Power-aware Performance Increase via Core/Uncore Reinforcement Control for Chip-Multiprocessors

Da-Cheng Juan
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA, USA
dacheng@cmu.edu

Diana Marculescu
Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA, USA
dianam@cmu.edu

ABSTRACT

Network-on-Chips (NoCs) have emerged as the backbone for the inter-core communication of a chip-multiprocessor (CMP). This paper evaluates and analyzes the advantages of managing the processing cores and the on-chip communication fabric in synergy for the purpose of **performance increase** under power constraints. A semi-supervised reinforcement learning (RL) based approach is proposed for performing dynamic voltage and frequency scaling (DVFS) so as to enable the efficient usage of the available on-chip power budget while maximizing performance. The experimental results show that, on average, overall performance is increased by 11% under iso-power conditions, while a core-only or an uncore-only performance boosting approach can only achieve 7% and 3% improvement in performance, respectively.

Categories and Subject Descriptors

B.8.0 [**Performance and reliability**]: General ; C.1.4 [**Parallel architectures**]: Distributed architectures ; I.2 [**Artificial intelligence**]: Miscellaneous

Keywords

Chip-multiprocessor, On-chip network, Power management, Dynamic voltage and frequency scaling, Reinforcement learning

1. INTRODUCTION

Over the last decade, microprocessor design trends have shifted to chip-multicores from the classic monolithic, single core systems. In a multi-core system, processing elements or cores must communicate with each other under parallel, multithreaded workloads, thereby potentially creating performance bottlenecks in the communication fabric [25]. To reduce this overhead, the network-on-chip (NoC)[1] paradigm [25, 14] has been proposed as a promising solution for on-chip communication for massively-integrated CMPs. However, the enhanced performance and capabilities of such platforms are usually constrained by the on-chip power consumption. While dynamic power management has been extensively studied for multi-core systems in the context of core-only or uncore (on-chip communication fabric) only, the cooperative power management of core and uncore[2] has remained a critical issue not sufficiently explored.

In the context of using dynamic voltage and frequency scaling (DVFS) for minimum power consumption under performance constraints, a possible cooperative power management for both core and uncore resources introduces additional challenges that require maintaining appropriate performance levels for parallel applications executing on the system. For example, in multithreaded applications, spin locks and other synchronization mechanism may amplify small timing differences into very different program execution paths [1], thereby impacting the memory system behaviors substantially. In addition, a significant mismatch between core and uncore frequency may cause unexpected traffic contentions and therefore, may result in significant performance penalty [24]. As a result, how to reliably control cores and uncores in synergy via DVFS while maintaining power constraints remains an open question that needs to be addressed in the context of advanced multi-core systems.

There is a large body of work that has been proposed to address uncore-agnostic power management, especially DVFS, for CMPs [7, 13, 27, 26, 23]. Recently, reinforcement learning (RL) and machine learning (ML) [2] have emerged as popular and robust power management schemes due to their adaptive properties [9, 22, 32, 8, 31]. However, none of the above work has addressed and evaluated the effectiveness of using RL on the power management for NoC-based CMPs, while also including uncore resources. The power management of uncore only or multi-processor systems-on-chip (MPSoC) has also drawn lots of attention from both industry and academia [15, 5, 21, 10, 18]. Although the power management for cores, uncores or MPSoCs has been extensively studied and discussed, none of the previous work has considered synergistic DVFS for NoC-based CMPs to maximize the performance under iso-power conditions.

To the best of our knowledge, this paper brings the following novel contributions:

[1]Network-on-chip (NoC) and on-chip networks will be used interchangeably throughout this paper.

[2]In this paper we refer to uncore resource as representing the communication fabric only, such as routers and links. Caches are included in cores.

