XenMVM: Exploring Potential Performance of Multi-core System in Virtual Machine Environment

Zhiyuan Shao, Jian Huang, Hai Jin, and Kan Hu

Services Computing Technology and System Lab

Cluster and Grid Computing Lab

School of Computer Science and Technology

Huazhong University of Science and Technology, Wuhan, 430074, China

hjin@hust.edu.cn

***Abstract*— In this work, we propose computing resource management system based on Xen VMM and multi-core system, called XenMVM. It adjusts the computing resource dynamically according to the actual workload generated by the applications running in the virtual machine to improve the resource utilization of the computer system. According to the shared L2 cache architecture of multi-core system, we propose the *Underlying Layout Aware Scheduling* (ULAS) which can schedule the virtual CPUs of virtual machines to the appropriate physical CPUs. The test result shows that ULAS can improve the performance about 4.5%~32.52%. Furthermore, to improve the whole performance of multiple virtual machines and meet the needs of specific jobs (i.e., urgent and I/O intensive) deployed in the virtual machines, a simplified method called Domain-based static priority is adopted in XenMVM. Using case studies we show that our proposal reduces the turnover time of the whole system by 20.45% compared with FCFS scheduling scheme.**

***Keywords*— Multi-core system; virtual machine; resource management; scheduling strategies**

I. INTRODUCTION

With the progress of multi-core processors, the pressure on the existing system software for the management of these computing resource and performance optimization is greatly increased.

Virtualization technology [2] provides us a key to alleviate these problems mentioned above. It enables increased utilization, reduced complexity and lower *total cost of ownership* (TCO). However, as the state-of-art virtualization techniques use static resource allocation mechanism, the system resources are underutilized. Usually, we can increase the utilization of computing resource by allowing multiple virtual machines to share the resource of a single server, but because of the limitation of the hardware architecture, such as the memory bandwidth bottleneck, shared cache architecture, it is entirely possible for one virtual machine which hosts I/O intensive jobs or suchlike to saturate the whole system resulting in degraded performance for the jobs running in other virtual machines. Resource management problem in virtual machine environment has become an open problem [3].

In this work, we propose a computing resource management scheme named XenMVM based on Xen. The scheme can adjust the computing resource dynamically according to the actual workload generated by the applications inside the *virtual machines* (VM). Compared with our previous work [1], the difference between the performance penalties of the two methods is negligible while the method we adopt in XenMVM is more suitable to cope with complex situations.

We explore the potential performance of multi-core system and design and implement the *Underlying Layout-Aware Scheduling* (ULAS) scheme based on the pinning mechanism provided by Xen and the traits of multi-core system's shared cache architecture. Using *NAS Parallel Benchmark* (NPB) [9] as the representative multithread applications to run inside the virtual machine, we find out that the execution performance gap reaches almost 4.5%~32.52% when mapping the virtual processors (VCPUs) of the virtual machine to different physical CPUs (PCPUs). The objective of the ULAS is to locate the threads generated by the above applications to the most suitable PCPUs for gaining the best performance.

Moreover, the situation will become more complex when deploying multiple virtual machines on the multi-core system. In the virtual machine environment, we can configure enough VCPUs to satisfy the needs of the applications running inside the virtual machines, but the execution time will be hard for the users to predict. Due to the bottleneck of the front end bus bandwidth in the multi-core system, one virtual machine which hosts concurrency intensive or communication intensive applications will bring down the performance of the whole system. We adopt the domain-based static priority scheduling to solve the problem based on the user behaviours in the virtual machines. All the requirements of computing resource will be sent to the resource manager located in the Dom0, the resource manager views all the PCPUs as a pool of resource and allocates computing resource according to the priority of each virtual machine and the remaining available resource. Comparing with the *First Come First Served* (FCFS) scheduling, the turn-round time of the whole system when adopting the static priority scheduling is much less. That means the performance of static priority scheduling is better than the FCFS scheduling in some situations.

This rest of this paper is organized as follows. Section II addresses the related works. Section III articulates the key design issues of our XenMVM system, including the system architecture and its components. Section IV discusses the scheduling methodologies. Section V conducts the experiment

978-1-4244-8812-4/10/$26.00 ©2010 IEEE

and presents the experiment results to justify the scheduling algorithms designed and implemented in our system. In section VI we summarize and conclude.

II. Related Works

We divide previous works into two groups: resource management and scheduling optimization for multi-core system.

