Price Theory Based Power Management for Heterogeneous Multi-Cores

Thannirmalai Somu Muthukaruppan * Anuj Pathania * Tulika Mitra
School of Computing, National University of Singapore
{tsomu, pathania, tulika}@comp.nus.edu.sg

Abstract

Heterogeneous multi-cores that integrate cores with different power-performance characteristics are promising alternatives to homogeneous systems in energy- and thermally constrained environments. However, the heterogeneity imposes significant challenges to power-aware scheduling. We present a price theory-based dynamic power management framework for heterogeneous multi-cores that co-ordinates various energy savings opportunities, such as dynamic voltage/frequency scaling, load balancing, and task migration in tandem, to achieve the best power-performance characteristics. Unlike existing centralized power management frameworks, ours is distributed and hence scalable with minimal runtime overhead. We design and implement the framework within Linux operating system on ARM big.LITTLE heterogeneous multi-core platform. Experimental evaluation confirms the advantages of our approach compared to the state-of-the-art techniques for power management in heterogeneous multi-cores.

Categories and Subject Descriptors C.0 [***COMPUTER SYSTEMS ORGANIZATION***]: General; D.4.7 [***OPERATING SYSTEMS***]: Organization and Design

Keywords Heterogeneous Multi-core, Power Management, Price Theory

1. Introduction

Relentless CMOS scaling at deep sub-micron level has resulted in increased power density in microprocessors, which forced computing systems to move in the direction of parallel architectures with homogeneous multi-cores. However, the emergence of dynamic and diverse workloads demand heterogeneous multi-cores, consisting of both *simple* and *complex* cores in the same chip, that can provide high performance at low energy consumption. Existing works show the potential of heterogenous computing in terms of energy efficiency and performance benefits [5, 8, 19, 20, 33]. There exist two different kinds of heterogeneity: *functional heterogeneity* and *performance heterogeneity*. Modern embedded systems exhibit functional heterogeneity by incorporating general-purpose processors, GPUs, and accelerators on a single chip. On the other hand, performance heterogeneity is demonstrated in asymmetric multi-cores, where the cores share the same instruction-set architecture but exhibit diverse power/performance characteristics. For example, ARM big.LITTLE (as shown in Figure 1) integrates high performing, complex, out-of-order ARM Cortex-A15 and energy-efficient, simple, in-order ARM Cortex-A7 cores in the same chip.

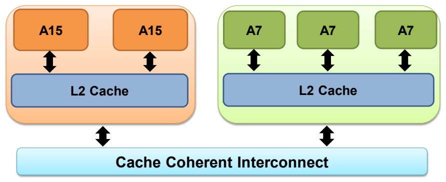


Figure 1: ARM big.LITTLE heterogeneous multi-core.

We propose a power management framework for asymmetric multi-cores exhibiting performance heterogeneity in mobile platforms. Our framework can satisfy the application demands expressed as Quality of Service (QoS) requirements with low energy consumption under a Thermal Design Power (TDP) constraint.

We first present the goals that our framework strives to achieve and then explain our design choices. Traditionally, centralized power management techniques [9, 18, 35] have been employed in embedded mobile platforms. However, centralized approaches suffer from scalability issues, especially in future many-core heterogeneous systems. The power management techniques for such platforms should have distributed decision making strategies.

* These authors contributed equally to this work.

ASPLOS '14, March 1–4, 2014, Salt Lake City, Utah, USA.
Copyright © 2014 ACM 978-1-4503-2305-5/14/03...$15.00.
http://dx.doi.org/10.1145/2541940.2541974

Second, the power management in heterogeneous multi-cores involves coordinating multiple knobs. Dynamic Voltage and Frequency Scaling (DVFS) is the most common energy saving scheme [18, 23, 35]. A good power management approach should incorporate DVFS to provide satisfactory user experience at very low energy consumption. Apart from DVFS, load balancing and task migrations are two major techniques that have significant impact on power in multi-cores [28]. Load balancing and task migration challenges are further exacerbated in the presence of heterogeneity, where the right core should be selected for the right task due to differing power-performance characteristics.

Finally, mobile platforms are now facing strict thermal design power constraint that excludes the possibility of running all the cores at their maximum frequency levels. In such a peak power constrained environment, we have to judiciously select the appropriate voltage-frequency level for each core. Moreover, in an overloaded system exceeding the TDP constraint, all the applications may not be able to meet their QoS targets. In such situations, it is important to maximize the user experience by favoring tasks with higher priority. Therefore, power management schemes cannot be oblivious to the user-level task priorities. Finally, the power management approach should be incorporated in modern commodity operating systems with minimal modifications for easy acceptability.

Most of the existing works focus only on a subset of the aforementioned objectives. It is important to notice that employing multiple energy-saving features (DVFS, load balancing, and migration) requires a coordinated approach. For example, load balancing distributes the workload across the cores and DVFS takes advantage of the balanced workload to reduce the frequency of the cores. While load balancing is beneficial across the same core types, it may conflict with task migration across different core types. This is because task migration needs to ensure that the power hungry cores are used sparingly and switched on only when absolutely necessary. It is challenging to synergistically employ all the different power management techniques within a single framework. Moreover, in a dynamic system with large number of tasks/cores and time-varying workload, arriving at the appropriate decisions with low computational overheads introduces additional challenges.

The novelty of our power management approach is that it incorporates all the objectives outlined earlier within a unified and scalable framework based on the foundations of *price theory* [21] from economics. We implement our approach within the *Linux fair scheduler* with minimal modifications to the kernel. Moreover, our evaluations are performed on a real heterogenous multi-core chip *ARM big.LITTLE*. Our main contributions are:

- We propose *a comprehensive, unified, distributed, and scalable power management technique for heterogeneous multi-cores under thermal design power constraint*.
- The framework is based on price theory that unifies individual power management schemes like DVFS, load balancing, and task migration. Furthermore, our price theory-based approach strictly follows the supply-demand based market mechanisms [21], thereby ensuring stability and efficiency.
- We implement our framework within the commodity Linux operating system for a prototype version of the ARM big.LITTLE platform. All the results reported are from the real system as opposed to simulations. Experimental evaluations show that our technique outperforms existing techniques [3, 25] designed for heterogeneous multi-cores.

2. System Overview

Architecture model. We target single-ISA heterogeneous multi-core architectures, which exhibit power-performance heterogeneity as in big.LITTLE [15] and Tegra [26] platforms. The target system is comprised of a set of cores $\mathcal{C}$ grouped in a set of voltage-frequency clusters $\mathcal{V}$, with each cluster having a separate voltage and frequency regulator. Each cluster v can operate at several discrete voltage-frequency (V-F) levels and consists of a set of cores $\mathcal{C}_v \subseteq \mathcal{C}$. All the cores within a cluster are symmetric in terms of micro-architecture and have to run at the same V-F level.

Task model. A task t is a computational entity that can execute on a core. Each task t is assigned a priority r_t by the user, where higher value means higher priority. $\mathcal{T}$ represents the set of all tasks.

Task to core mapping. Our framework dynamically maps the tasks to the cores. A task t is mapped to a core c_t. $\mathcal{T}_c \subseteq \mathcal{T}$ represents the set of tasks mapped to core c and $\mathcal{T}_v = \cup_{c \in \mathcal{C}_v} \mathcal{T}_c$ denotes the set of tasks mapped to the cores in cluster v. The idle task t_{idle} executes on a core without any active task. If there are no active tasks in an entire cluster, then we can power down that cluster. We define R_c, R_v, R as the sum of the priorities of all the tasks mapped to core c, cluster v, and the entire system, respectively.

Supply Model. Each core c can supply certain amount of computational resources S_c, which is constrained by the maximum supply $\hat{S}_c$. The computational resource is defined in terms of of *Processing Units* (PU), where one PU is equivalent to one million processor cycles per second. The higher the frequency of a core c, the more it can supply PUs (i.e., higher the value of S_c) and the maximum supply of PUs $\hat{S}_c$ is determined by the maximum possible frequency of the core. For example, a core running at 1000MHz (or 350MHz) produces a supply of 1000PUs (or 350PUs). Note that the amount of work (instruction processing) that can be achieved with one PU on a small core is generally less than the amount of work that can be done with one PU on a big core; that is, one PU on a big core is more valuable than one PU on a small core.

The supply of a cluster S_v is the same as the supply of any of the constituent cores, which have identical S_c values. The

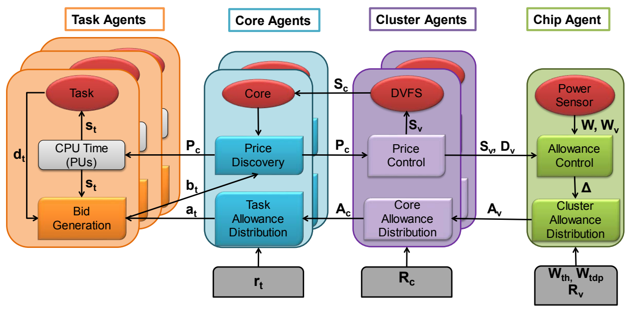


Figure 2: Agent Interaction Overview.

supply of the entire chip S is the summation of the cluster supply values. The current supply of PUs to task t on core c_t is represented by s_t ($\equiv s_t^{c_t}$). The supply of PUs to a task t has to be less than the supply produced on the core c_t it is mapped to, that is, $s_t \leq S_{c_t}$.

