Up By Their Bootstraps: Online Learning in Artificial Neural Networks for CMP Uncore Power Management

Jae-Yeon Won, Xi Chen, Paul Gratz and Jiang Hu
Texas A&M University
{jaeyeon9,xchen19,pgratz,jianghu}@tamu.edu

Vassos Soteriou
Cyprus University of Technology
vassos.soteriou@cut.ac.cy

Abstract

With increasing core counts in Chip Multi-Processor (CMP) designs, the size of the on-chip communication fabric and shared Last-Level Caches (LLC), which we term uncore here, is also growing, consuming as much as 30% *of die area and a significant portion of chip power budget. In this work, we focus on improving the uncore energy-efficiency using dynamic voltage and frequency scaling. Previous approaches are mostly restricted to reactive techniques, which may respond poorly to abrupt workload and uncore utility changes. We find, however, 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). ANNs, however, require training to produce accurate predictions. Architecting an efficient training mechanism without a priori knowledge of the workload is a major challenge. We propose a novel technique in which a simple Proportional Integral (PI) controller is used as a secondary classifier during ANN training, dynamically pulling the ANN up by its bootstraps to achieve accurate predictions. Both the ANN and the PI controller, then, work in tandem once the ANN training phase is complete. The advantage of using a PI controller to initially train the ANN is a dramatic acceleration of the ANN's initial learning phase. Thus, in a real system, this scenario allows quick power-control adaptation to rapid application phase changes and context switches during execution. We show that the proposed technique produces results comparable to those of pure offline training without a need for prerecorded training sets. Full system simulations using the PARSEC benchmark suite show that the bootstrapped ANN improves the energy-delay product of the uncore system by* 27% *versus existing state-of-the-art methodologies.*

1 Introduction

Due to chip power density limitations as well as the recent breakdown of Dennard's Scaling [10] over the past decade, performance growth in microprocessor design has largely been driven by core scaling. These trends have led to tens of cores, Chip Multi-Processor (CMP) designs, expected to grow to the thousands in the pursuit of exa-scale computing [6]. Achieving a consistent performance scalability in these designs to satisfy the demands of application data growth, requires a super-linear expansion in the last-level cache (LLC) size and on-chip communication bandwidth. 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 [18]. As the uncore expands relative the cores, occupying a greater portion of the CMP's real-estate, it has become a critical consumer of the overall CMP's power budget; therefore, energy-efficient uncore operation is of paramount importance in restraining the CMP's overall power envelope.

To achieve such a goal, this work focuses on dynamic voltage and frequency scaling (DVFS) for the CMP uncore. Although DVFS has been extensively studied in the literature [5, 13, 20, 23, 25, 26, 28], previous works have been mostly core-centric, i.e., they focus on reducing the power consumed in cores, paying 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 [19, 8, 7]. 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 [19] or Proportional Integral (PI) controller [8, 7], 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 [21]. 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 [31] and predicting traffic congestion hotspots in NoCs [17]. 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 [8] 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.

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 [7].
- We propose a novel, purely-online, tandem ANN-PI power manager, which further improves energy-delay product by 27% versus prior techniques [7] 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 examine the compute latency and power consumption of the proposed software ANN implementation versus a ANN hardware design, verifying that software ANN delay impact is insignificant and incremental power is $\sim .32\%$ of total CMP power.

2 Background

Here we first discuss Artificial Neural Networks (ANNs) basics, and then we discuss how the V/F state interacts with NoC and LLC (collectively referred to as the uncore).

2.1 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.

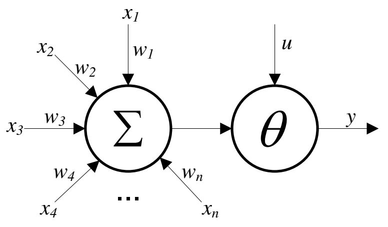


Figure 1: Model of a single neuron.

