Dynamic Lifetime Reliability and Energy Management for Network-on-Chip based Chip Multiprocessors

Milad Ghorbani Moghaddam
Marquette University

DYNAMIC LIFETIME RELIABILITY AND ENERGY MANAGEMENT FOR NETWORK-ON-CHIP BASED CHIP MULTIPROCESSORS

by

Milad Ghorbani Moghaddam, B.S., M.S.

A Dissertation Submitted to the Faculty of the
Graduate School, Marquette University,
in Partial Fulfillment of the Requirements for
the Doctor of Philosophy

Milwaukee, Wisconsin

December 2018

ABSTRACT

DYNAMIC LIFETIME RELIABILITY AND ENERGY MANAGEMENT FOR NETWORK-ON-CHIP BASED CHIP MULTIPROCESSORS

Milad Ghorbani Moghaddam, B.S., M.S.
Marquette University

In this dissertation, we study dynamic reliability management (DRM) and dynamic energy management (DEM) techniques for network-on-chip (NoC) based chip multiprocessors (CMPs). In the first part, the proposed DRM algorithm takes both the computational and the communication components of the CMP into consideration and combines thread migration and dynamic voltage and frequency scaling (DVFS) as the two primary techniques to change the CMP operation. The goal is to increase the lifetime reliability of the overall system to the desired target with minimal performance degradation. The simulation results on a variety of benchmarks on 16 and 64 core NoC based CMP architectures demonstrate that lifetime reliability can be improved by 100% for an average performance penalty of 7.7% and 8.7% for the two CMP architectures. In the second part of this dissertation, we first propose novel algorithms that employ Kalman filtering and long short term memory (LSTM) for workload prediction. These predictions are then used as the basis on which voltage/frequency (V/F) pairs are selected for each core by an effective dynamic voltage and frequency scaling algorithm whose objective is to reduce energy consumption but without degrading performance beyond the user set threshold. Secondly, we investigate the use of deep neural network (DNN) models for energy optimization under performance constraints in CMPs. The proposed algorithm is implemented in three phases. The first phase collects the training data by employing Kalman filtering for workload prediction and an efficient heuristic algorithm based on DVFS. The second phase represents the training process of the DNN model and in the last phase, the DNN model is used to directly identify V/F pairs that can achieve lower energy consumption without performance degradation beyond the acceptable threshold set by the user. Simulation results on 16 and 64 core NoC based architectures demonstrate that the proposed approach can achieve up to 55% energy reduction for 10% performance degradation constraints. Simulation experiments compare the proposed algorithm against existing approaches based on reinforcement learning and Kalman filtering and show that the proposed DNN technique provides average improvements in energy-delay-product (EDP) of 6.3% and 6% for the 16 core architecture and of 7.4% and 5.5% for the 64 core architecture.

ACKNOWLEDGEMENTS

Milad Ghorbani Moghaddam, B.S., M.S.

I would like to sincerely thank my advisor Dr. Cristinel Ababei who was the most impactful person throughout my Ph.D. study at Marquette University. I will always be grateful for his advice and encouragement, which were essential to the completion of this dissertation. I would like to thank him not only for helping me with his tremendous academic support, knowledge, trust and encouragement, but also for sharing his compassionate advice and experience that motivated me to take more professional steps in my future life. I cannot thank him enough for all his support and devotion throughout my study.

I would also like to thank Dr. Richard Povinelli, Dr. Henry Medeiros and Dr. Iqbal Ahmed for serving on my committee, reviewing my dissertation and providing very helpful suggestions that enriched the quality of my dissertation.

My gratitude expands to Dr. Fabian Josse, Dr. Edwin Yaz, Dr. Majeed Hayat, Dr. James Richie, Dr. Susan Schneider, Mrs. Katie Tarara and all the members of the department of Electrical and Computer Engineering for their guidance and support during my study at Marquette University.

I would like to express my gratitude to all my friends at Marquette University, specially my colleagues at Marquette Embedded Systems Lab (MESSLab): Wenkai Guan, Nathan Zimmerman, Kellen Carey, Ian Barge, Shaun Duerr and Alim Ahsan for all their help and possitive feedback.

I would also like to thank Dr. Kiarash Bazargan, in the department of Electrical and Computer Engineering of University of Minnesota, whose support, guidance and recommendations opened a new window to my future.

Most importantly, I'm grateful beyond words to my mother and father and my brothers, Mehrad and Masoud, who are the most important people in my world and I cannot thank them enough for all their support, positive thoughts, patience and understanding in each moments of my life. I sincerely dedicate my dissertation to them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF NOMENCLATURE AND ACRONYMS

Acronym	Definition
ANN:	Artificial Neural Network.
CMP:	Chip Multiprocessor.
CPI:	Cycles per Instruction.
DBN:	Deep Belief Network.
DEM:	Dynamic Energy Management.
DIC:	Delayed Instructions Count.
DNN:	Deep Neural Network.
DRM:	Dynamic Reliability Management.
DVFS:	Dynamic Voltage and Frequency Scaling.
EDAP:	Energy Delay Area Product.
EDP:	Energy Delay Product.
KNN:	K-Nearest Neighbor.
LSTM:	Long Short Term Memory.
MC:	Monte Carlo.
MTTF:	Mean Time To Failure.
NBTI:	Negative Bias Temperature Instability.
NN:	Neural Network.
NoC:	Network-on-Chip.
PUE:	Power Usage Effectiveness.
RBM:	Restricted Bultzman Machine.
REST:	Reliability Estimation Tool.

RNN: Recurrent Neural Network.

ROI: Region of Interest.

SVM: Support Vector Machine.

TDDB: Time Dependent Dielectric Breakdown.

V/F: Voltage/Frequency.

WSC: Warehouse Scale Computer.

CHAPTER 1

Problem Statement, Objective and Contributions

This dissertation addresses new challenges in designing chip multiprocessors which include lifetime reliability and energy consumption. It proposes effective solutions based on novel ideas to address these challenges. In this chapter, the problem statement and the main objectives are described in Sec. 1.1 and Sec. 1.2 respectively. Sec. 1.3 briefly discusses related work in area of lifetime reliability and energy management for CMPs. The contributions to be drawn from this research are then presented in Sec. 1.4. This chapter concludes with an outline and organization of the remainder of the dissertation in Sec. 1.5.

1.1 Problem statement

Chip multiprocessors have become popular in most computing systems, including desktop computers, portable devices, servers and datacenters also called warehouse scale computers (WSCs). While adopting CMPs provides better computational capability, there are new challenges and concerns for the designers as well.

A primary challenge that the designers face in this context is lifetime reliability which worsens with technology downscaling. Two of the most adverse wearout or aging mechanisms in deep submicron technologies include time dependent dielectric breakdown (TDDB) and negative bias temperature instability (NBTI). The impact of these failure mechanisms has become increasingly adverse due to the increased power densities and system complexity. Faster aging leads

to earlier performance degradation with eventual device breakdown and thus system failure due to errors. The shift from singlecore to multicore processors has somewhat alleviated the issue of increasingly large power densities. However, this issue persists especially with the advent of chip multiprocessors that integrate tens and hundreds of cores[1] on the same chip; some cores must be shut off to keep power densities under control, thereby not utilizing fully the available computational power of chip multiprocessors. This is commonly referred to as dark silicon problem. Consequently, researchers from both industry and academia recognize that lifetime reliability is becoming a primary design concern and have been investigating methods to mitigate the negative impact of these aging effects in order to increase the mean-time-to-failure (MTTF) of the devices and circuits. Moreover, given that the most important factor through which these aging mechanisms affect chips is the temperature, it is the chip multiprocessors' lifetime reliability that is especially affected – because their operation temperatures have been increasing due to the increased power densities.

On the other hand, also a big concern in CMPs is energy consumption, which is desired to be minimized without affecting the achievable performance. This is increasingly important due to the advent and wide spread of mobile devices but also due to the increasingly large energy consumption footprint of datacenters. Thus, we are interested in reducing energy consumption in mobile devices in order to prolong battery life. Reducing energy consumption in datacenters reduces costs and can have a beneficial impact of the environment. For example, in 2013, U.S. datacenters consumed an estimated 91 billion kilowatt-hours of electricity, enough

[1] It is predicted that actually future CMPs will integrate thousands of cores.

to power twice the households in New York City. By 2020, estimated consumption will increase to 140 billion kilowatt-hours, costing American businesses $13 billion per year in electricity bills and causing the emission of nearly 150 million metric tons of carbon pollution annually [1]. According to the U.S. Energy Information Administration, that is about 7% of total commercial electric energy consumption and it is projected that this number will increase [2]. Therefore, improving energy efficiency is not only important for the cost to companies, but for the environmental footprint of these WSCs as this computing domain rapidly expands [3]. Reducing energy consumption in CMPs has the additional benefit of reducing power dissipation, which in turn lowers chip temperatures that have a beneficial impact on the lifetime reliability of these devices and systems.

1.2 Objectives

The main objective of this work is to provide more efficient techniques to improve the lifetime reliability and energy consumption of the future network-on-chip based chip multiprocessors in order to have more reliable and cost efficient servers, data centers and portable devices.

1.3 Related Work

In this section, the previous work is discussed and classified into two major categories. The techniques that focus on improving the reliability of the chip multiprocessors and the techniques developed to manage the energy consumption of

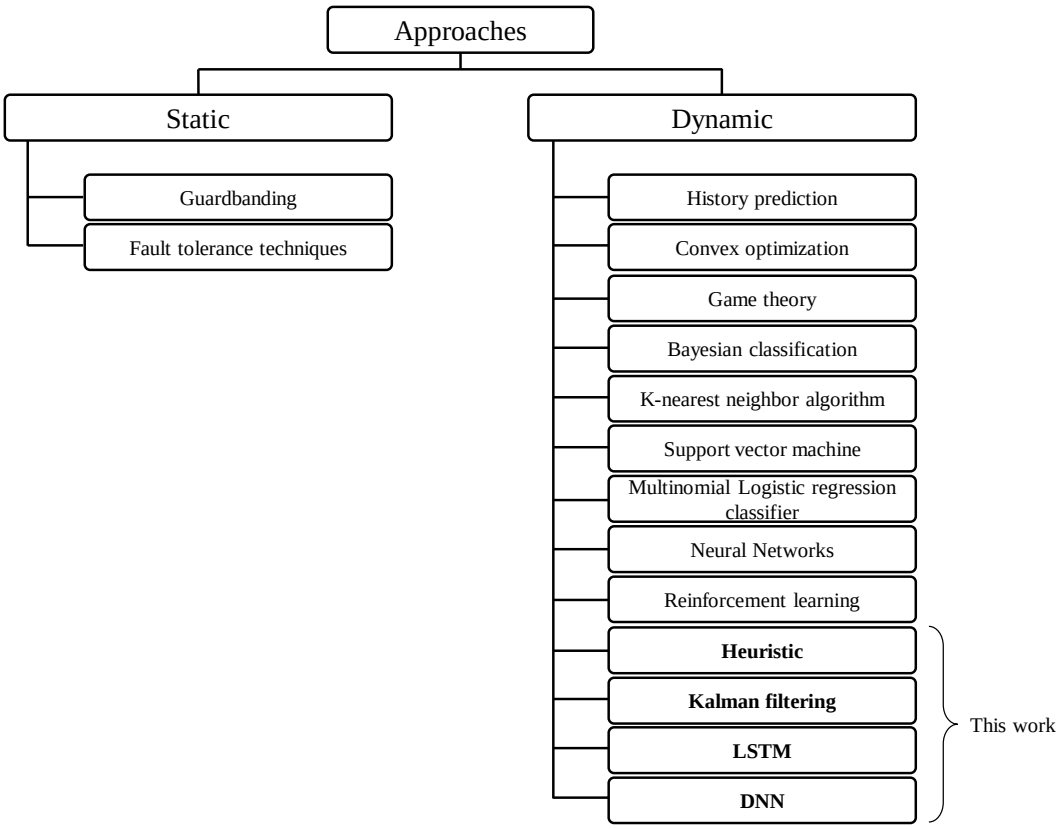


Figure 1.1: A taxonomy of approaches employed in energy and reliability management techniques.

the CMPs. Fig. 1.1 illustrates the main approaches and techniques that are employed in energy and reliability management in computation and communication components of a processor. These techniques are described next.

1.3.1 Reliability Management Techniques

Generally, we can classify reliability oriented design methods into two categories. The first category is that of *static approaches*, which address the problem of reliability at design time. Static design methods include guardbanding and fault tolerance techniques. For example, supply voltages are selected high enough to guarantee

correct functionality despite variation in threshold voltage or in temperature and supply noise. In this way energy gained from downscaling is sacrificed to combat reliability problems. However, if this sacrifice becomes too large, downscaling may become detrimental [4]. Fault tolerance techniques are based on fault detection and recovery mechanisms, which require energy and area overheads. Previous work employed fault tolerant techniques based on 1) error detection implemented through coarse grained replication or redundancy [5–7], 2) failure prediction used to take preventative measures to avoid, or at least mitigate the effects of device failures [8–10], and 3) error masking [11]. Simulated annealing is used to optimize both energy and reliability in [12]. A sequential quadratic programming based approach is proposed in [13] to maximize the lifetime of a multiprocessor system considering the electromigration effects in communication links. A wearout aware schedulability analysis technique is introduced in [14] for real-time independent tasks mapped to processor with dynamic voltage and frequency scaling capabilities. A convex optimization based approach is proposed in [15] to maximize the lifetime reliability of the cores of a multiprocessor system subject to electromigration wearout. The study in [16] uses genetic algorithms to identify voltages and frequencies of the cores of a multiprocessor system to maximize the lifetime and minimize the soft-error susceptibility. The main challenge of this category of methods is to reduce the energy and area overheads while reliability is still improved.

The second category of reliability oriented design methods is that of *dynamic approaches*. The main idea of this class of approaches is to dynamically

monitor the system during runtime and by using either *reactive* or *proactive* techniques to change the operation of the system such that reliability is improved. Note that these approaches may use support from the first category of static approaches discussed in the previous paragraph. A two phase dynamic reliability management algorithm to address various aging mechanisms is introduced in [17]. In the first phase, an application is profiled to find the maximum performance at which each hardware configuration can run while still maintaining the desired mean time to failure. In the second phase, the configuration with the highest performance and satisfying MTTF is selected for the remaining application's run. Dynamic reliability banking is proposed in [18] to address aging due to electromigration. Reliability slack is introduced in [19] and used for dynamic reliability management during periods of high processing demand. The authors of [20] exploit the natural variation in workloads to assign jobs to cores in a manner that minimizes the impact of NBTI and TDDB on lifetime reliability. The authors of [21] introduce Facelift, a technique to hide aging through aging-driven application scheduling and to slow it down by applying voltage changes at key times. A dynamic voltage and frequency scaling (DVFS) control and look-up table reliability estimation based DRM scheme is introduced in [22] for singlecore processors to address process variation aware oxide breakdown. The impact of job scheduling based power management on reliability is investigated in [23]. A dynamic tile partition algorithm is introduced in [24] to balance workload among active cores while relaxing stressed ones. A system level HW/SW reliability management scheme where a chip dynamically adjusts its own operating frequency and supply voltage over time as the devices age due to NBTI is introduced in [25]. The authors of [26]

study a control theoretic approach that uses data from aging sensors to compute the wearout degradation and to maximize the lifetime of homogeneous multicore systems. The same authors introduce a complete software implementation, working on a real mobile hardware platform, of a workload-aware dynamic reliability management technique to address TDDB wearout [27]. A reinforcement learning algorithm is proposed in [28] to optimize the lifetime of a multicore system by controlling the average temperature and thermal cycling. While the majority of previous work focus with their reliability oriented design methods only either on the computational portion of the system (i.e., singlecore or multicores) or on the communication component (i.e., buses or networks-on-chip), the authors of [29] concentrate on the combination of both. They use a neural network based reliability estimator and thread migration for dynamic reliability management of chip multiprocessors. The study in [30] introduces a wearout-decelerating scheme to mitigate the impact of NBTI and hot-carrier injection (HCI) in NoCs. Online adaptive aging-aware routing algorithm to avoid highly aged routers in NoCs was studied in [31]. The study in [32] presents a reliability management solution for dark silicon chips. The solution considers soft errors, process variations, and the thermal design power constraint. Simulation results were reported for 80x80 grid cells chips of LEON3 processors. This work is further extended in [33]. The same research group proposed in [34] a run-time approach that harnesses dark silicon to decelerate and balance temperature-dependent aging. Their solution also considered variability to improve the system performance for a given lifetime. They focus on NBTI and did not report if the communication among cores is via the NoC.

Furthermore, the study in [35] proposed a process variation- and aging-aware dynamic hierarchical mapping solution to maximize lifetime reliability of manycore systems while satisfying performance, power, and thermal constraints. The authors reported improved system lifetime reliability by up to 2 years for 64-core and 256-core systems. For discussion of additional reliability studies, we kindly refer the reader to the recent survey in [36].

1.3.2 Energy Management Techniques

Energy optimization in single and multicore processors received a lot of attention in the previous literature. The most popular techniques utilized by previous optimization solutions include DVFS, job scheduling and task migration. Among all these methods, DVFS has been the most effective one, since power consumption is related to the clock frequency and the square of the voltage supply, and energy consumption is the product of power consumption in time. Lowering only the clock frequency of a core helps to reduce the average power consumption for a given application while the total energy consumption remains the same to execute the application. Reducing the average power consumption in turn lowers the chip temperature, however, at the expense of longer execution times for the application. Lowering the supply voltage helps to reduce the total power consumption and this helps in turn to reduce the total energy consumption that is needed for the execution of a given application. DVFS changes both voltage and frequency dynamically and can be used to exploit both above benefits. However, it usually comes at the price of performance degradation due to frequency throttling. In the case of multicore processors, per-core DVFS is not yet widely supported (Intel Haswell-EP

and Samsung Exynos processors are said to support it). However, many recent studies have shown the benefits of per-core or per-cluser-of-cores DVFS capabilities [37–42]. Our work is under the assumption that such per-core DVFS may be possible in the future multicore processors and it is under this assumption that we implement and test the proposed ideas using the Sniper system simulator. All these techniques are used as primary control mechanisms to drive the operation of processors toward low energy consumption and such that performance is not significantly degraded. The control decisions are made based on estimations or predictions of the energy or other related variables in a reactive or proactive manner as part of the algorithm or strategy that implements the optimization solution. System monitoring and decision making are usually done periodically, at intervals called control periods or epochs. It is the prediction mechanism that differentiates the impact of a given energy optimization solution.

Previous work has employed a variety of methods including machine learning [43–45], game theory [46], and convex optimization [47] to find the optimal voltage-frequency pairs to manage the energy consumption of homogeneous (i.e., formed by identical cores) processors. More recently, heterogeneous processors are exploited towards additional optimization opportunities [48–50]. For example, the study in [49] proposed a joint temperature and energy management solution for heterogeneous multicore processors. Their heuristic uses both DVFS and temperature- and performance-aware task assignment strategy that maximizes the energy savings, while maintaining the temperature at safe levels.

Some of the previous studies focused on developing performance estimation techniques for DVFS enabled devices and then used them as heuristics inside the control mechanism to find the best voltage-frequency pairs to achieve energy optimization, considering the performance constraints. For example, the study in [51] presented a DVFS that automatically adapted the voltage and frequency for energy savings at runtime in high performance computing clusters formed by four Athlon64-based compute nodes connected via Gigabit Ethernet and another four-node quad-CPU cluster based on the Celestica A8440 server. Similarly, the study in [52] presented a performance-prediction model that is used by a per-CPU DVFS algorithm that makes DVFS decisions based on the index of CPU intensiveness. They verified the algorithm in a 9-node power-aware cluster formed by dual core processors. Other DVFS algorithms applied at the cluster of CPU nodes level include [53, 54]. Some recent work also took a more holistic approach and applied DVFS to both CPU and the DRAM subsystem to achieve additional energy savings [55, 56]. They reported for a server platform with an Intel i5-4590 quad-core processor and 8 GB of main memory as much as 22% energy savings with a low performance loss of only 4.8%. While the above previous performance-aware studies focused mainly on the cores inside a CMP, recent studies focused also on the interconnects and the shared last level caches (collectively called the uncore) to estimate the performance of the CMP and use that in DVFS based energy optimization algorithms. For example, the study in [57] uses the number of cache misses while the study in [58] uses the number of non-speculative reads that result in last-level cache misses (called leading loads), and the study in [59] extends that

for variable memory access latencies. Similarly, the authors in [60] take into consideration the off-chip (L2) I-cache misses and off-chip (L2) D-cache load misses in their estimation processes. The study in [61] proposed a DVFS policy for the uncore. The policy uses a technique similar to the TCP Vegas congestion control and was shown to result in significant energy savings.

Developing DVFS algorithm based on the future predicted workload has been another strategy for energy reduction in some studies. Previous work employed different types of predictors to periodically predict the next control period workload for the device and then select a lower voltage/frequency (V/F) pair, if the workload is lower than a threshold, to assure reducing energy consumption while not exceeding the performance limitations. One of the simplest prediction techniques is history prediction. Such predictions can be used to trigger frequency throttling early on in upstream NoC routers in order to lower the rate at which data is sent to downstream routers [62,63]. The study in [64] proposes a multinomial logistic regression-based classification technique, that classifies the workload at runtime, into a fixed set of classes, which are then utilized to design a DVFS algorithm. In [65], a multinominal logistic regression classifier is built using a large volume of performance counters for offline workload characterization. This classifier is queried at run-time for a given application to predict the workload, and then selection of the frequency and thread packing are done to maximize performance under a given power budget. The techniques in [43,66,67] use online learning to select the most appropriate frequency for the processing cores based on the workload characteristic of a given application. Another approach for the data classification or regression problems is the k-nearest neighbor algorithm (KNN) [68]. The study

in [69] uses supervised learning in the form of a Bayesian classifier for processor energy management. This framework learns to predict the system performance from the occupancy state of the global service queue. The predicted performance is then used to select the frequency from a pre-computed policy table. Reinforcement learning (RL) based optimization algorithms are proposed in [28, 70–72]. For example, the study in [28] used RL to learn the relationship between the mapping of threads to cores, clock frequencies, and temperatures, and employed that mapping information to develop better task mapping and DVFS solutions. The work in [70] used RL to learn the optimal control policy of the V/F levels in manycores and then exploited that to develop an efficient global power budget reallocation algorithm. The authors of [72] proposed an online DVFS control strategy based on core-level modular reinforcement learning to adaptively select appropriate operating frequencies for each individual core. Q-learning was used by the work in [73] to develop an algorithm that identifies V/F pairs for predicted workloads and given application performance requirements. In the context of dynamic VFI control in manycore systems with different applications running concurrently, the study in [74] investigated imitation learning and reported higher quality policies.

The authors of [75] develop an artificial neural network (ANN) based mechanism for network-on-chip power management. The offline training of the ANN is augmented with a simple proportional integral (PI) controller as a second classifier. The ANN is used to predict the NoC utility, which is then used to make DVFS decisions that lead to improvements in the energy-delay product. A neural network (NN) based model with eight outputs for different interface configurations of a mobile device was presented in [68] to do classification. Such classification

is used as the basis for setting the mobile device into the configuration state that reduces energy consumption. It was reported that NN and support vector machine (SVM) models provided the best prediction accuracy. The study in [76] proposed a deep neural network (DNN) model to model plant performance and to predict power usage effectiveness (PUE) in datacenters. Testing and validation at Google's datacenters showed that the DNN model can be an effective approach to exploit existing sensor data to model datacenter performance and to identify operational parameters that improve energy efficiency and reduce the PUE [76].

1.4 Contributions

The main contributions of this dissertation is to provide novel and efficient techniques for dynamic reliability and energy management that are described here in more details.

1.4.1 Dynamic Reliability Management

The majority of the previous work suffers from two limiting characteristics. On one hand, previous studies focus separately on either the *computational* component of a processor (i.e., single core or multicore) or the *communication* component, typically the network-on-chip. Not considering either of these components introduces significant errors, because both computational and communication units of multicore processors may become a reliability bottleneck. Such errors can mislead any lifetime reliability optimization method and result into suboptimal solutions. To address this issue, in [29, 83], we considered the study of CMPs in a unified

manner, as the combination of both computational and communication units. We found that when we do not consider for example the NoC unit in the reliability optimization, the errors in MTTF values, as the most popular way to measure lifetime reliability, may be off by as much as 60%. On the other hand, usually only one technique, such as DVFS, scheduling or thread migration, is used as the main optimization technique. Indeed, employing only one technique can miss further optimization opportunities that can be offered by hybrid approaches that typically combine multiple cross-layer techniques, to construct algorithms that are more versatile and applicable to a wider variety of benchmark applications and workloads. It is our intent with this dissertation to address this issue. Specifically, we propose and study a hybrid dynamic lifetime reliability management algorithm for CMPs that combines thread migration and dynamic voltage and frequency scaling techniques. The proposed algorithm uses a simple yet effective approach in order to seamlessly use the two techniques to adaptively change the CMP operation such that the lifetime reliability of the overall system is increased to the desired target with minimal performance degradation.

1.4.2 Dynamic Energy Management

The majority of the previous approaches rely on the performance estimation and workload prediction/classification based techniques. The effectiveness of all these methods is significantly affected by the estimation, prediction or classification accuracy. In most cases, complex relationships exist across long histories of processor usage that may not be detected by current techniques, which increases the number of mispredictions when using current prediction based solutions. While there

has been significant work, it is not clear how far the existing DVFS based energy optimization techniques are from the optimal solutions. We believe there is still room for improvement, and generally, we see this as the main limitation of previous work. As such, our main motivation is to investigate whether DNN models can be of any help in pushing the frontier of energy optimization in chip multiprocessors. This idea in turn is motivated by the immense success that DNN models have had in the last decade in many application domains including speech and pattern recognition, image processing, and datacenter operation.

First of all, we present a new heuristic algorithm for dynamic energy management of chip multi-processors that proposes a performance estimation based technique called delayed instruction count (DIC) which increases the estimation accuracy by eliminating the issues related to counting the misses and stalls. It also employs the DVFS technique to identify the best V/F pairs for all cores of the CMP. This is done using accurate and efficient estimations of the average cycles per instruction and the instruction count, which are done using a Kalman filtering as well as long short-term memory (LSTM) techniques. Then, we propose DNN models and develop related self-adaptive supervised learning methods to identify optimal V/F pairs in chip multiprocessors. Since the supervised learning needs labeled data for the training process, the Kalman filtering based DEM that we developed earlier is used to collect workload characteristics and their corresponding corrected V/F pairs as the labels for the training data. Using DNN models offers the advantages of being able to handle heterogeneity and to capture deep and complex relationships across long histories of processor usage. We see deep

learning techniques, such as the one we propose in this dissertation, as a new enabler in pushing the frontier of energy optimization in computing systems because they have the ability to model and predict complex behavior and relations between workload and hardware that otherwise currently is not possible.

1.5 Dissertation Organization

Chapter 2 presents background information on system failure mechanisms and then describes a technique to estimate the reliability of a network-on-chip based chip multiprocessor. In addition, the prediction/classification methods including Kalman filtering, long short term memory and deep neural network that we later use in the proposed dynamic energy management techniques are discussed next.

Chapter 3 describes the proposed techniques to dynamically manage the reliability of the network-on-chip based chip multiprocessors. These techniques employ DVFS and thread migration wisely to achieve reliability improvement without imposing significant penalty on the performance. The effectiveness of these techniques are then evaluated on various benchmarks via full-system simulations.

The proposed dynamic energy management techniques for CMPs are discussed in Chapter 4. The discussion starts with a mechanism to predict the performance loss when using various V/F pairs instead of executing with the maximum V/F pair all the time and then proposes effective techniques with the help of Kalman filtering, LSTM and DNN to find energy saving opportunities during the execution time without degrading performance beyond the user set threshold. The simulation results for each technique on different benchmarks are then presented

in the rest of the chapter.

Finally, Chapter 5 overviews the findings of this dissertation and discusses future work.

1.6 Related Publications

The topics discussed in this dissertation have been published in several conference and journal papers as follows:

[77] M.G. Moghaddam and C. Ababei, "Dynamic lifetime reliability management for chip multiprocessors," *IEEE Trans. on Multiscale Computing Systems*, 2018.

[78] M.G Moghaddam, W. Guan and C. Ababei, "Dynamic energy minimization in chip multiprocessors using deep neural networks," *IEEE Trans. on Multiscale Computing Systems*, 2018.

[79] C. Ababei, and M.G. Moghaddam, "A Survey of Prediction and Classification Techniques in Multicore Processor Systems," *IEEE Trans. on Parallel and Distributed Systems*, 2018.

[80] M.G. Moghaddam, C. Ababei, "Dynamic energy management for chip multiprocessors under performance constraints", *Microprocessors and Microsystems*, vol. 54, pp. 1-13, Oct. 2017.

[81] M.G Moghaddam, W. Guan and C. Ababei, "Investigation of LSTM based prediction for dynamic energy management in chip multiprocessors,"*IEEE Int. Green and Sustainable Computing Conference (IGSC)*, 2017.

[82] M.G Moghaddam, "Dynamic energy and reliability management for NoC-based chip multiprocessors, " *IEEE Int. Green and Sustainable Computing Conference (IGSC)*, 2017.

[83] M.G. Moghaddam, A. Yamamoto, and C. Ababei, "Investigation of DVFS based dynamic reliability management for chip multiprocessors," *IEEE Int. Conference on High Performance Computing & Simulation (HPCS)*, 2015.

CHAPTER 2

Background

In this chapter, we discuss the foundational concepts and materials used later in this dissertation. In Sec. 2.1, we talk about the main mechanisms that lead to device breakdown and system failure. Sec. 2.2 explains the reliability estimation technique that we employ in our work. An effective prediction technique using the Kalman filtering is described in Sec. 2.3. In addition, in Sec. 2.4, neural networks related concepts including feedforward neural network, recurrent neural network (RNN), long short term memory and deep neural network are described that we use later for accurate and fast prediction/classification purposes.

2.1 Aging Mechanisms

Aging mechanisms including time dependent dielectric breakdown and negative bias temperature instability are among the most increasingly adverse factors that can lead to delay errors and device breakdowns. In this section, we briefly describe these mechanisms.

2.1.1 Time Dependant Dielectric Breakdown

The growth in demand for faster devices has lead to technology downscaling in CMOS transistors. As a drawback, over the last years, the thickness of the gate oxide in CMOS transistors has been decreased dramatically, consequently having smaller threshold to electric field, which can cause dielectric breakdown in transistors. The reason is that the oxide can no longer properly insulate the gate terminal

and causes the charges to tunnel through in it and eventually become trapped in it, as shown in Fig. 2.1. The number of trapped charges in the oxide increases in time and after getting to a certain amount, the oxide breaks down and conducts the current from gate to substrate. Transferring current heats up the oxide and causes it to conduct even more current and consequently this feedback loop will eventually destroy the dielectric. Thus, due to having this characteristics, a device can suffer various soft breakdowns or a hard breakdown before the final hard breakdown occurs [84].

So, by the increase in temperature, the tunneling current will increase and consequently, the number of the trapped charges will increase as well. Therefore, it can be deduced that as the device gets hotter, the MTTF of a device decreases due to the TDDB mechanism.

As described in [17], the $MTTF_{TDDB}$ can be modeled by the following expression:

$$MTTF_{TDDB} \propto ((\frac{1}{V})^{a-bT} \times e^{\frac{X+\frac{Y}{T}+ZT}{KT}}) \tag{2.1}$$

where V is the Voltage supply, T is the temperature in Kelvin, k is the Boltzmann's constant and a,b,X,Y, Z are model parameters that are determined experimentally. These values are set in this model as $a = 78$, $b = -0.81$, $X = 0,759eV$, $Y = -66.8eV$, and $Z = -8.37e^{-4}eV$ based on the experimental data from [85].

