TR-IIS-15-003

An Asymmetry-aware Energy-efficient Hypervisor Scheduling Policy for Asymmetric Multi-core

Ching-Chi Lin, You-Cheng Syu, Yi-Chung Chen, Jan-Jan Wu, Pangfeng Liu, Po-Wen Cheng, and Wei-Te Hsu

Apr. 02, 2015 || Technical Report No. TR-IIS-15-003
http://www.iis.sinica.edu.tw/page/library/TechReport/tr2015/tr15.html

An Asymmetry-aware Energy-efficient Hypervisor Scheduling Policy for Asymmetric Multi-core

Ching-Chi Lin[1][2], You-Cheng Syu[1], Yi-Chung Chen[1][2], Jan-Jan Wu[2], Pangfeng Liu[1], Po-Wen Cheng[3], and Wei-Te Hsu[3]

[1] Department of Computer Science and Information Engineering
National Taiwan University
{deathsimon,arosusti,isaachen,pangfeng}@gmail.com
[2] Institute of Information Science, Academia Sinica
{deathsimon,isaachen,wuj}@iis.sinica.edu.tw
[3] Information and Communications Research Laboratories
Industrial Technology Research Institute
{sting,victor.hsu}@itri.org.tw

Abstract. Recently, asymmetric multi-core architecture have become an important issue in CPU design, software scheduling, and virtualization. In a virtualization environment, a hypervisor scheduler assigns virtual cores to physical cores for task execution. However, a load-balancing scheduling strategy for a symmetric multi-core platform (SMP) is unaware of core asymmetry. The deployment of such a strategy to an asymmetric platform may cause performance degradation or energy waste. To take full advantage of the improved power efficiency and performance made possible by asymmetric multi-core platforms, we need a new scheduling strategy. In this paper, we propose an energy-efficient, asymmetry-aware scheduling mechanism for hypervisors on asymmetric multi-core platforms. The goal is to generate an energy-efficient scheduling plan with guaranteed performance.

Keywords: Energy-efficient, Scheduling, Asymmetric multi-core, Virtualization, Hypervisor

1 Introduction

Recent trends have computing platforms moving from homogeneous multi-core architectures toward heterogeneous and asymmetric multi-core structures. An asymmetric multi-core platform consists of cores with the same ISA but different computing capabilities and power characteristics. An asymmetric multi-core platform seeks to achieve both high performance and low power consumption. Several asymmetric multi-core hardware models, such as ARM big.LITTLE [1], are already available and make into products.

The evolution of the multi-core platform has had an impact on task scheduling design. Given that all cores on a homogeneous multi-core platform are identical (i.e., they have the same computing and power characteristics), the objective of scheduling is "load balancing", i.e., distributing the task workload evenly to all cores [2]. "Load balancing" increases task throughput, minimizes task response times, and avoids overloading individual cores.

Scheduling goals differ between homogeneous and asymmetric multi-core platforms. On asymmetric multi-core platforms, the goal is to maximize power efficiency with modest performance sacrifices. Since computing capabilities and power characteristics differ with core type, asymmetric multi-core platforms require new scheduling strategies as traditional load-balancing scheduling strategies may lead to performance degradation or energy waste due to the system being unaware of and unable to compensate for core asymmetry.

The continuing evolution of multi-core platforms also affects the resource scheduling in virtualization environments. Virtual machine monitor (VMM), or hypervisor, provides a software virtualization environment in which a virtual machine can run with the illusion of full access to the underlying system hardware. A hypervisor scheduler assigns virtual cores in each virtual machine to physical cores, where each virtual core has a workload. The scheduler also determines the execution order and amount of time assigned to each virtual core according to a scheduling policy and optionally the workload of the virtual cores. For example, Xen uses a credit-based scheduler [3], and KVM uses completely fair scheduler [4]. The conventional hypervisor scheduler design also follows the idea of load-balancing, but this may lead to performance degradation or energy waste on asymmetric multi-core platforms. An example is presented in Section 2 to illustrate these drawbacks.

In this paper, we develop an energy-efficient asymmetry-aware scheduling mechanism for asymmetric multi-core platforms. The scheduling mechanism periodically generates a scheduling plan according to the requirement of each virtual core. A scheduling plan has two parts. First it indicates the amount of time each virtual core should run on each physical core, and secondly when a virtual core will run on each physical core.