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 1, and is defined by Equation 1 [21]:

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

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 [22]. Multi-level networks of these perceptron models, i.e. ANNs, do not have this limitation [15]. Here, each neuron is treated as a node, forming directed graph with input and output edges. An example ANN is illustrated in Figure 2, where each circle represents a neuron. The ANN in Figure 2 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

$$w_j(t+1) = w_j(t) + g \cdot (d - y) \cdot x_j \tag{2}$$

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

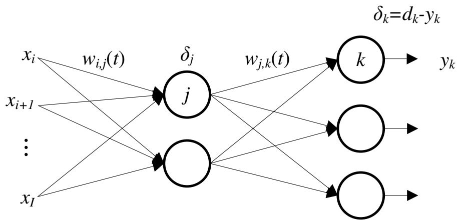


Figure 2: Multi-layer feed-forward ANN.

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 [14]. We use a 2-layer ANN topology in Figure 2, 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 middle layer node is obtained by

$$\delta_j = \sum_{k=1}^{K} (w_{j,k}(t) \cdot \delta_k) \tag{3}$$

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. 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

$$w_{i,j}(t+1) = w_{i,j}(t) + g \cdot \delta_j \cdot \frac{d\theta(\psi_j)}{d\psi_j} \cdot x_i \tag{4}$$

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

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) [9]: 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 3.

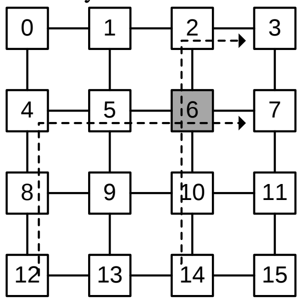


Figure 3: 16-core CMP in a 4×4 2D mesh array. The darkened tile indicates location of the Power Control Unit (PCU). The dashed lines indicate paths traversing the NoC.

Similar to the design originally proposed by Chen et al. [8, 7], 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 3). 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 [8].

2.3 PI controller

As a basis of comparison and as a subcomponent of our design we also utilize a proportional-integral (PI) controller [8, 7]. 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 5. K_p and K_i are the proportional and integral error gain, respectively, and are typically determined empirically.

$$u_t = K_p e_t + K_i \sum_{k=1}^{t} e_k \tag{5}$$

3 Related Work: DVFS in Multicore Systems

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.[28] 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 [30]. Son et al. proposed simultaneous CPU-NoC link DVFS for a specific application – parallel linear system solving [29]. Luo, et al., combined NoC link DVS with task scheduling of embedded systems [20]. Ogras, et al., applied state-space control for DVFS on tile-based designs where the NoC was partitioned and associated with processing cores [25]. Mishra et al. examined DVFS in NoC router designs [23]. The work of Guang et al. [13] 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 [26]. 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 [5]. There are very few previous works addressing DVFS for caches. Flautner et al. presented one such work, which applied DVS to individual cache lines [12].

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 [19, 8, 7]. Liang and Jantsch [19] 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. [8], 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. [7] 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 [4]. 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.

4 Tandem ANN and PI Control

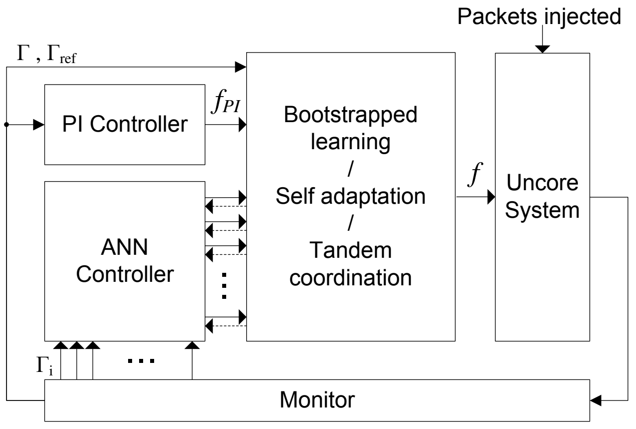

Packets injected
Γ , Γ_{ref}
PI Controller
f_{PI}
Bootstrapped learning / Self adaptation / Tandem coordination
f
Uncore System
ANN Controller
Γ_i
Monitor

Figure 4: Architecture of the uncore DVFS control system.

