Energy-Efficient Virtual Machine Scheduling in Performance-Asymmetric Multi-Core Architectures

Yefu Wang[1], Xiaorui Wang[1,2], and Yuan Chen[3]
[1]University of Tennessee, Knoxville [2]The Ohio State University [3]HP Laboratories

***Abstract*—Multi-core architectures with asymmetric core performance have recently shown great promise, because applications with different needs can benefit from either the high performance of a fast core or the high parallelism and power efficiency of a group of slow cores. This performance heterogeneity can be particularly beneficial to applications running in virtual machines (VMs) on virtualized servers, which often have different needs and exhibit different performance and power characteristics. Therefore, scheduling VMs on performance-asymmetric multi-core architectures can have a great impact on a system's overall energy efficiency. Unfortunately, existing VM managers, such as Xen, have not taken the heterogeneity into account and thus often result in low energy efficiencies.**

In this paper, we propose a novel VM scheduling algorithm that exploits core performance heterogeneity to optimize the overall system energy efficiency. We first introduce a metric termed energy-efficiency factor to characterize the power and performance behaviors of the applications hosted by VMs on different cores. We then present a method to dynamically estimate the VM's energy-efficiency factors and then map the VMs to heterogeneous cores, such that the energy efficiency of the entire system is maximized. We implement the proposed algorithm in Xen and evaluate it with standard benchmarks on a real testbed. The experimental results show that our solution improves the system energy efficiency (*i.e.*, performance per watt) by 13.5% on average and up to 55% for some benchmarks, compared to the default Xen scheduler.

I. Introduction

The design of multi-core processors with identical cores faces an inevitable dilemma: using only slow cores may compromise the performance of single-threaded applications, while using only fast cores may unnecessarily lead to low energy efficiency. As a result, multi-core architectures with asymmetric core performance (*i.e.*, a mix of fast and slow cores) have recently shown great promise, because they can offer applications the flexibility to run on a fast core for high single-threaded performance, or on a group of slow cores for high parallelism with better energy efficiency [4, 7, 9, 17, 18]. Multi-core systems with performance heterogeneity can be beneficial to today's high-performance computer servers that commonly host virtual machines (VMs) on multi-core architectures, because VMs hosting different applications often have different resource needs. Therefore, they have different preferences on the type of cores they run from an energy-efficiency perspective. Unfortunately, existing VM scheduling algorithms, such as Credit Scheduler in Xen, map VMs to the cores without considering the heterogeneities of the cores [2, 3, 6, 12, 25] which can lead to degraded overall energy efficiency.

There are two challenges in developing an energy-efficient solution for scheduling VMs on heterogeneous cores with performance asymmetry, such as cores with different CPU frequencies. First, it's difficult to characterize the application needs and resource usage. In many virtualized platforms, such as Amazon EC2, the platform administrators have no knowledge of the applications running inside the VMs because the users of the VMs may run any application at any time. Thus, technologies requiring accurate knowledge of the applications (*e.g.*, [18]) are impractical from a VM scheduling perspective. Second, in order to achieve better energy efficiency, both performance and power need to be considered. Mapping VMs to cores solely based on the performance metrics (*e.g.*, [7, 17, 18]) may lead to unnecessarily poor overall system energy efficiency.

In general, heterogeneous multi-core systems fall into two categories: multi-core systems containing heterogeneous cores with different special purpose functions (function asymmetry) and systems containing cores with homogeneous functions but heterogeneous performance and power characteristics (performance asymmetry) [7]. In this paper, we focus on performance asymmetry and particularly multiple cores with heterogeneous CPU frequencies, because they are technically mature and widely used. For example, many current designs, such as Intel's Single-chip Cloud Computer (SCC)[5], allow cores in a group to have a frequency different from those in other groups. To exploit these group-level or core-level frequency scaling technologies, many recently proposed power management solutions [1, 10, 11, 14, 19, 22] dynamically run cores at different frequencies, making the systems behave like multi-core systems with performance asymmetry.

In this paper, we present a practical and effective solution that schedules VMs on heterogeneous cores with different frequencies to maximize the overall energy efficiency of the system. We first introduce a metric termed *energy-efficiency factor* to characterize the power and performance behaviors of a virtual CPU (VCPU) of a VM on different cores. We then develop a light-weight method for dynamically estimating the energy-efficiency factors of VCPUs using hardware performance counters which are widely available in most multi-core processors. We then propose a scheduling algorithm that maps the VCPUs to different types of cores based on their estimated energy-efficiency factors, in such a way that the overall energy efficiency (*i.e.*, performance per watt) is maximized. Specially, our algorithm provides better improvement

This work was supported, in part, by NSF grants CNS-0915959, CNS-1143605, CNS-1143607 (NSF CAREER Award), and by ONR grant N00014-09-1-0750.

in energy efficiency when the application runs into phases with more CPU usages.