Demand Model. Each task t demands a certain amount of computational resources (PUs), which can vary dynamically during the course of the execution. In heterogenous multi-cores, a task demands different amount of PUs across different core types. For example, a task would demand more PUs on a small core compared to a big core to achieve the same application-level performance. The differing demands for computational resources on different core types model the **heterogeneity** of the architecture. The current demand of task t on core c_t is represented by d_t ($\equiv d_t^{c_t}$).

Let D_c represent the sum of demands of all the tasks mapped to core c. The core with the highest demand in a cluster is called the constrained core of the cluster. Let $\tilde{c}_v \in \mathcal{C}_v$ represent the constrained core of the cluster v. Then the demand of the cluster $D_v = D_{\tilde{c}_v}$ is defined as the demand of its constrained core $\tilde{c}_v$ and the demand of the entire chip D is the sum of the demands of the clusters.

Power model. The power consumption of a core c represented by W_c depends on the core type, its V-F level, and the workload. The power consumption of a cluster v is represented by W_v, while the entire chip power consumption is represented by W. The quality of the cooling solution determines the value of the TDP constraint W_{tdp}. As mentioned before, our goal is to keep the total chip power consumption below the TDP ($W < W_{tdp}$) while meeting the task demands at minimal energy.

3. Power management Framework

Our power management framework is designed on the foundation of the *Price Theory* [21] and the *Quantity theory of money* [13]. The resource allocation, DVFS, task mapping and migration are all controlled through the virtual market place, where the commodity being traded is processing unit (PU) using virtual money.

The framework is realized as a collection of autonomous entities called *agents*. Each agent represents a transactional body in the market. An agent can perform various functionalities such as earning, bidding, purchasing, and distribution of computational resources. Furthermore, the actions of an agent are prompted by the goals of the client(s) it represents. For example, an agent representing a task is motivated to meet the demand imposed by the task. Similarly, the agent representing the entire chip ensures that the thermal design power constraint is not violated.

The efficient functioning of the market is ensured through the regulations imposed on the actions of the agents. The regulations introduced are reflections of the power management goals. Interested readers may refer to [13, 21] for more details on economic equilibrium, supply, demand, inflation and deflation concepts.

Our power management framework consists of two main components: *Supply-Demand module* and *Load-Balancing plus Task migration module (LBT)*. Given a task to core mapping, the supply-demand module attempts to satisfy the demands of all the tasks with minimal power consumption under the TDP constraint. It relies on the concept of regulating inflation-deflation. The LBT module aims to reach a power efficient task to core mapping through load balancing and task migrations and employs the concept of reduced spending. Both the modules work in tandem to achieve the final design goal of power management in heterogenous multi-cores.

3.1 Overview of Agents

Figure 2 shows the interaction among the agents.

Task agent. Each task is represented by an agent, who is a buyer in the market. A task agent can receive, spend, or save money to purchase the computational resources (PU). An agent corresponding to a task gets an allowance (virtual money) that it uses to bid for the resources according to the demands of the task.

Core agent. Each core is represented by an agent who determines the price of the computational resources produced by the core. The price for PUs in a core emerges from the bids submitted by the task agents and the current supply of the core. The core agent then distributes the available resources among the task agents according to the bids. A core agent also distributes the allowances received from the cluster agent to the task agents.

Cluster agent. A cluster agent controls the price of the resources by manipulating the supply of PUs in the cores under its purview. The increase (or decrease) of the supply is achieved by varying the V-F levels of the cluster. A cluster agent also distributes the allowance received from the chip agent to the core agents.

Chip agent. The chip agent controls the amount of money in circulation in the system by manipulating the allowances, thereby ensuring that the total chip power does not exceed the TDP constraint. It then distributes the allowances to the cluster agents.

3.2 Supply-Demand Module

In this section, we explain the mechanisms employed by the supply-demand module in manipulating the V-F levels to meet the task demands at minimal power consumption. The supply-demand module requires all the task, core, cluster, and chip agents to work in synergy. In terms of price theory, the demands of all the tasks are satisfied only in an economic market without inflation (or deflation). Thus, controlling inflation (or deflation) is equivalent to providing enough supply to satisfy the current demand in the market. This is the basic principle employed in the supply-demand module.

3.2.1 Task and Core Dynamics

The main objective of the task agent is to sustain the demand of the task it represents. This objective is achieved through an iterative process consisting of three steps per round: bidding by the task agents, price discovery by the core agent, and purchase of the resources. The iterative process continues till the market stabilizes in an economic equilibrium.

The core agent gives an allowance a_t (the virtual money) to each task agent according to the priority of the task. The task agent bids an amount b_t to buy resources based on the current demand of the task d_t. If the bid is less than the allowance, then the difference $m_t = a_t - b_t$ is saved for future use. The bid cannot exceed the sum of allowance and savings. We also require the bid to be higher than a predefined minimum bid b_{min}. That is $b_{min} \leq b_t \leq a_t + m_t$.

For every round, each task agent submits a bid amount b_t based on the experience in the previous round. The task agent increases (or decreases) the bid amount if the supply received was less (or more) than the demand in the previous round. The agents keep the bid unchanged when the demand is satisfied.

Given the bids from all the tasks mapped to core c, the core agent representing c discovers the price per PU P_c as follows $P_c = \frac{\sum_{t \in \mathcal{T}_c} b_t}{S_c}$. Each task agent now purchases the resources at the value determined by the core agent and obtains its current supply $s_t = \frac{b_t}{P_c}\Big|_{t \in \mathcal{T}_c}$.

The bids in the $(N+1)^{th}$ round by the task agents depend on the supply, demand and prices observed in the N^{th} round. As mentioned earlier, the bidding amount is capped by the summation of allowance a_t and savings m_t for the task.

$$b_t^{N+1} = \max\Big(a_t + m_t,\ b_t^N + (d_t - s_t) \times P_c\Big)\Bigg|_{t \in \mathcal{T}_c} \tag{1}$$

Table 1: Task and Core Level Dynamics Example

Round	b_{t_a}	b_{t_b}	P_c	s_{t_a}	s_{t_b}	S_c
1	1	1	0.0066	150	150	300
2	1.33	0.66	0.0066	200	100	300

Running Example Table 1 shows the working of the agents of two tasks t_a and t_b executing on core c with supply $S_c = 300$ PUs. The current demands are $d_{t_a} = 200$ PUs and $d_{t_b} = 100$ PUs. Both the task agents begin with the initial bid of \$1. In round 1, the task t_b is over-supplied, while t_a is under-supplied. In round 2, by adjusting bids based on the supply-demand characteristics, the resources are effectively shared among the tasks according to their requirements.

3.2.2 Cluster Dynamics

The cluster agents are responsible for controlling the price and preventing both price inflation and deflation in the cores. When a core is undersupplied (oversupplied), we observe price inflation (deflation). The cluster agent adjusts the supply using DVFS to avoid either over-supply or under-supply situations in the cores, which in turn is reflected in the stable price of the PU.

In our architecture, all the cores within a cluster have to run at the same V-F level. Thus the supply can be modified only at the cluster level and not at the core level. So the cluster agent observes and responds to price inflation (or deflation) of only the constrained core because the constrained core represents the highest demand among the cores within the cluster. Thus, the supply of the cluster is controlled by the most constrained core. Note that given a task mapping, a non-constrained core may suffer from deflation, while the constrained core suffers from inflation. The cluster agent takes care of the inflation in the constrained core, which can further magnify the deflation in the non-constrained core. The LBT module (discussed in Section 3.3) is responsible for fixing the deflation in the non-constrained core through load balancing.

In order to identify inflation/deflation, we need a base price from which the relative changes can be observed. Every time the V-F level changes, we reset the base price to the new price observed in the market. While the V-F level is changing, we do not allow the task agents to change their bids until they have observed the effect of the new supply on their existing bids.

A user supplied parameter called tolerance factor δ defines the rate of inflation (or deflation) that the cluster agent can tolerate before increasing (or decreasing) the supply, i.e., DVFS by one level. Let P_c and $PBase_c$ represent the current and base price of the resources in a constrained core c, respectively. The cluster agent increases the supply when the current price $P_c \geq PBase_c + PBase_c * \delta$. Similarly, a decrease in supply is observed when $P_c \leq PBase_c - PBase_c * \delta$. The tolerance factor δ determines the response sensitivity of the cluster agents. The lower the value of δ, the faster the response of the cluster agent. The faster response results in frequent V-F level transitions, and hence thermal cycling [29], which can be detrimental to both the performance and the reliability of the hardware. Thus, it is important to carefully select the value of δ by taking into consideration the underlying hardware.

Running Example We demonstrate the cluster level dynamics by extending the example from Table 1 to Table 2. In

Table 2: Cluster Level Dynamics Example

Round	b_{t_a}	b_{t_b}	P_c	$PBase_c$	s_{t_a}	s_{t_b}	S_c
3	1.99	0.66	0.0088	0.0066	225	75	300
4	1.99	0.66	0.0066	0.0066	300	100	400

round 3, let us assume that the demand of t_a increases from 200 PUs to 300 PUs. Let the tolerance factor δ be 0.2. In round 3, the price increases to \$0.0088, which is higher than the tolerable value of $\$0.00796 = \$(0.0066 + 0.0066 \times 0.2)$, thus causing inflation in the system. In round 3, the cluster agent responds by increasing the supply S_c from 300 PUs to 400 PUs (highlighted in gray). At the new supply, both the tasks are satisfied and per unit price observed in fourth round is \$0.0066, which is set as new base price of c. Also, in round 4, the task agents do not change their bids as the new prices are determined only at the end of the round 4.