An overview of the proposed uncore DVFS control system is depicted in Figure 4. 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.

4.1 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. [7], which is defined as

$$\Gamma = \eta \cdot \lambda_U \tag{6}$$

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 al.[7] 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.

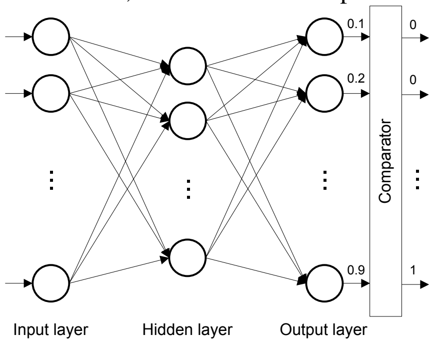


Figure 5: Proposed 3-layer feed-forward ANN.

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 5. 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 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

$$f(x) = ae^{-\frac{(x-b)^2}{2c^2}} \tag{7}$$

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 (6). The learning algorithm here is the common back propagation algorithm described in Section 2.1.

4.2 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 2.1. In Section 4.2.1, 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 4.2.2. We propose an "up by the bootstraps" learning technique using PI control as a secondary classifier in Section 4.2.3.

4.2.1 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[1]. 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}.

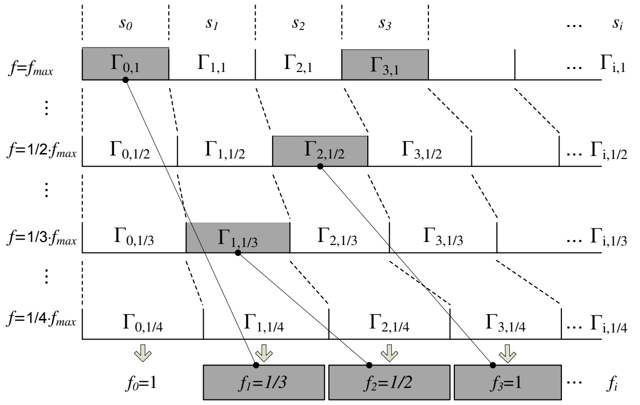


Figure 6: Input and target sets for ANN learning.

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

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

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 6 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 2.1.

4.2.2 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 2.1 except that the target frequency is obtained online as in the case of feedback control. In typical feedback control techniques, such as PI control [8], 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

$$e_i = \Gamma_i - \beta \cdot \Gamma_{ref,i} \tag{8}$$

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 [7], 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.

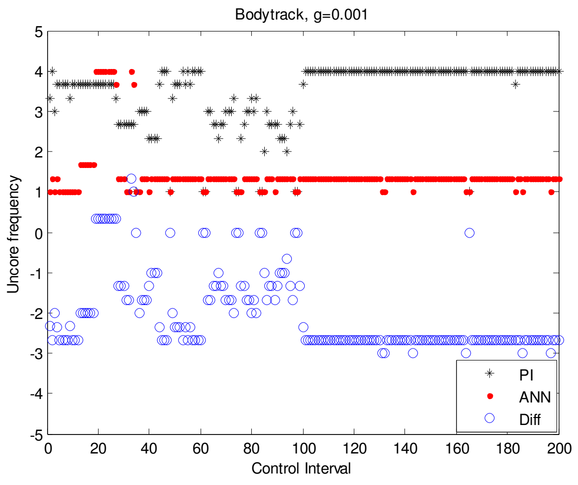
4.2.3 Bootstrapped Learning Using a PI Controller

Although the self-adaptation techniques presented in Section 4.2.2 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[2]. General-purpose CMP workloads can vary so greatly that developing a 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 [8, 7] 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 4.2.2) the ANN can exceed the performance of the PI controller. In Section 4.3, we will show that the PI controller may also be used for tandem control once the ANN is trained as well.