This paper makes the following contributions:

- We introduce energy-efficiency factor, a metric to characterize the power and performance behaviors of a VCPU on different cores and propose a statistical method to estimate the energy-efficiency factors of VCPUs online.
- We propose an VM scheduling algorithm that maps VMs to heterogeneous cores with different CPU frequencies to maximize a system's overall energy efficiency. Our algorithm can be applied directly to existing multi-core hardware and is completely transparent to the users.
- We present a method that integrates our VM scheduling algorithm, which requires no application-level knowledge of the VMs, with an application-level performance management solution to improve the application-level energy efficiency.
- We implement our scheduling algorithm in the Xen VM hypervisor on top of an existing VM scheduling algorithm and evaluate our algorithm on a real testbed using various standard benchmarks, such as SPEC CPU2006, SPECjbb2005, and RUBBoS. The results demonstrate the our algorithm effectively improves the energy efficiency over the default scheduler of Xen.

II. Background and Motivations

In this section, we first describe the energy efficiency metric used in our work. We then discuss CPU scheduling in Xen and use an example to show how the existing scheduling algorithm may result in poor energy efficiency.

A. Energy Efficiency

The main objective of the work is to optimize energy efficiency. Performance per watt has been widely used to measure the energy efficiency. Since our work is focused on low level VMs scheduling and has no application level knowledge, we use the number of instructions per second as the performance metric:

$$EnergyEfficiency = Performance/Power \qquad (1)$$

where $Performance$ denotes the throughput of a computer server in Billions of Instructions Per Second (BIPS) and $Power$ is the power consumption of the whole system. Note that the energy efficiency in terms of BIPS per watt is equivalent to Billion of Instructions per Joule. Hence, maximizing the energy efficiency in (1) also means minimizing the energy consumption in Joule required to complete a predetermined number of instructions.

The focus of this paper is on designing an energy efficient scheduling algorithm. A VM scheduling algorithm, such as the one proposed in this paper, typically has no knowledge of the application-level performance metric, *e.g.*, the throughput of web requests. Thus it is impractical to optimize the application-level performance directly. Furthermore, the lack of application-level performance knowledge makes it difficult for a scheduling algorithm to make compromises between power consumption and performance. Therefore, we make a design choice that the proposed algorithm does not directly change the core configurations, *e.g.*, the frequency levels of the cores.

With the objective of optimizing the energy efficiency, our algorithm can be applied in many systems to improve the energy efficiency of virtualized computers. For example, our algorithm can be integrated with other application-level management algorithms to achieve an energy-efficient management solution. We present a detailed example of the integration in Section IV-B.

B. CPU Scheduling in Xen

In Xen, each VM has one or more Virtual CPUs (VCPUs) when it is created. These VCPUs are the CPUs visible to the guest operating systems. Xen uses CPU schedulers to run VCPUs on physical cores. The VMs' weight and cap parameters determine the way how VCPUs' priority levels change as they consume CPU time slices. Credit Scheduler usually uses two levels of priorities to indicate whether a VCPU has exceeded its portion of CPU time defined by the weight and cap of the corresponding VM. In this paper we refer to the two priorities of VCPUs as **high** (the portion of CPU time has not been exceeded) and **low** (the portion of CPU time has already been exceeded).

Each core has a local run queue of VCPUs that are scheduled to run on the core and has 0 or 1 VCPUs currently running on the core. When the currently running VCPU blocks, yields, or completes its time slice, the core looks at the local run queue. If the local run queue is empty, or all VCPUs in the queue have low priority levels, the core looks for high priority VCPUs from queues of other cores to do load balancing. Load balancing guarantees that in the entire system, low-priority VCPUs will not run while any high-priority VCPUs are still waiting in the run queues. In this way, VCPUs with high priorities are balanced among the cores.