3.2.3 Chip Dynamics

While the cluster agent attempts to set the V-F level at the minimum value so as to meet the demand of the tasks, the chip level agent is responsible to ensure that the overall chip power does not exceed the TDP budget. The chip agent indirectly controls the power consumption of the chip by manipulating the allowance. It decides on the global allowance value A for the current round. The allowance A is distributed hierarchically throughout the system using the different cluster and core agents.

The global allowance is distributed as cluster allowances (A_v) to the cluster agents and the distribution is inversely proportional to power consumption. The cluster consuming more power is given less allowance, which is calculated as $A_v = A \cdot \frac{W-W_v}{W}$. The cluster allowance is distributed as core allowance (A_c) to the core agents of the cluster based on the priorities of task agents running on them, which is calculated as $A_c = A_v \cdot \frac{R_c}{R_v}$. Finally, the core allowance is further distributed as task allowances (a_t) to the task agents proportional to their priorities, which is calculated as $a_t = A_c \cdot \frac{r_t}{R_c}$.

When the chip agent increases the global allowance A, the task agents receive additional money to generate higher bids for the resources. The task agents with unsatisfied demands increase their bid with the additional money. This causes inflation in the under supplied clusters, triggering the respective cluster agents to control the inflation by increasing the supply (increase V-F level). This increased supply in the cluster results in increased power consumption.

On the other hand, when the chip agent decreases the global allowance A, all the task agents have less money at their disposal and hence are forced to bid lower values. This causes deflation in the clusters, which prompts the cluster agents to decrease the supply (decrease V-F level) to control the deflation, resulting in reduced power consumption.

When the chip agent decides to keep the allowance A constant, all the cores will reach stable equilibrium prices. With stable prices, neither inflation or deflation will be observed by the cluster agents resulting in a *steady-state* with no changes in V-F levels.

The global allowance for the $(N+1)^{th}$ round is set as follows $A^{N+1} = A^N + \Delta$, where A^{N+1} and A^N are the current and previous round allowances, respectively and Δ is the change in the allowance. The key question is how to dynamically set the Δ value. The Δ value is set according to the current total power consumption of the chip.

When the chip power consumption W is below the TDP, the primary goal of the chip agent is to meet the demands of the tasks. On the other hand, if the chip power exceeds TDP, then the chip agent is responsible to bring the power below TDP. In case the system has a demand that is unsatisfiable within the TDP, due to the discrete nature of the V-F levels the system will oscillate around the TDP. To stabilize the system near TDP when overloaded, we introduce a buffer zone near TDP where the system is ought to stabilize. The size of the buffer zone is decided by the parameter W_{th}. Thus, the spectrum of power consumption is divided into three regions:

Normal State. In the normal state, the power consumption of the entire chip is less than the pre-defined threshold $W < W_{th}$. In this state, the chip agent manipulates the Δ value based on the current total supply S and total demand D of the entire chip. When the demand is not satisfied in at least one of the clusters, the chip is under-utilized and the task agents need extra money to buy more resources. Therefore, the allowance is increased by an amount proportional to the difference between the supply and the demand. Therefore, the value of Δ is given by $\Delta = A^N \cdot \frac{D-S}{D}$.

Threshold State. In the threshold state, the power consumption of the chip is observed between W_{th} and TDP W_{tdp}. Ideally, it is desirable for the power consumption of the chip to stabilize in threshold state when the system is overloaded. The stability is attained in the threshold state by keeping the allowance constant through $\Delta = 0$. With larger buffer zone ($W_{tdp} - W_{th}$), the number of oscillations around the TDP reduces and the stable state is reached quickly, but the chip might be severely under-utilized. On the contrary, a smaller buffer zone leads to frequent oscillations around the TDP, but achieves higher utilization. The idea of stability here is similar to the concept of hysteresis in control systems.

Emergency State. In the emergency state, the power consumption of the chip is above W_{tdp} and must be brought down quickly. The allowances of the task agents have to be curbed to reduce the power consumption. In emergency state, the reduction in allowance is proportional to the deviation from the TDP. Hence, the value of Δ is given by $\Delta = A^N \cdot \frac{W_{tdp}-W}{W_{tdp}}$. Thus our system can achieve stability (stable equilibrium price) in either the normal state (supply

Table 3: Chip Level Dynamics Example

Round	A	a_{t_a}	a_{t_b}	b_{t_a}	b_{t_b}	m_{t_a}	m_{t_b}	P_c	$PBase_c$	d_{t_a}	d_{t_b}	s_{t_a}	s_{t_b}	S_c	D	S	W
4	4.5	3.0	1.5	1.99	0.66	1.01	0.84	0.0066	0.0066	300	100	300	100	400	400	400	.8W
5	4.5	3.0	1.5	1.99	0.66	1.01	0.84	0.0066	0.0066	300	300	300	100	400	600	400	.8W
	6.0	4.0	2.0	1.99	1.98	3.02	0.85	0.0099	0.0066	300	300	200	200	400	600	400	.8W
	6.0	4.0	2.0	1.99	1.98	3.02	0.85	0.0099	0.0066	300	300	200	200	500	600	500	.8W
6	6.0	4.0	2.0	1.99	1.98	5.03	0.86	0.0079	0.0079	300	300	250	250	500	600	500	2W
7	6.0	4.0	2.0	2.38	2.38	6.64	0.47	0.0095	0.0079	300	300	250	250	500	600	500	2W
	6.0	4.0	2.0	2.38	2.38	6.64	0.47	0.0095	0.0079	300	300	250	250	600	600	600	2W
8	6.0	4.0	2.0	2.38	2.38	8.25	0.10	0.0079	0.0079	300	300	300	300	600	600	600	3W
9	4.0	2.67	1.33	2.38	1.42	8.53	0	0.0063	0.0079	300	300	375	225	600	600	600	3W
	4.0	2.67	1.33	2.38	1.42	8.53	0	0.0063	0.0079	300	300	375	225	500	600	500	3W
10	4.0	2.67	1.33	2.38	1.33	8.81	0	0.0074	0.0074	300	300	320	180	500	600	500	2W
11	4.0	2.67	1.33	2.23	1.33	9.25	0	0.0071	0.0074	300	300	313	187	500	600	500	2W
.																	
.																	
16	4.0	2.67	1.33	2.01	1.33	12.26	0	0.0067	0.0074	300	300	300	200	500	600	500	2W

meets demand), or the threshold state (when overloaded), but never in the emergency state.

Savings An important by-product of our price theory based power management scheme is the concept of allowance savings by the task agents in the form of non-zero m_t values. How does a task end up with savings? The savings are incurred under two scenarios. First we note that the global allowance is increased by Δ when the demand is not satisfied in at least one of the clusters. Thus, the task agents belonging to the clusters in supply-demand equilibrium would have additional allowance that will be saved for future bidding. Second, the price per PU within a cluster is determined by the most constrained core. This leads to the savings of allowances by the task agents belonging to the non-constrained cores. The saved allowances would facilitate the task agents to outbid other task agents for more resources during the supply constrained situation in both the threshold state and the emergency state. The savings are especially beneficial for tasks with alternating high and low demand requirement. Such task agents save money during their dormant phase and use the saved allowance to outbid other tasks during their active phase.

We choose to cap the savings of a task agent at a fraction of its current allowance. This is because large amount of savings may allow the tasks to keep the system in an emergency state longer than permissible. The ideal factor for capping is determined by the designer with knowledge of the underlying hardware.

Running Example To illustrate the chip level dynamics, we further extend the example from Table 2 to Table 3. Let us set W_{tdp} and W_{th} to $2.25W$ and $1.75W$, respectively. Let us further assume that, for the given application, the system reaches the emergency state at 600 PUs supply ($3W$ power) and the threshold state at 500 PUs supply ($2W$).

The global allowance is \$4.5 in the beginning. Let the priorities of tasks t_a and t_b be 2 and 1 respectively. Thus task t_a receives higher allowance relative to task t_b due to the difference in priorities. In table 3, the change in values are highlighted in gray.

So far in round 4, we have met supply and demand of both the tasks by increasing the core supply to 400 PUs. Now the demand of task t_b increases to 300 PUs in round 5. As the demand at the core level by the two tasks cannot be met with the current supply of 300 PUs, we observe an increase in allowance (from \$4.5 to \$6.0) as well as price inflation. This price inflation forces the cluster agent to increase the supply to 500 PUs, which brings the chip to threshold state consuming $2W$ power. In the threshold state, the allowance is kept constant (observed in rounds 6-7). Meanwhile, the savings (m_{t_a} and m_{t_b}) are calculated based on the allowance allocated and bids by the tasks in each round.

In round 7, the price inflation again causes the supply to increase to 600 PUs, which pushes the system to the emergency state. Now we need to stabilize the system in the threshold state. This is achieved by decreasing the system-level allowance from \$6.0 to \$4.0. This decreased allowance percolates all the way to the tasks. By round 9, the task t_b has also used up all its savings. This is because the allowance was held constant in the threshold state (rounds 6-7). So with decreased allowance and zero savings, the task t_b is now forced to lower its bids. The lower bids cause price deflation, resulting in reduction of supply from 600 PUs to 500 PUs.

