectures for Parallel Processing, pp. 315–326, 2009.

[6] R. Figueiredo, P. Dinda, and J. Fortes, "Resource Virtualization Renaissance", IEEE Computer Magazine 38(5), Special Issue on Virtualization, pp. 28-31, May 2005.

[7] AMD Virtualization, URL: http://www.amd.com/us/products/technologies/virtualization.

[8] Rich Uhlig, et al., "Intel Virtualization Technology", Computer, Volume: 38, Issue: 5, May, 2005.

[9] G.E. Moore, "Cramming More Components onto Integrated Circuits", Electronics, vol. 38, no. 8, pp. 114–117, 1965.

[10] K. Asanovic, et al., "The Landscape of Parallel Computing Research: A View From Berkeley," Technical Report, University of California at Berkeley, 2006.

[11] D.M. Tullsen, S.J. Eggers, and H.M. Levy, "Simultaneous Multithreading: Maximizing On-Chip Parallelism," 22nd International Symposium on Computer Architecture, 1995.

[12] A. Kagi, J.R. Goodmand, and D. Burger, "Memory Bandwidth Limitations of Future Microprocessor", 23rd International Symposium on Computer Architecture, 1996.

[13] Z. Zhu and Z. Zhang, "A Performance Comparison of DRAM Memory System Optimizations for SMT Processors", 11th International Symposium on High-Performance Computer Architecture, pp. 213–224, 2005.

[14] J.L. Henning, "SPEC CPU 2000: Measuring CPU Performance in the New Millennium", Computer, vol. 33, no. 7, pp. 28–35, 2000.

[15] F.J. Cazorla, et al., "Predictable Performance in SMT Processors: Synergy between OS and SMTs," IEEE Transactions on Computer. vol. 55, no. 7, 2006.

[16] Jonathan Wildstrom, et al., "CARVE: A Cognitive Agent for Resource Value Estimation", 5th IEEE International Conference on Autonomic Computing, 2008.

[17] T. Wood, et al., "Profiling and Modeling Resource Usage of Virtualized Applications", Proc. of the 9th International Middleware Conference, December, 2008.

[18] Jia Rao et al., "VCONF: A Reinforcement Learning Approach to Virtual Machines Auto-configuration", 6th International Conference on Autonomic Computing, 2009.

[19] Sajib Kundu et al., "Application Performance Modeling in a Virtualized Environment", 16th International Symposium on High-Performance Computer Architecture, 2010.

[20] S. Raasch and S.Reinhardt, "The Impact of Resource Partitioning on SMT Processors", 12th International Conference on Parallel Architectures and Compilation Techniques, pp. 15–25, 2003.

[21] D. Tullsen, et al., "Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor", 23rd Annual International Symposium on Computer Architecture, p. 191, 1996.

[22] F.J. Cazorla, et al., "Dynamically Controlled Resource Allocation In SMT Processors", 37th annual IEEE/ACM International Symposium on Microarchitecture, 2004.

[23] S. Choi and D. Yeung, "Learning-based SMT Processor Resource Distribution Via Hillclimbing", 33rd Annual International Symposium on Computer Architecture, 2006.

[24] H. Wang, I. Koren, and C.M. Krishna, "An Adaptive Resource Partitioning Algorithm For SMT Processors", 17th International Conference on Parallel Architectures and Compilation Techniques, 2008.

[25] Jing Xu et al., "Autonomic Resource Management in Virtualized Data Centers Using Fuzzy-logic-based Control", Cluster Computing, Vol. 11, No. 3, September 2008.

[26] Ricardo Kollera, Akshat Vermab, and Raju Rangaswami, "Generalized ERSS Tree Model: Revisiting Working Sets", Performance Evaluation, Volume 67, Issue 11, Nov 2010.

[27] Ripal Nathuji, Aman Kansal, and Alireza Ghaffarkhah, "Q-Clouds: Managing Performance Interference Effects for QoS-Aware Clouds", Eurosys 2010.

[28] R. Knauerhase, et al., "Using OS Observations to Improve Performance in Multicore Systems", IEEE Micro, 28(3), 54–66, 2008.

[29] L. Weng and C. Liu, "On Better Performance from Scheduling Threads According to Resource Demands in MMMP", 39th International Conference on Parallel Processing Workshops, 2010.

[30] Aravind Menon, et al., "Diagnosing Performance Overheads in the Xen Virtual Machine Environment", 1st Conference on Virtual Execution Environments (VEE'05), 2005.