VCPU	On fast cores		On slow cores	
	Power	Performance	Power	Performance
VCPU 1	27.2W	0.303 BIPS	10.9W	0.111 BIPS
VCPU 2	49.0W	0.247 BIPS	19.6W	0.073 BIPS
VCPU 3	61.8W	0.121 BIPS	24.7W	0.031 BIPS

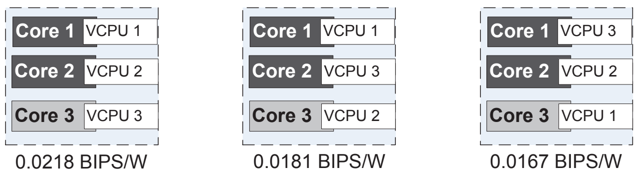

Fig. 1. The energy efficiency of the system can be improved by selecting an energy-efficient VCPU-core mapping.

C. Motivation Example

We use an example system consisting of two fast cores and one slow core to illustrate the motivation to design an energy efficient scheduler. In this example system, Core 1 and Core 2 are fast cores and Core 3 is slow. The power and performance behaviors of the three running VCPUs are listed in Figure

1. The power consumption of the components other than the CPU cores is 165.7 W. These numbers are obtained from a real system in our testbed. We study two cases.

Case 1: VCPUs 1, 2, and 3 are running on Cores 1, 2, and 3, respectively. The power efficiency defined in (1) is: $\frac{0.303+0.247+0.031}{27.2+49.0+24.7+165.7} = 0.00218$ BIPS per watt.

Case 2: In the same system, however, if VCPUs 1, 2, and 3 run on Cores 1, 3, and 2, respectively, the power efficiency becomes: $\frac{0.303+0.073+0.121}{27.2+19.6+61.8+165.7} = 0.00181$ BIPS per watt.

The observation in this example is that mapping VMs with different performance and power characteristics to heterogeneous cores will have an impact on the overall energy efficiency of the system. In the default Xen Credit Scheduler, VCPUs are moved between different cores based only on their priorities without considering their power and performance behaviors. As a result, it is highly possible that the VCPUs are mapped to cores in an energy-inefficient way.

D. Performance and Power Models

As discussed in Section I, we focus on multiple cores with heterogeneous CPU frequencies. The dynamic power of a core, *i.e.*, the part of the power that changes with the activities and type of the core, can be modeled as

$$Pd_i = a_i f_i \tag{2}$$

where f_i is the frequency of the i^{th} core and a_i is a parameter related to the activities of the VCPU [13]. Note that if the processor implements dynamic voltage and frequency scaling (DVFS) for individual cores, the change in the frequency and voltage of a core has a cubic impact on the dynamic power [16]. However, the linear model in (2) still provides an acceptable approximation because real processors only have a limited frequency range, which is evaluated in [22].

Therefore, the power consumption of the whole system with n active cores can be presented as

$$Power = a_1 f_1 + a_2 f_2 + \cdots + a_n f_n + B \tag{3}$$

where B is the power consumption of all components other than the CPU cores in the system.

The performance (BIPS) of a VCPU when it is running on a core with a frequency of f_i can be modeled as

$$Perf_i = c_i f_i \tag{4}$$

where c_i is the value of the Instructions per Cycle (IPC) of the VCPU. Moreover, we assume that c_i is not a function of the frequency. In reality, IPC may slightly change with the frequency because of off-chip accesses. However, this dependency is typically negligible because the change in an application thread's IPC when the frequency changes is typically smaller than the IPC change across different application threads [19].

III. Energy-Efficiency Factor

As shown in the example in Section II-C, the energy efficiency of the system can be improved by scheduling the VCPUs based on the detailed knowledge of their performance and power characteristics shown in Figure 1. However, it is difficult to obtain such parameters in practice. First, we need to know the applications running inside each VCPU, which is often impractical. Second, we need to conduct a comprehensive offline profiling to measure the characteristics of each application on each type of cores. However, core-level power measurement is difficult in hardware implementation and is not widely available in today's hardware. Further, such a profiling may involve a large number of applications and types of cores and make it a very time consumpting task. Finally, the power and performance behaviors of the applications often show significant variations from time to time. Thus, the static performance and power characteristics from offline profiling cannot be applied for the entire life cycle.

In this section, we introduce energy-efficiency factor, a metric to characterize the power and performance behaviors of a VCPU on different cores. We show this simple metric can well characterize how efficiently a VCPU runs on different types of CPU cores and be used to guide VCPU-core mapping to optimize the energy efficiency. We further propose an approach to estimating the energy-efficiency factor using the hardware performance counters in the cores.