As the supply is brought down to 500 PUs, the system reaches the threshold state again. This time the allowances are constant; hence the tasks cannot increase their bids preventing price inflation and subsequent increased supply (higher frequency) that takes the system back to emergency state. So the system stabilizes (round 16) in the threshold state where the power consumption is close to the TDP and the higher priority task (t_a) meets its demand, while the lower priority task (t_b) suffers.

3.2.4 Stability of the Supply-Demand module

We show that given a fixed task-to-core mapping, the supply-demand module ensures that the system reaches a stable state. By stable state here, we imply that there are no changes in the V-F levels and the resources allocated to the tasks.

The principle of price theory states that the market is only stable at a price equilibrium, which is the price at which the supply is equal to the demand. Once the supply meets the demand, it automatically prevents further inflation or deflation within the market. Let us assume that we start off in a stable state. The stability is perturbed as tasks enter/exit the system, or the demand within a task changes due to phase behavior or change in the input conditions. We show that the system will reach a (possibly) different stable state assuming there is no task migration in between.

There are three possible scenarios when the demand changes. In the first scenario, the demand can be satisfied in the normal state. In this case, we observe price inflation/deflation till the supply is equal to the demand and the system reaches stability. We always round up the demand to the next supply value so as to prevent oscillation between two consecutive supply values. Note that the price equilibrium is reached in the constrained core, which determine the V-F level. For the non-constrained cores, the supply might be greater than the demand because all the cores have the same supply value (V-F level). In this case, the price in the core(s) with over-supply will fall till the bid price hits the minimal bid value b_{min}. The same situation happens when the demand on the constrained core is less than the minimum supply value (minimum frequency level) and the system stabilizes at the minimum frequency.

In the second scenario, the demand can be satisfied in the threshold state. Here, the allowance is kept constant, which eventually translates to fixed bid prices by the tasks (because bid price cannot exceed allowance), preventing further inflation/deflation. In the absence of inflation/deflation, the V-F levels are not modified and all the clusters in the system stabilize to fixed V-F levels.

The third scenario is the most interesting one where the system needs to be in the emergency state to meet the demand. But we clearly cannot keep the system in the emergency state for long and have to ensure that it quickly stabilizes in the threshold state. This is the scenario illustrated in the example in Table 3. The stability is ensured by defining a buffer zone near TDP (the difference between W_{tdp} and W_{th}). The buffer zone should be designed such that the system cannot move from the normal state to the emergency state or vice versa without passing through the threshold state. Once the system reaches the emergency state, the allowances are reduced, which moves the system to the threshold state. Once in threshold state, the allowance is kept constant at reduced value, which leads to fixed bidding prices and hence price equilibrium.

3.3 Load Balancing and Task Migration (LBT) module

The supply-demand module achieves a steady-state with permissible power consumption for any given task mapping by manipulating the V-F levels. But the mapping itself may not be efficient in terms of performance and power leading to a sub-optimal solution. Thus, the goal of the LBT module is to find a task mapping that is superior in terms of both performance and/or power consumption relative to the current mapping. The LBT module first attempts to meet the task demands followed by improving power efficiency through load balancing within a cluster and task migration across the clusters. Load balancing within a cluster helps reduce the V-F level of the cluster, while task migration exploits the heterogeneity of the different clusters to potentially improve both the power and the performance behavior of the tasks. In the following, we first describe our task migration policy across heterogeneous clusters.

Give a fixed task-to-core mapping $\mathcal{M}$, let $s_t^{\mathcal{M}}$ and $b_t^{\mathcal{M}}$ be the steady-state supply and bid corresponding to the task t on core $c_t^{\mathcal{M}}$. Also $d_t^{\mathcal{M}}$ be the demand of task t on core $c_t^{\mathcal{M}}$. Recall that the demand of a task changes based on the core type; the demand is lower on a more powerful core compared to a simpler core for the same level of performance.

We define a metric $perf(\mathcal{M})$ to compare the performance of two different task mappings. Given two different mappings $\mathcal{M}$ and $\mathcal{M}'$, we define $perf(\mathcal{M}') > perf(\mathcal{M})$ if and only if

$$\left(\exists t \in \mathcal{T} : \frac{s_t^{\mathcal{M}'}}{d_t^{\mathcal{M}'}} > \frac{s_t^{\mathcal{M}}}{d_t^{\mathcal{M}}}\right) \text{AND} \left(\forall t' \in \mathcal{T} \text{ s.t. } r_{t'} > r_t : \frac{s_{t'}^{\mathcal{M}'}}{d_{t'}^{\mathcal{M}'}} \geq \frac{s_{t'}^{\mathcal{M}}}{d_{t'}^{\mathcal{M}}}\right)$$

Basically we sort all the tasks $\mathcal{T}$ in the system according to their priority r_t. For the mapping $\mathcal{M}'$ to be better than the mapping $\mathcal{M}$ in terms of performance, we need two conditions to be satisfied. The first condition is that there should exist a task t for which the supply-demand ratio in the mapping $\mathcal{M}'$ denoted as $\frac{s_t^{\mathcal{M}'}}{d_t^{\mathcal{M}'}}$ is higher than the ratio in $\mathcal{M}$. The second condition requires all the tasks with higher priority than t to have either better or equal supply-demand ratio in $\mathcal{M}'$ than $\mathcal{M}$.

We also define another metric $spend(\mathcal{M})$ to capture the aggregate spending by the tasks in the steady-state for the mapping $\mathcal{M}$. The value of $spend(\mathcal{M})$ is calculated as follows, $spend(\mathcal{M}) = \sum_{t \in \mathcal{T}} b_t^{\mathcal{M}}$. From the power perspective, it is desirable if a task movement brings down the aggregate spending without affecting the performance. The aggregate spending $spend(\mathcal{M})$ reduces only when the steady-state bids from all the tasks combined together is lower. This reduction in bid price is observed only when the steady-state demand lowers due to appropriate load balancing across the cores within a cluster and the migration of the tasks to the most efficient cluster in terms of heterogeneity. The reduced bids cause deflation by lowering the price, which in turn brings down the supply, i.e., V-F levels and hence the power consumption. Therefore, any reduction in aggregate spending will translate to reduction in the power consumption, provided that the performance remains unchanged. Thus mapping $\mathcal{M}'$ is more power efficient than mapping $\mathcal{M}$, that is, $power(\mathcal{M}') < power(\mathcal{M})$ if and only if

$$spend(\mathcal{M}') < spend(\mathcal{M}) \text{ AND } perf(\mathcal{M}') \geq perf(\mathcal{M})$$

In the LBT module, each task agent first estimates steady-state $perf(\mathcal{M})$ and $spend(\mathcal{M})$ for the current mapping $\mathcal{M}$. This is the baseline power-performance behavior that we want to improve upon. Next, each task agent estimates $perf(\mathcal{M}')$ and $spend(\mathcal{M}')$ for all possible mappings $\mathcal{M}'$ where only the task corresponding to this agent migrates to another cluster while the remaining tasks do not move.

Let us consider the potential migration of task t from the cluster v to the cluster v'. In order to estimate the performance and the spending of the current and the new mapping, we need the steady-state supply, demand, and bid price for the current cluster v and the target cluster v'. This steady-state behavior is estimated as follows.

- *Demand*: In the current cluster v, the steady-state demand is assumed to be the currently observed demand. For the target cluster v' (with different core type), the steady-state demand is estimated using off-line profiling, which is explained in detail in Section 5. We have recently developed a power-performance prediction model for single-ISA heterogeneous multi-core systems using a combination of program analysis, mechanistic modeling, and empirical modeling [27]. In future, we plan to include this estimation model within our price theory based power management framework to eliminate the off-line profiling step.
- *Supply*: The steady-state supply of a cluster is estimated to be the same as the steady-state demand, unless the supply is constrained by the TDP constraint. The steady-state supply per task can be estimated by distributing the total supply among the tasks in proportion to their priorities.
- *Bids*: The steady-state bids are calculated by estimating the price in the steady-state. The price at a higher V-F level *Z+1* can be estimated from the price observed in the current V-F level *Z* using the following equations,

  $$P_{Z+1} = P_Z + (P_Z \times \delta) \qquad (2)$$

  where δ is the tolerance factor. Using recursion, the price at the steady-state supply can be estimated. For example, let P_Z and δ be \$10 and 0.02 respectively. Let us also assume that the V-F level has to increase by 3 levels to achieve the steady-state supply. Then, by using Equation 2 recursively for each level, the price can estimated to be \$10.612.

Note that the task agents perform performance and savings estimations in parallel, which enables the computational overhead to be distributed across the entire chip, thus ensuring scalability. All the information required for the estimation is hierarchically disseminated from the cluster agents to the chip agents and subsequently to the task agents and is kept consistent with periodic message passing. The overhead of the computation and communication increases with increasing number of tasks, cores, and clusters.

To reduce this overhead, only the task agents in the constrained core of each cluster contemplate movement of the tasks. Furthermore, the overhead is high if the task agents consider migration possibility to all the other cores in the system. We only let the task agents consider the most over-supplied unconstrained core in the target cluster as a potential candidate for migration. Thus there exists a trade-off between the overhead and the quality of the solutions obtained. We demonstrate in Section 5 that this simple heuristic works quite well in practice.