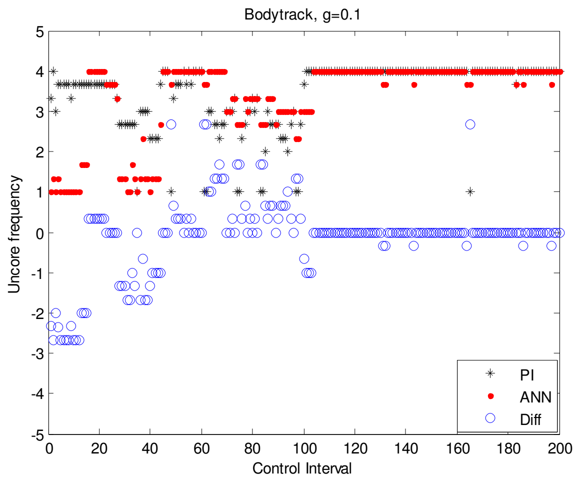
The offline supervised learning is often conducted based upon complete traces of many applications. By contrast, bootstrapped learning is performed during the beginning phases of each single application. Hence, it should be much faster and requires a greater learning gain (see Equation (2)). Furthermore, 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.

Figure 7 compares the bootstrapped learning with different gains applied to the *Bodytrack* application of PARSEC benchmark suite [2]. 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 7a 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

[2]We explored purely online training with the techniques discussed in Section 4.2.2, however the results were poor due to the long training time required, these results were dropped from this paper for brevity.



(a) gain=0.001



(b) gain=0.1

Figure 7: Bootstrapping learning applied to the *Bodytrack* application of the PARSEC benchmark suite [2].

200 intervals. Experimental results with $g = 0.1$ are shown in Figure 7b, which exhibits that the ANN output starts to follow the PI control after approximately 100 intervals.

4.2.4 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 (8). 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

$$g_i = \begin{cases} g_{max} & \text{if } e_i \leq \tau \\ g_{max} - \left(\frac{e_i - \tau}{e_{max} - \tau}\right) \cdot (g_{max} - g_{min}) & \text{if } \tau < e_i < e_{max} \\ g_{min} & \text{otherwise} \end{cases} \tag{9}$$

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

4.3 ANN-PI Tandem Control

As discussed in Section 4.2, the ANN controller proactively 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.

4.3.1 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

$$\bar{e}_i = \frac{\sum_{j=i-m+1}^{i} \sum_{l=1}^{n} e_{j,l}}{m \cdot n} \tag{10}$$

where $e_{j,l}$ is the error defined in Equation (8) 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

$$\xi_i = 1 - \left(\sum_{j=i-m+1}^{i} \frac{|f_{j,ANN} - f_{j,PI}|}{k-1} \right) / m \tag{11}$$

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 (11) is the average normalized difference between the ANN and the PI computed results in the past m intervals.

The rules for the choices between the PI or the ANN are listed in Table 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 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.

$\mathcal{I}_i$	$\bar{e}_i$	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 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.

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.

4.3.2 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 (8)) 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 4.2.2, except that it is directly applied to control decisions, while the adaptation is to improve the ANN.

4.3.3 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.

5 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. [8]. As with their work, there is a PCU (Power Control Unit) [9], 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 [18]. Every core collects its critical latency Γ information, 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. [8, 7].

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 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 2: ANN control computing runtime in PCU clock cycles.

As in the work by Chen et al. [8, 7], 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 8. 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 6.2.5.

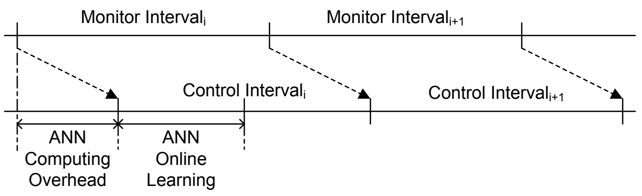


Figure 8: Pipelined monitoring and control intervals.