A. Definition of Energy-Efficiency Factor

Definition 1. ***energy-efficiency factor:*** *We define θ_i in (6) as the energy-efficiency factor of a VCPU i.*

We now show that for a two core system (*i.e.*, a slow core and a fast core), scheduling the VCPU with a higher energy-efficiency factor on the fast core achieves better energy efficiency. We consider a system with two heterogeneous cores: Core 1, a fast core with frequency f_1, and Core 2, a slow core with frequency f_2, ($f_1 > f_2$), and two VCPUs: VCPU 1 and VCPU 2 with different power and performance characteristics. For two different VCPU-core mappings, we calculate their energy efficiency as follows.

Mapping 1: VCPU 1 on Core 1 and VCPU 2 on Core 2. The power consumption of the entire system is $a_1 f_1 + a_2 f_2 + B$. The performance is $c_1 f_1 + c_2 f_2$. Hence, the energy efficiency as defined in Section II-A is $\frac{c1f1+c2f2}{a1f1+a2f2+B}$.

Mapping 2: VCPU 1 on Core 2 and VCPU 2 on Core 1. Similarly, the energy efficiency is $\frac{c1f2+c2f1}{a2f1+a1f2+B}$.

Now we prove that Mapping 1 is more energy-efficient than Mapping 2, that is

$$\frac{c_1 f_1 + c_2 f_2}{a_1 f_1 + a_2 f_2 + B} > \frac{c_2 f_1 + c_1 f_2}{a_2 f_1 + a_1 f_2 + B} \tag{5}$$

if and only if

$$\theta_1 > \theta_2$$

where

$$\theta_i = \frac{c_i}{a_i(f_1 + f_2) + B}. \tag{6}$$

Proof: (5) holds if and only if:

$$(c_1 f_1 + c_2 f_2)(a_2 f_1 + a_1 f_2 + B) > (c_2 f_1 + c_1 f_2)(a_1 f_1 + a_2 f_2 + B)$$

$$c_1(f_1 a_2 f_1 + f_1 B - f_2 a_2 f_2 - f_2 B) > \\ c_2(f_1 a_1 f_1 + f_1 B - f_2 a_1 f_2 - f_2 B)$$

$$c_1(f_1 - f_2)(a_2(f_1 + f_2) + B) > \\ c_2(f_1 - f_2)(a_1(f_1 + f_2) + B) \tag{7}$$

Because $f_1 > f_2$, (7) holds if and only if

$$\frac{c_1}{a_1(f_1 + f_2) + B} > \frac{c_2}{a_2(f_1 + f_2) + B} \tag{8}$$

□

From the above discussion, we conclude that, for a two-core system, mapping a VCPU with a higher energy-efficiency factor (*e.g.*, VCPU 1) on a fast core (*e.g.*, Core 1) leads to improved energy efficiency.

B. Online Estimation of energy-efficiency factor

Though it is possible to use the performance counters to estimate the values of a_i, c_i, and B in (6) and then calculate the energy efficiency factor, we choose to build a model to directly estimate the energy efficiency factor from the performance counters. This is because that due to the limited number of available performance counters in the modern processors, it is difficult to estimate multiple metrics simultaneously.

Our framework for estimating the energy-efficiency factor has offline and online parts. The offline part is performed with workloads in a *Training Set* and the online part is designed to provide estimations for the energy-efficiency factors for the runtime workloads which are typically unknown at design time. Note that the offline analysis is performed only once with a set of typical workloads in a training set to build a model.

1) Offline Analysis on Workloads in a Training Set: The offline part provides an energy efficiency estimation model for the online part as follows.

We randomly select 8 benchmarks in SPEC CPU 2006 , including *gcc*, *gobmk*, *bzip2*, *hmmer*, *astar*, *h264ref*, *omnetpp* and *sjeng*, as our training set to cover a large variety of CPU instruction combinations. We We collect data for training as follows. For each benchmark, we perform several experiments. In each experiment we run 12 copies of the same benchmark in 12 VMs on the 12 cores of the system. All cores are set to the same frequency level in each experiment and we repeat the experiments using all frequency levels. In each experiment, we collect 22 performance events listed in an extended version of this paper [24] and measure the average power consumption using a power meter. Using the collected training data, for each VCPU, we measure its *performance event rates* (*i.e.*, performance event counts per instruction) and energy-efficiency factor. The energy-efficiency factor of each VCPU is calculated based on (6) where the variables are inferred using a standard curve fitting technique in the power and performance models, (2) and (4).