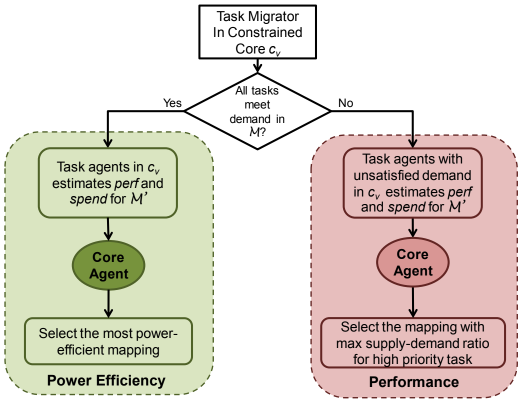

Task Migrator In Constrained Core c_v
All tasks meet demand in $\mathcal{M}$?
Yes
No
Task agents in c_v estimates *perf* and *spend* for $\mathcal{M}'$
Core Agent
Select the most power-efficient mapping
Power Efficiency
Task agents with unsatisfied demand in c_v estimates *perf* and *spend* for $\mathcal{M}'$
Core Agent
Select the mapping with max supply-demand ratio for high priority task
Performance

Figure 3: Task Migration in Constrained Core.

Figure 3 illustrates the flowchart of the task migration module in the constrained core c of a cluster v. If the demands of all the tasks are expected to be satisfied in the steady-state of the current mapping $\mathcal{M}$, then the goal is to reduce the power consumption. In this case, all the task agents in c estimate the performance and spending, if the task migrates to the most over-supplied unconstrained core in each of the other clusters, leading to a new mapping. Each task agent then sends the estimated $perf(\mathcal{M}')$ and $spend(\mathcal{M}')$ for the most power-efficient new mapping $\mathcal{M}'$ to the core agent. The core agent selects the most power-efficient mapping from the task agents and forwards this mapping to the chip agent through its cluster agent. The chip agent finally selects the most power-efficient mapping from all the clusters.

On the other hand, if some tasks are not expected to meet the demand in the steady-state in the constrained core, then only the clusters with unsatisfied demand consider possible task migration. For each such cluster, only the task agents with unsatisfied demand in the constrained core contemplate migration to the other clusters. In particular, at all levels (chip level, cluster level, and core level) we choose the highest priority task that improves its supply-demand ratio through migration without impacting the supply-demand ratio of the higher priority tasks. If two mappings have the same performance, then we select the one with better spending, i.e., better power consumption.

The load balancing process is very similar to the task migration. The only difference is that the target core is not in a different cluster but the most over-supplied unconstrained core within the same cluster. Thus only the cluster agent and the core agent are involved in this process. The chip or the cluster agent approves only one task movement at any given time. The movement of the other tasks, if any, will be performed in future LBT invocation.

3.3.1 Stability of the LBT module

The LBT module performs task migration either within the cluster or across clusters. It can introduce instability if and only if there exists cyclic movement of the tasks. However, our heuristic ensures that the number of task migrations is finite for the following reasons. If the demands of some tasks are not met in the current mapping, then we choose the mapping that improves the supply-demand ratio of the highest priority task with unsatisfied demand through migration. This, by definition, ensures that only the lower-priority tasks are impacted by this migration and the chain of task movements will end with the lowest priority tasks. Thus the number of migrations is bounded by $|\mathcal{T}|$.

On the other hand, if all the tasks meet their demands, then task migration reduces power consumption through reduced spending. This series of task migrations to reduce spending ends when the spending cannot be reduced further.

3.4 Invocation Frequency

The different modules involved in our framework have to be invoked at different rates to reduce the overhead in the system. As mentioned before, the bidding process by the task agents takes place in a series of rounds. The change in the supply level by the cluster agent occurs asynchronously based on inflation/deflation.

The load balancing within the cluster is invoked more frequently than the task migrator because task migration across clusters is expensive (2-4 ms) compared to migration within cluster (50-170 μs). The equations below summarize the periods for load balancing and task migration where linux scheduling epoch is 10ms.

$$task\_migration\_period = 2 \times load\_balancing\_period$$
$$load\_balancing\_period = 3 \times bid\_rounds\_period$$
$$bid\_rounds\_period = max\,(linux\_sched\_epoch,\ task\_period)$$

For workloads with periodic tasks, we set the bidding interval as the maximum of the linux scheduling epoch and the shortest period among the tasks. In our experiments, the shortest period for any task is 31.7ms; so we invoke the bidding round every 31.7ms, the load balancer every 95.1ms, and the task migrator every 190.2ms. The LBT module is disabled in the emergency state as the immediate goal is to bring the power below TDP through the supply-demand module.

4. Related Work

Increasing power consumption raises lot of challenges in modern embedded computing systems, where tablets, smart phones, PDAs are becoming widespread [24]. There exists a plethora of prior works on power management on homogeneous multi-core systems. Most of the works focus on power management using any combination of techniques like DVFS, load balancing and task migrations. Isci et al. [18] evaluate a DVFS based global power management policy with various objectives like prioritization, power balancing and throughput for different combinations of benchmarks. Rangan et al. [28] explore the use of thread migration in power management compared to the traditional DVFS scheme. In [35], the authors present a control theory based power management framework using per-core DVFS capability. Ma et al. [23] present a scalable power management solution for workloads that contain a mix of multi-threaded and single-threaded applications in homogeneous chip multiprocessor. However, these solutions are designed for homogeneous multi-core systems and require non-trivial modifications to adapt them to heterogeneous multi-cores.

Existing works [5, 8, 19, 20, 33] illustrate the potential of heterogeneous multi-cores to improve both the performance and power-efficiency. Craeynest et al. [33] propose a scheduling technique for asymmetric multi-cores using online performance estimation across different core types. Similarly, Koufaty et al. [19] propose a dynamic heterogeneity aware scheduler, which schedules tasks with very low memory stalls on complex cores for higher performance. Our approach handles both the performance and the power management in heterogeneous systems.

A study by Winter et al. [36] evaluates various scheduling and power management techniques for heterogenous multi-cores with special considerations to the scalability of the approaches. They propose a thread scheduling algorithm called Steepest Drop, which has little overhead but does not consider DVFS technique. The technique Pack & Cap proposed in [9] uses thread packing and DVFS to maximize performance under a TDP constraint. Schranzhofer et al. [31] introduce a static solution for task to core mapping problem in heterogeneous MPSoC. Our work dynamically incorporates all the three techniques (load balancing, task migration and DVFS) under a single price theory-based framework to meet performance demands at minimum energy consumption under a power budget. We have previously proposed [25] a hierarchical power management framework for heterogeneous multi-cores. The load balancing and task migration techniques proposed in [25] do not handle heterogeneity and task distribution efficiently.

Some existing works [4, 7, 10–12, 16, 22, 30] borrow economic theory ideas to develop power or thermal management schemes. Ebi et al. [10] propose an agent-based power distribution scheme for multi-cores, where the trading commodity is the power units. Agent based dynamic thermal management techniques are proposed in [4, 14], where negotiations are made in the market to make efficient task migration decisions. Roy et al. [30] propose an energy management technique for mobile devices based on abstractions such as isolation, delegation and subdivision. This technique requires building an offline energy model for a system, which consists of a multi-core that uses two different ISA (ARM11 and ARM9). In contrast, our technique targets single-ISA heterogeneous multi-core system that enables task migration.

Some prior works [7, 16, 22] employ welfare economics in datacenters to improve power efficiency. [16, 22] employ Mixed Integer Linear Programming (MILP) technique for determining the optimal allocation of resources. Lubin et al. [22] present power management in homogeneous multi-core datacenters. This approach is extended to heterogeneous systems in [16]. The solving time is quite high ($800ms$) for MILP formulation. This is only suitable for datacenter workloads exhibiting relatively stable phases so that allocation decisions can be made at long intervals (e.g., 10-minute interval). But such high overhead cannot be tolerated in a mobile platform with dynamic workloads where the allocation decisions need to be revised multiple times per second. Our dynamic price theory based power management mechanism is distributed in nature and thus accomplishes the optimization with minimal overhead.

5. Experimental Evaluation

We now proceed to evaluate our price-theory based power management framework, called *PPM*, for asymmetric multi-cores. First, we present the experimental setup and the workload selection. We compare PPM with a control-theory based power management framework for asymmetric multi-cores, called *HPM* [25] and the Linux asymmetry-aware scheduler [3]. Finally, we quantify the effects of savings and priorities followed by the overhead of PPM.

5.1 Experimental Setup

We use the Versatile Express development platform [2] for our experimental evaluation. It is a modular and flexible platform that allows for quick development of both hardware and software prototypes. The platform consists of a versatile express motherboard with a daughter board containing TC2 CoreTile test chip with ARM big.LITTLE heterogeneous multi-core at 45nm GP technology. The test chip consists of two core Cortex-A15 (big) cluster and three core Cortex-A7 (LITTLE) cluster as shown in Figure 1. Both big and LITTLE cores implement ARM v7A ISA and are connected together by CCI-400 cache coherent interconnect. The frequency can only be modified at the cluster level in the target platform. The architectural features of big and LITTLE core are detailed in [15, 27]. For heterogeneous multi-cores, it is important to discern the migration penalties across different clusters. As expected, we observe that the migration cost within a cluster is lower than the cost across the clusters. The migration cost within the big cluster is in the range of 54–105 μs depending on the frequency level, while for the LITTLE cluster the observed cost is 71–167 μs. However, the migration cost across the clusters is much higher: 1.88–2.16 ms for the migration from LITTLE to big cluster at different frequency levels, and 3.54–3.83 ms for the migration from big to LITTLE cluster.