6 Evaluation

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

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: System parameters used for full-system simulations.

6.1 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 NoC topology is a 4×4 2D mesh, with each node/router attached to a single processor core. Table 3 summarizes the baseline CMP setup.

Simulation experiments are performed using the gem5 [3] 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 [2]. 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 [16] and CACTI 6.0 [24] 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 [11]). During V/F transitions, the uncore operation is halted.

The ANN configuration is summarized in Table 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 between 0.001 and 0.1, as described in Section 4.2.4. 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 5).

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 4: ANN configuration parameters.

6.2 Experimental Evaluation

6.2.1 Overall Results

We compare the following 6 methods:

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

PI: best method from Chen et al. [7].

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

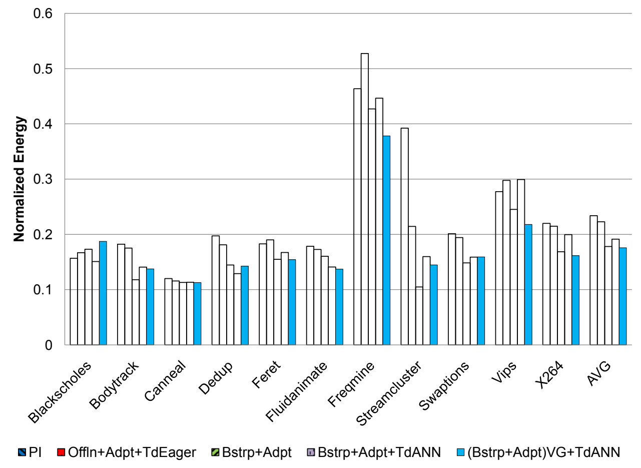
Bstrp+Adpt: bootstrapped learning, self-adaptation and ANN control.

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

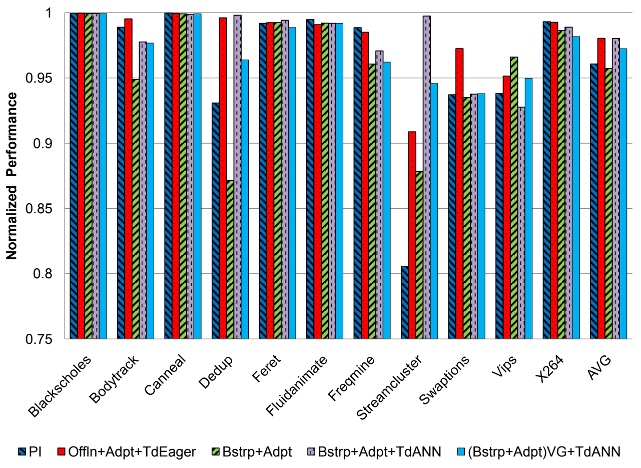
(Bstrp+Adpt)VG+TdANN: bootstrapped learning and self-adaptation with variable gain, ANN-centric tandem control.

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 9. Compared to the PI control [7], 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.

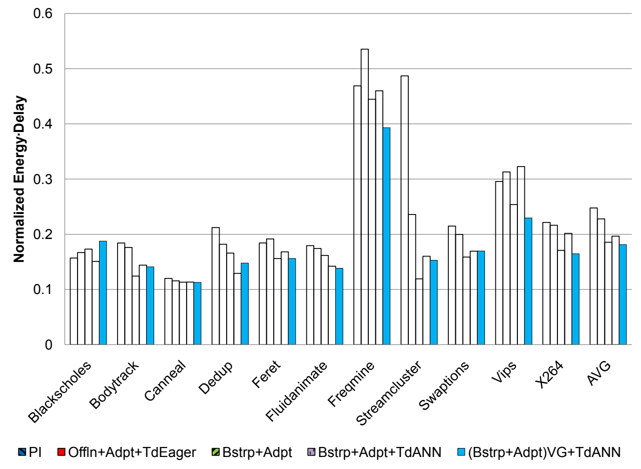
The energy-performance tradeoffs of our offline and bootstrapped learning schemes are depicted in Figure 10. To produce the results displayed in Figure 10, we found the average energy delay (ED) across all PARSEC benchmarks for each of the three techniques while sweeping many of the various parameters of each technique. The Pareto optimal points for each of the PI control, offline and bootstrapped methods were then plotted as shown in Figure 10. As the