A. Resource Management

In our previous work [1], we adopted AISD to adjust the computing resource allocated to virtual machine dynamically. With this strategy, the system will addictively increase the number of processors allocated to the virtual machine which sends request for more computing resources, and linearly decrease the number of processors allocated to those which send request for less resources. This work solved the problem caused by the confliction between the dynamic resource requirements of applications and the static nature of specified configuration. However, it did not discuss the scheduling issues of multiple virtual machines' computing resource.

Y. Zhang et al. [4] advocates the use of self-adaptation in the VMs themselves based on feedback about resource usage and availability. They defined a *Friendly VM* (FVM) to be a virtual machine that adjusts its demand for system resources. The objective of this research is to let the virtual machines on top of the same host machine share the underlying resources fairly, i.e., to distribute the resources among them evenly.

The objective of work [5] is to find an efficient way to manage the resource of large-scale share-memory multiprocessor hardware. In order to manage the computing resource of the underlying hardware, Cellular Disco employs a VCPU migration scheme. By this scheme, the VCPU will migrate and be re-mapped to a light-loaded physical processor when it finds that it is heavily loaded. Although Cellular Disco can achieve good load-balancing among the processing units and manage the large-scale machine well, the guest systems it hosts are statically configured. Therefore, it can not actively revise the configuration of the virtual machine to accommodate intensive workload generated by the HPC applications.

B. Scheduling Optimization for Multi-core Systems

R. A. Chowdhury et al. [6] present cache-efficient *chip multiprocessor* (CMP) algorithms with good speed-up for some widely used dynamic programming algorithms based on three types of caching systems: D-CMP with a private cache for each core, S-CMP with a single cache shared by all cores, and Multi-core, which has private L1 caches and a shared L2 cache. For each type of CMP, they specify the related parameters and then give a parallel schedule that ensures good performance with respect to both parallelism and cache efficiency.

T. Constantinous et al. [7] consider the influence on the performance of a single thread migration in various aspects, including frequency of migration, core warm-up nodes, subset of resources that are warmed-up, number of cores, and cache hierarchy organization. Their experimental results show that the performance loss due to activity migration on a multi-core with private L1s and a shared L2 can be minimized if a migrating thread continues its execution on a core that was previously visited by the thread, and cores remember their predictor state since their previous activation. S. Siddha et al. [8] use four kinds of workloads as the test suite to be run on a dual-core, dual-package SMP platform. The test results show that all the workloads benefited form distributing the load to two different packages.

All these works mentioned above focus on the shared cache architecture of the multi-core system, they either optimize the algorithms of applications to adjust to the underlying hardware architecture, or modify the scheduling algorithms of the operating systems.

III. XenMVM Architecture

In this section, we discuss the system architecture, its design issue, and the implementation.

A. System Architecture

Fig. 1 illustrates the system architecture of XenMVM. Similar to FVM, it has two main components: the performance monitor and the resource manager. The monitor is in charge of diagnosing the optimal resource requirement of the virtual machine, while the resource manager is in charge of allocating and scheduling computing resource according to the resource requirement sent by the virtual machine and the usage of underlying physical computing resources.

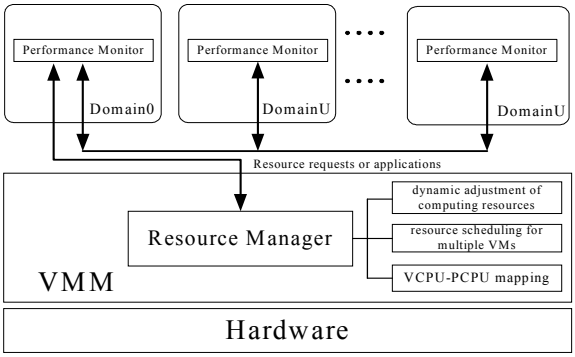


Fig.1 The system architecture of XenMVM

The resource scheduling strategies for multiple VMs are important to maximize the performance of the whole system, especially under the situation that the physical computing resources are limited. The basic unit of computing resource scheduling is VCPU. Mapping VCPUs to proper PCPUs is helpful to explore the potential performance of the underlying physical architecture.

B. Diagnosing Resource Requirement

In order to diagnose the optimal resource requirement of the virtual machine, we use the monitor module shown in Fig. 1 to gather the information of the VM, including time spent in system mode and user mode, the number of competitive threads and so forth to deduce the resource requirement: the average VCPU utilization and parallel level.