The firmware installed on the motherboard is responsible for booting the Linux kernel in the big.LITTLE chip. We use Ubuntu 12.10 Linaro release for Versatile Express [3] with Linux kernel release 3.8. The evaluation platform is equipped with sensors to measure frequency, voltage, power and energy consumption per cluster. The Linux 3.8 kernel release provides the hardware monitor (hwmon) communication interface to interact with all the sensors located in the test chip. The idle cluster (that did not boot the Linux kernel) can be powered down if necessary. We boot the Linux kernel in the LITTLE cluster. The powering down and modification of V-F levels are achieved using the drivers related to the oscillators.

In our framework, the agents are implemented in software as kernel modules. The core, cluster, and chip agents are instantiated during kernel boot process. The task agents are instantiated as and when the tasks are created. The communication between the tasks in user space and the agents in kernel space is performed using system calls. The cluster agent uses *cpufreq* utility to manipulate the frequency of the cluster. The voltage at each frequency level is automatically set by the hardware. The task migration is handled by the cluster and the chip agents using the task affinity through *sched_setaffinity* interface in the Linux scheduler.

The core agents are responsible for distributing the available resources among the tasks. This is achieved by manipulating the *nice* values of each task. In Linux kernel, *nice* values are the indirect indications of priorities for task management. For example, lower nice value manifests as higher priority and more resource consumption. As we use *nice* values for resource allocation, the user-level priorities of the tasks in our framework are set in the context of the Linux kernel by adding a new member *prio* in the structure *task_struct*. The *prio* value can be modified from the user space using system calls. For the sake of simplicity, we do not allow dynamic modification of the priorities (*prio*) of the tasks.

5.2 Workload Selection

We use benchmarks from PARSEC [6], Vision [34] and SPEC 2006 [1] suites. At present, our framework requires the tasks to express the performance demand. We employ the *Heart Rate Monitor* (HRM) [17] infrastructure to capture this information. HRM provides a simple and effective way to measure the performance of a task in terms of heartbeats per second (hb/s), which is defined as the throughput of the critical kernel in a task. For example, number of frames processed per second defines the heart rate of a video encoder.

Table 4: Illustration of conversion from heart rate to demand with min and max heart rate being 24 hb/s and 30 hb/s respectively.

Prog. phase	Current hr (hb/s)	Frequency (Mhz)	Utilization (%)	s (PU)	d (PU)
1	15	500	100	500	900
2	10	800	50	400	1080
3	40	1000	100	1000	675

For each application, the user can define the performance goal in terms of reference heart rate range and our goal is to maintain the heart rate within that range, while minimizing energy. Note that an application may have highly variable computation requirement due to phase behavior and hence may need different V-F levels or even migration to a different core type to keep the heart rate within the specified range. Table 4 illustrates the conversion of heart rate to demand for a particular task. Each row in Table 4 represents the different program phases of the same application. The user defines the performance goals in terms of minimum and maximum heart rate (24 hb/s – 30 hb/s). From the current observed heart rate, core frequency and task utilization, the demand of the task can be easily computed. For example, with the supply of 500 PUs, the current heart rate is 15 hb/s in program phase 1. It is clearly well below the reference range prescribed by the user. The required demand is estimated using the equation $d_t = \frac{target\ heart\ rate \times s_t}{current\ heart\ rate}$, where $target\ heart\ rate$ is the mean of the minimum and maximum heart rate range. In our example in Table 4, the $target\ heart\ rate = 27$. Similarly, in program phase 3, the current heart rate exceeds the predefined range and here the demand is lowered.

In the absence of HRM infrastructure, an approximate way to determine the demand in Linux operating system is to measure the time a task spends in the run-queue in a given epoch of scheduling. This per-entity load tracking proposed by Paul Turner [32] in kernels higher than 3.7 can be used in lieu of heartbeats. Table 5 summarizes the benchmarks along with inputs and the heartbeat insertion point.

We create 9 different multiprogrammed workload sets from the benchmarks based on the metric $intensity = \frac{\sum_{t \in \mathcal{T}} d_t^{A7} - S_{A7}^{max\text{-}freq}}{S_{A7}^{max\text{-}freq}}$, where $\sum_{t \in \mathcal{T}} d_t^{A7}$ is the total demand of all the tasks in the given workload and $S_{A7}^{max\text{-}freq}$ is the supply at the maximum frequency in the A7 cluster. The metric $intensity$ shows whether the demand of the entire task set in a workload can be accommodated in the A7 cluster at the highest frequency. If $intensity \leq 0$, the supply exceeds the demand and hence the demand from all the tasks can be satisfied in A7 cluster at highest frequency. On the other hand, if $intensity > 0$, some tasks will not meet their demand on A7 cluster and need to move to the more powerful A15 cluster. Therefore, based on the $intensity$ metric, we classify the workload sets into three types: a) *light* ($metric \leq 0$), b) *medium* ($0 < metric \leq 0.30$) and c) *heavy* ($metric > 0.30$). Table 6 summarizes the workload sets and their classification based on the $intensity$ value.

Table 6: Workload Sets

light	**l1**	texture_v, tracking_v, h264_s swaptions_l, x264_l, blackscholes_l
	l2	texture_v, multicnt_v, h264_b swaptions_l, bodytrack_l, blackscholes_l
	l3	tracking_v, multicnt_v, h264_s x264_l, bodytrack_l, blackscholes_l
medium	**m1**	swaptions_l, bodytrack_l, blackscholes_l texture_v, tracking_v, h264_b
	m2	texture_v, tracking_v, h262_s swaptions_n, bodytrack_n, x264_n
	m3	tracking_v, multicnt_v, blackscholes_n bodytrack_n, texture_f, h264_fo
heavy	**h1**	h264_fo, x264_n, blackscholes_n texture_f, swaptions_n, multicnt_f
	h2	blackscholes_n, x264_n, tracking_f bodytrack_n, texture_f, h264_s
	h3	h264_b, h264_fo, x264_n swaptions_n, bodytrack_n, tracking_f
* v-vga, f-fullhd, n-native, l-large * s-soccer, b-bluesky, fo-foreman		

The *LBT* module requires the average demand and power consumption of a task in different core types for speculation during load balancing and task migration. We obtain the average demand and power consumption of the tasks in both Cortex-A7 and Cortex-A15 through off-line profiling. The average metrics (demand and power consumption) do not capture the dynamic phases of a task. Nevertheless, it leads to better speculation than the absence of any knowledge whatsoever and the supply-demand module can handle wrong speculations by manipulating the V-F levels. Moreover, as mentioned before, we plan to include the power-performance estimation model for big.LITTLE [27] multi-core within our price theory based power management framework to eliminate the off-line profiling step in the future.

5.3 Comparative Study

We compare our price theory based power management framework *PPM* with Hierarchical Power Management (*HPM*) technique proposed in [25] and Heterogeneous aware scheduler in Linux kernel (*HL*) [3]. The HPM is a control-theory based power management framework that employs multiple PID controllers to meet the demand of tasks in asymmetric multi-cores under TDP constraint. However, the HPM scheduler uses naive load balancing and task migration strategy.

The HL scheduler released by Linaro in Linux kernel release 3.8 is aware of the heterogeneity in ARM big.LITTLE platform. The activeness of a task (the amount of time spent in the active task run-queue) is used as a proxy for migration decisions. For example, the HL scheduler migrates a task to A15 cluster (A7 cluster) once the time spent in the

Table 5: Benchmarks description

Benchmark	Benchmarks suite	Description	Inputs	Heartbeat Location
swaptions	PARSEC	Monte Carlo (MC) simulation to compute the prices.	native and large	every "swaption"
bodytrack	PARSEC	Tracks a human body with multiple through a series of image sequence.	native and large	every frame
x264	PARSEC	Video encoder.	native	every frame
blackscholes	PARSEC	Solves partial differential equation to calculate the prices for a portfolio.	native and and large	every 50000 option
h264	SPEC2006	Video encoder.	foreman, soccer and bluesky	every frame
texture	Vision	Motion, tracking and stereo vision	vga and fullhd	every frame
multicnt	Vision	Image Analysis	vga and fullhd	every frame
tracking	Vision	Motion, tracking and stereo vision	vga and fullhd	every frame

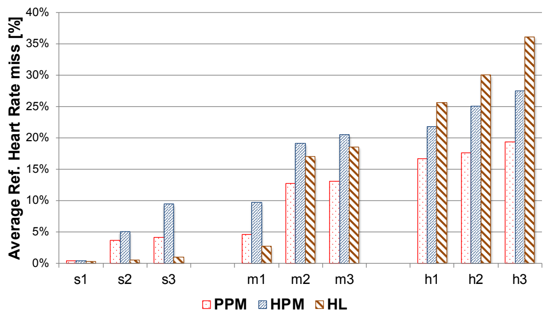


Figure 4: Comparison of the percentage of time the tasks do not meet the reference heart rate range (no TDP constraint).