Formally we define a *Virtual Core Scheduling Problem* to address this scheduling plan problem. We assume that the OS scheduler of a virtual machine is asymmetry-aware, and the hypervisor scheduler will schedule virtual cores to the asymmetric physical cores, so as to generate an energy-efficient scheduling plan with performance guarantees.

A scheduling plan should satisfy the following three constraints. First, each virtual core should run on each physical core for a certain amount of time to satisfy the workload requirement. Second, a virtual core can run on a single physical core at any time. Finally, the virtual core should not switch among physical cores frequently, so as to reduce the overheads.

We propose a three-phase solution for the *Virtual Core Scheduling Problem.* In the first phase, the scheduler determines the amount of time each virtual core should run on each physical core. Given this information, the second phase determines when a virtual core will run on each physical core, so that every virtual core will run on each physical core for the given amount of time from the first phase, and none of the virtual core will run on two physical cores simultaneously. Finally the third phase adjusts the execution order of virtual cores to reduce the number of core migration. Formally we define a *Virtual Core Migration Minimization Problem* that, given all the time steps in executing virtual cores, reorders these time steps to minimizes the number of physical core migration in order to reduce overheads.

This paper has the following contributions. (1) We identify the problems caused by using legacy scheduling algorithm on asymmetric multi-core platform. (2) We formally define the *Virtual Core Scheduling Problem* and propose a three-phase solution. (3) We build an asymmetry-aware hypervisor scheduler based on Xen scheduling framework. (4) We conduct experiments on an asymmetric multi-core ARM platform instead of simulations on SMP with DVFS. The experimental results indicate that our energy-efficient asymmetry-aware hypervisor scheduler achieves energy savings with guaranteed performance.

The remainder of this paper is organized as follows. Section 2 discusses the motivation and design challenge of the hypervisor scheduler for asymmetric multi-core platforms. Section 3 defines our models and constraints in detail, while Section 4 presents our solution. Experimental results are described in Section 5. Related works are presented in Section 6. Section 7 draws conclusions.

2 Backgrounds

An asymmetric multi-core platform consists of cores with the same ISA but different computing capabilities and power characteristics. The scheduler should take these differences of cores into consideration while assigning tasks to cores. For example, tasks with higher computing resource requirements, such as decoding or compressing, should be scheduled to cores with higher computing capabilities. On the other hand, other tasks such as downloading or music playing should be scheduled to power-efficient cores to reduce power consumption. Deploying the workloads evenly to all cores on asymmetric multi-core platform may lead to performance degradation or energy waste, as shown in Fig. 1.

Fig. 1 depicts the scheduling results of a load-balancing hypervisor scheduler and an asymmetry-aware hypervisor scheduler. Since the load-balancing hypervisor scheduler in Fig. 1(a) is not aware of the asymmetry between cores, it distributes the virtual cores evenly to the physical cores. In contrast, an asymmetry-aware hypervisor scheduler as shown in Fig. 1(b) assigns only the core with high resource requirements ($vCore_2$) to a performance "big" core, and assigns the other cores with low resource requirements to power-efficient "little" cores.

Although we expect that asymmetric multi-core platforms can provide both energy-savings and performance, several design challenges need to be solved. The first challenge is to design a new scheduling algorithm for the hypervisor scheduler in order to take advantage of the asymmetric multi-core architecture. Virtual cores should be deployed to physical cores for execution according to

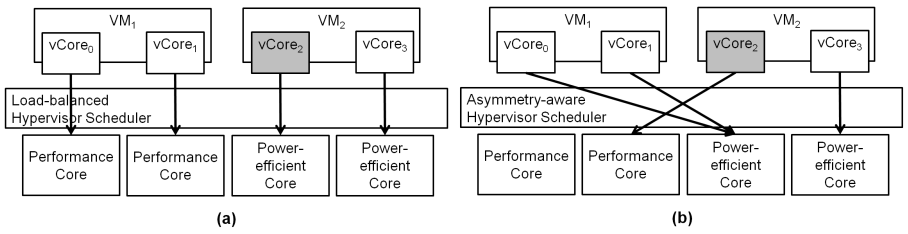


Fig. 1. Example of scheduling results (a) load-balancing hypervisor scheduler (b) asymmetry-aware hypervisor scheduler. $vCore_2$ has high resource requirement, while the other three have low resource requirement.

their resource requirements. The objective of the scheduler is to maximize power efficiency while minimizing the impact on performance.