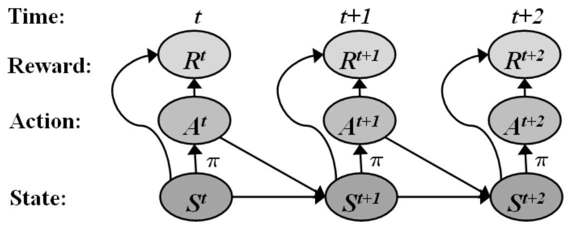


Figure 1: Markov decision process

- We present, for the first time, the evaluation of and comparison among core-only, uncore-only and *cooperative core/uncore DVFS control for performance boosting.* The experimental results confirm that performing DVFS for cores or uncores separately may not be effective for NoC-based CMPs as they are characterized by smaller performance per unit of energy gains.
- Compared to conventional DVFS schemes that address power reduction under performance constraints, we propose a *"reverse"* DVFS: maximize performance while ensuring that power stays within prescribed power constraints.
- To increase the scalability of RL for the adaptive control of advanced CMPs, we proposed a *semi-supervised RL* to reduce the exponential growth of state complexity – the proposed semi-supervised RL maintains a linear complexity in the number of cores. Furthermore, the semi-supervised RL does not compromise the adaptability of the original RL, making it an ideal control scheme for advanced CMPs.
- We evaluate the proposed RL-based, cooperative DVFS control for both cores and uncores with a wide spectrum of parallel, multithreaded applications. The experimental results confirm the effectiveness – on average 10.9% of program execution time is reduced while satisfying given power constraints.

The remainder of this paper is organized as follows. Section 2 introduces the background knowledge. Section 3 provides the problem formulation. Section 4 details the proposed semi-supervised RL. Section 5 presents the implementation flow. Section 6 demonstrates the experimental results, while Section 7 concludes this paper.

2. MARKOV DECISION PROCESS (MDP)

The concept of RL can be described via the interactions between an agent and the environment with uncertainties. The agent attempts to find the best action on the fly for interacting with the various states of the environment, in order to receive the highest reward. Therefore, a MDP-based learning model, such as V learning or Q learning [2], consists of: (1) an agent; (2) a finite state space $S \in \{s_1 \ldots s_n\}$; (3) a set of available actions $A \in \{a_1 \ldots a_m\}$; and (4) a reward function $r^t = r(S^t, A^t)$ where t represents the time. As a convention in statistics, capital variables such as S represent random variables and lowercase variables such as s_1 stand for the values observed. The goal of the agent is to maximize its expected long-term reward. This can be achieved by learning a policy π which can be viewed as a mapping between the states and the actions. Table 1 lists the parallel between MDP and dynamic power management concepts.

Table 1: MDP and dynamic power management

MDP	Dynamic power management
Agent	Controller
Select an action	Select a Voltage/Frequency
Environment states	Machine states
Rewards	High performance

Figure 1 presents a MDP model. At each time point, the agent chooses an action $a \in A$ based on the state S^t to receive the long-term highest reward. Based on the current state S^t and the action A^t , the state transition probability of S^{t+1} can be calculated as:

$$P(S^{t+1} = s_i | S^t = s_j, A^t = a_k) = \theta_i \qquad (1)$$

where $s_i, s_j \in S$ and $a_k \in A$. Here, we define this transition probability as θ_i. More precisely, θ_i represents the probability that state s_i will be reached given that s_j, a_k was reached during the previous time point. Given s_j, a_k, let $\theta = \{\theta_1, \cdots, \theta_{|S|}\}$ represent the probability for each value taken by S^{t+1}, *i.e.*, $\{s_1, \cdots, s_{|S|}\}$. For simplicity of explanation, we assume that the states of MDP in Figure 1 are observable instead of hidden. In other words, on-chip performance or power counters are readily available for all resources; if the states are hidden, expectation maximization (EM) [2] or other algorithms can be used to predict the state. Furthermore, the first-order Markov assumption is used here: the probability of S^{t+1} depends on only S^t and A^t, and therefore $P(S^{t+1}|S^t, A^t, S^{t-1}, \ldots) = P(S^{t+1}|S^t, A^t)$. No information before time t is required to calculate the conditional probability.

The new state S^{t+1} provides a reward to the agent, and the agent learns the control policy $\pi : S \rightarrow A$ to maximize the long-term expected reward $\sum_{t=0}^{t=\infty} \gamma^t \mathbb{E}[r^t]$, where γ is the discount factor between (0,1). γ is used to discount the future reward so that the long-term reward will converge to a certain value after a sufficiently long time. Since the state transition is not deterministic, the expected value is calculated as the expected reward: $\mathbb{E}_\theta[r^t] = \sum_{s_i \in S} r^t \times \theta_i$. Given $\pi : S \rightarrow A$, we can define $V^\pi(s)$ as $\sum_{t=0}^{t=\infty} \gamma^t \mathbb{E}_\theta[r^t]$, where $V^\pi(s)$ represents the long-term expected reward an agent can receive if the agent follows the action sequence chosen according to π, starting at state s. Then, the best policy is $\pi^* = \arg\max_\pi V^\pi(s), \forall s$. Assuming that the state transition probability θ is known, we can calculate the best π^* and the $V^\pi(s)$ in a recursive way to obtain the best control policy $V^{\pi *}(s)$. If θ remains unknown, the learning problem relies on the estimates of the state transition probability. Prior art [32, 18, 22] has shown that Q learning (similar to V learning) can be a good alternative in this context. In this paper, we will demonstrate that, by using *maximum a posteriori* (MAP) estimate with a proper prior distribution of θ, learning the best policy $V^{\pi *}(s)$ can be several times faster than Q learning.