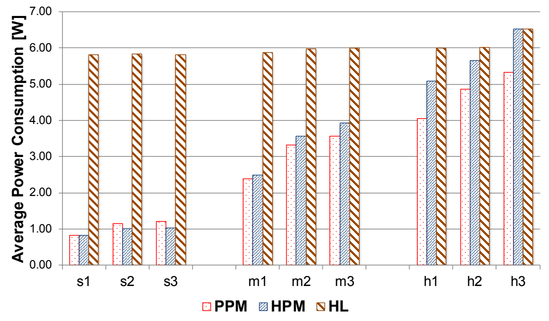


Figure 5: Comparison of power consumption (no TDP constraint).

active run-queue exceeds (falls below) certain predefined threshold. Furthermore, the HL scheduler does not react to the varying demands of the individual tasks. For the HL scheduler, we also employ *cpufreq on-demand* governor that changes the frequency value based on processor utilization. In all the experiments related to the comparative study, we set all the tasks to run at the same priority because HPM and HL do not take the priorities into consideration.

For the first comparative study, we assume that the system does not have any TDP constraint and hence can consume arbitrarily high power. Figure 4 plots the percentage of time the reference heart rate range of any task in the workload is not met, that is, the percentage of time the observed heart rate was smaller than the minimum prescribed heart rate for any of the task in the workload. It is evident that the HL performs better under light workloads (*l1, l2, l3*). This is expected as the HL scheduler migrates the tasks to the powerful A15 cluster at the first opportunity while HPM and PPM both take a more judicious approach. The impact is shown as significantly higher average power consumption for HL compared to HPM and PPM as shown in Figure 5.

On the contrary, the PPM scheduler outperforms both HPM and HL for medium (*m1, m2, m3*) and heavy (*h1, h2, h3*) workloads. The HPM scheduler implements a relatively simple and non-speculative load balancer and task migrator that is oblivious to the utilizations in the other clusters. As the HL scheduler migrates all the tasks to the A15 cluster, it results in inefficient usage of the resources for the more demanding workloads.

Figure 5 plots the average power consumption for the different techniques with no TDP constraint. HPM and PPM have comparable average power consumption across all types of workloads. The HL scheduler with on-demand governor results in an average power consumption of 5.99W, which is much higher than that of HPM (3.43W) and PPM (2.96W) across all the workloads.

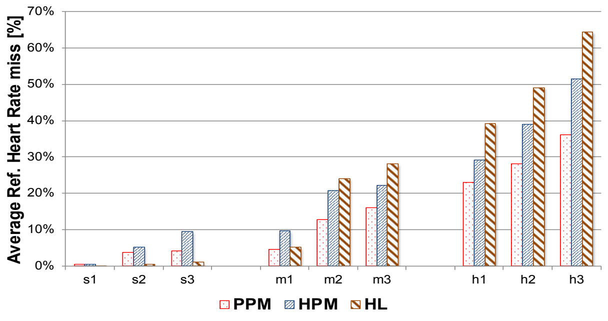
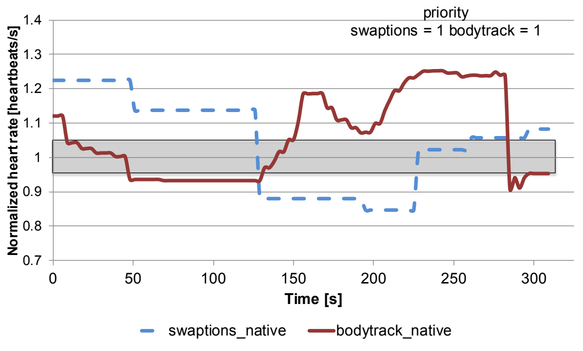
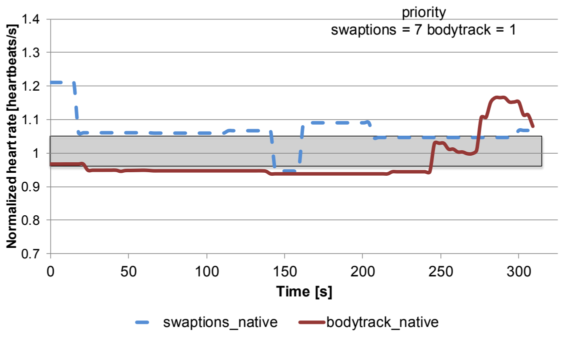


Figure 6: Comparison of the percentage of time the tasks do not meet the reference heart rate range under TDP constraint of 4W.

Next we study how the different techniques cope with strict TDP constraints. We observed through a series of ex-



(a)



(b)

Figure 7: Normalized performance of swaptions and bodytrack where [0.95,1.05] is the normalized performance goal.

periments that the TDP of our evaluation platform is 8W. To emulate a power-constrained environment, we artificially cap the power budget to 4W. For the HL scheduler, we switch off the A15 cluster once the power exceeds the TDP. This is because the observed maximum power in A7 cluster and A15 cluster are 2W and 6W, respectively. Thus powering down of the A15 cluster guarantees that the total power consumption will be well below the TDP constraint of 4W.

Figure 6 plots the percentage of time any task in the workload do not meet their reference heart rate range under TDP constraint of 4W. The tasks are able to meet their reference heart rate more often with PPM approach compared to HPM and HL. The improvements are 34% and 44% compared to HPM and HL, respectively under 4W TDP constraint.

5.4 Impact of priorities and savings

A unique aspect of our price theory-based solution is that we take into consideration the priorities of the tasks as well as their savings. To evaluate the effectiveness of these concepts, we schedule two demanding tasks on one core. We disable load balancing and task migration to study the behavior of the tasks with different priorities and phases of execution (where savings become useful).

Figure 7a shows the dynamically changing performance in terms of heartbeats per second for *swaptions* and *bodytrack* with the same priority. The black shaded region shows the expected performance range. In this case, *swaptions* and *bodytrack* spend 29.7% and 31.1% of time outside the expected performance range. In Figure 7b, we change the priority level of *swaptions* to 7. As can be observed from the figure, *swaptions* is now allocated more resources and hence spends 7.5% of time outside the performance range, while *bodytrack* now spends 57% of time outside the range.

To evaluate the advantage of savings, we choose *swaptions* and *x264*. Both the tasks are running at the same priority. In the initial phase (first 100s), *x264* exceeds the performance goals due to relatively less demand (dormant phase), while *swaptions* just about meets its demand. Thus *x264*

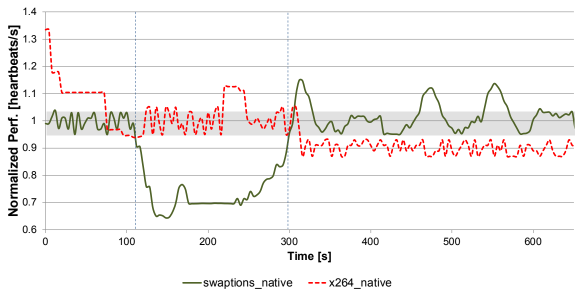


Figure 8: Normalized performance of swaptions and x264 when [0.95, 1.05] is the normalized performance goal.

manages to save a significant fraction of its allowance in this phase.

In the second phase (100 to 300 s), the performance demand of *x264* increases severely as it moves into its active phase. Now *x264* uses up its savings to buy more resources relative to *swaption*. At 300 s, the savings runs out and the high performance demand of *x264* cannot be sustained any further. This illustrate that the concept of savings offers transient benefits to the tasks that spend long time in the dormant phases with very few active phases.

5.5 Scalability

Our price theory based power management framework is scalable due to the distributed nature of the agents that work in parallel. We first provide a computational complexity analysis of our framework followed by quantitative results from the implementation.

The computational overhead arises from the calculations performed in the supply-demand module to decide on the bids, price, and supply level. It is evident that these computations are straightforward with negligible overhead. The primary overhead comes from the LBT module that needs to estimate the performance, spending of current and candidate future task mappings.

Let V be the number of clusters on chip, C be the number of cores per cluster, and T be the average number of tasks per core. Let M be the average computation performed to estimate $perf(\mathcal{M}')$ and $spend(\mathcal{M}')$ for each possible task mapping $\mathcal{M}'$. We estimate the cost and benefit of migrating a task to the most over-supplied unconstrained core in each cluster. Thus for each task in the constrained core, we consider V task mappings. So the worst-case computational overhead in the constrained core is $T \times V \times M$. Clearly, the overhead increases with number of tasks and clusters.

Our evaluation platform has two clusters and five cores. To measure the overhead of our approach, we inject all the 8 benchmarks in the system. The supply-demand module is invoked every 31.7ms. As discussed in Section 3.4, the load balancing and task migration are invoked every 95.1ms and 190.2ms, respectively. The overhead per invocation of the LBT module in the ARM big.LITTLE platform is only 0.003ms.

In order to quantitatively evaluate the scalability of our approach, we *emulate* systems with large number of cores and clusters. We randomly generate tasks with varying demands and feed this information to the A7 core running at the lowest frequency level (350MHz), which is a highly pessimistic scenario. We assume that this core is the constrained core. We randomly generate the supply/demand information for the other clusters (up to 256) and the cores (up to 16), and provide this information to the constrained core. The supply and demands are randomly chosen between 10–50 PUs, while the maximum supply of the cores in different clusters are between 350–3000 PUs. We then measure the time spent in the supply-demand module and the LBT module by this constrained core when the task migration module is triggered every 190ms (see in Section 3.4). Table 7 summarizes the overhead in the constrained core for varying number of tasks, cores, and clusters. As we measure the overhead in the small core at the lowest frequency, the overhead will be much smaller on big cores. As recommended for Linux kernel, we also compile the supply-demand module and the LBT module with -O2 compiler optimization flag. For a system with 256 clusters (16 cores per cluster, 32 tasks per core for a total of 131,072 tasks), we observe that the overhead drastically reduces from 11.4ms to 1ms with -O3 optimization flag.