Asymmetry-aware hypervisor scheduler is different from OS level asymmetry-aware scheduler. The scheduling unit is a process/thread from a program in OS level, while the hypervisor scheduler assigns virtual cores from a virtual machine to physical cores for execution. The behavior and workloads of a virtual core is much more complicate and harder to predict compared to a process since there are usually more than one process running on a virtual core. Directly applying asymmetry-aware algorithm from OS level scheduler to hypervisor scheduler may not be feasible. Therefore, we need to design a new asymmetry-aware scheduling algorithm for the hypervisor scheduler.

The second challenge is that we need an asymmetry-aware OS scheduler in the guest virtual machine. Even with an asymmetry-aware scheduler for the hypervisors, the scheduling mechanism in the guest OS should also be asymmetry-aware. However, conventional OS schedulers are not asymmetry-aware. Tasks are evenly distributed among all virtual cores. From the hypervisor perspective, the resource requirement of virtual cores from the same virtual machine are the same. Therefore, the hypervisor scheduler cannot take advantage of the benefits of the asymmetric multi-core architecture.

This paper focuses on the first challenge: developing an asymmetry-aware hypervisor scheduler. We assume that the scheduling mechanism in the guest OS is asymmetry-aware, i.e., tasks are scheduled to the corresponding cores according to their specific characteristics and resource requirements. This assumption is reasonable given the existence of other asymmetry-aware solutions for the OS scheduler, such as Linaro [5, 6].

3 Virtual Core Scheduling Problem

To design an asymmetry-aware hypervisor scheduler, we define the *Virtual Core Scheduling Problem* and derive our scheduling algorithm. First we introduce the computation and power models in an asymmetric multi-core system. We then define the problem based on the models, followed by the scheduling constraints.

3.1 Models

Our model includes two types of cores – virtual cores and physical cores. A virtual core vC_j is a tuple (v_j, t_j), where v_j is the operating frequency of the virtual core, and t_j is the type of virtual core. In this work, we apply only two types of cores, "big" for performance and "little" for power-saving. Core type of a virtual core is determined by its operating frequency. For example, if the frequency of a virtual core is greater than 800 MHz, then we classify it as "big". Otherwise the virtual core is "little". Similarly, a physical core pC_i is a tuple (f_i, t_i), where f_j is the frequency of the physical core, and t_i is its actual type. We assume that the hypervisor is aware of these core parameters.

To build our power model for a physical core, we first measure the power consumption of a performance "big" core and a power-efficient "little" core on a Juno ARMv8 development board [7] under different loads. The benchmark is *bzip2* from SPEC CPU2006. We apply a simple program, *cpulimit*, to limit the percentage of CPU usage of the benchmark. From Fig. 2, we can observe that the power consumption of a core is almost linear to the load. In addition, rates of increase vary among cores with different frequencies. We conclude that power consumption is related to core type, frequency, and load.

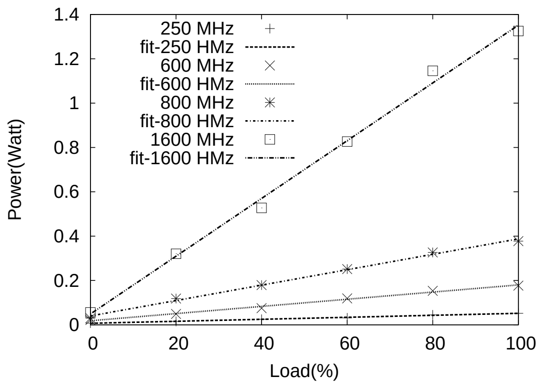


Fig. 2. Relationship between load and power of a performance core (800 & 1600 MHz) and a power-efficient core (250 & 600 MHz) executing *bzip2*

The power model of a physical core is defined as follows. According to our preliminary experiments, the power consumption of a physical core is a function of three factors – *core type*, *frequency*, and *load*. *Core load* is defined as the percentage of time a core is engaged in executing virtual cores. The power consumption increases almost linearly with core load. We define the power consumption of a physical core as a function of core type, frequency, and load.

