DYNAMIC VOLTAGE AND FREQUENCY SCALING TECHNIQUES FOR CHIP MULTIPROCESSOR DESIGNS

A Dissertation

by

JAE YEON WON

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Jiang Hu
Co-Chair of Committee,	Paul V. Gratz
Committee Members,	Le Xie
	Radu Stoleru
Head of Department,	Miroslav Begovic

August 2015

Major Subject: Computer Engineering

ABSTRACT

Due to chip power density limitations as well as the recent breakdown of Dennard's Scalingover the past decade, performance growth in microprocessor design has largely been driven by core scaling. These trends have led to Chip Multi-Processor(CMP) designs, currently with tens of cores,and expected to grow to the thousands in the pursuit of exascale computing.The more complicated CMP design is more leading power consumption relatively in computer architecture. The increased power consumption generates thermal issues, and so performance degradation. Therefore, it is certain that power efficient algorithm in CMP and main memory are essential. For the power efficiency, we focus on dynamic voltage/frequency scaling (DVFS) techniques for CMP and main memory.

In the first work, we focus on the "uncore", consisting of an on-chip communication fabric and shared LLC in CMP. The uncore now occupies as much as 30% of the overall die area, which is not negligible in CMP design, but has rarely researched. We find there are predictable patterns in uncore utility which point towards the potential of a proactive approach to uncore power management. In this work, we utilize artificial intelligence principles to proactively leverage uncore utility pattern prediction via an Artificial Neural Network (ANN).

Even though the uncore takes non-negligible portion of CMP power consumption, processor cores still exist as major power consumers. For core DVFS, We explore a novel approach with the potential to achieve synergistic energy-savings *and* performance gain in chip multiprocessors (CMPs). In current designs, performance must typically be traded-off to achieve energy savings or, conversely, performance gains come with significant energy overhead. Resources shared by processor cores, such as

on-chip interconnect and shared memory, play an increasingly critical role in determining the overall CMP performance. Our key observation is that per-core DVFS can be used as a client regulation mechanism for the shared resources. Based on this observation, we propose a new DVFS technique inspired by TCP Vegas, a congestion control protocol from the IP-networking domain.

In addition to uncore in CMP, main memory is also critical shared resource in total system. As uncore is critical resource for CMP performance while occupying critical portion of total CMP energy consumed, main memory is also critical for total performance and accounts for large fraction of total energy consumption. Most conventional approaches focused on utilization of cores and memory only for memory power management. We found, however, the uncore plays an important role of total system performance and its utilization must be considered as well for memory power management. From the observation, we propose shared resource utilization aware power management technique for main memory. Our technique chooses low V/F level of memory for some congested case in uncore, and so derives negligible performance degradation while saving more energy by the low V/F level. We also proposed coordination policies to avoid oscillation issues among individual DVFS techniques (i.e. over energy saving or over performance increment).

Full system simulations on PARSEC benchmarks show that our coordinated technique reduces total energy dissipation by over 47% across all benchmarks with less than 2.3% performance degradation.

DEDICATION

To my wife

ACKNOWLEDGEMENTS

First of all, I would like to give my appreciations to my advisers, Dr. Jiang Hu and Dr. Paul Gratz. I was very grateful to have had working with them during my doctoral studies. Their advises were very helpful for my researches and even beyond researches for my future life. Especially, I would like to take this chance to thank Dr. Jiang Hu for taking care of my family when I had my little kid. Also, I would like to thank my committee members, Dr. Le Xie and Dr. Radu Stoleru. Their comments and advises about my dissertation made my dissertation and researches more concrete and robust.

My appreciations go to all my friends in our research group. Also, I thank Chia-Yu Wu and Dr. Yong Zhang for your advises and discussions. I could have pleasant life during my studies at Texas A&M through the discussions with you.

I also thank my father and mother to encourage me pursuing my doctoral degree. Sincerely, I appreciate for your dedication of your whole life for me. Also, I would like to thank my sweeties, Ashley and Aiden to wake me up with big smiles every day. From deep down in my heart, I would like to give special thanks to my wife, Eunyoung Lee for your dedicated love to me.

Finally, the fund supporting from Korean government, National Institute for International Education is acknowledged.

NOMENCLATURE

DVFS	Dynamic Voltage Frequency Scaling
CMP	Chip Multi-processor
V	Voltage
F	Frequency
ANN	Artificial Neural Network
PI	Proportional Integral
LLC	Last Level Cache
PCU	Power Control Unit
ED	Energy Delay product
E	Energy
D	Delay(Run-Time)
RTT	Round Trip Time

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Historically, advances in transistor process technology yielded scaling performance and energy efficiency together with scaling transistor density, a phenomenon known as Dennard scaling. Recently, however, Dennard's scaling [16] has broken down, leading to much tighter constraints on power consumption with each passing process technology generation. As a result, industry and academia have shifted to many-core, chip-multiprocessor (CMP) designs as a means to utilize increasing transistor density to efficiently gain performance. As scaling continues, however, energy and power management promise to be continuing concerns.

The "uncore" in modern CMPs, consisting of an on-chip communication fabric and shared LLC, now occupies as much as 30% of the overall die area [33]. Even though the die area of the uncore is smaller than CMP cores and their own private caches, the portion of the uncore, 30%, is not negligible in CMP power consumption. Also, uncore plays an important role of total system performance as a shared resource of cores and bottle-neck between CMP cores and external memory. Regarding that current work-load varies from core-intensive to uncore or memory-intensive, uncore becomes more critical resource for some far uncore, memory or both-intensive applications.

With the importance of uncore power management, the power management for the CMP cores still exist as a main concern regarding that 70% of the overall die area is responsible for CMP cores and private caches. Also with increasing core-counts in CMP designs, the shared resources, such as interconnect and cache, have become critical to the overall performance. Properly allocating these shared resources among threads or tasks is essential for efficient performance scalability. The goal in

this context is to ensure that all clients (threads or tasks) achieve completion with low target delays and by doing so receive fair access to shared resources on chip. Interestingly, we find that there is a synergy between power management policies and techniques that achieve fair resource allocation for concurrently executing threads.

In addition to the CMP, main memory in computer architectures is usually adopted as a cost efficient memory device compared to cache memory [25]. Chip Multi-processor(CMP) includes multiple cores, private caches for each core and a shared cache for all cores. Even though cache memory is used as the fastest memory device, cache memory is not cost efficient to handle large size of data. To implement cache memory, it takes larger area than main memory. In other words, main memory plays an important role as a faster memory device to handle larger data. Certain types of main memory such as DRAM keep refreshing to hold data and takes comparable portion of energy consumption to CMP [2]. Therefore, power management for the main memory is necessary as much as CMP power management. In CMP power management, shared resource has been proven as critical resource for high performance and energy consumption [9, 62]. As such, main memory is also shared by all cores and shared cache of CMP, thus important for total performance and accounts for large fraction of total energy consumption.

To struggle with the power issue in CMP architecture, many researches has been conducted. Our first work focuses on dynamic voltage and frequency scaling (DVFS) for the CMP uncore. Although DVFS has been extensively studied in the literature [6, 22, 41, 47, 52, 53, 56], they have paid less attention on the uncore's power consumption. That is, they are restricted to either core DVFS or DVFS voltage/frequency (V/F) domains partitioning around cores, merely including a slice of the uncore. Classifying the uncore into separated V/F domains incurs large performance overhead in communication, as packets must pass between different V/F

domains, experiencing synchronization delays at each hop.

In this work, we consider a different, but practical scenario where the entire uncore comprises a single V/F domain. In such setting, data need not experience synchronization delays in the network-on-chip (NoC) fabric interconnecting the cores. A few recent works seek to address uncore and/or NoC power management via DVFS [39, 10, 9]. These approaches are largely *reactive*, i.e., they set V/F state based purely upon past uncore state. Such approach works well only when the uncore load and its performance impact change slowly. In realistic applications, however, uncore load and its utility (i.e. the system's performance sensitivity to the uncore) often have abrupt changes. A reactive controller, such as a rule-based [39] or Proportional Integral (PI) controller [10, 9], may tune the uncore V/F to a higher level due to high load or poor performance observed in the previous interval. Utility, however, can suddenly change in the next interval, and a V/F increase consequently wastes energy that could otherwise be saved.

To improve upon this behavior requires a more proactive approach – a technique which can predict the load and make corresponding decisions. This requires the controller to maintain knowledge that associates past application behavior patterns with future uncore utility. In this work, we explore the use of an Artificial Neural Network-based (ANN) technique to achieve the desired proactive/predictive control [43]. ANNs are a general neural model derived from biological systems, that can be applied to approximately classify nonlinear and dynamic behaviors. As such, ANNs are particularly useful in identifying patterns in a current system state and predicting future behavior accordingly. ANNs have been used in branch prediction [61] and predicting traffic congestion hotspots in NoCs [31]. For the purposes of this work, we propose that the ANN is fed by the individual measured state of each core, together with some history of recent state in those cores. Based upon this

input, the ANN will predict the future utility of the uncore, and the V/F state will be traced and set appropriately. This predictive control scheme allows faster, more proactive responses to abrupt state changes. The ANN's multi-input control is a clear advantage versus the single-input PI control [10] where information loss occurs during the data aggregation.

ANNs obtain their predictive ability via training of their internal parameters (weights). Thus, in typical ANN applications, a priori training set including inputs and desired output is required. For typical general-purpose processor implementations, difficulties exist in developing representative training sets, as this requires offline analysis of captive applications assumed to be similar to the expected workload of the processor. Architecting an efficient training mechanism without a priori knowledge of the workload's behavior is a significant challenge which we address in this work. We propose a novel technique in which a simple PI controller is used as a secondary classifier during a purely online training phase, dynamically pulling the the ANN up (by its bootstraps) to accurate prediction. Since the PI controller itself has been shown to produce reasonable power management, we propose that both the ANN and the PI controller work in tandem once the ANN training phase is complete. In this work we investigate novel policies determining which controller, the ANN or PI, should decide the next V/F state of the uncore, as well as when and how to modulate ANN online training during system runtime.

In addition to the uncore power management technique through DVFS, our work also considers DVFS techniques for cores and private caches as well. We observe that power-efficiency and shared resource management can be synergistic. There are two mechanisms by which this is true: First, and most obviously, shared resource management can balance resource utilization among threads and tasks, improving system's performance without impacting energy consumption and thus improving efficiency;

Second, Power management techniques, such as dynamic voltage/frequency scaling (DVFS) can be used as a means to implement shared resource management. Thus the shared resource management policy actually *drives* power management policy in a direct sense. For example, when an on-chip network is congested, decreasing a particular core's voltage/frequency reduces its packet injection and therefore improves the network quality of service for the other cores. If well managed, core performance may not be impacted as it must wait for data from the congested network in any event. Further, the reduced congestion can allow other cores which may be more performance critical to make greater forward progress by reducing network and memory latency.

Based on this observation, we develop a simple, low-overhead, per-core DVFS policy inspired by TCP Vegas, a network congestion control technique that aims at ensuring small queuing delays. Simulations on PARSEC benchmarks indicate that our policy can reduce energy dissipation by 43% on average without performance degradation (or even performance increment). This work also compares our technique to state-of-art- methodology and shows that our technique is more efficient than other techniques.

Beyond CMP power management techniques, many researches have been conducted for memory power management to reduce entire power consumption. Deng and et al. proposed active low-power modes for main memory through dynamic voltage frequency scaling [15]. Also many researches on power management of main memory have been proposed [12, 14, 45]. However, the researches focus on main memory only and do not consider with CMP power management. This would occur too much energy saving with much performance degradation or less energy saving due to over V/F scale. To solve this issue, a coordinated DVFS technique of cores in CMP and main memory [13]. The work solves oscillation issue of operating V/F

level without over energy saving. In this work, we explore coordinated memory DVFS techniques to maximize total energy saving. We do not consider only cores in CMP and main memory, but also shared resource in CMP. In general, lots of requests to main memory requires high voltage(V) and frequency(F) level to maximize system performance even though high V/F level derives less energy saving. Likewise, low V/F level is preferred when there are few requests to main memory for more energy saving. The low V/F level is sufficient to process few accesses of main memory with less performance degradation. However, we found that high V/F level to process lots of requests to main memory sometimes leads performance degradation while less energy saving. In this case, actually, the high V/F level generates congestion in uncore or memory controller, more requests to main memory or more cache coherence traffic. Our technique chooses low V/F level for the case, and so derives performance increment(or less performance degradation) by solving congestion while saving more energy by low V/F level.

The individual contributions of this work are as follows:

- We develop an ANN-based mechanism for uncore power management based upon offline training.
- We augment the offline-trained ANN controller with online self-adaptation and show that it improves the energy-delay product by 8% compared to a state-of-the-art previous work [9].
- We propose a novel, purely-online, tandem ANN-PI power manager, which further improves energy-delay product by 27% versus prior techniques [9] while removing the need for offline training. Compared to constantly high uncore V/F, the performance degradation from our approach is less than 3%.

- We develop resource sharing driven DVFS technique for CMP cores power management and show that it improves performance by 2.9% with 43% energy savings compared to maximum frequency simulations.

- Our proposed cores power management policy is compared to a conventional work [24] and show that our work improves performance by 19.2% with similar energy savings.

- We propose coordinated uncore power management policy to cope with other DVFS policies.

- We propose shared resource aware power management technique for memory and shows 66% energy saving of main memory without no performance degradation (18% energy saving of total energy).

- Our proposed memory DVFS technique is compared to a conventional work [15] and shows that our work saves 24% more energy and improves performance by 3.5%.

- We propose coordinated policy among cores, uncore and memory and compared to a conventional work [13]. Our work shows that our work saves 12% more energy and improves performance by 0.4%.

Our full system simulations on PARSEC benchmarks shows 66% energy saving of main memory and 18% energy saving of total energy consumption. In addition to solely main memory DVFS, our coordinated DVFS technique for core, uncore and main memory shows 47% energy saving with less than 2.3% performance degradation.

2. BACKGROUND AND RELATED WORK

2.1 Dynamic Voltage Frequency Scaling on CMP and Related Work

Many works utilize Dynamic Voltage and Frequency Scaling (DVFS) techniques to save energy; often, these schemes are independently applied to either the NoC or onto the cores to save power, but not holistically. The earliest work, utilizing only dynamic voltage scaling (DVS) by Shang et al.[56] regulated the voltage of individual NoC links independently to save power during periods of link under-utilization. Soteriou et al. also explored DVFS regulation of links in NoCs. In this work DVFS-specific instructions were inserted into a given application, based upon profiling, to instruct the voltage-frequency regulation of links during run-time [58]. Son et al. proposed simultaneous CPU-NoC link DVFS for a specific application – parallel linear system solving [57]. Luo, et al., combined NoC link DVS with task scheduling of embedded systems [41]. Ogras, et al., applied state-space control for DVFS on tile-based designs where the NoC was partitioned and associated with processing cores [52]. Mishra et al. examined DVFS in NoC router designs [47]. The work of Guang et al. [22] partitioned a multi-core chip into voltage/frequency islands and its NoC was also regionally mapped onto those islands. They proposed a rule-based DVFS control for each island according to queue occupancy. Next, Rahimi, et al., proposed another rule-based DVFS based on both link utilization and router queue occupancy [53]. Bogdan, et al., described a DVFS approach based on a fractional state model, where the NoC was also partitioned to be associated with each voltage/frequency island [6]. There are very few previous works addressing DVFS for caches. Flautner et al. presented one such work, which applied DVS to individual cache lines [19].

These previous works all partition the NoC or caches into fine-grained voltage/frequency domains. Another realistic scenario is that the NoC, or uncore, constitutes a single V/F domain, such that the interfacing overhead can be avoided [39, 10, 9]. Liang and Jantsch [39] tuned the voltage/frequency state of the NoC according to network load as predicted by injection rate. Network congestion, however, is often a poor indicator of the entire chip's performance. Further, the DVFS policy in this work is a simple rule-based approach. To capture the impact of the uncore upon overall system performance, Chen, et al. [10], proposed an approach using AMAT (Average Memory Access Time). They employed a PI (Proportional and Integral) controller to implement their DVFS policy. In a recent work, Chen et al. [9] developed the concept of critical latency, the product of LLC throughput demand and the latency of the LLC and NoC, as an expression of uncore utility. This formulation brings significantly more energy savings than any prior work to-date. A dynamic reference technique was introduced for the PI controller which also facilitates additional energy-efficiency improvements. Collectively, these three approaches can be broadly classified as *reactive*, i.e. the V/F state for the next control interval is set based upon the current state and some limited amount of history. In this work, we propose a *proactive* mechanism, in which an ANN is used to detect program phase patterns exhibited in uncore utilization demands. Through finer ANN-based predictions, the controller can make the uncore V/F level better trace the uncore utility changes, and discover opportunities for additional energy savings without degrading the performance of the uncore.

Bitirgen and et al. examined the use of an ANN to manage shared resource allocation in a multicore environment [5]. They show an ANN can be an effective tool for complex management problems within a given hardware budget. Unlike this prior work, here we examine the use of an ANN for a different problem power

management of the LLC and interconnect. Further, we explore the means of using a secondary classifier to provide online training and collaborative control.

Main memory is well known as a cost efficient and essential component in computer architecture. And, its power consumption compared to CMP (cores and uncore) takes large portion and the system requires power management for the memory parts. One work about memory power management technique utilizes estimated time for certain work-load [15]. The policy selects memory V/F level based on the estimated time for the certain work-load. They proposed an OS-level power management technique for memory through run-time estimation and used simplied model for accurate estimation.

Some work also focus on Quality-of-Service to enhance fairness and entire performance. Memory QoS techniques include fair queuing scheduling for its access [50], request grouping [49] and fairness-driven source throttling [17]. Cache QoS is obtained by enforcing priority-based capacity limit among threads [27] and access bandwidth allocation [51]. Coordinated QoS among NoC, cache and memory is studied in [37].

We proposed an on-chip memory power management technique and utilized measured performance counters for cores, shared memory and memory. We observed performance of the shared resource beyong memory itself affects the performance of memory. Thus, our technique which is based on measurement can be used in any architecture.

Many researches about independent power management technique have been proposed [26, 52] Even though the techniques are efficient for the specific component, it would affect other power management techniques for other connected components. Without awareness of other power management results, each techniques would overreact and oscillate between too far up and down.

To avoid the oscillation issues, a coordinated power management technique has

been proposed [13]. The work presented the coordinated power management for cores and main memory. In the work, they proposed new policy for cores' DVFS technique while using a pre-proposed technique for memory power management. In addition to new technique for core power management, they proposed coordinated DVFS through awareness of results of other power management techniques. They suggested OS-level power management policy which has longer control interval(i.e. 5ms of epoch) and used simplied model for more accurate estimation of processing time. Also, they assumed all running applications are single threaded application for more accurate estimation of run-time.

In our proposed work, we proposed a coordinated DVFS technique for cores and memory and additionaly we also considered shared resource, uncore, power management. Uncore takes large portion of energy consumption in CMPs and so is critical for power management of entire CMPs [9, 42, 62]. Also, uncore affects to other components' power management. In our work, we proposed an on-chip power management technique which is faster than OS-level power management. Our policy can also be used in any architectures because it is based on measurement of control interval.

2.2 Uncore Power Management

We consider a common case in multicore processor design where the entire chip is composed of an array of identically-sized tiles. Each tile contains a processor core and private caches. The communication fabric is a 2D mesh NoC with one router residing in each tile. A shared LLC is partitioned into slices and distributed uniformly among these tiles. The NoC and the LLC together are referred to as the *uncore* system. We further assume that the CMP contains a Power Control Unit (PCU) [11]: a small micro-controller with direct control of the uncore's V/F state via memory-mapped

micro-architectural registers, which emulates our proposed power management policy in software. This PCU is associated with one of the central tiles in the 2D mesh as indicated by the darkened tile #6 of Figure 2.1.

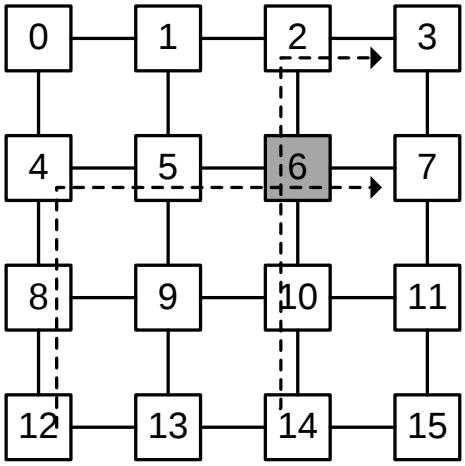


Figure 2.1: 16-core CMP in a 4×4 2D mesh array. The darkened tile indicates location of the Power Control Unit (PCU). The dashed lines indicate paths traversing the NoC [62].

Similar to the design originally proposed by Chen et al. [10, 9], we assume that data (*i.e.* L1 Miss rate, the L2 Miss rate, etc.) used in measuring uncore utility, are encoded into the unused bits in the packet headers being routed onto the NoC. This data is opportunistically collected when these packets pass through the router containing the PCU (Figure 2.1). This approach minimizes the overhead of monitoring, since no extra status packets are created, and there is no need for a secondary overlay status network to convey statistical uncore utility information. Although this implies some staleness in the collection of status data, Chen et al. found that, with appropriate extrapolation, the data obtained produces results nearly indistinguishable from omniscient data collection for the $50K$-cycle control intervals [10].