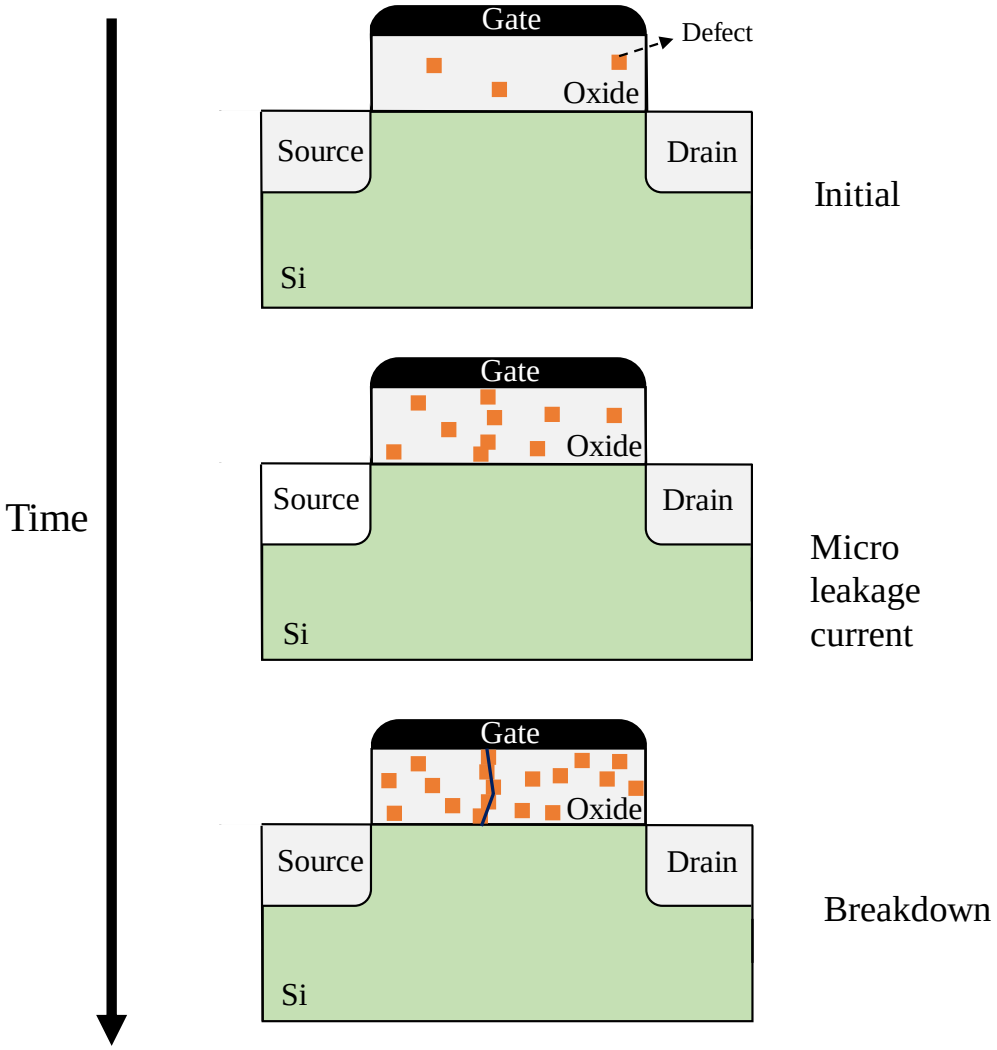


Figure 2.1: Time dependent dielectric breakdown process.

2.1.2 Negative bias temperature instability

When negative voltages have applied to the gate in MOSFET devices (especially in PMOS), it gradually degrades the performance in time. That is due to increasing the threshold voltage of the transistor, degrading the drain current and consequently degrading the speed [86, 87]. Experimental analysis shows that this problem, which is known as negative bias temperature instability, increases exponentially with rise in temperature [88, 89]. In other words, as described in [90],

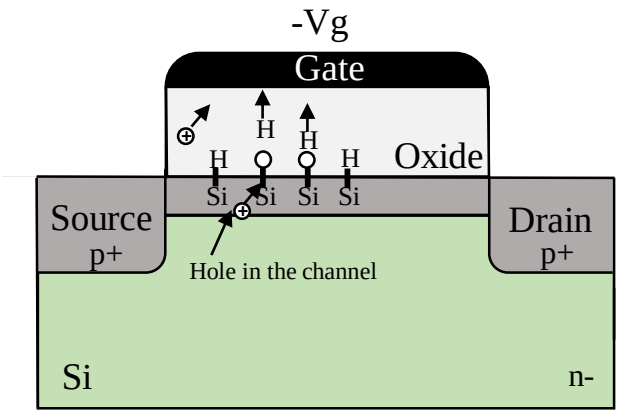


Figure 2.2: Negative bias temperature instability process.

this mechanism is characterized by a positive shift in the absolute value of the PMOS, which occurs when a device is biased in strong inversion, but with a small, or no lateral electric field. The shift is generally attributed to hole trapping in the dielectric bulk, and/or to the breaking of Si-H bonds at the gate dielectric interface by holes in the inversion layer, which generates positively charged interface traps as shown in Fig. 2.2.

MTTF due to the NBTI mechanism is modeled as shown in 2.2, where $A = 1.6328$, $B = 0.07377$, $C = 0.01$, $D = 0.06852$, $\beta = 0.3$ and T is the temperature in Kelvin as in [29], and K is Bultzman's cost.

$$MTTF_{NBTI} \propto ([ln(\frac{A}{1+2e^{\frac{B}{kT}}}) - ln(\frac{A}{1+2e^{\frac{B}{kT}}} - C)]\frac{T}{e^{\frac{D}{kT}}})^{\frac{1}{\beta}} \tag{2.2}$$

2.2 Reliability Estimation Tool

According to Sec. 2.1.1 and 2.1.2, the temperature of the silicon has a significant impact on the MTTF of the system. Since in a CMP, the temperature of the components changes based on the assigned workload, we need a mechanism to estimate the MTTF of a CMP with the given components' temperature. In this section we briefly introduce the lifetime reliability estimation tool (REST) described in [29], which we use it later in section 3.

The REST tool is based on a Monte Carlo (MC) algorithm that works with failure times for TDDB and NBTI aging mechanisms modeled as Weibull distributions. What distinguishes Rest from the previous work is that both the computational and communication components of the studied chip multiprocessor system are considered in a unified manner to compute the lifetime reliability (as MTTF) of the CMP. The flow chart of the MC algorithm is shown in Fig. 2.3.

The input to the REST tool is a list of temperatures for all routers and blocks of each processor core as computed by HotSpot [105]. These temperatures are utilized during each iteration of the MC algorithm to generate samples (or instances) from the probability distributions associated with each router and core block forming the CMP system. The generation of these samples is based on equations in described in Sec. 2.1.1 and 2.1.2 that model the mean time to failure for each type of wearout or aging mechanism and which have been derived by the materials science and reliability engineering communities.

The pseudocode description of the MC algorithm is shown in Fig. 2.4. It

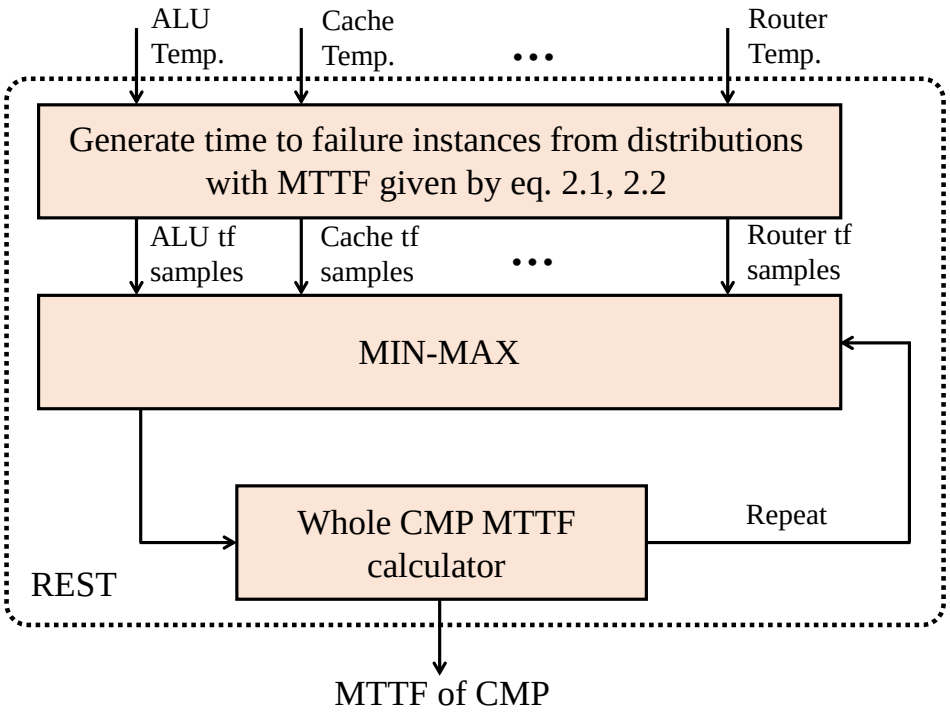


Figure 2.3: Block diagram of the Monte Carlo simulation based time to failure evaluation methodology for CMPs.

consists of the following steps 1) for each failure mechanism run $N = 10^5$ simulations: (a) for each block, generate failure time instances from the corresponding distribution, and (b) use MIN-MAX analysis of these times according to the system's configuration to calculate the time to failure tf_{min}^{j} during simulation iteration $j = 1, ..., N$. 2) calculate the time to failure for the current failure mechanism as $tf_l = (\sum_{j=1}^{N} tf_{min}^{j})/N$. 3) calculate the value of the overall MTTF or time to failure of the CMP as the minimum among the failure times due to each failure mechanism. We selected $N = 10^5$ because in our experiments we found that this number is a good tradeoff between computational runtime and statistical significance of results.

During each MC simulation iteration, we need to generate random instances of failure times for each subblock. This is realized by the $generate\_instance(MTTF_l)$ procedure called in line number 8 in Fig 2.4, which draws samples from Weibull distributions whose means are given by equations (2.3) and (2.4). Because the Weibull cumulative distribution function is given by:

$$F(x) = 1 - e^{-(\frac{x}{\alpha})^{\beta}} \tag{2.3}$$

$$x_{sample} = \alpha.[-ln(1-u)]^{\frac{1}{\beta}} \tag{2.4}$$

where $u = rand(0, 1)$ is a random number generated uniformly from the interval $[0,\ 1]$. α and β are the scale and the shape factors characterizing the Weibull distribution. In our implementation of $generate\_instance(MTTF_l)$, we utilize a value of $\beta = 1.64$ as in [91] while α is derived from the expression of the mean of the Weibull distribution:

$$\alpha = \frac{MTTF_l}{\Gamma(1 + \frac{1}{\beta})} \tag{2.5}$$

where $\Gamma(.)$ is the Gamma function.

Algorithm: Monte Carlo algorithm of REST tool

1: **In:** CMP floorplan and temperature of all blocks
2: **Out:** Estimate of MTTF of whole CMP
3: **for** $l \leftarrow 1$ to F **do** // F: number of failure wearout types
4: Calculate $MTTF_l$ using equations that model wearout mechanisms
5: **for** $j \leftarrow 1$ to N **do** // $N = 10^5$ Monte Carlo iterations
6: $tf^j_{min} \leftarrow INF$ // Initialize
7: **for** $k \leftarrow 1$ to S **do** // S: number of blocks
8: $tf_k \leftarrow generate\_instance(MTTF_l)$
9: **if** $tf_k < tf^j_{min}$ **then** // Generalization: MIN_MAX
10: $tf^j_{min} = tf_k$
11: **end if**
12: **end for**
13: **end for**
14: $tf_l = \frac{\sum_j^N tf^j_{min}}{N}$
15: **end for**
16: return $tf = MIN\{tf_l\}$ // Estimate of MTTF of whole CMP

Figure 2.4: Pseudocode description of the Monte Carlo simulation implemented by the reliability estimation, REST tool.

2.3 Kalman Filtering

In this section, we present a description of Kalman filtering, which we use later in this work as an estimation technique to estimate the future workload on the cores of the CMP. The Kalman filter uses a set of recursive equations and employs a feedback control mechanism in a way that minimizes the variance of the estimation error [92]. It is an adaptive filter applied to predict the state x of a discrete-time controlled process. Using the notation from [93], the process can be described by the following state and output equations.

$$x_n = Ax_{n-1} + Bu_{n-1} + w_{n-1} \tag{2.6}$$

$$z_n = Hx_n + v_n \tag{2.7}$$

where A, B, and H are matrices. A is the state transition model applied to the previous state x_{n-1}. It relates the states at time steps $n-1$ and n, in the absence of process noise or control input. B relates the optional control input u to the state x, and the matrix H relates the state x to the measurement or observation z. The random variable w_{n-1} models the process noise assumed to be a white Gaussian noise with zero mean and covariance Q, $w \sim N(0, Q)$. Similarly, the random variable v_n is the measurement noise also assumed to have a Gaussian distribution with zero mean and covariance R, that is independent from Q, $v \sim N(0, R)$.

Then, we define the *a priori* and *a posteriori* estimate errors as $e_{\bar{n}} = x_n - \hat{x}_n^-$ and $e_n = x_n - \hat{x}_n$, where $\hat{x}_n^-$ is our a priori state estimate given the knowledge on the process prior to step n and $\hat{x}_n$ is our a posteriori state estimate after measurement z_n has been made. Based upon these estimates, the a priori and a posteriori estimate error covariances are given by the following expressions:

$$P_n^- = E[e_n^- e_n^{-T}] \tag{2.8}$$

$$P_n = E[e_n e_n^T] \tag{2.9}$$

To estimate the states of a process with measurements, the Kalman filter employs a feedback control technique in which the state at some time is estimated first and feedback is then provided in the form of noisy measurements. Thus, a Kalman filter is constructed in two phases. The first phase is called the *predict phase* (also called the time update phase), and here the state x is predicted *a priori*

as $\hat{x}_n^-$. The second phase is called the *update phase* (also called the measurement update phase). This is where the predicted $\hat{x}_n^-$ is updated *a posteriori* as $\hat{x}_n$.

In the predict phase, the filter first projects the state ahead from the previous state $\hat{x}_{n-1}$ and certain input matrix Bu_{n-1}. The filter then projects the error covariance ahead with process noise covariance Q. The two equations that accomplish that are:

$$\hat{x}_n^- = A\hat{x}_{n-1} + Bu_{n-1} \tag{2.10}$$

$$P_n^- = AP_{n-1}A^T + Q \tag{2.11}$$

Where P_n^- and P_n represent the estimated error covariance for a priori and a posteriori errors, respectively, at time n. They are calculated as shown in equations (2.8) and (2.9).

The update phase starts right after the predict phase with the measurement of the actual state value at time n. The three equations utilized in this phase are:

$$K_n = P_n^- H^T (HP_n^- H^T + R)^{-1} \tag{2.12}$$

$$\hat{x}_n = \hat{x}_n^- + K_n(z_n - H\hat{x}_n^-) \tag{2.13}$$

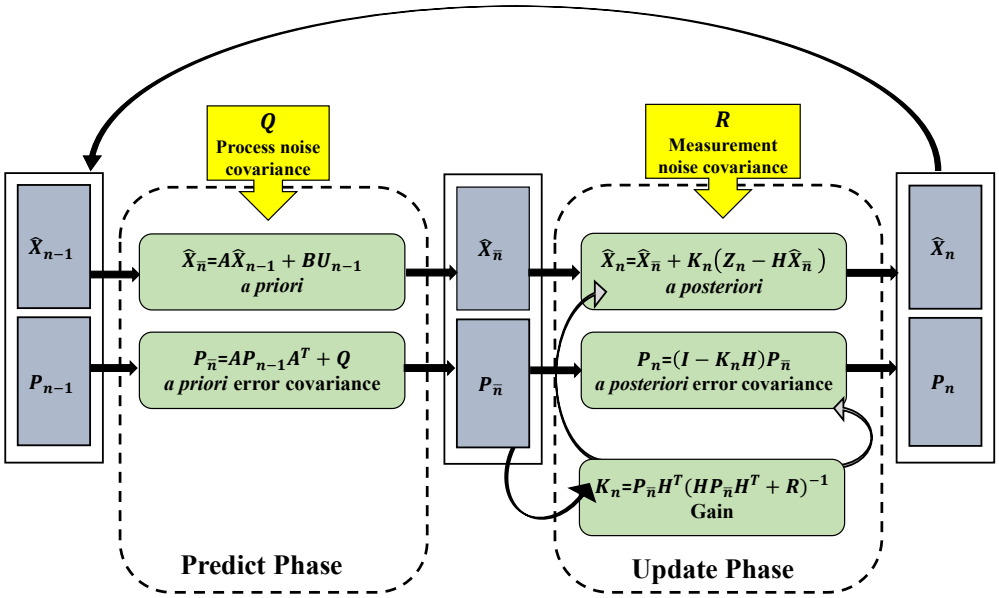


Figure 2.5: Kalman filter predict phase and update phase procedure.

$$P_n = (1 - K_n H)P_n^- \tag{2.14}$$

The Kalman gain, K_n, is first computed by using the a priori estimate error covariance P_n^- and measurement noise covariance R. It is chosen to maximize the a posteriori error covariance P_n. The filter then updates the current state vector $\hat{x}_n$ and a posteriori estimate error covariance P_n, using the Kalman gain. Fig. 2.5 summarizes how Kalman filter works.

2.4 Neural Networks

Among different machine learning models, the NN model is one of the most popular ones. The idea behind NN is to model the human brain architecture to mimic the learning process of the brain, but on computers. The human brain is modeled

as a network of millions of neurons connected to each other. The input signals provided by the body sensors are given to some of the neurons. These neurons process the signals and then pass their decisions to other connected neurons. It is assumed that final decisions are made by the last connected neurons. The NN model simplifies this process by proposing an architecture composed of connected layers of nodes and transfer functions as neurons. The nodes communicate with each other through weighted signals whose weights are adjusted via a repetitive computational process called learning.

2.4.1 Feed-Forward Neural Networks

The simplest and most popular NN architecture is the feed-forward neural network, which is illustrated in Fig. 2.6. The information in this network is transferred from one layer to the next in the forward direction only and no cyclic connections exist between layers. Each node represents a neuron that receives its weighted inputs from the nodes on the previous layer and calculates the output (i.e., decision) that is passed to the next layer. The transfer function of the node sums together all the decisions from the nodes in the previous layer and adds them to a bias value. The result then is passed through an activation function to generate the output. This process takes place in the forward direction through all layers up to the output layer, which produces the final output decisions. The values of the weights and biases are crucial as they affect the accuracy of the final decision. These values are determined during the training process of the network.

In supervised training, for a set of known features and labels (i.e., inputs

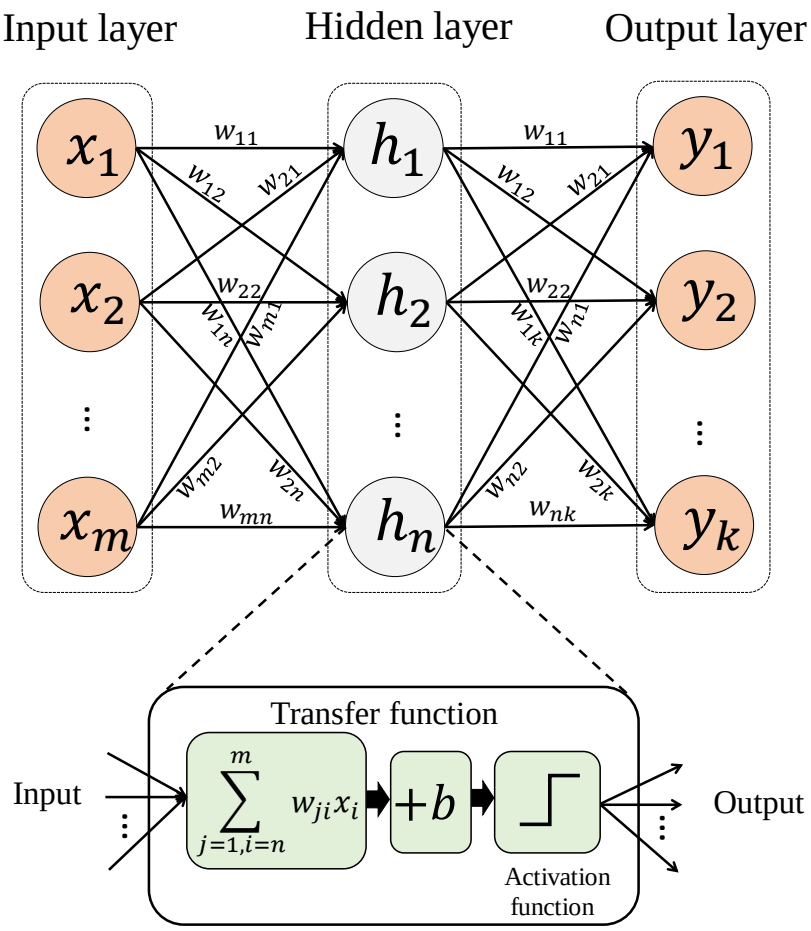


Figure 2.6: Typical neural network architecture.

and their corresponding output decisions), the final decisions produced by the NN model are compared to the labels by means of a cost function. Then, an optimizer is employed to minimize the generated cost by updating the weights through the network going in the backward direction as a backpropagation process. Usually, the optimizer uses a gradient descent optimization approach [94]. The training process is repeated on different sets of features and labels, thereby determining the optimized weights and biases. Once trained, the NN model can be utilized to provide estimations on new data of interest. That is, the outputs of the final layer can be used directly for classification purposes.

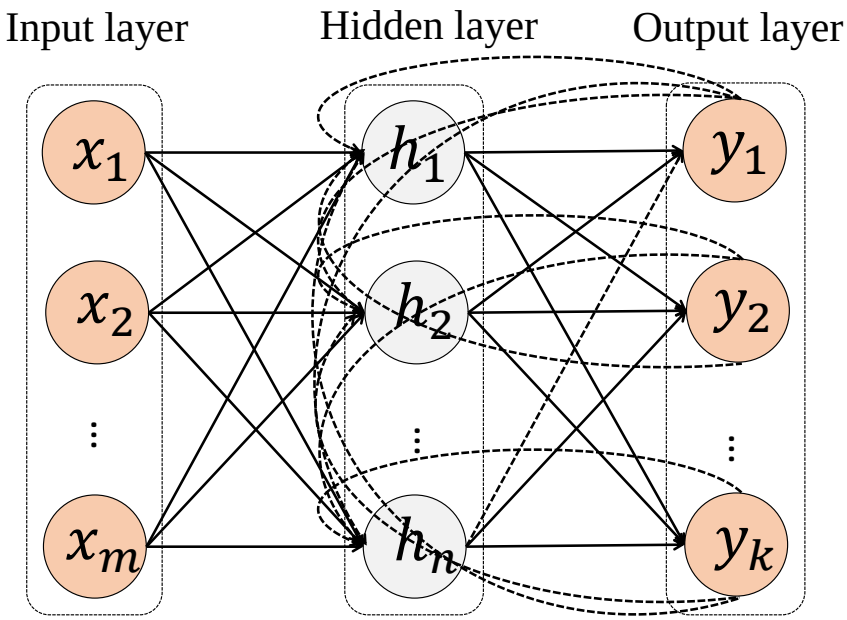


Figure 2.7: Simplified diagram of a recurrent neural network.

2.4.2 Recurrent Neural Networks

Because feedforward NNs do not have any cycles or loops, their temporal modeling capability is rather limited. Therefore, in situations where the prediction of the output must depend on long histories of the input feature sequence, the recurrent neural networks can represent a better model. The RNN model includes cyclic connections between different layers as illustrated in the simplified diagram from Fig. 2.7. The challenge that the RNN model faces though is that it can be difficult to train standard RNNs to solve problems that require learning long-term temporal dependencies. This is because the gradient of the loss function decays exponentially with time; this is known as the vanishing gradient problem.

2.4.3 Long Short Term Memory

To address the issues that standard RNN models face, the long short-term memory was proposed [96]. The LSTM network is an RNN that uses special units in addition to the standard units. LSTM units include *memory cells* that can store information for long periods of time in addition to special units called gates that control the flow of information. In other words, these gates are used to determine what to store as well as when to allow reads, writes and erasures of information into/from cells.

Fig. 2.8 shows a simplified diagrams of the three different cells used by feedforward NNs, RNNs, and LSTM networks. It can be observed that the LSTM cell is more complex. The added complexity is due to the input, forget and output gates that decide whether to let new inputs in, erase the present cell state, and let the state impact the output at a given time step. These gates are activated through weighted signals connected to an activation function. These weighted signals are adjusted during the learning process. That is, the cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, backpropagation of errors, and adjustment of weights via the gradient descent technique [97].

2.4.4 Deep Neural Networks

Structurally, a DNN model is just a feed-forward neural network with many hidden layers [98], as illustrated in Fig. 2.9. The main difference compared to traditional NNs is that DNNs have more hidden layers. That helps DNNs to capture more

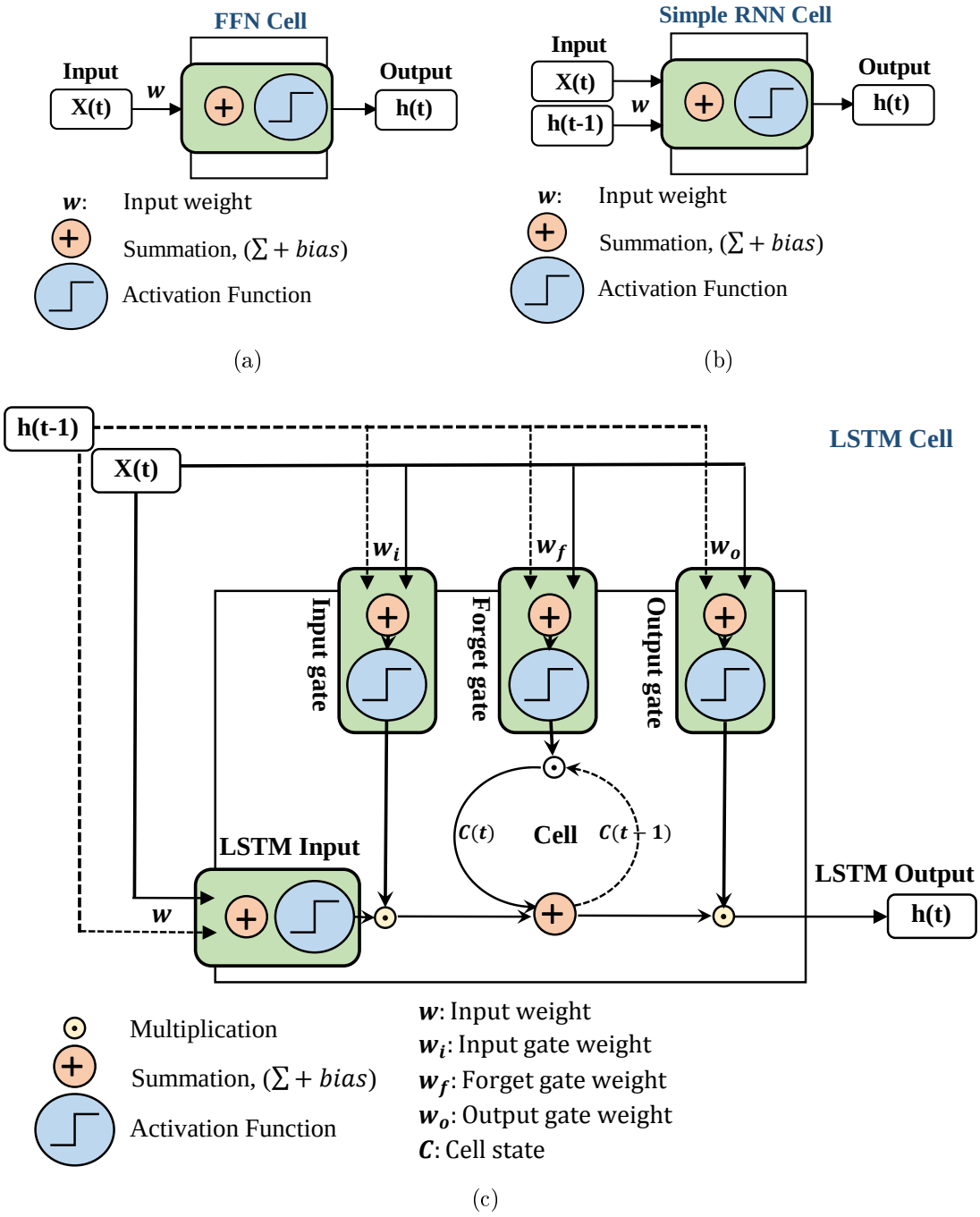


Figure 2.8: Simplified diagrams of three different cells (a) feedforward NN cell, (b) RNN ell, and (c) LSTM cell.

complex nonlinear relationships [99].

An important development in the world of machine learning was when Hinton and colleagues [100, 101] showed that deep belief networks (DBNs) can serve

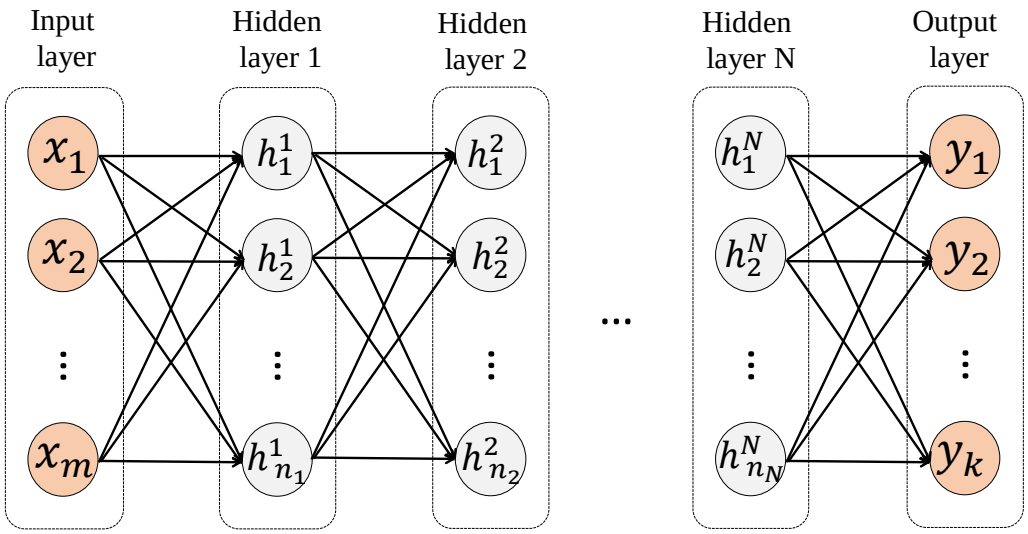


Figure 2.9: A deep neural network is a neural network with many hidden layers.

as the basis for DNN pretraining. They showed that one can effectively pretrain a DNN one layer at a time. That can be done by handling individual layers as unsupervised restricted Boltzmann machines (RBM) separately. Then, the entire stack of layers can be fine-tuned using supervised backpropagation. Moreover, the pretraining can also be followed by other discriminative learning techniques to further fine-tune the weights. During this process, a final layer is added to the DNN [102]. The variables on this final layer are the desired outputs from the training data. These outputs of the final layer can be used directly for classification purposes.

CHAPTER 3

Proposed Dynamic Reliability Management

3.1 Introduction

The idea of dynamic reliability management is to continuously monitor the CMP system and then periodically make decisions to *update* or *tune* different control knobs with the goal of shifting the system's operation to a mode where lifetime reliability is as close as possible to a desired value that is usually set by the user. Such a target lifetime reliability is usually reached after several *control periods* because of the inertia or delay it takes for different portions of the CMP chip to heat-up or cool-off. The challenging aspect of any DRM scheme is to achieve the above goal with minimal performance penalty and hardware overheads. In this chapter, we propose effective dynamic reliability management algorithms that employ dynamic voltage and frequency scaling and thread migration.

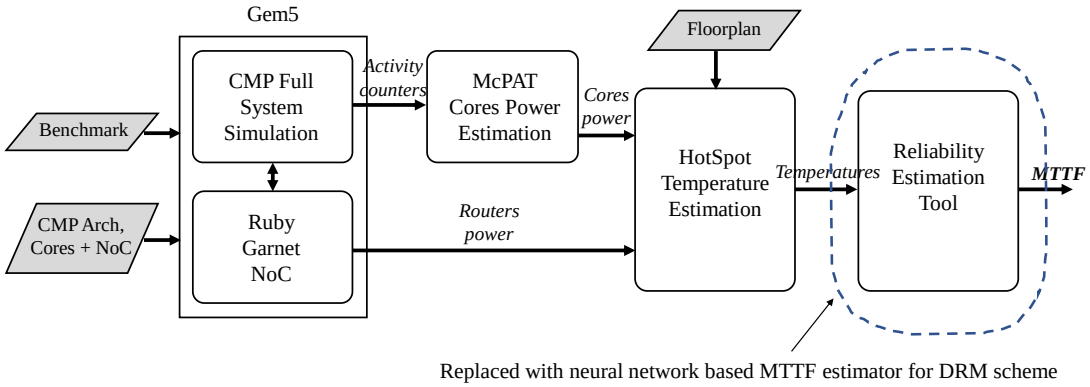
3.2 DVFS based Technique

In this section, our objective is to investigate the use of DVFS as a control knob to dynamically control lifetime reliability of CMPs seen as the unified combination of both cores and networks-on-chip.