Some existing works [16, 22] provide mixed integer-linear programming (MILP) formulation of the power management problem, which can provide the optimal solution. The MILP-based approach has low overhead for small systems with fewer core types, clusters, and tasks. The complexity of the MILP approach increases exponentially with increasing number of clusters. The average overhead reported in [22] for solving the MILP formulation is 29 minutes, whereas a greedy approximation takes 5.16 minutes for 1000 homogeneous nodes, each node consisting of four cores. All the experiments in [22] were performed on a 3.2GHz dual-processor dual-core platform with 8GB of memory. In [16], the authors report 800ms linear solver time per invocation at 10-minute interval in a datacenter environment consisting of 160 Xeon nodes or 225 Atom nodes or some combination of them. Clearly, the MILP based approaches are infeasible in a modern embedded platform with dynamically varying workloads requiring frequent invocation of the solver. In contrast, the estimated overhead of our price theory-based approach in a 256 cluster system (16 cores per cluster, 32 tasks per core for a total of 131,072 tasks) is only 11.4ms per invocation at 190ms interval on a Cortex-A7 running at 350 MHz.

Table 7: Computational overhead for varying number of clusters V, cores per cluster C, and tasks per core T.

V	C	T	Total Tasks	Avg. overhead [%]	Avg. overhead [ms]
4	2	8	64	0.02	0.038
		32	256	0.11	0.21
	4	8	128	0.03	0.057
		32	512	0.16	0.30
16	8	8	1024	0.75	1.42
		32	4096	0.96	1.82
	16	8	2048	0.81	1.54
		32	8192	1.37	2.67
256	8	8	16384	3.48	6.62
		32	65536	5.12	9.74
	16	8	32768	4.16	7.90
		32	131072	6.0	11.4

6. Conclusion

In this paper, we propose a price theory-based power management framework for heterogeneous multi-cores to minimize the power consumption while satisfying the performance goals under a power budget constraint. We incorporate various power management techniques like DVFS, load balancing and task migrations in a single, unified and comprehensive framework, which is highly scalable and distributed in nature. Our solution is integrated within the Linux fair scheduler with minimal modifications to the kernel and implemented on a real heterogeneous multi-core platform. Empirical results confirm the superiority of our approach compared to the existing techniques.

Acknowledgments

We would like to thank Mihai Pricopi and Vanchinathan Venkatramani from NUS for their help in setting up the evaluation platform, Sanjay Vishin from CSR for providing us the evaluation platform and the insightful discussions. We also thank the anonymous reviewers and our shepherd, Professor Margaret Martonosi, for their constructive feedback. This work was partially supported by CSR research funding and Singapore Ministry of Education Academic Research Fund Tier 2 MOE2012-T2-1-115.

References

[1] SPEC CPU Benchmarks. `http://www.spec.org/benchmarks.html`.

[2] ARM Ltd., 2011. `http://www.arm.com/products/tools/development-boards/versatile-express/index.php`.

[3] Linaro Ubuntu release for Vexpress, November 2012. `http://releases.linaro.org/13.02/ubuntu/vexpress/`.

[4] M. A. Al Faruque, J. Jahn, T. Ebi, and J. Henkel. Runtime thermal management using software agents for multi-and many-core architectures. *Design & Test of Computers, IEEE*, 27(6):58–68, 2010.

[5] S. Balakrishnan, R. Rajwar, M. Upton, and K. Lai. The impact of performance asymmetry in emerging multicore architectures. In *ACM SIGARCH Computer Architecture News*, volume 33, pages 506–517. IEEE Computer Society, 2005.

[6] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The PARSEC benchmark suite: characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.

[7] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *ACM SIGOPS Operating Systems Review*, volume 35, pages 103–116. ACM, 2001.

[8] J. Chen and L. K. John. Efficient program scheduling for heterogeneous multi-core processors. In *Proceedings of the 46th Annual Design Automation Conference*, pages 927–930. ACM, 2009.

[9] R. Cochran, C. Hankendi, A. K. Coskun, and S. Reda. Pack & Cap: adaptive DVFS and thread packing under power caps. In *Proceedings of the 44th annual IEEE/ACM international symposium on microarchitecture*, pages 175–185. ACM, 2011.

[10] T. Ebi, M. Faruque, and J. Henkel. Tape: Thermal-aware agent-based power economy multi/many-core architectures. In *Computer-Aided Design-Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, pages 302–309. IEEE, 2009.

[11] T. Ebi, J. Jahn, and J. Henkel. Agent-based thermal management for multi-core architectures. In *Organic ComputingA Paradigm Shift for Complex Systems*, pages 587–588. Springer, 2011.

[12] T. Ebi, D. Kramer, W. Karl, and J. Henkel. Economic learning for thermal-aware power budgeting in many-core architectures. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2011 Proceedings of the 9th International Conference on*, pages 189–196. IEEE, 2011.

[13] M. Friedman. Quantity theory of money. *J. Eatwell et al*, pages 1–40, 1989.

[14] Y. Ge, Q. Qiu, and Q. Wu. A multi-agent framework for thermal aware task migration in many-core systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(10):1758–1771, 2012.

[15] P. Greenhalgh. Big.LITTLE processing with ARM Cortex-A15 & Cortex-A7: Improving energy efficiency in high-performance mobile platforms. *white paper, ARM September*, 2011.

[16] M. Guevara, B. Lubin, and B. C. Lee. Navigating heterogeneous processors with market mechanisms. In *HPCA*, pages 95–106, 2013.

[17] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal. Application heartbeats for software performance and health. In *ACM Sigplan Notices*, volume 45, pages 347–348. ACM, 2010.

[18] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*, pages 347–358. IEEE, 2006.

[19] D. Koufaty, D. Reddy, and S. Hahn. Bias scheduling in heterogeneous multi-core architectures. In *Proceedings of the 5th European conference on Computer systems*, pages 125–138. ACM, 2010.

[20] 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*, pages 81–92, 2003.

[21] S. Landsburg. *Price theory and applications*. South-Western Pub, 2010.

[22] B. Lubin, J. O. Kephart, R. Das, and D. C. Parkes. Expressive power-based resource allocation for data centers. In *Proc. of the 21st International Joint Conference on Artificial Intelligence*, pages 1451–1456, 2009.

[23] K. Ma, X. Li, M. Chen, and X. Wang. Scalable power control for many-core architectures running multi-threaded applications. In *ACM SIGARCH Computer Architecture News*, volume 39, pages 449–460. ACM, 2011.

[24] T. Mudge. Power: A first class design constraint for future architectures. In *High Performance Computing–HiPC 2000*, pages 215–224. Springer, 2000.

[25] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin. Hierarchical power management for asymmetric multi-core in dark silicon era. In *Proceedings of the 50th Annual Design Automation Conference*, page 174. ACM, 2013.

[26] NVidia Corporation. The Benefits of Multiple CPU Cores in Mobile Devices., 2011.

[27] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin. Power-performance modeling on asymmetric multi-cores. In *Compilers, Architecture and Synthesis for Embedded Systems (CASES), 2013 International Conference on*, pages 1–10. IEEE, 2013.

[28] K. K. Rangan, G.-Y. Wei, and D. Brooks. Thread motion: fine-grained power management for multi-core systems. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 302–313. ACM, 2009.

[29] T. S. Rosing, K. Mihic, and G. De Micheli. Power and reliability management of socs. *Very Large Scale Integration*

(VLSI) Systems, IEEE Transactions on, 15(4):391–403, 2007.

[30] A. Roy, S. M. Rumble, R. Stutsman, P. Levis, D. Mazières, and N. Zeldovich. Energy management in mobile devices with the cinder operating system. In *Proceedings of the sixth conference on Computer systems*, pages 139–152. ACM, 2011.

[31] A. Schranzhofer, J.-J. Chen, and L. Thiele. Dynamic power-aware mapping of applications onto heterogeneous MPSoC platforms. *Industrial Informatics, IEEE Transactions on*, 6 (4):692–707, 2010.

[32] P. Turner. Sched: Entity Load-tracking Re-work., 2011. `https://lkml.org/lkml/2012/2/1/763`.

[33] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer. Scheduling heterogeneous multi-cores through performance impact estimation (PIE). In *Proceedings of the 39th International Symposium on Computer Architecture*, pages 213–224. IEEE Press, 2012.

[34] S. K. Venkata, I. Ahn, D. Jeon, A. Gupta, C. Louie, S. Garcia, S. Belongie, and M. B. Taylor. SD-VBS: The San Diego vision benchmark suite. In *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*, pages 55–64. IEEE, 2009.

[35] X. Wang, K. Ma, and Y. Wang. Adaptive power control with online model estimation for chip multiprocessors. *Parallel and Distributed Systems, IEEE Transactions on*, 22(10):1681–1696, 2011.

[36] J. A. Winter, D. H. Albonesi, and C. A. Shoemaker. Scalable thread scheduling and global power management for heterogeneous many-core architectures. In *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, pages 29–40. ACM, 2010.