2.2.1 PI Controller

As a basis of comparison and as a sub-component of our design we also utilize a proportional-integral (PI) controller [10, 9]. A PI controller has two components, the proportional "P" component calculates error, e_t, as the difference between the reference value and measured output in a closed loop. While the P component achieves steady-state rapidly, it is highly sensitive to noise and thus can be vulnerable to multi-core system input patterns which can change dynamically. To increase robustness, the integral "I" of past error is added in a weighted sum. The final output, u_t of the controller is calculated as shown in Equation 2.1. K_p and K_i are the proportional and integral error gain, respectively, and are typically determined empirically.

$$u_t = K_p e_t + K_i \sum_{k=1}^{t} e_k \tag{2.1}$$

2.3 Concept of Artificial Neural Networks

An ANN is an information processing paradigm, inspired by biological neural networks, that attempts to capture the learning behavior, response behavior and general functionality of a biological central nervous system, so as to emulate a form of intelligence artificially. An ANN consists of computation nodes called neurons and interconnections between them, called synapses. ANNs are used to determine relationships between sets of input data to sets of output data, so as to identify and understand patterns.

ANN networks are usually organized in layers of neurons, where information processed from each subsequent layer is fed as an input to the next layer, until the last layer computes a useful result. This model, employed in this work, is known as the multi-layer perceptron ANN. A typical single neuron model is depicted in

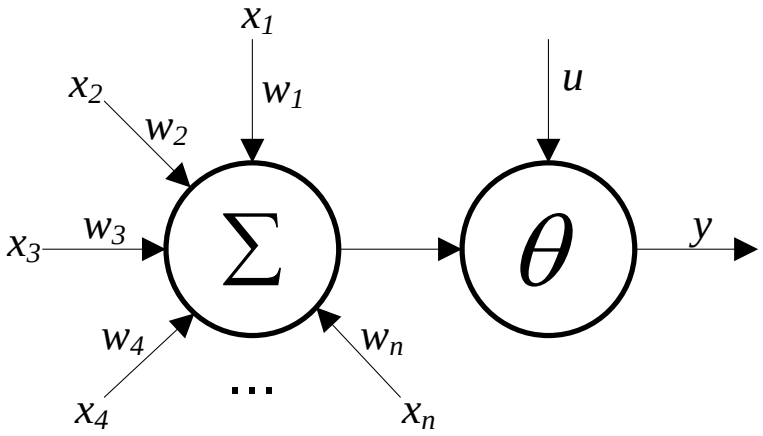


Figure 2.2: Model of a single neuron [62].

Figure 2.2, and is defined by Equation 2.2 [43]:

$$y = \theta \left(\sum_{j=1}^{n} w_j x_j - u \right) \tag{2.2}$$

In this equation, $x_1, x_2, \cdots, x_n$ are its inputs, $w_1, w_2, \cdots, w_n$ are the weight parameters, u is the threshold parameter, θ is the activation function and y is its output. The activation function can take various forms, *e.g.* if θ is a step function, the output changes from 0 to 1 when the weighted sum of the inputs is exceeds a threshold u.

While single neurons can perform classification functions based upon their inputs, this function is limited to linearly separable patterns [46]. Multi-level networks of these perceptron models, i.e. ANNs, do not have this limitation [28]. Here, each neuron is treated as a node, forming directed graph with input and output edges. An example ANN is illustrated in Figure 2.3, where each circle represents a neuron. The ANN in Figure 2.3 does not contain any cycles, and is therefore called a feed-forward ANN. Other ANN variants exist which have cycles, however, we do not

consider them in this work, so as to reduce complexity.

An ANN is a very flexible framework, capable of modeling many different and complex systems, through configuration of its topology, its activation functions and by tuning its parameters. When the ANN is employed as a controller, its output is the control variable and its inputs are from the states/outputs of the system to be controlled. Through a learning procedure, the ANN weights are tuned to associate certain input patterns with a desired output(s).

There are two general forms of ANN learning algorithms: supervised and unsupervised. Under supervised learning, the ANN is typically trained iteratively with a data set that has known solutions starting from an arbitrary set of parameters. In each iteration, the ANN output is compared to the known solution, and the parameters are tuned such that the difference between them is reduced or converges, i.e. they form a "good match." For the simplest ANN, one that has a single neuron, each input weight w_j is updated by

$$w_j(t+1) = w_j(t) + g \cdot (d - y) \cdot x_j \tag{2.3}$$

where d is the known solution, g ($0.0 < g < 1.0$) is a learning gain factor and t indicates iteration index.

For a multi-layer network, the learning procedure includes two passes of backward traversals along the network, from the output(s) to the inputs. The errors are back-propagated in the first traversal and the edge weights are updated during the second traversal [23]. We use a 2-layer ANN topology in Figure 2.3, as an example, to illustrate this process. The error, δ_k, is defined as $\delta_k = d_k - y_k$ for each output node k. For each edge (j, k) between the middle layer and the output layer, its weight is $w_{j,k}$. The errors are back-propagated to the middle layer and the error δ_j at each

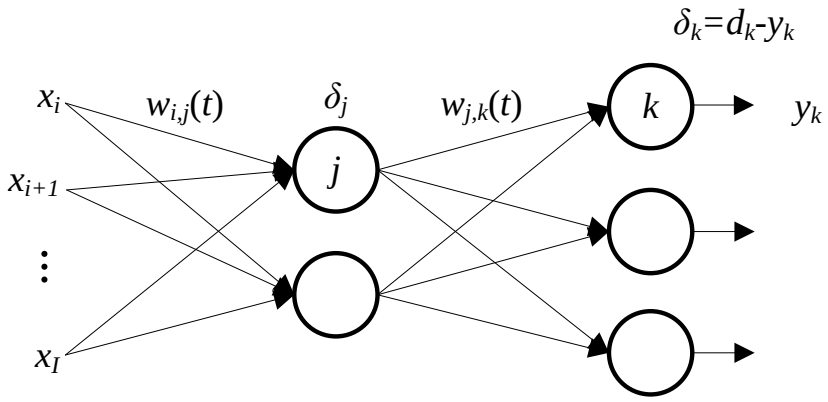


Figure 2.3: Multi-layer feed-forward ANN [62].

middle layer node is obtained by

$$\delta_j = \sum_{k=1}^{K} (w_{j,k}(t) \cdot \delta_k) \tag{2.4}$$

where K is the fanout of node j. For the edge weight update, we illustrate with the edges from the inputs to the middle layer in Figure 2.3. Let the weighted sum of inputs to node j be $\psi_j = \Sigma w_{i,j}(t) \cdot x_i$. Edge weights $w_{i,j}$ are updated by

$$w_{i,j}(t+1) = w_{i,j}(t) + g \cdot \delta_j \cdot \frac{d\theta(\psi_j)}{d\psi_j} \cdot x_i \tag{2.5}$$

This procedure is repeated for all edges in a layer-by-layer backward traversal of the network.

2.4 Quality of Service in CMP

In a CMP with shared resources that have a specific bandwidth, such as on-chip networks, caches and memory controllers, contention resolution can shape overall performance. For example, if a few cores dominate memory access, fairness suffers

and the other cores may form bottlenecks that harm the overall application performance [17]. Contention resolution is a central subject of Quality-of-Service (QoS). The exact concept of QoS is somewhat complicated and can be interpreted in different ways depending on one's emphasis. While Grot, *et al.*, summarize ten attributes of CMP Network-on-Chip (NoC) QoS [21], in practice, most prior work in CMP QoS is focused on service prioritization [7, 27], service guarantees [35, 21, 20, 60], service fairness [50, 51, 17] or some combination of these. Ultimately, however, for most CMP applications, the focus is overall CMP performance and application runtime.

QoS can be achieved through a combination of arbitration at the service side, service categorization and regulation at the client (processor cores) side. For on-chip networks, typical QoS approaches include source throttling [**?**], router arbitration [21], or a combination of both [35]. Source throttling can be based on assigned injection rates [35], application types and network congestion [**?**]. Global QoS arbitrates according to packet age [36] or whether a flow conforms to its assigned rate [21]. Memory QoS techniques include fair queuing scheduling for its access [50], request grouping [49] and fairness-driven source throttling [17]. Cache QoS is obtained by enforcing priority-based capacity limit among threads [27] and access bandwidth allocation [51]. Coordinated QoS among NoC, cache and memory is studied in [37].

Existing CMP QoS works mostly treat energy consumption as an implementation overhead to achieve fairness and pay little attention to its interaction with power management.

3. ONLINE LEARNING IN ARTIFICIAL NEURAL NETWORKS FOR IMP UNCORE POWER MANAGEMENT

3.1 Tandem ANN and PI Control

An overview[1] of the proposed uncore DVFS control system is depicted in Figure 3.1. Besides an ANN controller, it includes a PI controller, as the PI controller plays a role complementary to the ANN control and has very low overhead. The center of this system is the coordination between the two controllers. In this section, we will first introduce the ANN controller architecture. Then, we will describe the ANN learning including how to utilize the PI controller for a bootstrapped learning. The last part will be on the new techniques of tandem ANN-PI control operations.

3.1.1 ANN Controller Architecture

The output of our ANN controller is the uncore V/F level. Its inputs should reflect the uncore performance as well as how sensitive whole system performance is to uncore latency, effectively a measurement of the uncore's *utility* to the system. To this end, we adopt the *critical latency* metric introduced by Chen et al. [9], which is defined as

$$\Gamma = \eta \cdot \lambda_U \tag{3.1}$$

where λ_U is the uncore latency and η is the criticality factor. The uncore latency covers the overall request excluding the memory access latency, i.e., NoC travel latency plus LLC access latency. The criticality factor is the product of private cache miss rate and the ratio of *load* instructions versus total instructions. Chen et

[1]Reprinted with permission from "Up by Their Bootstraps: Online Learning in Artificial Neural Networks for CMP Uncore Power Management", by Jae-Yeon Won, Xi Chen, Paul Gratz, Jiang Hu and Vassos Soteriou, 15-19 Feb. 2014, The 20th IEEE International Symposium on High Performance Computer Architecture(HPCA), © 2014 IEEE

al.[9] collect the critical latency data from all cores and average them into a single value as the input to a PI controller. In contrast, an ANN controller can directly process multiple inputs, and is therefore able to utilize detailed, per-core information.

The DVFS control action is performed periodically in every control interval $\mathcal{I}$. Since the ANN controller accepts multiple inputs, it may examine monitored Γ of an arbitrary number of history intervals. Thus, if the ANN controller examines the past m intervals, including the current interval, and there are n cores, then it has $m \cdot n$ inputs.

From the inputs to the output, there can be different numbers of layers of neurons, which implies a tradeoff between capability and overhead. From our experience, a 3-layer structure performs well and has limited overhead. Such a structure is depicted in Figure 3.2. If the uncore V/F has k levels, we use k outputs, each of which indicates the selection of a corresponding V/F level. The value of each output is a number

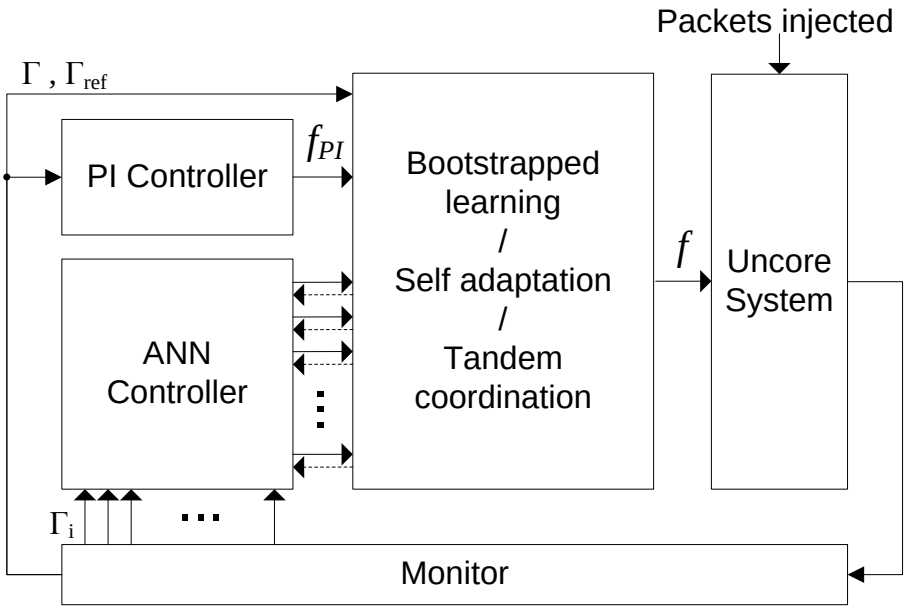


Figure 3.1: Architecture of the uncore DVFS control system [62].

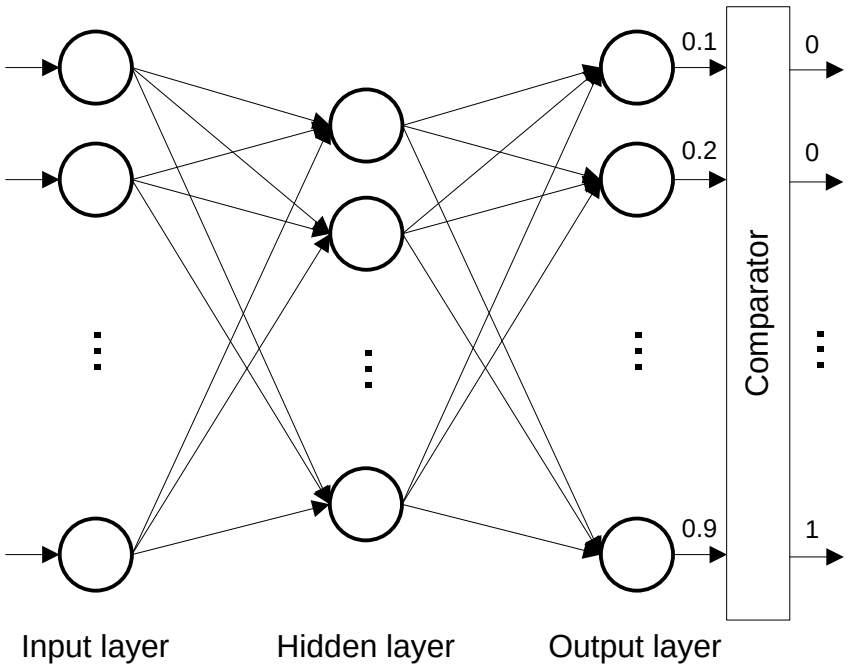


Figure 3.2: Proposed 3-layer feed-forward ANN [62].

between 0 and 1. Since an output can take fractional value, we use a comparator to select the output with the maximum value, and round the other outputs to zero.

For the activation functions, we employ the commonly used Gaussian functions defined by

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}} \tag{3.2}$$

We set a to 1 to maintain the neuron's output dynamic range between 0 to 1. Parameters b and c can be adjusted according to input values which is introduced in Equation (3.1). The learning algorithm here is the common back propagation algorithm described in Section 2.3.

3.1.2 ANN Learning

ANN learning is a procedure of identifying/improving the weight parameters based upon the expected output(s) for a given input set. ANN learning can be carried out offline or online. The basic supervised learning is introduced in Section 2.3. In Section 3.1.2.1, we describe how to apply traditional supervised learning offline for our ANN controller. New online self-adaptation techniques for tuning the ANN controller are discussed in Section 3.1.2.2. We propose an "up by the bootstraps" learning technique using PI control as a secondary classifier in Section 3.1.2.3.

3.1.2.1 Offline Supervised Learning

In offline supervised learning, first a set of cases with known solutions for ANN training is created. Since the DVFS control is carried out periodically for each control interval, this set should include the target uncore V/F level of every control interval. The target level should be the optimal level defined by the minimum uncore V/F level such that the runtime increase is no more than $\alpha\%$ compared with the highest V/F level, where α is a parameter. Ideally, the optimal V/F level can be found by enumerating all combinations, e.g., simulate all V/F levels in interval $\mathcal{I}_i$ and then simulate all V/F levels in interval $\mathcal{I}_{i+1}$ for every case at $\mathcal{I}_i$. By approximation, we enumerate uncore V/F levels for the entire trace, i.e., if there are k V/F levels, the entire trace is simulated for k times, each with a different uncore V/F level. These simulation results are partitioned into control intervals and the target V/F level is chosen for each interval.

The interval partitioning starts with simulation result of the highest frequency, i.e., uncore frequency is f_{max}, and each interval consists of κ clock cycles. Finding the corresponding intervals of other simulations with different uncore V/F is challenging, as the executed instruction count in multithreaded benchmarks tends to vary with

uncore V/F state[2]. In lieu of instruction count we use a count of the number of committed *store* instructions from each thread, as this number tends to be invariant with uncore latency, to determine overall runtime of equivalent intervals from one uncore V/F to the next. For example, if there are 9876 *store* instructions in the first interval for uncore frequency f_{max}, we define the first interval for other uncore frequency traces $f < f_{max}$ by the cycle when the 9876th *store* instruction is committed. This procedure is repeated for subsequent intervals and all k uncore frequency levels. For each interval, the target frequency is the minimum one such that the runtime of this interval is no greater than $(1 + \alpha\%) \cdot \kappa$ clock cycles, where the cycles are in terms of f_{max}.

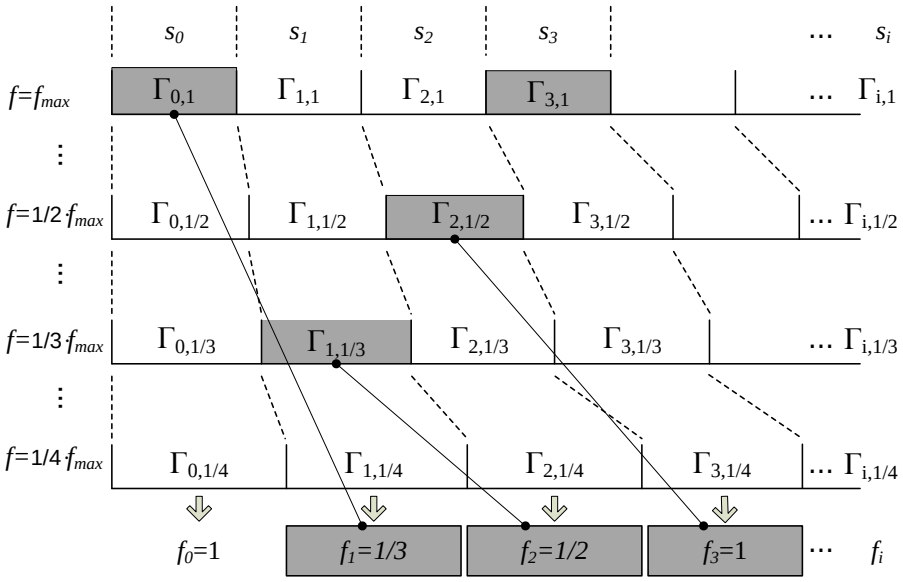


Figure 3.3: Input and target sets for ANN learning [62].

[2]Spin-locks and other synchronization primitives tend to vary in instruction counts when uncore latency is changed.

The ANN controller decides the uncore V/F level of interval $\mathcal{I}_{i+1}$ based on the critical latencies observed from n cores of the last m intervals. Likewise, we use the the critical latencies of all n cores across intervals $\mathcal{I}_{i-(m-1)}$, $\mathcal{I}_{i-(m-2)}$, $\cdots$, $\mathcal{I}_i$ and the target uncore frequency at interval $\mathcal{I}_{i+1}$ as one training set. Figure 3.3 shows a simple example with $m = 1$ where the shaded intervals correspond to the target frequencies and s_i is the number of *store* instructions at interval $\mathcal{I}_i$. In interval 1, the $\Gamma_{1,1/3}$ is the observed critical latency when the uncore operates at $f_1 = \frac{1}{3} f_{max}$. The critical latency $\Gamma_{1,1/3}$ for all cores, and the target frequency $f_2 = \frac{1}{2} f_{max}$ of interval 2, form a data set for the supervised learning. This procedure is repeated for $\Gamma_{2,1/2}$ and $f_3 = f_{max}$, and so on. Once the training data sets are obtained, supervised learning is performed as described in Section 2.3.

3.1.2.2 Online Self-Adaptation

While offline supervised learning can produce good results, it has a weakness. The actual applications may have quite different characteristics from the training cases. In other words, an ANN well-trained for certain workloads may perform poorly on different workloads (i.e. the workloads it was not trained on). To overcome this weakness, we propose two online self-adaptation techniques: feedback adaptation and self-sharpening.

Feedback Adaptation: Feedback adaptation is similar to supervised learning described in Section 2.3 except that the target frequency is obtained online as in the case of feedback control. In typical feedback control techniques, such as PI control [10], the controller attempts to correct the error of the system's output with respect to a reference. The error at interval $\mathcal{I}_i$ is defined by

$$e_i = \Gamma_i - \beta \cdot \Gamma_{ref,i} \tag{3.3}$$

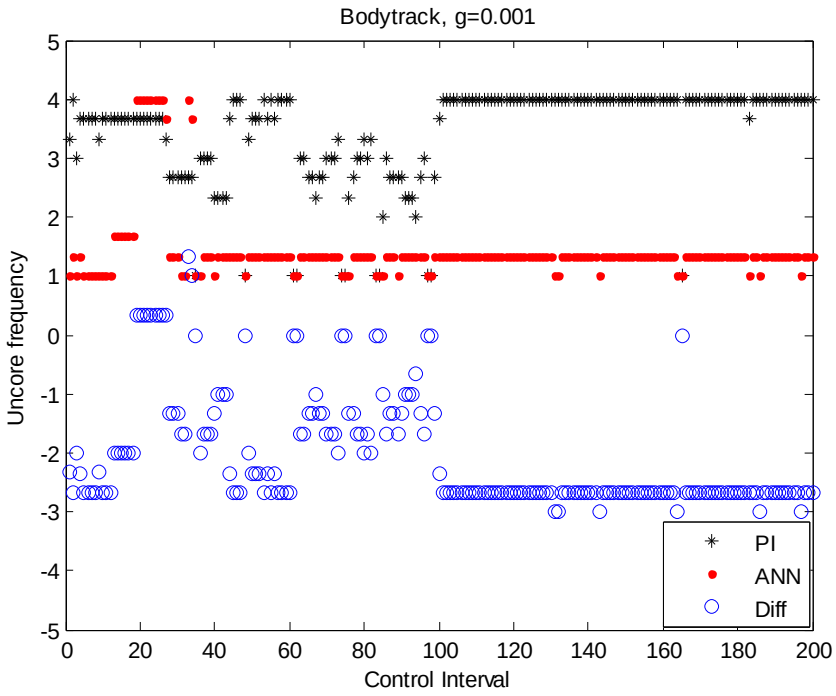
where Γ_i is the critical latency observed during interval $\mathcal{I}_i$, β is a coefficient, and $\Gamma_{ref,i}$ is the reference. We adopt the idea of dynamic reference [9], which is the critical latency when no data packet experiences queuing delay. The coefficient β is typically selected to have a value of 1.1, implying that a small queuing delay during NoC congestion is allowed. The adaptation action is taken only if the error magnitude $|e_i|$ is greater than a certain threshold τ. If there is a large positive (negative) error, we set the target frequency to be one level above (below) the uncore frequency used in interval $\mathcal{I}_i$. This target frequency together with the critical latencies of all cores across intervals $\mathcal{I}_{i-m}, \mathcal{I}_{i-(m-1)}, \cdots, \mathcal{I}_{i-1}$, form a data set to train the ANN once. Such trainings are interleaved with the ANN control operation and thus can be conducted at run-time.