We then use linear regression to build a linear model to approximate the energy-efficiency factor of a VCPU

$$\theta = \sum_{i=1}^{m} \alpha_i r_i + \alpha_0 + \beta f \tag{9}$$

where r_i is the event rate of the i^{th} hardware performance event. α_i and β are the constant parameters derived from linear regression. The integer m is the number of performance event rates that are used in our model. f is the frequency of the core on which the VCPU is running. More details are presented in an extended version of this paper [24]. This linear estimation model (9) only adds a small computational overhead ($m + 1$ multiplication operations and $m + 2$ addition operations) to the scheduling algorithm.

2) Online Estimation for Unknown Workloads: When working online, we treat every VCPU as an unknown VCPU. Each time the VCPU changes from the running state to another state, we measure its performance event rates in the time slice it runs. We next estimate its energy-efficiency factor using the model in (9). Note that the model cannot be dynamically updated online because CPUs are not always equipped with power sensors.

IV. Energy-Efficient VM Scheduling

Based on the conclusion in Section III-A, our goal is to dynamically keep the VCPUs with a greater energy-efficiency factor running on faster cores. In this section, we first assume that the system only has two types of cores: fast cores and slow cores indicating which cores have higher or lower frequencies. We then introduce how to extend our technique to more frequency levels.

A. Algorithm

Our algorithm inherits the majority of the Credit Scheduler and only uses a different strategy for load balancing. We do not move VCPUs among cores when the VCPUs are currently running because moving a running VCPU incurs a large overhead including context switches and cache warm up. Because load balancing operations only move VCPUs that are waiting for CPU cores to idle cores, the overhead, especially the context switch overhead, is significantly lower than moving running VCPUs.

Before load balancing, a threshold of the energy-efficiency factor is calculated based on the running VCPUs. The threshold is defined as the H^{th} highest energy-efficiency factors among all the running VCPUs where H is the number of fast cores. When a slow core performs load balancing, *i.e.*, when it fails to find a high-priority VCPU on its local run queue, it searches for one on other cores, beginning with the fast cores. Preference is given to the VCPUs with an energy-efficiency factor lower than the threshold calculated before load balancing. Similarly, when a fast core performs load balancing, it searches the run queues of cores for a VCPU, starting with the slow cores. Preference is given to the VCPUs with an energy-efficiency factor higher than the threshold.

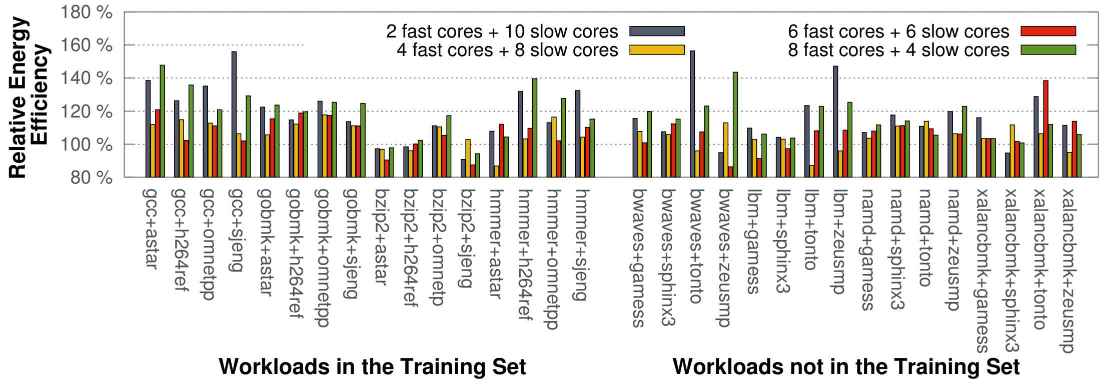

Fig. 2. Relative energy efficiency of our energy-efficient algorithm and the default Xen Credit Scheduler in 4 sets of core frequency settings.