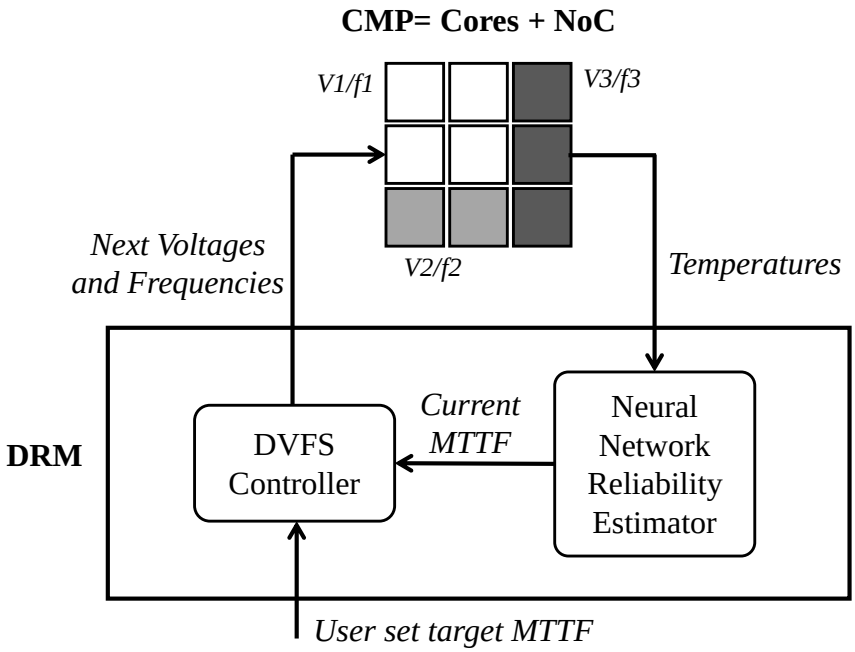
There are two important aspects regarding the construction of the DRM scheme that need to be emphasized. First, in order to be able to use it in real time,

the DRM scheme must be very efficient such that its runtime overhead is very small and therefore performance is not significantly affected by the time it takes 1) to estimate current lifetime reliability and 2) to make decisions to update voltages and frequencies. To estimate reliability *statically*, we adopt the lifetime reliability estimation approach proposed in [29] because, as illustrated in Fig. 3.1.a, it treats the CMP system as the combination of both cores and network-on-chip. In other words, this approach does not rule out major components that can become lifetime reliability bottlenecks, thereby minimizing estimation errors of MTTF of the whole CMP system[1]. According to the reliability estimation approach, illustrated in Fig. 3.1.a, Gem5 full system simulator provides the core activity counters as well as the router powers. These activity counters are then used by McPAT power calculator [104] to calculate the cores' powers. HotSpot temperature calculator [105] takes the cores' powers as well as the routers' powers and provides the temperatures. Having the temperatures and the CMP floorplan, REST lifetime reliability calculator tool [106] estimates the MTTF of the CMP. Obviously, using a reliability estimation approach as illustrated in Fig. 3.1.a *dynamically* is not practical due to the rather long computational runtimes of the components in the flow. But it is good to be noted that in real systems, the temperatures would be directly collected by sensors placed on the CMP chip. Thus, the only bottleneck in the flow is the REST tool which similar to [29]. Hence, we replace it with a neural network based *estimator* as shown in Fig. 3.1.b. The NN based lifetime reliability estimator is very efficient because it translates to only the evaluation of a function that takes as input the cores' temperatures (as indicated in Fig. 3.1.a) as well as specific weights that are

[1]Note that the vast majority of previous work focused on either cores as the computational component or on network-on-chip or bus as the communication component.



(a)



(b)

Figure 3.1: (a) Block diagram of complete flow to *statically* estimate lifetime reliability (measured as MTTF) of the whole system as combination of cores plus network-on-chip, (b) The proposed dynamic reliability management scheme uses DVFS controller to set voltages and frequencies of individual tiles in the next control period such that *current* MTTF approaches *target* MTTF. The CMP systems is composed of a number of tiles. A tile is the combination of one core and one NoC router.

computed *statically* during the training process.

The second important aspect regarding the construction of the DRM scheme is to ensure that lifetime reliability estimations are accurate (and therefore

the entire scheme to ultimately be accurate), we must include in such estimations both cores and network-on-chip, because they are interdependent components of the same system. This is precisely what we do in our DRM implementation. This is very important because, as reported in [29], disregarding any of the two components during lifetime reliability estimation is prone to errors that can be as high as 60%, thereby significantly misleading any technique that attempts to optimize lifetime reliability.

The block diagram of the DRM scheme, represented in Fig. 3.1.b, is essentially implemented as a control algorithm inside our customized Gem5 based full system simulation framework. During a regular simulation of a given benchmark, for a given architecture of the CMP, information about the temperatures of all the core components and routers of the network-on-chip is used as input into the neural network based MTTF estimator. The projected or estimated MTTF is compared to the desired target MTTF by the DVFS controller, which then decides for each core whether the clock frequency must be throttled, increased, or left unchanged. The logic behind the DVFS controller is simple: if the estimated current MTTF is less than the target MTTF, then, throttle the frequency of the core to the next lower frequency from the set of frequencies we work with (and lower its supply voltage too); otherwise, raise the frequency to the next higher frequency (and raise its supply voltage too); if the estimated current MTTF is within the vicinity (dictated though a user set parameter δ) of the target MTTF, then keep the same frequency for the core. The pseudocode of this control algorithm is shown in Fig. 3.2.

Algorithm: DRM Scheme

In: Desired $MTTF_{target}$; δ hysteresis bandwidth; γ maximum percentage of updated tiles in a control period; core activity counters and routers power
Out: Frequencies and supply voltages for all tiles for next control period
Use neural network based MTTF estimator to find current MTTF of each tile and of whole CMP
if $MTTF_{CMP} < MTTF_{target} - \delta$ **then**
    Sort all tiles in increasing order of their MTTF
    for $i \leftarrow 1$ to γn **do** // n: number of tiles
        if $MTTF_i < MTTF_{target} - \delta$ **then**
            Switch down frequency and voltage of this tile
        end if
    end for
else if $MTTF_{CMP} > MTTF_{target} + \delta$ **then**
    Sort all tiles in decreasing order of their MTTF
    for $i \leftarrow 1$ to γn **do**
        if $MTTF_i > MTTF_{target} + \delta$ **then**
            Switch up frequency and voltage of this tile
        end if
    end for
end if

Figure 3.2: Pseudocode of the DVFS based DRM scheme. This control algorithm is implemented as a callable routine inside the Gem5 simulation framework. Parameters δ and γ can be set by user to allow for calibration of how *aggressive* the DRM policy is.

3.3 Hybrid DVFS and Thread Migration based Technique

In this section we propose a novel algorithm to enhance the efficiency of the proposed DVFS based dynamic reliability management technique explained in 3.2 by combining it efficiently with the thread migration based DRM technique.

The block diagram of the proposed DRM approach is shown in Fig. 3.3. Similar to the previous approach, the idea is to implement a control algorithm, which continuously monitors the temperatures of all components of both computational and communication units of the CMP hardware platform. This algorithm operates periodically according to a pre-defined *control period* and uses the input

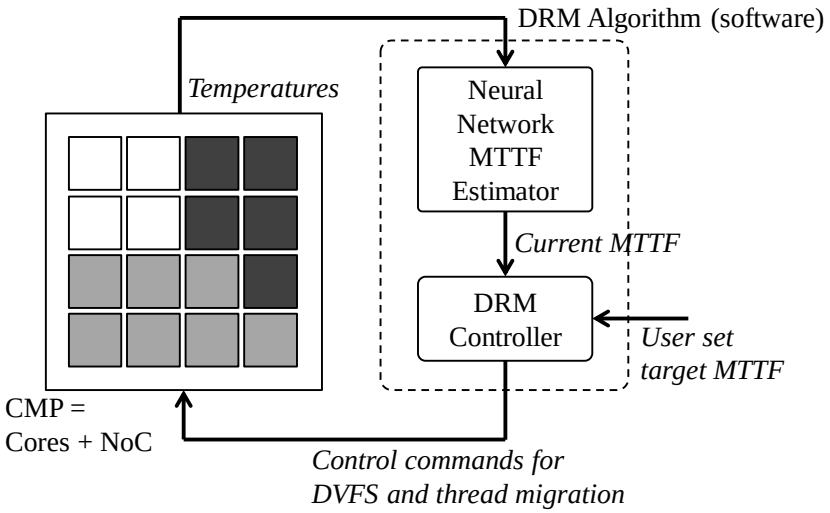


Figure 3.3: The proposed dynamic reliability management algorithm has two components, the MTTF online estimator and the DRM controller. The CMP is composed of a number of tiles and each tile contains a core and a NoC router.

temperatures as well as the user set desired lifetime reliability target to generate output control commands that dictate how thread migration among tiles and DVFS of individual tiles is done during the next control period. These commands are generated such that lifetime reliability converges toward the desired target. The algorithm is implemented in software and has two main components. The first component estimates the current lifetime reliability and is implemented with a neural network model. Its role is to produce an estimate of the MTTF of the entire CMP as a way to quantify or measure the lifetime reliability.

The second component shown in Fig. 3.3 is the DRM controller. Its role is to compare the currently estimated or projected MTTF to the desired target and then decide for each tile (core + NoC router) whether the clock frequency must be throttled, increased, or left unchanged or whether threads should be migrated from hot to colder tiles. The control loop from Fig. 3.3 shares in philosophy with

any other closed-loop control theory algorithm. However, the context in which we use elements of control theory is specific in this case to the optimization of lifetime reliability for chip multiprocessors, which we handle in a unified manner, as the combination of both cores and network-on-chip. The neural network based estimation is another specific element. The most challenging aspect of the proposed DRM algorithm is to figure out a way to make these decisions such that performance is not affected too much. The next section elaborates on how that is done.

Here, we describe how we arrived to the implementation of the logic behind the DRM controller from Fig. 3.3. First, based on our experience from Sec. 3.2 and previous study [29], we present several design insights.

We found that lifetime reliability can be more effectively improved using DVFS based techniques, but at the expense of larger performance penalties when compared to thread migration based techniques. This suggests that, for applications where performance degradation can be tolerated, DVFS based DRM schemes can be used to trade performance for larger MTTF improvements. In contrast, for applications where performance degradation is not acceptable, thread migration based DRM schemes may be a better choice. However, thread migration is limited in its ability to significantly improve MTTF even if it would be acceptable to degrade performance. That is because no matter how much one would shuffle jobs among tiles, if the application benchmark is computationally intensive and all cores are heavily utilized, temperature profile will be always high anyways. The problem is that we do not know at design time what kind of application benchmarks will be run on a given instance of a chip multiprocessor. A subtle design

insight here is that the above statements about thread migration techniques are true only when the application benchmark has a number of active threads that is comparable to the total number of tiles of the CMP. That was the case in our study from [29], where the used Parsec benchmarks were specifically compiled for the duration of the region of interest to run a number of threads equal to the number of cores. In such situations, a thread migration technique has little ability to improve the thermal profile via shuffling threads between tiles because all cores are already busy. However, if the application benchmark is compiled such that the number of active threads is smaller than the number of available cores, at any given time, then, thread migration can effectively be used to achieve significant reliability improvements. While this may not seem realistic or desirable because we would like all cores to do useful work at all times, applications that are programmed to be running parallel on multicore platforms may have situations when some of the tasks running on some cores finish early compared to other tasks running on other cores. Situations like that offer the opportunity to be exploited for example for thread migration. Moreover, with the increasing power dissipation on multicore platforms, researchers are already talking about the new era of dark-silicon [32, 34, 35]. That is, in dark-silicon, many cores would be shut-down or not be utilized in order to keep the total number of cores executing at the same time small enough and thus keep the temperatures low. These situations are examples when the number of threads can be less than the number of cores.

To further understand the relation between the number of available free cores (i.e., currently not having any running threads on them), which usually have colder temperatures, and the amount of MTTF improvement when thread

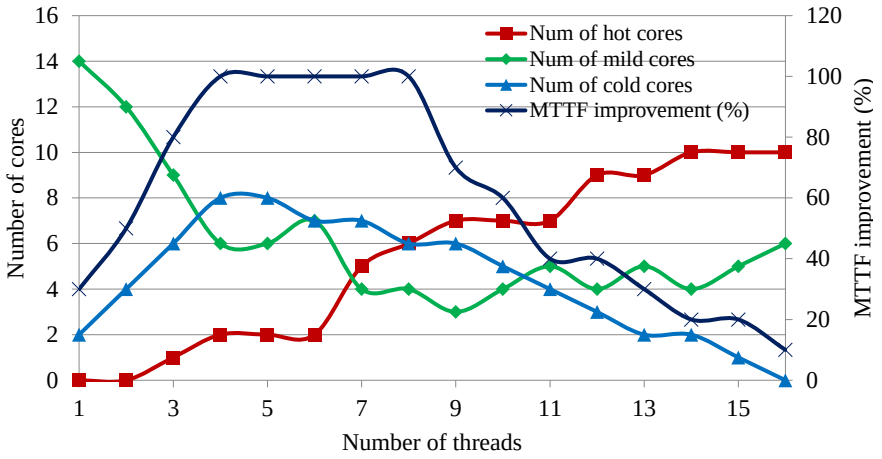


Figure 3.4: Plot showing the amount of MTTF improvement using a thread migration based DRM scheme over the reference case when no DRM scheme is used at all. A tile is denoted as cold if its temperature $T < 40°C$, as mild if $40°C \leq T \leq 60°C$, and hot if $T > 60°C$.

migration is used as the main technique for lifetime reliability management, we conducted the following experiment using our modified Gem5 based simulation framework. This simulation framework will be described in more details later on. We use an NoC based CMP with 16 cores to run several benchmarks that are compiled to run during the ROI using a number of threads varied between 1 and 16. Thus we conduct 16 different full system simulations for the given benchmark. We use our own thread migration based DRM scheme [29] and the objective is to see with how much MTTF of the whole CMP system can be improved. The results of this experiment for the *blacksholes* benchmark are shown in Fig. 3.4.

The plot from Fig. 3.4 confirms the intuition that more cold tiles available provide more opportunities to the thread migration DRM scheme to migrate threads in the attempt to keep tile temperatures within a power profile that improves the MTTF. In our simulations, we found that when the number of hot cores is less than half of the total number of cores, the MTTF can be improved

Algorithm: DRM Controller

In: Desired $MTTF_{target}$; δ hysteresis bandwidth; γ maximum percentage of updated tiles in a control period; K repetition number
Out: Frequencies and supply voltages for all tiles and thread id running on each tile during next control period
Read in temperatures of all tiles of the CMP
Use neural network based MTTF estimator to calculate MTTF of each tile and of the entire CMP
if $NumberColdTiles \geq NumberHotTiles$ **then**
    for next K control periods **do**
        Use thread migration technique
    end for
else
    if $0.8 \cdot MTTF_{target} < MTTF_{CMP} < 1.2 \cdot MTTF_{target}$ **then**
        for next K control periods **do**
            Use thread migration technique
        end for
    else
        for next K control periods **do**
            Use DVFS technique
        end for
    end if
end if

Figure 3.5: Pseudocode of the proposed DRM algorithm. In our experiments, this algorithm is implemented as a callable routine inside the Gem5 simulation framework. Parameters δ, γ, and K can be set by the user to allow for calibration of how *aggressive* the DRM strategy is. The thread migration and DVFS techniques are described in Fig. 3.6 and Fig. 3.7. The values 0.8 and 1.2 were found empirically to provide good results.

via thread migration to larger extents than when most of the cores are hot. This observation motivates us to implement the DRM controller as described in Fig. 3.5. The main idea of the controller is to 1) use as much as possible the thread migration technique because it is the cheapest to implement and provides good enough MTTF improvements with minimal performance penalty when there are enough cold tiles available and 2) use the DVFS technique when thread migration cannot be used.

The input into the DRM algorithm includes temperatures of all the major modules of the tiles formed by cores (i.e., integer execution unit, caches, etc.) and

NoC routers of the assumed regular mesh NoC as well as individual tile supply voltages. Temperatures and tile supply voltages are used by the neural network estimator to estimate lifetime reliabilities (as MTTF) of each tile containing a core and a router as well as of the overall CMP. Then, depending on the current number of cold tiles and on the comparison between the currently estimated MTTF with the desired target MTTF, the algorithm uses either the thread migration technique or the DVFS technique.

The logic behind the thread migration technique (see Fig. 3.6) is that if the currently estimated MTTF is less than the target MTTF, it moves threads from hot to cold cores to more uniformly balance the overall temperature profile, thereby increasing the current MTTF. In case that the estimated MTTF is higher than the target, no thread is migrated. The logic behind the DVFS technique (see Fig. 3.7) is that if the estimated current MTTF is less than the target MTTF, then, throttle the frequency of the core to the next lower frequency from the set of frequencies we work with (and lower its supply voltage too); otherwise, raise the frequency to the next higher frequency (and raise its supply voltage too); if the estimated current MTTF is within the vicinity (dictated though a user set parameter δ) of the target MTTF, then keep the same frequency for the core.

3.4 Simulation Results

This section contains the simulation results for the proposed DRM techniques discussed in Sec. 3.2 and 3.3.

```
Routine: Thread migration technique
1: if MTTF_CMP < MTTF_target − δ then
2:     Sort all tiles in increasing order of their MTTF
3:     for i ← 1 to γn/2 do // n: number of tiles
4:         if MTTF_i < MTTF_target − δ then
5:             Migrate the thread in i^th tile to the (n − i)^th tile and vice versa
6:         end if
7:     end for
8: end if
```

Figure 3.6: Pseudocode of routine describing the thread migration technique called by the proposed DRM algorithm from Fig. 3.5

```
Routine: DVFS technique
1: if MTTF_CMP < MTTF_target − δ then
2:     Sort all tiles in increasing order of their MTTF
3:     for i ← 1 to γn do // n: number of tiles
4:         if MTTF_i < MTTF_target − δ then
5:             Switch down frequency and voltage of this tile
6:         end if
7:     end for
8: else if MTTF_CMP > MTTF_target + δ then
9:     Sort all tiles in decreasing order of their MTTF
10:    for i ← 1 to γn do
11:        if MTTF_i > MTTF_target + δ then
12:            Switch up frequency and voltage of this tile
13:        end if
14:    end for
15: end if
```

Figure 3.7: Pseudocode of routine describing the DVFS technique called by the proposed DRM algorithm from Fig. 3.5

3.4.1 Simulation Setup

Because we do not have access to NoC based CMP platforms with tens of cores that we investigate in this study (CMP architectures, which often times are exploratory), we resort like the rest of the research community to the next best way to test and validate our ideas, simulation tools. To test our DRM algorithm, we have developed our own simulation framework, which is implemented on top of the popular Gem5 full system simulation tool [103]. In addition, we integrate in

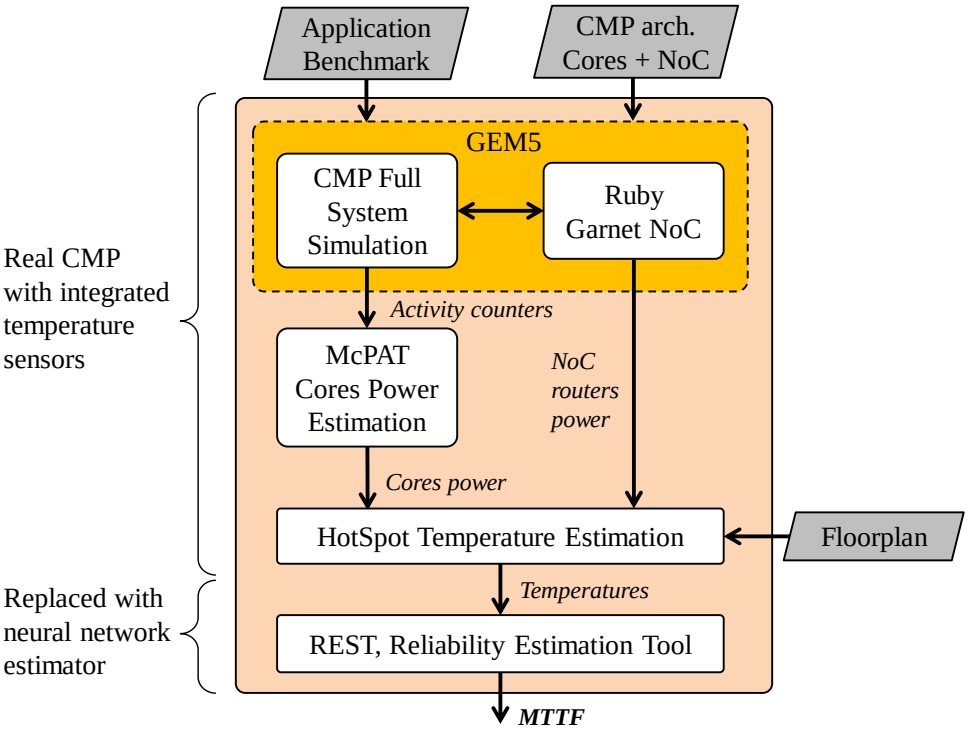


Figure 3.8: Block diagram of the complete simulation framework to simulate a given application benchmark and to estimate lifetime reliability, measured as MTTF, of the entire system as combination of cores plus network-on-chip. Note that when the REST tool is replaced by the neural network MTTF estimator, supply voltages are also provided together with temperatures as inputs to the estimator.

our simulation framework several other point tools as shown in the block diagram from Fig. 3.8.

Gem5 tool is one of the most popular full system simulators capable of simulating entire computing systems constructed around singlecore or multicore processors. In the case of chip multiprocessors, the tool can model and simulate NoC communication between cores. It provides detailed timing and performance data and also integrates capabilities to estimate NoC router and link power consumption. Hence, simulation of a given application benchmark is accurate and it also accounts for the operating system. Unfortunately, as we mentioned earlier,

Gem5 tool does not output temperatures of cores, which in real processors would be available through temperature sensors integrated on chip. Therefore, we must use two additional tools in a sequence as shown in Fig. 3.8. Specifically, we first use McPAT power calculator [104] to compute power consumption values for cores based on the performance data (activity counters) from Gem5. Then, the power values of all cores and NoC routers are fed into the HotSpot temperature calculator [105] to estimate temperatures. Finally, the temperature values are used as input into the REST tool (described in Sec. 2.2) to estimate MTTF of each tile as well as of the entire CMP. The default architectural configuration parameters utilized in our custom Gem5 based simulations, unless otherwise specified, are shown in Table 3.1.

We would like to emphasize that within a simulation framework like this, we can perform any exploratory investigations for any chip multiprocessor architecture of interest. In addition, the simulation framework has the advantage of being able to stretch the ROI execution time for a given application benchmark in order to allow the complete sequence of processing steps illustrated in Fig. 3.8 to be performed. In real life deployment of the proposed DRM algorithm though, this sequence of steps would not be necessary because the on chip temperature sensors would provide temperature information directly. This is indicated on the left hand side of Fig. 3.8, where these processing steps would be shortcut by temperature sensor readings. The same figure shows that the REST tool from this simulation setup would be replaced in real life deployment with a neural network based estimator described in detail, including training data generation and the training process, in the previous work [29].

Table 3.1: Architectural configuration parameters.

Parameter	**Value**
Technology node	65nm
Frequencies	2GHz downto 1.4GHz, with 100MHz step
VDDs	1.1V downto 0.95V, with 25mV step
Core	Alpha EV6 21264
Core CPU model	Out of order (Detailed CPU)
Branch predictor	2 bit counter
Reorder buffer	80-entries
L1 ICache	32KB
L1 DCache	64KB
L2	2MB
Network	2D regular mesh, 1 router per core
Tile floorplan	Router to the top of core ALU
Link bandwidth	32 bits
Routing algorithm	XY
Number of virtual channels (VCs)	2

In all our simulations, we consider both the computational (i.e., cores) and communication (i.e., network-on-chip) components in a unified manner for the purpose of the MTTF calculation.

3.4.2 DVFS based Technique

We conduct full system simulations on several Parsec benchmarks [107] to investigate the DRM scheme for two different CMP architectures composed of 4 cores and 16 cores respectively. Each of these architectures uses regular mesh NoCs: 2x2 and 4x4. In our simulations, we set as target or desired average MTTF a value that is 100% longer than what it is when no DRM is applied, which is our reference case. In other words, we are interested in doubling the average lifetime of the investigated CMP architectures. Fig. 3.9 − Fig. 3.12 show the simulation results for *blackscholes*, *canneal*, *bodytrack*, and *dedup* Parsec benchmarks run as

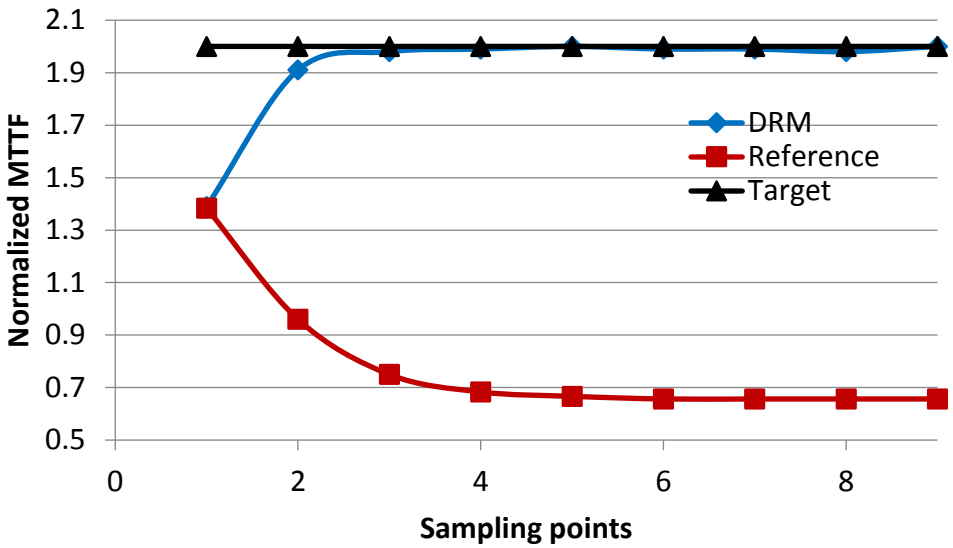


Figure 3.9: Gem5 with DVFS based DRM simulation of *blackscholes* benchmark.

applications with 16 threads on a CMP architecture with 4x4 tiles. The plots show only the period of time that covers the so called region of interest of the Gem5 simulation. For each simulation shown in these figures, the Gem5 simulator is stopped a number of times during the ROI (this number depends on the actual length of the ROI and the selected *control period* discussed in the previous section) to perform DRM and update the frequencies and voltages of each tile. Each of these stop-times corresponds to a data point out of the *sampling points* shown on the horizontal axis in Fig. 3.9 – Fig. 3.12.

We note that for some benchmarks the MTTF fluctuates around the target MTTF. This is for example the case of *bodytrack* and *dedup* benchmarks. We suspect that this is primarily due to the variation in the workload that each core must do for these particular benchmarks during different control periods inside the ROI. This may also be as a result of the changes in dependencies created by frequency throttling among jobs that are executed on different cores. We noticed

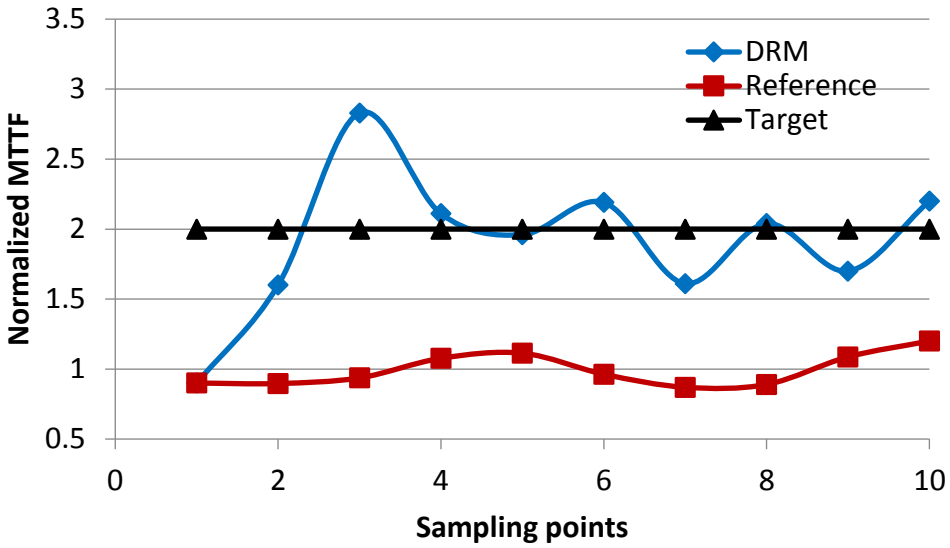


Figure 3.10: Gem5 with DVFS based DRM simulation of *canneal* benchmark.

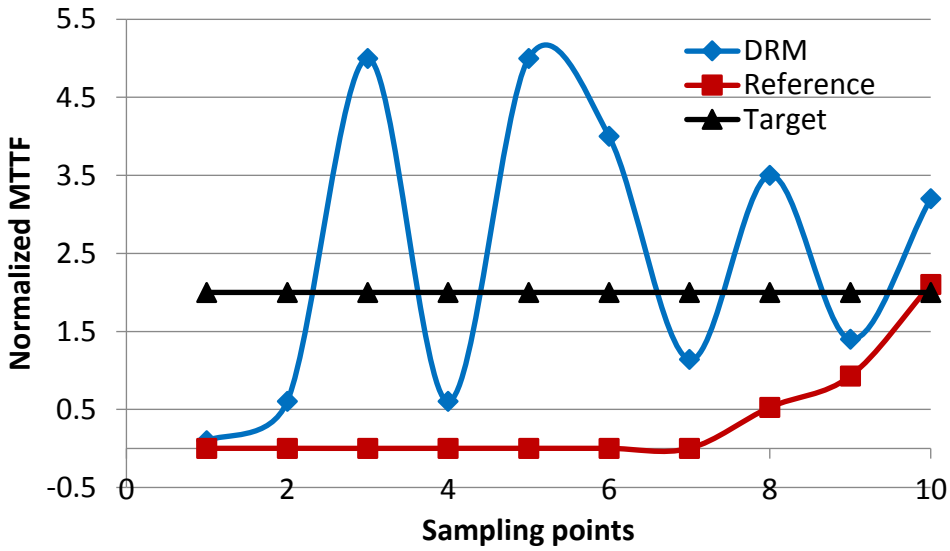


Figure 3.11: Gem5 with DVFS based DRM simulation of *bodytrack* benchmark.

that when all cores are loaded with work uniformly throughout the ROI (as is the case of the *blackscholes* and *canneal* benchmarks), the overall MTTF is more stable. Better calibration of the proposed DRM algorithm from Fig. 3.2 can help address such fluctuations.

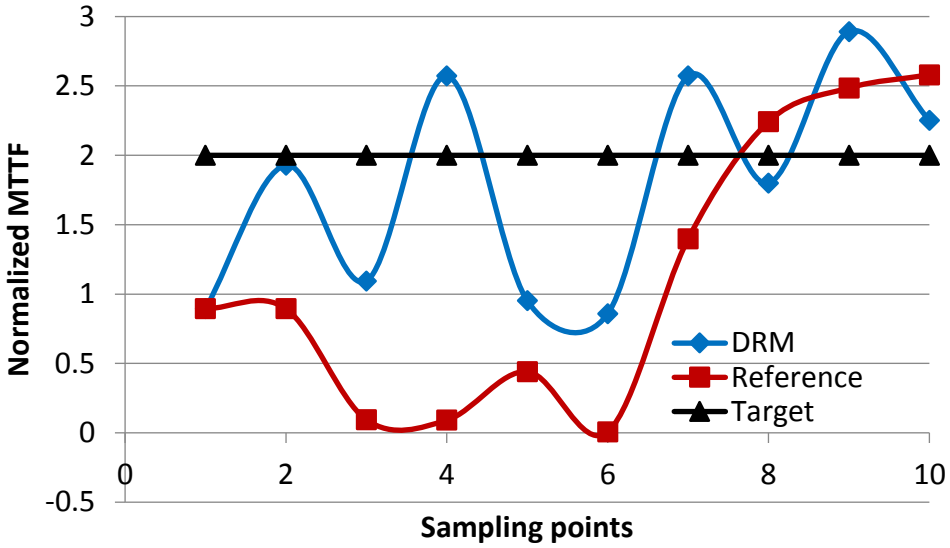


Figure 3.12: Gem5 with DVFS based DRM simulation of *dedup* benchmark.

Table 3.2: Summary of simulations shown in Fig. 3.9 – Fig. 3.11

Benchmark	**Avg. MTTF improv.**	**Perf. penalty**	**ROI exec. time (reference run)**	**Gem5 sim. time**
blackscholes	100%	11.8%	69 ms	6 h
canneal	100%	16.96%	103 ms	12 h
bodytrack	100%	9.3%	139 ms	9 h
dedup	100%	15.8%	376 ms	18 h

Table 3.2 summarizes the information presented in these plots. The performance penalty includes the time spent to perform the reliability estimation as shown in Fig. 3.1.b and to execute the DRM algorithm presented in Fig. 3.2.

3.4.3 Hybrid DVFS and Thread Migration based Technique

To evaluate our proposed approach, in our simulations, we conduct experiments on several application benchmarks to investigate the proposed DRM algorithm for two different CMP architectures composed of 16 cores and 64 cores, respectively. Each

of these architectures use 4×4 or 8×8 regular mesh networks-on-chip with default configuration parameters described in Sec. 3.4.1. The values of the parameters from Fig. 3.5 are $\delta = 10\%$, $\gamma = 50\%$, and $K = 4$. These values were found empirically to provide good results.

We report simulation results for several Parsec [107] and Splash2x [107] application benchmarks. In our simulations, similar to Sec. 3.4.2 we set as target or the desired average MTTF a value that is with 100% longer than what it is when no DRM algorithm is used, which is our reference case. In other words, we are interested in doubling the average lifetime of the investigated CMP architectures.