Suppose there are N virtual processors in a virtual machine, by the data sampled from */proc* file system, the utilization rate of individual virtual processor at time t can be computed by formula (1).

$$U_{VCPU}(t) = (t_{user} + t_{sys} + t_{iowait} + t_{irq} + t_{softirq}) / t_{total} \quad (1)$$

where t_{user} denotes the time spent in user mode, t_{sys} denotes the time spent in system mode, t_{iowait} denotes the time spent on waiting for the accomplishment of I/O operations, t_{irq} and $t_{softirq}$ represent the time spent on hardware interrupts and software interrupts, respectively, so t_{total} is computed by using formula (2).

$$t_{total} = t_{user} + t_{sys} + t_{iowait} + t_{irq} + t_{softirq} + t_{idle} \quad (2)$$

In this formula, t_{idle} denotes the time that the virtual processor is idle. To obviate the effects from the background processes, we do not consider the time spent on low priority processes (i.e., t_{nice}), and the time consumed by the virtual machine itself (i.e., t_{steal}).

With the definitions above, the average CPU utilization rate can be computed using formula (3).

$$U(t) = \sum_{i=1}^{N} U_{VCPU_i}(t) / N \quad (3)$$

We also use the system parallel level as one of the parameters which determine the real resource need. The system parallel level at time t is defined in this paper as the quotient of formula (4) as following.

$$P(t) = \sum_{i=1}^{N} l_{runqueue_i} / N \quad (4)$$

In this formula, $l_{runqueue_i}$ represents the length of run queue at the i^{th} virtual processor.

To smooth out the average VCPU utilization rate, *Exponentially-Weighted Moving Average* (EWMA) [4] is adopted in our system. We define the effective average VCPU utilization rate $\overline{U}(t)$ in following formula, and γ is set to be 0.2 in our experiments.

$$\overline{U}(t) = (1-\gamma) \times \overline{U}(t-1) + \gamma \times U(t) \quad (5)$$

C. Resource Adjustment

XenMVM uses Xen [2] as the VMM, and its *para-virtualization* (PV) guest system to host the multithreaded applications. To gain fine-grained control of computing resource, we control the resource of virtual machines by simply control the number of virtual processors they have, and use scheduling algorithm in the VMM (i.e., Credit) to map more or less physical processors or computing cores to the virtual processors.

By considering the resource adjustment patterns of multithreaded applications, we adopt AISD strategy.

The behaviours of applications are hard to be predicted, they may invoke multifarious processes (such as system calls, library functions) during the lifetime of applications. To solve the threshing problem caused by the process of adjustment, we propose a scheme called *Forecasting and Time-delayed Subtraction* (FTDS) scheme. Using the statistic of history resource requirements and the current computing resource utilization to assist the manager in making decisions can improve the accuracy and timeliness of resource adjustment.

As shown in Fig.2, it depicts the general control block diagram for *Forecasting and Time-delayed Subtraction* scheme used in XenMVM. The whole control process is a dynamic feedback cycle. The PID [10] algorithm consists of three parts: proportional algorithm, integral algorithm, and derivative algorithm. The proportional algorithm adjusts the output signal in direct proportion to the control input.

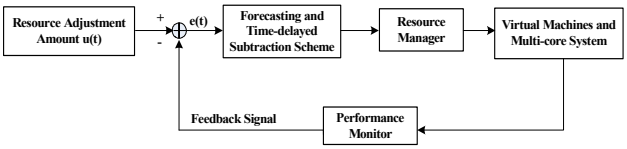


Fig.2 Block diagram for *Forecasting and Time-delayed Substraction* scheme used in XenMVM

In XenMVM, we use the performance monitor to sample the computing resource requirement as the feedback signal, and use the resource manager to adjust the computing resource of virtual machines according to resource adjustment amount allocated by the resource manager. To guarantee the performance of applications, the resource manager will satisfy the request of increasing computing resource sent by VMs as soon as possible, while delaying some time to make decisions when VMs send the requests of reducing computing resource.

IV. Scheduling Methodologies

A. Underlying Layout Aware Scheduling

The scheduling issue has a strong relationship with the system architecture. Based on the Dell server with two way processors, each of which has 2 cores, the processor is Intel Xeon CPU 5110, 1.6Hz, with 32KB L1 I-Cache, 32KB L1 D-Cache and 4MB L2 cache, and the server is configured with 2GB main memory, 160G SCSI hard disk and 1Gb Ethernet card, we make such an experiment: mapping the two threads created by the NPB programs LU.A.2 and IS.A.2 when running on the server.