3. PROBLEM FORMULATION

Before elaborating on the proposed semi-supervised[3] RL, we introduce the architecture and the problem formulation used herein. The architectural configuration described here

[3]The definition of semi-supervised learning used in this paper is different from [2].

is to facilitate the explanation of the proposed methodology. Detailed implementation is provided in Section 5.

3.1 Target Architecture

The architecture used throughout this paper is a symmetric, NoC-based CMP that consists of 16 tiles, and each tile contains a Pentium4® core, a private L1 cache, a shared L2 cache and an on-chip router. Table 2 provides the detailed architectural parameters. These 16 tiles are placed in a 4×4 mesh manner. A flit-based mechanism is used for the NoC architecture. The router design follows the standard 5-stage pipeline [17]. Furthermore, each processing core and its corresponding on-chip router are assumed equipped with Intel's Turbo Boost® technology [12]. Therefore, the voltage and frequency pairs, denoted as V/F pairs, can be set to (1) turbo: 1.3V, 3.75GHz, (2) baseline (nominal): 1.0V, 3GHz, (3) low: 0.8V, 2.35GHz and (4) very low: 0.65V, 2.0GHz. These V/F pairs are also listed in Table 2. For conciseness, in the remainder of this paper, when we mention the change of frequency, we actually refer to the change of both voltage and frequency, *i.e.*, V/F pairs. In order to compare with conventional core-only or uncore-only DVFS, we assume that each core and router can be set to different frequencies to best explore the advantages of the cooperative DVFS control.

3.2 Detailed Formulation

The formulation of the cooperative core/uncore DVFS is described as follows. First, the following two inputs are given: (1) a NoC-based CMP, and (2) a parallel, multi-threaded workload that can be executed on the target CMP. The decision variables here are the frequency of each core and router. Under initial conditions, all frequencies are set to the baseline value (3GHz). The objective function to be maximized is overall performance. In this paper, the throughput of the CMP is used as the performance metric. Generally, performance can be expressed as a function of machine states s_i^t and the frequency. Finally, the power consumption of the whole CMP, including both cores and uncores, must be less than or equal to the power constraint (Pow_{const}) at all times.

4. METHODOLOGY

This section elaborates on two main components of the proposed framework: *maximum a posteriori* (MAP) and *semi-supervised* RL.

4.1 Maximum A Posteriori (MAP) Estimate

The key step in MDP is to learn the unknown state transition probability, $\theta_i = P(S^{t+1} = s_i | S^t = s_j, A^t = a_k)$, as mentioned in Section 2. θ heavily depends on the characteristics of the application workload and the underlying processor design. In other words, θ is both machine- and application-dependent. Therefore, θ has to be learned on the fly during the program execution, which is known as the main strength of RL − adaptivity.

Here, we applied *maximum a posteriori* (MAP) to estimate the values of θ. MAP is closely related to maximum likelihood estimate (MLE), but it employs an augmented optimization which incorporates a *prior* distribution over the parameters of interests, θ in this case. Generally, MAP leads to a more accurate and robust estimate than MLE [2]. First, we define d as the observed data of $S^{t+1} = s^{t+1}$ given ($S^t = s_j, A^t = a_k$), and we want to estimate θ, denoted by $\hat{\theta}$ which equals the maximum value of $P(\theta|d)$ (or called mode in statistics). $P(\theta|d)$ can be interpreted as: after d happened, how should the probability of θ be updated? Based on the Bayes rule [30], $P(\theta|d)$ can be rewritten as:

$$P(\theta|d) \propto P(d|\theta) \times P(\theta) \tag{2}$$

where $P(\theta|d)$ is known as the *posterior distribution*, $P(d|\theta)$ represents the data likelihood and $P(\theta)$ is the *prior* distribution. The normalization term $1/P(d)$ is not shown here since it is a constant (under the piecewise stationary condition) and does not affect estimates. Once we determine the prior distribution $P(\theta)$, the posterior distribution $P(\theta|d)$ can be updated by Eq(2) as the data likelihood $P(d|\theta)$ changes since more state transitions are observed by the agent (or the controller). The likelihood function $P(d|\theta)$ follows the multinomial distribution:

$$\frac{N!}{\prod_{i=1}^{i=|S|} x_i!} \prod_{i=1}^{|S|} \theta_i^{x_i} \tag{3}$$

where $x_i \in \{0, \cdots, N\}$ and $\sum x_i = N$. x_i represents the occurrences of $S^{t+1} = s_i$, given $S^t = s_j, A^t = a_k$. N represents the total number of occurrences of $S^t = s_j, A^t = a_k$. The numerical intuition behind this multinomial likelihood is that, given N, we could calculate how likely S^{t+1} will take on each s_i by using its corresponding occurrence x_i. With Eq(3), the data likelihood can be calculated, but we still need to determine the prior distribution to calculate the posterior distribution in Eq(2).

The prior distribution is usually selected based on one's domain knowledge. By selecting a good prior, the posterior distribution can converge faster, and thus the estimate of state transition probability, $\hat{\theta}$, can be obtained earlier during the program execution. In this work, we propose to use a Dirichlet distribution as the prior distribution:

$$P(\theta) = \frac{\prod_{i=1}^{i=|S|} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{i=|S|} \alpha_i)} \prod_{i=1}^{|S|} \theta_i^{\alpha_i - 1} \tag{4}$$

where α_i represents the "hallucinated[4]" counts of s_i, and $\Gamma(\alpha_i)$ is the Gamma function that equals the factorials of $(\alpha_i - 1)$. α_i is similar to x_i in Eq(3) except that x_i represents the actual counts of s_i , whereas α_i stands for our "belief" or "domain knowledge" of how many times s_i should happen before any data are given. Empirically, the range of α_i is set to $10^1 - 10^2$.

Next, we multiply Eq(3) by Eq(4) to obtain the posterior distribution:

$$\begin{aligned} P(\theta|d = \{x_1 \cdots x_{|S|}\}) &= \frac{\prod_{i=1}^{i=|S|} \Gamma(\alpha_i + x_i)}{\Gamma(N + \sum_{i=1}^{i=|S|} \alpha_i)} \prod_{i=1}^{|S|} \theta_i^{\alpha_i + x_i - 1} \\ &= \frac{\prod_{i=1}^{i=|S|} \Gamma(\beta_i)}{\Gamma(\sum_{i=1}^{i=|S|} \beta_i)} \prod_{i=1}^{|S|} \theta_i^{\beta_i - 1} \end{aligned} \tag{5}$$

where $\beta_i = \alpha_i + x_i$. The numerical intuition of β_i is that, besides the actual observations (x_i), we add the hallucinated counts (α_i) to adjust $P(\theta|d)$. In other words, intuitively the

[4]These hallucinated counts α_i can be any positive integer or 0. If the learning process is sufficiently long, θ will be learned correctly and α_i is irrelevant. This is known as "the prior is forgotten" [2]

Table 2: Architectural parameters

Core Parameters	Values	Uncore Parameters	Values	DVFS V/F Pairs		Values
Number of cores	16	Number of routers	16	Turbo	V_{dd}	1.3V
Core model	Pentium 4®	Nominal frequency	3.0 GHz		Freq	3.75GHz
Nominal frequency	3.0 GHz	Router pipeline stages	5 stages	Baseline	V_{dd}	1.0V
L1-I/D caches	Private 64KB, 8-way SA, LRU	Flit size	16 Bytes	(Nominal)	Freq	3.0GHz
L2 caches	Shared 4MB, 32-way SA, LRU	Number of virtual channels	4 per port	Low	V_{dd}	0.8V
Cache coherence	MOESI protocol [20]	Buffer size	4 Bytes		Freq	2.35GHz
DRAM	2 GB	Network topology	4×4 mesh	Very Low	V_{dd}	0.65V
Technology	45nm node with V_{dd} =1.0V	Routing algorithm	X-Y routing		Freq	2.0GHz