Self-sharpening: The self-sharpening technique is based on the observation that the ANN should ideally have one output of value 1, while the other outputs have a value of 0. In typical operations, however, the ANN produces a set of fractional outputs in $[0, 1]$. Thus, if the ANN output is $\{0.1, 0.2, \cdots, 0.9\}$, the uncore frequency selection is effectively the same as if the output is $\{0, 0, \cdots, 1\}$. Under self-sharpening, we set the ANN output error as $\{-0.1, -0.2, \cdots, 0.1\}$ and back propagate this error through the ANN, as carried out with supervised learning, reinforcing the ANN's decision.

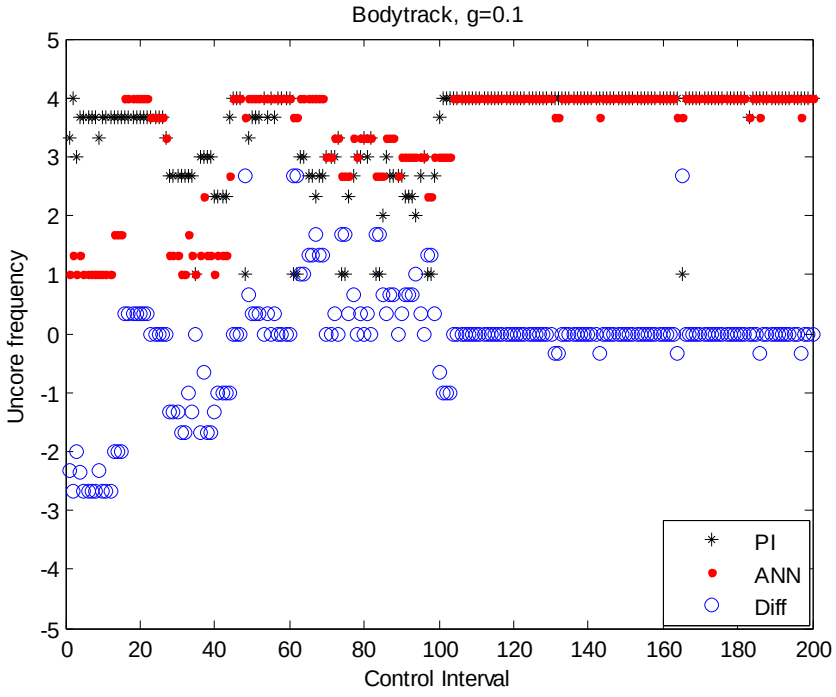
3.1.2.3 Bootstrapped Learning Using a PI Controller

Although the self-adaptation techniques presented in Section 3.1.2.2 can improve upon the performance of offline supervised learning by refining the ANN's behavior according to the actual workload demands, it cannot completely replace offline learning[3]. General-purpose CMP workloads can vary so greatly that developing a

[3]We explored purely online training with the techniques discussed in Section 3.1.2.2, however the results were poor due to the long training time required, these results were dropped from this



(a) gain=0.001



(b) gain=0.1

Figure 3.4: Bootstrapping learning applied to the *Bodytrack* application of the PARSEC benchmark suite [62].

representative set, at design time, for training may be impossible. Therefore, an ANN controller design which does not rely on offline learning is often desirable. Ideally, one would prefer the ANN training as purely online, i.e. during the application's runtime, without the need for an a priori training set. There are, however, several challenges to this form of pure, "up by its bootstraps", online training. For example, online training requires knowledge of the desired output for any given input, at runtime, before the ANN itself is trained well enough to produce that output. Although the PI controller [10, 9] has its weakness, it has very low overhead and it requires very little start-up delay in producing V/F control at its best ability. We thus propose instantiating a PI controller for online training of the ANN. In this "bootstrapped" learning, the ANN learns from the PI controller while the PI controller is controlling the V/F state of the uncore. Hence, the PI control is a surrogate for the training set in supervised learning. The PI controller provides a realistic, dynamically generated training set for online ANN training. When combined with continuous online self adaptation (feedback-adaption and self-sharpening described in Section 3.1.2.2) the ANN can exceed the performance of the PI controller. In Section 3.1.3, we will show that the PI controller may also be used for tandem control once the ANN is trained as well.

The offline supervised learning is often conducted based upon complete traces of many applications. By contrast, bootstrapped learning is performed during the beginning phases of each single application. Hence, it should be much faster and requires a greater learning gain (see Equation (2.3)). Furthermore, bootstrapped learning is focused on the behavior of a single, ongoing application, while the offline supervised learning is intended to be more general. As a result, bootstrapped learning is much more focused to the application at hand and can perform significantly better.

paper for brevity.

Figure 3.4 compares the bootstrapped learning with different gains applied to the *Bodytrack* application of PARSEC benchmark suite [3]. The x-axis holds the indication of the control interval, and the y-axis indicates the uncore frequency selection in terms of the ratio of f_{max} versus uncore frequency; for example, value 4 implies that the uncore frequency is $\frac{1}{4}f_{max}$. The green crosses are the V/F selections chosen by the PI controller, the red dots are those chosen by the ANN, and the blue circles represent the differences between them. Figure 3.4a shows the results with a learning gain of $g = 0.001$, which is common for the offline supervised learning. One can see that the ANN output remains quite different from the PI controller's output after 200 intervals. Experimental results with $g = 0.1$ are shown in Figure 3.4b, which exhibits that the ANN output starts to follow the PI control after approximately 100 intervals.

3.1.2.4 Variable Learning Gain

To further improve the learning efficiency, we propose a variable gain scheme, which can be applied with the bootstrapped learning and the online self-adaptation. In this scheme, the gain g can vary in a range $[g_{min}, g_{max}]$ according to the error e_i defined by Equation (3.3). A small (large) error means the result is close to (far from) a desired one, based on which the learning should be more (less) emphasized and use a large (small) gain. Using this rationale, the variable gain is given by

$$g_i = \begin{cases} g_{max} & \text{if } e_i \leq \tau \\ g_{max} - \left(\frac{e_i - \tau}{e_{max} - \tau}\right) \cdot (g_{max} - g_{min}) & \text{if } \tau < e_i < e_{max} \\ g_{min} & \text{otherwise} \end{cases} \tag{3.4}$$

where τ and e_{max} are two constant parameters.

3.1.3 ANN-PI Tandem Control

As discussed in Section 3.1.2, the ANN controller pro-actively adjusts the uncore V/F level according to its experience, learned either offline or bootstrapped online. This methodology, however, may not always be accurate. One can predict rain from heavy clouds, but heavy clouds do not always yield rain. Alternately, the PI controller always bases its V/F selection only upon current observations. Thus, ANN control and PI control can be viewed as complementary to each other. We propose three ANN-PI tandem control schemes, elaborated next.

3.1.3.1 ANN-Centric Tandem Control

In the first scheme, ANN-Centric Tandem Control, after the ANN is fully trained, both the ANN and PI controllers make their V/F selection for the next control interval. One of their results is chosen to be applied to the uncore. The choice depends on the average error defined by

$$\bar{e}_i = \frac{\sum_{j=i-m+1}^{i} \sum_{l=1}^{n} e_{j,l}}{m \cdot n} \tag{3.5}$$

where $e_{j,l}$ is the error defined in Equation (3.3) for control interval $\mathcal{I}_j$ and core l. This is the average control error among all n cores across the past m control intervals. The choices also rely on the consistency (ξ_i) between ANN and PI control, which is defined as

$$\xi_i = 1 - \left(\sum_{j=i-m+1}^{i} \frac{|f_{j,ANN} - f_{j,PI}|}{k-1} \right) / m \tag{3.6}$$

where k is the number of uncore V/F levels, $f_{j,ANN}$ ($f_{j,PI}$) is the uncore frequency level computed from the ANN (PI) in control interval $\mathcal{I}_j$. In dividing with $k-1$, the difference is normalized to be no greater than 1. The second term in Equation (3.6) is the average normalized difference between the ANN and the PI computed results

in the past m intervals.

$\mathcal{I}_i$	$\bar{e}_i$	ξ_i	$\mathcal{I}_{i+1}$
PI	$\downarrow$	$\downarrow$	ANN
PI	$\downarrow$	$\uparrow$	PI
PI	$\uparrow$	$\downarrow$	ANN
PI	$\uparrow$	$\uparrow$	ANN
ANN	$\downarrow$	$\downarrow$	ANN
ANN	$\downarrow$	$\uparrow$	ANN
ANN	$\uparrow$	$\downarrow$	PI
ANN	$\uparrow$	$\uparrow$	ANN

Table 3.1: Rules governing the choice between the ANN controller and the PI controller decision under ANN-centric tandem control. Parameter $\bar{e}_i$ represents the error occurred in previous V/F selections, and ξ_i represents the consistency between the ANN and PI controller decisions [62].

The rules for the choices between the PI or the ANN are listed in Table 3.1. The first row says that the ANN result will be chosen for interval $\mathcal{I}_{i+1}$ if the control in interval $\mathcal{I}_i$ is based on the PI, the average error ($\bar{e}_i$) is small and the consistency between the ANN and the PI results (ξ_i) is low. According to the second row, if the PI is chosen for interval $\mathcal{I}_i$, the average error is low, and the consistency is high, then the PI control result is chosen for interval $\mathcal{I}_{i+1}$. The other rows of Table 3.1 can be interpreted in the same way.

This scheme is intentionally biased in favor of the ANN, only in rows 2 and 7, where the advantage of PI is obvious, is the PI controller chosen for the next control interval. In all the other cases, the ANN result is selected for actual use. The intent under this technique is to select the ANN as soon as it begins producing reasonably accurate results, under the assumption that the ANN can perform better in the long run once training is complete.

3.1.3.2 Eager Tandem Control

We introduce an alternative scheme for the ANN-PI tandem control, which is solely based on the control error e_i (defined by Equation (3.3)) at the control interval $\mathcal{I}_i$. If $e_i > \tau > 0$ ($e_i < -\tau < 0$), where τ is a threshold, and the critical latency is significantly greater (less) than the reference, then the higher (lower) frequency between the ANN and PI results is chosen for the next interval $\mathcal{I}_{i+1}$. The rationale for this technique is the same as that of the feedback adaptation technique described in Section 3.1.2.2, except that it is directly applied to control decisions, while the adaptation is to improve the ANN.

3.1.3.3 Credit-Based Tandem Control

As another variant of the eager tandem control scheme, we concentrate not only to e_i, but also to the method selected in interval $\mathcal{I}_i$. If $e_i > \tau > 0$ ($e_i < -\tau < 0$) and the method chosen in $\mathcal{I}_i$ gives the higher (lower) frequency, this method is more credible and will be chosen again for $\mathcal{I}_{i+1}$. Otherwise, the other method is chosen for $\mathcal{I}_{i+1}$. Although this scheme also uses e_i as in the eager tandem control, the e_i here is employed to compare which method performs better in $\mathcal{I}_i$. The one which performs better in $\mathcal{I}_i$ is assumed more trustworthy.

3.2 Design Implementation

In this section we describe the implementation details including monitored data collection and control computation. For data collection, we employ a similar scheme to that proposed by Chen et al. [10]. As with their work, there is a PCU (Power Control Unit) [11], which is a microcontroller which handles power management for the CMP system. The microcontroller is similar to that utilized in current CMP designs such as in the Intel i7 [33]. Every core collects its critical latency Γ informa-

tion, and encodes it (piggy-backed in the header flit) onto the unused bits of each outgoing packet. If a packet passes by the PCU, even when the corresponding tile is not its destination, the Γ information is downloaded to the PCU. The PCU retains all relevant data in its local memory. We have experimentally verified that the proposed monitor technique incurs negligible error relative ideal monitoring. More details about the data collection design can be found in the prior work of Chen et al. [10, 9].

In our design, all computation required by our schemes is performed *in emulation*, by running software onto the PCU (i.e. there is no actual ANN hardware, the ANN is emulated in software on the PCU). The computation mainly consists of (a) computing the PI control decision, (b) ANN training, (c) computing the ANN control decision, and (d) choosing between the ANN and PI results in the tandem control schemes. Items (a) and (d) exhibit very low complexity, and their overhead is negligible relative to our control interval size. The ANN training process, including self-adaptation and bootstrapped learning, does not block the ANN control computation, and is therefore not timing-critical. We therefore focus here on estimating the computational cost of the ANN control decisions. Table 3.2 shows the ANN control computation runtime with different numbers of history intervals. These data are obtained based on a baseline 16-core CMP design. As it would be expected, the computational overhead increases with the number of history intervals.

# history intervals	Runtime @hidden layer	Runtime @output layer	Total runtime
10	27,513	1,741	29,254
5	13,779	1,741	15,520
1	2,772	1,741	4,513

Table 3.2: ANN control computing runtime in PCU clock cycles [62].

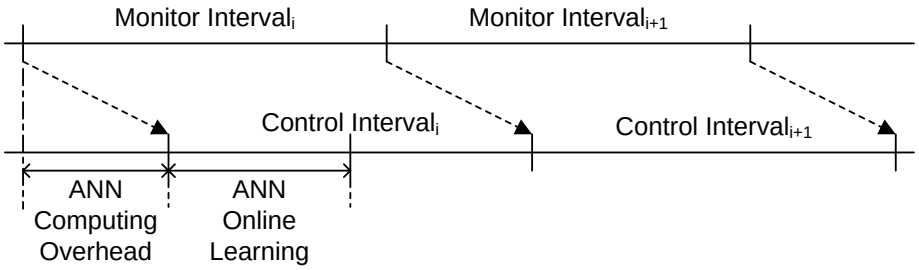


Figure 3.5: Pipelined monitoring and control intervals [62].

As in the work by Chen et al. [10, 9], we assume a control interval to be 50 thousand core clock cycles. The ANN control computation accounts for a significant portion of the overall control interval. In order to minimize the negative effect of this latency, we use two sets of different intervals for the critical latency monitoring and the control output change as illustrated in Figure 3.5. The two sets of intervals are offset by the ANN control computation time, effectively pipelining the overhead. By doing so, the ANN control computing does not block either the monitoring process or the control output change. However, the ANN computation does lead to increased staleness in the monitored data by the time the control decision is implemented. The impact of this computational latency is examined in Section 3.3.2.5.

3.3 Evaluation

In this section, we describe our experimental setup and subsequent evaluation of our proposed techniques.

3.3.1 Experiment Setup

The experimental baseline platform is a 16-core CMP with a 2-level cache hierarchy, split L1i and L1d private caches, and a combined, shared L2 last-level cache. Cache coherence is maintained via a MESI directory cache coherent protocol. The

Parameter	Configuration
# of cores	16
Core frequency	Fixed at 1GHz
L1 data cache	2-way 256KB, 2 core cycle latency
L2 cache (LLC)	16-way, 2MB/bank, 32MB/total 10 core cycle latency
Directory cache	MESI, 4 core cycle latency
NoC	4x4 2D mesh, X-Y DOR 4-flits depth/VC
Uncore V/F	10 levels, voltage: 1V - 2V frequency: 250MHz - 1GHz
Control interval	50000 core cycles
V/F transition	100 core cycles per step

Table 3.3: System parameters used for full-system simulations [62].

NoC topology is a 4×4 2D mesh, with each node/router attached to a single processor core. Table 3.3 summarizes the baseline CMP setup.

Simulation experiments are performed using the gem5 [4] full system simulator, with the *Ruby* memory model and the *Garnet* network simulator [1]. The benchmark applications are taken from the PARSEC shared-memory, multi-processor, benchmark suite [3]. Specifically, we use the 11 PARSEC benchmarks currently supported by our simulation infrastructure, *Blackscholes, Bodytrack, Canneal, Dedup, Ferret, Fluidanimate, Freqmine, Streamcluster, Swaptions, Vips, and X264*. In each case, the entire benchmark is simulated, but only the Region Of Interest (ROI), is evaluated. The performance metric is evaluated as the runtime of the entire ROI. The energy consumption evaluation includes both dynamic and leakage energy. ORION 2.0 [30] and CACTI 6.0 [48] are used to estimate the energy consumption of the NoC and the LLC, respectively, both of which are based on $65nm$ CMOS process technology.

To focus on the evaluation of our uncore DVFS techniques, the core frequency

is fixed at $1GHz$ throughout the simulations. There are 10 uncore frequency levels between $f_{max} = 1GHz$ and $250MHz$. For each frequency, there is a corresponding voltage level between $1V$ and $2V$, which is roughly the minimum voltage allowing correct uncore operation. The control interval is 50 thousand unscaled core clock cycles at $1GHz$ and each step uncore V/F level change takes 100 core cycles (100 cycles per step is sufficient assuming on-die regulation [18]). During V/F transitions, the uncore operation is halted.

The ANN configuration is summarized in Table 3.4. The ANN inputs are the critical latencies as viewed by each of the 16 cores in the past 5 intervals; thus the ANN has 80 first-layer nodes. The hidden layer and the output layer each has 10 nodes. The learning gain g is set to 0.001 for the offline learning. For the bootstrapped learning, the gain is either set at 0.1 or set as a variable value between 0.001 and 0.1, as described in Section 3.1.2.4. The error threshold τ, error bound e_{max}, and consistency threshold ξ are used in the tandem control. The values of these parameters are identified empirically. Each ANN control computation takes 15 thousand core cycles, but does not block any uncore operations (see Section 3.2).

3.3.2 Experimental Evaluation

3.3.2.1 Overall Results

We compare the following 6 methods:

Baseline: the uncore constantly operates at highest V/F.

PI: best method from Chen et al. [9].

Offln+Adpt+TdEager: ANN trained with offline learning, operates with self-adaptation and eager tandem control.

Bstrp+Adpt: bootstrapped learning, self-adaptation and ANN control.

Parameter	Configuration
# history intervals	5
# nodes at input layer	5×16
# nodes at hidden layer	10
# nodes at output layer	10
Offline learning gain	0.001
Constant bootstrapped learning gain	0.1
Variable bootstrapped learning gain	$[0.001, 0.1]$
Error threshold τ	0.001
Max error bound e_{max}	0.1
Consistency threshold ξ	0.6
Computing overhead	15K core cycles

Table 3.4: ANN configuration parameters [62].

Bstrp+Adpt+TdANN: bootstrapped learning, self-adaptation and ANN-centric tandem control.

(Bstrp+Adpt)VG+TdANN: bootstrapped learning and self-adaptation with variable gain, ANN-centric tandem control.

Among the many techniques we investigated, the above includes only the best offline learning-based method and three best bootstrapped learning-based approaches. The results are normalized with the baseline and displayed in Figure 3.6. Compared to the PI control [9], our best method can reduce the uncore energy and the energy-delay product by 25% and 27%, respectively. The performance degradation from our DVFS is less than 3% of the baseline. All applications show improved energy-delay versus PI control except *Blackscholes*. Blackscholes presents some difficulties for the ANN as it has two very short phases (beginning and end of simulation) with many misses, broken up by a long phase with few misses. Hence, the ANN's training on the initial phase is to short to be beneficial for that phase, nor it is useful for the middle phase. Similarly the final phase is ill-served by the training on the middle

phase.

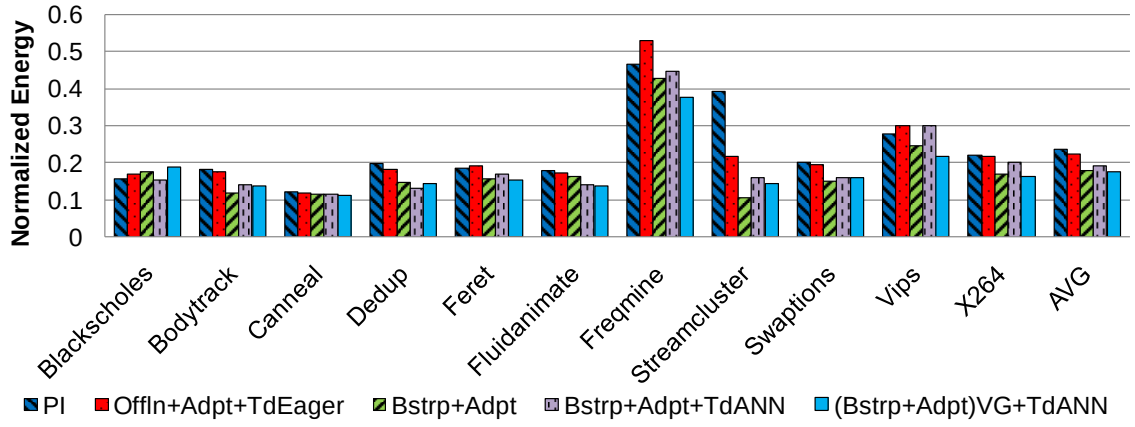
The energy-performance tradeoffs of our offline and bootstrapped learning schemes are depicted in Figure 3.7. To produce the results displayed in Figure 3.7, we found the average energy delay (ED) across all PARSEC benchmarks for each of the three techniques while sweeping many of the various parameters of each technique. The Pareto optimal points for each of the PI control, offline and bootstrapped methods were then plotted as shown in Figure 3.7. As the same figure shows, offline learning-based methods generally dominate the PI control-based methods primarily due to their superior runtime. All PI and offline results are both dominated by bootstrapped learning in terms of both energy and runtime.