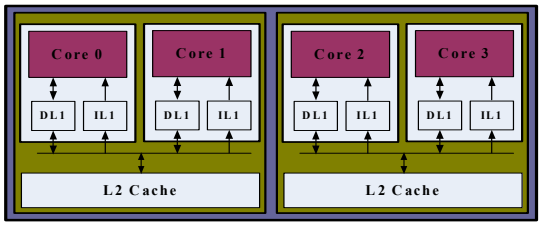
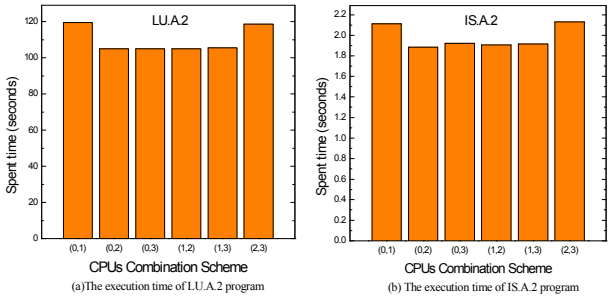


Fig.3 Multi-core system with private L1 cache and shared L2 cache

According to the */proc/cpuinfo* file, we can obtain the topology of the processors as shown in Fig.3, i.e., core 0 and 1 belong to CPU 0, and core 2 and 3 belong to CPU 1. As shown in Fig. 4, it can be observed that the performance is worse than others when we map the threads to core 0, 1 or core 2, 3. Table I shows the cache miss statistic of different

CPUs combination schemes when running NPB programs. The number of cache miss events happened when mapping the two threads to core 0, 1 and core 2, 3 is much more than other schemes. It provides sufficient evidence to help us explain the phenomena shown in Figure 4. It proves that the performance of the applications running in the multi-core system has a strong relationship with the cache architecture.



(a)The execution time of LU.A.2 program (b) The execution time of IS.A.2 program

Fig. 4 The execution time of NPB programs under different CPUs combination schemes

TABLE I THE CACHE MISS STATISTIC OF DIFFERENT CPUS COMBINATION SCHEMES WHEN RUNNING NPB PROGRAMS

CPUs Combination	L2_LINES_IN					
	LU.A.2			*IS.A.2*		
	***Samples* (×500)**	***%***	***Total* (× 500)**	***Total* (× 500)**	***%***	***Total* (× 500)**
(0, 1)	2837230	41.58	6823447	81576	64.18	127096
(0, 2)	2850570	53.22	3477142	56511	63.86	88497
(0, 3)	1844789	52.35	3523670	56422	63.50	88856
(1, 2)	1852774	53.80	3443831	56093	63.35	88547
(1, 3)	1850796	53.80	3440084	55916	62.99	88767
(2, 3)	3083905	41.22	7481411	83788	64.83	129239

We conduct further experiment on a new test bed, which is configured as in Table II. One PV (*para-virtualization*) guest domain configured with 4 VCPUs and 512 MB memory is created to host the NPB benchmark. The number of VCPUs in Domain0 is set to 2.

TABLE II THE CONFIGURATION INFORMATION OF TEST BED

CPU	Two way 1.6 GHz Intel Xeon processors, each of which has 4 cores
Memory	2 GB
Hard Disk	160 GB SCSI
Cache	32 KB L1 I-Cache, 32 KB L1 D-Cache, 4MB L2 Cache
VMM	Xen 3.4-unstable
Guest OS	Red Hat Enterprise Linux Server 5.1
Benchmark	NPB (NAS Parallel Benchmark)
Others	MPICH 1.2.7pl

Among the four schemes illustrated in Fig. 5, the first one means the VCPUs of both Dom0 and DomU are not pinned to any PCPUs. The second one means the VCPUs of Dom0 are pinned to PCPUs whose processor ids are 6 and 7, respectively. The third one means the VCPUs of DomU are pinned to PCPUs whose processor ids range from 0 to 3. The last one means all the VCPUs are pinned to the specific PCPUs whose processor ids are mentioned above. We use the standard deviation to illustrate the fluctuation of these schemes. The smaller the standard deviation is, the more stable the execution performance of job running in VM is. As shown in Fig. 5, in most cases, the performance stability is much better than others when pinning all the VCPUs to specific PCPUs.