Fig. 3.13 shows the simulation results for *blackscholes* benchmark using 16 threads on a CMP architecture with 4×4 tiles. The plot shows only the period of time that covers the region of interest of the Gem5 simulation. During each simulation, the Gem5 simulator is halted a number of times during the ROI (this number depends on the actual length of the ROI and the selected *control period* discussed earlier in this chapter) to perform DRM and to update the frequencies and voltages of each tile or to perform thread migration. Each of these stop-times corresponds to a data point out of the *sampling points* shown on the horizontal axis in Fig. 3.13. Note that in these figures the horizontal axis represents sampling points and not actual ROI execution time. That is because the actual length of the ROI portion when DVFS technique is used becomes longer in absolute time due to frequency throttling. This figure shows that the proposed DRM algorithm can bring and maintain the MTTF above or just beneath the desired objective (within δ parameter from Fig. 3.5). It can be noted that MTTF fluctuates around

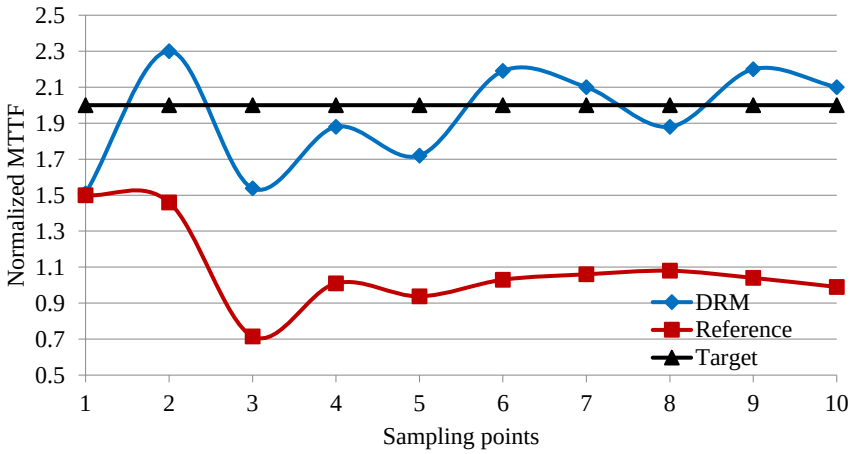


Figure 3.13: Simulation results for *blackscholes* benchmark on an architecture with 4×4 tiles (i.e., 16 cores). Similar results were obtained for the other benchmarks.

the target MTTF. That is because 1) of the variation in the workload that each core must do during different control periods inside the ROI and 2) of the inherent inertia when dealing with latent variables like temperature.

Fig. 3.15 shows the thermal profiles of the 4×4 CMP architecture in the reference case (i.e., no DRM algorithm being used) and in the case when the proposed DRM algorithm is used. These thermal maps correspond to the fifth sampling point from Fig. 3.13 and show that the DRM algorithm successfully manages to pull down the temperature of all tiles in the CMP architecture, thereby improving the MTTF of the entire system.

For an 8×8 CMP architecture, Fig. 3.14 shows the simulation results for *cholesky* benchmark compiled to use 64 threads. The dip formed by point 4-6s of the reference plot is because this particular benchmarks has a behavior that creates a workload pattern or traffic which is much heavier in the middle of the ROI period. During this dip, the activity counters registered inside the Gem5 simulations are much higher. This directly translates into increased temperature values that in

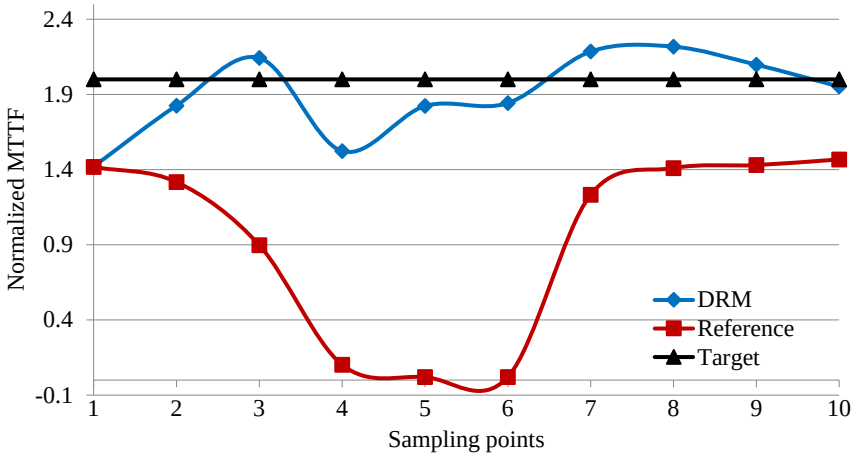


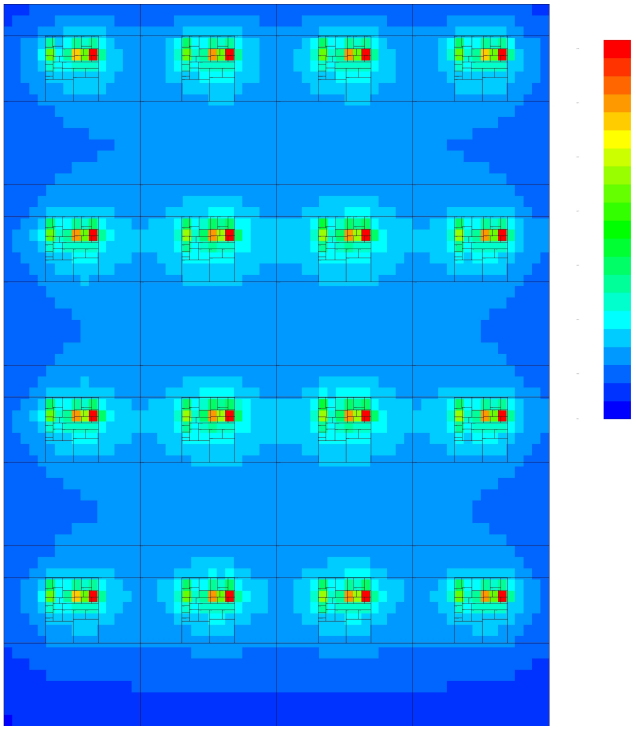
Figure 3.14: Simulation results for *cholesky* benchmark on an architecture with 8×8 tiles (i.e., 64 cores). The MTTF of the reference case improves in the second part of the ROI because the actual workload decreases (some threads finish much earlier) for this particular benchmark.

turn trigger a degradation of the lifetime reliability. Likewise, Fig. 3.16 shows the thermal profiles, which again indicate that the DRM algorithm successfully manages to pull down the temperature of all tiles in the CMP architecture, thereby improving the MTTF of the entire system.

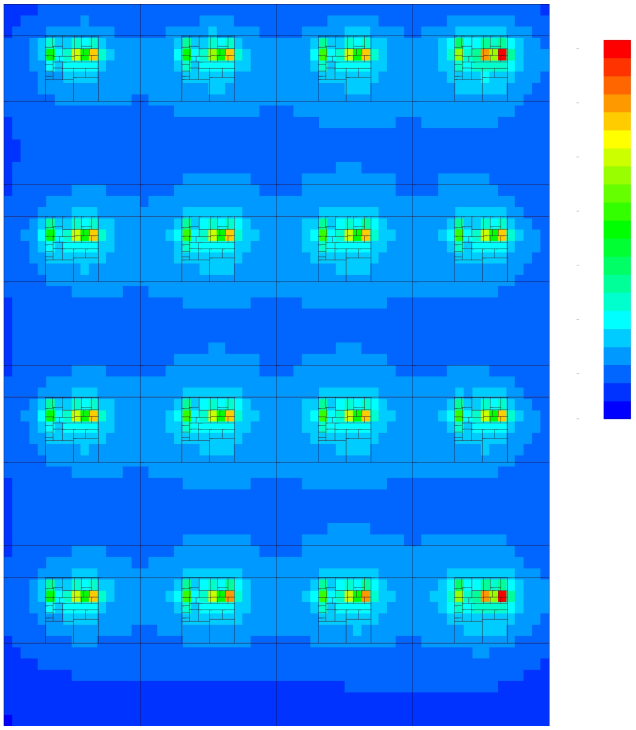
We present summary plots (see Fig. 3.17 and Fig. 3.18) that show the performance penalty and the change in energy delay area product (EDAP) for a user set target MTTF improvement of 100%, for each of the investigated benchmarks for both CMP architectures. For all simulated benchmarks, the target MTTF was reached. However, the achieved MTTF has fluctuations/oscillations around the target as observable in Fig. 3.13 and Fig. 3.14.

3.5 Discussion

The results indicate a significant improvement in lifetime reliability when we use only DVFS based scheme. However, this improvement is at the expense of some

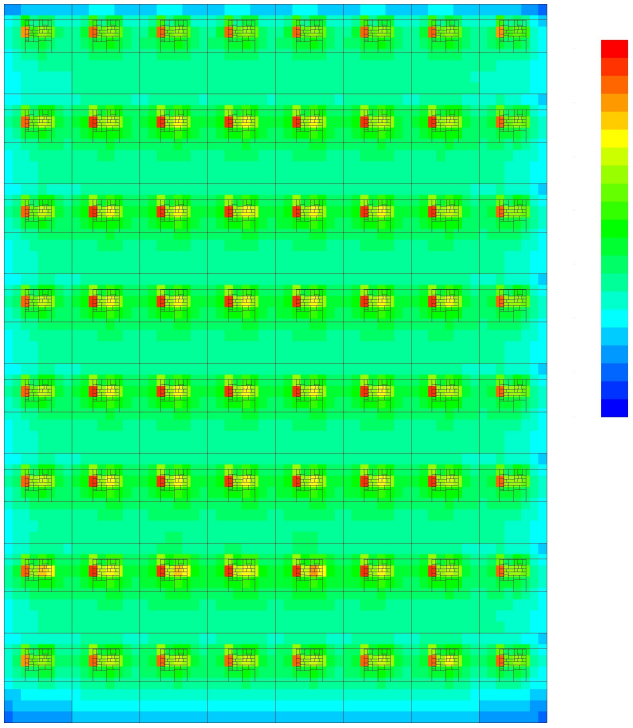


(a)

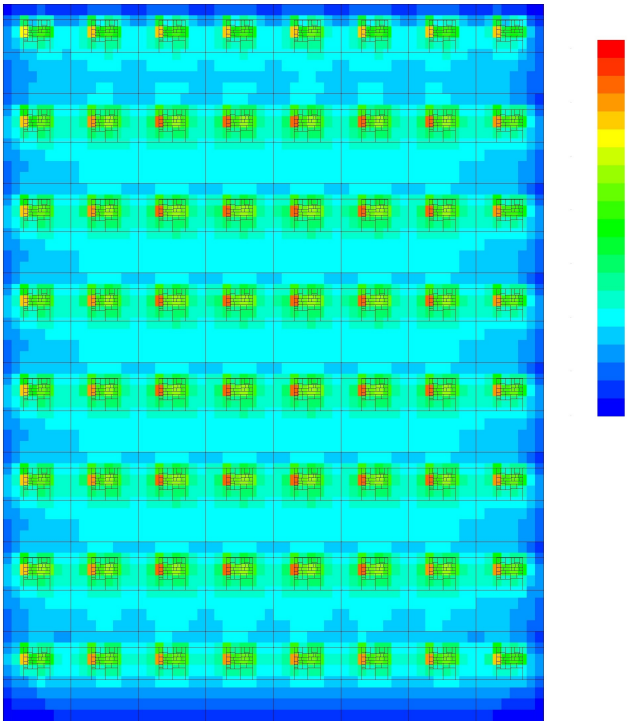


(b)

Figure 3.15: (a) Thermal profile of the 4×4 CMP architecture running *blackscholes* benchmark with no DRM algorithm, (b) Thermal profile of the same architecture when the proposed DRM algorithm is used. The color-coded temperature range is $20°C$ (blue) to $120°C$.



(a)



(b)

Figure 3.16: (a) Thermal profile of the 8×8 CMP architecture running *cholesky* benchmark with no DRM algorithm, (b) Thermal profile when the proposed DRM algorithm is used.

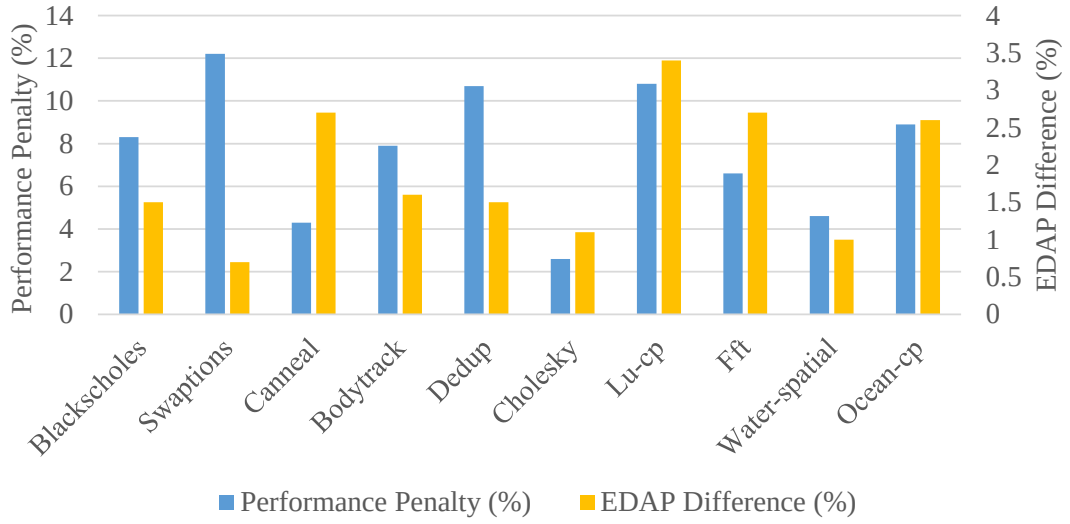


Figure 3.17: Summary of simulations results for 4×4 CMP architecture for a target MTTF improvement of 100% (i.e., double lifetime). Each data point is the average of all values obtained during the hold times or sampling points illustrated in Fig. 3.13 for a given benchmark.

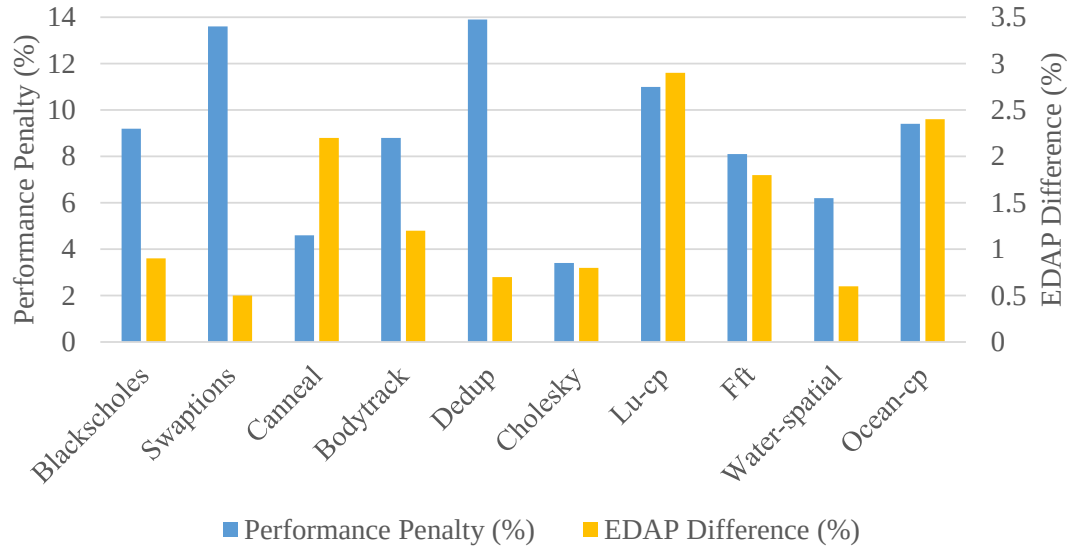


Figure 3.18: Summary of simulations results for 8×8 CMP architecture for a target MTTF improvement of 100%.

performance penalty, as shown in Table 3.2. When compared to the thread migration based DRM scheme studied in [29] [2], we note that when only the thread

[2]We consider the DRM scheme from [29] for comparison purposes because it is the only other DRM scheme that considers both cores and NoC in a unified manner. All other previous DRM schemes do not include the NoC component in their optimization, and therefore their reported MTTF values may be off by as much as 60% as reported in [29].

migration was employed, the MTTF was improved by only up to 50% with up to 9.16% performance penalty. However, in the proposed DVFS based approach, MTTF can be improved by 100% but with higher performance penalties (up to 16%). This suggests that, for applications where performance degradation is not acceptable, a thread migration based DRM scheme may be a better choice. In applications where performance degradation can be tolerated, the proposed DVFS based DRM scheme can be used to trade performance for larger MTTF improvements. Note that, *frequency throttling* can theoretically improve MTTF significantly – at the limit, if cores are completely stopped, MTTF becomes infinity. On the other hand, *thread migration* is limited in its ability to significantly improve MTTF even if it would be acceptable to degrade performance – that is because no matter how much one could shuffle jobs among cores, if the benchmark is computationally intensive and all cores are heavily utilized, temperature profile will be high anyways.

The combination of both thread migration and DVFS techniques presented in Sec. 3.3 offers a better tradeoff between MTTF improvement and performance degradation. As we see in Fig. 3.17 and Fig. 3.18, the hybrid technique performs better than each of the approaches when only either thread migration or DVFS is employed. The results show that we can still have the 100% improvement in lifetime reliability, but with 7.7% and 8.7% performance penalty in average.

In the simulations, we notice that some applications are highly correlated in terms of their performance penalty and EDAP. This correlation is higher for example for benchmarks *lu-cp* and *fft* when executed on 8×8 CMP architectures.

On the other hand, this correlation is higher for benchmarks *blackscholes* and *bodytrack* when executed on 4×4 CMP architectures. We suspect that this correlation could be in part due to the specific characteristics of traffic patterns that each benchmark creates on each core and through the network-on-chip. We noticed in our simulations that when the CMP system runs fully with all cores being busy all the time, it is very difficult to find room for improvement; no matter what one would do as DVFS or thread migration, the penalty in performance is more prevalent.

CHAPTER 4

Proposed Dynamic Energy Management

4.1 Introduction

In this chaper, we investigate three different DVFS based approaches for dynamic energy management under performance constrains. We first develop an effective algorithm for performance loss estimation where DVFS technique is employed and then we propose novel approaches using Kalman filtering, a LSTM and a DNN to dynamically optimize the energy of the CMP.

4.2 Delayed Instruction Count Performance Estimation

The idea of the proposed dynamic energy management is to continuously monitor the CMP system operation and to periodically make decisions to *tune* different control *knobs* with the objective of shifting the system's operation to states where energy consumption is reduced as much as possible but without degradation of performance beyond the user specified threshold. In our case the *knob* is represented by the voltage/frequency pairs, which can be set individually for each of the cores of the CMP processor, effectively done as part of the DVFS algorithm. This is under the assumption that by default (i.e., in the reference or base case) all cores operate at the highest frequency to achieve the best possible performance. *Tuning* the knob translates into frequency throttling or frequency increase (if throttling has been done before for a given core), at opportune times, in order to save energy.

The key challenge in achieving that is to find a way to dynamically change between frequency voltage pairs such that the performance degradation is not more than the acceptable threshold, which is set by the user as a percentage, such as 5% performance degradation. The performance constraint is what complicates the problem in this case. We address this challenge by introducing a new concept, that of *delayed instructions count*, which we use to calculate dynamically the amount of *performance loss* that we would introduce if we were to switch the current voltage-frequency pair for a given core for the *next control period* to a throttling pair (i.e., lower frequency). This amount of performance loss is estimated with respect to the reference case, which is always that of the highest frequency. We describe next the derivation of the expression that we propose to use for the estimation of performance loss.

Assume that we denote with CPI the average CPU cycles per instruction when the CPU is not stalled and does useful work at a clock frequency that we refer to as f_{cpu}. Assume also that we denote with CPI' the average system cycles per instruction that the CPU is stalled because it has to wait due to branch misses, TLB misses, lowest level cache misses, pipeline stalls, etc. Note that with these notations, we consider the execution of a given instruction as being made up of two portions. One portion is given by the CPI as average number of CPU frequency cycles per instruction and the other portion is given by CPI' as average number of system frequency cycles per instruction. This is illustrated for a simple example in Fig .4.1.

The performance of a processor for a given application is quantified via

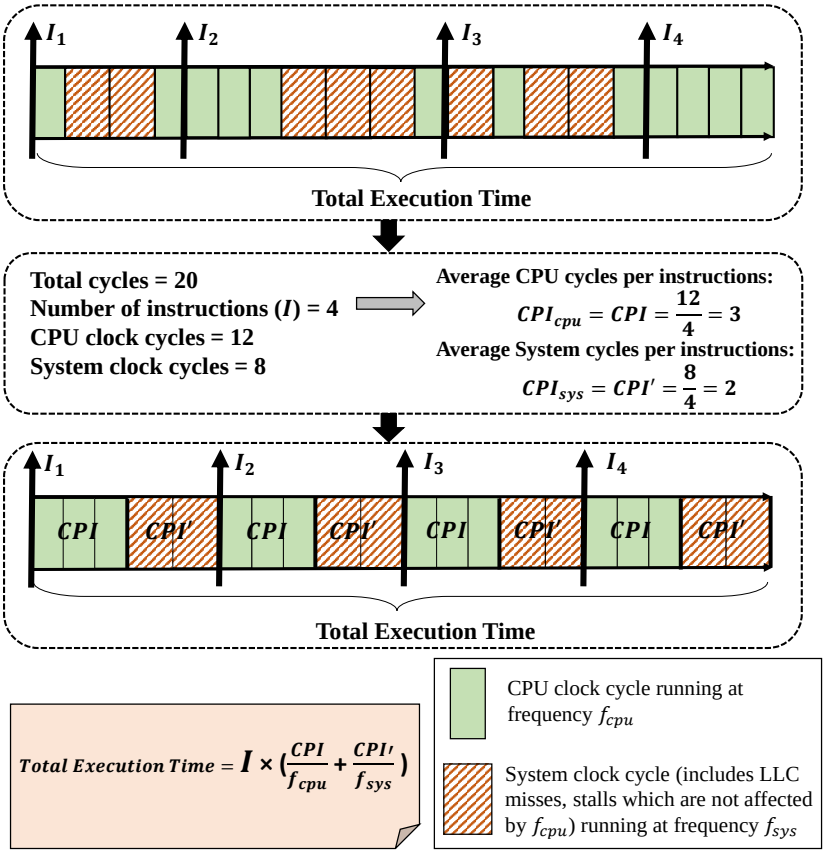


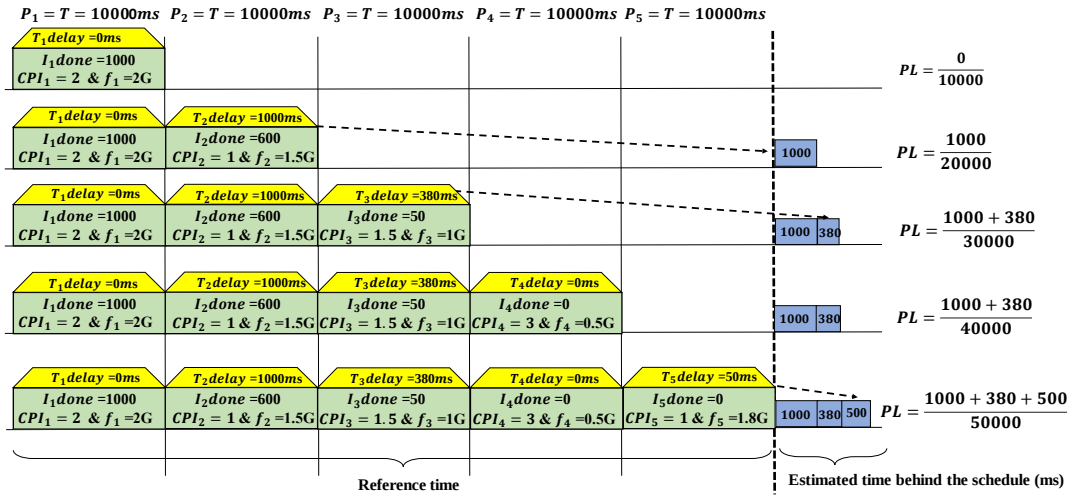
Figure 4.1: Example utilized to illustrate the two different average cycles per instruction, CPI and CPI', which are used to estimate the total execution time.

the total execution time, T_{total}, given by the following expression.

$$T_{total} = I \times (\frac{CPI}{f_{cpu}} + \frac{CPI'}{f_{sys}}), \tag{4.1}$$

where I is the total number of instructions, f_{cpu} is the clock frequency that the processor is operated at, and f_{sys} is the system clock frequency.

The total execution time of the application is partitioned into a number of control periods. During the execution of the application, the system calls the proposed algorithm at the end of each control period in order to decide about the V/F pairs for all cores during the next control period. Assume we refer to such



$I_P$done = Number of instructions done with frequency f_P in period P
CPI_P = CPU Cycles Per Instruction in period P
Let's assume $I_PNotDone$ as the number of estimated instructions not done in period P, due to running with frequency f_P instead of frequency f_H (f_H=2GHz in this example) T_Pdelay = Estimated delay for running $I_PNotDone$ instructions with CPI_P and frequency f_H calculated as: $$T_Pdelay = I_Pdone \times \frac{CPI_P \times (\frac{f_H}{f_P} - 1)}{f_H}$$
PL = Estimated Performance Loss $$PL = \sum_{i=0}^{P} \frac{D_i}{T}$$

Figure 4.2: Example utilized to illustrate the estimation of total performance loss (PL) so far, up to and including the currently completed control period and just before the start of a new control period for a given core.

periods with the generic index P. Then, applying the same rationale as that for deriving equation (4.1), for a generic control period P, we can write the expression for the duration of the period T_P as:

$$T_P = I_{P_{Done}} \times (\frac{CPI_P}{f_P} + \frac{CPI'_P}{f_{sys}}) \tag{4.2}$$

Where, CPI_P and CPI'_P are the average cycles per instruction during period P.

$I_{P_{done}}$ represents the number of instructions executed during period P when the core operates at a particular clock frequency f_P.

Similarly, having the same average cycles per instruction, if the execution is done at the highest frequency f_H, a control period of the same walltime duration would have executed say $I_{P_{Max}}$ instructions.

$$T_P = I_{P_{max}} \times (\frac{CPI_P}{f_H} + \frac{CPI'_P}{f_{sys}}) \tag{4.3}$$

Using the equations (4.2) and (4.3), the expression for $I_{P_{max}}$ can be derived as:

$$I_{P_{max}} = I_{P_{Done}} \times (\frac{\frac{CPI_P}{f_P} + \frac{CPI'_P}{f_{sys}}}{\frac{CPI_P}{f_H} + \frac{CPI'_P}{f_{sys}}}) \tag{4.4}$$

Which can be rewritten as:

$$I_{P_{max}} = I_{P_{Done}} \times (\frac{CPI_P(\frac{f_H}{f_P}) + CPI'_P(\frac{f_H}{f_{sys}})}{CPI_P + CPI'_P(\frac{f_H}{f_{sys}})}) \tag{4.5}$$

Let us denote as $I_{P_{NotDone}} = I_{P_{max}} - I_{P_{Done}}$ the number of instructions that turned out not to be executed or done in the last control period due to the fact that the frequency was throttled from f_H to f_P. These delayed or postponed instructions will introduce a delay penalty compared to the case when all the instructions would have been run at f_H. This number of instructions is what we call the DIC, which are postponed for later, and which will result in some performance degradation. The expression for it can be written as:

$$I_{P_{NotDone}} = I_{P_{Done}} \times ((\frac{CPI_P(\frac{f_H}{f_P}) + CPI'_P(\frac{f_H}{f_{sys}})}{CPI_P + CPI'_P(\frac{f_H}{f_{sys}})}) - 1) \tag{4.6}$$

That can be simplified to:

$$I_{P_{NotDone}} = I_{P_{Done}} \times (\frac{CPI_P(\frac{f_H}{f_P} - 1)}{CPI_P + CPI'_P(\frac{f_H}{f_{sys}})}). \tag{4.7}$$

Using again an expression similar to that from equation (4.1), we can calculate the extra time it would take to run $I_{P_{NotDone}}$ instructions at the highest clock frequency f_H. Let us refer to that as $T_{P_{delay}}$, which is essentially the penalty incurred in control period P because of running at a lower frequency, f_P:

$$\begin{gathered} T_{P_{delay}} = I_{P_{NotDone}} \times (\frac{CPI_P}{f_H} + \frac{CPI'_P}{f_{sys}}) = \\ (I_{P_{Done}} \times (\frac{CPI_P(\frac{f_H}{f_P} - 1)}{CPI_P + CPI'_P(\frac{f_H}{f_{sys}})})) \times (\frac{CPI_P}{f_H} + \frac{CPI'_P}{f_{sys}}) \end{gathered} \tag{4.8}$$

Which can be reduced to:

$$T_{P_{delay}} = I_{P_{Done}} \times (\frac{CPI_P(\frac{f_H}{f_P} - 1)}{f_H}) \tag{4.9}$$

Finally, the total performance loss that is incurred over all the control periods, is calculated as the following summation:

$$PL = \sum_{P=1}^{N} \frac{T_{P_{delay}}}{T} = \sum_{P=1}^{N} \frac{I_{P_{Done}} \times (\frac{CPI_P(\frac{f_H}{f_P} - 1)}{f_H})}{T}, \tag{4.10}$$

where N is the total number of periods and T is the duration or length of the control period.

Knowing previously selected frequencies for each of the cores of the CMP that were used in the last control period, together with the statistics about how many instructions have been executed by each core and what was the average CPI in the last control period (in system simulators as well as on current operating systems running on real multicore hardware, these data are readily available), we

use equation (4.10) to estimate how much aggregated extra delay we have incurred up to the current control period. This is illustrated for a very simple example in Fig. 4.2, where we can see for example that at the end of the first control period the PL is zero because the execution during the first control period was done at the highest clock frequency $f_1 = 2\ GHz$. However, at the end of the second control period, we have incurred a performance loss of 1000/20000 because a lower frequency of $f_2 = 1.5\ GHz$ was used (see second row in Fig. 4.2), and so on.

4.3 Kalman Filtering based Technique

The expression in equation (4.10) gives us a good measure of the loss suffered in the control period that just finished execution. However, we would like to use it to estimate the performance loss during the next, incoming control period and based on that to be able to make an informed decision about what V/F pair to use that gives us maximum energy reduction within the limit of allowable performance degradation. The issue now however is that we need a way to predict what the actual workload will be in the next control period. In other words, at the end of the control period index P, we need a predictor for instruction counts and CPI of the next control period, that of index $P+1$. To do that we use a Kalman filtering based approach. We use a Kalman filtering based approach because we found it to be the best compromise between complexity of implementation, efficiency, and accuracy of prediction while considering a history of w past control periods.

The block diagram from Fig. 4.3 is essentially implemented as a control algorithm inside our customized Sniper based CMP system simulation framework.

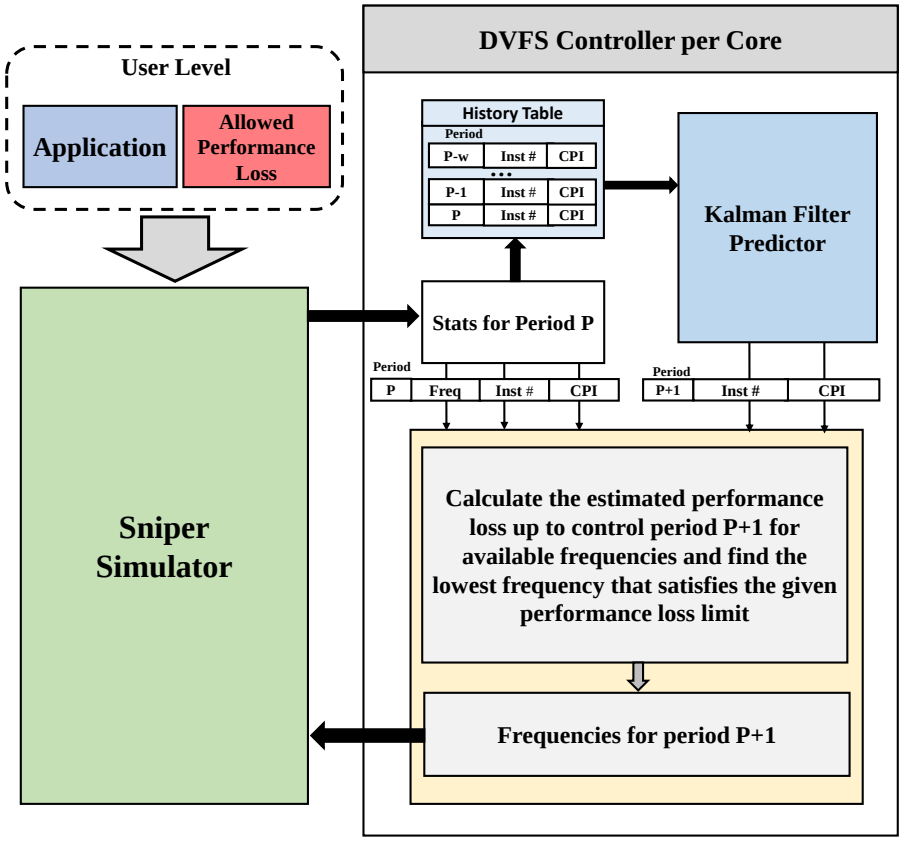


Figure 4.3: Block diagram of the proposed DVFS based dynamic energy management (DEM) scheme as implemented inside our custom Sniper simulator.