3.2 Problem Definition

The *Virtual Core Scheduling Problem* is defined as follows. For every time period, the hypervisor scheduler is given a set of virtual machines, each of which has a set of virtual cores. Given the operating frequency of each virtual core, the scheduler will generate a scheduling plan, such that the power consumption is minimized while performance is guaranteed.

A scheduling plan consists of two parts. First it indicates the amount of time each virtual core should run on each physical core. We define $a_{i,j}$ as the percentage of time virtual core j is on physical core i in a time interval, $0 \leq a_{i,j} \leq 1$. The second part of a scheduling plan is the execution order of the virtual cores on each physical core.

The scheduler must guarantee the performance of the virtual cores. We define the performance of each virtual core as the ratio of the computing resource assigned to the computing resource requested. The **computing resource**, or number of CPU cycles, can be computed by multiplying the frequency by the execution time. Since the execution time is fixed in each interval, the performance is equivalent to the frequency ratio. For example, if a virtual core with a frequency 800 MHz is scheduled to a physical core with a frequency 1200 MHz, and the execution time is 60% of the time interval, then the performance of this virtual core is $0.6 * 1200/800 = 0.9$.

Constraints There are several constraints on a feasible scheduling plan in the *Virtual Core Scheduling Problem.* Assume there are m virtual cores and n physical cores in total. A virtual core cannot be executed by two or more physical cores simultaneously, i.e. there can be no overlapping. Therefore the percentage summation of a virtual core j in one time interval cannot be greater than 1. On the other hand, a physical core can provide only a fixed amount of resources each interval, therefore the loading of core i cannot be greater than 1.

As mentioned earlier, the scheduler should guarantee performance. We define a scaling factor s, which equals to the ratio of the total resource requirement to the amount of resources the physical cores can provide. The actual amount of resources assigned to each virtual core is its resource requirement times s. The scaling factor guarantees fairness, i.e. the performance of every virtual cores are the same, thus neither virtual core is sacrificed for the other.

4 Three phase scheduling

In this section, we propose our solution for the *Virtual Core Scheduling Problem.* The solution consists of three phases. In the first phase, the scheduler generates a set of $a_{i,j}$, i.e. the percentage of time virtual core j is on physical core i in a time interval. In the second and the third phase, the scheduler determines the execution order of each virtual core on a physical core.

The reason we separate our solution into three phases is that, with the information on the amount of time each virtual core should run on each physical core, it is difficult to satisfy the following two constraints simultaneously – a virtual core can only run on a single physical core at any time, and the virtual core should not migrate among physical cores frequently. Instead we formulate the first constraint as an edge coloring problem and solve it with an efficient algorithm [8]. Then we adjust the solution from the edge coloring problem to reduce the number of core migrations in order to satisfy the second constraint.

4.1 Phase 1

We formulate the first phase into an optimization problem according to our computation and power model for asymmetric multi-core platforms. Assume that there are n physical cores in total. The goal is to generate the amount of time each virtual core should run on each physical core, such that the objective function is minimized. The objective function is the summation of the power consumption of physical cores, which is a function of core type, frequency, and load. The output must satisfy some constraints, as mentioned in Section 3.2.

Given the objective function and these constraints, we can apply techniques such as linear programing to generate the output . However, such output may not be feasible in real world computing systems, thus we make some modifications to make the solution feasible to real world computers. The problem of the output generated by linear programming is that it may be any arbitrary number between 0 and 1. However, it is not possible to issue an interrupt at an arbitrary time to migrate a virtual core. For example, assume the scheduling period is 1 second, $a_{i,j} = 12.34567\%$ is not acceptable since the architecture cannot issue an interrupt at a granularity of 0.0000567 second.

We make the following modifications to the original problem. We divide a scheduling time interval into k “time slices”. In each time slice, a physical core only executes tasks from one virtual core. The range of $a_{i,j}$ is changed from an arbitrary number between 0 and 1 to an integer between 0 and k. With this modification, the new goal is to generate the amount of “time slice” each virtual core should run on each physical core, such that the objective function is minimized while satisfying the constraints.