(a) Normalized energy



(b) Normalized performance



(c) Normalized energy × delay product

Figure 9: Overall full-system experimental results.

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.

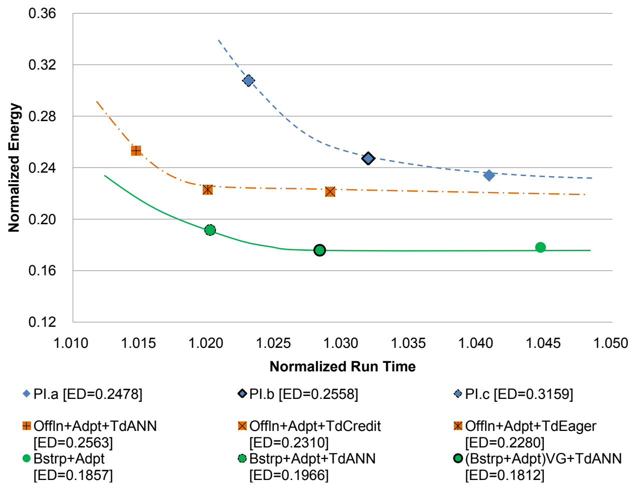


Figure 10: Pareto optimal points and normalized energy-normalized run-time curves for PI control-, offline-, and bootstrapped learning-based methods.

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.

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

Figure 11 shows the benefit of the online self-adaptation (see Section 4.2.2) on two options of the pre-operational learning for ANNs: the offline supervised learning (Section 4.2.1) and the online bootstrapped learning (Section 4.2.3) 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-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 [7] 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 4.2.2). From Figure 11, it is clear

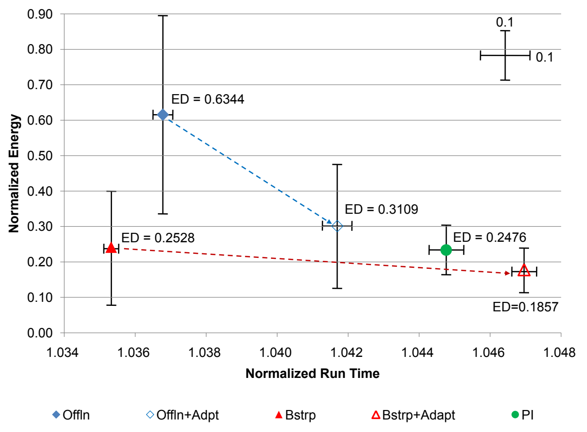


Figure 11: Effect of online self-adaptation. ED: energy×delay product.

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.

6.2.3 Effectiveness of ANN-PI Tandem Control

Three ANN-PI tandem control techniques are proposed in section 4.3: ANN-centric tandem (TdANN), eager tandem (TdEager) and credit-based tandem (TdCredit). Figure 12 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.

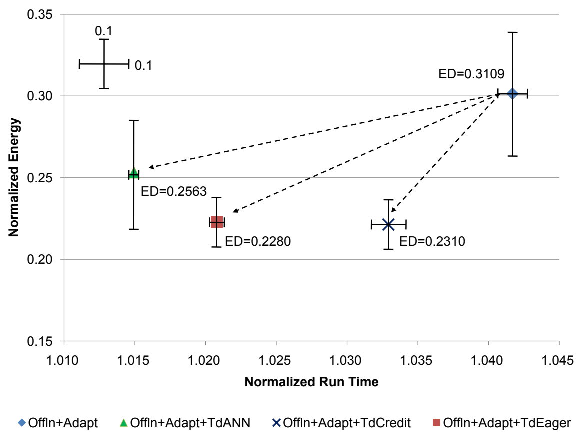


Figure 12: ANN-PI tandem control based on offline supervised learning.