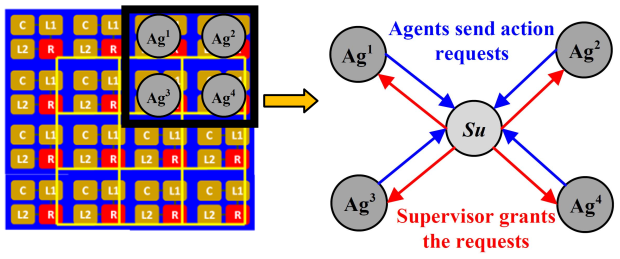
domain knowledge is used to aid the estimate of the state transition probabilities. Furthermore, it can be seen that Eq(5) and Eq(4) are in the same general form of Dirichlet distributions (only the parameters α, β are different), which is known as *"conjugate"* prior and posterior pairs. Each time a new state transition d is observed, the posterior distribution can be updated by the multiplication of new data likelihood and the prior by using Eq(2), and thus, the updated posterior can be fed back into Eq(2) as an *updated prior* ready to estimate the new posterior since they are in the same form. Our experimental results show that by using the MAP with conjugate pairs, the best control policy can be learned several times faster than conventional Q learning.

4.2 Semi-supervised Reinforcement Learning

One critical issue of RL is the lack of scalability due to the exponential growth of machine states [29, 32]. Conventionally, the machine states are defined by three performance counters – instruction per cycle (IPC), misses per kilo instruction (MPKI) and router buffer utilization (BU). These three metrics represent the respective metric of performance from cores, caches and on-chip networks, and thus are sufficiently representative as the machine states of the whole CMP. Therefore, for each tile $|S| = |IPC| \times |MPKI| \times |BU|$ and the total number of states for a CMP is $O(|S|^m)$, where m is the number of tiles. For a dual- or a quad-core system, this issue is not severe. However, for future many-core systems, this exponential growth will require a significant control overhead in terms of memory and computational power.

To increase RL's scalability, we separate the "centralized" agent into several "distributed" agents (one deployed per tile), and arrange a supervisor who can coordinate the actions among agents to best exploit the power budget for the performance increase. Therefore, the agent in each tile has the freedom to learn its control policy dynamically, but at the same time they are "semi-supervised" by the supervisor (implemented as a kernel thread), to work in synergy. By deploying this semi-supervised RL, the complexity of states and actions is reduced to $O(m \times |S|)$ and $O(m \times |A|)$, respectively, which is linear with the number of on-chip resources, instead of exponential.

We illustrate the scheme of the proposed semi-supervised RL in Figure 2. The target NoC-based CMP, shown in Figure 2a, is grouped into four clusters. Each cluster contains one supervisor and four agents (one agent deployed per tile). The power constraint, denoted as Pow_{const}, is treated as the total power budget and distributed equally into four clusters. Therefore, each cluster has the power budget of $\frac{Pow_{const}}{4}$. Figure 2b shows the hierarchy of the supervisor and agents within a cluster. Each agent learns its own best policy by using the RL described in Sections 2 and 4.1. During each control epoch, each agent sends a request to the supervisor,



(a) An NoC-based CMP (b) Supervisor and agents

Figure 2: Scheme of semi-supervised RL

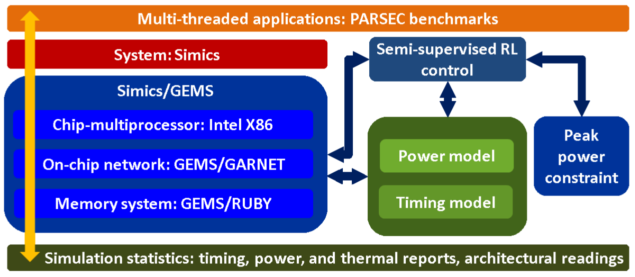

Multi-threaded applications: PARSEC benchmarks
System: Simics
Semi-supervised RL control
Simics/GEMS
Chip-multiprocessor: Intel X86
On-chip network: GEMS/GARNET
Memory system: GEMS/RUBY
Power model
Timing model
Peak power constraint
Simulation statistics: timing, power, and thermal reports, architectural readings

Figure 3: Simulation infrastructure

querying if its own best action (*i.e.*, the frequencies of the core and the router) can be taken. The supervisor will then estimate the total power required. If the power consumption stays within the assigned budget, all requested actions will be granted. Otherwise, the supervisor will constrain each agent from taking its nominal frequency instead of the original best action. The power model used in this paper is described in Section 5.2. With semi-supervised RL, the total state numbers of the CMP is $16 \times |S|$. For the number of actions $|A|$, the core and the router in each tile have four V/F pairs to select from, as described in Section 3.1, which makes $|A| = 4 \times 4$ and the total action numbers is $16 \times |A|$. As a result, the proposed semi-supervised RL reduces the complexity from exponential to linear, and therefore, it is highly scalable.