To solve the problem efficiently, we apply a greedy heuristic to generate the amount of “time slices” for each virtual core. For every physical core, we compute the “power-consuming” ratio γ, which equals to the power consumption per CPU cycle of that core, under its current loading. The scheduler first sorts the virtual core in ascending order of their operating frequency, and repeats the following process until all virtual cores are assigned to physical cores. First, it assigns a virtual core to the physical core with available computing resources and the least γ. Then, it updates γ and load information of the physical core. If the physical core cannot provide sufficient resource to the virtual core, the scheduler searches for the next physical core that meets the requirement of the virtual core. Notice that even if a virtual core is assigned to more than one physical core, the total amount of “time slice” of this virtual core should not exceed k, which is the amount of time slice in a scheduling interval, to avoid overlapping execution.

4.2 Phase 2

The scheduler decides the execution order of each virtual core on a physical core in the second phase. In the first phase, the scheduler generates the amount of time each virtual core should run on each physical core, where $a_{i,j}$ is an integer indicating the amount of "time slices" for which virtual core j is on physical core i in a time interval. With this information, the scheduler needs to decide the execution order of virtual cores such that no two physical cores will be simultaneously executing the same virtual core in the same time slice.

We can formulate the problem into the Open Shop Scheduling Problem (OSSP) [8]. A set of jobs (virtual cores) must each be processed for a given amount of time (time slices) at each of a given set of workstations (physical cores),in an arbitrary order. The goal is to determine the time at which each job is to be processed at each workstation. The OSSP is NP-hard if the number of workstation is greater than three, and the number of jobs with varying processing times is greater than three. Fortunately, in our problem the virtual cores can preempt each other. With pre-emption, OSSP can be solved in polynomial time. A detailed description of the algorithm can be found in [8].

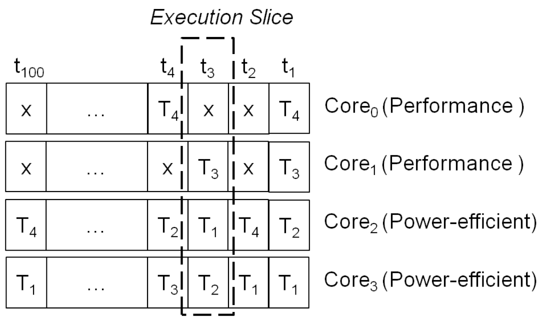


Fig. 3. Example of a scheduling plan. T_i denotes virtual core i.

Fig. 3 is an example of a scheduling result. The "x" means that a core is idle during the time slice. We define an *execution slice* as the virtual cores that are executed simultaneously in a single time slice. Each physical core executes the corresponding virtual core at each time slice according to the scheduling plan.

4.3 Phase 3

In practice, migrating virtual cores between cores incurs overhead, which affects task performance on the virtual core. In a scheduling result, one can exchange the order of any two *execution slices*, and the scheduling result is still feasible. However, exchanging the order of *execution slices* will change the number of virtual core migration between cores. Virtual core migration between cores incurs overhead and thus needs to be minimized.

We define the *Virtual Core Migration Minimization Problem* as follows. Given a scheduling plan, we want to find an order of the *execution slice*, such that the cost is minimized. The cost is the overall number of virtual core migration between physical cores. This problem is NP-Complete, which can be proved by reducing the *Hamiltonian cycle problem.* We propose a greedy heuristic to find a solution.

Given a set of distinct execution slices, our greedy heuristic generates the execution order of these slices in order to minimize the number of virtual core migrations. The greedy heuristic works as follows: (1) It computes the cost of every unselected execution slice. The cost is the number of migration increased if inserting the slice into the queue; (2) It then chooses the execution slice with the least cost, and inserts it into the queue. If there are several slices with the same cost, choose the one with the least "residues cost". "Residues cost" is the summation of the number of migration between this slice and all the unselected slices. The rational is to reduce the number of migrations between the slices already in the queue and the other unselected slices. Repeat step (1) and (2) until all the slices are in the queue. The pseudo code is in Algorithm 1.