In the remainder of this section we explore and analyze the behavior of our proposed techniques across several axes. All of the subsequent results are the average among the 11 PARSEC benchmarks.

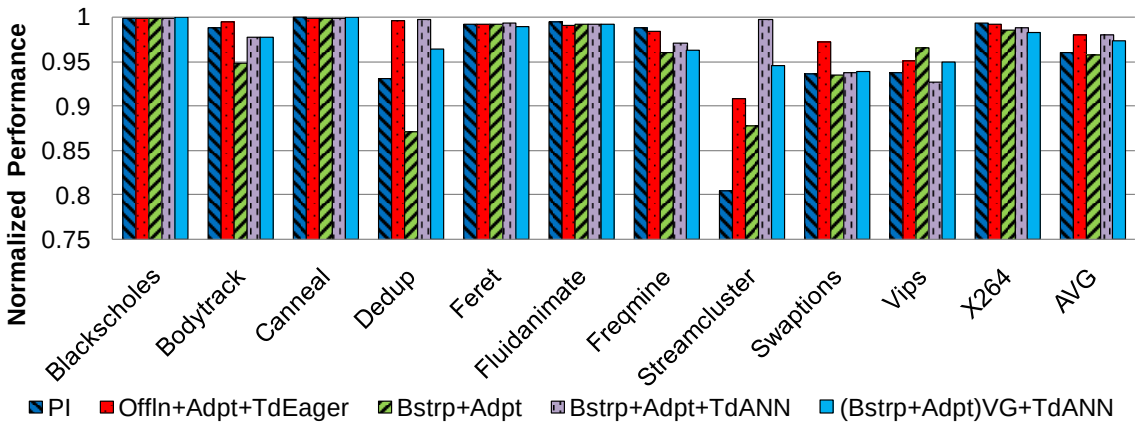
3.3.2.2 Effect of Online Self-Adaptation on Offline vs. Bootstrapped Learning

Figure 3.8 shows the benefit of the online self-adaptation (see Section 3.1.2.2) on two options of the pre-operational learning for ANNs: the offline supervised learning (Section 3.1.2.1) and the online bootstrapped learning (Section 3.1.2.3) to determine the effect of training on a generic set versus training specifically on the application to be run. Under offline supervised learning, the ANN is trained using the entire PARSEC benchmark suite, except the benchmark under test. The number of required iterations depends on the size of learning gain. When bootstrapped learning is conducted initially, we found experimentally that 600 intervals and 0.1 learning gain is sufficient to train the ANN.

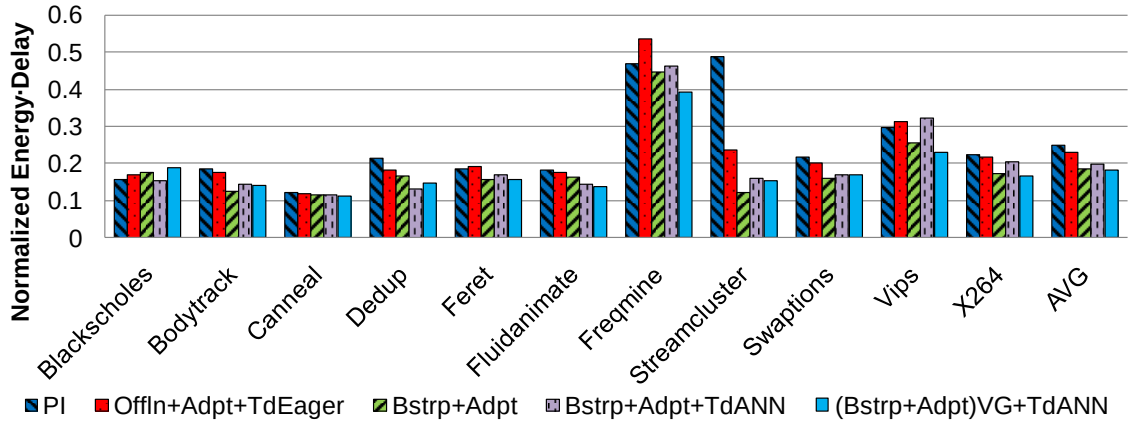
After the training phase, the ANN controls the uncore DVFS without any self-adaptation to isolate the effects of the pre-operational learning as shown to color-



(a) Normalized energy



(b) Normalized performance



(c) Normalized energy×delay product

Figure 3.6: Overall full-system experimental results [62].

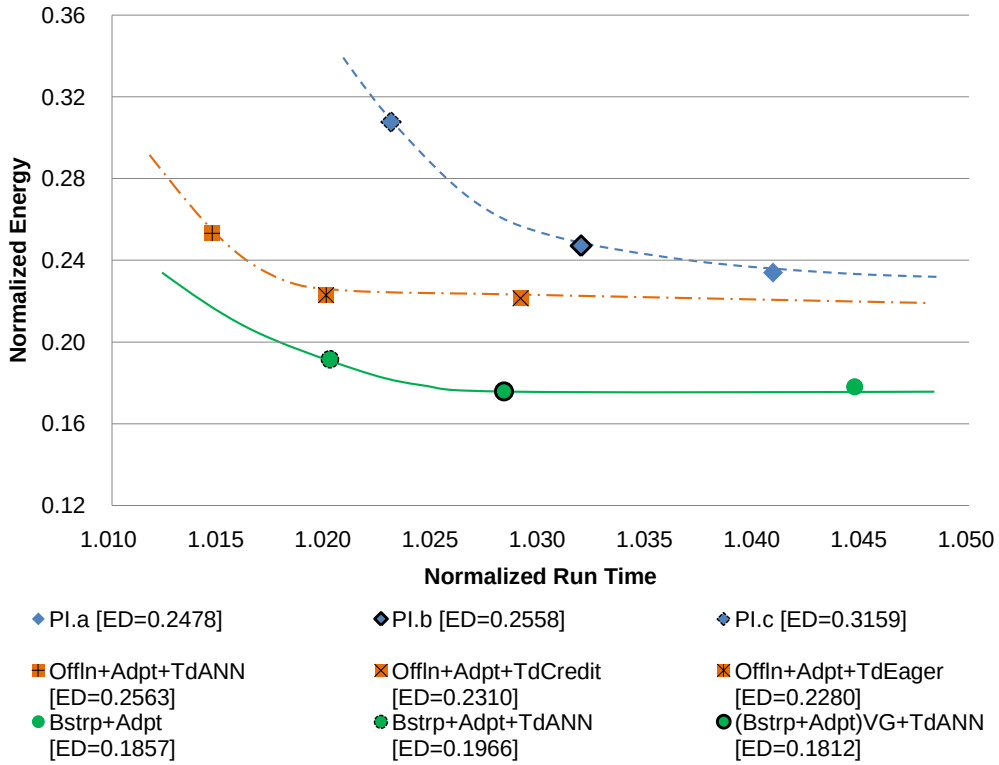


Figure 3.7: Pareto optimal points and normalized energy-normalized run-time curves for PI control-, offline-, and bootstrapped learning-based methods [62].

filled shapes. Blue-filled diamond and red-filled triangle show the results of these two methods. For reference, we also include the equivalent results of the PI controller [9] which is shown as green-filled circle. It is clear that the bootstrapped learning significantly outperforms the offline supervised learning. This result highlights the benefits of learning on the specific application to be run versus a generic set of different applications.

The unfilled shapes show the benefit of the online self-adaptation (see Section 3.1.2.2). From Figure 3.8, it is clear that self-adaptation benefits both offline supervised learning and the bootstrapped learning, significantly improving the energy-delay product. The benefit comes with a trade-off in runtime degradation for greater energy savings.

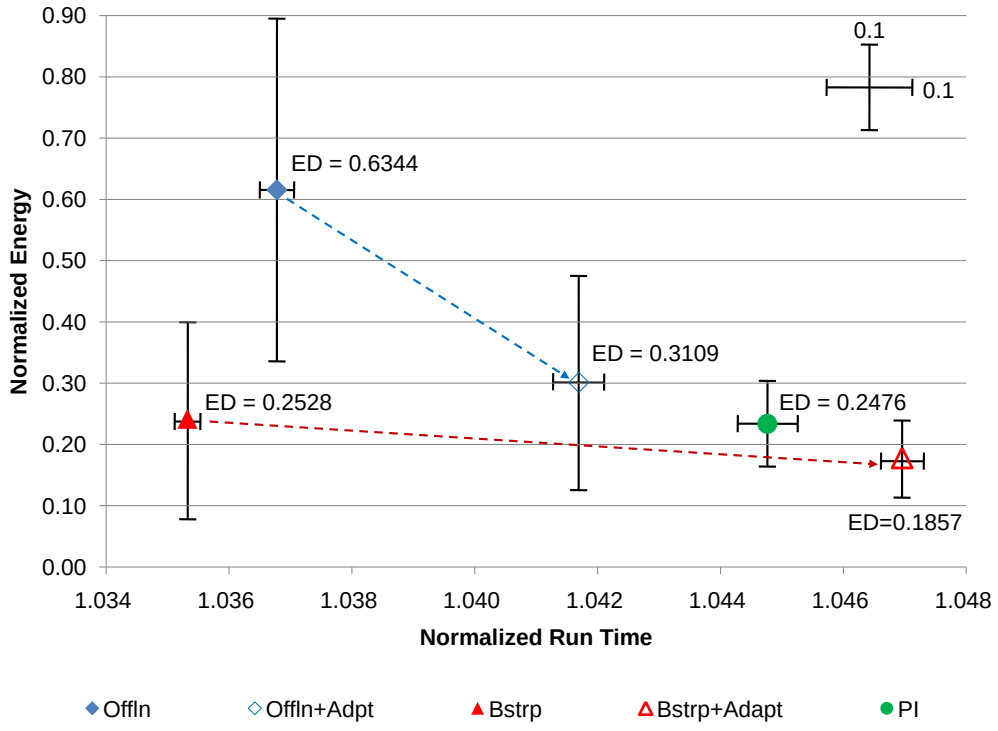


Figure 3.8: Effect of online self-adaptation. ED: energy×delay product [62].

3.3.2.3 Effectiveness of ANN-PI Tandem Control

Three ANN-PI tandem control techniques are proposed in section 3.1.3: ANN-centric tandem (TdANN), eager tandem (TdEager) and credit-based tandem (TdCredit). Figure 3.9 shows the results of these techniques integrated with the offline learning and online self-adaptation. They are compared with the result only when using the offline learning and online self-adaptation. It can be seen that the tandem control techniques improve the overall energy-performance tradeoff.

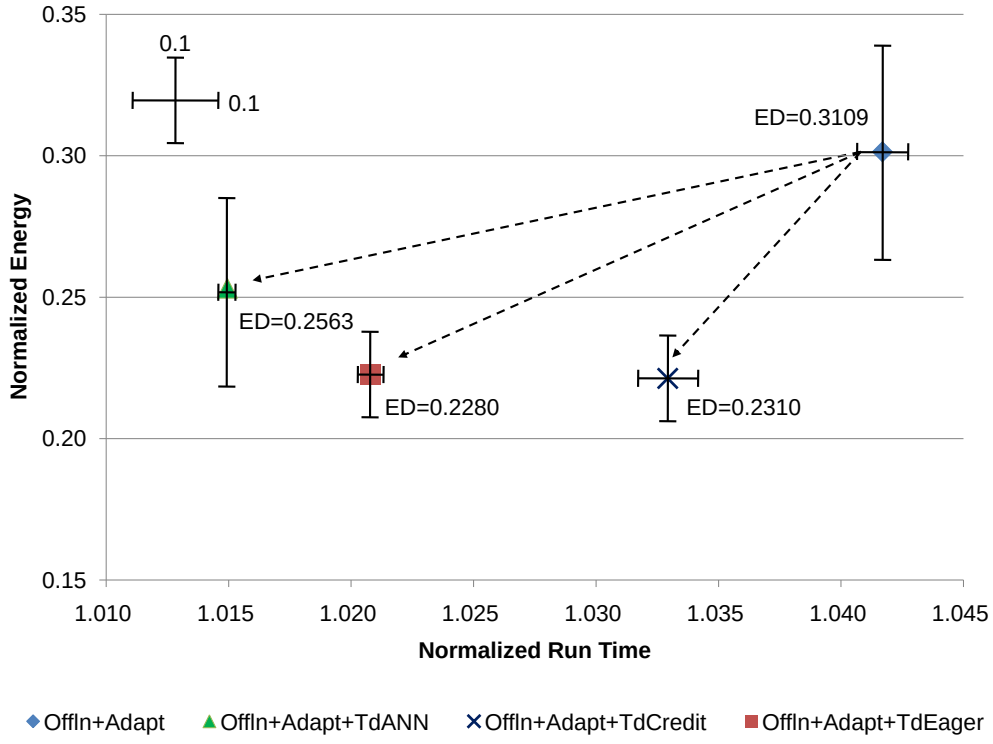


Figure 3.9: ANN-PI tandem control based on offline supervised learning [62].

3.3.2.4 Effect of Variable Learning Gain

In Section 3.1.2.4, the variable gain learning technique is proposed to improve the efficiency of the bootstrapped learning and self-adaptation. Here we show its effectiveness by comparing it with learning under a constant gain using two approaches (1) offline learning + self-adaptation + ANN-centric tandem control, and (2) bootstrapped learning + self-adaptation + ANN-centric tandem control. The results in Figure 3.10 show that the variable gain causes a 2% reduction in energy and a 1% increase in runtime. Overall, it reduces the energy-delay product by $6-8\%$.

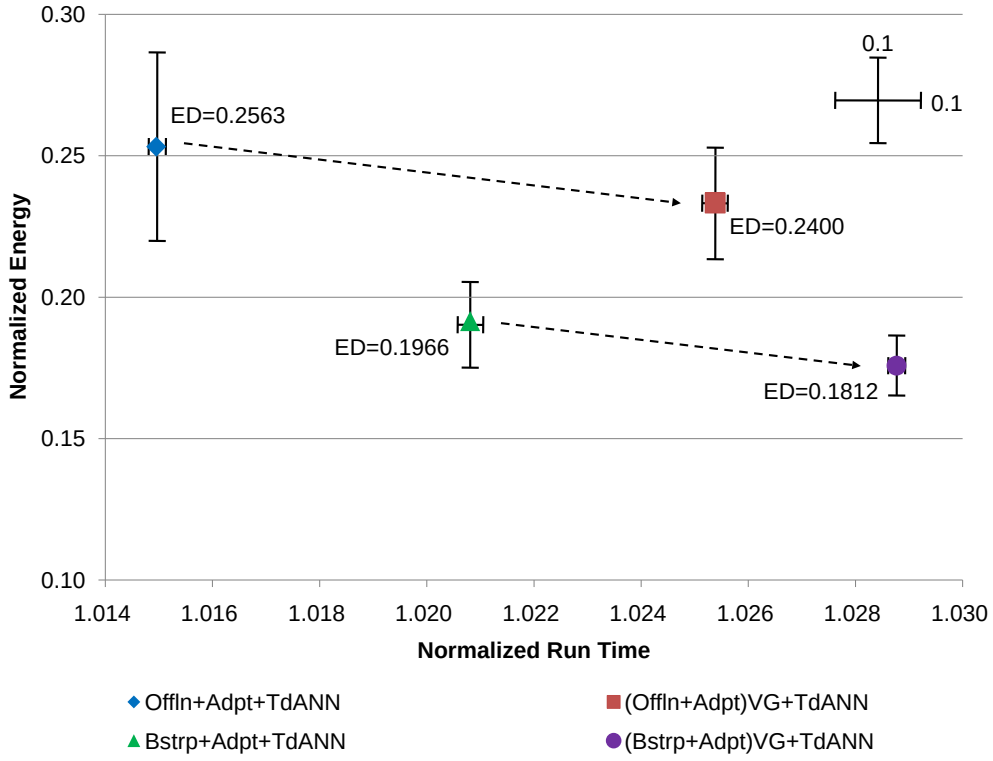


Figure 3.10: The effect of variable learning gain [62].

3.3.2.5 ANN Overheads

We use the bootstrapped learning and the ANN-centric tandem control as a platform to evaluate the impact of the ANN computation overhead.

The ANN computation is performed in the CMP's power control unit (PCU), a small microcontroller on die. The delay for the PCU to compute the next time window's DVFS set point is 15K cycles and online ANN learning computation requires an additional 18K cycles. The ANN implementation thus consumes 66% of the 50K cycles of each interval, as shown in Figure 3.5. Compared to the total CMP power, the incremental power required by the PCU due to the ANN implementation represents a negligible .32% increase. The ANN controller program occupies 7KB of text

and 5KB for data memory in the PCU's memory. We also conducted a cost/benefit analysis of a hardware ANN implementation. Leveraging data from a recent work by Rasheed [54], we estimate the computational delay of the same ANN controller designed in hardware to be a negligible 93 cycles, while the incremental average power increase over the software implementation would be $\sim 49.83mW$, assuming the ANN is power gated when not in use. Due to the reduction in computation latency, the hardware ANN implementation yields a small improvement of less than .2% energy saving and 1% performance increase relative the software implementation. Thus, while the hardware ANN does slightly improve performance, the requirement for extra hardware design makes the software implementation of the ANN controller a more attractive option.

3.4 Conclusions

The CMP uncore (i.e. the shared last level caches (LLC) and Networks-on-Chip(NoC)) constitute a significant and increasing part of overall CMP power dissipation. This work focused on Dynamic Voltage and Frequency Scaling (DVFS) of the uncore. To fine-tune DVFS uncore control, and achieve the best possible power savings, while maintaining high performance in the entire multi-core chip, various Artificial Neural Network-based (ANN) techniques are explored. Conventional ANN approaches rely on offline learning, which is inadequate to handle the large variety of realistic multicore applications. We propose novel approach, wherein a Proportional Integral (PI) controller, which can adapt to short system changes, but lacks long-term pattern recognition, is used in tandem with the ANN, bootstrapping the ANN during the ANN's initial learning process. This is beneficial when the multicore chip swaps between various applications rapidly, or when program phases change rather frequently, imposing varying utility demands upon the uncore. Compared to a state-

of-the-art previous work, the new approach can reduce the energy-delay product by 27%.

4. ENERGY SAVINGS WITHOUT PERFORMANCE LOSS THROUGH RESOURCE SHARING DRIVEN POWER MANAGEMENT

4.1 Resource Sharing Driven Per-Core DVFS

We consider a CMP architecture where (1) on-chip communication is implemented via a packet-switched network, i.e., a Network-on-Chip (NoC), (2) each processor core has an individual V/F domain, i.e., per-core DVFS [32], and (3) the V/F levels of the uncore (NoC + LLC) are fixed. Without loss of generality, we assume the core frequency is the allowed maximum for each supply voltage level in the given process technology.

DVFS can be implemented as a feedback control system, where a given metric is monitored, and the V/F level of a specific core is adjusted according to the monitored result. The execution is carried out at the end of periodic time intervals, called control intervals. At the end of a control interval, the controller collects monitored results, based on which it computes the V/F level for the next control interval.

In computer networks, one approach to congestion control is to regulate client packet injections according to perceived network congestion. We observe that the regulation procedure in computer networks can be viewed as similar to a DVFS control procedure, in that reducing the V/F level of a given core effectively throttles the packet injections from this core. This observation motivates us to design a DVFS policy[1] based on network congestion control.

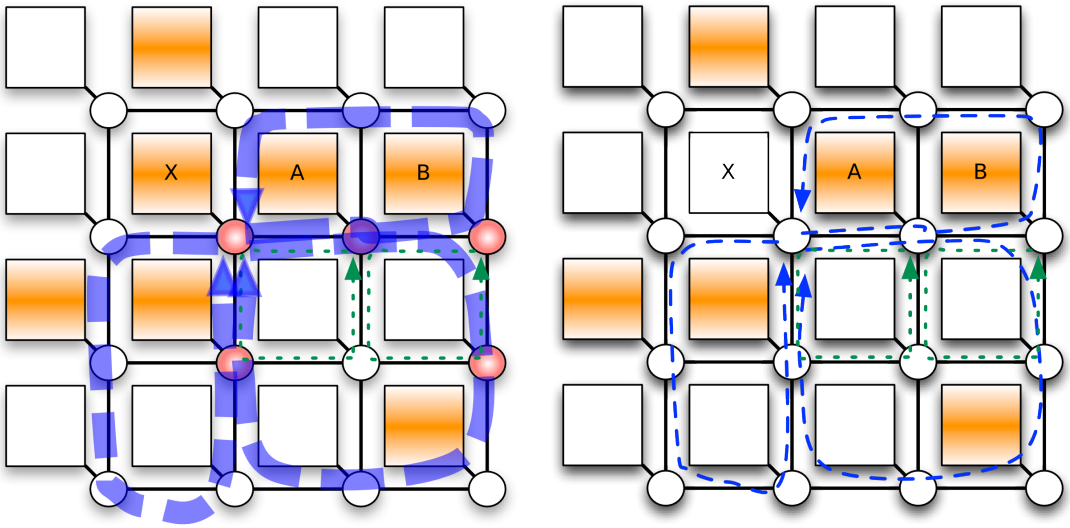
Consider the example in Fig. 4.1(a), where the large, dashed blue curves indicate the dominant round-trip request/reply packet routes from core X. As shown, packets

[1]Reprinted with permission from "Having Your Cake and Eating It Too: Energy Savings without Performance Loss through Resource Sharing Driven Power Management", by Jae-Yeon Won, Paul Gratz, Srinivas Shakkottai and Jiang Hu, 22-24 Jul. 2015, ACM/IEEE International Symposium on Low Power Electronics and Design(ISLPED), © 2015 IEEE/ACM.

from this node have a significant round trip time relative to other cores (e.g. core A and B) which have shorter communication loops (smaller dotted green curves). Decreasing voltage/frequency (V/F) level of core X reduces its injection rate and the shared resource contention, as shown in Fig. 4.1(b). Because core X is already experiencing long round-trip times, a moderate reduction of its V/F level has little effect on its own performance. The reduced congestion, however, can have a strong effect on cores which are being more productive with their use of the network and shared last level cache (LLC), (*e.g.*, core A and B), allowing them to make greater forward progress. This approach provides an appealing distinction from the conflicts and tradeoffs typically seen in the design process. The per-core DVFS throttles injections not only to uncore (on-chip interconnect + LLC), but also indirectly to DRAM. Therefore, it regulates the overall shared resources as a whole.