B. Integration with an Application-Level Performance Management Algorithm

We now discuss an example way of integrating the proposed VM scheduling algorithm with an application-level performance balancing solution as follows. We integrate our algorithm with a simple but effective application-level performance management solution that checks and balances the application-level progress of all VMs in every minute. Note that our algorithm is transparent to the application-level performance management algorithm. However, the application-level algorithm affects the scheduling decisions of our algorithm when adjusting the weight parameter in the management interface of the scheduler because the weight parameter determines the priority of the VCPUs. For the VMs with faster progress than the average, the management algorithm decreases the corresponding weight parameter by a level of 40. As discussed in Section II-B, decreasing the weight parameter means to decrease the portion of CPU time the VM is allowed to consume. Likewise, for the VMs with slower progress than the average, the algorithm increases the corresponding weight parameters. Thus, this application-level management algorithm balances the progress of the VMs by shifting allocated CPU time from the VMs with a faster progress to those with a slower progress to minimize the overall execution time of all applications.

V. Evaluation

Our algorithm is implemented in Xen hypervisor and evaluated in a hardware testbed. The implementation details are not presented here due to page limitations, but are available in an extended version of this paper [24].

A. Improvement in Energy Efficiency

In our first set of experiments, we select 2 out the 8 benchmarks in our training set as different combinations to test heterogeneous workloads. We then do the same to 8 other benchmarks not in our training set to compose a different group of heterogeneous workloads. For each of the two benchmarks in a combination, we run 12 independent copies in 12 VMs (*i.e.*, 24 VMs in total). Each VM is allocated with one VCPU. The experiments are conducted in 4 different configurations of fast cores (2 GHz) and slow cores (0.8 GHz).

Figure 2 presents the energy efficiency (*i.e.*, performance per watt) of the proposed algorithm relative to the energy efficiency of the default Credit Scheduler. In 54 out of the 64 experiments conducted to the benchmarks in our training set, as shown in Figure 2 (left side), the proposed algorithm outperforms the baseline, *i.e.*, the relative energy efficiency is greater than 100%. On average, the proposed algorithm improves the energy efficiency by 13.5%. In the other 10 experiments, the proposed algorithm shows a worse energy efficiency than the default Credit Scheduler. Our improvement in energy efficiency over the baseline is due to the reason that in the load balancing operations, our algorithm always drives the system to a state such that VCPUs with high energy-efficiency factors run on fast cores, thus improves the energy efficiency in load balancing. In contrast, the baseline moves VCPUs between cores in load-balancing operations without considering the behaviors of the VCPUs and thus results in degraded energy efficiency. Similarly, in 56 out of the 64 experiments conducted to the benchmarks not in our training set, the proposed algorithm outperforms Credit Scheduler. On average, our algorithm improves the energy efficiency by 10.2%. This set of experiments shows that, although our algorithm requires building a model using the workloads in a training set, the algorithm works with other workloads without rebuilding a model.

In Figure 3, we randomly select 24 benchmarks from SPEC CPU 2006 and run them in our 24 VMs. In 4 experiments, we run 2, 4, 6 and 8 out of the 12 total cores as fast cores, respectively. In all these 4 experiments, our algorithm shows a better energy efficiency compared with the default Credit Scheduler. The average improvement is 7.3%.

Because virtualized platforms are often used in web hosting applications, we evaluate the proposed algorithm in server workloads: a PHP implementation of RUBBoS and a java-based benchmark SPECjbb2005. We build 9 VMs. Each VM is allocated with 6 VCPUs. The first two VMs run SPECjbb2005 and the other VMs run RUBBoS. In 4 experiments, we run 2, 4, 6 and 8 out of the 12 total cores as fast cores, respectively. Figure 4 shows the energy efficiency of the proposed algorithm relative to the default Credit Scheduler. On average, the proposed algorithm improves the energy efficiency by 9.5%.

Further, we test the example of integrating the proposed

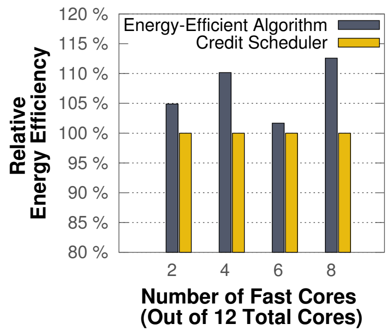

Fig. 3. Relative energy efficiency of proposed energy-efficient algorithm and the default Credit Scheduler running a mix of 24 different benchmarks.

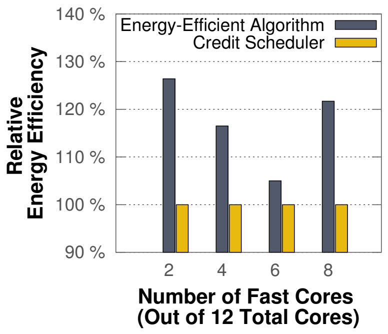

Fig. 4. Relative energy efficiency of proposed energy-efficient algorithm (relative to that of the default Credit Scheduler) running web workloads (RUBBoS and SPECjbb).