During a regular simulation of a given application or benchmark, for a given architecture of the CMP, information about the activity counters (i.e., number of instructions executed by each core and CPI) is fed to our algorithm. Our algorithm is *dynamic*, i.e., applied directly at runtime, and does not need any *static* application analysis or profiling that would be done in advance in order to identify improvement possibilities. The execution of the given application is done as a series of control periods. The information collected from the Sniper simulator at the end of each control period is recorded for a *moving window of w past periods* and is utilized to make predictions about the next control periods' instruction counts

Algorithm 1: Dynamic Energy Management (DEM) Under Performance Constrainsts

Input:
α: acceptable performance loss ratio threshold; $I_{P_{done}}$, CPI_P for just ended control period
Output:
(V_{P+1}, f_{P+1}) V/F pairs for all cores for next control period

Definitions:
T duration of each control period
$I_{(P+1)_{done}}$, CPI_{P+1} predicted with Kalman filter predictor
$FreqList$: list of available frequencies sorted from low to high (f_H)

$T_{delay} = 0$
$T_{ref} = 0$
if end of *control period* index P **then**
  for each *core* in CMP **do**

    $T_{ref} += T$
    $T_{delay} += \frac{I_{P_{done}} \times (\frac{f_H}{f_P} - 1) \times CPI_P}{f_H}$
    $Freq\_set = False$

    for $Freq$ in $FreqList$ **do**

      $T_{ref_{next}} = T_{ref} + T$

      $PredT_{delay} = T_{delay} +$
      $\frac{I_{(P+1)_{done}} \times (\frac{f_H}{Freq} - 1) \times CPI_{P+1}}{f_H}$

      $PredPerfLoss_{P+1} = PredT_{delay} / T_{ref_{next}}$

      if $PredPerfLoss_{P+1} < \alpha$ **then**
        $f_{P+1} = Freq$
        $Freq\_set = True$
      end if
    end for
    if $Freq\_set = False$ **then**
      $f_{P+1} = f_H$
    end if
  end for
end if

Figure 4.4: Pseudocode of the DVFS algorithm. This control algorithm is implemented as a callable routine inside our modified Sniper CMP simulator. It corresponds to the block at the bottom in Fig. 4.3. The parameter α is set by the user.

and CPI. The prediction is done with a Kalman filter based predictor as shown in Fig. 4.3.

The predictions are then used inside the algorithm for estimating the performance loss using equation (4.10) and for deciding the V/F pairs for all cores for

the next control period. The V/F pairs are selected to maximize energy savings but without violation of the performance loss constraint, which is the user specified. We assume that, based on the criticality of the application, the user defines a tolerable performance loss ratio. The pseudocode of this control algorithm is shown in 4.4. The algorithm works with a list of V/F pairs, out of which new V/F pairs are selected for cores if that results into energy reduction without violating the performance degradation ratio threshold specified by the user. For each period, using the predicted instruction counts and cycles per instruction estimated by Kalman predictor for the next control period, the algorithm finds the lowest frequency to save the maximum possible energy that satisfies the performance constraints.

4.4 LSTM based Technique

The method described in the previous section proposed a Kalman filtering approach to predict the workload in the next control period. Our objective in this section is to investigate other prediction approaches. Specifically, we are interested in the use of the LSTM model due to its ability to capture history in time series.

The block diagram of the proposed dynamic energy management scheme using a LSTM predictor is shown in Fig. 4.5. The scheme is implemented as a control loop inside our customized Sniper simulation framework. For each application or benchmark, the system simulator is halted periodically. At each stop, statistics about the performance counters (i.e., number of instructions executed by each core and CPI values) are collected and fed into the algorithm. The algorithm records the last statistics for a moving window of w past control periods. It sends

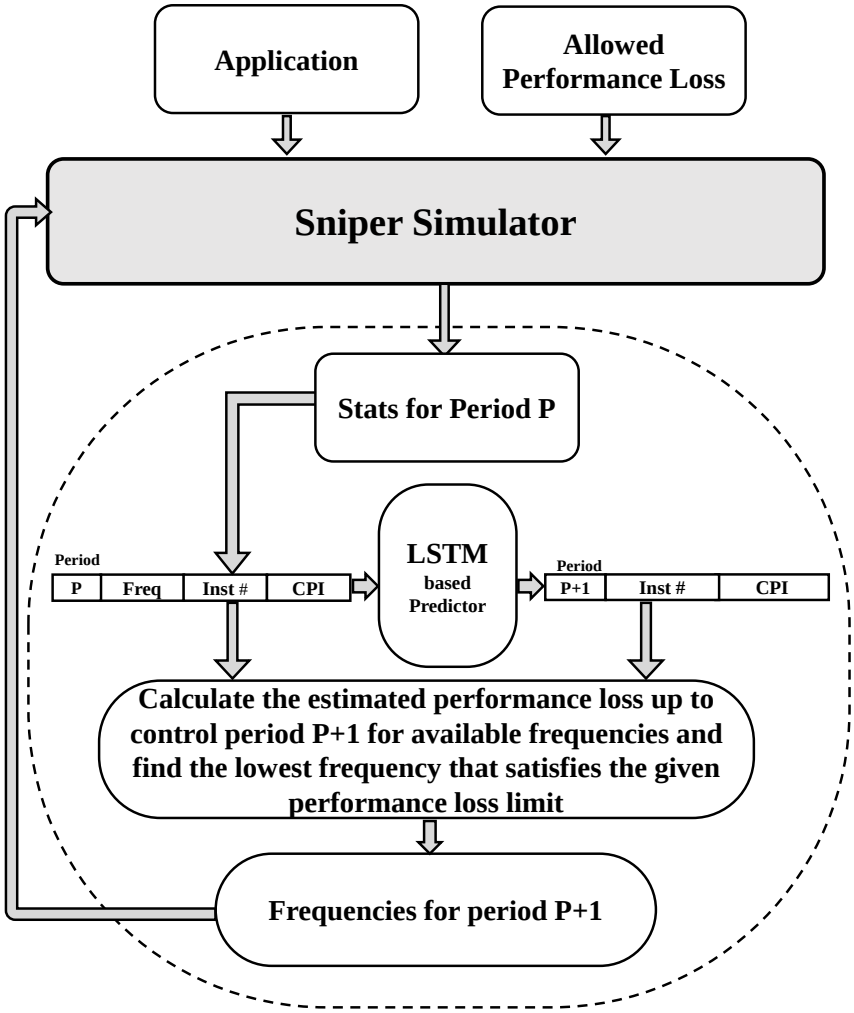


Figure 4.5: Block diagram of the DVFS based dynamic energy management scheme as implemented inside our custom Sniper simulator.

this information to the LSTM predictor that predicts the workload for the next control period based on the characteristics of the recorded past.

To use the predictor, the LSTM model is first trained using supervised learning. Training data include CPI and instruction count and are collected and organized as input features for a moving window of $w = 20$ in order for the prediction to take into consideration the past 20 data sequences. This is similar to the Kalman filter configuration used in Sec. 4.3 against which we will compare later

on. The collection process is done during separate runs of the custom Sniper simulation framework and without any DEM algorithm. The model is trained with 20,000 samples collected at intervals of 1 ms. The LSTM model itself is rather simple. It is constructed with just one hidden layer of 4 LSTM blocks or neurons and the sigmoid activation function is used for each block.

The DEM algorithm utilizes the predicted CPI and instruction count to estimate the performance loss using equation (4.10). These estimations are then used by the heuristic that decides the actual V/F pair to be used for each core in the next control period. These V/F pairs reduce energy consumption without degrading performance beyond the user set threshold. The pseudocode of the heuristic is described in Fig. 4.6. In each control period, P, the CPI and the instruction count for the current period (CPI_P, I_P) as well as for the next control period (CPI_{P+1} , I_{P+1}), as predicted by the LSTM predictor, are passed to the heuristic algorithm. The algorithm estimates the performance loss for the available frequencies listed in ascending order and selects the lowest frequency that satisfies the performance constraints and that lead to maximum energy savings.

4.5 Dynamic Energy Management using DNN

The main idea of the dynamic energy optimization approach proposed in this section is to use DNN models for prediction or classification. Note that, as in the case of many other application domains including speech recognition, pattern recognition, and recommending systems that have been revolutionized lately by the use of DNN models, the merit of this work lies in the application of the DNN

Algorithm: Dynamic Energy Management using LSTM based predictions

Input:
γ: acceptable performance loss ratio threshold; $I_{P_{done}}$, CPI_P for just ended control period
Output:
(V_{P+1}, f_{P+1}) V/F pairs for all cores for next control period

Definitions:
T each control period duration
$I_{(P+1)_{done}}$, CPI_{P+1} predicted with LSTM-based predictor
$FreqList$: list of available frequencies sorted in ascending order (f_H is the highest frequency)

$T_{delay} = 0$
$T_{ref} = 0$
if end of *control period* index P **then**
  for each *core* in CMP **do**

    $T_{ref} += T$
    $T_{delay} += \frac{I_{P_{done}} \times (\frac{f_H}{f_P} - 1) \times CPI_P}{f_H}$
    $Freq\_set = False$

    for $Freq$ in $FreqList$ **do**

      $T_{ref_{next}} = T_{ref} + T$

      $PredT_{delay} = T_{delay} +$
      $\frac{I_{(P+1)_{done}} \times (\frac{f_H}{Freq} - 1) \times CPI_{P+1}}{f_H}$

      $PredPerfLoss_{P+1} = PredT_{delay} / T_{ref_{next}}$

      if $PredPerfLoss_{P+1} < \gamma$ **then**
        $f_{P+1} = Freq$
        $Freq\_set = True$
      end if
    end for
    if $Freq\_set = False$ **then**
      $f_{P+1} = f_H$
    end if
  end for
end if

Figure 4.6: Pseudocode of theLSTM based algorithm. This algorithm is implemented as a callable routine inside our modified Sniper CMP simulator. The parameter γ is set by the user.

model to a specific practical problem rather than the DNN model itself, which has been known for decades already.

The system level block diagram of the proposed optimization framework

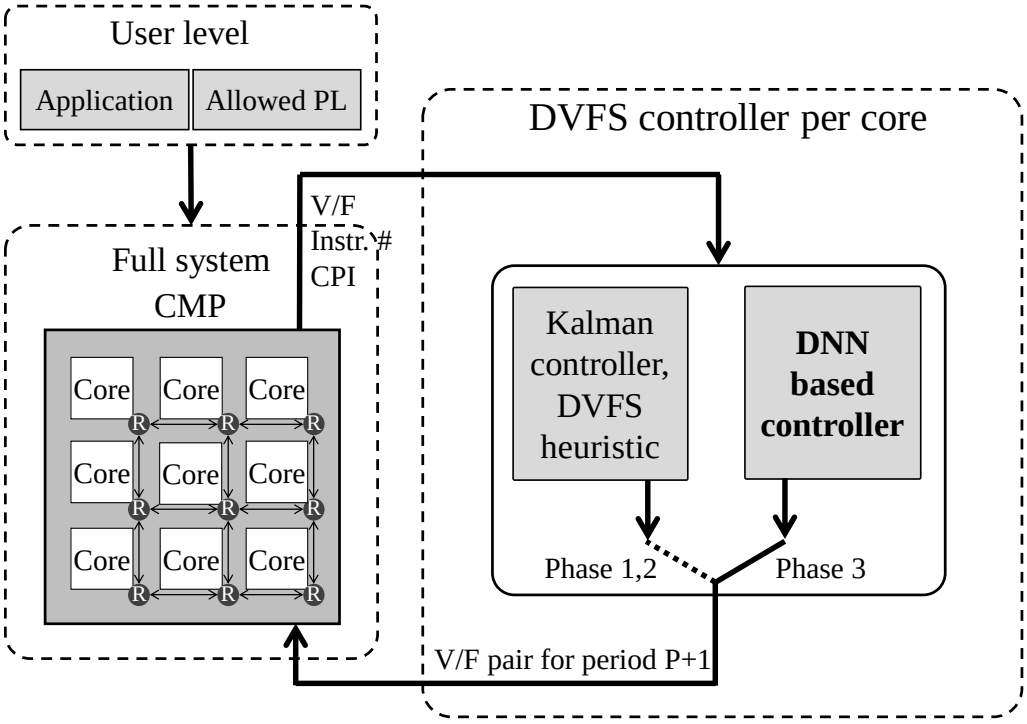


Figure 4.7: The proposed dynamic energy optimization algorithm switches to DNN based prediction once the DNN model has been constructed. The Kalman filtering based controller block operates similarly to that in Fig. 4.3.

is shown in Fig. 4.7, as it is implemented in our Sniper based full system simulation tool discussed later in Sec. 4.6. The proposed framework is implemented mainly in software and is responsible for managing all related activities, including creating, maintaining, and storing specific information about the *DNN controller*. The information about the DNN topology, related weights, as well training data represents what is denoted as *DNN data*. The primary objective of the proposed dynamic energy optimization approach is to reduce the energy consumption of the CMP. This can be achieved by throttling CMP core frequencies to the lowest possible V/F levels while meeting as much as possible the execution deadline of all executed tasks. Two of the key elements of the optimization framework include 1)

the *DNN controller* with its associated *Kalman controller* and self-learning technique and 2) the DVFS algorithm that decides the V/F levels for each of the cores for the next control period.

As discussed earlier, the *Kalman controller* is implemented with the help of a series of Kalman filters and works with a sliding window of m previous control periods. Thus, in generating a training data pair, we consider past history covering the last m control periods. According to the experiences in Sec. 4.3 and 4.4, we prefer the use of the Kalman based predictor rather than the LSTM one due to both the ease of implementation and the very good performance demonstrated in workload estimation. In addition, having in place an existing approach for dynamic energy optimization provides a way to achieve energy reductions also during the first two phases of the proposed approach, when we collect training data and train the DNN model.

The implementation of the DNN model based energy optimization algorithm includes three phases as illustrated in Fig. 4.8. In the first phase, we collect input samples (i.e., input features) and their corresponding outputs (i.e., labels) as the initial training data set. The features capture the benchmark behavior and the labels represent the V/F pairs identified to lead to energy reduction. In the second phase, the training data is used to train the DNN model. Lastly, in the last phase, the DNN model is employed to directly predict optimal V/F pairs for each CMP core for a given workload at runtime. These phases are described in more details next.

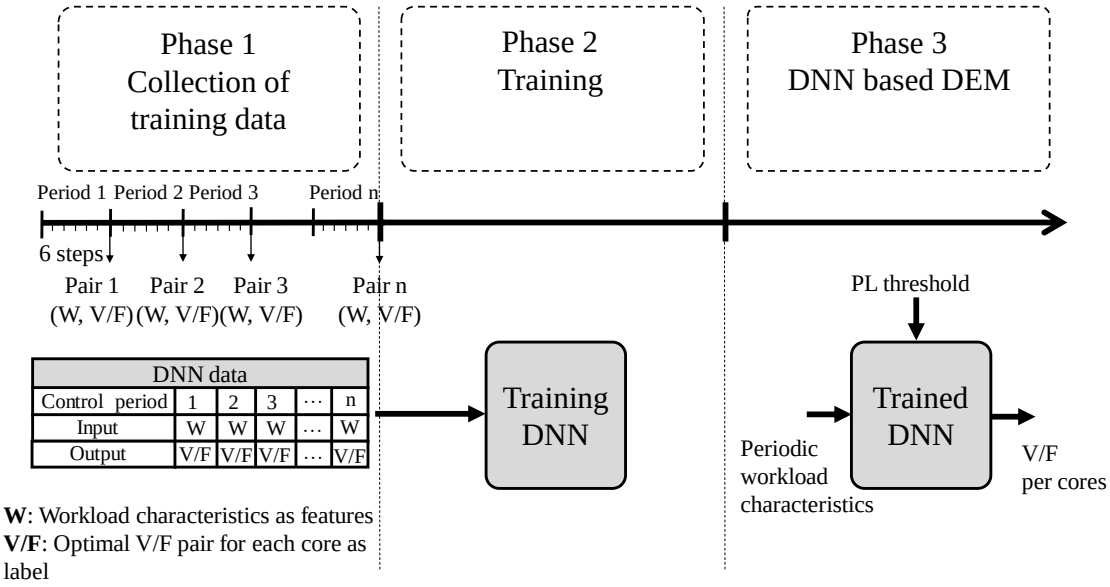


Figure 4.8: Illustration of the three phases of the implementation and usage of the DNN model.

4.5.1 Phase 1: Collection of Training Data

One of the main challenges of working with DNN models is training. This is a two-faceted challenge: first, labeled data is necessary for training and second, the training process may become computationally intensive and require long training times for increasingly large training data sets. In addition, workloads can vary greatly and developing a representative training data set is very difficult because a DNN well trained for certain workloads may perform very poorly on different workloads. To address the lack of training data when it comes to chip multiprocessors, we propose to use a new self-adaptive supervised training technique. We develop the ability to generate training data automatically in three phases as illustrated in Fig. 4.8.

Phase 1 begins when a new CMP system starts to be used in a datacenter.

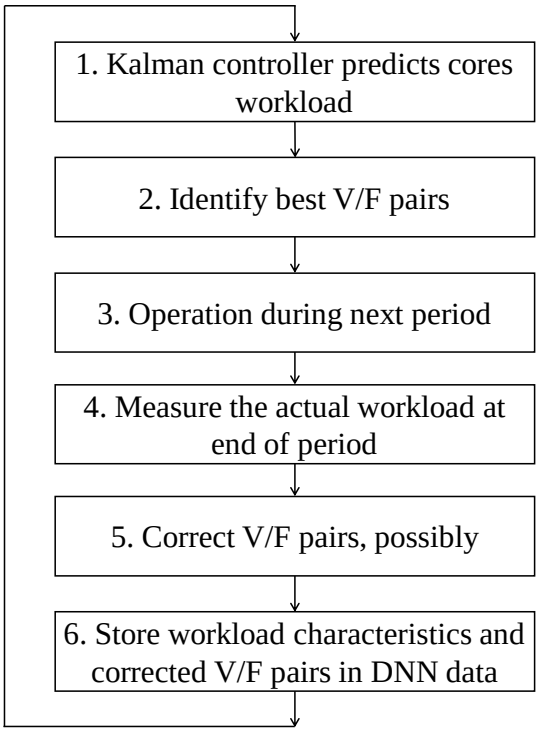


Figure 4.9: Steps of the procedure to generate one training data point during one control period in Phase 1.

This is the time when the CMP operation starts to be monitored for the purpose of generating input-output training data pairs in Fig. 4.8. The generation of training data is done for each control period. For each such control period we generate, at the end of the period, training data pairs by recording input values and the *corrected* outputs (as V/F levels) which would have been better if set at the beginning of the control period. The six steps followed to generate the corrected V/F pairs in a given control period are shown in Fig. 4.9.

1. The *Kalman controller* is used to make predictions about the workload of each CMP core in the next control period.

2. Then, equation (4.10) is used to estimate the performance loss and to identify the

lowest available V/F pairs at which energy could be saved without violating the performance loss threshold in the next control. The selection of these V/F pairs is done with the efficient DVFS heuristic algorithm from Sec. 4.3.

3. Proceed with the execution of the next control period at selected V/F pairs for all CMP cores.

4. At the end of the just executed control period, measure the actual just executed workload.

5. Repeat Step 2 but use the actual measured workload to find possible corrections to the just used V/F pairs. The corrected V/F pairs are the ones that ideally should have been identified earlier in Step 2. These corrected V/F pairs are used as outputs in the training data set because they would have helped reduce energy without violation of the performance degradation threshold.

6. The above steps are done for a moving window of m control periods in order to generate one training data point of input features and corrected V/F pairs added to the DNN data. These will be used later for the actual training of the DNN model.

It is important to note that, in theory, one could conduct the whole process of collecting training data with a set-up that does not use the Kalman filtering based prediction combined with the DVFS heuristic. Instead, one could just run the selected benchmark testcases at the default (highest frequency and voltage pair) all the time during Phase 1. While this would eliminate the need for the Kalman filtering and simplify the overall implementation of the proposed framework, the

issue would be that the training data would not be diverse and would always have as input features values that would characterize core operations at the maximum frequency all the time. When the Kalman filtering based technique is used, training data points are generated also for input features that characterize core operations at throttled frequencies as well. Therefore, the training data resulting from the proposed approach is more diverse and characterizes better the operations of all cores (among the 16 or 64) at many different clock frequencies. In addition, as already mentioned, the Kalman filtering based technique provides an alternative way to achieve energy reductions also during the first two phases of the proposed approach.

4.5.2 Phase 2: Training of the DNN Model

So far, we have developed a mechanism to collect the runtime statistics and construct the training data set. Instruction count and average CPI values together with the corrected V/F pairs have been recorded as the *features* and the *labels* of the DNN data characterizing all the control periods of Phase 1. Note that the labels are transformed into the *one hot* format before actual use. In the *one hot* format, for each class in the output we consider a digit which can be zero or one. If the label belongs to class number k, the k-th digit is set to "1" while the other digits are set to "0". The collected training data set is now used for supervised training of the proposed DNN model. The input features are passed to the feed-forward model. At each node, the weights and biases are applied to the given inputs and then the result gets activated through an activation function. We use the $RELU$ function (like that shown in Fig. 2.6) as the activation function because

it helps to mitigate the vanishing gradient problem described in [95]. The final result of the output layer is used to calculate the cost. That is, the *cross entropy cost function* compares the generated output with the stored labels (recall, these are the corrected V/F pairs) to calculate the cost based on the prediction error. The *gradient decent optimizer* uses this cost to optimize the weights and biases in the backward direction. Specifically, as the gradient decent optimizer algorithm we use the *AdaGrad* method, which was shown to give the best results [94]. This method adapts the learning rate to the model parameters and performs larger updates for infrequent parameters and smaller updates for frequent ones. Thus, it is well suited for dealing with sparse data, which we see in our case.

4.5.3 Phase 3: Prediction Using the DNN Model

Now, that we have trained the DNN model as the *DNN controller* from Fig. 4.7, we can use it in realtime to identify V/F pairs at any time. This is the phase where the role of making predictions and deciding the V/F pairs is switched from the *Kalman controller* and the DVFS heuristic to the *DNN controller*. Collection of training data can still be performed in parallel, in order to prepare for future periodic retraining of the DNN model to address application variability. However, we do not do this in this work.

4.6 Simulation Results

This section contains the simulation results for the proposed dynamic energy management techniques discussed earlier in this chapter. The results demonstrate the

Table 4.1: Architectural configuration parameters.

Parameter	Value
Technology node	45nm
Core	Intel X86
Core CPU model	Out of order (Detailed CPU)
Frequencies	2GHz downto 1GHz, with 100MHz step
VDDs	$f \geq 1.8G : 1.2V$, $1.8G > f \geq 1.5G : 1.1V$, $1.5G > f \geq 1G : 1V$
Transition latency	2000 ns
Branch predictor	2 bit counter
Reorder buffer	80-entries
L1ICache/1core	32KB
L1DCache/1core	64KB
L2/1core	256KB
L3/4cores	8MB
Network	2D regular mesh, 1 router per core
Link bandwidth	64 bits

effectiveness of the Kalman filtering, LSTM and DNN based approaches and compare them with a state-of-the-art technique [72].

4.6.1 Simulation Setup

We implemented the proposed DEM schemes described earlier inside the Sniper based CMP system simulation framework [108], which is integrated with the McPAT power calculator [104]. We conducted extensive simulations on several Parsec and Splash2x benchmarks [109] to investigate the performance of the proposed algorithm. In our simulations, we used two different network-on-chip based CMP architectures composed of 16 (4x4) cores and 64 (8x8) cores. Each of these architectures uses a regular mesh NoC topology. The default architectural configuration parameters utilized in our custom Sniper based simulations are listed in Table 4.1.

4.6.2 Kalman Filtering based Technique

In the DEM approach discussed earlier in Sec. 4.3, our primary prediction technique uses Kalman filtering. This is used to predict the instruction count and the average CPI in the next control period for each core. Based on our simulations, we found empirically that a history window of 20 periods and values of 1, 1, 0.5, and 0.5 for A, H, Q and R filter parameters provided consistently very accurate predictions. Also it is assumed that there is no control input exists and B is set to 0. Therefore, we use these parameter values for all our simulations unless stated otherwise. For example, in Fig. 4.10, we show the predictions of both the CPI and the instruction count for a sample core while running the *radiosity* benchmark with 64 threads on a 64 core CMP architecture. The predicted values follow very closely the actual observed values, which turn out to occur at the end of the next control period. Therefore, we conclude that the Kalman filtering based prediction is very effective and computationally efficient. It serves well for our purpose, as it will be made clear in the next set of simulation results.

As mentioned earlier, this approach uses a Kalman filter per core for scalability but most importantly for accuracy because we are interested in doing DVFS at per core level rather than for the whole processor. The Kalman filter is implemented as a C++ routine, which is integrated inside the Sniper system simulator that we use for our simulations. Whenever the Kalman filter needs to be executed, this routine is called and it executes very fast. Its runtime is very small at less than 0.05% of the duration of a control period. This performance overhead is included in the measurement of the total execution time for a given benchmark. Thus, the

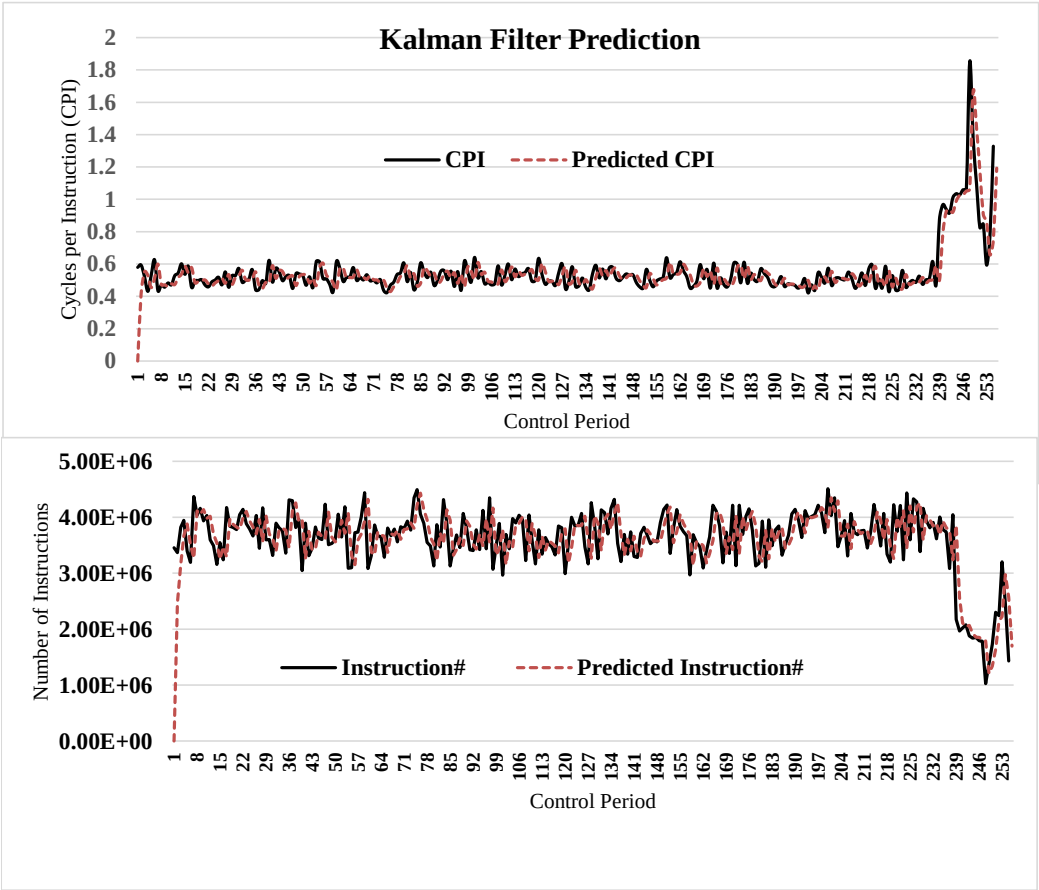


Figure 4.10: Plots that show the comparison between the predicted values of the CPI and the instruction count for the next control period and the actual values that occurred and were observed at the end of the next control period. These traces are for a sample core (out of 64 cores) during the execution of *radiosity* benchmark.

performance for a given benchmark when the proposed DEM is used includes also the overhead due to the execution of the Kalman filter routines.

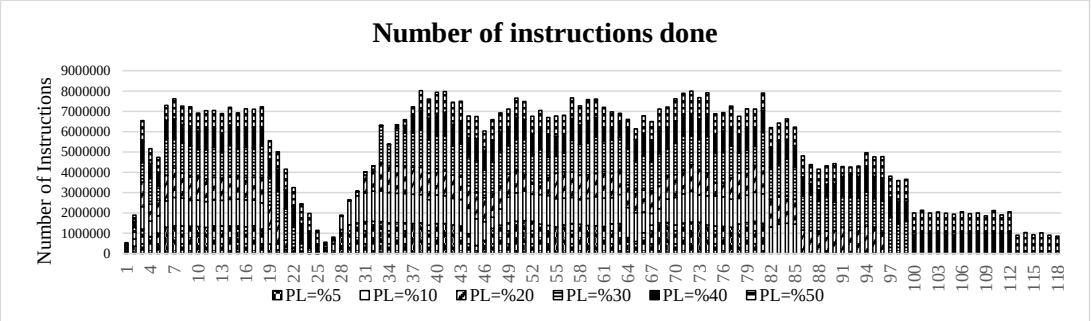
In the next set of simulation experiments, we investigate the performance of the proposed algorithm for several different values of the performance loss constraint or threshold. Recall that this constraint is set by the user who has the best knowledge about what is the acceptable performance degradation for a given application. We assume that the user sets this constraint based on her knowledge of the criticality of the application at hand. The higher the criticality, the lower

the PL should be selected. We considered six different PL values including 5%, 10%, 20%, 30%, 40% and 50%. For example, a value of $PL = 5\%$ means that the user wants the proposed algorithm to try to reduce energy consumption as much as possible but without incurring a performance degradation of more than 5%.

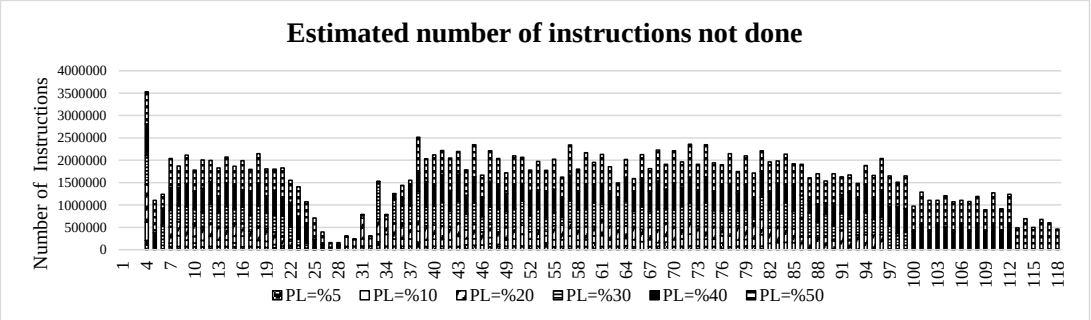
Therefore, for each such PL value, we run the proposed DVFS based dynamic energy management algorithm to find out what is the maximum achievable energy reduction under the specified performance degradation constraint. The results reported here focus on ROI during the execution of a given benchmark. During each simulation, the custom Sniper simulator is stopped periodically after a constant amount of time (i.e., control period) and the proposed DEM algorithm described in Fig. 4.5 is called to find the best V/F pairs for all cores for the next control period. In our simulations the control period is set to $1ms$, but it can be set to other values as well. During each such stop, the DEM algorithm estimates the delayed instructions count in current control period and predicts the behaviour of the application on each core for the next period using Kalman filter predictor. Then, it tries to control the delay incurred due to the delayed instructions by selecting the lowest possible frequency, which still satisfies the acceptable performance penalty threshold.