5. IMPLEMENTATION

In this section, we describe the experimental setup and the simulation infrastructure in detail.

5.1 Performance Modeling Infrastructure

Figure 3 provides an overall view of the simulation infrastructure used in this paper. First, Simics [19] and GEMS [16] are used as a full-system, many-core simulator to evaluate the proposed RL control. The operating system is configured as Linux in Simics. Furthermore, RUBY [16] and GARNET [17] are embedded in GEMS to enable the functional and timing simulations of the cache system and the on-chip communication fabrics, respectively. Detailed parameters are mentioned in Section 3.1. Default values are used as system parameters if not specifically mentioned. For the workloads considered, we use PARSEC [6] as multi-threaded, parallel applications to evaluate the benefits of the proposed control strategy. PARSEC covers a wide spectrum of data sharing and synchronization, which creates both on-chip and off-chip communication close to realistic workloads.

5.2 Power Model

We use the power model proposed by [3] to calculate the power consumptions of processing cores. This power model is calibrated by Intel® Xeon® X7350, and the difference between the actual power consumption and the fitted one from [3] is less than 10%. Therefore, it can accurately calculate the power consumption of NoC-based CMPs emulated by our simulation infrastructure.

For the power model of NoC, we use Orion [11] built in GARNET to provide the power consumption of the on-chip communication fabrics, including routers and links. We further modify and shrink the technology node in Orion to 45nm by using the parameters provided by Orion 2.0 [4]. Finally, CACTI [28] is integrated into RUBY to calculate the power dissipated by L1 and L2 caches in different states (*e.g.*, Read, Write and Standby).

5.3 Controller

The proposed semi-supervised RL-based controller can be implemented as a kernel thread executing on each processing core. Also, the proposed MAP described in Section 4.1 can be implemented very efficiently with a table-lookup technique. We implemented and optimized the controller in C++ and evaluated it on our full-system simulator. The timing overhead is less than 0.03 ms, while each control epoch (or period) is 1.2ms (therefore, controller overhead is around 2%). The power overhead of the proposed control is less than 0.32 Watts per tile, which is less than 1% of the thermal design power (TDP)[5].

For the overhead in memory usage, extra memory space is needed to record all state transition probability. As mentioned in Section 4.2, the number of states per tile is $|S| = |IPC| \times |MPKI| \times |BU|$. Here, we use four states to represent $|IPC|$, three states for $|MPKI|$ and two states for $|BU|$, all based on the utilization rates from [27][5]. Hence, the total number of states per tile is 24 and thus the overhead in memory space is very low.

As a comparison, we re-implement the DVFS controls proposed by [27] and [5]. We select *"Threshold-based"* control of [27] and aggressively upscale the frequency settings for improving the performance. The original baseline frequency in [27] is raised to the Turbo mode (3.75GHz), and the rest of settings are changed in a similar fashion. For uncore-DVFS, the *"freqboost"* of [5] is used to maximize the performance − the frequency is set to 3.75GHz (Turbo) by default. The frequency of a router will be reduced if the occupancy of its downstream routers' buffers is greater than 60% [5], which is considered as potential network congestion. Finally, the power constraint is set to the larger value of the peak power consumption from core-only DVFS or uncore-only DVFS, which is around 350Watts to 470 Watts.

[5]Since the power model [3] is originally for quad-core (Xeon® X7350, TDP=130Watts) and we apply it on the 16-core CMP, the new TDP will be 130 × 4 = 512 Watts.