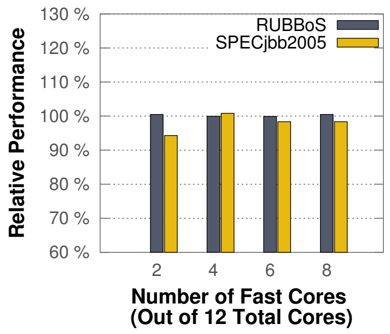

Fig. 5. The performance of the proposed energy-efficient algorithm and the default Credit Scheduler running web workloads (RUBBoS and SPECjbb).

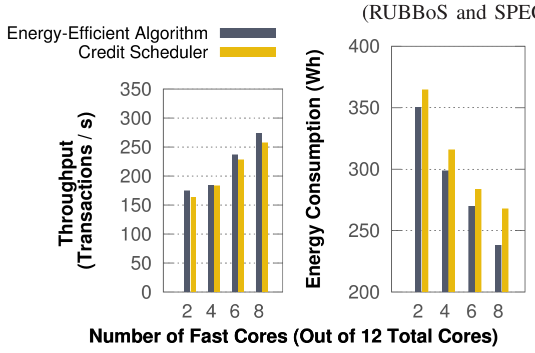

Fig. 6. Throughput and energy consumption for 1 million request per application.

energy-efficient algorithm with an application-level performance management algorithm presented in Section IV-B. We build 4 VMs, each allocated with 9 VCPUs, and run SPECjbb in 2 of them and run RUBBoS in the other 2 VMs. Figure 6 compares the application-level performance (in throughput) and energy consumption of the proposed algorithm against the default Credit Scheduler. In all these experiments, our algorithm shows a higher throughput (4.1% on average) and lower energy consumption (6.3% on average). During the entire execution time of the applications, the scheduling algorithm moves the VCPUs among the cores to maximize the overall energy efficiency in every load balancing operation, hence minimizing the overall energy consumption required to execute the set of VCPUs with a predetermined number of instructions. In contrast, the Credit Scheduler moves VCPUs among VMs without energy efficiency considerations.

VI. Related Work

VM scheduling is one of the most important problems in virtualization research. Some prior work also attempt to improve the default Xen schedulers [2] for better performance, including scheduling VMs based on the priorities of the processes inside the VMs [6], based on the number of network packages [3], and based on the number of pending I/O requests[12]. In contrast to these schedulers, our algorithm improves the energy efficiency in heterogeneous multi-core systems. The Credit Scheduler in Xen has been recently improved by [25] to first saturate a package on the chip while leaving others idle, so that lower power states can be used. Our algorithm is complementary to this technique by further improving the energy efficiency of the non-idle packages on the chip. Energy efficiency for virtualized servers has been research extensively (e.g., [20, 21, 23]), but those studies are not designed for virtual machine scheduling.

For non-virtualization environments, several previous studies investigate CPU scheduling algorithms that improve the performance of web servers. A survey of these papers is available in [15]. Compared with these techniques, our algorithm requires no knowledge of the application-level details, thus is more transparent to the users. Gupta et al. [4] demonstrated discussed the benefits of using performance asymmetric chips to host web applications. Several prior papers propose thread scheduling algorithms for heterogeneous multi-core systems in non-virtualized operating systems to improve the overall performance [1, 7, 9, 17, 18]. Our algorithm is completely transparent and requires no knowledge of the applications running inside the VMs while the solutions in these papers require prior knowledge of the applications from offline profiling. For example, [7] requires knowing a stall threshold calibrated from the workloads. As discussed in Section I, a VM scheduler should not make any assumptions on the workloads as the system administrator may not know the VMs that will run in the system. Furthermore, Kumar et al. [8] use simulation to show the potential energy benefits in dynamically switching workloads among heterogeneous cores based on the online measurement of power and performance behaviors. Compared with their work, our algorithm does not rely on an online power sensor.

VII. Conclusions

In this paper, we have presented a novel algorithm that schedules VMs on multi-core architectures with performance heterogeneity to optimize the overall system energy efficiency. We first introduce a metric called energy-efficiency factor to characterize the power and performance behaviors of the application of each VM on different cores. Our scheduling algorithm dynamically estimates the VMs' energy-efficiency factors based on available hardware performance counters and then maps the VMs to the cores, such that the energy efficiency of the entire system is maximized in a light-weight way.