The plots in Fig. 4.11 show simulation data collected during a sample run of the *barnes* benchmark on a 64 core architecture for the four different PL constraints. The plot in Fig. 4.11.a shows the number of instructions executed during each control period on one of the 64 cores of the CMP architecture. Please contrast that with the predicted number of instructions that are delayed for later

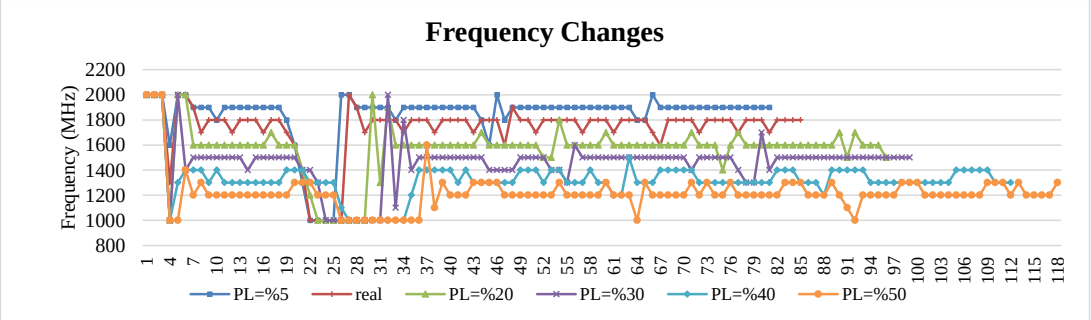
execution shown in Fig. 4.11.b. This is the number of instructions that could have been executed in addition during the just completed control period, if the highest frequency had been used. In other words, because the execution in the just completed control period was done at a lower frequency (as dictated by the DVFS algorithm, in order to reduce energy consumption), these instructions are delayed and thus their execution will be "rolled over" during the incoming control periods. This plot is as we expected; the larger the threshold for performance loss, the larger the number of instructions that are not completed and postponed for later. This in turn will result in longer overall execution time for a given benchmark. The way frequency was varied is shown in Fig. 4.11.c while the calculated performance loss at the end of each control period is shown in Fig.4.11.d. We can see that for a tight PL threshold like 5%, the core frequency is higher than when the threshold is large. In other words, for example, when the PL thresold is relaxed to say 50%, the algorithm pushes the frequency way down in order to save as much energy as possible while trying to keep the estimated performance loss within the limit of 50%, as seen in the top curve of the plot in Fig. 4.11.d. Similar plots can be collected for any of the 64 cores of the CMP architecture and for any of the simulated benchmarks. Noteworthy, we observe that sometimes the estimated performance loss overshoots as shown by the curve corresponding to the PL threshold of 50% in Fig. 4.11.d. This happens when the frequency for the just completed control period was selected too low. As a result the performance degradation violates the desired threshold for a short period of time. This is a direct result of the prediction error that was experienced at the end of the previous control periods. We do not have currently a way to eliminate these "artifacts",



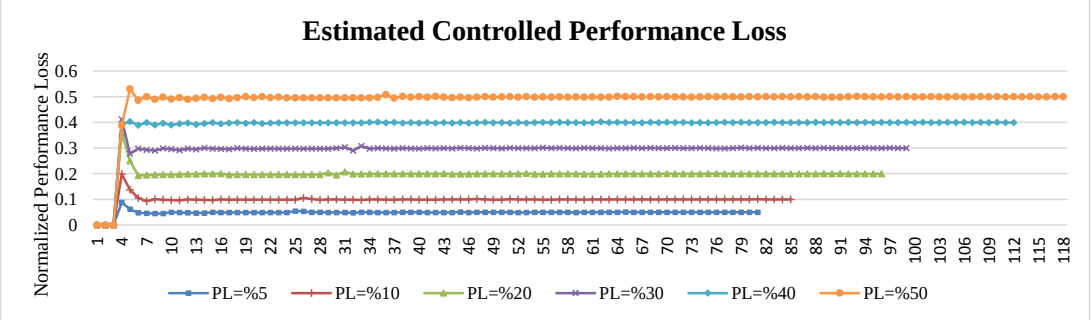
(a)



(b)

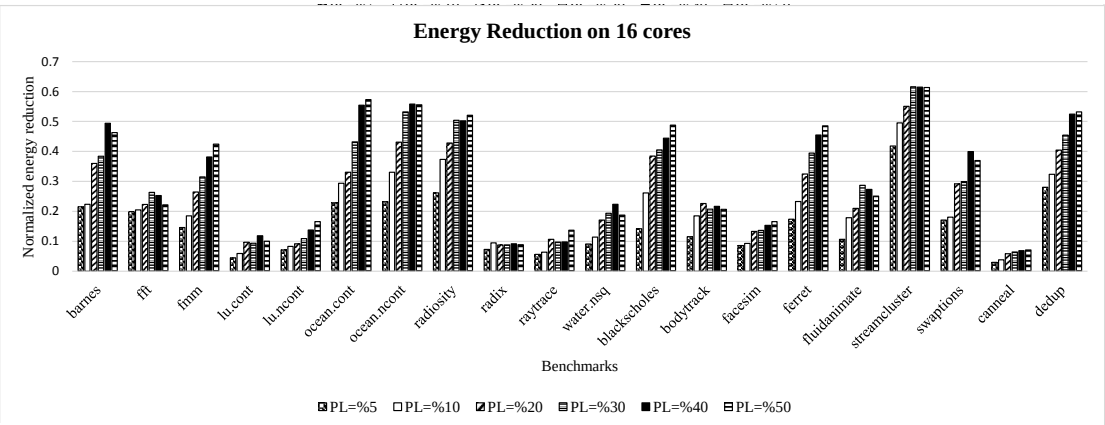


(c)

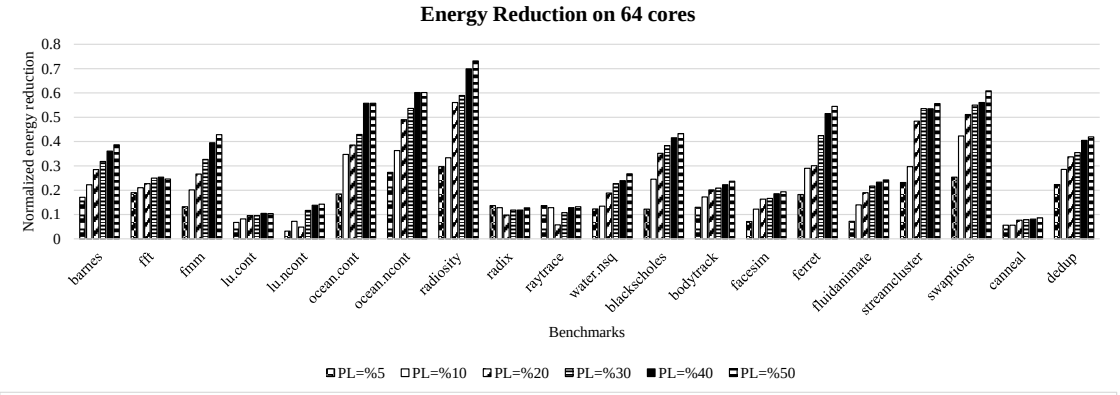


(d)

Figure 4.11: Simulation results for a sample run of the *barnes* benchmark. The x axis represents the index of the control periods. Note that when the frequency is higher, the total execution time, measured as *walltime*, is shorter and therefore the total number of control periods is smaller.



(a)

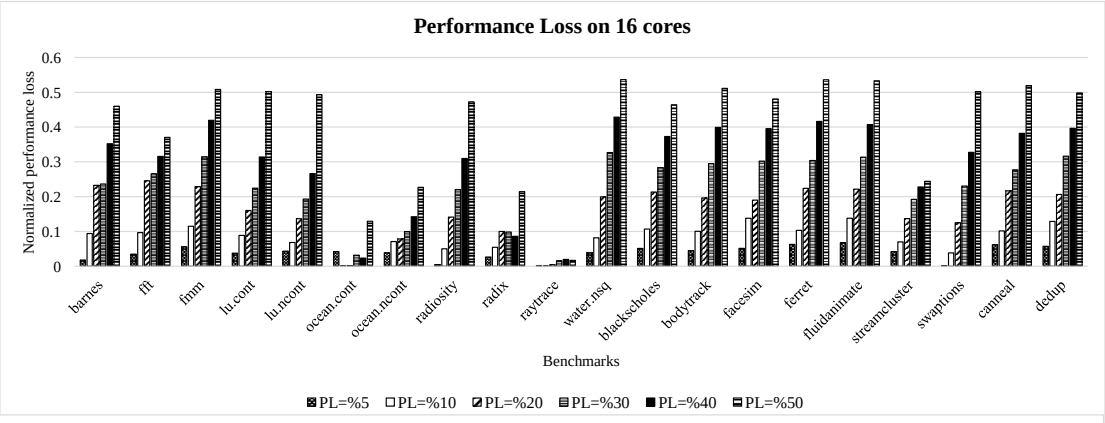


(b)

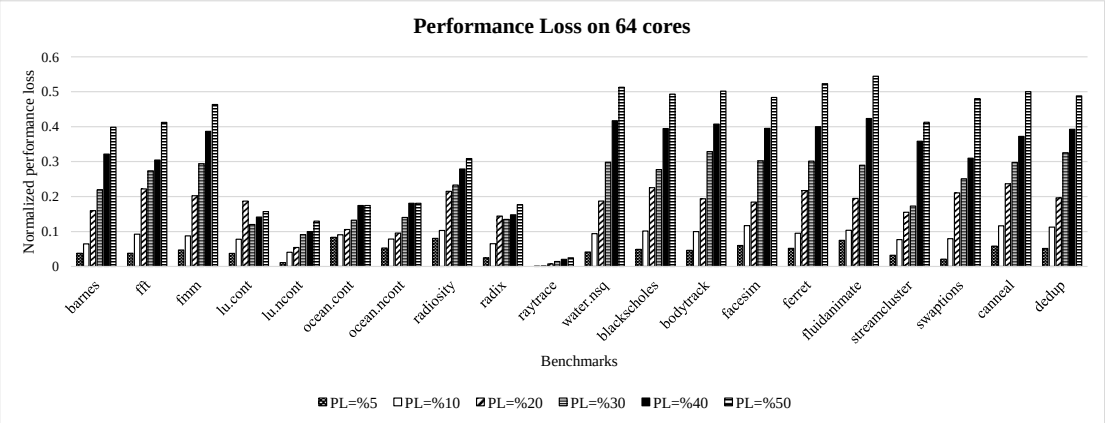
Figure 4.12: Energy reduction percentages. (a) 16 core architecture with 4x4 mesh NoC. (b) 64 core architecture with 8x8 mesh NoC.

unless we wanted to become over-conservative in the way we allow the proposed DVFS algorithm to throttle core frequencies. Therefore, we consider these short lived PL threshold hikes as acceptable.

The energy reduction for all the benchmarks that we investigated is shown in Fig. 4.12 for two different CMP architectures. The y axis of these plots shows normalized values for simplicity and clarity. For example, a value of 0.2 on the y axis of Fig. 4.12.a means a 20% energy reduction. We can see that energy savings are consistent across the board and, as expected, the savings are bigger when the performance loss constraint is more relaxed. Note that for some benchmarks the



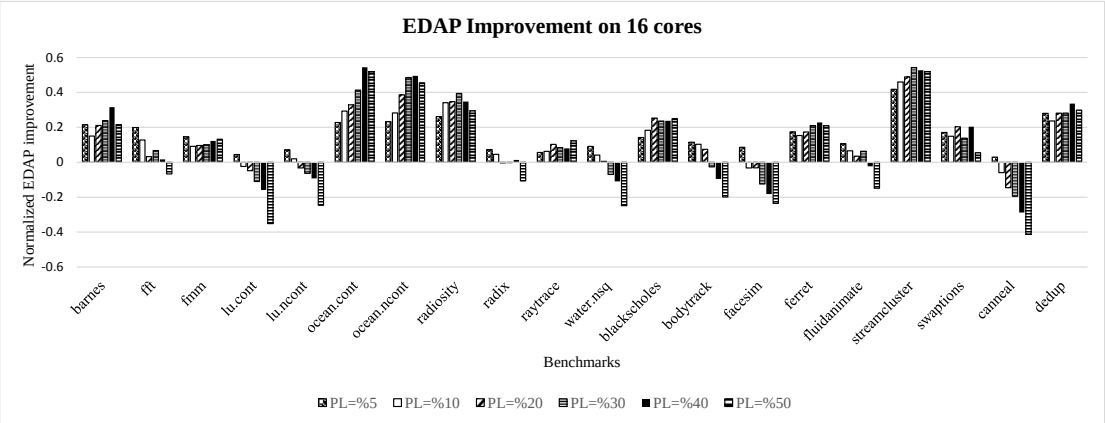
(a)



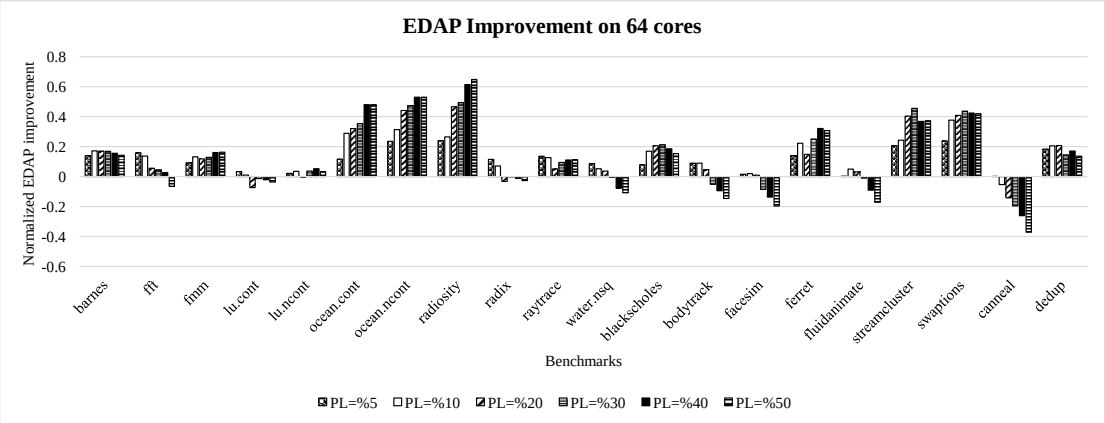
(b)

Figure 4.13: Performance loss percentages. (a) 16 core architecture with 4x4 mesh NoC. (b) 64 core architecture with 8x8 mesh NoC.

energy savings can be as high as 60% for either of the two CMP architectures. The performance loss for all benchmarks and for both CMP architectures is shown in Fig. 4.13. Please note that the plots in this figure actual performance loss as calculated and reported by the Sniper tool simulator, and it includes also the overhead of executing the Kalman filter routines. Again, the y axis shows normalized values similar to the plots in Fig. 4.12. Note that for each of the four different values for the PL threshold, the actual performance loss calculated at the end of the execution of each benchmarks is kept within limits reasonably well. Some benchmarks experience slightly larger performance degradation and that is due to



(a)

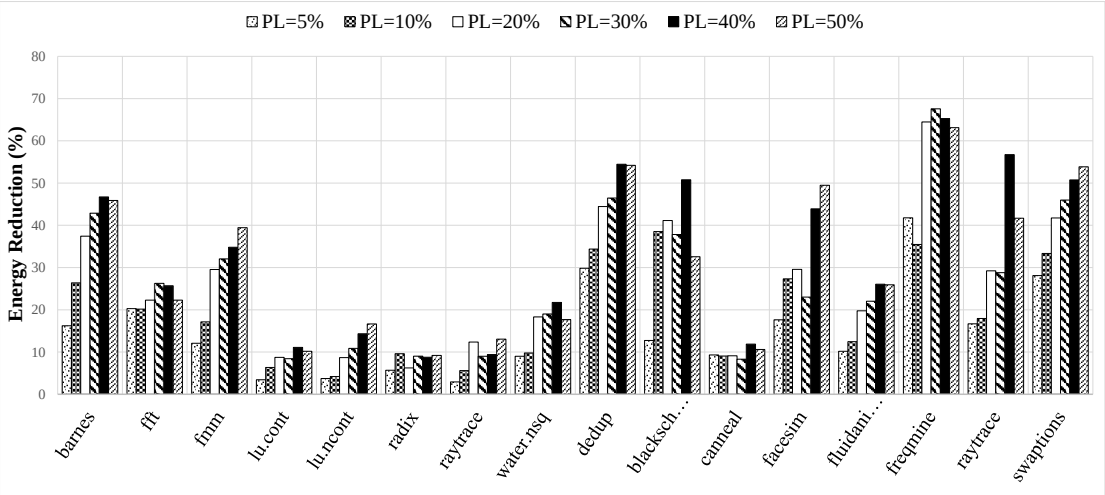


(b)

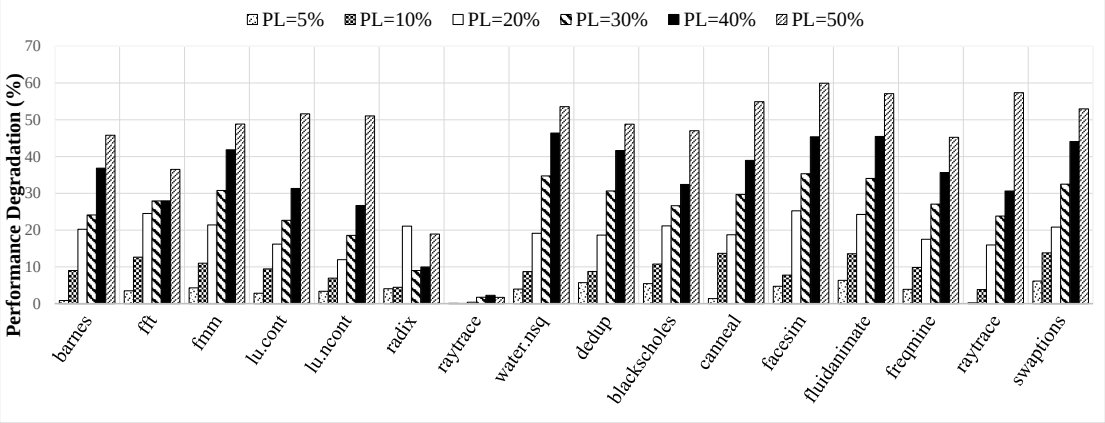
Figure 4.14: Energy Delay Area Product (EDAP) percentages. (a) 16 core architecture with 4x4 mesh NoC. (b) 64 core architecture with 8x8 mesh NoC.

the “artifacts” discussed earlier.

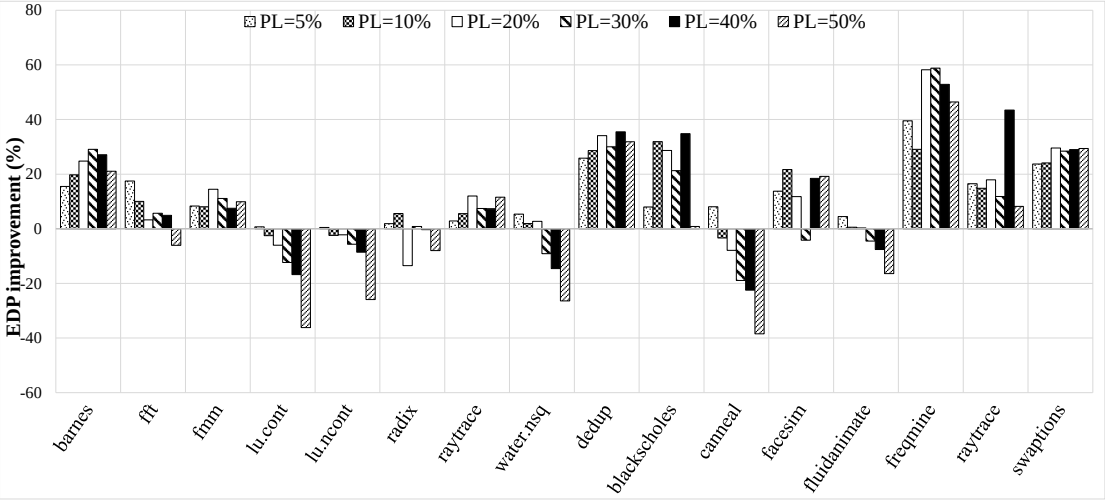
Finally, we also show in Fig. 4.14 the change in energy delay area product (EDAP). In all cases the area is actually the same and does not affect these plots. So, in the figures that compare the EDAP with other implementations, the term EDP can be considered as well.



(a)

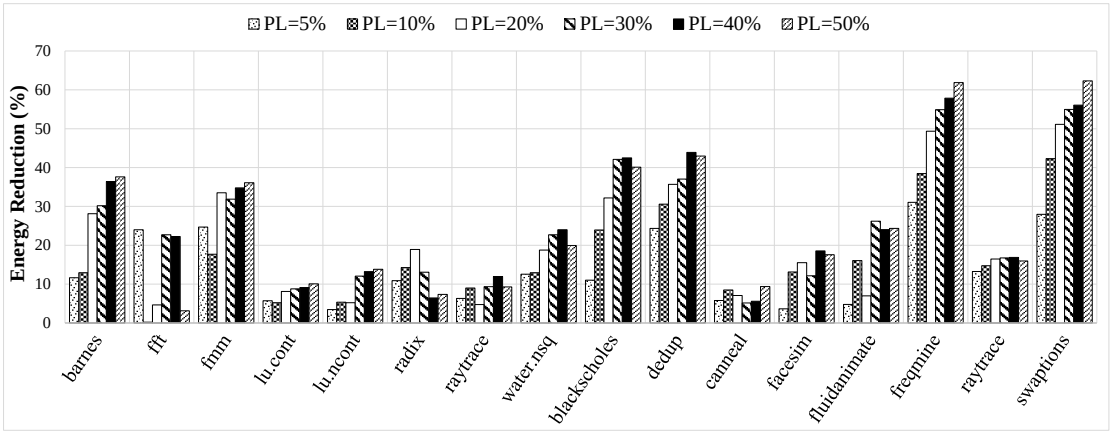


(b)

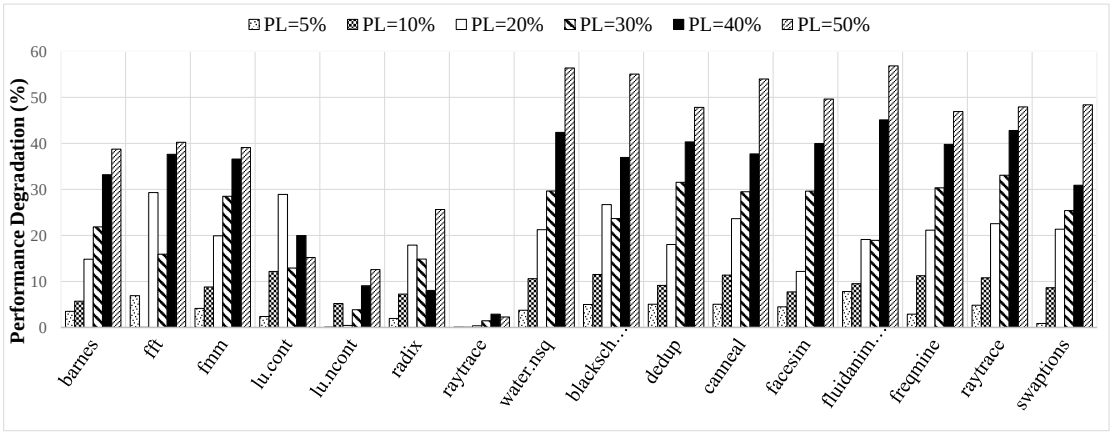


(c)

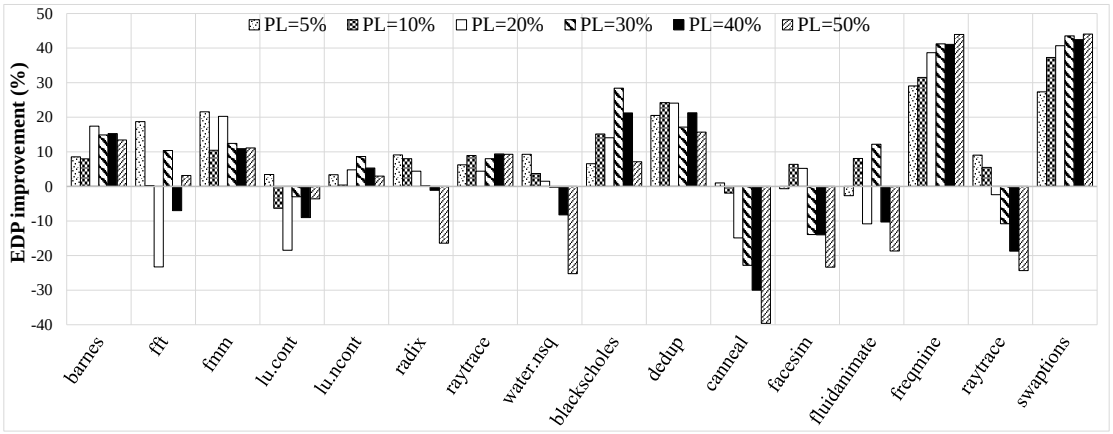
Figure 4.15: Simulation results for the 16 core CMP: (a) energy reduction percentages, (b) performance degradation percentages, and (c) EDP improvement percentages. Comparison is done versus the case when no DEM is used.



(a)



(b)



(c)

Figure 4.16: Simulation results for the 64 core CMP: (a) energy reduction percentages, (b) performance degradation percentages, and (c) EDP improvement percentages. Comparison is done versus the case when no DEM is used.

4.6.3 LSTM based Technique

In order to simulate the LSTM approach, we integrated the machine learning library Keras [110] in our simulation framework and employed it to build and train the LSTM predictor. To speed-up the training process, we take advantage of the acceleration provided by a K40c Tesla GPU that we have available on the workstation with an eight core processor that runs Linux Ubuntu 16.04. The simulation framework is implemented such that Sniper is stopped periodically (1 ms intervals) and the algorithm from Fig. 4.6 is called as a routine that finds the V/F pairs for each core for the next control period.

The dynamic energy management algorithm described in Fig. 4.5 is investigated for several different application criticality levels, indicated as the tolerable performance loss percentage. Specifically, similar to Sec. 4.6.2, we focus on six different PL values including 5%, 10%, 20%, 30%, 40%, and 50%. For example, if the user sets a value of $PL = 20\%$, it means that the objective of the DEM algorithm is to reduce the energy consumption as much as possible without degrading the performance by more than 20% compared to the case when no DEM is implemented and all cores operate at the highest clock frequency all the time.

First, we compare the DEM algorithm to the case when no DEM algorithm is used at all. Fig. 4.15 and Fig. 4.16 show the results for the energy reduction, the total performance degradation, and the EDP on the selected benchmarks for 16 and 64 core CMP architectures. These plots show that while the DEM algorithm reduces the energy consumption, it keeps the total performance loss under

Table 4.2: Average EDP improvement of data from Fig. 4.15.c and Fig. 4.16.c

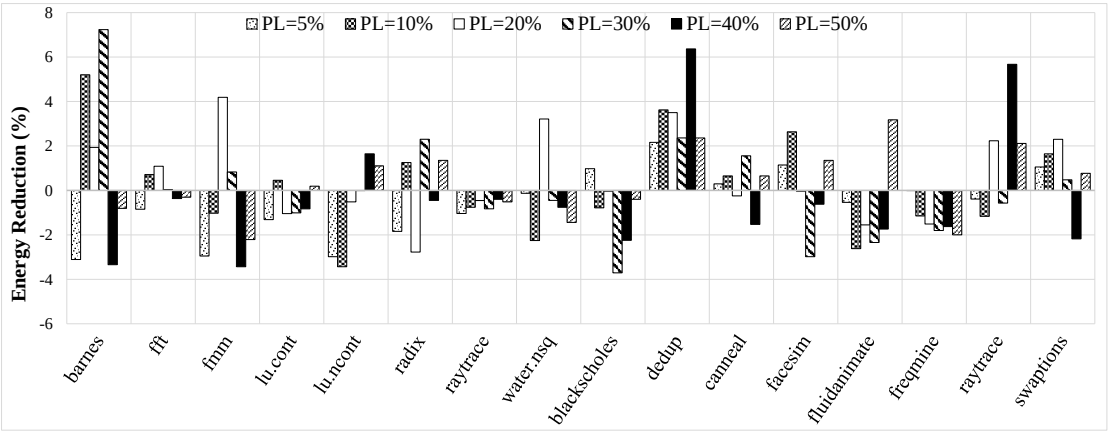
PL	**Avg. EDP improvement 16 core (%)**	**Avg. EDP improvement 64 core (%)**
5%	12.02	10.66
10%	12.08	9.98
20%	13.01	6.60
30%	9.36	9.14
40%	11.92	4.29
50%	1.31	-0.02
Avg.	9.95	6.77

the desired threshold fairly well. The energy savings increase as the tolerable performance degradation is increased suggesting that the DEM algorithm provides a good mechanism to trade off performance versus energy consumption. However, for some benchmarks the performance degradation is slightly larger than expected. This is due for the most part to the prediction errors of the LSTM based predictor.

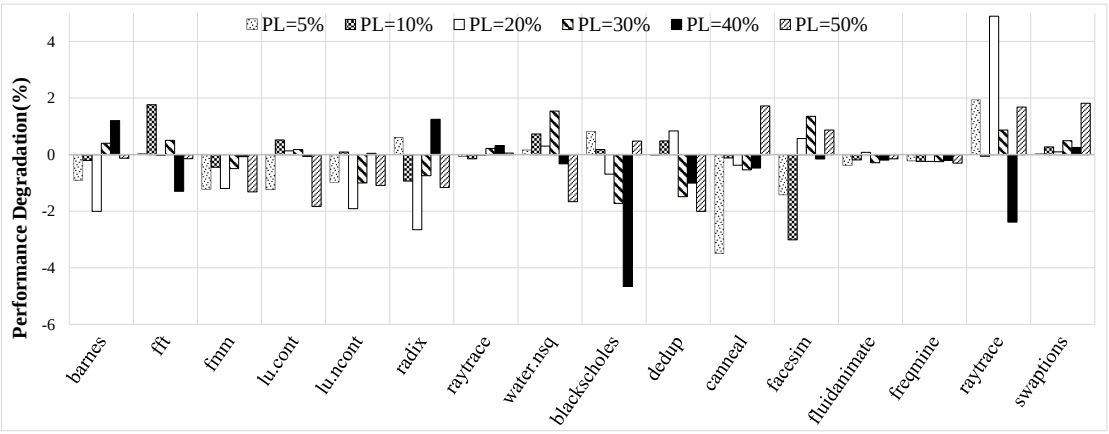
Next, we compare the DEM algorithm against the algorithm presented in the Sec. 4.3, where Kalman filtering was used as the prediction technique. Fig. 4.17 and Fig. 4.18 compare the energy reduction, the total performance degradation, and the energy delay product. The EDP data points are also summarized in Table 4.3. We note however that when we use the LSTM technique, the results are generally slightly and not significantly better that the case when Kalman filtering is used for prediction purposes.

4.6.4 Dynamic Energy Management using DNN

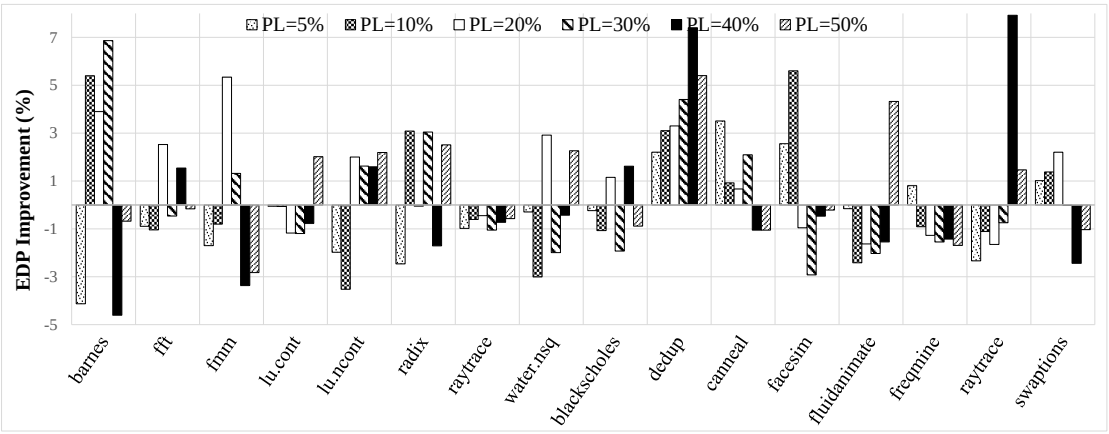
In our Sniper based simulation framework we have implemented three energy optimization approaches: the reinforcement learning (RL) approach described in [72],



(a)

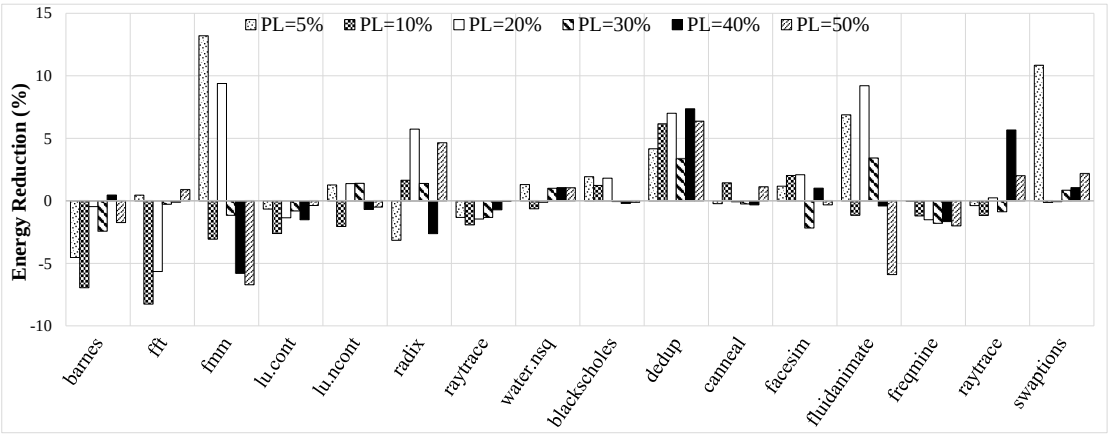


(b)

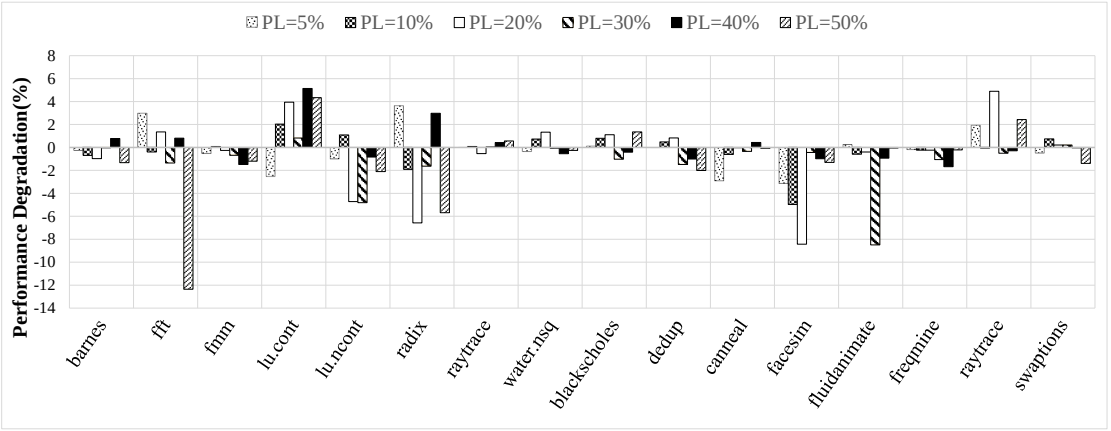


(c)

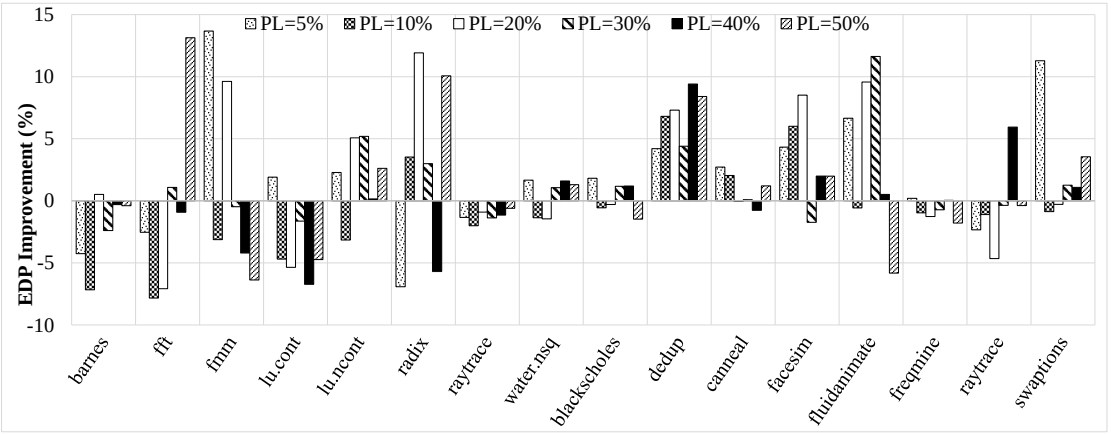
Figure 4.17: Simulation results for the 16 core CMP: (a) energy reduction percentages, (b) performance degradation percentages, and (c) EDP improvement percentages. Comparison is done versus the DEM algorithm that uses Kalman filtering for prediction from Sec. 4.3.