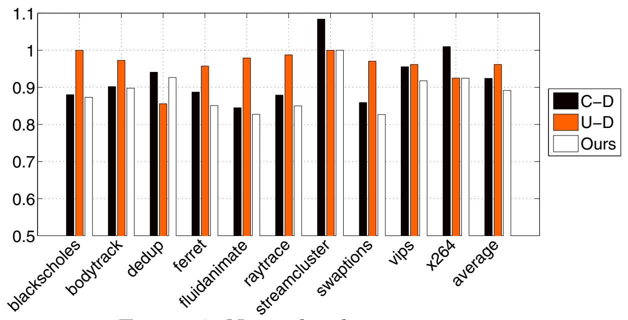


Figure 4: Normalized execution time

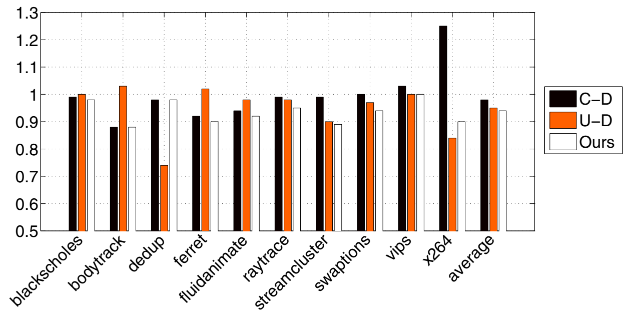


Figure 5: Normalized energy-delay product

6. EXPERIMENTAL RESULTS

This section presents the experimental results, including (1) the overall comparisons, and (2) the network analysis under four control schemes, including baseline (no-DVFS), core-only, uncore-only and cooperative DVFS.

6.1 Overall Analysis

Figure 4 illustrates the analysis of the program execution time for each PARSEC benchmark. "C-D" stands for the Core-only DVFS based on [27], whereas "U-D" is the Uncore-only DVFS based on [5]. All results are normalized to the "Baseline", *i.e.*, the case without any DVFS. On average, the proposed semi-supervised RL achieves 10.9% reduction of the execution time, and outperforms C-D and U-D almost in all cases except *Dedup*. *Dedup* is implemented using a "pipeline programming model" [6] that has two properties: (1) distinct characteristics for each thread, and (2) frequent thread migration: every few thousand cycles. These two facts make the state transition probabilities hard to estimate. However, the proposed control still reduces the execution time by around 7.5%. In most core-bound applications, such as *blackscholes*, both C-D and the proposed control approach outperform U-D. On the other hand, in memory-bound applications, such as *X264*, both U-D and the proposed control have better reduction than C-D. It is also worth mentioning that C-D has longer execution time in *StreamCluster* and *X264*, which will be explained later in Section 6.2. These results clearly demonstrate the effectiveness and robustness of the proposed control, compared to C-D and U-D.

Figure 5 provides the comparisons of energy-delay products. It can be seen that the proposed control achieves the lowest energy-delay product on average − 6% reduction compared to the baseline, whereas C-D and U-D achieve only 2% and 4.7% reduction, respectively. This means the energy efficiency of the proposed control policy is higher than both C-D and U-D. Note that the power constraint is

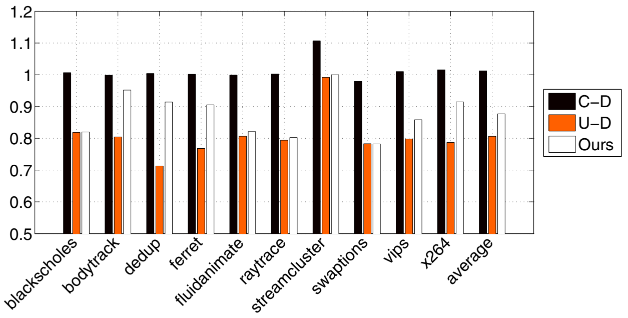


Figure 6: Normalized end-to-end network latency

met at all times and for each benchmark. Furthermore, the peak temperatures (calculated by the thermal model proposed by [3]) of all three control policies stay within 60° C for each benchmark, and thus no thermal emergency occurs due to the "*reverse*" DVFS.

6.2 Network Latency Analysis

Here, we examine the network latency under four different control schemes. Figure 6 presents the average end-to-end network latency for each benchmark. As it can be seen, U-D effectively reduces the network latency by around 19.1%. Considering the reduction of execution time shown in Figure 4, it can be seen that the shorter network latency cannot be directly reflected on the shorter execution time in most cases. This fact reveals that U-D, *i.e.*, uncore-only DVFS can be very effective, but only in certain cases. On the other hand, even if the router frequency is fixed at the nominal value, we can see that C-D has longer network latency in *StreamCluster*, *Vips* and *X264*. This is because the uncore-agnostic DVFS may change the packet injection rate in the network, which results in unexpected congestion and thus longer latency. This also explains why we observe that C-D has longer execution time in *StreamCluster* and *X264* in the previous section. In most cases, the network latency of the proposed control is longer than the latency of U-D (sometimes it is a tie) and shorter than the latency of C-D. This is because the proposed control needs to strike a balance to distribute the power budget to cores or routers for boosting up the frequency, in order to achieve the best performance increase. For example, compared to C-D, the proposed control boosts up the router frequencies (and hence achieves a shorter latency) in *X264*, which in turn reduces the program execution time. All the above results demonstrate that, instead of core-only or uncore-only DVFS, a cooperative control of both cores and uncores is more effective and robust for NoC-based CMPs, which also demonstrates the strength of the proposed semi-supervised RL control.