6.2.4 Effect of Variable Learning Gain

In Section 4.2.4, 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 13 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 $6-8\%$.

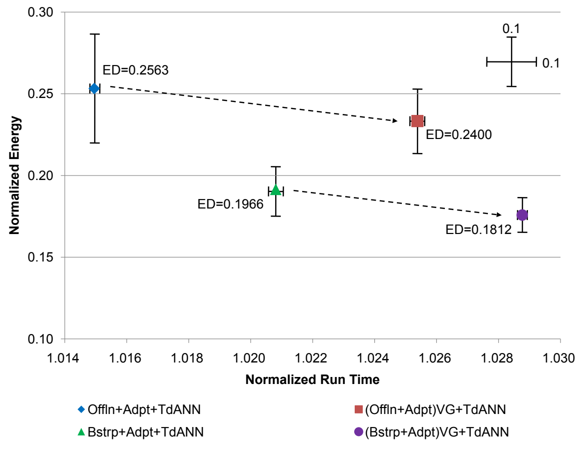


Figure 13: The effect of variable learning gain.

6.2.5 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 8. 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 [27], 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.

7 Conclusions

The CMP uncore constitutes 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 CMP, 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 CMP 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 CMP 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%.

8 Acknowledgements

This work is supported in part by the Intel Corporation. Special thanks Prof. Theocharis Theocharides at UCY for valuable advice and guidance.

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 *ISPASS*, pages 33–42, 2009.

[2] C. Bienia, S. Kumar, J. Singh, and K. Li. The PARSEC benchmark suite: Characterization and architectural implications. In *PACT*, 2008.

[3] 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.

[4] R. Bitirgen, E. Ipek, and J. F. Martinez. Coordinated management of multiple interacting resources in chip multiprocessors: A machine learning approach. In *MICRO*, pages 318–329, 2008.

[5] 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 *NOCS*, pages 35–42, 2012.

[6] S. Borkar. Thousand core chips: a technology perspective. In *DAC*, pages 746–749, 2007.

[7] X. Chen, Z. Xu, H. Kim, P. Gratz, J. Hu, M. Kishinevsky, and U. O. ad R. Ayoub. Dynamic voltage and frequency scaling for shared resources in multicore processor designs. In *DAC*, 2013.

[8] 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 *NOCS*, pages 43–50, 2012.

[9] 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 *ISSCC*, pages 56–57, 2012.

[10] 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, Oct. 1974.

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

[12] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy caches: simple techniques for reducing leakage power. In *ISCA*, pages 148–157, 2002.

[13] 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.

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

[15] A. K. Jain, J. Mao, and K. Mohiuddin. Artificial neural network: A tutorial. *IEEE Computer*, 29:31–44, 1996.

[16] 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, Jan. 2012.

[17] E. Kakoulli, V. Soteriou, and T. Theocharides. Intelligent hotspot prediction for network-on-chip-based multicore systems. *TCAD*, 31(3):418–431, Mar. 2012.

[18] C. Kowaliski. Gelsinger reveals details of Nehalem, Larrabee, Dunnington, 2008.

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

[20] 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, Apr. 2007.

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

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

[23] 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 *MICRO*, pages 292–303, 2009.

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

[25] U. Y. Ogras, R. Marculescu, and D. Marculescu. Variation-adaptive feedback control for networks-on-chip with multiple clock domains. In *DAC*, pages 614–619, 2008.

[26] 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.

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

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

[29] 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.

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

[31] 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.