4.1.1 TCP Vegas in Network Control

We describe TCP Vegas [8], a well-known network congestion control protocol from computer networking, which provides the inspiration for our DVFS policy. The objective of TCP Vegas is to achieve a fair allocation of network capacity across different flows. It uses measured delays to adjust packet injection rates in such a manner that the throughput obtained by each flow is independent of the propagation delay between its source and destination. Further, it ensures that queuing delays are kept low (ideally, zero) and so congestion effects are mitigated. Towards this goal, TCP Vegas regulates the number of "outstanding" packets from a client. If a packet is injected into network but its source client has not received the corresponding acknowledgment (ACK) yet, this packet is outstanding. The set of outstanding packets for a client is referred as the "congestion window". TCP Vegas regulates window size of each client according to observed round-trip time (RTT), which is



(a) Without resource sharing driven DVFS.

(b) Reducing V/F of core X throttles its packet injection, alleviates NoC congestion and improves performance at cores A and B.

Figure 4.1: A 16-core CMP with on-chip network and per-core DVFS. Orange (white) cores are in a high (low) voltage/frequency state. Red circles are network routers experiencing congestion [63].

the time from a given packet's injection to the time when its corresponding ACK is received. The main idea is send a "marked" packet and to measure the time elapsed and number of ACKs received between the sending of the marked packet and the reception of its own ACK. We index this control interval by j, refer to the time elapsed as RTT_j, and the number of ACKs received as $\#ACK_j$. Then the metric used for the control is:

$$\Delta = \frac{|WINDOW|_j}{RTT_{ref}} - \frac{\#ACK_j}{RTT_j} \tag{4.1}$$

where $|WINDOW|_j$ is the congestion window size (in packets) at the beginning of interval j, and RTT_{ref} is the reference RTT (the smallest observed RTT thus far,

which represents the pure propagation delay between source and destination).

The first term here specifies a reference throughput (packets per second) that would arise if there were no queuing delay, while the second term represents the actual throughput observed in control interval j. A large value of Δ implies low throughput and (potential or actual) network congestion. TCP Vegas then reduces window size linearly (and equivalently, injection rate) in order to alleviate the congestion. Similarly, if Δ is small, the window size must be increased. Finally, if $\Delta < 0$, it means that RTT_{ref} must be updated to reflect the most recent RTT sample. In the interest of reducing oscillations, two thresholds α and β are maintained. The window size is changed only if Δ lies outside the interval $[\alpha, \beta]$.

4.1.2 Fairness though TCP Vegas

Our goal in the CMP setting is to ensure that none of the cores is starved for network access, while ensuring that RTT between issuing requests and receiving responses remains low. This goal ties in well with a concept known as *proportional fairness,* a metric often employed in the network resource allocation context [55]. Suppose that the set of flows is denoted by $\mathcal{R}$, with each $r \in \mathcal{R}$ representing a flow. Also, let a vector of capacity allocations to the flows be denoted by $X = [x_1, x_2, \cdots, x_r, \cdots]$. Consider two candidate allocations X and Y. Define the *proportional* increase in allocation for a flow r between the two allocations by $\frac{y_r - x_r}{x_r}$. Equivalently, note that if we multiply this quantity by 100, it would represent the percentage increase of allocation for flow r. Then an allocation X is said to be proportionally fair if for any other allocation Y, the following holds

$$\sum_{r \in \mathcal{R}} \frac{y_r - x_r}{x_r} \leq 0. \tag{4.2}$$

Thus, the total proportional (or percentage) increase over all flows would be less than the total proportional (or percentage) decrease for any other allocation Y, as compared to allocation X. Notice that this definition would not allow an increase in allocation to flows that already have a large allocation at the expense of those that have relatively lower allocations, hence ensuring a minimal allocation for all flows.

In TCP Vegas, the thresholds α_r and β_r are chosen to be inversely proportional to the flow's propagation delay, RTT_{ref}. It has been shown that with such a choice of thresholds, TCP Vegas achieves proportional fairness [40].Now, by multiplying both sides of (4.1) by RTT_{ref}, the control logic of Vegas can be interpreted as follows. The first term would simply be $|WINDOW|_i$, which is the number of outstanding packets, while the second term is the actual throughput times the propagation delay, which is the number of packets in flight through the links (the "pipe-size"). The difference between the two is then the number of packets buffered in the queues between the source and destination. Hence, the thresholds $\alpha \cdot RTT_{ref}$ and $\beta \cdot RTT_{ref}$ represent the target number of packets for each flow that are allowed to be buffered in the network. Choosing α_r and β_r, to be inversely proportional to RTT_{ref} would imply that all flows are allowed a similar number of buffered packets. Assuming uniform usage of links, this means that an appropriate choice of thresholds would limit the queuing delays seen by flows. Thus, TCP Vegas is a good choice to enable each flow to get a fair throughput, while ensuring small RTTs for all.

4.1.3 TCP-Vegas Based DVFS Control

Inspired by TCP Vegas, we design the following shared resource driven DVFS controller. Since packet injection rate (or window size) typically increases with core frequency, we replace the window size in Equation (4.1) by the relative core frequency $\frac{f}{f_{max}}$ where f_{max} is the frequency corresponding to the highest supply voltage level.

In a cache-coherent CMP, request packets into the NoC typically will receive a corresponding response packet. For example, when a core executes a load which misses in the local private caches, a request for the data will be sent into the NoC. This request will later be replied with the data from either the LLC or the DRAM. Hence, we can use the number of response packets received to estimate uncore and DRAM throughput. Thus, Equation (4.1) is modified as follows for our DVFS control.

$$\delta = \frac{f_j \ / \ f_{max}}{RTT_{min}} - \frac{\#RESPONSES_j}{\sum_{k \in INTERVAL_j} RTT_{j,k}} \tag{4.3}$$

where f_j is the core frequency at control interval j. Like Equation (4.1), the first term of Equation (4.3) represents the reference throughput at frequency f_j and the second term estimates the actual throughput from observations. We choose the reference RTT to be RTT_{min}, which is conceptually the RTT when there is no congestion in the uncore or DRAM. In practice, we first find an initial value of RTT_{min} through offline analysis. At runtime, the RTT_{min} is updated in the event a smaller RTT is encountered.

The V/F level of a core is tuned according to the δ observed at each interval. In order to avoid oscillations, two thresholds α and β ($\alpha < \beta$) are employed to determine if the V/F should be changed. If $\delta < \alpha$, actual throughput is close to expected throughput. This means round-trip time of the packets is low and the packets are unlikely to cause congestion in network. Thus, it increases the number of injections by incrementing the core V/F level to increase its performance. If $\delta > \beta$, actual throughput is low compared to the expected throughput and its round trip time is high. Then, the core V/F level is decremented to reduce the packet injections to network, because these packets are likely to congest the network and degrade the performance of other cores which show high throughput. If δ is between the two

threshold values, the current core V/F level is maintained without change.

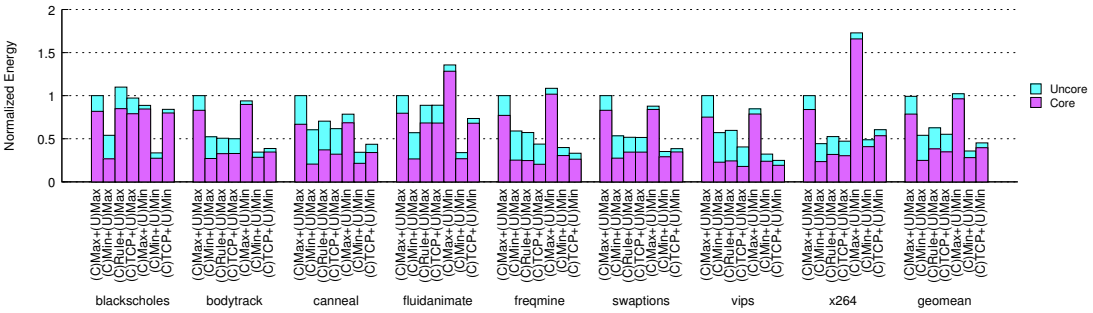
Our DVFS policy throttles packet injection when the observed RTT is high. There could be three reasons for high RTT: the on-chip interconnect is congested, shared cache banks are under high contention or the memory controllers are highly contended. For the former case, reducing injections can decrease queuing delay for packets of almost all cores. For the later two cases, the throttling reduces contention at the cache banks and/or memory controller. Essentially, the controller dynamically adjusts the frequency of each core such that they all experience the same target delay per request. Thus, a natural metric of fairness in this context is the standard deviation of average RTTs in a control interval among all cores.

Our DVFS policy is primarily based on injected packets. Thus, we also consider the case when few packets are injected. When the injection rate from a specific core to the uncore is very low, the core V/F level is instead determined proportionally by the number of retired instructions in the that core instead of the TCP-Vegas inspired policy, *i.e.,* when there are significant numbers of instructions retired, the core is run a max V/F, when an insignificant number of instructions are retired, the core is run at min V/F.

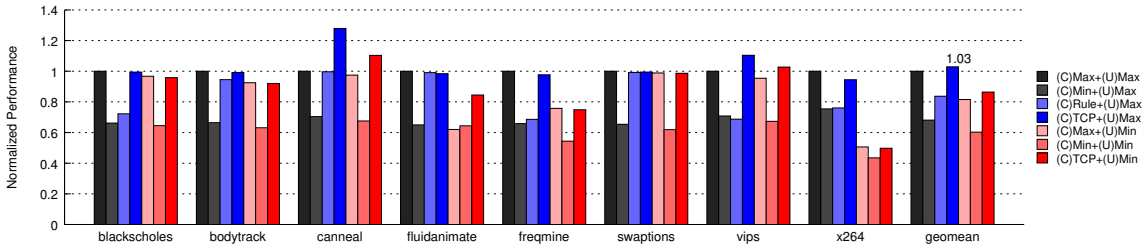
4.2 Experimental Results

4.2.1 Experiment Setup

The experimental baseline platform is a 16-core CMP with a 2-level cache hierarchy, split L1i and L1d private caches, a shared L2 last-level cache (LLC) and four DRAM memory controllers. Cache coherence is maintained via a MESI directory cache coherence protocol. The cores are interconnected via a 4×4 2D mesh NoC topology. Table 4.1 summarizes the baseline CMP setup. Each core and the uncore have their own separated voltage/frequency domain. V/F levels for the cores range



(a) Normalized energy



(b) Normalized performance

Figure 4.2: Full-system simulation results of single application PARSEC benchmark suite [63].

from 2.66GHz - 1.6GHz, following the states found in the Intel Core i7 processor [59]. V/F level for the uncore is fixed to 1GHz as a maximum frequency or 250MHz as a minimum frequency.

All experiments are conducted using the gem5 simulator system [4] with a 4-wide O3 core CPU model, X86 ISA, *Ruby* memory model and *Garnet* NoC model. We used CACTI 6.5 [48], ORION 2.0 [30] and McPAT 1.0 [38] to model the energy of the NoC, cache and cores, respectively. The energy consumption model includes leakage and dynamic energy based on 32nm process technology. The application testcases are taken from the PARSEC shared-memory, multi-processor, benchmark suite [3]. We assume that the DVFS control is implemented as firmware running on the Power Control Unit, a microcontroller dedicated to CMP power management which is often

Table 4.1: Configuration and parameters of the experiment platform [63].

Parameter	Configuration
Core CPU	4-wide out of order processor
Pipeline	7-stage pipeline
ROB size	192-entry reorder buffer
# of cores	16
Core V/F	9 levels, voltage: 1.55V - 0.8V frequency: 2.66GHz - 1.6GHz
L1 data cache	2-way 256KB, 2 core cycle latency
Directory cache	MESI, 4 core cycle latency
L2 cache (LLC)	16-way, 2MB/bank, 32MB/total 10 uncore cycle bank latency
NoC	4x4 2D mesh, X-Y DOR 4-flits per VC
Uncore V/F	1GHz(Max) or 250MHz(Min)
Control interval	50 μs
V/F transition per step	100 ns
DVFS threshold	$\alpha = 0.18$, $\beta = 0.22$
Technology	32nm

seen in modern Intel CMP designs [34]. The data collection from different cores is realized by the techiques of [9]. As in the work by Chen et al. [9], we assume a control interval to be $50\mu s$ and each step V/F level change takes $100ns$ ($100ns$ per step is sufficient assuming on-die regulation [18]). Experiments for control intervals of $\sim 1ms$, which is close to the scheduling interval of the OS, have also been conducted and exhibit similar results.

In the figures found in this section, the following V/F settings and policies are examined:

- (C)Max+(U)Max: All cores and the uncore constantly run at the maximum V/F level. This result is the baseline, to which the other results are normalized.
- (C)Min+(U)Max: All cores run at their minimum V/F while the uncore runs at its maximum V/F level.

- (C)Rule+(U)Max: All cores run DVFS using the method proposed by Herbert and Marculescu [24] while the uncore runs at its maximum V/F level.

- (C)TCP+(U)Max: All cores run our resource sharing driven DVFS (Section 4.1.3) while the uncore runs at its maximum V/F level.

- (C)Max+(U)Min: All cores run at their maximum V/F while the uncore runs at its minimum V/F level.

- (C)Min+(U)Min: All cores and the uncore constantly run at the minimum V/F level. These results provide an approximated lower bound on energy dissipation.

- (C)TCP+(U)Min: All cores run our resource sharing driven DVFS while the uncore runs at its minimum V/F level.

Note that the (U)Min results are included to show the impact of a system in which more contention is found in the memory system, providing more potential performance upside to (C)TCP's resource sharing based power management.

We use the eight PARSEC benchmarks currently supported in our infrastructure, *blackscholes*, *bodytrack*, *canneal*, *fluidanimate*, *freqmine*, *swaptions*, *vips*, *x264*. In the experiments, all benchmarks are executed to the end of simulation, but only the Region Of Interest (ROI) is evaluated. We examine both single application testcases, where the 16 threads of the application are run on all 16 cores of the CMP, as well as multi-application runs, where the CMP is partitioned into two sets of 8 cores and each application takes up one partition.

Quantitative analysis of the multi-application cases is not as straightforward as single application benchmarks because the applications have different runtimes. We use the following simulation methodology: the shorter application is re-executed

multiple times while a long application is running. By doing so, we ensure there are always two applications running at any time. To estimate the overall performance, we extract the runtime of the first ROI of each application and calculate the geometric mean of their normalized performance. The normalization is with respect to simulation with the static maximum V/F level. The overall energy is estimated as the total core and uncore energy over the ROI of the long application.

4.2.2 PARSEC Application Cases and Analysis

Fig. 4.2 shows single application, full-system experimental results for each PARSEC benchmark, with the rightmost clusters being the geometric mean results. Fig. 4.2(a) shows normalized energy with respect to simulations where both cores and the uncore operate at the maximum V/F level. In the figure we see the proposed resource sharing driven power management provides an average energy savings of 43% versus a maximum frequency simulation. Some benchmarks, such as *freqmine* and *vips*, actually show more energy saving than the minimum frequency simulation, this is because of the large increase in run-time caused at the minimum frequency. Overall, the energy saving is close to that shown for the minimum frequency simulations, except in the case of *blackscholes*, *canneal* and *fluidanimate*. *Blackscholes* is a highly balanced, CPU-bound benchmark with very low shared resource utilization. Thus, for this benchmark the best policy is to maintain a high frequency to avoid system performance degradation.

Fig. 4.2(b) shows the performance of each benchmark, normalized against a maximum frequency simulation. Here, normalized performance represents speed-up. In the figure, no benchmark shows more than a 5% performance loss. Some benchmarks actually show a performance gain. In particular, *canneal* shows a 28% performance improvement. This benchmark is memory-bound and shows particularly high con-

tention within the NoC, thus greater resource allocation fairness actually improves performance *while* saving energy. It is important to note that even in nominally CPU-bound applications such as *bodytrack* and *swaptions*, we see significant energy savings with no appreciable change in performance. In these applications it is often the case that not all cores benefit from DVFS relative to other threads in that workload. This is due to several effects, including program phase behavior - leading to small memory-bound phases within otherwise CPU-bound applications, workload imbalances - leading to some threads waiting on others to finish, and the location within the network the thread is running - leading to differential resource utilization efficiency. Overall, our resource sharing driven DVFS shows a 2.9% average performance gain, while showing a 43% energy saving. Our DVFS policy also shows 19.2% better performance with similar energy saving versus a prior work, rule-based approach [24].

The rightmost three bars in each cluster of Fig. 4.2 show results for when the uncore is statically set to the minimum V/F level, "(U)Min". As a result, the uncore is more congested. Comparing the rightmost three bars we see that the performance gain from our approach doubles to 6% versus the core max, uncore min V/F case, (C)Max+(U)Min. These results highlight the effect of improved fairness in resource allocation when this resource (the uncore) is more contended. Moreover, the energy saving in this case also increases to 55% in this case.

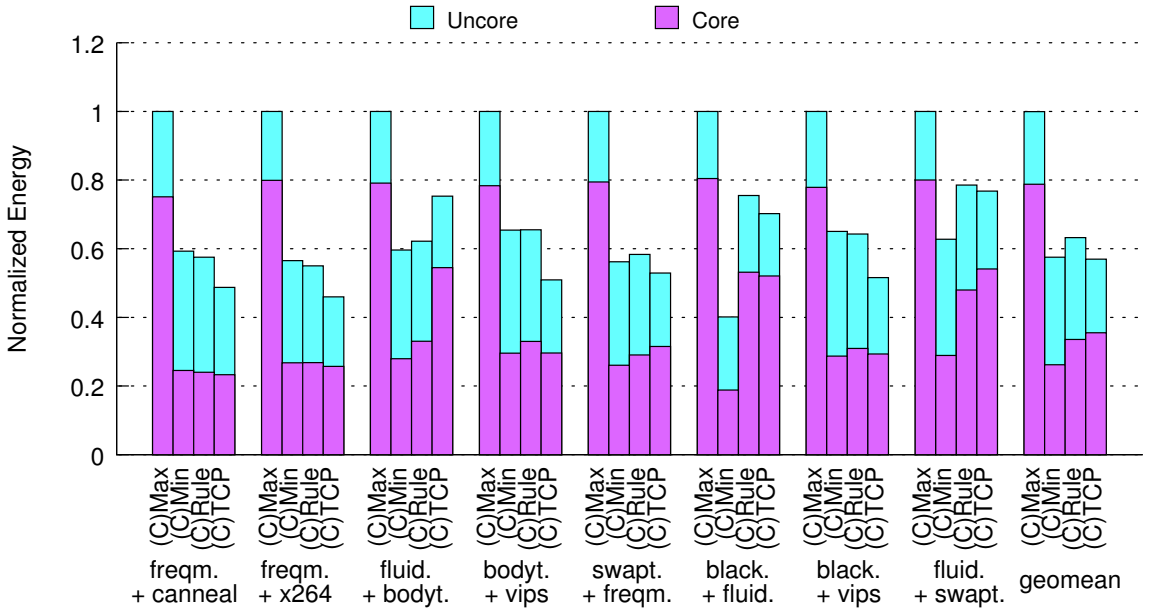
We also examined the improvement in fairness according to the RTT-per-core delay spread metric discussed in Section 4.1.2, and summarize the results in Table 4.2. Columns 2 and 3 are the results from simulations with the uncore V/F at its maximum level. Our approach considerably reduces both the mean and standard deviation (σ) of RTTs among cores, indicating that the resource is more fairly shared among the cores. Columns 4 and 5 show results for when the uncore V/F is set to the

Table 4.2: Standard deviation (σ) and mean of RTTs from our TCP-DVFS, normalized with respect to results from cores at the max V/F level [63].