(a)



(b)



(c)

Figure 4.18: Simulation results for the 64 core CMP: (a) energy reduction percentages, (b) performance degradation percentages, and (c) EDP improvement percentages. Comparison is done versus the DEM algorithm that uses Kalman filtering for prediction from Sec. 4.3.

Table 4.3: Average EDP improvement of data from Fig. 4.17.c and Fig. 4.18.c

PL	**Avg. EDP improvement 16 core (%)**	**Avg. EDP improvement 64 core (%)**
5%	-0.32	2.08
10%	0.31	-0.94
20%	1.05	1.95
30%	0.34	1.26
40%	0.10	0.14
50%	0.69	1.29
Avg.	0.36	0.96

the Kalman filtering approach from Sec. 4.3, and the proposed DNN model based approach. This makes the collection of simulation results easier and all the comparisons consistent because all simulations are done within the same simulation tool and on exactly the same benchmarks. The simulation framework includes all the functions to implement the three phases of the proposed energy optimization approach. For training the DNN model, we employ Google's Tensorflow machine learning library [111]. All simulations are conducted on a Linux Ubuntu 16.04 machine that runs on an Intel Xeon eight core processor equipped with a K40c Tesla GPU.

As discussed, the objective of the optimization algorithm is to minimize energy under user set performance constraints. The user can set such constraints based on known or assumed application criticality levels, which translate into acceptable performance losses. For example, a video streaming application could be categorized as high criticality while an email application can be treated as a rather low criticality level in the sense of expected response or execution time. In this context, a certain criticality level can be assigned a performance loss threshold or constraint. In this implementation, for simplicity, we assume the same criticality

for all simulated benchmarks, by setting the PL threshold to $PL = 10\%$. This threshold can easily be changed in our framework. A PL threshold of $PL = 10\%$ means that the user wants the proposed algorithm to save as much as energy possible but without degrading the performance of the application with more than 10% compared to the case when no energy optimization was done. Thus, the proposed DNN model should ideally be able to suggest the best set of V/F pairs for all cores to ensure energy reduction but within the acceptable performance degradation. To achieve that, Phase 1 discussed earlier in Sec. 4.5.1 must first be done for the given PL threshold such that the training data is collected for that PL. Then, in Phase 2, the training data is used to train the DNN model, which will then be plugged in into the DNN controller used to proactively provide V/F settings to all cores during all control periods within the execution time of the application.

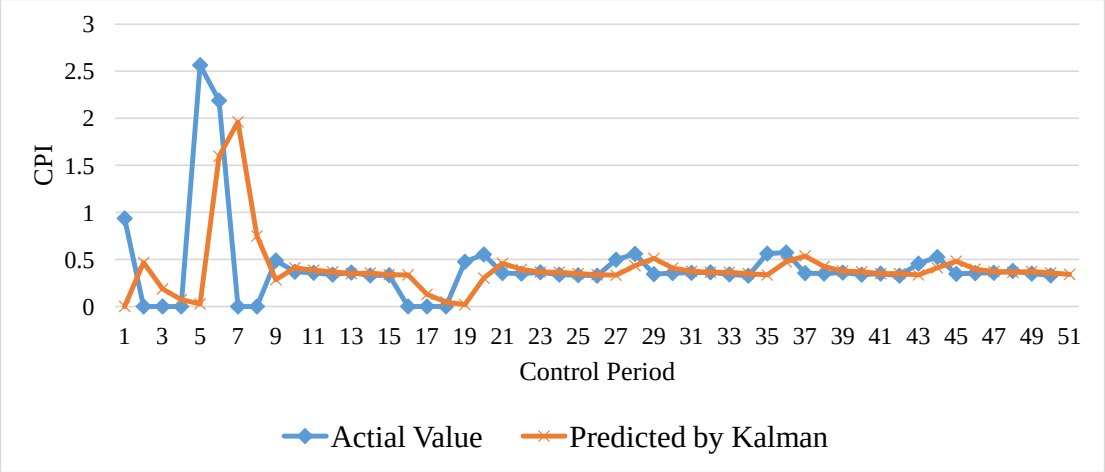
4.6.4.1 Phase1: Collection of Training Data

We have implemented all six steps discussed earlier in Sec. 4.5.2 in the custom Sniper simulator, which is paused during each control period for the purpose of collecting training data points. During Phase 1, we use the Kalman filtering based prediction, as illustrated in Fig. 4.7. We used half of the benchmarks, selected arbitrarily, for collection of training data. But, only 70% of that training data is actually used for training; the remaining 30% is used for model testing and validation. The Kalman filtering technique is used to predict the instruction count and the average CPI for each core of the CMP architecture during each control period. Similar to the Kalman filtering based DEM approach we described in Sec. 4.3, we use the same values for the filter parameters: $A = 5$, $H = 1$, $Q = 1$, $R = 0.5$

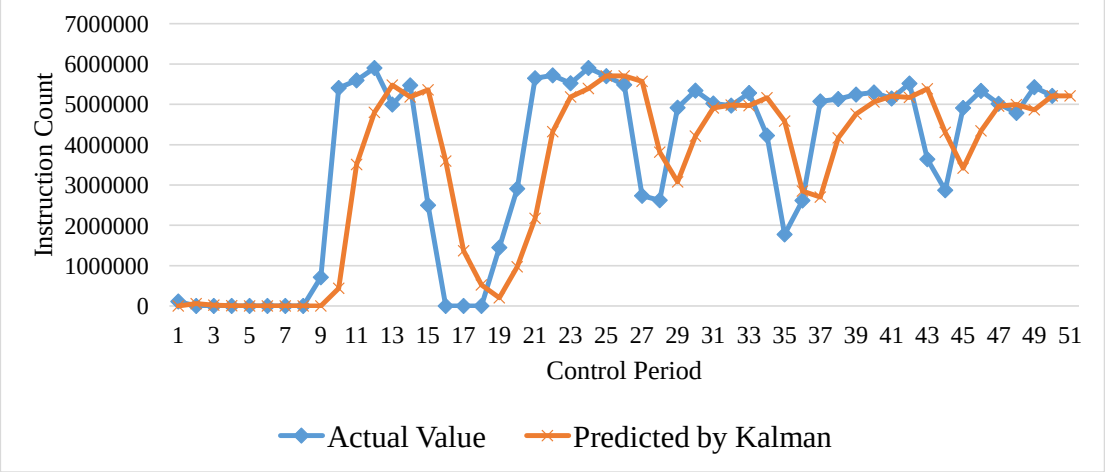
and $B = 0$. These Kalman parameters were found to provide good results.

For example, Fig. 4.19 shows the values of the CPI and the instruction count predicted by the Kalman filter as well as their actual values for a sample core while running the *fmm* benchmark with 16 threads on a 16 core CMP architecture and 10% PL constraint. These are values predicted during each control period in step 1 above. Note that, based on the results demonstrated in Sec. 4.6.2, Kalman filtering provides excellent prediction accuracy, while it has no need to be trained similar to the LSTM based approach (Sec. 4.4), which is the reason we use it for collection of training data as well as for comparison. The Kalman filtering prediction does not perform very well though during abrupt changes of the predicted variable. The corresponding frequency values calculated in step 2 from Fig. 4.9 are plotted in Fig. 4.20, which also shows the adjusted or corrected frequency values.

It is the corrected values that are then used as labels together with performance counters of the cores, caches, memory, and NoC to create training data points. Recall that a training data point is constructed with input features for a moving window of m past control periods. In our simulations, we use a value of $m = 5$ to always capture the workload behavior of the 5 past control periods. However, this parameter can be changed by the user. In this example, during each control period, we collected 62 performance counters plus the frequency value for which the counters values were generated. These performance counters include statistics from CPU performance counters, different levels of caches, stalls, uncore memory accesses, TLBs, branch predictors, ALU activities including integer,



(a)



(b)

Figure 4.19: CPI and instruction count values collected during step 1.

floating point multiply/divide operations and others that are available inside the Sniper simulator. In summary, each training data point (saved in the DNN data) includes a vector of 5x63 values as the input feature plus one value as the output label; that is a total of 316 values that required roughly 2.5KB per core. Thus, we need 16x2.5KB=40KB and 64x2.5KB=160KB of memory for the 16 core and 64 core CMP architecture, respectively.

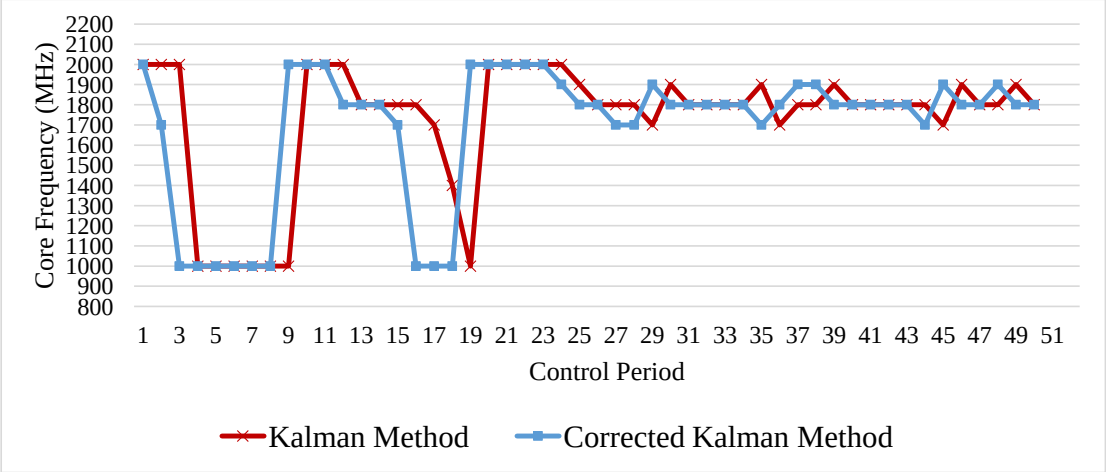
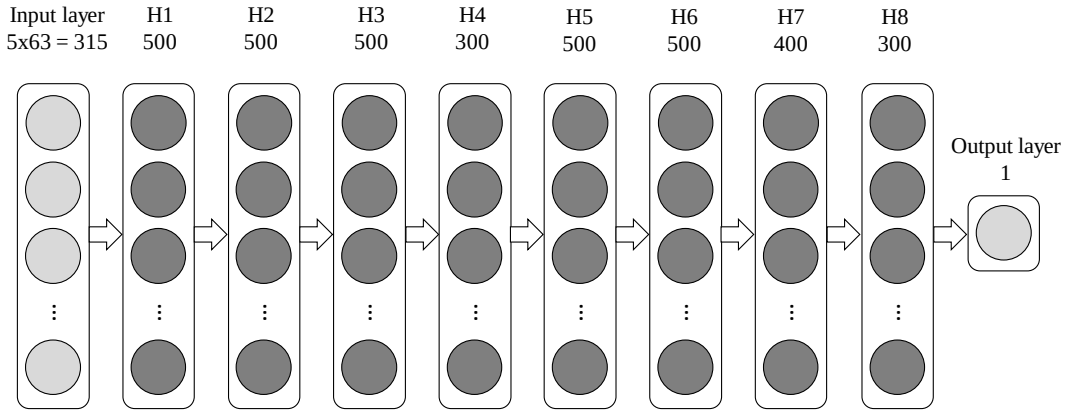


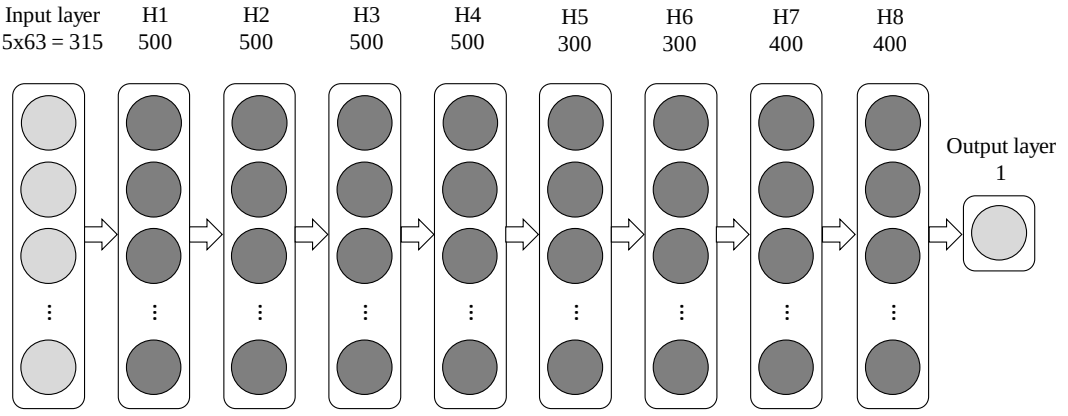
Figure 4.20: Frequency values calculated in step 2 from Fig. 4.9.

4.6.4.2 Phase2: Training of the DNN Model

At this point, we needed to decide about the exact topology of the DNN model. The previous literature does not provide helpful recipes in terms of how one should size-up a DNN model. Most often, previous literature just reported the exact DNN topology without further elaborations. In our case, we conducted a design space exploration type of search to identify the topology for the DNN model that provided the best results for a few selected benchmarks. We started with one hidden layer and increased the number of layers until no further improvement was noticed. For a given number of layers, we varied the number of units per layer as 300, 400, or 500. At the end of this search, we have found empirically that an eight layer DNN model was a good topology that provided good results, yet it is manageable in terms of training times and required storage. The final selected DNN models are shown in Fig. 4.21. While not necessary, we found that conducting a separate search to identify the topology for the DNN model for any new CMP architecture, leads to slightly better results.



(a)



(b)

Figure 4.21: Topologies of the DNN models for a) 16 core CMP architecture and b) 64 core CMP architecture.

Once the DNN models were selected, training was done using the training data set collected as described in the previous section. Tensorflow generated the DNN model (i.e., information about the network topology, number of hidden layers, number of units on each layer, and all weights), which used about 2 MB of memory. During training, we used a learning rate of 0.001 and a number of training steps of 2000. The trained DNN model provided about 80% accuracy on the testing and validation data set, which contained 30% out of the collected training data set. This phase required about 15 minutes for the 16 core CMP architecture and 1.5 h

for the 64 core CMP architecture.

4.6.4.3 Phase3: Runtime Prediction using the DNN Model

Once the DNN model is trained, we are ready to evaluate its performance. This corresponds to Phase 3 in Fig. 4.8. Essentially, the DNN model is used directly to identify V/F pairs for all cores during each control period during the execution of a given benchmark. Evaluation of the DNN model is pretty fast on the machines we used in our simulations. The overhead of finding the V/F using the DNN is in the order of milliseconds, which should not be an issue in practice where applications run for much longer times (or even continuously) in datacenter servers. In such cases, the control period would be selected much longer too, compared to the region of interest duration that system simulators like the one used in this paper focus on.

As an initial test of the DNN model, we compare its V/F predictions to those computed using the Kalman filtering based heuristic for the *fmm* benchmark testcase. Fig. 4.22 shows some of the results of this comparison, corresponding to only one of the cores of the 16 core CMP architecture. We observe that, the DNN model is good enough at predicting the right frequencies (i.e., V/F pairs).

In a first set of simulations, we compare the proposed DNN model based dynamic energy management algorithm to the case when no DEM algorithm is used at all. The results of this comparison are reported in Fig. 4.23 and Fig. 4.24 for the two CMP architectures. These figures report percentages in energy reduction, in total performance (i.e., benchmark execution time) degradation, and

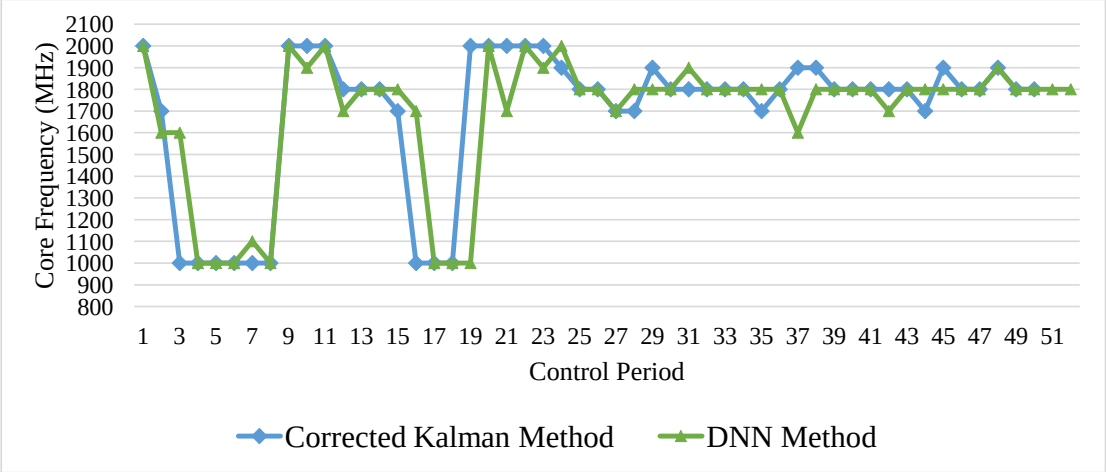


Figure 4.22: Comparison of the predicted frequencies by the DNN model to those calculated by the Kalman filtering technique.

in energy-delay-product improvement. In the second set of simulations, we compare the proposed DNN model based approach to the reinforcement learning approach described in [72] and the Kalman filtering approach proposed in Sec. 4.3. We selected to work with a moving window of control periods of length $m = 5$ for Kalman filtering and DNN model based approaches. The results of this comparison are reported in Fig. 4.25 and Fig. 4.26 for the two CMP architectures.

4.7 Discussion

Looking at Fig. 4.12, Fig. 4.13 and Fig. 4.14 in Kalman filtering approach and Fig. 4.15 and Fig. 4.16 in LSTM approach and Fig. 4.23 and Fig. 4.24 in DNN approach, we note that in the majority of cases, the proposed DEM techniques achieve significant energy reduction while keeping the total performance loss under the user specified performance constraint fairly well. Nevertheless, the performance degradation is slightly larger than expected in some benchmarks. We attribute this

due to the fact that in our predictor engines the misprediction is inevitable in some cases and also in the DNN approach the DNN model is not a perfect oracle. In simulation results in all the proposed techniques, we can see that in the majority of the benchmarks the EDP is improved. In some difficult instances, that is not the case. The reason for that is because we apply the proposed algorithm and report results only for the region of interest of a given benchmark and not for the whole duration of execution. These benchmarks are created such that during the ROI all cores are fully utilized and thus usually there is little room for improving performance via frequency throttling when everything is busy almost all the time. In experimental setups like this, it is unlikely that both energy consumption and performance can be improved because all cores are working all the time. This is not so in some previous works, which reported wishfull energy consumption reduction and performance improvement at the same time. We suspect that is possible only if simulations are monitored outside the ROI where some of the cores do not have threads scheduled and thus one can find room for execution optimization. This is something that is actually unclear in previous works, which do not discuss whether their results are reported for ROIs or not. Note that, in our simulations, the length of each control period is 1 ms. We are forced to work with such a small control period because the total length of the ROI of the benchmarks that we use in simulations is relatively short. However, this parameter would be changed to larger values in real-life deployment where workload benchmarks are executed continuously or for very long times and not for just tens or hundreds of ms that is the typical length of the ROI in full system simulators like Sniper and Gem5. Note that such short ROI simulations on these simulators take hours or days to be

accomplished.

Another interesting aspect is that beyond a PL threshold of 40% the EDP is not improved anymore as seen in Table 4.2. In other words, the DEM algorithm can offer benefits only when the PL threshold set by the user is less than 40%; beyond that, the performance degrades too much compared with how much energy is saved.

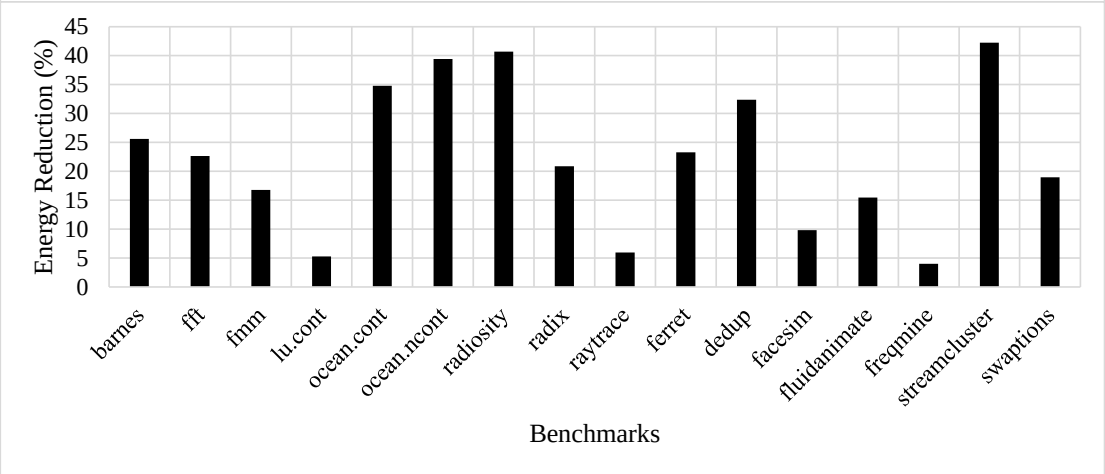
Comparing the Kalman filtering and LSTM techniques in Table 4.3, we see that both techniques demonstrate almost similar results. However, as we mentioned earlier, due to the ease of implementation and no training requirement, the Kalman filtering technique has been preferred among the LSTM based approach in our work. Looking at Fig. 4.25 and Fig. 4.26, we note that the proposed DNN model based approach provides consistently better energy delay product values than both our proposed Kalman filtering based technique and the RL technique described in [72]. On average, the EDP improvement is 6.3% and 6% for the 16 core CMP architecture and 7.4% and 5.5% for the 64 core CMP architecture, respectively. This is summarized in Table 4.4.

Table 4.4: Average improvement in terms of EDP values.

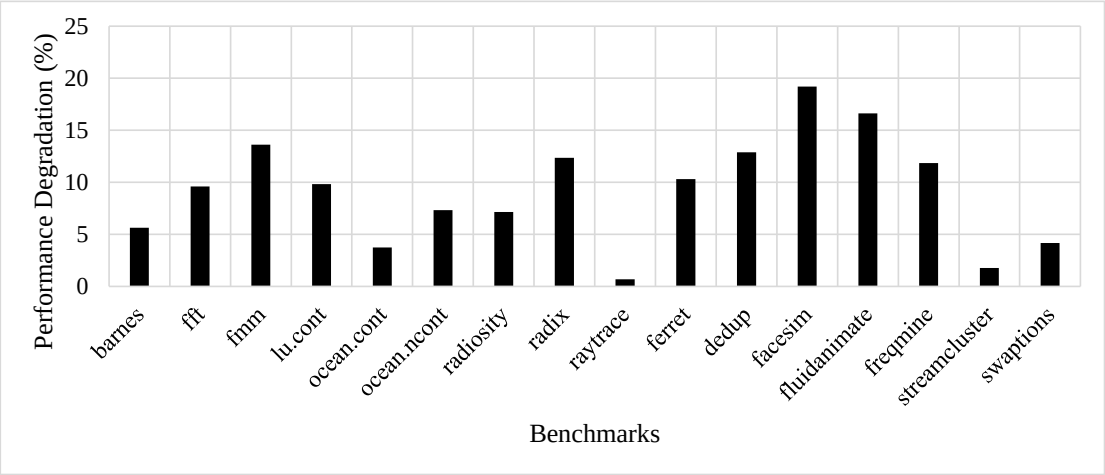
Comparison	**16 core CMP architecture**	**64 core CMP architecture**
DNN vs. RL	6.3%	7.4%
DNN vs. Kalman	6%	5.5%

While the improvement is within the range of 5.5%-7.4% on average, we consider that this is valuable. Aside from the fact that this study does improve

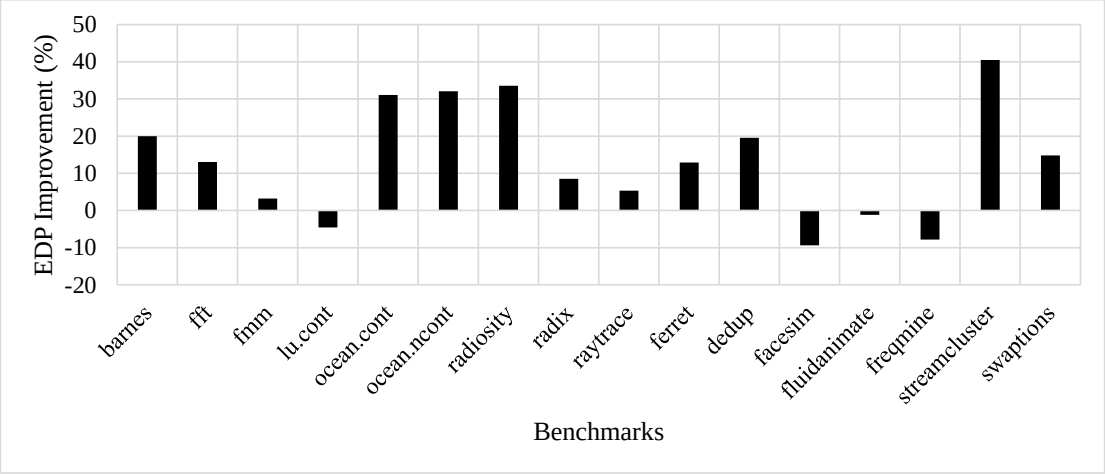
results over the existing approaches, and despite that DNN models require training data collection and training, this work sheds light on what a relatively straightforward DNN based approach for energy optimization would be able to achieve. This can be useful information for other researchers who may be interested in employing DNN models at the processor level - our results would provide an informed starting or reference point. We consider our work as a step towards what other researchers see as a necessity to address the complexity in designing CMP systems and machine learning techniques [112].



(a)

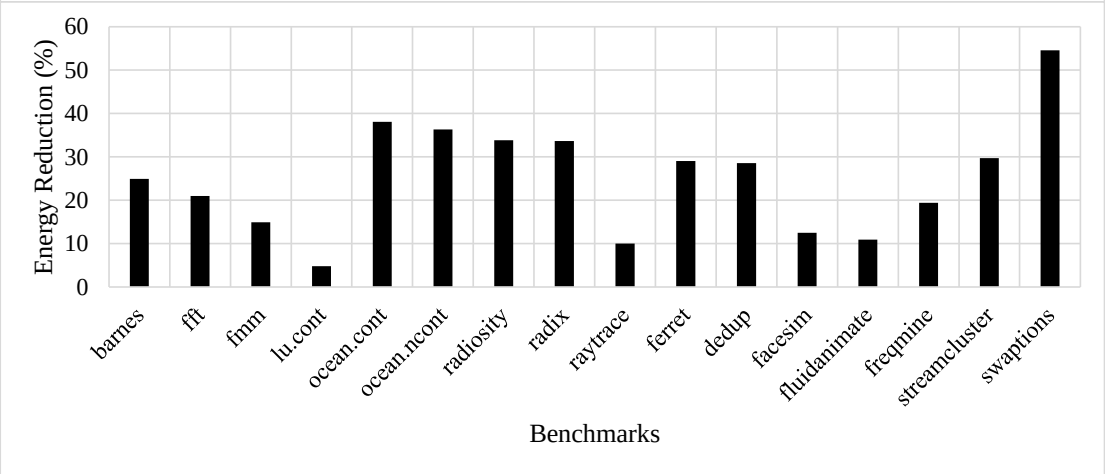


(b)

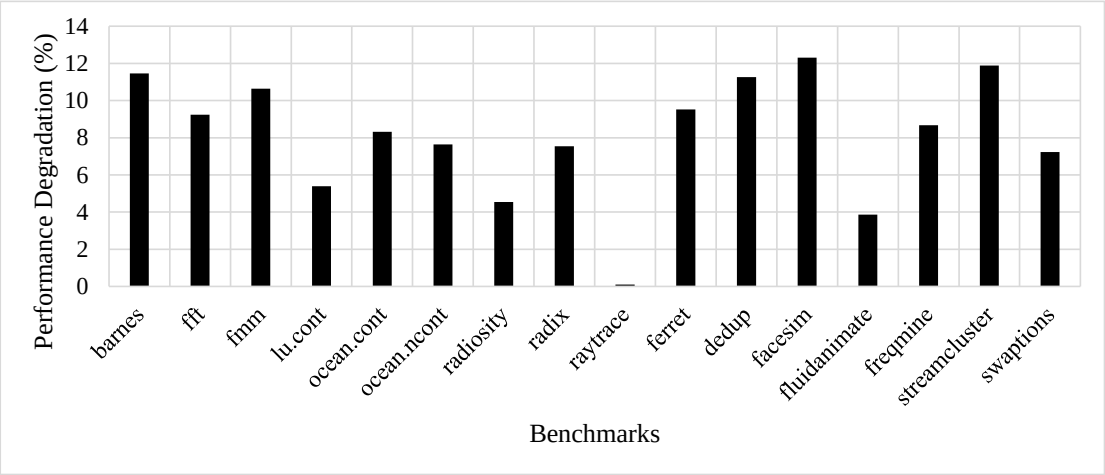


(c)

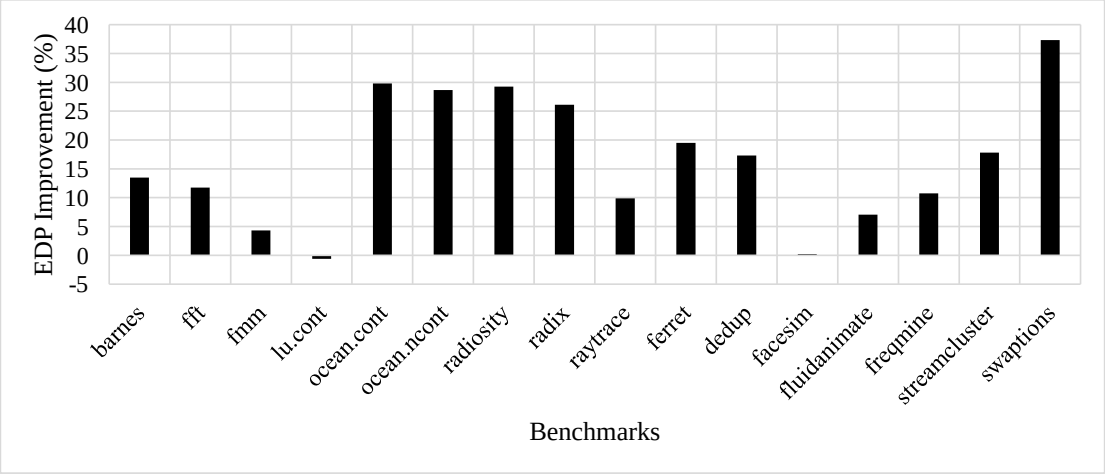
Figure 4.23: Comparison of the proposed DNN model based energy optimization algorithm vs. no optimization at all for 16 core CMP. (a) percentage of energy reduction, (b) percentage of performance degradation, and (c) percentage of EDP improvement.