Algorithm 1 Greedy heuristic for the Virtual Core Migration Minimization Problem

```
Require: A set of distinct execution slices.
Ensure: The execution order of slices that minimize the number of core migrations.
 1: Empty the queue of execution slice
 2: repeat
 3:     for Every slices not in queue do
 4:         Compute the cost
 5:     end for
 6:     if More than one slice with the least cost then
 7:         Compute the residues cost of these slices.
 8:         Assign the slice with the least residues cost as candidate.
 9:     else
10:         Assign the slice with the least cost as candidate.
11:     end if
12:     Insert the candidate slice into queue.
13: until All slices are in queue
```

5 Experimental Results

We build an asymmetry-aware scheduler to evaluate our algorithm. The scheduler is based on Xen scheduling framework. The scheduler is brought up by Xen during the booting process, and works in the initial domain of Xen. Our scheduler periodically generates a scheduling plan according to the three-phase algorithm, and assigns virtual cores from the user domain to physical cores for execution.

Recall that a scheduling interval is divided into "time slices", as mentioned in Section 4.1. In our scheduler, we set a scheduling interval to be 1 second, with 40 "time slices" in an interval.

We compare the power consumption of our asymmetry-aware scheduler with credit-based scheduler [3], a proportional fair share core scheduler used by the Xen hypervisor. The experiment platform is a Juno ARM 64-bit (ARMv8) development board [7]. There are two performance "big" A57 cores along with four power-efficient "little" A53 cores on the board. ARM energy probe and DS-5 are used to measure the power consumption of the two core clusters during execution.

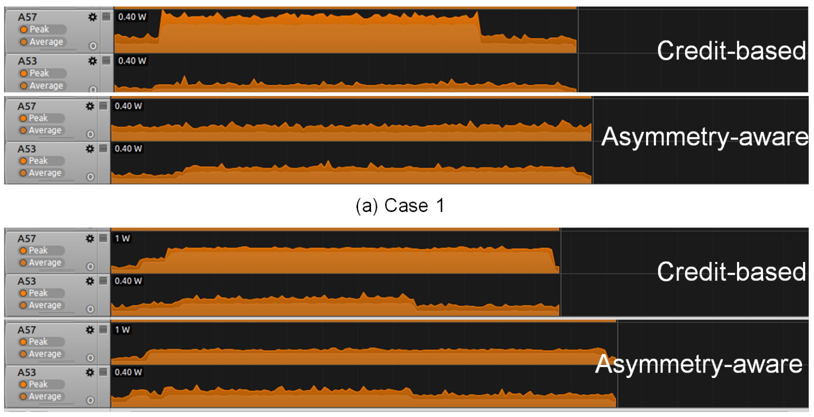
We boot three dual-core virtual machines in our experiment, one Dom0 and two DomUs. Dom0(domain zero) is the initial domain started by the Xen hypervisor, while DomU is the unprivileged domain started by user. Guest virtual machine runs in DomU. We dedicate two power-efficient cores to Dom0, each hosting one virtual core from Dom0. Our scheduler works on one of these virtual cores, while the other one handles the processes from Xen. The workloads from the two DomUs are executed on the other two performance cores and two power-efficient cores.

The benchmark in our experiment is CoreMark [9]. CoreMark is an industrial standard benchmark that aims to measure CPU performance in embedded systems. Our experiments include two cases. In the first case, both guest virtual machines are assigned with light-weight workload, *coremark-1*. We then increase the workloads of guest 0 to *medium (coremark-4)*, and guest 1 to *heavy (coremark-8)* in the second case. We compare the power consumption of our asymmetry-aware scheduling mechanism with the default credit-based scheduler in Xen under the two cases.

Table 1. Performance and Power Consumption of the Two Cases

	Energy(J)		Time(Sec.)	Saving
Case 1	Credit-based	9.817	25.2	1 - (4.948/9.817)
	Asymmetry-aware	4.948	27	= 49.6%
Case 2	Credit-based	34.890	63	1 - (24.775/34.980)
	Asymmetry-aware	24.775	71	= 29.2%

Table 1 summarizes the performance and power consumption results. Our asymmetry-aware scheduler achieves power savings of 49.6% compared to the credit-based scheduler in case 1, and 29.2% in case 2. The execution time of the two cases increase less than 10% compared to the credit-based scheduler. The energy saving comes from the fact that our scheduling mechanism only assigns virtual cores to a performance core when necessary. In the first case, the two light-weight virtual machines use only two power-efficient cores according to our scheduling plan, while credit-based scheduler distributes the workloads to the two performance cores and two power-efficient cores. In the second case, our scheduling plan reduce the amount of workloads running on performance core,



(a) Case 1

(b) Case 2

Fig. 4. Power Consumption During the Execution of Benchmarks (measured with ARM energy probe and DS-5)

thus using less energy. Fig. 4 is the power consumptions during execution. From the results, we conclude that our asymmetry-aware scheduling mechanism has better power efficiency and performance guarantee.