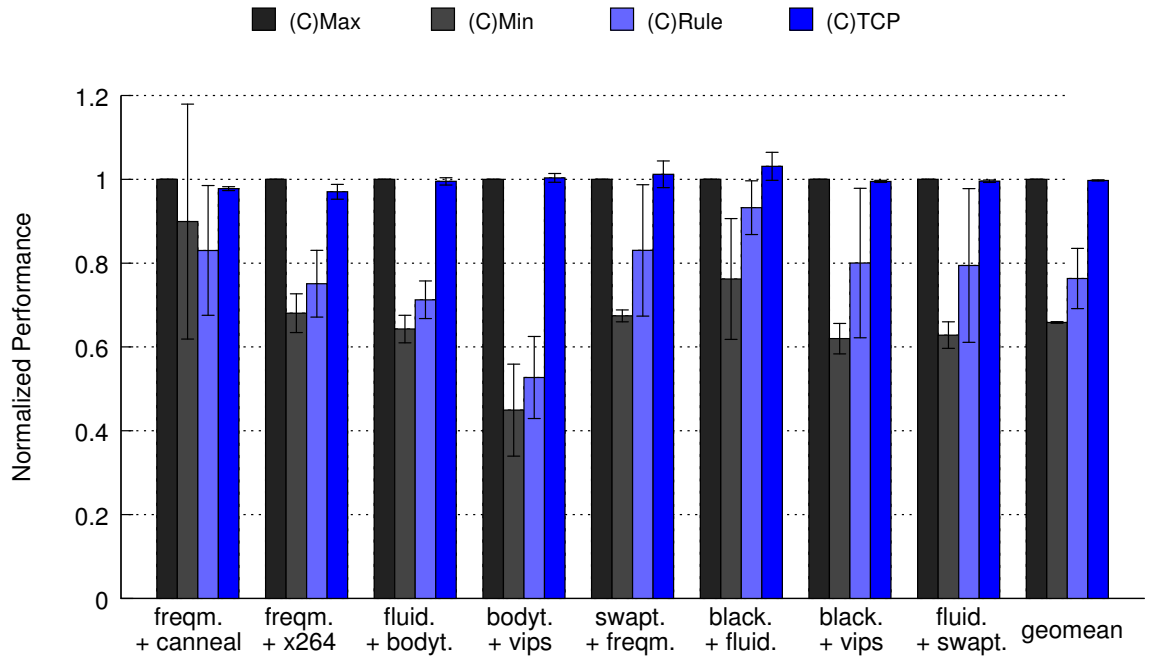
	Uncore max V/F		Uncore min V/F	
	σ	Mean	σ	Mean
blackscholes	0.47	0.65	0.24	0.42
bodytrack	0.30	0.56	0.35	0.45
canneal	0.14	0.52	0.33	0.49
fluidanimate	0.44	0.55	0.35	0.43
freqmine	0.32	0.49	0.31	0.42
swaptions	0.28	0.53	0.17	0.41
vips	0.97	0.52	0.23	0.34
x264	0.51	0.47	0.35	0.37
gmean	0.38	0.54	0.28	0.41

minimum level, including the baseline. In this case, the uncore is presumably more congested. Here, the RTT mean and σ reduction by our approach is even greater. The standard deviation is reduced 72% on average. This greater σ reduction coincides with greater performance gain as discussed previously. These results generally indicate that our approach improves CMP performance via improving fairness in shared resource utilization.

Fig. 4.3 shows overall full-system simulation results for randomly chosen sets of two simultaneous PARSEC benchmarks. In each case, eight cores are allocated to each application. Fig. 4.3(a) shows the normalized energy consumption for each testcase. In Fig. 4.3(b), normalized performance is shown and each line bar above the bar chart represents the performance of each single benchmark, separately. The labels of two benchmarks are sorted by higher performance shown in (C)TCP. For example, in a combination of *swaptions* + *freqmine*, *swaptions* shows 4.4% performance improvement while 1.9% performance degradation is seen in *freqmine*. We found that, in this configuration, the benchmarks showed an overall average per-



(a) Normalized energy

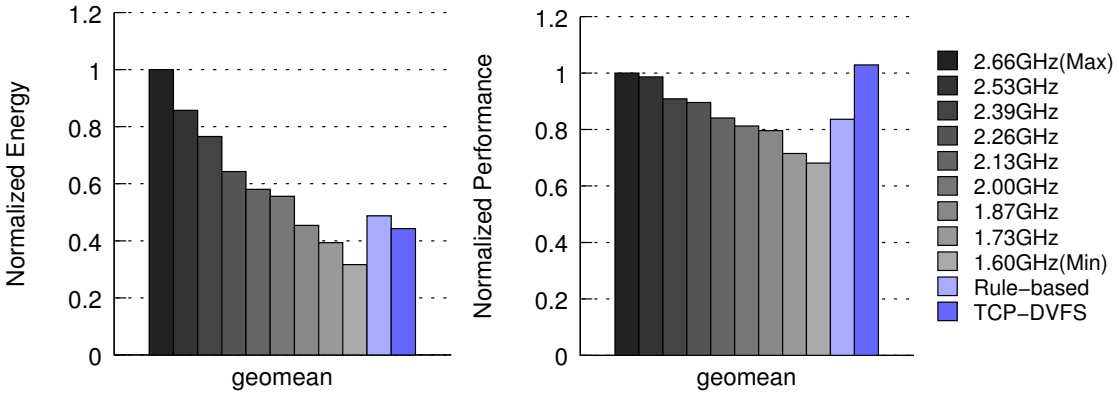


(b) Normalized performance

Figure 4.3: Multi-applications experiment results with uncore at the maximum V/F [63].

formance degradation of 0.3% while also saving $\sim$ 56% in core energy consumption and $\sim$ 42% in total energy consumption through our core-only TCP-DVFS policy. Compared with *(C)Rule* [24], our approach leads to much less disparity between the application runtimes of each combination.

To gain intuition on the benefits of dynamic V/F scaling in our technique we also examine the simulations for static V/F levels across all benchmarks. Figure 4.4 shows the geometric mean across all eight PARSEC benchmarks for the static V/F levels, together with the rule-based per-core DVFS [24] and our QoS per-core DVFS (with uncore at the maximum frequency). In the figure, the energy-performance tradeoff among different static V/F levels is obvious. Our result achieves the performance of a static 2.66GHz core frequency, while consumes energy of only 1.87GHz static frequency. Moreover, our performance is significantly better than the rule-based previous work.



(a) Normalized energy (b) Normalized performance

Figure 4.4: Comparison with static core V/F levels in single-application cases [63].

4.3 Conclusion

We propose a resource sharing-driven DVFS technique for CMP designs, whose performance is largely affected by resource shared among cores. In this technique, the DVFS serves as client regulator so as to manage shared resource, especially fairness, of the shared resources. As such, significant energy saving is obtained with negligible performance loss and sometimes performance gain. This technique is bolstered by network control theory yet is simple to implement and has low overhead. Full system simulation results show that this technique can reduce energy dissipation by over 40% with more than 2.0% performance gain compared to baseline and 19.2% performance increment with similar energy savings compared to a prior work [24].

5. COORDINATED DYNAMIC VOLTAGE FREQUENCY SCALING TECHNIQUE FOR MAIN MEMORY

5.1 Coordinated Memory DVFS with CMP Power Management

First, our proposed memory DVFS technique which is robust to randomized resource utilization is described in section 5.1.1. Also, in section 5.1.2 we propose a coordinated power management technique through dynamic voltage and frequency scaling (DVFS) with CMP (core + uncore) power management.

5.1.1 Shared Resource Utilization Aware Memory DVFS

Generally, memory DVFS is achieved to save energy by decreasing voltage(V) and frequency(F) when utilization of memory is low. In other word, lowering V/F level of memory derives energy saving and less performance degradation when there is few packets to access memory. As such, high utilization of memory requires high V/F level to avoid lots of performance degradation. In this view, low V/F level of memory is preferred for CPU-bounded applications because lowering V/F level of memory does not degradate total performance too much. For memory-bounded applications, high V/F level of memory is preferred to avoid performance degradation even though less energy saving in memory. We however found that it is not true that the highest V/F level of memory guarantees best performance among all V/F levels. A conventional work [15] used a core utilization factor and a memory utilization factor without knowledge of uncore utilization. Uncore (LLC + interconnection) utilization here means how many packets go into last level cache and interconnected blocks. High uncore utilization can possibly generate even much more congestion by the packets responded from memory side. Higher V/F level of memory responses(injects) more packets into the interconnected block compared to lower V/F level of memory. Thus,

the congestion by higher V/F level of memory degradates performance more than one by lower V/F level of memory.

Especially, some randomized applications, i.e. simulated annealing, with higher V/F level of memory generate more packets and congestion in uncore. In the randomized application of chip multiprocessor, multiple threads access data from the shared resource simultaneously and proceed their own computations. If more than one thread access same data, recovery process to solve the conflict can be required after each threads finish their own computations. For instance, simulated annealing algorithm starts with randomly distributed data. The randomly distributed data has more possibility to swap two data because the initial data is not aligned. This phase possibly generates more conflicts among all threads and more recovery processes. In this case, lower V/F level of memory operates as a serializing packets responder(injector) to uncore. The serialized packets generates less conflict among threads and have less recovery process. The less recovery process would contribute less number of accessing cache and memory, cache coherence traffic, and so higher performance.

Higher performance by lower V/F level of memory is shown in Figure 5.1. The graphs shows the number of accumulated instructions executed at each control intervals in maximum and minimum memory frequency simulations. For the first 1.0×10^6 instructions, minimum frequency simulation takes 54 control intervals while maximum frequency simulation takes 62 control intervals. It means that minimum frequency of memory shows faster run-time (higher performance) compared to maximum frequency of memory. The total number of memory access in minimum frequency simulation is decreased 22% compared to one in maximum frequency simulation. This means that the minimum frequency of memory generates less last level cache misses. The less cache misses could contribute faster run-time. More conflicts

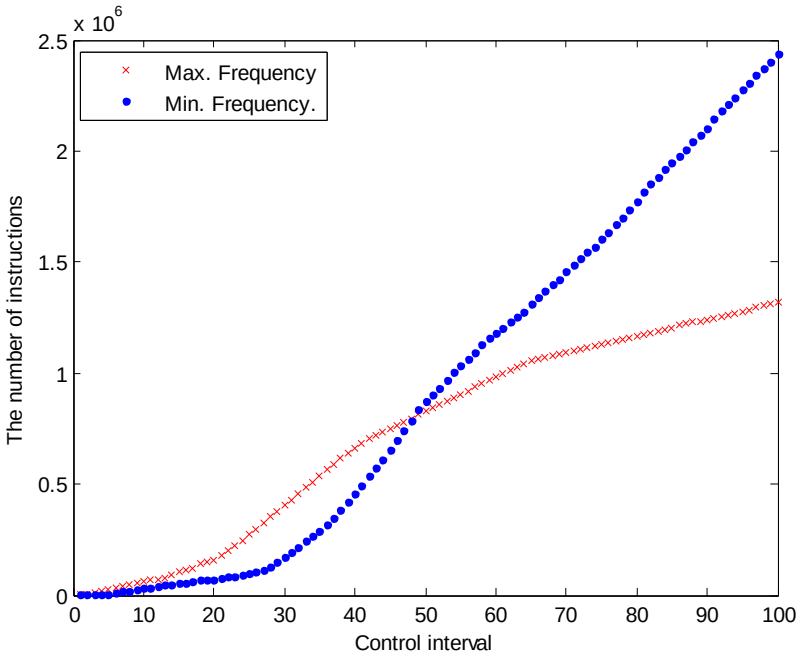
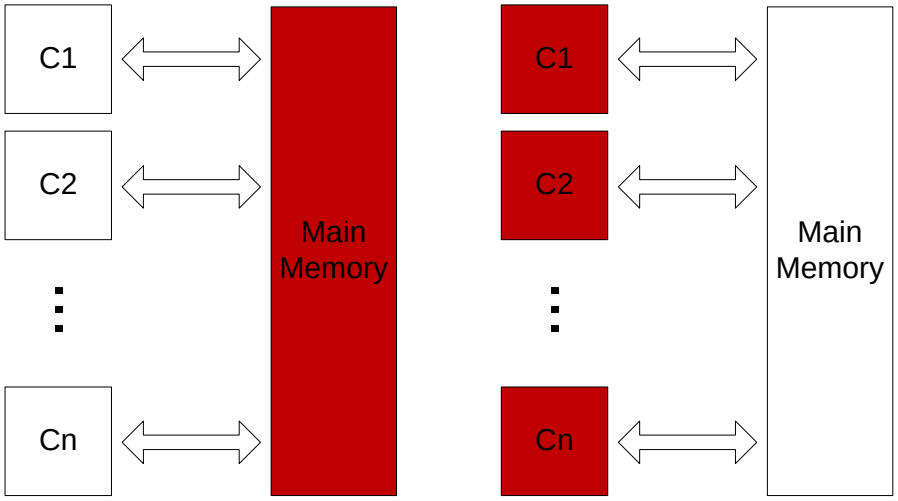


Figure 5.1: The number of instructions executed in *canneal*

generated by high frequency of memory have more cache coherence traffic as well as memory accesses. High frequency has high possibility to access data in the same address, and we term it a conflict here. Accessing same data from more than a thread occurs more cache coherence traffic. Thus, high frequency of memory generates more cache coherence traffic as well as more memory accesses. Through the above two observations, memory V/F level affects utilization of the shared resources such as LLC and interconnect components. Thus, uncore utilization factor is useful for memory power management.

In this section, we propose a DVFS technique through awareness of shared resource congestion for memory power management. Figure 5.2 shows memory power management technique of conventional approaches. In Fig. 5.2 and Fig. 5.3, red col-



(a) Memory intensive behavior (b) Core intensive behavior

Figure 5.2: Conventional main memory DVFS.

ored components represent busy status with lots of work load executed while white colored components has less work load executed. Figure 5.2(a) represents that work load of memory intensive application exists in main memory. In this case, power management policy of the conventional approaches hold high frequency for main memory to avoid lots of performance degradation. In Fig. 5.2(b), work load of core intensive application is in cores and less work load exists in main memory. Conventional power management policy holds low frequency to save energy with less performance degradation.

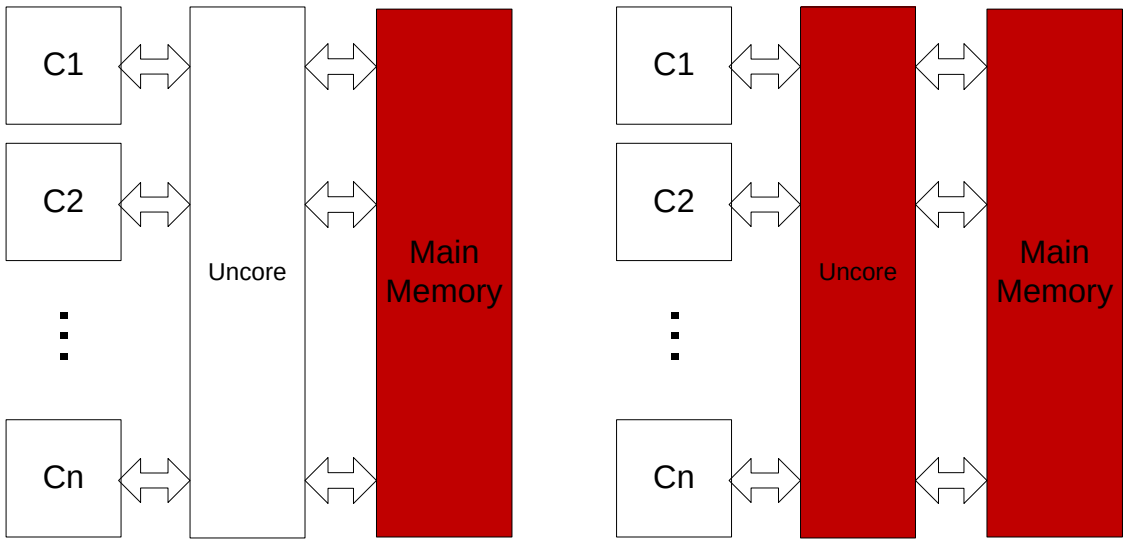
Our proposed shared resource aware power management policy is shown in Fig. 5.3. In core intensive application, our technique works same way to the conventional approaches. In our approach, low frequency is preferred to save energy with less performance degradation because there is less work load exists in main memory. Thus, lowering frequency of main memory does not degradate system performance. for the memory intensive application, we do not consider only utilization of main memory,

but also the shared resource. The packets injected from cores go through the shared resource and arrive to main memory. Also, responding packets from main memory go to cores through the shared resource. Figure 5.3(a) and Fig. 5.3(b) show memory intensive work load distribution. In Fig. 5.3(a), there is less packets in the shared resource and most of work load is in main memory. In this case, our policy holds high frequency of main memory to avoid performance degradation. However, our power management technique holds low frequency of main memory when there is much work load in the shared resource. In this case, high frequency of memory sends fast responses to cores, but cannot arrive at cores due to congestion in the shared resource. In other words, the fast frequency is less effective to avoid system performance degradation. Also, lowering frequency of memory does not degradate system performance much compared to faster frequency of memory. In addition, fast responses from memory possibly make more congestion in the shared resource and degradate even more performance degradation. Thus, lowering frequency of main memory contributes saving energy without less performance degradation and performance increment by solving congestion in the shared resource.

We used three utilization metrics to consider the policies described in Fig. 5.3. One is the number of instructions, I_{CORE} as core utilization metric. For utilization of last level cache (LLC) and main memory, we used the number of LLC accesses and main memory, I_{UNCORE} and I_{MEM}. With the utilization metric, we define global and local memory utilization rate as shown in Eq. 5.1 and Eg. 5.2.

$$Mem_{Global} = \frac{I_{MEM}}{I_{CORE}} \tag{5.1}$$

$$Mem_{Local} = \frac{I_{MEM}}{I_{LLC}} \tag{5.2}$$



(a) Memory intensive behavior without congestion of the shared resource

(b) Memory intensive behavior with congestion of the shared resource

Figure 5.3: Our shared resource congestion aware memory DVFS.

Mem_{Global} and Mem_{Local} represent relative memory utilization to cores and LLC, respectively. When Mem_{Global} is low, it means that less work load is in memory, and our policy holds low frequency to save energy. High Mem_{Global} represents lots of memory accesses as shown in Fig. 5.3. In this case, we consider Mem_{Local} as well as Mem_{Global}. In our technique, high frequency of memory is selected when Mem_{Local} is also high because high Mem_{Local} represents memory work load is relatively higher than the one in the shared resource. When Mem_{Local} is low, our policy holds low frequency of memory because shared resource is bottleneck of total system performance. Lowering frequency of memory in this case does not degradate system performance while saving more energy.

The proposed main memory DVFS policy adopts simple approach to reduce implementation cost. If $Mem_{Global} < \underline{\Phi_{Global}}$, the main memory V/F level is decreased by one step. If $Mem_{Global} > \overline{\Phi_{Global}}$, the main memory V/F level depends

on Mem_{Local}. If $Mem_{Local} < \underline{\Phi_{Local}}$ ($Mem_{Local} > \underline{\Phi_{Local}}$), the main memory V/F level is decreased (increased) by one step. Otherwise, the main memory V/F stays unchanged. Four thresholds $\overline{\Phi_{Global}}$ $\underline{\Phi_{Global}}$, $\overline{\Phi_{Local}}$ and $\underline{\Phi_{Local}}$ are employed here.

5.1.2 Coordinated DVFS policy

Many work on power management for certain components have been explored, there could be the coordination issue on combining them. It means that an efficient power management technique of a component could be not efficient when it operates with power management of other components. For instance, a core DVFS technique and a memory DVFS technique select high frequency when both components have high work load. On the next control interval, in case of less work load on both of the cores and memory, the both of the DVFS technique select lower frequencies for both. However, lowering frequency of the cores reduces work load on the memory as well. Thus, lowering frequency of memory without coordination may not be necessary and cause performance degradation. As we mentioned in section 5.1.1, the utilization of memory memory is highly related to the uncore(shared resources)'s status. Therefore, we also develop an coordinated uncore DVFS technique in section 5.1.2.1. And then, we explain our proposed coordinated main memory DVFS policy in section 5.1.2.2.

5.1.2.1 Coordinated uncore DVFS policy

In this section, we develop an uncore DVFS scheme that is applied in conjunction with the QoS-driven per-core DVFS. As with several recent works [39, 10, 29, 9, 62], we consider the case where the on-chip interconnect is implemented with a Network-on-Chip (NoC) and the entire uncore shares a single voltage/frequency domain. NoCs have been widely recognized as a scalable approach to cope with increasingly large demand for on-chip communication bandwidth. While we argue that placing the

uncore in one shared V/F domain has a key advantage – it avoids performance and power overhead of domain-crossing, our technique is conceptually applicable in other forms of on-chip interconnect, as well as divided uncore V/F domains.

As uncore DVFS policy has been investigated previously [39, 10, 29, 9, 62], a natural question is whether we can adopt the existing approaches and directly combine them with our QoS-driven per-core DVFS. In the case of Juan *et al.*'s DVFS scheme [29], their uncore DVFS policy is tightly coupled with their own core DVFS policy and is difficult to transplant to our system. Other uncore policies can be viewed as "stand-alone" and assume that core V/F levels are fixed [39, 10, 9, 62]. In Liang et al.'s approach, uncore V/F is tuned based on the packet injection rate into the network [39]. There is a risk in directly combining such injection-based techniques with our core DVFS. When core V/F levels change, the network injection is not always a true reflection of real traffic demand. A low injection rate due to the core V/F scaling may mislead to uncore V/F decrease. Then, the consequently increased RTTs may further decrease core V/F levels. Such false feedback can result in a downward spiral of both core and uncore V/F even if there is large work/traffic load and cause large performance loss. Other recent works [9, 62], adjust uncore V/F according to uncore latency or the RTT excluding DRAM latency. A naïve application of these approaches with our core DVFS is prone to oscillation. For example, when the network is congested, uncore increases its frequency while some cores may simultaneously decrease their frequencies. This may result in over correction and underload of the network. The underload then demands uncore frequency decrease, core frequency increase, and so on.