(a)

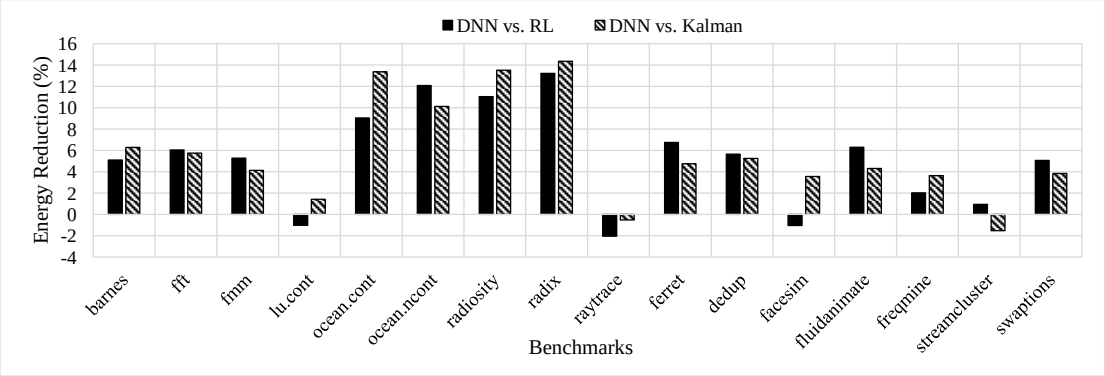


(b)

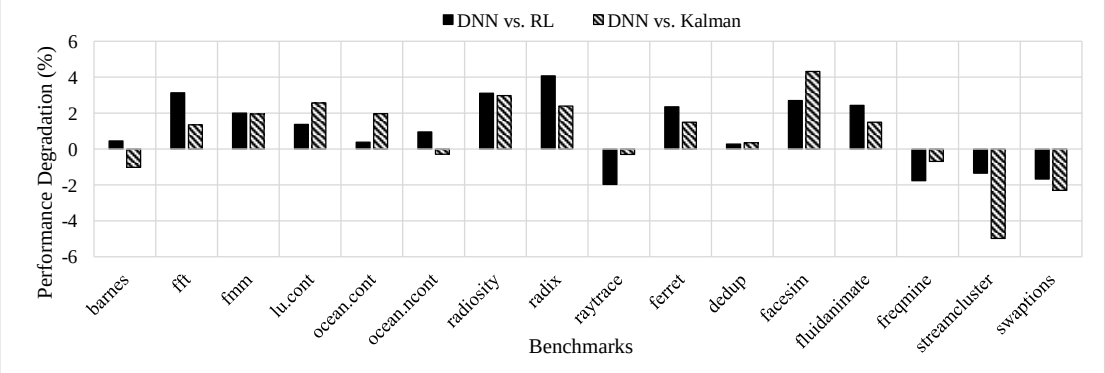


(c)

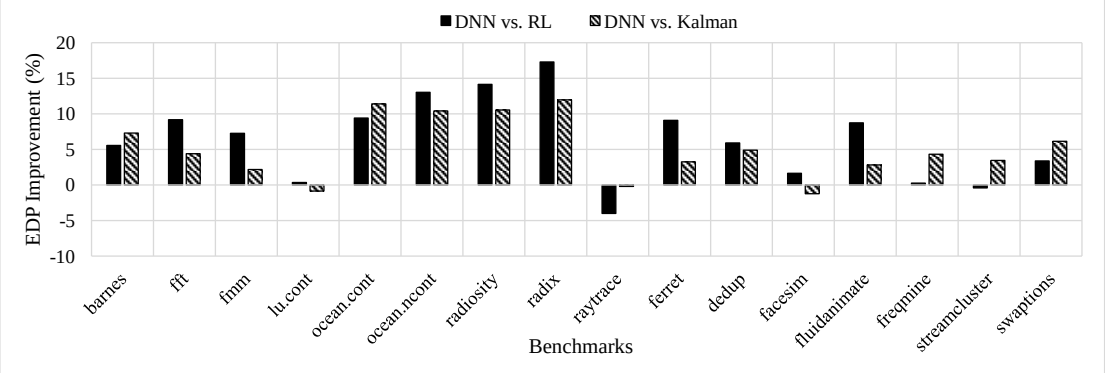
Figure 4.24: Comparison of the proposed DNN model based energy optimization algorithm vs. no optimization at all for 64 core CMP. (a) percentage of energy reduction, (b) percentage of performance degradation, and (c) percentage of EDP improvement.



(a)

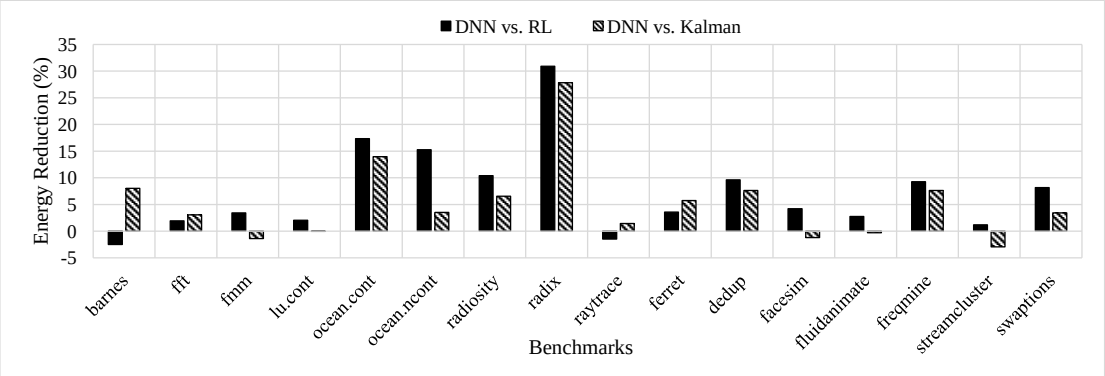


(b)

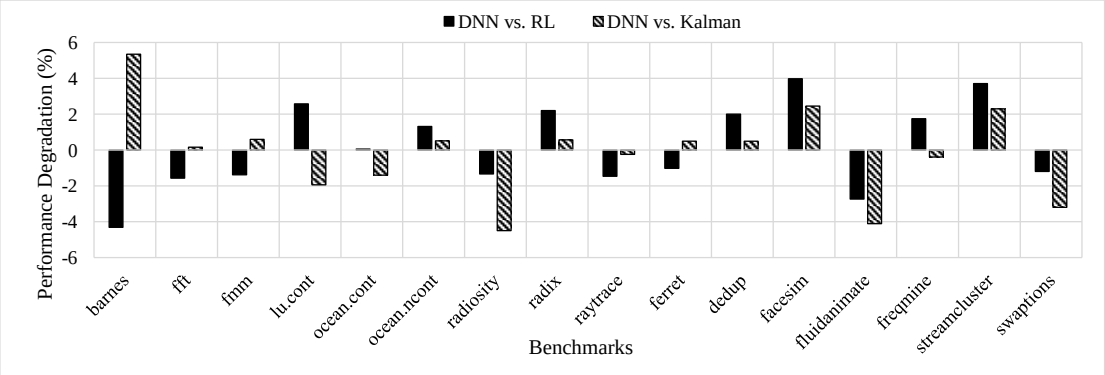


(c)

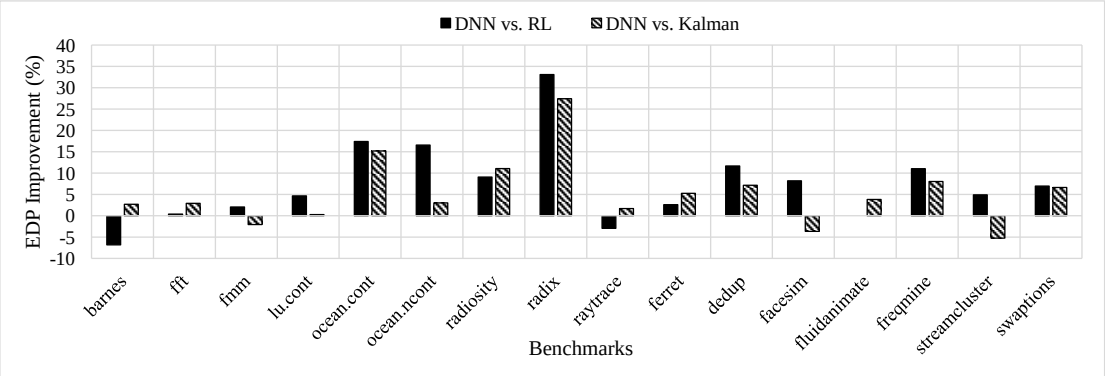
Figure 4.25: Comparison of the proposed DNN model based energy optimization algorithm against the RL and the Kalman filtering based approaches for 16 core CMP. (a) percentage of energy reduction, (b) percentage of performance degradation, and (c) percentage of EDP improvement.



(a)



(b)



(c)

Figure 4.26: Comparison of the proposed DNN model based energy optimization algorithm against the RL and the Kalman filtering based approaches for 64 core CMP. (a) percentage of energy reduction, (b) percentage of performance degradation, and (c) percentage of EDP improvement.

CHAPTER 5

Conclusion and Future Work

In this dissertation, we investigated techniques for dynamic reliability management and dynamic energy management in future network-on-chip based chip multiprocessors to address emerging design challenges related to reliability and energy consumption. Our main contributions can be summarized as follows:

- We proposed a dynamic reliability management approach for NoC based chip multiprocessors that considers both computation and communication components in a unified system. The proposed DRM is a hybrid approach that takes benefit of both DVFS and thread migration techniques in order to increase the lifetime reliability of the overall system to the desired target with minimal performance degradation.

- We proposed novel algorithms for dynamic energy management under performance constraints. We proposed a very effective heuristic that also uses the DVFS technique and a very efficient workload prediction technique based on Kalman filtering and LSTM. Either of the two prediction methods is employed to estimate the workload in the next control period for each of the processor cores. These estimates are then used to select V/F pairs for each core of the CMP during the next control period as part of a DVFS technique. The objective of the DVFS technique is to reduce energy consumption under performance constraints that are set by the user.

- For the first time, we investigated the use of deep neural network models for energy optimization under performance constraints in chip multiprocessor systems. We introduced a dynamic energy management algorithm implemented in three phases. In the first phase, training data is collected by running several selected instrumented benchmarks. A training data point represents a pair of values of cores' workload characteristics and of optimal V/F pairs. This phase employs Kalman filtering for workload prediction and an efficient heuristic algorithm based on dynamic voltage and frequency scaling. The second phase represents the training process of the DNN model. In the last phase, the DNN model is used to directly identify V/F pairs that can achieve lower energy consumption without performance degradation beyond the acceptable threshold set by the user

We tested the proposed algorithms with a variety of benchmarks on 16 and 64 core NoC based CMP architectures. For the proposed hybrid DRM approach, full-system based simulations using a customized GEM5 simulator demonstrated that lifetime reliability can be improved by 100% for an average performance penalty of 7.7% and 8.7% for the two CMP architectures.

Furthermore, the simulation results demonstrated that the proposed dynamic energy management approach can achieve up to 55% energy reduction for 10% performance degradation constraints. In addition, the proposed DNN approach was compared against existing approaches based on reinforcement learning and Kalman filtering/LSTM and was found that it provides average improvements in EDP of 6.3% and 6% for the 16 core architecture and of 7.4% and 5.5% for the

64 core architecture, respectively.

As potential future work, there are several interesting ideas that could be investigated as described next.

- Study the use of other reliability models such as electromigration (EM), thermal cycling (TC), and hot carrier injection (HCI) in order to capture the fact that lifetime should also be a function of the current state of degradation [113].

- Extend the DNN model to an approach where both DVFS and task mapping are used in a hybrid solution. Currently, it is unclear how the accuracy of the DNN model would be affected if it were used in a system that combines DVFS and task migration.

- Use snapshots of the whole system at a time, recorded as an image and investigate convolutional neural networks (CNN) to take into account possible correlations among different cores.

- Develop a multi-objective approach that takes into the consideration the reliability, energy and performance at the same time.

References

[1] J. Whitney and P. Delforge, "Data center efficiency assessment - scaling up energy efficiency across the data center industry: evaluating key drivers and barriers," *Natural Resources Defense Council (NRDC) Report*, 2014. [Online]. Available: https://www.nrdc.org/sites/default/files/data-center-efficiency-assessment-IP.pdf

[2] Annual Energy Outlook, U.S. Energy Information Administration (EIA), 2016. [Online]. Available: http://www.eia.gov/forecasts/aeo/data.cfm#enconsec

[3] United States Environmental Protection Agency, "Report to Congress on server and data center energy efficiency," *Report*, 2007. [Online]. Available: https://www.energystar.gov/ia/partners/prod_development/downloads/EPA_Datacenter_Report_Congress_Final1.pdf

[4] A. Dehon, H.M. Quinn, and N.P. Carter, "Vision for cross-layer optimization to address the dual challenges of energy and reliability," *ACM/IEEE Design Automation and Test in Europe Conf. (DATE)*, March 2010.

[5] F. Angiolini, D. Atienza, S. Murali, L. Benini, and G. De Micheli, "Reliability support for on-chip memories using Networks-on-Chip," *IEEE Int. Conf. on Computer Design (ICCD)*, Oct. 2007.

[6] R. Vadlamani, J. Zhao, W. Burleson, and R. Tessier, "Multicore soft error rate stabilization using adaptive dual modular redundancy," *ACM/IEEE Design Automation and Test in Europe Conf. (DATE)*, March 2010.

[7] S. Feng, S. Gupta, A. Ansari, and S. Mahlke, "Shoestring: probabilistic soft error reliability on the cheap," *Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, March 2010.

[8] J. Blome, S. Feng, S. Gupta, and S. Mahlke, "Self-calibrating online wearout detection," *Int. Symp. on Microarchitecture (MICRO)*, pp. 109-120, Dec. 2007.

[9] Y. Li, Y.M. Kim, E. Mintarno, D.S. Gardner, and S. Mitra, "Overcoming early-life failure and aging for robust systems," *IEEE Design & Test of Computers*, vol. 26, no. 6, pp. 28-39, 2009.

[10] B. Datta and W. Burleson, "Circuit-level NBTI macro-models for collaborative reliability monitoring," *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, May 2010.

[11] M. Choudhury, V. Chandra, K. Mohanram, and R. Aitken, "TIMBER: time borrowing and error relaying for online timing error resilience," *ACM/IEEE Design Automation and Test in Europe Conf. (DATE)*, March 2010.

[12] L. Huang and Q. Xu, "Energy-efficient task allocation and scheduling for multi-mode MPSoCs under lifetime reliability constraint," *ACM Int. Conference on Design Automation and Test in Europe (DATE)*, 2010.

[13] S. Wang and J.-J. Chen, "Thermal-aware lifetime reliability in multicore systems," *Int. Symp. on Quality Electronic Design (ISQED)*, 2010.

[14] A. Masrur et al., "Schedulability analysis for processors with aging-aware autonomic frequency scaling," *IEEE Int. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2012.

[15] A. Das, A. Kumar, and B. Veeravalli, "Reliability-driven task mapping for lifetime extension of networks-on-chip based multiprocessor systems," *ACM Int. Conference on Design Automation and Test in Europe (DATE)*, 2013.

[16] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele, "Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs," *ACM Int. Conference on Design Automation and Test in Europe (DATE)*, 2014.

[17] J. Srinivasan, Lifetime reliability aware microprocessors, Ph.D. Thesis, University of Illinois at Urbana-Champaign, 2006.

[18] Z. Lu, J. Lach, M.R. Stan, and K. Skadron, "Improved thermal management with reliability banking," *IEEE Micro*, vol. 25, no. 6, pp. 40-49, 2005.

[19] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge, "Multi-mechanism reliability modeling and management in dynamic systems," *IEEE Trans. on Very Large Scale Integration Systems (TVLSI)*, vol. 16, no. 4, 2008.

[20] S. Feng, S. Gupta, A. Ansari, and S. Mahlke, "Maestro: orchestrating lifetime reliability in chip multiprocessors," *Int. Conf. on High-Performance Embedded Architectures and Compilers (HiPEAC)*, 2010.

[21] A. Tiwari and J. Torrellas, "Facelift: hiding and slowing down aging in multicores," *ACM/IEEE Int. Symp. on Microarchitecture (MICRO)*, 2008.

[22] C. Zhuo, D. Sylvester, and D. Blaauw, "Process variation and temperature-aware reliability management," *ACM/IEEE Design Automation and Test in Europe Conf. (DATE)*, 2010.

[23] A.K. Coskun, R.D. Strong, D.M. Tullsen, and T.S. Rosing, "Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors," *SIGMETRICS/Performance*, 2009.

[24] J. Sun, A.K. Kodi, A. Louri, and J.M. Wang, "NBTI aware workload balancing in multi-core systems," *IEEE Int. Symp. on Quality Electronic Design (ISQED)*, 2009.

[25] O. Khan and S. Kundu ,"A self-adaptive system architecture to address transistor aging," *ACM/IEEE Design Automation and Test in Europe Conf. (DATE)*, 2009.

[26] P. Mercati, A. Bartolini, F. Paterna, T.S. Rosing, and L. Benini, "Workload and user experience-aware dynamic reliability management in multicore processors," *ACM Int. Design Automation Conference (DAC)*, 2013.

[27] P. Mercati et al., "A Linux-governor based dynamic reliability manager for android mobile devices," *ACM/IEEE Design Automation and Test in Europe Conf. (DATE)*, 2013.

[28] A. Das, R. A. Shafik, G. V. Merrett, B. M. Al-Hashimi, A. Kumar, and B. Veeravalli, "Reinforcement learning-based inter- and intra-application thermal optimization for lifetime improvement of multicore systems," *ACM Int. Design Automation Conference (DAC)*, 2014.

[29] A.Y. Yamamoto and C. Ababei, "Unified reliability estimation and management of NoC based chip multiprocessors," *Microprocessors and Microsystems*, vol. 38, no. 1, pp. 53-63, Feb. 2014.

[30] H. Kim, S.B.K. Boga, A. Vitkovskiy, S. Hadjitheophanous, P.V. Gratz, V. Soteriou, and M.K. Michael, "Use it or lose it: proactive, deterministic longevity in future chip multiprocessors," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 20, no. 4, Sep. 2015.

[31] Z. Ghaderi, A. Alqahtani, N. Bagherzadeh, "Online monitoring and adaptive routing for aging mitigation in NoCs," *ACM/IEEE Design Automation and Test in Europe Conf. (DATE)*, 2017.

[32] M. Salehi et al., "dsReliM: Power-constrained reliability management in dark-silicon many-core chips under process variations," *CODES+ISSS*, 2015.

[33] M. Salehi et al., "Run-time adaptive power-aware reliability management for many-cores," *IEEE Design & Test*, 2017.

[34] D. Gnad et al., "Hayat: harnessing dark silicon and variability for aging deceleration and balancing," *DAC*, 2015.

[35] V. Rathore et al., "HiMap: A hierarchical mapping approach for enhancing lifetime reliability of dark silicon manycore systems," *DATE*, 2018.

[36] H. Hong, J. Lim, H. Lim, and S. Kang, "Lifetime reliability enhancement of microprocessors: mitigating the impact of negative bias temperature instability," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, pp. 1-25, Sep. 2

[37] W. Kim, M.S. Gupta, G.-Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," *IEEE Int. Symposium on High Performance Computer Architecture (HPCA)*, 2008.

[38] T. Kolpe, A. Zhai, and S.S. Sapatnekar, "Enabling improved power management in multicore processors through clustered DVFS," *ACM/IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2011.

[39] K. Chakraborty and S. Roy, "Architecturally homogeneous power-performance heterogeneous multicore systems," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 4, pp. 670-679, Apr. 2013.

[40] A.A. Sinkar, H.R. Ghasemi, M.J. Schulte, U.R. Karpuzcu, and N.S. Kim, "Low-cost per-core voltage domain support for power-constrained high-performance processors," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 4, pp. 747-758, Apr. 2014.

[41] R. Jevtic, H.-P. Le, M. Blagojevic, S. Bailey, K. Asanovic, E. Alon, and B. Nikolic, "Per-core DVFS with switched-capacitor converters for energy efficiency in manycore processors," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 4, pp. 723-730, May 2015.

[42] R. Schone, T. Ilsche, M. Bielert, D. Molka, and D. Hackenberg, "Software controlled clock modulation for energy efficiency optimization on Intel processors," *IEEE Int. Workshop on Energy Efficient Supercomputing (E2SC)*, 2016.

[43] G. Dhiman and T.S. Rosing, "Dynamic voltage frequency scaling for multitasking systems using online learning," *ACM/IEEE Int. Symposium on Low Power Electronics and Design (ISLPED)*, 2007.

[44] M. Moeng and R. Melhem, "Applying statistical machine learning to multicore voltage and frequency scaling," *ACM Int. Conference on Computing Frontiers*, 2010.

[45] H. Jung and M. Pedram, "Improving the efficiency of power management techniques by using bayesian classification," *Int. Symposium on Quality Electronic Design (ISQED)*, 2008.

[46] D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, and L. Torres, "Dynamic and distributed frequency assignment for energy and latency constrained MP-SoC," *ACM/IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2009.

[47] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, L. Benini, and G.D. Micheli, "Temperature control of high-performance multi-core platforms using convex optimization," *ACM/IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2008.

[48] H. Sayadi, D. Pathak, I. Savidis and H. Homayoun, "Machine learning-based approaches for energy-efficiency prediction and scheduling in composite cores architectures," *IEEE Int. Conference on Computer Design (ICCD)*, 2017.

[49] S. Sharifi, A.K. Coskun, and T.S. Rosing, "Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor SoCs," *ACM/IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2010.

[50] H. Sayadi and H. Homayoun, "Scheduling Multithreaded Applications onto Heterogeneous Composite Cores Architecture," *IEEE Int. Green and Sustainable Computing Conference (IGSC)*, 2017.

[51] C.H. Hsu and W.-C. Feng, "A power-aware run-time system for high-performance computing," *ACM/IEEE Conference on Supercomputing*, 2005.

[52] R. Ge, X. Feng, W. Feng, and K.W. Cameron, "CPU MISER: a performance-directed, run-time system for power-aware clusters," *IEEE Int. Conf. on Parallel Processing (ICPP)*, 2007.

[53] S. Huang and W. Feng, "Energy-efficient cluster computing via accurate workload characterization," *IEEE/ACM Int. Symposium on Cluster Computing and the Grid (CCGRID)*, 2009.

[54] B. Rountree, D.K. Lownenthal, B.R. de Supinski, M. Schulz, V.W. Freeh, and T. Bletsch, "Adagio: making DVS practical for complex HPC applications," *ACM/IEEE Conference on Supercomputing*, 2009.

[55] V. Sundriyal, M. Sosonkina, "Joint frequency scaling of processor and DRAM,"

The Journal of Supercomputing, vol. 72, no. 4, pp. 1549-1569, Apr. 2016.

[56] V. Sundriyal, M. Sosonkina, F. Liu, M.W. Schmidt, "Dynamic frequency scaling and energy saving in quantum chemistry applications," *IEEE Int. Parallel and Distributed Processing Symposium (IPDPS)*, 2011.

[57] G. Keramidas, V. Spiliopoulos, and S. Kaxiras, "Interval-based models for run-time DVFS orchestration in superscalar processors," *ACM Int. Conference on Computing Frontiers*, 2010.

[58] B. Rountree, D.K. Lowenthal, M. Schulz, and B.R. de Supinski, "Practical performance prediction under dynamic voltage frequency scaling," *Int. Green Computing Conference and Workshops*, 2011.

[59] R. Miftakhutdinov, E. Ebrahimi, and Y.N. Patt, "Predicting performance impact of DVFS for realistic memory systems," *Int. Symposium on Microarchitecture (MICRO)*, 2012.

[60] S. Eyerman and L. Eeckhout, "A counter architecture for online DVFS profitability estimation," *IEEE Trans. on Computers*, vol. 59, no. 11, pp. 1576-1583, March 2010.

[61] J.Y. Won, P. V. Gratz, S. Shakkottai, and J. Hu, "Resource sharing centric dynamic voltage and frequency scaling for CMP cores, uncore, and memory," *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, vol. 21, no. 4, p. 69, Sep. 2016.

[62] C. Ababei and N. Mastronarde, "Benefits and costs of prediction based DVFS for NoCs at router level," *IEEE Int. SoC Conference (SOCC)*, 2014.

[63] L. Shang, L.-S. Peh, and N. K. Jha, "Dynamic voltage scaling with links for power optimization of interconnection networks," *Int. Symposium on High-Performance Computer Architecture (HPCA)*, 2003.

[64] A. Das, A. Kumar, B. Veeravalli, R.A. Shafik, G.V. Merrett, and B.M. Al-Hashimi, "Workload uncertainty characterization and adaptive frequency scaling for energy minimization of embedded systems," *ACM/IEEE Design, Automation & Test in Europe Conference (DATE)*, 2015.

[65] R. Cochran, C. Hankendi, A.K. Coskun, and S. Reda, "Pack & cap: adaptive DVFS and thread packing under power caps," *ACM/IEEE Int. Symposium on Microarchitecture (MICRO)*, 2011.

[66] H. Shen, J. Lu, and Q. Qiu, "Learning based DVFS for simultaneous temperature, performance and energy management," *ACM/IEEE Int. Symposium on Quality Electronic Design (ISQED)*, 2012.

[67] R. Ye and Q. Xu, "Learning-based power management for multicore processors via idle period manipulation," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 33, no. 7, pp. 1043-1056, July 2014.

[68] B.K. Donohoo, C. Ohlsen, S. Pasricha, Y. Xiang, and C.W. Anderson, "Context-aware energy enhancements for smart mobile devices," *IEEE Trans. on Mobile Computing*, vol. 13, no. 8, pp. 1720-1732, July 2014.

[69] H. Jung and M. Pedram, "Supervised learning based power management for multicore processors," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 29, no. 9, pp. 1395-1408, Sep. 2010.

[70] Z. Chen and D. Marculescu, "Distributed reinforcement learning for power limited many-core system performance optimization," *ACM/IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2015.

[71] H. Shen, Y. Tan, J. Lu, Q. Wu, and Q. Qiu, "Achieving autonomous power management using reinforcement learning," *ACM Trans. on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 2, article 24, March 2013.

[72] Z. Wang, Z. Tian, J. Xu, R. Maeda and H. Li, "Modular reinforcement learning for self-adaptive energy efficiency optimization in multicore system," *ACM/IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2017.

[73] D. Biswas, V. Balagopal, R. Shafik, B. Al-Hashimi and G. Merrett, "Machine learning for run-time energy optimisation in many-core systems," *ACM/IEEE Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2017.

[74] R.G. Kim, W. Choi, Z. Chen, J.R. Doppa, P.P. Pande, D. Marculescu and R. Marculescu. "Imitation learning for dynamic VFI control in large-scale manycore systems," *IEEE Trans. on VLSI Systems*, vol. 24, no. 9, pp. 2488-2501, Sep. 2017.

[75] 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," *HPCA*, 2014.

[76] J. Gao, "Machine learning applications for data center optimization," *Google White Paper*, 2014. [Online]. Available: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42542.pdf.

[77] M.G. Moghaddam and C. Ababei, "Dynamic lifetime reliability management for chip multiprocessors," *IEEE Trans. on Multiscale Computing sustems*, 2018.

[78] M.G. Moghaddam, W. Guan and C. Ababei, "Dynamic energy minimization in chip multiprocessors using deep neural networks," *IEEE Trans. on Multiscale Computing sustems*, 2018.

[79] C. Ababei and M.G. Moghaddam, "A Survey of Prediction and Classification Techniques in Multicore Processor Systems," *IEEE Trans. on Parallel and Distributed Systems*, 2018.

[80] M.G. Moghaddam and C. Ababei, "Dynamic energy management for chip multiprocessors under performance constraints," *Microprocessors and Microsystems*, vol. 54, pp. 1-13, Oct. 2017.

[81] M.G. Moghaddam, W. Guan and C. Ababei, "Investigation of LSTM based prediction for dynamic energy management in chip multiprocessors," *IEEE Int. Green and Sustainable Computing Conference*, 2017.

[82] M.G. Moghaddami, “Dynamic energy and reliability management in network-on-chip based chip multiprocessors,” *IEEE Int. Green and Sustainable Computing Conference*, 2017.

[83] M.G. Moghaddam, A. Yamamoto, and C. Ababei, “Investigation of DVFS based dynamic reliability management for chip multiprocessors," *IEEE Int. Conference on High Performance Computing &Simulation (HPCS)*, 2015.

[84] J.H. Stathis, “Reliability limits for the gate insulator in CMOS technology," vol. 46, no. 2, pp. 265-286, Mar. 2002.

[85] E. Wu, J. Sune, W. Lai, E. Nowak, J. McKenna, A. Vayshenker, D. Harmon,“Interplay of voltage and temperature acceleration of oxide breakdown for ultra-thin gate oxides, Solid-State Electron," vol. 46, no. 11, pp. 1787-1798, Nov. 2002.

[86] M.A. Alam, H.Kufluoglu, D. Varghese, S. Mahapatra, “A comprehensive model for PMOS NBTI degradation: Recent progres,"*Microelectronics Reliability*, vol. 47, no. 6, pp. 853-862, Jun. 2007.

[87] A.T. Krishnan, V. Reddy, S. Chakravarthi, J. Rodriguez, S. John, S. Krishnan , “NBTI impact on transistor and circuit: models, mechanisms and scaling effects [MOSFETs]," *IEEE Int. Electronics Device Meeting (IEDM)*, 2003.

[88] W. Abadeer, W. Ellis, “Behaviour of NBTI Under AC Dynamic Circuit Conditions," *Int. Physics Reliability Symposium (IPRS)*, 2003.

[89] A.T. Krishnan, V. Reddy, S. Chakravarthi, J. Rodriguez, S. John, S. Krishnan , “Effects of Measurement Temperature on NBTI," *IEEE Electron Device Letters*, vol. 28, no. 4, pp. 298-300, Apr. 2007.

[90] J. Keane, X. Wang, D. Persaud and C. Kim, “An all-in-one silicon odometer for separately monitoring HCI, BTI, and TDDB," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 4, pp. 817-829, Apr. 2007.

[91] X. Li, J. Qin and J. Bemstein ,“Compact Modeling of MOSFET Wearout Mechanisms for Circuit-Reliability Simulation," *IEEE Trans. on Device and Material Reliability*, vol. 8, no. 1, pp. 98-121, Mar. 2008.

[92] G. Welch and G. Bishop, An Introduction to the Kalman Filter. Chapel Hill, NC: Univ. North Carolina, Chapel Hill, 1995.

[93] S. Bang, K. Bang, S. Yoon, and E. Chung, “Run-time adaptive workload estimation for dynamic voltage scaling,” *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 28, no. 9, pp. 1334-1347, Aug. 2009.

[94] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, Sep. 2016.

[95] The vanishing gradient problem, 2017. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap5.html

[96] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, Nov. 1997.

[97] A beginner's guide to recurrent networks and LSTMs, 2017. [Online]. Available: https://deeplearning4j.org/lstm.html

[98] Kevin P. Murphy, Machine Learning: A Probabilistic Perspective, MIT Press, 2012.

[99] Y. LeCun, "Learning invariant feature hierarchies," *ECCV*, 2012.

[100] G. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, pp. 1527-1554, July 2006.

[101] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313. no. 5786, pp. 504-507, July 2006.

[102] L. Deng and D. Yu, "Deep learning: methods and applications," *NOW Foundations and Trends in Signal Processing*, vol. 7, no. 3-4, June 2014.

[103] 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. Sewall, M. Shoaib, N. Vaish, M. D. Hill, D.A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News Archive*, 2011.

[104] S. Li, J.H. Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, N.P. Jouppi, "McPAT: an integrated power, area, timing modeling framework for multicore and manycore architectures," *IEEE/ACM Int. Symposium on Microarchitecture (MICRO)*, 2009.

[105] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron and M.R. Stan, "HotSpot: a compact thermal modeling method for CMOS VLSI systems," *IEEE Trans. on Very Large Scale Integration Systems (TVLSI)*, vol. 14, no. 5, 2006.

[106] REST: Reliability ESTimation for chip multiprocessors (CMPs), 2014. [Online]. Available: https://code.google.com/p/reliability-estimator

[107] M. Gebhart, J. Hestness, E. Fatehi, P. Gratz, and S.W. Keckler, "Running PARSEC 2.1 on M5," The University of Texas at Austin, *Technical Report TR-09-32*, Oct. 2009.

[108] T.E. Carlson, W.Heirman, and L. Eeckhout, "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation," *Int. Conf. for High Performance Computing, Networking, Storage and Analysis*, 2011.

[109] PARSEC and Splash2 benchmarks, 2017. [Online]. Available: http://parsec.cs.princeton.edu

[110] Keras: the Python deep learning library, 2017. [Online]. Available: https://keras.io/

[111] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S., Corrado, A. Davis, J. Dean, M. Devin and S. Ghemawat, "Tensorflow: large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, March 2016.

[112] R.G. Kim, J.R. Doppa, P.P. Pande, D. Marculescu and R. Marculescu. "Machine learning and manycore systems design: a serendipitous symbiosis," *Submitted to Learning*, Dec. 2017. [Online]. Available: https://scirate.com/arxiv/1712.00076

[113] L. Huang and Q. Xu, "AgeSim: a simulation framework for evaluating the lifetime reliability of processor-based SoCs," *ACM Int. Conference on Design Automation and Test in Europe (DATE)*, 2010.