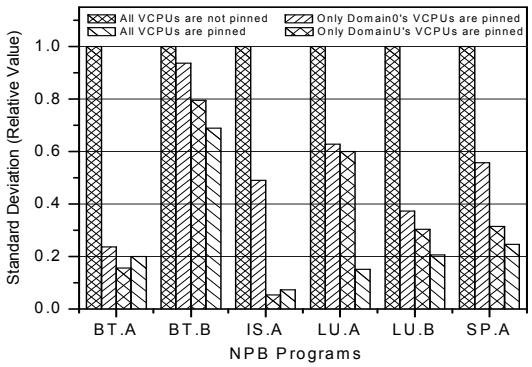


Fig. 5 The standard deviation comparison of four mapping schemes

B. Scheduling Algorithm for Multiple Virtual Machines

In this section, we discuss the computing resource scheduling schemes or algorithms for multiple virtual machines based on a single multi-core system. We simply define these as the behaviour characteristics of one virtual machine, and these behaviour characteristics will affect the scheduling strategies directly.

In XenMVM, we define a structure named *request_queue* to save all the resource requirements in special forms which includes VM ID and VCPU amount. To reduce the communication overhead, the unsatisfied VM will stop sending resource requirement in short period until there are idle PCPUs released by other VMs.

According to the behaviour characteristics of virtual machine, we propose another scheduling algorithm called domain-based static priority algorithm. Each virtual machine has a priority value which can be reset manually. When the resource manager receives several resource requirements from different VMs at the same time, the VM whose priority value is higher than others will have the priority to gain the resource allocation, and if there is no idle resource left, it will deprive resource from other VMs with lower priority until its resource request get satisfied. Previous studies have concluded that latency-sensitive applications will perform best if they are placed within their own domain and given preferential treatment [11]. In practical applications, if we place applications of different types within VMs with different priorities respectively, the whole performance of the system will be better than the mixed situation's.

V. PERFORMANCE EVALUATION

A. Experimental Setup

On the test bed defined in Table II, three PV guest domains configured with 512MB memory are created. In XenMVM, each VM will be configured to have one VCPU initially, and when it asks for more processing resource, it will always be allocated the smallest numbered computing core from the resource pool, and all the VCPUs will be pinned to appropriate PCPUs by ULAS scheme. Dom0 is configured to have two VCPUs, and are pinned to the computing cores tagged 6 and 7 respectively.

B. Case Study

In this case, we add our ULAS and domain-based static priority scheduling schemes into XenMVM. Once there are jobs starting in the VM, the performance monitor will send resource requests to the resource manager according to the workloads. Once the VM gets the computing resource it needs, all the newly added VCPUs will be pinned to appropriate PCPUs. In the following, we will simulate the actual situation to evaluate the whole performance of XenMVM.

We conduct experiments as follows. A total of three PV VMs are used in each experiment, with each VM hosting three jobs whose execution time rang from 16s to 56s in non-virtualized environment. The configuration of each VM is the same as that mentioned above.

As shown in Fig. 6, it illustrates the scenario that there are multiple jobs running in the multiple virtual machines environment. The interval between each two jobs is randomized. In this scenario, we adopt the FCFS strategy to allocate and schedule computing resources for these jobs. Because there is no overlapping phenomena happens there, that is to say, no two jobs start running at the same time, each job can get adequate computing resource without competing with other VMs. Each job can gain the best execution performance, and the turnover time of the whole system is 29.24s.

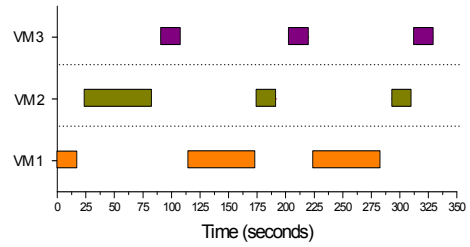


Fig. 6 The sequence diagram of multiple jobs running in the multiple VMs environment using FCFS scheduling scheme

Fig. 7 shows a similar scenario but the interval between two jobs is shortened in order to simulate the overlapping situation which often occurs in real applications. The shaded part in Fig. 7 indicates the performance loss caused by the overlapping phenomena. The turn-over time of the whole system is 48.02s, which indicates the system performance decrease by 64.23% because of the resource competition between VMs.