We propose a new uncore DVFS technique in coordination with our core DVFS. It is a rule-based approach guided by *predicted latency effect*, and can largely overcome the aforementioned problems in directly using prior approaches [9, 62]. For core i at

the end of control interval j, the predicted latency effect for interval $j+1$ is defined by:

$$\phi_{i,j+1} = \lambda_{i,j} \cdot \frac{\#INJECTIONS_{i,j}}{\#INSTRUCTIONS_{i,j}} \cdot \frac{f_{i,j+1}}{f_{i,j}} \tag{5.3}$$

where $\lambda_{i,j}$ is the average uncore latency and $f_{i,j}$ is the operating frequency of core i in interval j. The ratio of injection count and instruction count serves as a criticality factor for the uncore latency. Note that $f_{i,j+1}$ is computed by core i at the end of control interval j and is to be executed for interval $j+1$. As such, the ratio $\frac{f_{i,j+1}}{f_{i,j}}$ predicts the change of injection rate in the next interval. The average predicted latency effect among all of N cores is

$$\Phi_{j+1} = \frac{\sum_{i=1}^{N} \phi_{i,j+1}}{N}. \tag{5.4}$$

The proposed uncore DVFS policy is very simple. If $\Phi_{j+1} > \overline{\Phi}$ ($\Phi_{j+1} < \underline{\Phi}$), the uncore V/F level is increased (decreased) by one step. Otherwise, the uncore V/F stays unchanged. Two thresholds $\overline{\Phi}$ and $\underline{\Phi}$ are employed here.

A key difference from prior efforts [9, 62] is that our observation variable Φ_{j+1} accounts for the injection rate change due to the future core V/F state in the next interval. This anticipation reduces the likelihood of oscillation. For example, when network congestion appears, some cores may lower their V/F levels. Then, the ratio $\frac{f_{i,j+1}}{f_{i,j}}$ would discount the predicted latency effect. If many cores lower their frequencies, the overall Φ_{j+1} may increase very little despite the uncore latency increase. As such, the uncore may keep its frequency unchanged and therefore reduce the chance of oscillation. The two-level threshold $\overline{\Phi}$ and $\underline{\Phi}$ is to further reduce the risk of oscillations. In this regard, the rule-based policy is more flexible than prior approaches [9, 62] on coping with the oscillations.

5.1.2.2 Coordinated main memory DVFS policy

As we mentioned in section 5.1.2, coordination among all separates power management policy is very important to avoid over performance degradation along with less power saving. A prior work [15] proposed a power management policy with the coordination between cores and memory. Also, they assumed in-order processor model to increase accuracy of the estimation of their power management policy. In this section, we propose a main memory DVFS policy which is coordinated with our uncore DVFS scheme and TCP-based DVFS scheme. Our technique can be applied to more complicated processor model because it is based on the measurements of the system counter.

As our coordinated uncore DVFS technique, our main memory DVFS technique also considers the effect of cores' DVFS and uncore DVFS in the same control interval. In our main memory DVFS policy, we utilize I_{CORE}, I_{LLC} and I_{MEM} as shown in Eq. 5.1 and Eq. 5.2. The numbers of instructions are updated by the cores' DVFS and uncore DVFS results. For core i at the end of control interval j, the predicted number of instructions and LLC access for interval $j+1$ is defined by:

$$I_{CORE,j+1} = \sum_{i=1}^{N} \#INSTRUCTIONS_{i,j} \cdot \frac{f_{i,j+1}}{f_{i,j}} \tag{5.5}$$

$$I_{LLC,j+1} = \sum_{i=1}^{N} \#LLCACCESSES_{i,j} \cdot \frac{f_{i,j+1}}{f_{i,j}} \tag{5.6}$$

where $f_{i,j}$ is the operating frequency of core i in interval j and N is the number of all cores. Note that $f_{i,j+1}$ is computed by core i at the end of control interval j and is to be executed for interval $j+1$. As such, the ratio $\frac{f_{i,j+1}}{f_{i,j}}$ predicts the change of instruction and injection rate to LLC in the next interval.

I_{MEM} is affected by the results of the cores' DVFS and the uncore DVFS as shown in:

$$I_{MEM,j+1} = \left(\sum_{i=1}^{N} \#MEMACCESSES_{i,j} \cdot \frac{f_{i,j+1}}{f_{i,j}} \right) \cdot \frac{fu_{j+1}}{fu_j} \tag{5.7}$$

where fu_j is the operating frequency of uncore in interval j. In our proposed coordinated main memory DVFS technique, $I_{CORE,j+1}$, $I_{LLC,j+1}$ and $I_{MEM,j+1}$ are used to replace I_{CORE}, I_{LLC} and I_{MEM} of Eq. 5.1 and Eq. 5.2, respectively.

5.2 Experimental Results

5.2.1 Experiment Setup

The experimental baseline platform is a 16-core CMP with a 2-level cache hierarchy, split L1i and L1d private caches, a shared L2 last-level cache (LLC), four DRAM memory controllers and memory subsystem. We adopt a DDR-style memory subsystem of *Ruby* memory model of Gem5 [4]. Cache coherence is maintained via a MESI directory cache coherence protocol. The cores are interconnected via a 4×4 2D mesh NoC topology. Table 5.1 summarizes the baseline CMP setup and memory model. Each core, the uncore and the DRAM memory have their own separated voltage/frequency domain. V/F levels for the cores range from 2.66GHz - 1.6GHz, following the states found in the Intel Core i7 processor [59]. V/F levels for the uncore range from 1GHz - 250MHz to facilitate comparison with prior work in uncore power management [9]. The DRAM V/F levels ranges from 800MHz - 200MHz which is a standard range of comercial DDR3 models. All parameters along with all V/F levels and energy model of DDR3 follow Micron's power calculator [44].

All experiments are conducted using the gem5 simulator system [4] with a 4-wide O3 core CPU model, X86 ISA, 7-stage pipeline, 192-entry reorder buffer, *Ruby* memory model and *Garnet* NoC model. We used CACTI 6.5 [48], ORION 2.0 [30],

Table 5.1: Configuration and parameters of the experiment platform.

Parameter	Configuration
Core CPU	4-wide out of order processor
Pipeline	7-stage pipeline
ROB size	192-entry reorder buffer
# of cores	16
Core V/F	9 levels, voltage: 1.55V - 0.8V frequency: 2.66GHz - 1.6GHz
L1 data cache	2-way 256KB, 2 core cycle latency
Directory cache	MESI, 4 core cycle latency
L2 cache (LLC)	16-way, 2MB/bank, 32MB/total 10 uncore cycle bank latency
NoC	4x4 2D mesh, X-Y DOR 4-flits per VC
Uncore V/F	10 levels, voltage: 1.35V - 0.8V frequency: 1GHz - 250MHz
DRAM Memory V/F	10 levels, voltage: 1.575V - 1.475V frequency: 800MHz - 200MHz
Memory configuration	4 DDR3 channeals, 8 2GB DIMMs
Control interval	50 μs
V/F transition	100 ns
DVFS threshold	$\overline{\Phi_{Global}} = 0.0015$, $\underline{\Phi_{Global}} = 0.0005$ $\overline{\Phi_{Local}} = 0.3$, $\underline{\Phi_{Local}} = 0.01$
Technology	32nm

McPAT 1.0 [38] and Micron's power calculator [44] to model the energy of the NoC, cache, cores and DDR3 memory, respectively. The energy consumption model includes leakage and dynamic energy based on 32nm process technology.

Our coordinated DVFS techniques for uncore and memory are compared against a baseline, no-DVFS design, the full set of static V/F levels, a conventional memory DVFS approach [15] and a technique of co-scaling V/F levels of cores and memory proposed by Deng and et al. [13].

In the figures found in this section, the following V/F settings and policies of our coordinated DVFS (uncore and memory) are examined:

- (C)Max+(U)Max+(M)Max: All cores, the uncore and the memory constantly run at the maximum voltage/frequency. This result is the baseline, to which the other results are normalized.

- (C)Min+(U)Min+(M)Min: All cores, the uncore and the memory constantly run at the minimum V/F level. These results provide an approximated lower bound on energy dissipation.

- (M)Memscale: Memory runs DVFS using the method proposed by Deng [15] while all cores and the uncore run at its maximum V/F level.

- (M)Global: Memory runs accoring to Eq. 5.1 in Section 5.1.1 while all cores and the uncore run at its maximum V/F level.

- (M)Local: Memory runs accoring to Eq. 5.2 in Section 5.1.1 while all cores and the uncore run at its maximum V/F level.

- (M)Glo+Loc: Memory runs our proposed shared resource aware DVFS (Section 5.1.1) while all cores and the uncore run at its maximum V/F level.

- (C/M)Coscale: All cores and memory run a conventional coordinated DVFS [13] while the uncore runs at its maximum V/F level.

- (C/M)Coord.DVFS: All cores and memory run our proposed coordinated DVFS and the uncore runs at its maximum V/F level. This setting is conducted to compare with the conventional work [13].

- (C/U/M)Coord.DVFS: All cores, uncore and memory run our proposed coordinated DVFS.

The application testcases are taken from the PARSEC shared-memory, multiprocessor, benchmark suite [3]. We use the eight PARSEC benchmarks currently

supported in our infrastructure, *blackscholes*, *bodytrack*, *canneal*, *fluidanimate*, *freqmine*, *swaptions*, *vips*, *x264*. In the experiments, all benchmarks are executed to the end of simulation, but only the Region Of Interest (ROI) is evaluated.

5.2.2 PARSEC Application Cases and Analysis of Coordinated DVFS

Figure 5.4 shows energy distribution of multi-thread applications, *PARSEC* [3], in CMP (cores and uncore) and main memory. CMP including cores and uncore takes about 68% of entire energy consumption while main memory takes 32% in our environment setup. This graph proves that cores and memory takes large portion of entire energy and cores and memory power management is very essential to save entire energy.

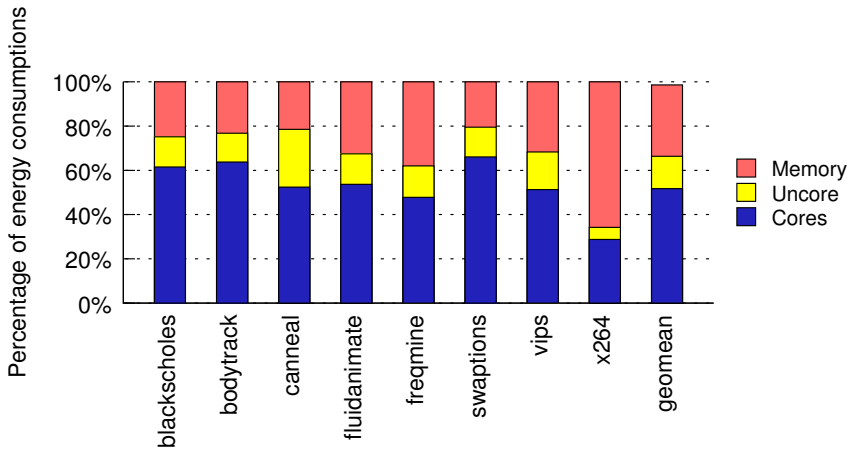


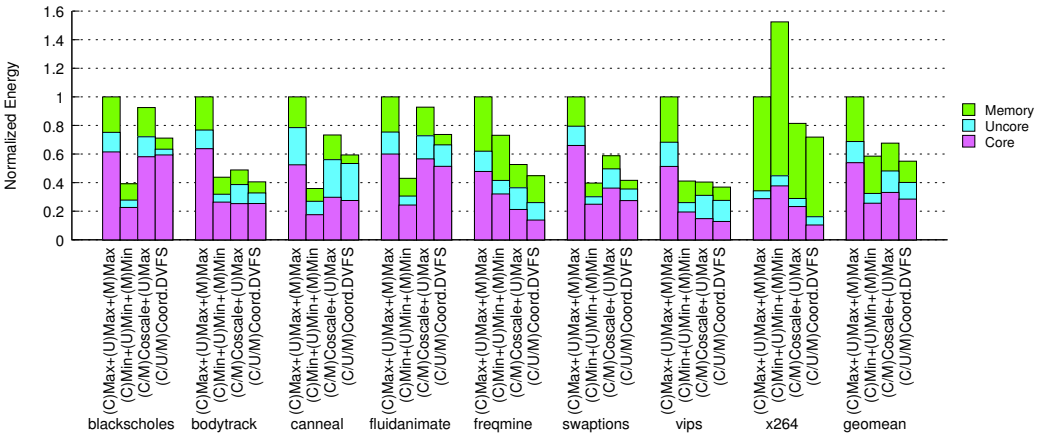
Figure 5.4: Energy distribution of *PARSEC* [3] applications in CMP and main memory.

This graph also shows each applications have different characteristic, and so diverse energy distribution. For instance, energy consumption of uncore in *canneal*

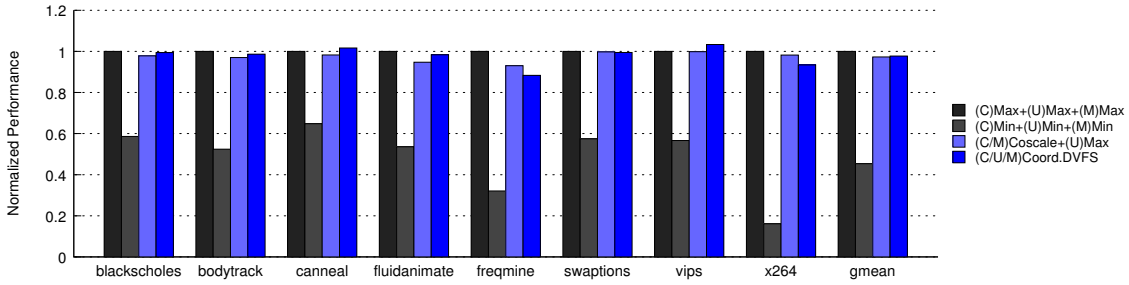
takes 25% of total energy consumption (Core+Uncore+Memory). Even though overall energy consumption of uncore is relatively small, 15%, it is not negligible and more important even for some uncore intensive applications such as *canneal* and *vips*.

5.2.2.1 Overall results of proposed coordinated DVFS for CMP and memory

We describe the results of our proposed coordinated DVFS for CMP(cores and uncore) and memory in this section. Our technique is compared to maximum and minimum frequencies for all CMP and memory.



(a) Normalized energy



(b) Normalized performance

Figure 5.5: Full-system simulation results of PARSEC benchmark suite [3].

In the Fig. 5.5, minimum frequency simulation, (C)Min+(U)Min+(M)Min, shows 48.7% energy savings while showing over 50% performance degradation compared to the maximum frequency simulation. It is also referred as maximum energy savings. The huge performance degradation derives more total energy consumption which includes other components of the system beyond CMP and memory. Also, 50% performance degradation is enough users to feel slowness of the system, so the minimum V/F level is not appropriate for the power management policy. Especially, the minimum frequency simulation of *x264* shows more energy consumption than maximum frequency simulation due to huge performance degradation. Overall, our policy shows similar energy savings, 47%, to the minimum frequency setting while showing negligible performance degradation, 2.3%. In addition, our policy saves energy more than 30% for all various applications. In some applications, our power management policy shows more energy saving than minimum frequency simulation by negligible performance degradation. Comparing to a conventional work [13], our results show 0.4% performance increment and 12% more energy savings of total energy which includes cores, uncore and memory.

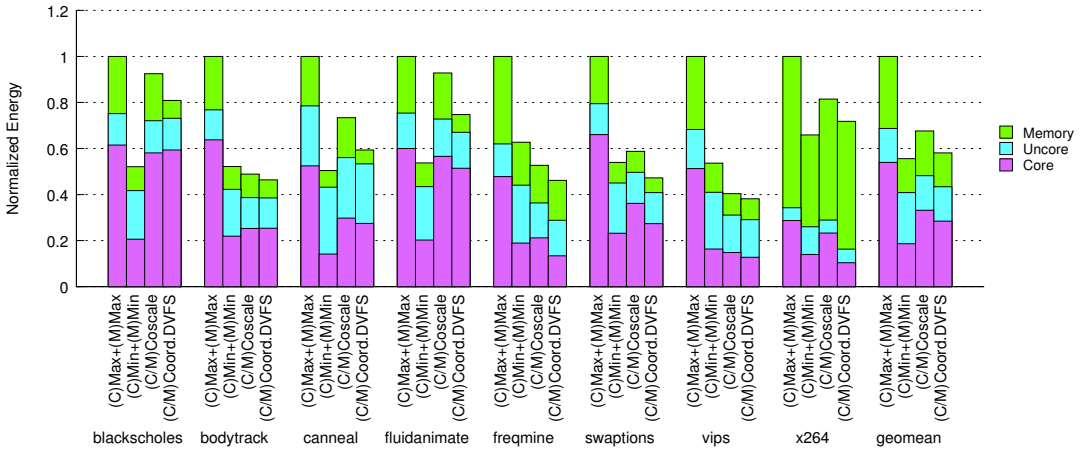
In Fig. 5.5(a), our technique shows similar energy savings of cores, over 50%, to minimum frequency simulations. For some core-bound applications such as *blackscholes* and *fluidanimate*, our technique chooses higher V/F level to avoid much performance degradation even though with less energy savings. In this application, energy savings are shown at uncore, memory or both. Regarding that there is little room to save energy of cores in core-bound applications, our technique saves energy in uncore and memory for *blackscholes* and memory for *fluidanimate*. Overall energy savings of uncore in our policy is about 30% to minimize total performance degradation due to lowering V/F level. It means that uncore usually has high utilization in both of core-bound and memory-bound applications. Energy saving of memory in our pol-

icy is over 60% compared to maximum frequency simulations and 20% more energy savings than the conventional work [13].

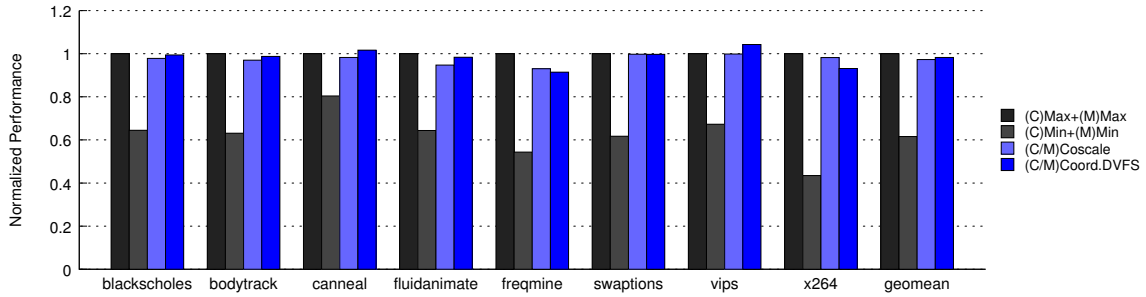
5.2.2.2 Comparison with conventional work for coordinated power management

In this section, we compare our coordinate DVFS technique to a conventional DVFS technique for CMP cores and memory [13]. The previous work is about power management policy based on estimated performance and energy consumption for CMP cores and memory. This work does not include uncore DVFS policy. In summary of their work, they find a combination of V/F levels of cores and memory through exploration of all possible V/F levels. Also, the combination of V/F levels stays within certain performance degradation. For fair comparison, uncore V/F level is fixed to maximum frequency in all simulations. Figure 5.6 shows normalized energy consumption and performance to maximum V/F level for cores and memory. In Fig. 5.6(b), our results show overall 2% performance degradation compared to maximum frequency simulations. In our experiment of the conventional work [13], we set allowed performance degradation to 2% for fair comparison with our proposed technique.

In the Fig. 5.6, minimum frequency simulations for cores and memory shows 44.6% energy savings while showing over 40% performance degradation compared to the maximum frequency simulation. The conventional work [13] shows 35.1% total energy savings with 2.7% performance degradation. Our proposed work shows 44% energy savings which is almost similar to the minimum frequency simulations only with 1.7% performance degradation. Regarding that the minimum frequency simulations shows 40% performance degradation, our proposed coordinate power management technique is more efficient than the minimum frequency simulation. Also, compared to the conventional work, our policy works more efficiently. While



(a) Normalized energy



(b) Normalized performance

Figure 5.6: Full-system simulation results of PARSEC benchmark suite [3] (Uncore frequency is fixed to maximum frequency.

their approach is based on the estimated performance degradation and energy consumption, it is not trivial to acquire accurate model and requires more complicated model for multi-threads applications.

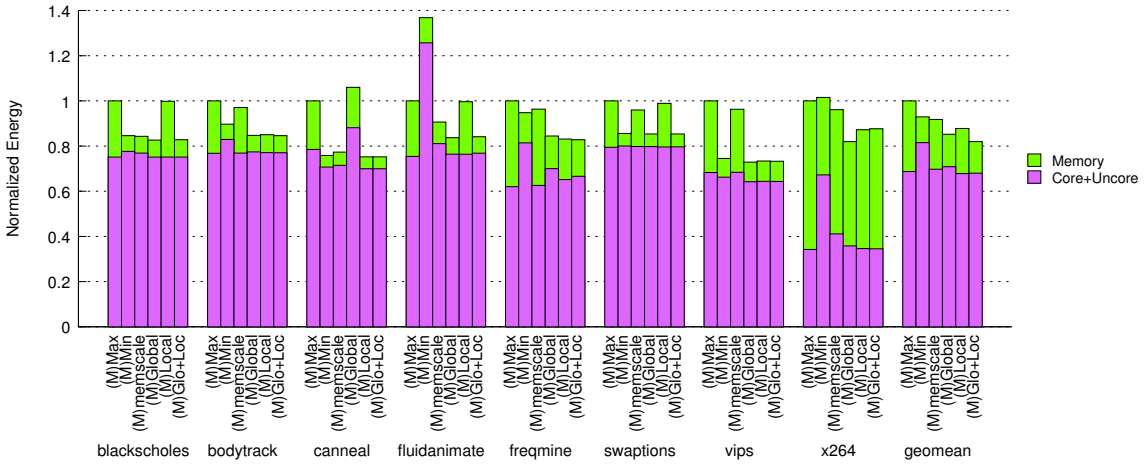
5.2.2.3 Overall results of our proposed memory DVFS technique

In this section, we analyze the effect of our proposed memory DVFS technique with a conventional work [15]. Figure 5.7(a) shows normalized energy of memory and total energy to maximum frequency simulation. The total energy consumption includes cores, uncore and memory. System performance of our proposed technique

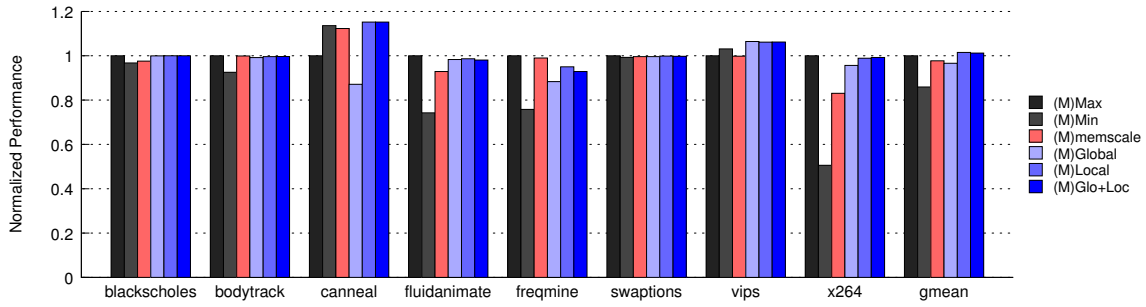
is shown in Fig. 5.7(b).