6 Related Works

Several studies have investigated asymmetric multi-core scheduling in virtual environments. Kumar et al. [10] test and validate that scheduling the controlling domain in a hypervisor on slower core saves power while only slightly affecting guest domain performance. In our work, the scheduler not only schedules the controlling domain on slower core, but also schedules the cores from guest domain to achieve energy-efficiency while keeping performance. Kazempour et al. [11] implement an asymmetry-aware scheduler for hypervisors that focus on both energy-efficiency and fair sharing of physical fast cores among virtual cores. However, their scheduler does not reduce the usage of fast but power-hunger cores. On the other hand, our scheduler only assigns virtual cores with high computing resource requirement to fast cores, thus saves more energy. Kwon et al. [12] propose a scheduler that characterizes the efficiency of each virtual core and map them to the most area-efficient physical core. They consider two important aspects, performance fairness among virtual machines, and performance scalability for changing availability of fast and slow cores. Their scheduler aims to increase the overall system throughput, while ours aims to reduce power consumption. Wang et al. [13] present a method to dynamically estimate the energy-efficiency factors of each virtual core, and map the virtual cores to heterogeneous cores. However, their method requires offline training and hardware performance supports. In contrast, our scheduler only requires the frequency information of each core, which can be fetched from common driver such as *cpufreq* in Linux. In addition to these differences, the evaluation environments of prior efforts mentioned above are all simulated asymmetric multi-core environments, in which DVFS/DCVS is applied to symmetric multi-core to emulate asymmetry between cores. However, different types of cores may have different architectures, such as pipeline, on actual asymmetric multi-core platform. In this paper, our evaluation data is more convincing since we conducted the experiments on a real asymmetric multi-core platform.

7 Conclusion

In this paper, we develop an energy-efficient asymmetry-aware scheduling mechanism along with a scheduler for asymmetric multi-core platforms. The goal is to generate an energy-efficient scheduling plan with guaranteed performance. Our solution contains three phases – indicates the amount of time each virtual core should run on each physical core, when a virtual core will run on each physical core, and reduce the number of core migrations. The experimental results show that the asymmetry-aware strategy results in a potential energy savings of up to 49.6% against the credit-based method, while still providing guaranteed performance.

References

[1] Greenhalgh, P.: Big. little processing with arm cortex-a15 & cortex-a7. ARM White Paper (2011)

[2] Shirazi, B.A., Kavi, K.M., Hurson, A.R., eds.: Scheduling and Load Balancing in Parallel and Distributed Systems. IEEE Computer Society Press, Los Alamitos, CA, USA (1995)

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

[4] Pabla, C.S.: Completely fair scheduler. Linux J. **2009**(184) (August 2009)

[5] Linaro: Research update on big.little mp scheduling (2012)

[6] Linaro: A solution to support arm's big.little technology (2014)

[7] ARM: Juno arm development platform

[8] Gonzalez, T., Sahni, S.: Open shop scheduling to minimize finish time. J. ACM **23**(4) (October 1976) 665–679

[9] EEMBC: Coremark

[10] Kumar, V., Fedorova, A.: Towards better performance per watt in virtual environments on asymmetric single-isa multi-core systems. SIGOPS Oper. Syst. Rev. **43**(3) (July 2009) 105–109

[11] Kazempour, V., Kamali, A., Fedorova, A.: Aash: An asymmetry-aware scheduler for hypervisors. In: Proceedings of the 6th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments. VEE '10, New York, NY, USA, ACM (2010) 85–96

[12] Kwon, Y., Kim, C., Maeng, S., Huh, J.: Virtualizing performance asymmetric multi-core systems. In: Proceedings of the 38th Annual International Symposium on Computer Architecture. ISCA '11, New York, NY, USA, ACM (2011) 45–56

[13] Wang, Y., Wang, X., Chen, Y.: Energy-efficient virtual machine scheduling in performance-asymmetric multi-core architectures. In: Network and service management (cnsm), 2012 8th international conference. (Oct 2012) 288–294