References

[1] J. Chen and L. K. John. Efficient program scheduling for heterogeneous multi-core processors. In *DAC*, 2009.

[2] L. Cherkasova, D. Gupta, and A. Vahdat. Comparison of the three cpu schedulers in xen. *SIGMETRICS Perform. Eval. Rev.*, 35(2):42–51, 2007.

[3] S. Govindan, A. R. Nath, A. Das, B. Urgaonkar, and A. Sivasubramaniam. Xen and co.: communication-aware cpu scheduling for consolidated xen-based hosting platforms. In *VEE*, 2007.

[4] V. Gupta and R. Nathuji. Analyzing performance asymmetric multicore processors for latency sensitive datacenter applications. In *HotPower*, 2010.

[5] Intel Labs. *The SCC Platform Overview Revision 0.7*. Intel Labs, 2010.

[6] D. Kim, H. Kim, M. Jeon, E. Seo, and J. Lee. Guest-aware priority-based virtual machine scheduling for highly consolidated server. In *Euro-Par*, 2008.

[7] D. Koufaty, D. Reddy, and S. Hahn. Bias scheduling in heterogeneous multi-core architectures. In *EuroSys*, 2010.

[8] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen. Single-ISA heterogeneous multi-core architectures: The potential for processor power reduction. In *Micro*, 2003.

[9] V. Kumar and A. Fedorova. Towards better performance per watt in virtual environments on asymmetric single-isa multi-core systems. *SIGOPS Oper. Syst. Rev.*, 43:105–109, July 2009.

[10] K. Ma, X. Li, M. Chen, and X. Wang. Scalable power control for many-core architectures running multi-threaded applications. In *ISCA*, 2011.

[11] K. Ma and X. Wang. PGCapping: Exploiting power gating for power capping and core lifetime balancing in cmps. In *PACT*, 2012.

[12] D. Ongaro, A. L. Cox, and S. Rixner. Scheduling I/O in virtual machine monitors. In *VEE*, 2008.

[13] M. D. Powell, A. Biswas, J. S. Emer, S. S. Mukherjee, B. R. Sheikh, and S. M. Yardi. CAMP: A technique to estimate per-structure power at run-time using a few simple parameters. In *HPCA*, 2009.

[14] K. K. Rangan, G.-Y. Wei, and D. Brooks. Thread motion: fine-grained power management for multi-core systems. In *ISCA*, 2009.

[15] P. Rong and M. Pedram. Battery-aware power management based on markovian decision processes. In *ICCAD*, 2002.

[16] E. Rotem, A. Mendelson, R. Ginosar, and U. Weiser. Multiple clock and voltage domains for chip multi processors. In *Micro*, 2009.

[17] J. C. Saez, M. Prieto, A. Fedorova, and S. Blagodurov. A comprehensive scheduler for asymmetric multicore systems. In *Proceedings of the 5th European conference on Computer systems*, EuroSys, 2010.

[18] D. Shelepov, J. C. Saez Alcaide, S. Jeffery, A. Fedorova, N. Perez, Z. F. Huang, S. Blagodurov, and V. Kumar. HASS: a scheduler for heterogeneous multicore systems. *SIGOPS Oper. Syst. Rev.*, 43:66–75, April 2009.

[19] R. Teodorescu and J. Torrellas. Variation-aware application scheduling and power management for chip multiprocessors. In *ISCA*, 2008.

[20] X. Wang and Y. Wang. Co-Con: Coordinated control of power and application performance for virtualized server clusters. In *IWQoS*, 2009.

[21] Y. Wang, R. Deaver, and X. Wang. Virtual Batching: Request batching for energy conservation in virtualized servers. In *IWQoS*, 2010.

[22] Y. Wang, K. Ma, and X. Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. In *ISCA*, 2009.

[23] Y. Wang, X. Wang, M. Chen, and X. Zhu. Power-efficient response time guarantees for virtualized enterprise servers. In *RTSS*, 2008.

[24] Y. Wang, X. Wang, and Y. Chen. Energy-efficient virtual machine scheduling in performance-asymmetric multi-core architectures, Tech Report. http://wangyefu.com/sched.pdf, 2011.

[25] G. Wei, J. Liu, J. Xu, G. Lu, K. Yu, and K. Tian. The on-going evolutions of power management in xen. In *Xen Summit*, 2009.