Based on the scenario mentioned above, we adopt the static priority strategy and set the priority of VM-1 to be higher than other twos. Fig. 8 shows the performance results of multiple jobs running in the multiple VMs using static priority scheduling scheme. The turnover time of the whole system is 38.20s, which improve the performance by 20.45% compared with FCFS scheduling scheme.

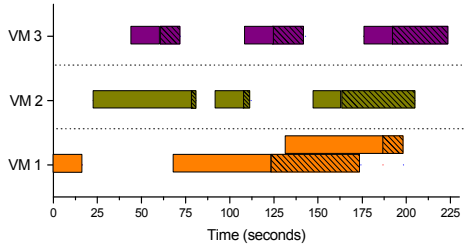


Fig. 7 The sequence diagram of multiple jobs running in the multiple VMs environment with overlapping phenomenon

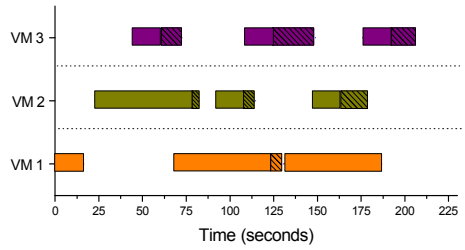


Fig. 8 The sequence diagram of multiple jobs running in the multiple VMs environment using static priority scheduling scheme

VI. Conclusions and Future Work

In this paper, we propose a computing resource management system scheme named XenMVM and present its prototype implementation and show that it can adjust the computing resource dynamically according to the actual workload generated by the applications running in VMs.

We propose several computing resource scheduling schemes to explore the potential performance of multi-core system and improve the whole performance of multiple VMs. Using NPB as the test suite, we show that the ULAS scheme can improve the performance at about 4.5%~32.52% on the platform of multi-core system. Finally, we study the computing resource scheduling issue of multiple VMs, our work demonstrates that using domain-based static priority scheduling scheme can improve the throughput of the whole system by 20.45% compared with FCFS scheme.

Acknowledgment

This work is supported by National 973 Basic Research Program of China under grant No.2007CB310900, National Natural Science Foundation of China under grant No.60903022.

References

[1] Z. Shao, H. Jin, Y. Li, and Jian Huang, “Virtual Machine Resource Management for High Performance Computing Applications”, *Proceedings of 2008 International Workshop on Virtualization Technology (IWVT’08) with ISCA’08*, June 21, 2008, Beijing, China.

[2] 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”, *Proceedings of 19th ACM Symposium on Operating Systems Principles*, October 2003, pp.164-177.

[3] P. Shivam, A. Demberel, P. Gunda, D. Irwin, L. Grit, A. Yumerefendi, S. Babu, and J. Chase, "Automated and on-demand provisioning of virtual machines for database applications", *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, Beijing, China, pp.1079-1081.

[4] Y. Zhang, A. Bestavros, M. Guirguis, I. Matta, and R. West, "Friendly Virtual Machines: Leveraging a Feedback-Control Model for Application Adaptation", *Proceedings of the 1st ACM/USENIX Conference on Virtual Execution Environments (VEE'05)*, June 2005, Chicago, Illinois, pp.2-12.

[5] K. Govil, D. Teodosiu, Y. Huang, and M. Rosenblum, "Cellular disco: resource management using virtual clusters on shared-memory multiprocessors", *ACM Transactions on Computer Systems (TOCS)*, 2000, 18(3): 229-262.

[6] R. A. Chowdhury and V. Ramachandran, "Cache-efficient Dynamic Programming Algorithms for Mulicores", *Proceedings of SPAA'08*, June 14-16, 2008, Munich, Germany, pp.207-216.

[7] T. Constantinous, Y. Sazeides, P. Michaud, D. Fetis, and A. Seznec, "Performance Implications of Single Thread Migration on a Chip Mulit-Core", *ACM SIGARCH Computer Architecture News*, September 2005, 33(4): 80-91.

[8] S. Siddha, V. Pallipadi, and A. Mallick, "Process Scheduling Challenges in the Era of Multi-core Processors", *Intel Technology Journal*, 2007, 11(4): 361-370.

[9] NASA. NAS Parallel Benchmarks. http://www.nas.nasa.gov/Software/NPB/

[10] M. J. Willis, *Proportional-Integral-Derivative Control*, University of Newcastle.

[11] D. Ongaro, A. L. Cox, and S. Rixner, "Scheduling I/O in Virtual Machine Monitors", *Proceedings of VEE'08*, March 5-7, 2008, Seattle, Washington, USA.