In Fig. 5.7, we analyze three proposed policies. The V/F levels of cores and uncore is set to maximum V/F level at all simulations in the graph. First policy, (M)Global, is based on global memory access ratio as shown in Eq. 5.1. Second, we analyze (M)Local which is only depends on local memory access ratio as shown in Eq. 5.2. Finally, our proposed memory DVFS technique, (M)Glo+Loc, combines those two policies as shown in right most bars in the graphs. We also added maximum and minimum frequency simulations and a conventional work [15] for comparison. Our (M)Global shows over 60% energy saving in overall. However, it shows less energy savings in *canneal*. In *canneal*, our (M)Local shows much energy savings, 75%, which is similar to minimum frequency simulations. It however does not save much energy on some core -bound applications such as *blackscholes*, *fluidanimate* and *swaptions*. Our final technique, (M)Glo+Loc, shows much energy saving, 75%, in *canneal* and showing 15% performance increment. This means that lowering V/F of memory when uncore has many packets or congestion affects energy saving and performance increment.

We also compared our technique to a convention work for memory DVFS [15]. Our technique shows 20% more energy saving with 3.4% performance increment compared to the work. As we described in section 5.2.1, we uses out-of-order model of CPUs to simulate more realistic architecture. This results proves that our technique is efficient in realistic architecture regarding that it is not trivial to estimate processing time of instructions in the convention approach [15]. In summary, our final technique, (M)Glo+Loc, shows 65% energy saving of memory and 20% total energy saving even with slight performance increment, 1.2%, compare to maximum frequency simulation.



(a) Normalized total energy



(b) Normalized performance

Figure 5.7: Full-system simulation results of PARSEC benchmark suite [3] (Frequencies of cores and uncore are fixed to maximum).

5.2.2.4 Comparison with static V/F

We also conducted some static simulations to compare with our proposed technique for memory power management. Figure 5.8 shows normalized energy and performance compared to maximum V/F level simulation of average of eight PARSEC testcases, *blackscholes*, *bodytrack*, *canneal*, *fluidanimate*, *freqmine*, *swaptions*, *vips*, *x264*. For the static V/F levels, we selected all V/F levels which are used in our DVFS policy. The left ten clusters show the effects of the static simulations. Lower V/F level shows more energy saving of memory and more total energy saving in some high V/F levels. However, the two lowest V/F levels of memory, 267MHz and 200MHz, show less total energy saving even though with more memory energy savings because of lots of performance degradation. Performance degradation (or run-time increment) affects more energy consumption. In summary, our method shows similar energy saving of memory compared to minimum frequency simulation. Also, total energy saving of our technique is over all static simulations while there is no performance degradation in our technique.

5.2.2.5 Cache coherence traffic analysis

In this section, we analyze the traffic of cache coherence protocol as one of the congestion metric of uncore (LLC + interconnect). In general, operating frequency affects the timing of cache access through each cores and main memory. The operating frequency of memory responds (or injects) packets by requests of each cores. Lower frequency of memory injects packets to uncore slower than higher frequency. The slow injection from memory and other injection from each cores change cache coherence protocol among private caches and shared cache. In some applications which have less shared resource, there have more chance to vary cache coherence traffic due to varying operating frequency.

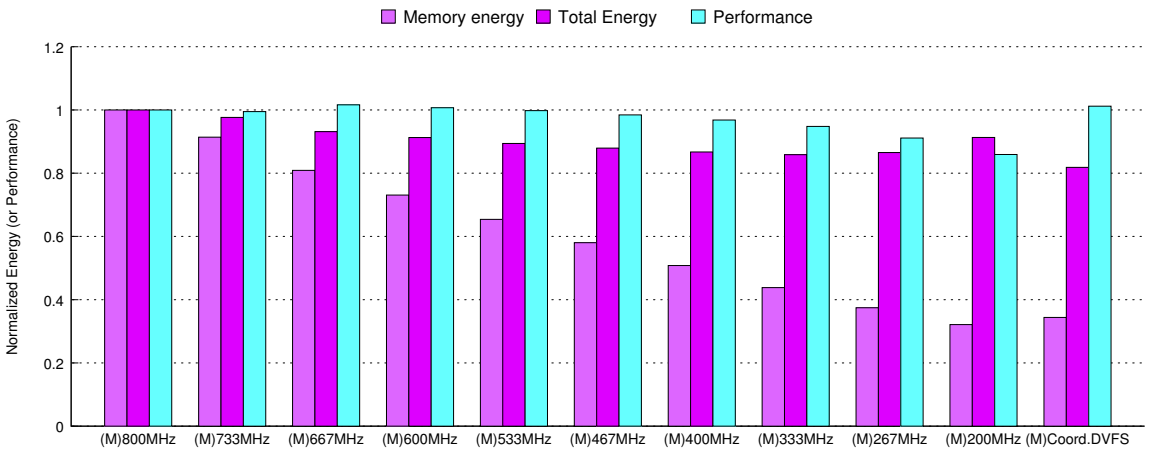
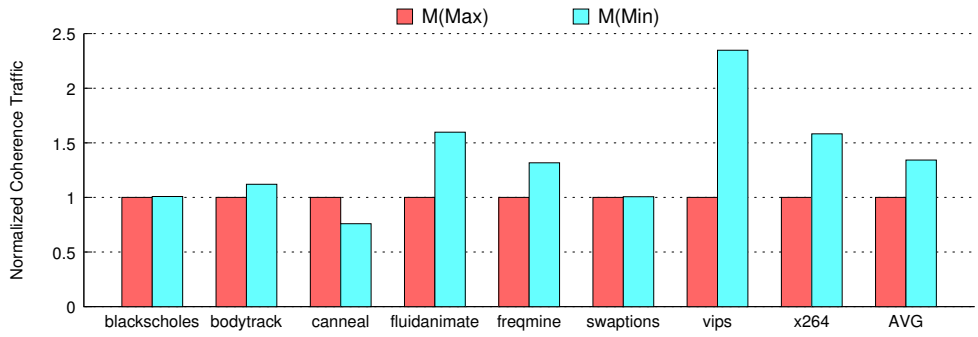
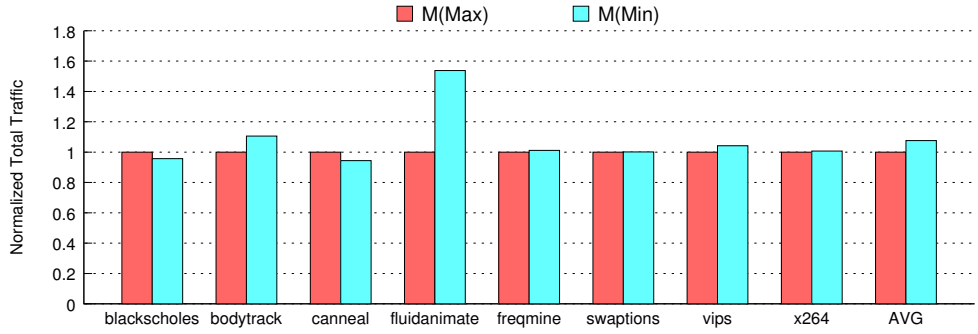


Figure 5.8: Comparison with static memory V/F levels (Frequencies of cores and uncore are fixed to maximum).

Figure 5.9(a) shows cache coherence traffic among private caches and shared cache. We conducted two simulations, maximum and minimum frequency of memory while keeping maximum frequencies for cores and uncore. Our results show 34% increment of cache coherence traffic at minimum frequency of memory. However, *canneal* shows 25% decrement of cache coherence traffic. Referring *canneal*'s behavior, each cores randomly choose two data sets and swap or not according to certain rule. And, each cores keep one of the previous grabbed data and choose new data to increase data reuseness. While the cores choose two data sets, there could be conflicts among the cores. For instance, same data can be chosen by more than one core and swapped from the cores. The conflict represents this situation and requires recovery process. The recovery process may increase the number of cache accesses and coherence traffic. Our result of *canneal* shows decrement of cache coherence traffic. In means that maximum frequency of memory injects more packets into uncore within certain amount of time and generates more conflicts and cache coherence



(a) Normalized cache coherence traffic



(b) Normalized total traffic (coherence + noncoherence)

Figure 5.9: Cache coherence traffic in the shared source

traffic.

We also conducted additional simulation to analyze total traffic which includes non coherent protocol as shown in Fig. 5.9(b). Our results show 7.5% increment overally and 6% decrement in *canneal*.

DVFS policy, we considered shared resource along with cores in CMP and main memory. Our key idea is that high V/F level of memory is not useful when shared resource is congested. Instead, lowering V/F level of memory saves more energy by reducing V/F level of memory without performance degradation. Additionally, lowering V/F level of memory reduces congestion of the shared resource and contributes performance increment. Full system simulations of our proposed memory

power management technique on PARSEC benchmarks shows 66% energy saving of main memory and 18% energy saving of total energy consumption without performance degradation. In addition to memory DVFS, our coordinated DVFS technique for core, uncore and main memory shows 47% energy saving with less than 2.3% performance degradation.

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

This work focused on power management through dynamic voltage and frequency scaling (DVFS) for cores and uncore of CMP and main memory. The cores of CMP and main memory take large portion of CMP energy consumed. As such, the CMP uncore constitutes a significant and increasing part of overall CMP power dissipation to achieve consistent performance scalability in these designs to satisfy the demands of application data growth, requires a super-linear expansion in uncore. More specifically, the first work focused on DVFS technique for the uncore (i.e. LLC and on-chip communication fabric). In this work, various Artificial Neural Network-based (ANN) techniques are proposed to achieve the best power savings while maintaining negligible performance degradation. Also, we propose novel approach, wherein a Proportional Integral (PI) controller, which can adapt to short system changes, but lacks long-term pattern recognition, is used in tandem with the ANN. In addition to uncore power management, we propose a resource sharing-driven DVFS technique for CMP cores' designs. Our technique shows benefits of performance increment even with energy saving for intensive shared resource-based applications. As such, significant energy saving is obtained with negligible performance loss and sometimes performance gain. For the last work, we propose a power management technique with coordination among CMP cores, uncore and main memory. Our technique uses CMP uncore utilization factor because uncore plays an important role as a path(or bottleneck) between CMP cores and main memory. Full system simulation results show that this technique can reduce energy dissipation by over 47% with almost no performance degradation (sometimes performance gain) compared to baseline.

REFERENCES

[1] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha. GARNET: a detailed on-chip network model inside a full-system simulator. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pages 33–42, 2009.

[2] L. Barroso and U. Holzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2009.

[3] C. Bienia, S. Kumar, J.P. Singh, and K. Li. The PARSEC benchmark suite: characterization and architectural implications. In *Proceedings of the ACM/IEEE International Conference on Parallel Architectures and Compilation Techniques*, pages 72–81, 2008.

[4] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *ACM Computer Architecture News*, 39(2):1–7, May 2011.

[5] R. Bitirgen, E. Ipek, and J. F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 318–329, 2008.

[6] P. Bogdan, R. Marculescu, S. Jain, and R. T. Gavila. An optimal control approach to power management for multi-voltage and frequency islands multiprocessor platforms under highly variable workloads. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip*, pages 35–42, 2012.

[7] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. QNoC: QoS architecture and design process for network on chip. *Elsevier Journal of Systems Architecture*, 50(2-3):105–128, February 2004.

[8] L. S. Brakmo and L. L. Peterson. TCP Vegas: end to end congestion avoidance on a global internet. *IEEE Journal of Selected Areas in Communications*, 13(8):1465–1480, October 1995.

[9] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, and U. Ogras ad R. Ayoub. Dynamic voltage and frequency scaling for shared resources in multicore processor designs. In *Proceedings of the ACM/IEEE Design Automation Conference*, 2013.

[10] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, and U. Ogras. In-network monitoring and control policy for dvfs of cmp networks-on-chip and last level caches. In *Proceedings of the ACM/IEEE International Symposium on Networks-on-Chip*, pages 43–50, 2012.

[11] S. Damaraju, V. George, S. Jahagirdar, T. Khondker, R. Milstrey, S. Sarkar, S. Siers, I. Stolero, and A. Subbiah. A 22nm IA multi-CPU and GPU system-on-chip. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 56–57, 2012.

[12] H. David, C. Fallin, E. Gorbatov, U. R. Hanebutte, and O. Mutlu. Memory power management via dynamic voltage/frequency scaling. In *Proceedings of the ACM International Conference on Autonomic computing*, pages 31–40, 2011.

[13] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini. Coscale: Coordinating cpu and memory system dvfs in server systems. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 143–154, 2012.

[14] Q. Deng, D. Meisner, A. Bhattacharjee, T. F. Wenisch, and R. Bianchini. Multiscale: Memory system dvfs with multiple memory controllers. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 297–302, 2012.

[15] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini. Memscale: Active low-power modes for main memory. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 225–238, 2011.

[16] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. Leblanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid State Circuits*, SC-9(5):256–268, October 1974.

[17] E. Ebrahimi, C. J. Lee, O. Mutlu, and Y. N. Patt. Fairness via source throttling: a configurable and high-performance fairness substrate for multi-core memory systems. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 335–346, 2010.

[18] S. Eyerman and L. Eeckhout. Fine-grained DVFS using on-chip regulators. *ACM Transactions on Architecture and Code Optimization*, 8(1):1–24, April 2011.

[19] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture*, pages 148–157, 2002.

[20] B. Grot, J. Hestness, S. W. Kecklet, and O. Mutlu. Kilo-NOC: a heterogeneous network-on-chip architecture for scalability and service guarantees. In *Proceed-*

ings of the ACM/IEEE International Symposium on Computer Architecture, pages 401–412, 2011.

[21] B. Grot, S. Keckler, and O. Mutlu. Preemptive virtual clock: a flexible, efficient, and cost-effective qos scheme for networks-on-chip. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 268–279, 2009.

[22] L. Guang, E. Nigussie, L. Koskinen, and H. Tenhunen. Autonomous DVFS on supply islands for energy-constrained NoC communication. *Lecture Notes in Computer Science: Architecture of Computing Systems*, 5455/2009:183–194, 2009.

[23] Simon S. Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall International, 1999.

[24] Sebastian Herbert and Diana Marculescu. Analysis of dynamic voltage/frequency scaling in chip-multiprocessors. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 38–43, 2007.

[25] International Technology Roadmap for Semiconductors (ITRS) Working Group. International Technology Roadmap for Semiconductors (ITRS), 2011 Edition.

[26] 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 *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 347–358, 2006.

[27] R. Iyer. CQoS: a framework for enabling QoS in shared caches of CMP platforms. In *Proceeding of the ACM International Conference on Supercomputing*, pages

257–266, 2004.

[28] Anil K. Jain, Jianchang Mao, and K.M. Mohiuddin. Artificial neural network: A tutorial. *IEEE Computer*, 29:31–44, 1996.

[29] D.-C. Juan and D. Marculescu. Power-aware performance increase vis core/uncore reinforcement control for chip-multiprocessors. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 97–102, 2012.

[30] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi. ORION 2.0: a power-area simulator for interconnection networks. *TVLSI*, 20(1):191–196, January 2012.

[31] E. Kakoulli, V. Soteriou, and T. Theocharides. Intelligent hotspot prediction for network-on-chip-based multicore systems. *IEEE Transactions on Computer-Aided Design*, 31(3):418–431, March 2012.

[32] W. Kim, M. S. Gupta, G. Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture*, pages 123–134, 2008.

[33] Cyril Kowaliski. Gelsinger reveals details of Nehalem, Larrabee, Dunnington, 2008.

[34] R. Kumar and G. Hinton. A family of 45nm IA processors. In *Proceedings of the IEEE International Solid-State Circuits Conference*, pages 58–59, 2009.

[35] J. W. Lee, M. C. Ng, and K. Asanovic. Globally-synchronized frames for guaranteed quality-of-service in on-chip networks. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture*, pages 89–110, 2008.

[36] M. M. Lee, J. Kim, D. Abts, M. Marty, and J. W. Lee. Probabilistic distance-based arbitration: providing equality of service for many-core CMPs. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 509–519, 2010.

[37] B. Li, L. Zhao, R. Iyer, L.-S. Peh, M. Leddige, M. Espig, S. E. Lee, and D. Newell. CoQoS: coordinated QoS-aware shared resources in NoC-based SoCs. *Elsevier Journal of Parallel and Distributed Computing*, 71(5):700–713, May 2011.

[38] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi. Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architecture. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 469–480, 2009.

[39] G. Liang and A. Jantsch. Adaptive power management for the on-chip communication network. In *Proeedings of the Euromicro Conference on Digital System Design*, pages 649–656, 2006.

[40] S. H. Low, L. Peterson, and L. Wang. Understanding Vegas: A duality model. *Journal of the ACM*, 49(2):207–235, March 2002.

[41] J. Luo, N. K. Jha, and L.-S. Peh. Simultaneous dynamic voltage scaling of processors and communication links in real-time distributed embedded systems. *TVLSI*, 15(4):427–437, April 2007.

[42] J. F. Martinez and E. Ipek. Dynamic multicore resource management: a machine learning approach. *IEEE Micro*, 29(5):8–17, September 2009.

[43] W.S. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bull. Mathematical Biophysics*, 5:115–133, 1943.

[44] Micron. *Calculating Memory System Power for DDR3*, 2007.

[45] R. Miftakhutdinov, E. Ebrahimi, and Y. N. Patt. Predicting performance impact of dvfs for realistic memory system. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 155–165, 2012.

[46] Marvin L Minsky and Seymour A Papert. *Perceptrons - Expanded Edition: An Introduction to Computational Geometry*. MIT press Boston, MA:, 1987.

[47] A. K. Mishra, R. Das, S. Eachempati, R. Iyer, N. Vijaykrishnan, and C. R. Das. A case for dynamic frequency tuning in on-chip networks. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 292–303, 2009.

[48] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. CACTI 6.0: a tool to model large caches. Technical report, HP Laboratories, 2009.

[49] O. Mutlu and T. Moscibroda. Parallelism-aware batch scheduling enhancing both performance and fairness of shared DRAM systems. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture*, pages 63–74, 2008.

[50] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith. Fair queuing memory systems. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*, pages 208–222, 2006.

[51] K. J. Nesbit, J. Laudon, and J. E. Smith. Virtual private caches. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture*, pages 57–68, 2007.

[52] U. Y. Ogras, R. Marculescu, and D. Marculescu. Variation-adaptive feedback control for networks-on-chip with multiple clock domains. In *Proceedings of the*

ACM/IEEE Design Automation Conference, pages 614–619, 2008.

[53] A. Rahimi, M. E. Salehi, S. Mohammadi, and S. M. Fakhraie. Low-energy GALS NoC with FIFO-monitoring dynamic voltage scaling. *Microelectronics Journal*, 42(6):889–896, June 2011.

[54] F. Rasheed. Artificial neural network circuit for spectral pattern recognition. Master's thesis, Texas A&M University, College Station, 2013.

[55] Srinivas Shakkottai and R. Srikant. *Foundations and Trends in Networking: Network Optimization and Control*, volume 2. Now, 2007.

[56] L. Shang, L. Peh, and N. K. Jha. Power-efficient interconnection networks: dynamic voltage scaling with links. *IEEE Computer Architecture Letters*, 1(1), 2002.

[57] S. W. Son, K. Malkowski, G. Chen, M. Kandemir, and P. Raghavan. Integrated link/CPU voltage scaling for reducing energy consumption of parallel sparse maxtrix applications. In *IPDPS*, 2006.

[58] V. Soteriou, N. Eisley, and L.-S. Peh. Software-directed power-aware interconnection networks. *ACM Transactions on Architecture and Code Optimization*, 4(1), March 2007. Article No. 5.

[59] V. Spiliopoulos, S. Kaxiras, and G. Keramidas. Green governors: A framework for continuously adaptive DVFS. In *Proceedings of the International Green Computing Conference and Workshops*, pages 1–8, 2011.

[60] R. Stefan, A. Molnos, A. Ambrose, and K. Goossens. A TDM NoC supporting QoS, multicast, and fast connection set-up. In *Proceedings of the ACM/IEEE Design Automation and Test in Europe*, pages 1283–1288, 2012.

[61] G. Steven, R. Anguera, C. Egan, F. Steven, and L. Vintan. Dynamic branch prediction using neural networks. In *Euromicro Symposium on Digital Systems Design*, pages 178–185, 2001.

[62] J.-Y. Won, X. Chen, P. Gratz, J. Hu, and V. Soteriou. Up by their bootstraps: online learning in artificial neural networks for cmp uncore power management. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture*, 2014, © 2014 IEEE.

[63] J.-Y. Won, P. Gratz, S. Shakkottai, and J. Hu. Having your cake and eating it too: Energy savings without performance loss through resource sharing driven power management. In *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2015, © 2015 IEEE/ACM.