7. CONCLUSION

In this paper, we evaluate and present the advantages of controlling cores and uncores in synergy for NoC-based CMPs. We also propose a control mechanism based on semi-supervised RL that is highly scalable for advanced CMPs. Experimental results show that the proposed control achieves 11% reduction on the program execution time.

8. ACKNOWLEDGMENTS

The authors would like to thank Dr. Wei-Heng Lo, Dr. Tsung-Yu Tsai, Dr. Andrea Bartolini, Dr. Paul Bogdan and anonymous reviewers for the constructive suggestions. This work was supported in part by an Intel URO grant and NSF Grant CNS-1128624.

9. REFERENCES

[1] A.R. Alameldeen. Ipc considered harmful for multiprocessor workloads. *MICRO*, 2006.
[2] C.M. Bishop. Pattern recognition and machine learning. *Springer*, 2006.
[3] A. Bartolini *et al.* A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores. *GLSVLSI*, 2010.
[4] A.B. Kahng *et al.* Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. *DATE*, 2009.
[5] A.K. Mishra *et al.* A case for dynamic frequency tuning in on-chip networks. *MICRO*, 2009.
[6] C. Bienia *et al.* The parsec benchmark suite: characterization and architectural implications. *PACT*, 2008.
[7] C. Isci *et al.* An analysis of efficient multi-core global power management policies: maximizing performance for a given power budget. *MICRO*, 2006.
[8] G. Dhiman *et al.* Dynamic power management using machine learning. *ICCAD*, 2006.
[9] H. Jung *et al.* Supervised learning based power management for multicore processors. *TCAD*, 2010.
[10] H. Matsutani *et al.* A multi-vdd dynamic variable-pipeline on-chip router for cmps. *ASPDAC*, 2012.
[11] H. Wang *et al.* Orion: a power-performance simulator for interconnection networks. *MICRO*, 2002.
[12] J. Charles *et al.* Evaluation of the intel core i7 turbo boost feature. *IISWC*, 2009.
[13] J. Li *et al.* Dynamic power-performance adaptation of parallel computation on chip multiprocessors. *HPCA*, 2006.
[14] L. Benini *et al.* Networks on chips: a new soc paradigm. *Computer*, 2003.
[15] L. Shang *et al.* Dynamic voltage scaling with links for power optimization of interconnection networks. *HPCA*, 2003.
[16] M. Martin *et al.* Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Computer Architecture News*, 2005.
[17] N. Agarwal *et al.* Garnet: A detailed on-chip network model inside a full-system simulator. *ISPASS*, 2009.
[18] P. Bogdan *et al.* Optimal power management of multidomain multiprocessor platforms under highly variable workloads. *NOCS*, 2012.
[19] P. Magnusson *et al.* Simics: A full system simulation platform. *Computer*, 2002.
[20] P. Sweazey *et al.* A class of compatible cache consistency protocols and their support by the ieee futurebus. *ISCA*, 1986.
[21] P. Zhou *et al.* Noc frequency scaling with flexible-pipeline routers. *ISLPED*, 2011.
[22] R. Bitirgen *et al.* Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. *MICRO*, 2008.
[23] R. Cochran *et al.* Pack & cap: adaptive dvfs and thread packing under power caps. *MICRO*, 2011.
[24] R. Das *et al.* Performance and power optimization through data compression in network-on-chip architectures. *HPCA*, 2008.
[25] R. Marculescu *et al.* Outstanding research problems in noc design: system, microarchitecture, and circuit perspectives. *TCAD*, 2009.
[26] R. Teodorescu *et al.* Variation-aware application scheduling and power management for chip multiprocessors. *ISCA*, 2008.
[27] S. Herbert *et al.* Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. *ISLPED*, 2007.
[28] S Thoziyoor *et al.* A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies. *ISCA*, 2008.
[29] T. Ebi *et al.* Tape: thermal-aware agent-based power rconomy for multi/many-core architectures. *ICCAD*, 2009.
[30] T. Hastie *et al.* The elements of statistical learning: data mining, inference, and prediction. *Springer*, 2009.
[31] Y. Tan *et al.* A framework of stochastic power management using hidden markov model. *DATE*, 2008.
[32] Y. Wang *et al.* Deriving a near-optimal power management policy using model-free reinforcement learning and bayesian classification. *DAC*, 2011.